Prof. Dr. A. Podelski,
Dr. B. Westphal

Summer term 2021

Softwaretechnik/Software Engineering

`http://swt.informatik.uni-freiburg.de/teaching/SS2021/swtvl`

Exercise Sheet 2

Early submission: Monday, 2021-05-10, 13:00 Regular submission: Tuesday, 2021-05-11, 13:00

Exercise 1 – Cost Estimation (7/20 Points)

This exercise is about estimating the effort for the development of a video game as specified by the requirements given in Appendix A.

In the following, assume that the development is conducted by a team of 6–7 people, all of them computer science students with typical 3rd-semester programming knowledge. The project is part of the students' study plan, and each team is supported by a tutor. The final result (game and source code) is assessed in an expert review, but there are no plans to distribute the created software to users, i.e., there is no need for later maintenance releases.

(i) If each team member contributes ca. 180 hours of work in total, what is the overall development effort in *person-months* (PM)? (1)

Hint: Submission are only comprehensible if the used definition of person-month is provided.

(ii) In the tutorial on this exercise sheet, we want to apply the Delphi method to estimate the size of the video game code given the requirements.

To prepare the tutorial, state your team's estimation of delivered $KLOC_{pars}$. (1)

Hint: Recall that an essential ingredient of a Delphi estimation is a thorough explanation of the reasons for the stated estimation.

(iii) COCOMO, intermediate level, considers effort coefficients based on so-called *cost drivers* (cf. Table 2 on page 6; a short explanation of the cost drivers can be found in Appendix B).

Determine ratings for all cost drivers for the game project.

Explain your rating for the cost driver 'Required Development Schedule' and for two other cost drivers: One where the factor determined by your rating is closest to the lowest factor and one where the factor determined by your rating is closest to the highest factor. (3)

(iv) Categorize the considered video game according to the COCOMO 81 software project types as indicated in Table 1 and briefly explain how you chose the project type.

Then calculate the estimated effort E in person-months by using the COCOMO formula

$$E = a \cdot (KLOC)^b \cdot EAF$$

where a and b are the coefficients specified by the project type in Table 1, $KLOC$ is your estimation from Task (ii), and EAF is the *effort adjustment factor*, calculated as the multiplication of the factors for all cost drivers (see Table 2) according to your categorization.

How big is the difference between the planned effort according to Task (i) and the result of the COCOMO calculation? What are possible reasons for the deviation? (2)

Characteristics of the Type				**a**	**b**	**Software Project Type**
Size	**Innovation**	**Deadlines/ Constraints**	**Dev. Environment**			
Small (<50 KLOC)	Little	Not tight	Stable	3.2	1.05	Organic
Medium (<300 KLOC)	Medium	Medium	Medium	3.0	1.12	Semi-detached
Large	Greater	Tight	Complex HW/ Interfaces	2.8	1.20	Embedded

Table 1: Classification of software projects and corresponding coefficients for the COCOMO model.

Exercise 2 – Process Model Application (8/20 Points + 5 Bonus)

Assume you are the manager of a company that develops radio communication technology for fire alarms. Your company is hired by an external client to develop the software of a new radio module for fire detectors. Once developed, the module needs to be certified to comply with its corresponding industry standard, and you are also required to provide technical documentation.

(i) Consider the activities in Figure 1 (together with their respective artefacts and responsibilities). Those are the established activities in your company.

Provide a *graphical process model* for delivering a finished radio module by using those building blocks. (5)

(ii) Your company has the following staff available, with their corresponding qualifications for roles:

- Alex: Requirements analyst, Test manager
- Dana: Product manager
- Robin: Certification manager, Technical writer
- Chris: Software developer, Software tester, System tester

All of them have full-time availability (1 PM per calender month), except for Robin, who works only part-time (0.5 PM per month).

Consider the effort estimates indicated in Figure 1 and analyze the expected total effort of the project and its expected minimum duration in months by assigning the staff to each of the roles required and considering their availability. The project is supposed to start on 2021-06-01.

Present your analysis using a *Gantt chart*[1]. (2)

(iii) To validate the process model from Task i, that is, to check whether actual processes are sufficiently well represented in the model, we can try to match it against (phases of) actual processes.

For this task, assume that it is not uncommon in this company that, four weeks into developing the software module, the QA identifies issues and raises them as part of QA results. It usually takes the developer(s) 2 days to analyse the issue and fix it, and it takes QA people another day to confirm that the issue is no longer observable.

Discuss how and in how far your process model from Task i matches the assumed process scenario. (1)

(iv) Now assume that your customer is experiencing delays due to understaffing. The customer wants to extend your contract to also develop the software for the sensor module of the fire detectors. Your company would still be responsible for quality assurance, certification, and documentation.

[1]See `https://en.wikipedia.org/wiki/Gantt_chart`.

At the same time, an *intern* with the same capabilities as Chris but with only part-time availability (0.5 PM per month) has joined your company and can be considered in addition to the staff named in Task (ii).

a) Extend your graphical process model from Task (i) to include the development of the new module. Clearly indicate which parts you have added. (3 Bonus)

b) Analyze the expected total effort and the minimum duration of the expanded project with the given staff including the intern. (2 Bonus)

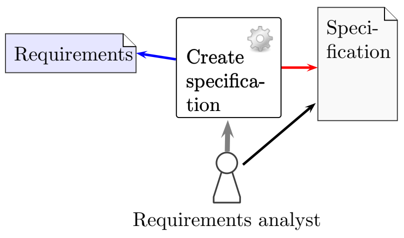
Exercise 3 – Creating Process Models (5/20 Points)

Working on software engineering tasks is an activity and the creation of each solution to a software engineering task does have a process. So it should be possible to model these processes.

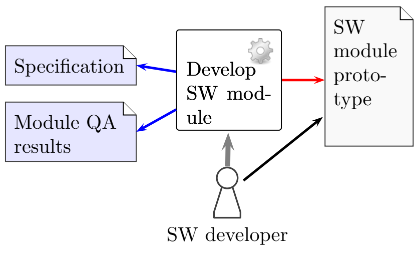
(i) Create a process model of the way of working that is employed by your team between the issue of an exercise sheet of this course and your final submisison (or make up a plausible way of working, if you do not want to disclose the one that your team actually follows). Use the basic process model building blocks (activity, role, etc.) and briefly describe the activity, role, etc. that they represent. (3)

(ii) If a team is looking for a way of working that promises that every team member gets about the same learning effect, would you recomend the approach that you have modelled in Task (ii)?

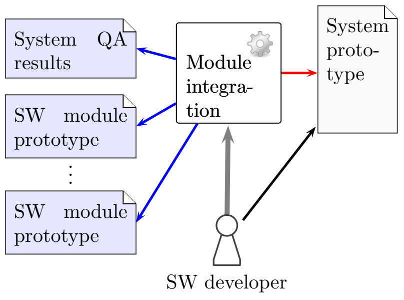
Why (not)? Discuss advantages and disadvantages on the basis of the process model. (2)



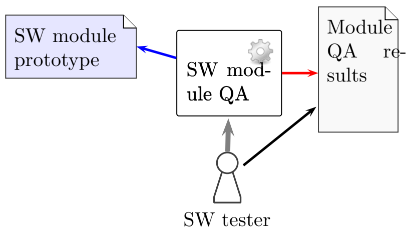
(a) Create specification. (0.5 PM)



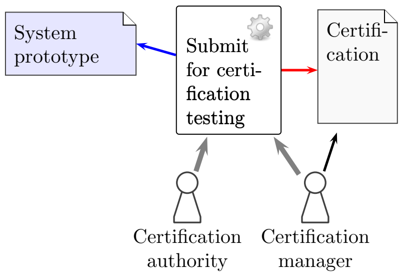
(b) Develop software module. (1.5 PM)



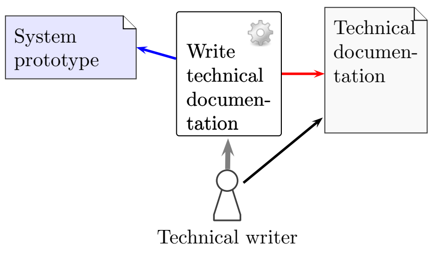
(c) Module integration. (0.5 PM per module)



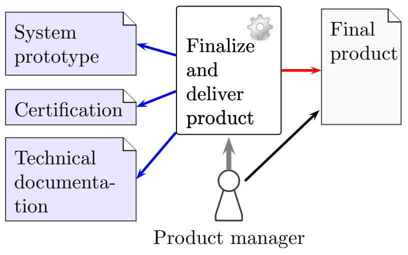
(d) Software module quality assurance. (0.5 PM)



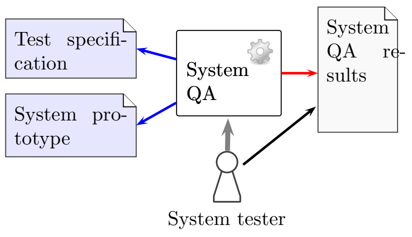
(e) Submit for certification testing. (0.5 PM)



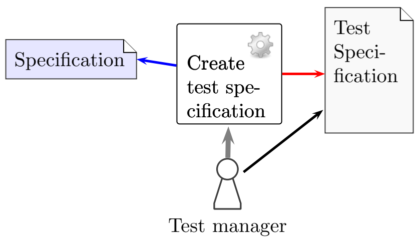
(f) Write technical documentation. (0.5 PM)



(g) Finalize and deliver product. (0.75 PM)



(h) System quality assurance. (0.5 PM)



(i) Create test specification. (0.25 PM)

Figure 1: Activities, artefacts, and responsibilities of your company.

A Requirements for the Video Game

Your task is to develop a video game that satisfies the requirements described below. *A German version is available at: https://sopranium.de/Anforderungen*

Functional Requirements

- The game must have either 2D or 3D graphics (no ASCII).
- There must be at least two players, one of which has to be human.
- Game characters have to be controlled indirectly.
- It must be possible to save and load the game at any point in time. However, the save/load function does not necessarily have to be controllable by the player.
- There must be a pause functionality in the game.
- The game must contain at least the following types of objects:
  - at least 5 controllable objects,
  - at least 5 selectable objects,
  - at least 5 non-controllable objects, 3 of which should allow collisions, and
  - at least 3 controllable objects that allow collisions and are movable (from now on called *active game objects*)
- The game must be able to handle at least 1,000 active game objects simultaneously. The final game does not necessarily have to have that many objects, but at least a tech-demo must be available that can demonstrate this functionality upon request.
- The game must comprise at least the following types of actions:
  - At least 10 different actions (such as walk, use abilities, etc.)
  - It must be possible for all active game objects to move from any arbitrary point in the game world to any other reachable point, without obstructing each other or getting blocked, etc.
- The game must contain sound effects and music.
- The game must collect and display at least 5 different statistics.
- The game must have achievements.
- The game must run in real time: Players must be able to execute actions while their opponents execute actions and must be able to react at any time.

Quality Requirements

- Develop a *good* product. Good means, that it must appeal to your customers. It is your task to convince them that your game is the best.
- The quality of the graphics is not relevant for the game ratings.
- The graphics of the game should be coherent.
- The acoustic effects should be coherent.
- The texts displayed by the game must be free of grammar and spelling errors. In case special characters are used, they must be displayed correctly.
- The rules of usability for games must be followed.

Additional Constraints

- The game must be written in the programming language C# or F#.
- You must use the framework MonoGame version 3.6.
- The game must run on Windows 7 x86/x86_64.
- You should use Visual Studio 2015 as integrated development environment.
- Once every week the game code must not generate any warnings or errors (either from the compiler or from ReSharper).
- Every version committed in the version control system must compile and generate an executable.

B Cost Drivers of the COCOMO 81 Intermediate Model

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware Attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.85	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.85	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Table 2: Cost drivers and corresponding factors for the COCOMO intermediate estimation model (cf. Appendix B).

Required Reliability

This reflects the extent that a software product can be expected to perform its intended functions satisfactorily.

- Very Low: The effect of a software failure is simply the inconvenience incumbent on the developers to fix the fault.
- Low: The effect of a software failure is a low-level, easily-recoverable loss to users.
- Nominal: The effect of a software failure is a moderate loss to users, but a situation for which one can recover without extreme penalty.
- High: The effect of a software failure can be a major financial loss or a massive human inconvenience.
- Very High: The effect of a software failure can be the loss of human life.
- Extra High: No rating - defaults to Very High.

Database Size

This is the relative database size to be developed where size refers to the amount of data to be assembled and stored in non-main storage: D/P = (Database size in bytes or characters) / (Program size in LOC)

- Very Low: No rating - defaults to Low.
- Low: $D/P < 10$
- Nominal: $10 \leq D/P \leq 100$
- High: $100 \leq D/P \leq 1{,}000$
- Very High: $D/P > 1{,}000$
- Extra High: No rating - defaults to Very High.

Product Complexity

Complexity is assessed as the subjective average of four types of control functions: control, computation, device-dependent, or data management operations.

Control Operations

- Very Low: Straight-line code with a few non-nested structured programming operations: DOs, CASEs, IF-THEN-ELSEs. Simple predicates.
- Low: Straight forward nesting of structured programming operators. Mostly simple predicates.
- Nominal: Mostly simple nesting. Some intermodule control. Decision tables.
- High: Highly nested structured programming operators with many compound predicates. Queue and stack control. Considerable intermodule control.
- Very High: Reentrant and recursive coding. Fixed-priority interrupt handling.
- Extra High: Multiple resource scheduling with dynamically changing priorities. Microcode-level control.

Computational Operations

- Very Low: Evaluation of simple expressions: e.g., $A = B + C * (D - E)$.
- Low: Evaluation of moderate-level expressions, e.g., D = sqrt(B^2 - 4.0 ∗ A ∗ C).
- Nominal: Use of standard math and statistical routines. Basic matrix and vector operations.
- High: Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns.
- Very High: Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations.
- Extra High: Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data.

Device-Dependent Operations

- Very Low: Simple read and write statements with simple formats.
- Low: No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap.
- Nominal: I/O processing includes device selection, status checking and error processing.
- High: Operations at the physical I/O level (physical storage address translations; seeks, reads, etc). Optimized I/O overlap.
- Very High: Routines for interrupt diagnosis, servicing, masking. Communication line handling.
- Extra High: Device timing-dependent coding, microprogrammed operations.

Data Management Operations

- Very Low: Simple arrays in main memory.
- Low: Single file sub-setting with no data structure changes, no edits, no intermediate files.
- Nominal: Multi-file input and single file output. Simple structural changes, simple edits.
- High: Special purpose subroutines activated by data steam contents. Complex data restructuring at the record level.
- Very High: A generalized, parameter-driven file structuring routine. File building, command processing, search optimization.
- Extra High: Highly coupled, dynamic relational structures. Natural language data management.

Run-time Performance Constraints

This reflects the degree of execution time constraint imposed upon a software product. The rating is expressed in terms of available execution time expected to be used.

- Very Low: No rating - defaults to Nominal.
- Low: No rating - defaults to Nominal.
- Nominal: $\leq$ 50% use of available execution time
- High: 70% use of available execution time
- Very High: 85% use of available execution time
- Extra High: 95% use of available execution time

Memory Constraints

This reflects the percentage of memory expected to be used by the software product and any subsystems consuming the memory resources. Memory refers to direct random access storage such as RAM, disks or optical drives.

- Very Low: No rating - defaults to Nominal.
- Low: No rating - defaults to Nominal.
- Nominal: $\leq$ 50% use of available storage
- High: 70% use of available storage
- Very High: 85% use of available storage
- Extra High: 95% use of available storage

Virtual Machine Volatility

This reflects the level of volatility of the virtual machine underlying the software product to be developed. The virtual machine is defined as the complex of hardware and software the product will call upon to accomplish its tasks. The ratings are defined in terms of the relative frequency of major and minor changes.
Major change: significantly affects roughly 10% of routines under development.
Minor change: significantly affects roughly 1% of routines under development.

- Very Low: No rating - defaults to Low.
- Low: Major change every 12 months
- Nominal: Major change every 6 months; Minor: 2 weeks
- High: Major: 2 months; Minor: 1 week.
- Very High: Major: 2 weeks; Minor: 2 days.
- Extra High: No rating - defaults to Very High.

Computer Turnaround Time

This reflects the level of computer response time experienced by the project team developing the software product. The response time is the average time from when the developer submits a job to be run until the results are back in the developer's hands.

- Very Low: No rating - defaults to Low.
- Low: Interactive.
- Nominal: Avg turnaround time $<$ 4 hours.
- High: 4 - 12 hours.
- Very High: $>$ 12 hours.
- Extra High: No rating - defaults to Very High.

Analyst Capability

Analysts participate in the development and validation of requirements and preliminary design specifications. They consult on detailed design and code activities. They are heavily involved in integration and test. The ratings for analyst capability are expressed in terms of percentiles with respect to the overall population of software analysts. The major attributes to be considered are ability, efficiency, thoroughness, and the ability to communicate and cooperate. This evaluation should not include experience (that is captured in other factors) and should be based on the capability of the analysts as a team rather than individuals.

- Very Low: 15th percentile
- Low: 35th percentile
- Nominal: 55th percentile
- High: 75th percentile
- Very High: 90th percentile
- Extra High: No rating - defaults to Very High.

Applications Experience

This represents the level of equivalent applications experience of the project team developing the software product.

- Very Low: $\leq$ 4 month experience.
- Low: 1 year of experience.
- Nominal: 3 years of experience.
- High: 6 years of experience.
- Very High: 12 years of experience
- Extra High: No rating - defaults to Very High.

Software Engineer Capability

This represents the capability of the programmers who will be working on the software product. The ratings are expressed in terms of percentiles with respect to the overall population of programmers. The major factors which should be considered in the rating are ability, efficiency, thoroughness, and the ability to communicate and cooperate. The evaluation should not consider the level of experience of the programmers (it is covered by other factors) and it should be based on the capability of the programmers as a team rather than as individuals.

- Very Low: 15th percentile
- Low: 35th percentile
- Nominal: 55th percentile
- High: 75th percentile
- Very High: 90th percentile
- Extra High: No rating - defaults to Very High.

Virtual Machine Experience

This represents the experience of the project team with the complex of hardware and software that the software product calls upon to accomplish its tasks, e.g., computer, operating system, and/or database management system (the programming language is not considered as part of the virtual machine).

- Very Low: $\leq$ 1 month experience.
- Low: 4 months of experience.
- Nominal: 1 year of experience.
- High: 3 years of experience.
- Very High: No rating - defaults to High.
- Extra High: No rating - defaults to High.

Programming Language Experience

This represents the level of programming language experience of the project team developing the software project. The ratings are defined in terms of the project team's equivalent duration of experience with the programming language to be used.

- Very Low: $\leq$ 1 month experience.
- Low: 4 months of experience.
- Nominal: 1 year of experience.
- High: 3 years of experience.
- Very High: No rating - defaults to High.
- Extra High: No rating - defaults to High.

Use of Modern Programming Practices

This represents the degree to which modern programming practices are used in developing the software. Specific practices included here are:

(i) Top-down requirements analysis and design

(ii) Structured design notation

(iii) Top-down incremental development

(iv) Design and code walkthroughs or inspections

(v) Structured code

The ratings are as follows:

- Very Low: No use.
- Low: Beginning use.
- Nominal: Some use.
- High: General use.
- Very High: Routine use.
- Extra High: No rating - defaults to Very High.

Use of Software Tools

This represents the degree to which software tools are used in developing the software product.

- Very Low: Basic tools, e.g., assembler, linker, monitor, debug aids.
- Low: Beginning use of more productive tools, e.g., High-Order Language compiler, macro assembler, source editor, basic library aids, database aids.
- Nominal: Some use [of] tools such as real-time operating systems, database management system, interactive debuggers, interactive source editor.
- High: General use of tools such as virtual operating systems, database design aids, program design language, performance measurement and analysis aids, and program flow and test analyzer.
- Very High: General [use] of advanced tools such as full programming support library with configuration management aids, integrated documentation system, project control system, extended design tools, automated verification system.
- Extra High: No rating - defaults to Very High.

Required Schedule

This represents the level of constraint imposed on the project team developing a software product. Ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort.

- Very Low: 75% of nominal.
- Low: 85% of nominal.
- Nominal: 100%
- High: 130% of nominal.
- Very High: 160% of nominal.
- Extra High: No rating - defaults to Very High.

Sources

- Boehm, Barry W., *Software Engineering Economics*, Prentice Hall, 1981.
- Clark, Brad, *COCOMO® 81 Intermediate Model Implementation*, Online resource at the University of South California.