Prof. Dr. A. Podelski, Dr. B. Westphal
S. Feo Arenis

Sommersemester 2016

Softwaretechnik/Software Engineering

http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl

Exercise Sheet 1

Early submission: Wednesday, 2016-04-27, 12:00 Regular submission: Thursday, 2016-04-28, 12:00

Exercise 1 – Metrics (10)

1.1. Lines of Code Metrics

Consider the following lines of code metrics:

- LOC_{tot} = Total number of lines of code
- LOC_{ne} = Number of non-empty lines of code
- LOC_{pars} = Number of lines of code that that do not consist entirely of comments or non-printable characters.

(i) Calculate the value of the LOC metrics for the Java program in the file `MyQuickSort.java` accompanying this exercise sheet. (2)

(ii) The LOC metrics are often used as derived measure for the complexity or effort required to develop the code being measured.

In particular the family of LOC metrics is notorious for being subvertible. If a metric is subvertible, its value can be manipulated to increase it arbitrarily while preserving the same program semantics. I.e., for every program, there always exists a semantically equivalent program (that performs the same computation, and thus should have needed roughly similar effort to develop) but has substantially different metric values. Convince yourself of this claim for the case of LOC_{pars}: give two semantically equivalent programs with substantially (at least an order of magnitude) different metric values. Can you recognize a pattern that allows you to increase the value of the LOC_{pars} metric to any number you wish? (2)

(iii) What is the value of the LOC metrics for the largest method you have programmed? Describe briefly what the method does, the program to which it belongs and in what programming language. (1)

1.2. Cyclomatic Complexity

Consider the *cyclomatic complexity* or *McCabe* metric.

(i) Construct the control flow graph (CFG) for the method `quickSort` of the class `MyQuickSort` in the file accompanying this sheet and calculate the value of its cyclomatic complexity as shown in the example of Fig. 1. (4)

(ii) In the example, we introduced additional, auxiliary nodes to the control flow graph that serve as junction points for control paths, see e.g., the round node directly after node number 5. Another possibility of constructing the CFG would be to directly connect the nodes representing program locations. In our example, there would be a direct edge from node 5 to node 6 and from node 8 to node 6.

Does this choice of CFG construction alter the value of the cyclomatic complexity metric? Justify your answer. (1)

For program:

```
1  void insertionSort(int[] array) {
2    for (int i = 2; i < array.length; i++) {
3      tmp = array[i];
4      array[0] = tmp;
5      int j = i;
6      while (j > 0 && tmp < array[j-1]) {
7        array[j] = array[j-1];
8        j--;
9      }
10     array[j] = tmp;
11   }
12 }
```

Cyclomatic complexity is defined for graph G corresponding to the program as

$$v(G) = e - n + p$$

Number of edges: $e = 11$
Number of nodes: $n = 6 + 2 + 2 = 10$ (nodes are marked with the corresponding line numbers)
External connections: $p = 2$

$v(G) = 11 - 10 + 2 = 3$

Corresponding graph G

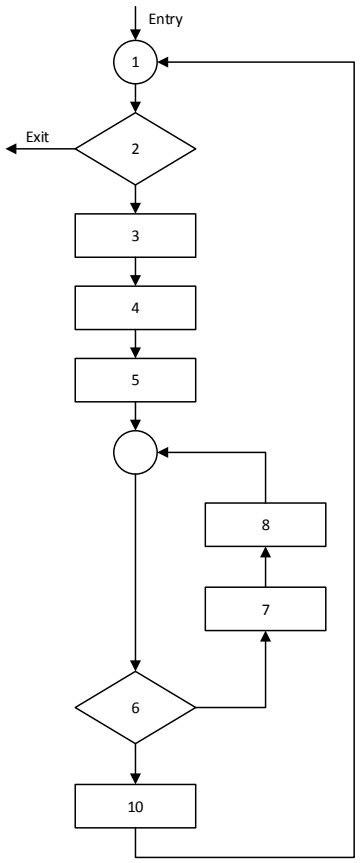

Figure 1: Example of the calculation of cyclomatic complexity

Exercise 2 – Cost Estimation (10)

For the following tasks, assume you are a team of 5-6 bachelor students with typical 3rd-semester programming knowledge for the development of a game. A rundown of the requirements is given in Appendix A.

(i) Calculate the effort of the *Softwarepraktikum* based on the fact that the project awards 6 ECTS to each member of a team of 5-6 people. One ECTS point is equivalent to 30 hours of work.

- What is the resulting expected volume in person-months (PM)?

Give a range for your calculation and describe how you compute your result. (1)

(ii) Estimate a range in $KLOC_{pars}$ for the lines of *effective* code (i.e., not including comments or documentation) that will be delivered given the requirements. Use the Delphi method to perform your estimation and describe the process. In particular, write down what your initial estimations were and how they changed after each round. (2)

(iii) Categorize the software project according to the COCOMO 81 project types as indicated by Table 1. Briefly explain how you chose the project type. (1)

(iv) Determine the ratings for the attributes of the project according to the cost drivers of COCOMO, intermediate level, as shown in Table 2. For each cost driver, briefly explain why the corresponding classification was chosen as shown in Figure 2. A short explanation of the cost drivers can be found in Appendix B. (4)

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 estimation model.

Memory Constraints: Nominal Assets are relatively small and few, and modern execution platforms provide large memory capacities. The game is not expected to occupy more than half the available memory, which assigns the rating Nominal.

Computer Turnaround Time: Low The compilation time of the game is mostly negligible, other tasks, such as code analysis run asynchronously and do not affect the programmer's progress.

Figure 2: Example ratings with justification for the *Softwarepraktikum* project.

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	0.82	
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	0.82	
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.

(v) Calculate the estimated effort E in person-months by using the COCOMO formula

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

where EAF is the *effort adjustment factor*, calculated as the multiplication of the factors for all cost drivers according to your categorization and a and b are the coefficients specified by the project type in Table 1.

How big is the difference between the planned effort according to ECTS and the result of the COCOMO calculation? What are possible reasons for the deviation? Which estimate is more plausible to you and why? (2)

References

Ludewig, J. and Lichter, H. (2013). *Software Engineering.* dpunkt.verlag, 3. edition.

A Requirements for the *Softwarepraktikum* Project

Your task is to develop a computer game that satisfies the requirements described hereon. *A german version is available at:* `https://sopra.informatik.uni-freiburg.de/`

Functional Requirements

- The game must have either 2D or 3D graphics (no ASCII).
- There must be at least two game figures, one of which should be human-like.
- Game figures should be controlled indirectly.
- It must be possible to save and load the game at any point in time. The save/load function must not necessarily be controllable by the player.
- The game must have a pause function.
- The game must contain at least the following types of objects:
  - at least 5 controllable,
  - at least 5 selectable,
  - at least 5 non-controllable, 3 of which should allow collisions and
  - at least 3 controllable, allowing collisions and movable (from here on: *active game objects*)
- The game must be able to handle at least 1000 active game objects simultaneously. The completed game must not necessarily have that many objects, at least a tech-demo must be available that can demonstrate that functionality upon request.
- The game must comprise at least the following types of actions:
  - At least 10 different actions (such as walk, capabilities, 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 frameworks MonoGame version 3.5 (30.3.2016) or XNA Version 4.0 (XNA Game Studio 4.0 (Windows Phone SDK 7.1): September 28, 2011) or newer versions.
- The game must run on Windows 7 x86/x86_64.
- You should use as integrated development environment Visual Studio 2015.
- The game code must not generate any warnings or errors (either from the compiler or from ReSharper). The condition must be checked weekly.
- Every version committed in the version control system must compile and generate an executable.

B Cost Drivers of the COCOMO 81 Intermediate Model

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: $100 \leq D/P \leq 1000$
- Very High: $D/P > 1000$
- 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 x (D - E).
- Low: Evaluation of moderate level expressions, e.g. D = sqrt(Bˆ2 - 4.0 x A x 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: ≤ 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 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 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 user 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.