# Assessment Criteria for BSc Theses at the Faculty of Informatics, ELTE

|  |  |  |
| --- | --- | --- |
|  | **Criteria** | **Scores** |
| 1. | The difficulty of the programming task solved | 4 |
| 2. | The structure, the language and style, and the appearance of the thesis | 4 |
| 3. | User documentation | 4 |
| 4. | Software design document (Developer documentation) | 4 |
| 5. | Implementation (Developer documentation) | 4 |
| 6 | Testing (Developer documentation) | 4 |
| 7. | Program execution | 6 |
|  | Total score: | 30 |

If the score to any of the above criteria is 0, the total score will be 0 as well.

# Guidelines

The description below should be applied in a flexible manner, as it is not always possible to account for all the factors listed under each criterion in a thesis. However, it is important to explain why the maximum score will not be given. Although there is no consensus on the relationship between scores and grades, the table below provides some guidelines:

0 -14: unsatisfactory (fail)

15-18: satisfactory (pass)

19-22: average

23-26: good

27-30: excellent

# 1. The difficulty of the programming task solved

If the task barely meets the level of difficulty required in a thesis, it is worth 1 point. Excess scores are given, for instance, if:

* the implementation of the task requires knowledge beyond the compulsory curriculum;
* the solution contains algorithms the implementation of which is not self-evident;
* the implementation required the use of various platforms;
* a large database had to be created with stored procedures and with elements managed by server agents; and a connection had to be established with various other servers (e.g. Notification Server);
* the software is part of a larger system the environment of which also needed studying before starting to develop the program itself.

# 2. The structure, the language and style, and the appearance of the thesis

Students must comply with the requirements set in the general guide. In addition, the following questions are also taken into account for the evaluation:

* Does the thesis contain all the compulsory parts (the official thesis registration form, the user and the developer documentations, the bibliography and the table of contents)?
* Is the structure and sectioning of the thesis appropriate, logical and comprehensible? How are the parts of the thesis connected? Are they in harmony with each other?
* Is the text of the thesis clear? Did the student edit the text appropriately (numbering, heading, table of contents, styles, running heads, numbering and labelling of figures etc.)?
* What is the thesis like in terms of aesthetics, grammar and style?
* Are the amounts of illustrations and texts in balance? Are the figures legible and appropriate?
* Are the materials taken from other sources adequately and properly quoted?

# 3. User documentation

Prepared for a specific target audience, it includes installation (or operation) and end-user documentation, and is designed to quickly and easily assist end-users to use the program.

It contains:

* a short description of the task (What is the software for?);
* the target audience (Who and when can use the program? And what for?);
* the minimum and the optimal HW/SW requirements;
* How to install the program, and if appropriate; how to start the program (unless it is a new component of an existing system and not an independent application). Here, the evaluators check whether the installation guide corresponds to the actual installation process.
* general user information (e.g. unusual screen, key or mouse operation; explanation of error messages, etc.);
* A description of system functions. Due to the nature of the task, students should present them as a process, supported by screenshots. They should group functions according to user levels. Here evaluators should consider if the description fits the subtask defined in the developer documentation and describes the functions/cases defined there.
* Run-time system messages (error messages, information messages, warning messages, etc.) and their explanation – along with any operational actions that may be required. Evaluators should check if there are any security or troubleshooting instructions given.
* Any other information needed to use the software.

# 4. Solution plan

It is a plan of the software system, part of the developer documentation that describes the purpose and the structure of the application as well as how it works. After carefully studying it, one can basically write the source code of the application.

It includes the following items:

* A description of the system's architecture (including those of subsystems and layers, the specification of standards, technologies, as well as the development methods used, and a definition of the tools and components used). When evaluating, consider how well the layers of the application are separated (e.g. user interface, logic, and data source).
* A description of the database, if there is one. It is worth including an overview diagram to illustrate the tables and their relationships, and a description of the field structure of each table – and any stored procedures, functions, triggers, etc. – in separate tables.
* A description of the module and/or class structure (major modules and/or classes and their methods, and their relationships). The main functions of each package and the main methods of important classes are to be described by the input data, output data, and actions.
* The user interface design (screen and list designs, as well as the menu design), if there are any. There should be an overview diagram showing navigation options and directions between screens (windows, web pages). The most important event handlers should be highlighted.

# 5. Implementation

The implementation part of the developer documentation shows what decisions were made during the implementation process (data representation, the components used, the language elements used in the code, etc.). The documentation should not contain the source program (only the details considered important); it is sufficient to record it on the data carrier enclosed. In addition, the implementation should include a components' plan (the interrelationships of the physical components of the application) and an installation guide. When evaluating, consider:

* whether the content and the structure of the source code correspond to the plan;
* how well the student knows the development tool used (e.g. Do modern, effective language elements predominate? Or do language elements resulting in an overcomplex, cumbersome, inapt and wordy source code prevail instead?) Is the use of the chosen language elements justified?
* what the appearance of the source code is like; how clear it is (regarding its structure, paragraphs, segmentation, comments, etc.);
* to what extent the code can be modified; and whether the student used the language tools of code reuse (functions, inheritance and generic elements);
* whether the student strove for efficient data representation;
* to what extent the code is capable of self-documentation, i.e. how expressive or conventional the chosen identifiers (e.g. variable names) are; and whether the comments help comprehending the code;
* whether the necessary control and error handling functions are included; and whether exception handling is solved in general;
* how well the solution manages human and machine resources, such as user time and patience, and disk capacity or memory capacity.

# 6. Testing

Being part of the developer documentation, it should present the testing aspects and summarize the experience gained during testing. In addition, it also contains an analysis of the scalability of the software.

When evaluating, consider whether the documentation:

* contains test plans and test cases (They can be grouped as system tests and module tests, or as black box testing and white box testing);
* reports on lessons learnt by the student during testing, which led to changes in previous implementation decisions, or, perhaps, in some elements of the plan (N.B. such experience does not detract from the value of the thesis);
* includes an evaluation of the results of high-volume runs;
* analyses the correctness of the result given by the program (especially in optimization problems with several correct solutions, which can be ranked by an objective function);
* analyses the efficiency of program execution.

# 7. Program execution

#### Accuracy:

* Does the program work in accordance with the task definition and the solution plan?

#### Robustness:

How well-protected is the program (in terms of reliability/fault tolerance) against user errors?

* Does the program provide effective working conditions for the user even if they work with realistic (large volumes of) data?
* If the system crashes when being ethically tested, the program cannot be accepted.

#### User-friendliness (standard interface, comfort):

* Is the program easy to use? Clear? Flexible? Aesthetic?
* Does the user interface support solving tasks of a given professional field?
* Do the legends/inscriptions/subtitles/tooltips meet the conventions? Is the wording clear? Is the spelling good?
* Is there help (situation-sensitive help)?
* Can operations be interrupted when necessary, and are they protected against interruption?
* Does the student know or take into account user traditions?