PROGRAM SPECIFICATION

INTRODUCTION

• We start from:
  • requirements specifications (made by the client)
  • project plan
  • financial feasibility study

• The comprehension of the requirements is essential

• Basically it is an analysis task comprising:
  • discovery
  • modelling
  • refinement
  • specification

• High probability of communication problems:
  • misunderstanding
  • ambiguity
  • disinformation
PROGRAM SPECIFICATION
INTRODUCTION

• The specification must define:
  • domain model
  • functionality
  • performance
  • interfaces with other systems
  • design constraints

• The tasks are:
  • identifying the problem
  • evaluating and synthesizing the solution of “what” and not “how”
  • modelling
  • specification
  • revision
  • user manual (¡!)
  • revision of the project plan
PROGRAM SPECIFICATION

WAYS TO SPECIFY

• Class diagrams
• Use-case diagrams
• Hierarchical diagrams
• Data flow diagrams (DFD)
• State diagrams, sequence diagrams, collaboration diagrams, ...
• Text
  • 70% of the errors
  • Well written (comprehensible)
  • Meyer defined the 7 “sins” of the specifier
    • Noise – redundant or not necessary information
    • Silence – lack of information
    • Over-specification – detail more than necessary
    • Contradiction – between parts of the specification
    • Ambiguity – due to lack of precision, for example because of the use of synonyms and homonyms
    • Forward references – using terms not previously defined
    • Remorse – introducing constraints on previous statements
• Glossary

Cyclical process among the documents
All the elements of a specification must be consistent with each other
PROGRAM SPECIFICATION
DETAILED DEFINITION

• Contents:
  • What is expected from the system
  • What functionalities it will contain
  • What environment will it work in
  • What constraints must be met

• 2 kinds of functionalities:
  • User’s
    • Explicit
    • Implicit
  • System’s
    • Non-functional requirements

• 2 types of requirements:
  • Compulsory
  • Optional
PROGRAM SPECIFICATION
INFORMATION DOMAIN

• Any computer application is about data processing
• Processing information comprises 3 aspects:
  • Contents
  • Structure
  • Flux
• The quality and precision of the model allow for:
  • Better extraction of the functionality required
  • Making this functionality more ductile to changes
• Good abstractions are reusable

Contents

• It’s the individual data and their grouping into real entities from our domain
  • Ex: A bibliographical cataloguing system is composed by entries with author, title, etc.
• Each entity will correspond to a class
• Each datum will correspond to an attribute of a class
• They are described by means of a template for each class and attribute
INFORMATION DOMAIN: CONTENTS

- Classes extraction:
  - First approximation: model the “vocabulary of the system”: extract nouns (classed) and adjectives (attributes)
  - Refinement: abstractions with their own entity
  - For each abstraction, identify a set of responsibilities attribute properties (and operations) to it
  - Make sure each class is crisply defined and a good balance among responsibilities of the classes is held:
    - Classes too big models hard to change and very little reusable
    - Classes too small end up with more abstractions than you can reasonably manage or understand

```
<table>
<thead>
<tr>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>active?: BOOLEAN = true</td>
</tr>
<tr>
<td>name: STRING</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Set_name (N: STRING): BOOLEAN
Get_name: STRING
... |

Responsibilities
-- keep the information about the student data updated
... |
```
INFORMATION DOMAIN: STRUCTURE (I)

Types of Relationships:

1. Dependency: Use relationship

```
A  ---->
   
B
```

2. Generalization: Inheritance relationship (“is-a”)

```
PERSON
    ▲
STUDENT
```

3. Association or instance: Structural relationship
   3.1. Pure association
```
BOOK  has
  0..10  0..1
READER
```

   3.2. Aggregation: The class “part” makes sense without being attached to a specific class “whole”
```
CENTRE
  ▲
member
  1..*
STUDENT
```

   3.3. Composition: The “part” makes no sense without the specific “whole”
```
CENTRE  ◀  1..*
   
DEPARTMENT
```
INFORMATION DOMAIN: STRUCTURE (II)

Examples:
- Clear aggregation:

```
  +------------------+
  |  PERSON          |
  +------------------+
    |                 |
    v                 
  +------------------+
  |  DATE            |
  +------------------+
```

- Doubt association-aggregation:

```
  INVOICE         INVOICE_LINE
    |              |
    v              v
1..*               1..*  
```

Modelling of structural relationships:
- For each pair of classes, if it is necessary to see objects of one class from the objects of the other data-driven view
- For each pair of classes, if objects of one class need to interact with objects of the other one (not as parameters of an operation) behaviour-driven view
- For each association, if one of the classes is structurally or organizationally a “whole” in comparison with the other class aggregation/composition
- Label the relationship with cardinality, role-names, navigability
PROGRAM SPECIFICATION

NON-FUNCIONAL REQUIREMENTS

• They are requirements not directly related to user functionalities

• They must be measurable:
  • “response time < 3ms” versus “fast response time”
  • “it is learnt in 3 days by an administrative assistant” versus “easy learning”

• They deal with:
  • User interface and human factors
  • Documentation
  • Hardware remarks
  • Performance features
  • Errors and extreme conditions processing
  • System interface
  • Quality factors
  • System modifications
  • Security issues
  • Development environment

[only the ones providing additional information must be included!]