An Introduction to AMEISE
Software Management Training focussed on Quality

Handout for Trainees

During the SESAM/AMEISE training you have the challenge to plan and manage a simulated software project at the SESAM/AMEISE[1] simulator.

Your task is to successfully conduct a software development project. The details of the project will be related to you after starting the system.[2] Success will be calculated according to the degree of the project meeting the customer's requirements in terms of both, functionality provided and non-functional system and project parameters attained. Hence you have to take a close look at the time arrangement and the quality requirements of the customer. Managing your project, make sure that you stay within the allocated budget and make sure to be ready for delivery on time.

When assessing your performance as project manager, it is not only important that you reach the given objectives. It is also important to consider in which way (social cost etc., which are not modelled) you will reach them.

1. Using SESAM/AMEISE

When using SESAM/AMEISE you act as software development manager. You steer the project in hiring employees and communicating with them using the commands listed in the appendix, List of Commands.

In addition to these application-commands, there are a few system commands which are to be issued by pressing the respective button on the graphical interface.

- *Proceed*. After the system has been initialized, a *proceed* causes the simulator to process all commands that have been composed on the respective day and eventually advances the clock to the morning of the next day.
- *Send command* is a button in the AMEISE SimulationPlus interface. It is to be pressed after the user has composed a complete and syntactically correct command from the SimulationPlus composition panels. (These composition panels provide selection of commands, persons, or documents).[3]

---

[1]) AMEISE is a project developed by Universität Klagenfurt, Fachhochschule Technikum Kärnten, and Universität Linz, based upon the SESAM project of Universität Stuttgart. In parts applying to both systems, the original SESAM, which was further extended by its developers, and the derived and extended AMEISE, the system is referred to as SESAM/AMEISE. Portions pertaining just to AMEISE are labelled with "AMEISE".

[2]) It is also possible that your instructor gives you the details of the project already ahead of time. This will allow you to plan the project carefully before starting the simulation process.

[3]) In the plain text interface called "Simulation", hitting the return button after typing a complete command phrase has the same effect as hitting the *send command* button in the Client+ user interface.

All other system commands have to be selected from pull-down panels. These commands comprise work organisation (e.g. for loading the intermediate results of a halted simulation) as well as switching between user interfaces (e.g. Treeview or AMEISEWall) or switching from development mode to the evaluation mode or to AMEISE aid components.[4]

2. A typical simulation run

You are invited to peruse the AMEISE tutorial at http://ameise.uni-klu.ac.at.[5]

2.1. Registration

Following the installation specific instructions of the tutor, your system will show a login window for registration and a window in the background called AMEISE Client with pull-down menus for *Game*, *Options*, and *Help*.

You can start the simulation by identifying yourself with *user-id* and *password* assigned to you from your instructor. The system will respond by opening a small window for a simulation run selection. Having made your choice you close this window by pressing *Choose game* on the bottom of this little window. When you start a specific simulation run the first time, the Client performs some initialization routines. So please be prepared that it takes some seconds till the Client accepts your first commands.

The sequel of this introduction assumes that AMEISE is operated with the QA 200 (for Quality Assurance, 200 AFPs) model driving the simulation. [6] [7]

2.2. Initialization

Initially, your project has no employees assigned. There are just a number of currently idle persons around. *Show available developers* shows a list of available developers. You can inspect the vitae of each of them by issuing the command *information about developer*.[8] After having obtained sufficient information about them, you should be ready to *hire* one or several of them.

- Take care! Your budget is tight and the time limit, though 9 month, is quite tight too. Hence, be careful in your decisions to staff your project. Wasting manpower will waste your financial resources. Being overly prudent might block you from meeting the deadline!

---

[4]) Not all of them might be available in your version as your instructor might decide that you should strive to work without them.

[5]) Unfortunately, the tutorial is currently available only with explanations in German language. However, you could still get an overall impression of the look-and-feel of the system.

[6]) The SESAM/AMEISE simulator is capable of simulating quite different processes, since the specific knowledge about the simulated process is kept in a rule base which models the specific application. In the case described here, this rule base describes software development according to a conventional, waterfall like, phase driven process.

[7]) The QA-model allows you to focus realistically at developing quality software without guilt-edging the system.

[8]) Possibly, your instructor has given you already a summary of the vitas of all developers available to be employed in your project. In this case, your instructor probably has made the remark that you should choose the ones to start with in a preparatory phase and do also as much pre-planning of the project before you start with the simulation run.

- You need not hire your complete team on the very first day. Those developers you did not hire will remain interested to join your project.
- Using the command *release*, you may release a person who was working on your team. (The person has to be idle in your project before you can release him or her). To release a person to the company budget is not a social problem; you may re-*hire* the very same person at a later point in time.
  But release-and-hire sequences are risky in so far, as the company may assign some other duty to a released person and this other duty has priority. Thus, a released person might be otherwise busy for a period of 0 to 60 days (randomly). The command *hire* remains unsuccessful for this developer till this person is available again.
- If you have ever forgotten who is working on your project, – What a manager are you, if this could ever happen? – you can ask the system, using the command *show hired employees*.

Managing is planning. Hence, as project manager, you have the chance to run after the project and be curious what its outcome will eventually be, or you can strive to control it. In the latter case, you will, based on the vitae of the developers and your knowledge about software development produce a plan that indicates at which time the project will need actors possessing certain specific qualifications. From this, you can plan whom you will hire when and when you will release a person and either reduce your staff or replace this person with another one, more qualified for the task.

Certainly, projects are never run exactly like planned. But the better you plan it, the better you can react on surprising situations.

2.3 Normal operation

A simulation run consists of a sequence of commands addressed to the virtual employees and *proceeds*[9]*)*. The commands and the feedback are presented in English.

Some commands, notably those which *inspect* the intermediate status (quality, completeness) of some document or what a developer is currently doing, give immediate feedback. In the list of commands they are preceded by an asterisk. *Determine spent resources* inspects the financial situation of the project and gives also immediate feedback.

All other commands are, after being composed, queued when issued using *send command.* The simulator executes these commands only after the user has entered a *proceed.* Only then, the system returns the feedback. All assignments of tasks fall into this category. (E.g., *specify Diana,* Diana might respond that *she is happy about this assignment* and start working on the specification).

- After *send*, a command is just echoed in capital letters in the “feedback window” of the client. This echo, followed by *Command accepted*, indicates that you have issued a syntactically correct command understood by the system. But only the feedback after proceed can assure you that the system understood the command in the way you wanted it to be understood. – Stay alert till you get it!

[9]) *Proceed* as well as *send command* are commands that communicate with the system rather than with the employees.

- The feedback *Unfortunately SESAM does not understand you!* points to a syntactically incorrect command (incomplete filling of parameters, mistyped command in textual user interface).
  It is unlikely that you get this message when using the SimulationPlus user interface. When using the Simulation interface, you have to be careful, since SESAM is case sensitive. For very long commands, you might rely on the callback-mechanism to choose one of the alternatives in a correctly typed manner.

- You have also the possibility to *proceed* not just to the next day but for more than one day (e.g. by issuing *proceed 5 steps*, the manager arranges for 5 days ahead of time, since she plans to go on leave for a week).
  But be careful, you can react to messages you get during your "vacation" only after you are back, i.e., on the $(n+1)^{st}$ day of work. To control project time for keeping the deadline you might want to advance the clock of the project for several days only if you are sure, nothing of importance will happen in this period.

- As in real life you have no possibility to wait after each action what its consequences might be. So think twice what set of actions should be done on a day.

- Each action, each command to employees, consumes some of the project manager's time. Hence, if you issue a very long list of commands (exceeding a model specific limit) before issuing a *proceed*, the system might determine that the day is over and the simulation proceeds one step[10] without an explicit *proceed* command.

- After each *proceed* the model gives you feedback.

If there is no feedback from the simulation (even after the next *proceed*), the system had problems with the execution of this command. A lot of reasons might cause a command to fail. Possibly, the respective developer was not hired or the preconditions for the assigned activity were not given. There are some exceptions, which are discussed in section 2.5.

Because of the complexity of the model (a lot of calculations must be done) the response time of the system after *proceed* or *proceed n steps* could increase during the simulation run. Please wait patiently until the system returns with feedback or with the new simulation date. (Also the initialisation of the model at the beginning of the simulation run consumes some time.) But the time used to compute the next state after completing the commands of the day has no effects on the project time.

2.4. Terminating a project or interrupting a simulation

As each change of state is recorded in the data base, you need not call for intermittent saves. You may stop the simulation at any time and resume your simulation run in the state where it has been closed, using the pull down menu *Game*.

To terminate a simulation, there are two options:

- With *deliver system* you deliver the system to the customer. This is the very last step to be done in the project. After the *deliver system* command has been issued, the simulator will start to perform clean-up operations. Those compute initial evaluations that are necessary for producing the customers report sent to you in his letter of ac-

[10]) The SESAM simulator has an integrated calendar. This ensures that developers only work on working days and not at weekends. Therefore, SESAM skips the weekends and shows the next working day. However SESAM doesn't know public holidays of your country.

ceptance (or complaint). Hence, you have to perform all activities you consider important (e.g. releasing personnel) before delivering the system. But obviously, you can switch to the explanation component panel (EC-Unit) and check the success-parameters of your project even after system delivery.

- With *finish project* you abort operation of the simulator. This tells the simulator that you are no longer interested in any evaluations resulting from the work performed. Hence, the SESAM core can perform no comprehensive ex-post analysis after this command. (Punctual AMEISE evaluations are still possible though).
  Hence, when you are in a hurry and run out of your personal working time, it is advised that you attempt to deliver even a lousy system. If some minimal requirements are met, your performance up to the point where you hectically attempted to close down work can be analyzed by your instructor and at least feedback on your performance in earlier phases can be given.

A better way to react to personal time pressure is to break and resume your work at a later point in time. You can stop your work at any time after having received the response from the system to your most recent command. You do so using the respective system command of the pull-down menu and close down the client. When re-starting the system with the proper model in the same round in which you saved your work, the system will resume the simulation at the point where you interrupted. – Just make sure that not only the system remembers your previous decisions. You also should know what has happened in the project before you broke. To help your memory, you can look at the feedback window and scroll up to the day(s) that elapsed your mind.

2.5. Special features of the QA model

This section describes special features of the QA model. To eliminate problems some of these features are discussed here.

1. The *QA model* is based on an empirical data basis. The quantitative parameters reflect projects conducted according to the waterfall model. Therefore, it is wise keep with the phase-sequence of the *waterfall* model to the extent possible. You should also take a look at the reference documents for the particular project phase. (E.g., an integration test cannot be done, if you haven't integrated the components before. This is as obvious as many other logical necessities. However, so far forgetting integration has been a very frequent mistake of student-managers).
2. The model foresees the following *phases or activities*:
   - requirements elicitation,
   - specification,
   - architectural or system design,
   - module design,
   - coding,
   - integrating (modules or subsystems),
   - writing (user-) documentation (handbook, manuals).

Each of these phases or activities (depending on perspective) yields a *corresponding document*. (Except for pure requirements elicitation, it yields its results directly into specification).

Non-executional documents may be subjected to a review. Executional sub-products may be subjected to tests. Depending on level of aggregation, managers can call for module test, integration test, or system test.

Each quality assessment activity calls for an associated corrective activity afterwards. Corrections can be made for a specific document. With *correct all documents*, developers can be assigned the task to update all downstream documents existing (or partially existing) so far according to a given review or test-report.

- The system will allow you to ask your employees to perform tasks in any arbitrary order. But you will get a successful product only, if the order of operation is somehow meaningful. This implies that, normally, you do not resume a phase that has already produced a (somehow) complete document. You rather ask somebody to *correct* this document using an appropriate test- or review-report.

3. You need not follow any prescribed sequence of activities when assigning tasks to developers. But the task assignment has to be reasonable, i.e., the developers have to have a basis to work from.

   - The QA Model assumes that such a reasonable basis is lacking, if the document on which the current work should be based (usually the result of the previous phase) is not completed to at least a degree of 50 %.

4. Plan your *personnel resources* as carefully as you would plan them in real projects. Be careful and look at the qualifications of each developer and decide which of them is relevant for the project and in which project phase you will need them. You should also think about the period a developer needs to feel home with the project.

   - The skills of persons do not change throughout the project. Thus, there is no basic learning. But persons remain familiar with the work they did and they will remember work they have once seen.

   If you do no longer need a developer, you might take him or her off your payroll by firing this person. But you can only *release* a person that is currently idle. In case you have to release somebody for urgent budgetary reasons, you have first to relieve this person from work by asking him/her to immediately *quit* the current activity or to *finish* it by stopping whenever a subtask or work package has just been completed.

5. Time consuming activities, like writing code or designing the system, will be decomposed into smaller parts (*work packages*). Therefore, it is possible to ask more than one developer to work on the same document in parallel.

   But be careful and think about the communication effort, which will be needed to organize the teamwork. In extreme cases, communication overhead can eat up more time than an additional person may bring to the project (remember Brooke's law)!

6. The *date* which is shown at the last position in the feedback window is the actual simulation date. All application commands sent will be executed at this day (The system command *proceed*, however, advances the simulation time one working day. On Fridays this might be up to three calendar days).

The date format is ISO-norm, that means ‘year/month/day/hour:minute’. Weekends will be skipped by the system automatically.

7. *Reviews* require many eyes. You may form review teams out of two or three developers. You may also form a team of two developers and a customer to review an artefact.

   A customer representative, due to the different background, reviews a document in a different manner than a software engineer. The customer representative will, therefore, find different categories of mistakes.

   Reviewers need some time for preparation before they meet to discus the artefact and report errors. Since reviewing is a strenuous activity, each person can only spend a portion of the day for reviewing. Hence, a person can perform a review and perform some other development activity on the side. – Obviously, this activity is slowed down by the review task. But as the QA-model currently knows only full time assignment, persons acting exclusively as reviewers will be quite costly.

   Typically, review-teams report intermediate results in intervals of two to three days.

   - If you ask an author of the document to perform a *review* of a document he or she has developed or helped to develop, this author will not only be blind against the errors contained in the document. He or she will be so defensive that even the otherwise unbiased reviewers will become blindfolded. Hence, you should never attempt to include in a review-team an author of the document to be reviewed.

8. If you assign an activity to a person, which you haven’t hired already, this person will ignore your message. Only after the next *proceed* command you will get the feedback that this person was not assigned as developer to your project. In this case you have to hire the developer and repeat the command. Thus, be sure that a developer is hired when you give him/her a task.

9. If the system gives no feedback (even not after the next “proceed”), the respective command was not executed. This can happen for different reasons.

   - The system reacts favourably after the current proceed, but reports after the *proceed* thereafter (i.e. after 1 day of work) that the task has been completed, but nothing was achieved (e.g. no errors found). In that case, the system determined that the respective activity could not be done in a meaningful way (e.g., you ask for *system test* but have not integrated the subsystem before) or you are asking for a task that was already achieved (e.g. *review* a document whilst this document has been reviewed already but you have not yet assigned anybody to perform the *corrections* called for in the review report. – After these corrections have been made, you may call for a second, third, … review at any time of course).

   There are some exceptions to the feedback principle, though:

   - *Finish activity* returns no feedback, if the appropriate developer has currently no task to perform.
   - *Show available developers* returns no feedback, if there is no available developer left.
   - *Show hired employees* returns no feedback, if there is no developer hired yet.

- *Deliver system* executes automatically three *proceeds* representing project clean up and post-mortem activities. The reactions of the customer will be shown three days after you delivered.
- *Integrate xyxyxy* is only successful if at least 50 % of the requirements are already realised in code. Otherwise the command will not be executed.

Some situations where user actions last without an effect:

- If a developer executes a task, then you cannot assign him/her the *same task* again. Such a command has no effect and the system returns no message.
- A developer can only be *released*, if he/she isn't working on a document or reviewing some document. If these preconditions are given, the command will be executed and the system will immediately return a message.
- *Release* und *hire*: If a developer was released, he/she might get a task from someone else. Hence, a released developer might become non-available for a period of up to 60 days (random). This means, that you cannot assign him/her to your project within this period. The command *hire* remains unsuccessful for this developer until this employee is available again.

10. There are also situations in which the system returns feedback only after more than one *proceed*.
    - With *finish activity* the system returns the feedback not until the developer has finished his/her task. This might take some days.

    In this case the user may only find out after executing more than one *proceed* whether a *finish* command was successful or not.

11. A developer can only work on one task at a time. E.g., a person cannot design the system and implement it in parallel.

    But it is possible to ask a developer to perform a review in addition to a normal development task, like writing the code, design or test, … (see remarks on reviews).

    If you assign a developer, who has already something to do, an additional full task, one of the following situations might result:

    a. Because of the new task, he/she finishes the actual activity (only the part at which he/she is working about). This can take a while. After the activity is finished the system returns the message that the developer has accepted the new task.

    b. The developer ignores the new task for the time being, but he/she starts the new activity after he/she has finished the old task. In this case you get no message from the system.

    c. The developer ignores the new task even if he/she has finished the current duty. In this case (most probable conflict resolution) you get no message from the system concerning the new task. After reporting that the current task has been completed, the respective developer is idle as if you have never told him anything to do.

    Maybe it's sometimes not clear whether a developer acts according to case b or c. Therefore, it is strongly advised to let each developer complete (or finish, or quit) the current task before you assign him/her another one.

12. The model knows weekends and skips them automatically, but public holidays are not considered by the model. Further, the QA model assumes a very dedicated and healthy team. Thus, no days off for vacation or sickness leave are explicitly considered.
13. The time in the date specification changes twice a year for clock change between summer time and winter time (at the end of March and at the end of October). But time is irrelevant for the QA model, only the day is important, so do not worry about it.
14. Amounts over a million Euro will be shown in exponential notation (1 million = 1.00E06).
15. Reviews and tests detect faults or failures respectively. They do not correct them! You have to ask a developer to correct them. If the faulty document was already used as reference for another document, you have the possibility to ask a developer to correct the errors, detected in the reference document and also in all following documents. It is advisable to use the command *correct all documents* for this end. But you may of course correct each of the documents individually.
16. Don't forget to integrate the code whenever you have performed changes on it.
17. The software cannot be delivered to the customer, if there are less than 50 % of the requirements integrated.

Now, good luck for your simulation run!