

Safety Net Hospital

Métodos Formais em Engenharia de

Software

- 4MIEIC04 –

Luis Pedro Silva Couto – [mieic1204994@fe.up.pt](mailto:mieic1204994@fe.up.pt)

Porto, 2017

Indice

[1. Informal system description and list of requirements 3](#_Toc500724217)

[1.1. Informal system description 3](#_Toc500724218)

[1.2. List of requirements 3](#_Toc500724219)

[2. Visual UML model 4](#_Toc500724220)

[3. Formal VDM++ model 5](#_Toc500724221)

[4. Model validation 6](#_Toc500724222)

[5. Model verification 7](#_Toc500724223)

[6. Code generation 8](#_Toc500724224)

[7. Conclusions 9](#_Toc500724225)

[8. References 10](#_Toc500724226)

# Informal system description and list of requirements

## Informal system description

In this project we had to model all the information needed to manage a Safety Net Hospital.

Safety net hospitals provide care for populations which are of low-income, traditionally exploited, receiving public assistance, and uninsured. Safety net hospitals are not defined by their ownership terms; rather they are more devoted to providing the best possible care for those who are barred from health care due to various circumstances.

## List of requirements

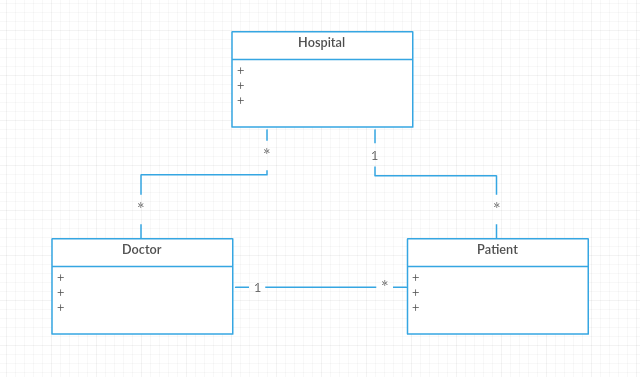
|  |  |  |
| --- | --- | --- |
| **Identifier** | **Priority** | **Description** |
| R1 | Mandatory | It should be possible to list all hospitals in the network |
| R2 | Mandatory | It should be possible to list all doctors working in the network |
| R3 | Mandatory | It should be possible to list all patients attending the network |
| R4 | Mandatory | It should be possible to know the hospital with higher number of patients |
| R5 | Mandatory |  |

a. Requirements should include any relevant constraints (regarding safety, etc.).

b. Each requirement should have an identifier.

c. You may have optional requirements.

# Visual UML model



1. A use case model, describing the system actors and use cases, with a short description of each major use case.
2. One or more class diagram(s), describing the structure of the VDM++ model, with a short description of each class, plus any other relevant explanations.

# Formal VDM++ model

1. VDM++ classes, properly commented.
2. Needed data types (e.g., String, Date, etc.) should be modeled with types, values and functions.
3. Domain entities should be modeled with classes, instance variables and operations. You are expected to make adequate usage of the VDM++ types (sets, sequences, maps, etc.) and create a model at a high level of abstraction.
4. The model should contain adequate contracts, i.e., invariants, preconditions, and postconditions. Postconditions need only be defined in cases where they are significantly different from the operation or function body (e.g., the postcondition of a sqrt(x) operation, which simply states that x = RESULT \* RESULT, should be significantly different than the body); for learning purposes, you should define postconditions for at least two operations.
5. During the development of the project, if you foresee that the size of the VDM++ model will be less than 5 pages (or 7.5 pages in case of groups of 3 students) or more than 10 pages (or 15 pages in case of groups of 3 students), you should contact your teacher to possibly adjust the scope of the system or the modeling approach being followed.

# Model validation

a. VDM++ test classes, containing adequate and thorough test cases defined by means of operations or traces.

b. Evidences of test results (passed/failed) and test coverage. It is sufficient to present the system classes mentioned in 4 painted with coverage information. Ideally, 100% coverage should be achieved. Optionally, you may include figures of examples exercised in the test cases.

c. You should include requirements traceability relationship between test cases and requirements. Ideally, 100% requirements coverage should be achieved. It is sufficient to indicate in comments the requirements that are exercised by each test.

# Model verification

1. An example of domain verification, i.e., a proof sketch that a precondition of an operator, function or operation is not violated. You should present the proof obligation generated by the tool and your proof sketch.
2. An example of invariant verification, i.e., a proof sketch that the body of an operation preserves invariants. You should present the proof obligation generated by the tool and your proof sketch.

# Code generation

1. You should try to generate Java code from the VDM++ model and try to execute or test the generated code. Here you should describe the steps followed and results achieved.

# Conclusions

a. Results achieved.

b. Things that could be improved.

c. Division of effort and contributions between team members

# References

* <https://en.wikipedia.org/wiki/Safety_net_hospital>