AMEISE
An Interactive Environment to Acquire Project-Management Experience

Roland Mittermeir[1)], **Andreas Bollin**[1)], **Elke Hochmüller**[2)], **Susanne Jäger**[1)], **Daniel Wakounig**[1)]

[1)] **Institut für Informatik-Systeme**
Alpen-Adria Universität Klagenfurt
Klagenfurt, Österreich
{roland,andi, susi, daniel}
@isys.uni-klu.ac.at

[2)] **Studiengang**
Telematik/Netzwerktechnik
Fachhochschule Technikum Kärnten
Klagenfurt, Österreich
E.Hochmueller@cti.ac.at

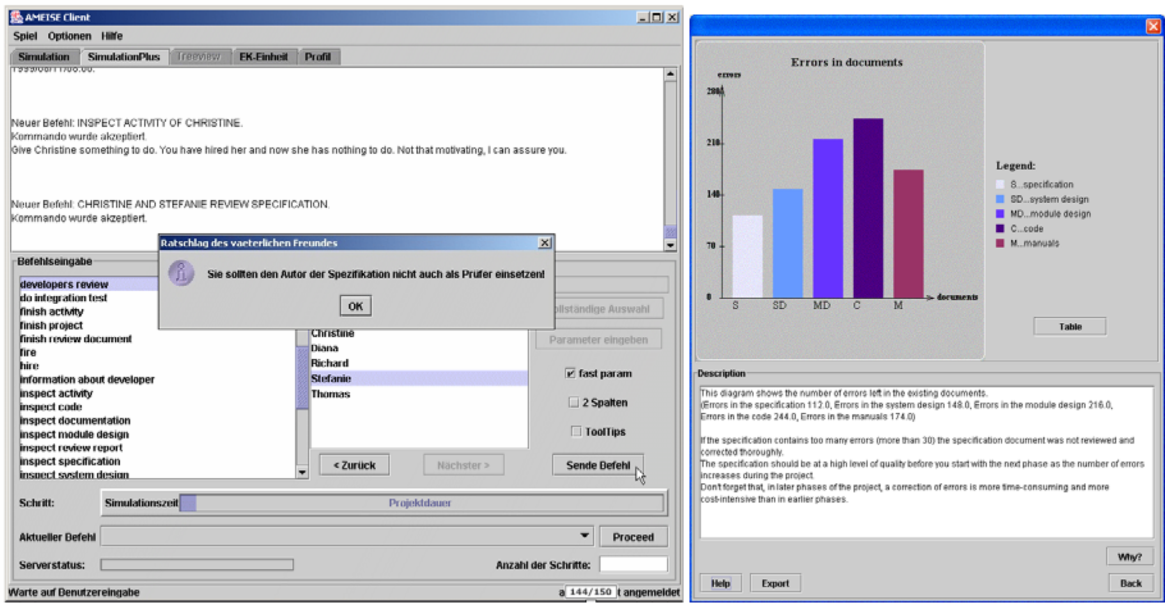
Abstract

This paper reports on experience with AMEISE, a simulation environment for the management of software engineering projects. AMEISE allows trainees to act as project managers in a virtual development situation.
The system keeps track of every decision and of all relevant simulation states. This allows for automatic as well as discursive assessment of the students performance as project managers.
The paper presents the motivation and didactical considerations of developing and using such a system as well as the experiences gained by using it in various courses during the last years.

1. Motivation

Learning how to manage a Software Engineering Project cannot be achieved by only reading books or listening to lectures. It requires a combination of theoretical knowledge and practical skills. While the former can be acquired in practical lectures, practical skills are not to be imparted that easily. Notably, students have not the chance to revoke decisions that are considered problematic at a later time of their project and failures usually involve high social cost. Though there are well known best-practices (e.g. [Hump 98]), content presented in lectures quite often suffers from credibility problems, even when enriched with empirical data. This deficiency has many roots, but mainly it is due to the fact, that such lectures are based on data from other people's projects – projects that are considered to be different anyway.

Gaining practical experience is, for sure, inevitable. But experience gained in lab situations is usually characterized by a lot of guidance. To be realistic, students should be entrusted far more responsibility. Consequently, they should be allowed to make a complete mess of the project, to try to save it from decline, and to learn from past mistakes. In general it means to allow for a trial and error approach, which might lead to very time-consuming courses [Hoch 02]; and, who dares to conduct real-world projects with students and to let them fail? One way-out is simulation.



Neuer Befehl: INSPECT ACTIVITY OF CHRISTINE.
Kommando wurde akzeptiert.
Give Christine something to do. You have hired her and now she has nothing to do. Not that motivating, I can assure you.

Neuer Befehl: CHRISTINE AND STEFANIE REVIEW SPECIFICATION.
Kommando wurde akzeptiert.

Ratschlag des vaeterlichen Freundes

Sie sollten den Autor der Spezifikation nicht auch als Prüfer einsetzen!

OK

This diagram shows the number of errors left in the existing documents.
(Errors in the specification 112.0, Errors in the system design 148.0, Errors in the module design 216.0, Errors in the code 244.0, Errors in the manuals 174.0)

If the specification contains too many errors (more than 30) the specification document was not reviewed and corrected thoroughly.
The specification should be at a high level of quality before you start with the next phase as the number of errors increases during the project.
Don't forget that, in later phases of the project, a correction of errors is more time-consuming and more cost-intensive than in earlier phases.

Fig. 1: The AMEISE System contains several support and assessment components.

In this paper we present AMEISE, a simulation environment that allows students to gain first experience in managing a software engineering project (with the focus on software quality). AMEISE, *A Media Education Initiative for Software Engineering*[1], has been developed by a consortium of three Austrian institutions of higher education: the Alps-Adriatic University Klagenfurt, the Carinthia Tech Institute, and Linz University[2]. AMEISE uses a simulation engine called SESAM, *Software Engineering Simulated by Animated Models*[3], developed under the direction of Jochen Ludewig at the University of Stuttgart [DL 00]. The AMEISE team adopted the educational model proposed by the team at Stuttgart and also used the SESAM system as initial prototype that was evolved in a series of iterations to the currently available AMEISE-system [MHBJN 03, Mitt 06] and it is still subject to further evolution.

The paper is structured as follows: Section 2 discusses the educational model behind AMEISE and the related extensions made to the SESAM system. Section 3 addresses the specific challenges related to an interface providing student-managers with comprehensive information about the state of their project. Section 4 presents the experience gained during AMEISE courses – be it in classroom situations or in blended-learning settings with simulation runs totally unattended by tutors. The paper closes with an outlook, planned further improvements of the system, and upcoming tests of the didactical concepts.

2. AMEISE and Educational Challenges

The approach AMEISE follows is distinct from the mainstream of eLearning approaches with pre-canned (multimodal and multi-medial) content in so far, as the systems knowledge base is just a complex set of rules that are not directly accessible to students. The content to

---

[1]) AMEISE web-site: http://ameise.uni-klu.ac.at
[2]) Development of AMEISE has been supported by the Austrian Ministry of Science and Education, bm:bwk, in their eLearning support initiative „New Media in Tertiary Education", NML. Web-site: www.nml.at.
[3]) SESAM web-site: http://www.iste.uni-stuttgart.de/se/research/sesam/

learn from is “produced” interactively by each student’s assignments of work to various members of her or his development team. Thus, the “content” that serves as basis for learning is created on the fly by the student’s activities acting as technical manager. The challenge for the system (and its developers) is to readily analyze this content and present it in the most suitable form to both, the student who produced it as well as the tutor, who should give further in-depth interpretations and hints.

While the system’s interface grew from an initial pseudo-natural language textual interface to something closer to the state of the art (see Fig. 1, left side), we would not claim it to be multi-medial in a technical sense. It is rather multimodal, as the blended learning approach foresees that the results of the simulation run is discussed in group as well as in private with the students of a given course. With the feedback thus obtained (see Fig. 1, right side) students are ready to return to the system and strive for managing a second, hopefully improved, software engineering project.

2.1. Educational Aim and Description of the Task to Solve

The basic educational model is founded on experimental learning. This necessarily implies also to some extent the characteristics of learning from one's own failures. But to make sure that this is not de-motivating, our approach as well as the approach proposed by the developers of the SESAM-system, which has a parental role to AMEISE, recommend that students perform at least two simulation runs. In both, they should strive to obtain their best possible performance. However, usually, in the first one, a number of issues occur that give rise for discussion and improvement. In the second run, students can benefit from the experience gained. Thus, in general, they achieve more of the criteria marking success of the project or at least improve on the respective success dimension.

Concerning success dimensions, one should mention that steering a software development project to success implies to reach several, to some extent contradictory success criteria under a set of defined constraints. To mention just some constraints, projects use employees having different qualifications in different technical (and social) aspects. These (potential) employees have different salary expectations. Their qualifications result in different speed and, being (simulated) humans, to different error rates. Another important constraint results from the technical nature of software projects. Irrespective of the approach followed, there is a logical order in which some activities are to be performed. However, there is also a lot of freedom of scheduling or even skipping certain activities. Hence, student-managers have to decide, not only whom to hire (and release, if no longer needed) but also when to ask a person to perform which task. At this point, it should be mentioned that the virtual employees are keen in doing exactly what they are asked to do (unless already otherwise busy), but they are in no way motivated to overrule their managers orders if those don’t make sense. This might result in their attempting an infeasible task and to give up after a day of unsuccessful attempts brief termination message. Before receiving a new order, they will lean back and goof.

Such situations, obviously, bear negatively on the student-managers performance, since projects have obviously budgetary constraints and an agreed upon fixed delivery date. At this date, the project (both software and documentation) should be handed over to the customer at a specified minimum level of quality (degree of completeness and acceptable error rate). Thus, the key criteria of success is whether the project was d*elivered on time, within budget*, how *complete* were the delivered *software* and how complete was the delivered *documentation* as well as how many *errors* were found in the delivered *software* and in the delivered *documentation* respectively.

To arrive at these results, the simulation model rests on an empirically rich set of rules that are invoked depending on the commands the student-manager issues on a particular day of work. The model we are using focuses besides on general project management issues particularly on quality aspects. Students should learn among other things that quality cannot be "tested into a product". Quality has to be taken care of right from the beginning of the project and they have to experience that such dull things as technical reviews – hardly ever practiced in educational settings – are valuable in real and sizable projects.

Students also have to experience the merits of planning. A person basically works only on a single task per time instance. Split attention (if at all possible) causes less productivity. Teamwork is necessary to meet time- and quality-constraints. But teamwork causes communication and communication causes overhead. Thus, one should not be too liberal in striving towards a million-monkey approach. Scheduling and planning is also important when assigning certain tasks to be performed on a semi-finished piece of work to team members. A colleague of mine will be more productive in identifying faults, I have committed. But I should be more productive in making the necessary corrections. A third person might even introduce more new faults by correcting those previously identified.

Considering these and similar constraints, one can see that even with a relatively simple project (200 adjusted function points of effort to be expended over a development period of nine months) causes those, who do not adequately plan, update their plan when they cannot follow it, and keeping track of the assignments they made throughout the development period will towards the end, when money and time become scarce, run quite into trouble. Thus, enough material is produced that can be discussed in the feedback period.

2.2. Educational Setting

We used the AMEISE simulation system mostly in the context of general software engineering courses. Hence, supplementary development goals of the AMEISE extensions to the original SESAM system were to allow its use in larger classes. Thus an evaluation support for the tutor became necessary and mechanisms for self-evaluation by students were also deemed desirable. Only in exceptional cases, a special course was defined where we used the system. These special courses (electives) served besides their educational purpose also the role of didactical test environment to explore the proper use of certain new features.

The standard setup of AMEISE simulations follows a general pattern. It usually consists of the following phases:

- *Preparation:* It is highly advisable that the particular simulation model in use is introduced some days ahead of the first simulation. The type and depth of this introduction varied over the different courses depending on project-management related aspects dealt with in the core of the respective course. Sometimes it was confined to just minimal verbal hints and handing over a short user's guide. But even in these cases, it takes about an academic hour (45 minutes) to present features and idiosyncrasies of the system and to make students adequately aware that a simulated project is just a simulation. Hence, in real projects, they would have to control even further aspects. Till getting in contact with the real system, students can familiarize themselves by means of a flash-animated web-tutorial.
  The participants of recent courses got an AMEISE handout with the most useful hints and/or a list of the virtual software developers including their hourly rates of wages

and a rating of their working experience with several facets of software engineering. Giving them this list saves the time of the interviewing phase before hiring and taking respective notes that would otherwise be necessary (or at least strongly advisable).

- *Tool and Syntax Explanation:* The final contact with the AMEISE environment usually takes place right before the first simulation. Sometimes the participants can even get acquainted to the environment by running a rather simplified simulation using the so-called mini-QA model (reduced size of project, reduced number of available developers with identical qualifications). Allowing students to play around in the context of the real project they have to pursue would also be an option to familiarize them with system specifics. However, we never used the latter option.
- *Planning the first simulation:* The participants are encouraged to prepare their first simulation such that they plan the prospective allocation of virtual software developers to the respective phases and activities within the software development process. How much effort goes into this step is up to the respective student (or group of students). If not prepared ahead of time, planning is part of the respective simulation session itself.
- *First simulation:* Usually, the first simulation will be processed in the classroom in the presence of the instructor. Thus, any ad-hoc questions which may arise during the simulation run can be addressed on the spot.

  Technically, this need not be the case. However our experience shows that with students as well as with people having already industrial project experience, enough questions pop up to warrant the presence of a tutor in the class room.
- *Feedback for first simulation:* Basically, there are four different kinds of feedback possibilities which can also be arbitrarily combined:
  - onLine assessment in the presence of the tutor interpreting and explaining the results provided by the evaluation component
  - self-directed onLine assessment in the absence of the tutor
  - feedback from the tutor in a plenary session with group discussion and the possibility to compare interesting aspects.
  - delivery of a generated evaluation report for each participant.

  We recommend to include at least after the first simulation run the group discussion approach in the options of feedback approaches. It allows students not only to see effects of management decisions by their peers. It allows also the instructor to highlight the difference between effects due to the very nature of a software development task from effects that might be attributed to the specific way certain aspects are represented in the model driving the simulation.
- *Planning the second simulation*: For didactical reasons, a second simulation run is absolutely necessary. It will be a valuable opportunity for the participants to show that they can improve their simulation results. Because of the motivation to achieve better results, the preparation time will slightly increase in comparison to that of the first planning period.
- *Second simulation:* As initial difficulties with the AMEISE environment were already overcome during the first simulation run, the presence of the tutor is not necessarily required for the second run. Usually, the time spent on running the second simulation is less than the time used for the first one.
- *Feedback for second simulation:* The kinds of feedback possibilities basically correspond to those indicated before. If an in-class discussion takes place, not only the results of the second simulation run will be analysed and discussed but also the deviations with respect to the first simulation. The simulation environment currently takes the two simulation runs as distinct simulations though.

In an overall assessment of the approach, one has to see that students need not only feedback on their performance after the simulation runs but to feel comfortable with the system. The amount of information provided throughout their acting as project manager is also important. Here we don’t discuss it as feedback but rather as usability properties of the simulation.

One should state already here that this is aside form the general positive appraisal of the system a constant source of critique. Students want more and “better” information throughout their work. However, in this respect we are rather stubborn. The manager should know what individual members of her or his team are doing at a given time. If not, they have to be asked and in this case, the system gives instantly an honest reply. When asked how far they have proceeded with their work, their answer is biased though by a random fuzzyfication. This is realistic to the extent that different humans would also differ in their optimism/pessimism in assessing the work still to be done. Finally, one should mention that the kind of information provided by the system is realistic indeed. Managers won’t walk into an office and inspect a software artefact to assess when a person will be done and available again. They will ask and get some estimate.

Nevertheless, run-time feedback is an issue where we still explore new options to provide student managers with an even more realistic picture (see section 3.4).

3. Assessment and Evaluation Components

As didactical aims may vary, instructors may customize the system according to their specific didactical aims. According to our experience, it is rather advisable to use the system in a rather plain fashion during the first run. Only the evaluation components should be available. Whether the support components discussed below are made available is a matter of taste (and to a moderate extent also of the performance needed). Other features such as the group comparison or the rollback facility are not recommended for the initial runs, since students might be lured in just playing around and loosing track of the key aim to get familiar with the basic challenges and mayor pitfalls of steering a software development project.

Another alternative to use right from the beginning would be to contract certain features out to an *external software house*. This allows for focusing just on specific development phases while certain tasks are simulated in a reasonable but neutralized manner.

The system has been designed for blended learning situations. Hence, one major difference to SESAM is the support for accompanying feedback during simulation runs as well as support for immediate feedback right after the end of a simulation. Thus students get their first feedback while their memory is still fresh. AMEISE allows for increased learning experience. For this, the SESAM simulator has been extended by several support components. Some of them are integrated into the user interface and are accessible by the students; some of them are for instructors only. Depending on their use they can be categorized as support components, self-assessment, and assessment tools. This section describes them briefly.

3.1 Support Components

As discussed earlier in this paper, project management implies to act on one’s own authority. However, typically one is not really alone and there are colleagues around that might be asked for advice. Having friends around, one might also get advices when not explicitly asking for them. AMEISE meets these requirements by providing both, information sources that get active on demand and agents running in the background.

Providing good advice in situations depending on the students past activities is challenging. It requires the capability of recognizing the situation the student has manoeuvred him- or herself into at this very moment (the state) and furthermore to know about past and (possible) future effects of decisions. Technically this problem is solved by two agents, whose responsibility is to store relevant states and special markers (e.g. reaching of milestones) in the database.

The *State Manager* is the agent responsible for determining and storing state information. A state consists of entities and relations in the simulation engine. Hence, the State Manager directly observes internal SESAM variables and places respective markers in the AMEISE database[4]. Management of these markers depends on specific actions and command of the trainees. Thus the storage is done by an agent (called AMEISE *marking agent*) running in the background of the trainees' client. This agent recognizes (a pre-defined set of) activities and state changes. They are defined by the model-designer and stored as a set of SQL queries in the database. These queries can be simple (representing only queries to match specific user commands), but they might also contain complex relationships between entities in the state space [Nuss 03].
The agent's activities are triggered by state changes in the database. Every state change in the engine results in trying to find suitable SQL queries that are then to be executed. E.g., the set of queries might contain an entry that looks for the beginning of a specific project phase: the specification. Thus it prescribes to listen to a command called "<developer> write the specification", and to make sure that the state space then also contains a <developer> that is working on the system's specification. In exactly that case the marking agent then assigns a label entitled "BEGIN_SPECIFICATION" to this state in the database. The marking agent plays also a special role when comparing simulation runs with each other returning to previous states of the project.

The agents discussed so far are not visible to the user of the system. But there are two support components that are visible. The first one is called AMEISE *Friendly Peer*. It makes use of the information stored in the database. It recognizes markers, the state, and state changes. By chains of rules based on Boolean logic even complex situations and conditions can be covered [Jaeg 03]. These expressions (including relevant explanation text) are stored as SQL templates in the model-relevant part of the database. They are composed on the fly during the simulation run. When made available by the instructor, the friendly peer might, e.g., detect that the student assigned the least qualified developer to the specification task. A small window then pops up and tells the student about the importance of the specification phase and recommends changing the developer.

In contrast to the friendly peer, *advisor* is a reactive support component. It approximates an experienced colleague who might be asked for guidance and advice whenever the student manager feels at loss. The advisor component makes use of the state information and markers kept in the portion of database that keeps record of each individual simulation run. When called, it provides the user a list of questions (predefined in the model specific portion of the database) that might be asked. As an example the student has the possibility to ask whether the choice of the developer writing the specification was a good one or not.

[4]) For a detailed discussion of the AMEISE architecture, see [MHBJN 03] or the AMEISE site at http://ameise-ni-klu.ac.at

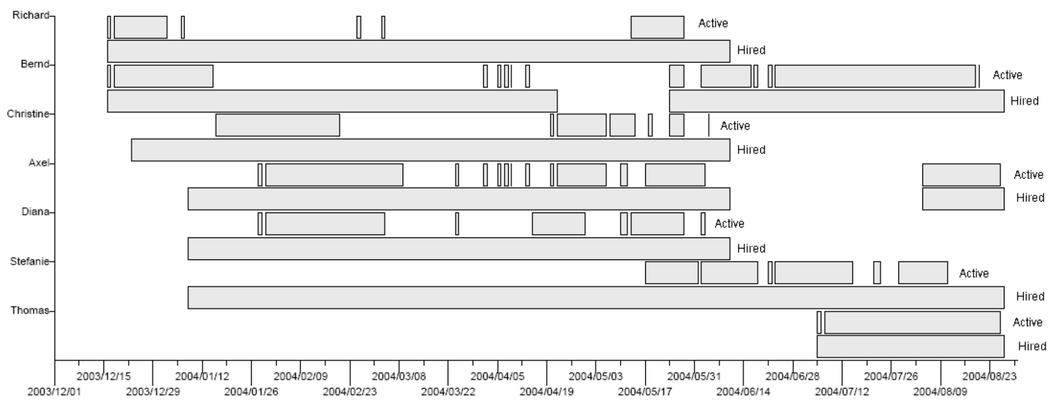


Fig. 2: The AMEISE self-assessment component provides several diagrams that can be generated by the user. Here the deployment of developers is presented (were the bars display the deployment and actual activities of several employees).

3.2 Self-Assessment Component

At the end of the simulation immediate feedback is important. The self-assessment component (also called AMEISE *evaluation component*) fulfils this requirement. When activated by the user at the end of a simulation[5] it generates diagrams and tables one would typically find in an AMEISE assessment report. Fig. 1 and 2 present two of such diagrams. On the right side of Fig. 1 the propagation of errors in documents across various phases is presented. Fig. 2 shows the timeline for each engaged developer while hired (lower bar) and while being active (upper bar). This particular project was not really successful as developers, while being on the project's payroll had lot of time where they were not active.

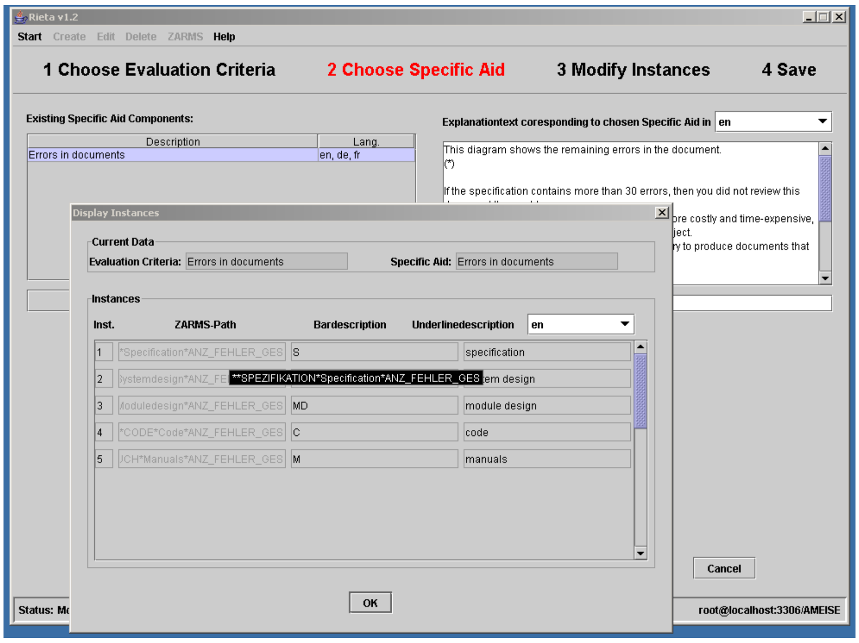
The evaluation component aggregates data stored in the student/simulation-run specific portion of database. As with the advisor component, the logic (including the explanation text) behind the diagrams is stored as SQL query templates in the model specific part of the database. The component itself is well-suited for a first assessment and provides motivating feedback to the student. However, the certain effects are far too complex for automatic evaluation. Therefore, the report cannot replace the detailed assessment report generated by the tutor of the simulation run and the ongoing discussions.

3.3 Assessment Support

Self-explanatory diagrams are necessary, but assessing a project trace needs more than a handful of Gantt charts. As every simulation run is different and might contain different situations of interest, it is up to the tutor identifying those parts of the simulation first and then aggregating the data for presenting them in a suitable and conveying manner. Here, AMEISE provides a tool called AORTA (short for *Analysis Of Relevant Tree Aspects*).

As mentioned above, AMEISE makes it possible to go several steps back in the simulation and to try different strategies. The game trace can then be seen as a tree of paths through the

[5]) The full power of the evaluation component can only be used after the simulation has been terminated. Limited intermediate evaluations are possible, though, at any time.



Existing Specific Aid Components:

Description	Lang.
Errors in documents	en, de, fr

Explanationtext coresponding to chosen Specific Aid in: en

This diagram shows the remaining errors in the document.
(*)

If the specification contains more than 30 errors, then you did not review this
...ore costly and time-expensive,
...ject.
...ry to produce documents that

Current Data
Evaluation Criteria: Errors in documents
Specific Aid: Errors in documents

Instances (Underlinedescription: en)

Inst.	ZARMS-Path	Bardescription	Underlinedescription
1	*Specification*ANZ_FEHLER_GES	S	specification
2	Systemdesign*ANZ_FE **SPEZIFIKATION*Specification*ANZ_FEHLER_GES		tem design
3	Moduledesign*ANZ_FEHLER_GES	MD	module design
4	*CODE*Code*ANZ_FEHLER_GES	C	code
5	JCH*Manuals*ANZ_FEHLER_GES	M	manuals

Fig. 3: The RIETA tool is used to add functionality to helper-components. Here, the diagram of Fig. 1 (right side) is constructed by adding descriptive text and providing attributes to be displayed in the diagram (e.g. the specification's number of errors).

simulation. AORTA focuses on this game tree and is able to display all diagrams and tables of the self-assessment component in respect to these paths. To sustain the tutor the component offers more functionality: it allows for tracing the complete simulation run, comparing several runs with each other, and exporting statistical data in comma separated format (for later external analysis). In addition to that it is also able to export all parts of the assessment report in PDF format, which eases the distribution of the result among the participants of the course.

In the above discussion, only the users' view onto the assessment has been described. At least as important is the backend: the tool that enables adding new queries (and thus functionality) to the supportive, model-dependent components. AMEISE provides a tool called *RIETA* (short for *Rule Insertion and Editing Tool for AMEISE*). It allows to first define so-called "specific aids", which are either references to attributes in the simulation state (e.g. actual costs of the project, or number of errors within a specific document), or the logical combination of and comparisons between them. Instances of these aids can then be combined in order to form diagrams or tables. Fig. 3 shows that these aids can be used to display the bar chart (errors in documents) and can be annotated with some explanatory text.

3.4 Challenges

Immediate Feedback and detailed assessment reports have turned out to be pedagogically valuable. But there are still some challenges. From the students' point of view the feedback is sufficient, but still ever-recurring effects (e.g. productivity loss) have to be explained on simulation basis by the tutor – every simulation run is different. The assessment report contains all necessary information, but the exploration of the relationship within the data needs guidance. At least some of the effort and time spent by discussing a single simulation run can be saved by a more personalized assessment report containing references and context-dependent feedback.

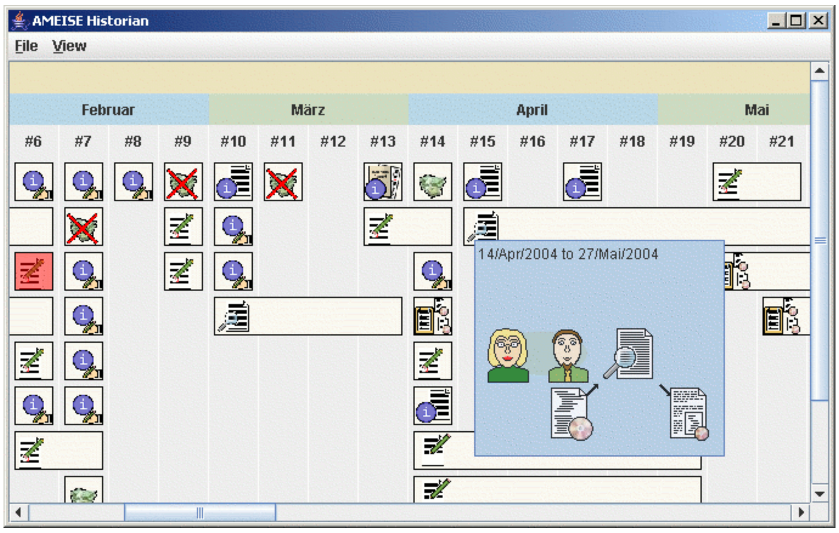


Fig. 4: The assessment of the approach is sustained by a framework presenting the project trace in an iconic manner.

From the tutors' point of view a fast detection of relevant effects (students should learn from) is more important. Scanning the report and looking for effects takes time as it has to be done by hand. Up to now the Historian tool is only able to mark un-successful commands of the user. However, the detection of relevant effects is much more difficult. A typical example: within an assessment report it does not make sense to state that it would have been wise to hire a more experienced developer when none was available at that time of the simulation. Knowledge about the specific context at this point of the simulation is required, knowledge that first has to be stored in order to be incorporated into report-generating tools.

Both requirements are part of ongoing extensions. The assessment report is going to be enriched by references to related data (within the report itself and to related literature), and one thesis is concerned with the identification of the necessary context. In a first step the context information is then to be used to personalize the assessment report. In a second step a focus mechanism will be integrated into AORTA and the context information will be used to ease the highlighting of interesting areas in a simulation trace.

Whereas AORTA is well suited for the generation of reports, up to now the identification of relevant decisions is left to the tutor. To identify critical regions in the trace, tutors are typically looking for failed activities and analyse the trace by looking at the plausibility of the sequence of commands. Both tasks are time consuming. To ease this situation, a component called *Historian* has been introduced. It displays the simulation trace in a graphical/iconic manner, and is able to mark failed activities [Putz 05]. Fig. 4 displays such a typical game trace, and the activity marked in red tells the tutor that it has not been successful.

4. Experience and Effects

As already stated, AMEISE uses the simulation engine of SESAM as provided and still maintained by their original developers [DL 00]. Hence, test simulations could be carried out right from the beginning of the AMEISE project. This allows us to continuously gain experience (like stated in section 3.4) which in turn influenced the overall progress of AMEISE developments. Reporting in detail on the project management results achieved by the students would go far beyond the scope of this paper. Instead, after giving an overview of different course settings and stating some critical aspects related to the standard setup of AMEISE simulations (cf. section 2.2), this section summarizes some noteworthy experience gained and effects observed during various AMEISE courses.

4.1. Settings of AMEISE Simulations

Over the last four years, AMEISE simulations were processed within the scope of several courses on Software Engineering at both Carinthian institutions of higher education, the Alps-Adreatic University Klagenfurt as well as the Carinthia Tech Institute. The number of students attending our AMEISE courses was in the range between 10 and 54.

The first and foremost course in 2002 was attended not only by students but also by lecturers. This course still used the SESAM environment in order to collect initial experience leading to more detailed requirements on the AMEISE project. The next course already applied the distributed AMEISE architecture but still used the pseudo natural language interface provided by SESAM. In spite of numerous complaints about the text-based user-interface, the trainees were very interested in participating and they even volunteered for running their simulations during vacation time.

The general nature of this first series of courses at Klagenfurt University was rather experimental. Performance aspects, motivational issues as well as observations on specific difficulties influenced further developments. In particular, these courses provided essential inputs for designing the support components. The urgency of developing a new, more supportive user interface (see Fig. 1 left-hand side) was recognized, too.

Since 2003, AMEISE has been used in courses on a regular basis. Generally, the standard setup as described above will be applied. In some courses it is up to the students to decide about a possible second simulation run. However, in most cases the students are eager to show that they can improve their results or even beat the results of their colleagues.

At Klagenfurt University, AMEISE perfectly fits the purpose of the compulsory course "System Development Process" in the graduate curriculum of "Applied Informatics". Having gained knowledge from preceding courses on Software Engineering and experience from the internship semester in industry, students attending this course will learn about improving the CMM maturity level of software companies [Hump 98] in order to be able to discus Humphrey's recommendations in the light of their own industrial experience and map them onto small and medium sized companies. The number of course participants usually varies between ten and twenty persons. Undergraduate students have the possibility to use AMEISE in the two courses "Management of Software Projects" and "IT Project Management and Change", the latter is part of the curriculum "Information Management".

At the Carinthia Tech Institute, former undergraduate courses on Software Engineering included the development of a small software system in order to be able to get a deeper understanding of and experience in the software development process [Hoch 02]. The excessive effort for both the students and the instructors was enough of a reason to abandon this project-centred educational model in favour of AMEISE simulations. Hence, AMEISE is used in the undergraduate courses "Software Engineering" of the curricula "Telematics / Network Engineering" and "Medical Information Technology".

A completely different setting was an in-house AMEISE course at a software-developing company. A quality manager and a project manager initiated an AMEISE course for their managers of software projects. Some of the eleven participants already possessed some years of experience in project management. The course started with a half-day lecture on quantitative aspects in project management including Function Point Analysis and COCOMO (the participants already knew about FPA and some of them also about COCOMO II). The subsequent part on AMEISE again followed the general pattern as given above.

Besides this, we made AMEISE also available to other educational institutions[6]. Some used similar approaches as recommended in Sec. 2.2, others departed from these recommendations. We will discus our experience based on the standard setup described above.

- *Preparation:* It is crucial that this phase is seriously considered. However, in spite of insisting on the respective off-line preparation, many students are less eager to read the short manual we pass to them. The on-line tutorial seems to be more appealing. Hence, we feel obliged to keep it current with extensions and changes to the system.
- *Tool and Syntax Explanation:* We consider it still advisable to relegate this phase to the first simulation setting (but keep it separated from this by a short break). To ask students doing it on their own, e.g. by having them run the mini-QA-model, might not be the best use of students time.
- *Planning the first simulation:* Usually students do not invest sufficient time and effort into this task. This is mostly due to their lack of experience in related tasks. A typical students planning time is less than half an hour (in some cases, it is negligible). In a course we held for practitioners, they used between half an hour (minimum) up to little more than an hour. The results of their (first!) simulation showed that this upfront investment paid off well.
- *First simulation:* Usually, the first simulation runs as expected by the instructors. Plans that existed initially that all simulations are unattended done in a remote manner have been given up. There are still too many technical as well as subject-related questions to be addressed.
  In order to fit all students to workstations in a single room, as well as for didactical reasons, we have them usually working in pairs or groups of three. This allows for discussions within the team. We consider these didactically valuable.
- *Feedback:* As a standard feedback procedure, we use all of the following options in our AMEISE courses:
  - immediate onLine assessment in the presence of the tutor interpreting and explaining the results provided by the evaluation component after the first run as well as self-contained onLine assessment in the absence of the tutor after the second run.
  - feedback from the tutor in a plenary session in depth discussion after each simulation.
  - each participant will get a generated evaluation report describing the simulation results.

  Among these feedback possibilities, we consider the discussions as most valuable. The delivery of the written report thereafter is of course a necessity. Experience gained in simulation runs (approx. 12 groups) shows that it takes about 1 hour to produce the assessment report. It then takes about 3 hours to scan the 12 reports, and

[6] AMEISE has already been put into action at Ecole Nationale d'Ingénieurs de Tunis (several times since winter term 2004), at RWTH Aachen (2 times since summer term 2005), University of Applied Sciences Kufstein Tirol winter term 2005), and University of Heidelberg (summer term 2006).

about 4 hours to produce some PowerPoint slides summarizing the main results for later use in the lecture.

- *Second simulation:* We always offer this option and most students benefit from this option. They do it even, if the schedule permits only to do it during vacation time.
- *Further simulations:* As the system is used in several courses, possibilities exist that some students in the compulsory course “System Development Process” had already worked with AMEISE in other courses before. We ask those, to set themselves a specific further goal such as “use minimal number of developers”, “reach some key parameters exactly”, “minimize some key parameters”, … This works satisfactory, as these additional goals are not trivially to obtain and hence, a new challenge arrives. But we do not encourage a fourth simulation, since this seems to be too much of training for the tool instead of gaining project management experience.
  On the other hand, if the number of such students is large in a course, this has overall negative effects, since certain rumours of how to trick the system become student-gossip. This leads occasionally to strange behaviour not intended by the instructors.

For sake of completeness we would also mention that we held so far three AMEISE tutorials for instructors who wanted to use AMEISE with their students. They also comprise the execution of simulation runs. Here, the didactical aims are totally different from those of regular students. Hence, we don’t discus them here any further.

4.2. Experience gained so far

The most important experience was the high degree of the students' acceptance of AMEISE as integral part of the courses on Software Engineering. In addition to the rather theoretical characteristics of knowledge in software project management offered by traditional courses, the big benefit of AMEISE bases on the hands-on experience gained through AMEISE simulations. Hence, students can particularly learn that is not impossible but rather cost-intensive to add quality to a software product in cases when intermediate products already were of poor quality.

A further result was the experience that even with a rather small project (200 AFPs) carried out with a small gaggle of software developers it can easily happen that the student project managers lose control if they do not plan their projects carefully and update their plans continuously. It certainly might be argued against that in real world projects with real colleagues it will not be likely that one forgets whether Richard or Christine was concerned with a particular activity three months ago. The new version of the SESAM user interface and the intended AMEISE "next generation" user interface try to overcome these deficiencies by using graphical elements and ideograms.

The planning dimension is certainly the most significant difference between the students' and the practitioners' simulation runs. While most of the students started with their simulations soon after a brief interview phase in which they learned more about costs and qualification profile of the various software developers, the experienced project managers spent more than one hour on planning and preparing their simulation runs. Although their elaborated plans did not hold till the very end of their simulation, the results of their first simulation runs nevertheless correspond on the average to the results of the students' second simulation runs. The practitioners' second simulation runs even resulted in excellent cost and time outcome as well as in extremely high output quality.

Generally, it was noticed that the preparation phase is crucial to the later results of the simulation runs. The Carinthia Tech Institute students who attended the first AMEISE course spent also a substantial effort on the preparation of their first simulation run. One reason was certainly also the fact that they carried out their simulations at the Klagenfurt University which was motivation enough to strive hard to compare well with university students. Hence, their first simulation runs yielded results above average. This effect could not be observed anymore when the simulations where carried out in-house.

The new AMEISE user interface providing a model-specific command-input list with parameter selection was beneficial to the total duration of the simulation runs. The interaction with the system became notably easier. Thus, the trainees had not to worry about intricacies of the system but could better concentrate on their actual project management tasks. The time spent for a simulation run using the text-based interface exceeded 4 hours on average. With the more usable AMEISE interface the average simulation time was reduced to slightly more than three hours. As usually signs of fatigue will arise between the third and fourth hour of work, a noticeable improvement of the overall results could be achieved.

We would like to stress an experience we had with French speaking instructors that should be mentioned: Although we made any effort, to provide them with a French user interface (English and German were available right from the beginning), we figured not only that our translation was in certain corners incomplete and that the (compiler of the) core of the system had problems in coping with French accents. We have also seen that a few commands had unforeseen interactions between technical and colloquial use of terms. This led to an excessive number of problems, both with the instructors and later with their students. Eventually, it caused us to re-adjust English and German commands to keep the translation consistent while avoiding the above problems.

Another effect we noted already when comparing simulation results from Stuttgart and Klagenfurt, but much more from Klagenfurt and Tunis that the embedding of such a system depends heavily on the overall general as well as university specific culture.

The simulation runs were either carried out by single persons or by a team of two. These different settings did not yield any observable discrepancies in the results. However, in cases of team-work we could observe that upcoming decisions were always discussed between the members. This prevented from quick-click effects which might be possible especially with the AMEISE point-and-click interface. Brief discussions between the group members also helped in bridging the time gap caused by performance problems during early simulation runs.

Last but not least, we got very positive feedback on the two support components, advisor and friendly peer. For both components, the trainees supplied us with valuable input on additional critical simulation situations which can be supported by these components, too. Initial performance problems could be solved so far. While some early simulation runs had to be carried out without using the friendly peer in favour of achieving acceptable performance, recent simulations can fully benefit from all the helper components.

In general, we may report that the students were quite excited about the experience gained with AMEISE. Asked about the difference between having been told most of the things that were discussed in the feedback sessions in earlier software engineering classes and the discussions on the basis of the simulation runs, the standard reply was: “Now, this knowledge is founded on (my own) experience!” or: “But now, I not only heard you telling this, now I felt it!”

5. Outlook

Since we use AMEISE not just as eLearning environment but also as sizable legacy system (total approximating currently 55.000 LOC Java code plus roughly 70.000 LOC of code in other languages), it is a constantly evolving system.

The part that experienced the highest degree of evolution is certainly the user interface. At the time we started our development, SESAM had a text based pseudo-natural language command interface for accepting the student-managers orders and a text based protocol reporting responses of the individual team members or of the system on their reaction to these commands or completion of tasks assigned. Our initial efforts concerned the input side. We offered the menu-based command interface shown in Fig. 1 as an option to the pseudo natural language version. However, from the very beginning, ideas to produce an even more innovative interface were around. The difficulty we met, however, was that this interface had to cope with too many dimensions at a time. To illustrate the problem, consider the interlinking of developer & task at a given point in time. The desire to show the progressing of tasks, i.e. the emergence of an artefact follows from this requirement, adding another dimension. This, of course has to be seen in the context of other artefacts already available and in some situations even in the context of artefacts yet to be developed. The *Historian*, proposed in the context of a diploma thesis [Putz 05] is a conceptual answer for this problem. Its prototype, discussed in section 3.4 is ready for integration into the system. It still needs to survive the real time test in a class-room situation. The same applies for a sliding time wall currently developed as diploma thesis, that provides a compromise between a high resolution view on the current situation and a wide angle perspective on aspects dealt with before (or after) the day(s) currently under focus.

Ongoing research is also needed in the area of focusing on relevant decisions and parts of the simulation trace. To explain effects within the simulation, instructors typically have to know about the rules stored in the SESAM model (that are executed by the simulation engine under specific circumstances). These rules describe positive and negative effects taking place when proceeding in the simulation. To scan and know about hundreds of rules is not feasible, and one diploma thesis is currently developing mechanism to automatically deduce the relevant rules that lead to a very specific situation in the simulation run. With this a more crisp assessment report and extensive feedback can be provided.

In this list of further extensions, we like to mention also one that was demanded by students (and even more violently by practitioners) was a support tool informing student-managers who has been working on a given artefact already. This information is helpful when assigning correction tasks as well as when assigning QA-tasks, notably reviews. We rejected this demand so far with the argument, that it is the managers' duty to keep track of the project and providing too much system support would run against one of the individual sub-purposes. Of course, memorizing all these things is hardly possible. But one might/should take notes. Right now, we are on the brink of giving up to the argument of "Not I am doing it, my secretary does it". Thus, eventually, there will be an administrative assistant as support tool. But in contrast to other support tools, this *administrative assistant* has to be paid a salary and therefore the student manager has to decide whether this service will be worth the money needed.

A further item to be mentioned is that, though students are free to issue basically any command at any time in the project, the empirical data the project is based from is from linear (waterfall) document driven models. It would be interesting to establish a new model according to agile processes. An initial attempt in this direction did not yield the desired effect due to lack of validated data. Currently, attempts are under way to overcome this bottleneck.

Aside from these technical extensions, quite a number of didactical experiments are in the offing. These have to be done with care, since they must not be conducted in conflict with the educational goals we are pursuing with AMEISE. Though we use the system in several courses, there is still only limited room for carefully planned experience. Hence, we are happy that institutions who acquired after taking part in an AMEISE-tutorial an AMEISE licence become development partners at least in so far as the otherwise free licence requires that we obtain feedback on their use of the system and obtain a report on the experience gained. Thus, an open-source community, exchanging software and experience is slowly developing.

6. Summary

The paper reports on AMEISE, an educational simulation environment to allow students to acquire project management experience in a non-threatening situation. We consider it important to stress that in order to obtain the educational merits, adequate preparation of students and proper embedding of the simulation environment in an advanced software engineering course is mandatory.

REFERENCES

[Hump 98] Watts S. Humphrey: *Managing the Software Process*; Addison-Wesley Publ., SEI Series in SE, 1989.

[DL 00] Anke Drappa und Jochen Ludewig: *Simulation in Software Engineering Training*; in Proceedings, 23rd International Conference on Software Engineering, IEEE-CS and ACM, May 2001: 199 - 208.

[Hoch 02] Elke Hochmüller: Project-centered Software Engineering Education, Technical Report, Carintha Tech Institute, Klagenfurt, Austria, Oct. 2002.

[MHBJN 03] R.T. Mittermeir, E. Hochmüller, A. Bollin, S. Jäger, M. Nusser: AMEISE – A Media Education Initiative for Software Engineering: Concepts, the Environment and Initial Experiences; in Auer (ed): Proceedings International Workshop ICL – Interactive Computer Aided Learning, Villach, Sept. 2003, ISBN 3-89958-029-X.

[MN 03] Markus Nusser. (In German) Projektunterstützung durch Verlaufsanalysen und Agenten. Univ. Klagenfurt, Institut für Informatik-Systeme, 2003.

[PP 05] Peter Putzer. Trace Representation of Simulated Software Development Processes. Univ. Klagenfurt, Institut für Informatik-Systeme, 2005.

[SJ 03] Susanne Jäger. (In German) Entwicklung eines elektronischen Tutors zur Analyse von Projektverläufen. Univ. Klagenfurt, Institut für Informatik-Systeme, 2003.

[Mitt 06] R. T. Mittermeir: Facets of Software Evolution; in: Madhavji, N.H., Lehman, M.M., Ramil, J.F. and Perry, D.W.: Software Evolution, Wiley 2006.