



Universidad  
Carlos III de Madrid

PhD THESIS

# **A Bayesian Model for Change Impact on SW Estimations**

**Author:**

**Jorge Ocón Alonso**

**Advisor:**

**Daniel Borrajo Millán**

**DEPARTAMENTO DE INFORMÁTICA  
ESCUELA POLITÉCNICA SUPERIOR**

**Leganés, Octubre, 2015**

**TESIS DOCTORAL**

**A Bayesian Model for Change Impact on SW Estimations**

**Autor: Jorge Ocón Alonso**

**Director: Daniel Borrajo Millán**

**Firma del Tribunal Calificador:**

Firma

**Presidente:**

**Vocal:**

**Secretario:**

**Calificación:**

**Leganés, de de**

## Table of contents

|            |   |           |
|------------|---|-----------|
| <b>1</b>   | <b>Introduction</b>   | <b>9</b>  |
| <b>2</b>   | <b>Methodology for this thesis</b>  | <b>12</b> |
| <b>3</b>   | <b>State of the art</b>   | <b>14</b> |
| <b>3.1</b> | <b>STATE OF THE ART OF CHANGE IMPACT ANALYSIS</b>   | <b>15</b> |
| 3.1.1      | Key elements and concepts in the impact analysis process                                    | 16        |
| 3.1.2      | Techniques for change impact identification   | 21        |
| 3.1.3      | Conclusions on impact analysis  | 23        |
| <b>3.2</b> | <b>STATE OF THE ART OF SOFTWARE ESTIMATIONS</b>   | <b>24</b> |
| 3.2.1      | Types of techniques used for estimations  | 24        |
| 3.2.2      | Estimations for Change impact   | 29        |
| 3.2.3      | Weaknesses of the traditional model for software estimations                                | 30        |
| 3.2.4      | Handling uncertainty  | 31        |
| <b>3.3</b> | <b>BAYESIAN BELIEF NETWORKS</b>   | <b>33</b> |
| 3.3.1      | Causal models and Bayesian Belief Networks  | 40        |
| 3.3.2      | BBN,s used for change impact estimations  | 45        |
| 3.3.3      | The AREL Model  | 48        |
| <b>3.4</b> | <b>DESIGN RATIONALE</b>   | <b>51</b> |
| 3.4.1      | Design Rationale: Services  | 52        |
| 3.4.2      | Capture of data   | 54        |
| 3.4.3      | Notation  | 55        |
| 3.4.4      | Rationale for Design: Rationale and SEURAT  | 57        |
| <b>3.5</b> | <b>CONCLUSIONS: CURRENT FLAWS OF ESTIMATIONS AND EXISTING PROBLEMS FOR DESIGN RATIONALE</b> | <b>59</b> |
| <b>3.6</b> | <b>AREL AND SEURAT: ANALOGIES AND DIFFERENCES</b>   | <b>63</b> |
| <b>4</b>   | <b>Objectives of the thesis</b>   | <b>65</b> |
| <b>5</b>   | <b>THE APES-CIE Model</b>   | <b>71</b> |

|            |   |           |
|------------|---|-----------|
| <b>5.1</b> | <b>APES-CIE MODEL: OVERVIEW</b>   | <b>72</b> |
| <b>5.2</b> | <b>TRADE-OFFS</b>   | <b>76</b> |
| <b>5.3</b> | <b>DECISIONS</b>  | <b>76</b> |
| <b>5.4</b> | <b>DECISION ELEMENTS</b>  | <b>77</b> |
| 5.4.1      | Requirements  | 78        |
| 5.4.2      | Assumptions   | 79        |
| 5.4.3      | Design elements   | 80        |
| 5.4.4      | Goals   | 81        |
| 5.4.5      | Environmental   | 81        |
| <b>6</b>   | <b>The APES CIE system</b>  | <b>83</b> |
| <b>6.1</b> | <b>GENERAL CHARACTERISTICS OF THE APES-CIE SYSTEM</b>                     | <b>85</b> |
| <b>6.2</b> | <b>ASSUMPTIONS AND DEPENDENCIES</b>                                       | <b>86</b> |
| <b>6.3</b> | <b>OPERATING ENVIRONMENT</b>  | <b>87</b> |
| <b>6.4</b> | <b>EXAMPLE. DEVELOPMENT OF A MEMORY TOOL FOR A MISSION CONTROL CENTRE</b> | <b>91</b> |
| 6.4.1      | Step 1: Adding decision elements  | 92        |
| 6.4.1.1    | Step 1.1 Adding Requirements  | 92        |
| 6.4.1.2    | Step 1.2 Adding Assumptions   | 95        |
| 6.4.1.3    | Step 1.3: Adding Environmental issues                                     | 96        |
| 6.4.1.4    | Step 1.4 Adding Goals   | 97        |
| 6.4.1.5    | Step 1.5 Adding Design Elements   | 97        |
| 6.4.2      | Step 2: Completing the tree: adding trade-offs and decisions              | 98        |
| 6.4.2.1    | Step2.1 Adding trade-offs   | 98        |
| 6.4.2.2    | Step 2.2: Adding decisions, causes and effects                            | 99        |
| 6.4.3      | Step 3: Statistical analysis: using the tree view                         | 105       |
| 6.4.3.1    | Step 3.1 Adjusting CPT's for decision inputs                              | 107       |
| 6.4.3.2    | Step 3.2 Adjusting CPT's for decision's outputs                           | 111       |
| 6.4.3.3    | Step 3.3 Forward propagation: predictive reasoning                        | 114       |
| 6.4.3.4    | Step 3.4 Backward propagation: diagnostic reasoning                       | 117       |
| 6.4.3.5    | Using Threshold warnings  | 119       |

|           |  |            |
|-----------|--|------------|
| 6.5       | APES-CIE FROM THE AEROSPACE'S METHODOLOGICAL PERSPECTIVE | 120        |
| <b>7</b>  | <b>Evaluation</b>  | <b>124</b> |
| 7.1       | USE CASE 1: ON-BOARD SOFTWARE DEVELOPMENT                | 125        |
| 7.2       | USE CASE 2: ROVER FOR OIL & GAS PLATFORM AT SEA          | 129        |
| 7.3       | EVALUATION METHOD  | 133        |
| 7.3.1     | Design Support   | 133        |
| 7.3.2     | Ease of use  | 136        |
| 7.3.3     | Maintenance  | 138        |
| 7.3.4     | Learning support   | 139        |
| 7.3.5     | Documentation  | 140        |
| 7.3.6     | Sensitivity analysis                                     | 141        |
| 7.4       | REVIEW OF THESIS'S OBJECTIVES                            | 142        |
| <b>8</b>  | <b>Conclusions</b>                                       | <b>145</b> |
| <b>9</b>  | <b>Future work</b>                                       | <b>150</b> |
| <b>10</b> | <b>Bibliography</b>                                      | <b>152</b> |

## **List of figures**

|  |     |
|--|-----|
| Figure 2-1: Methodology for this thesis  | 12  |
| Figure 3-1: Steps in change impact and associated sets of SLOs.                      | 18  |
| Figure 3-2 Change Impact analysis and its associated SLOs                            | 20  |
| Figure 3-3: Serial connection: A->B->C   | 36  |
| Figure 3-4: Diverging connection: A->B, A->C, ... A->E                               | 36  |
| Figure 3-5: Converging connection: B->A, C->A, ... E->A                              | 36  |
| Figure 3-6: An example of a small BBN  | 37  |
| Figure 3-7: Simple BBN with four variables, their causal links and CPTs              | 42  |
| Figure 3-8: Example of a BBN as shown using Netica, a BBN tool                       | 43  |
| Figure 3-9: Elements and their relations in the AREL model.                          | 49  |
| Figure 3-10: DRL ontology, as depicted in (Lee, 1989)                                | 56  |
| Figure 3-11: Seurat's model for argumentation's: RATSpeak, as shown in (Burge, 2005) | 59  |
| Figure 4-1: The APES project and its two main outcomes                               | 66  |
| Figure 5-1: The CIE Model  | 74  |
| Figure 6-1: Selecting the Change Impact View in the Show View Dialog in Eclipse.     | 90  |
| Figure 6-2: Selecting a project for the change impact view.                          | 90  |
| Figure 6-3: First steps: adding a requirement to an empty change impact tree         | 93  |
| Figure 6-4: Partial view of the requirements in the Change Impact Tree               | 94  |
| Figure 6-5: Adding a requirement that already exists                                 | 95  |
| Figure 6-6: Adding an assumption   | 95  |
| Figure 6-7: Environmental issues of the example                                      | 96  |
| Figure 6-8: Goals for the example  | 97  |
| Figure 6-9: Example of design elements   | 97  |
| Figure 6-10: Trade-off for our example   | 99  |
| Figure 6-11: Decision inputs and output folders, generated automatically             | 100 |
| Figure 6-12: Initial tree for propagation  | 101 |
| Figure 6-13: Decision Elements and its underlying network                            | 103 |
| Figure 6-14: BBN Model for the “development of a memory tool” example.               | 106 |
| Figure 6-15: Editing a CPT for a decision: develop a new Java application            | 108 |
| Figure 6-16: Change impact tree view showing probabilities for a decision            | 110 |
| Figure 6-17: Editing an output for an output of a decision (outcome).                | 111 |
| Figure 6-18: Editing the outcome based on those decisions that influence it          | 112 |

|   |     |
|---|-----|
| Figure 6-19: Probability propagation once the CPTs have been filled                       | 113 |
| Figure-6-20: Setting a fact for an input once it is known.                                | 114 |
| Figure 6-21: Tree view probability propagation once a fact for an input is set            | 115 |
| Figure 6-22: Setting a fact for a decision.   | 116 |
| Figure 6-23: Reverting a decision fact to “unknown”.                                      | 118 |
| Figure 6-24: Setting a fact for an output goal, analysis of decisions to reach that goal. | 118 |
| Figure 6-25: Setting threshold for High reusability if probability is lower than 60%.     | 119 |
| Figure 6-26: Probabilities affected and Warning for the High reusability.                 | 120 |
| Figure 6-27: Software lifecycle process, as depicted in (ECSS-E-40, 2009)                 | 121 |
| Figure 7-1: Different Trade-offs to be assessed and identified.                           | 127 |
| Figure 7-2: Decisions associated to GNC schedulability.                                   | 129 |
| Figure 7-3: Trade-offs related to the rover for Oil & Gas.                                | 131 |
| Figure 7-4: Decisions for reading gauges.   | 132 |

**List of Tables**

|  |     |
|--|-----|
| Table 7-1: Questions on design and their corresponding average values. ....      | 136 |
| Table 7-2: Questions on ease of use and their corresponding average values ..... | 138 |
| Table 7-3: Questions on maintenance. ....  | 139 |
| Table 7-4: Questions on learning support.....                                    | 140 |
| Table 7-5: Questions on documentation .....                                      | 141 |
| Table 7-6: Questions on sensitivity Analysis.....                                | 142 |
| Table 7-7: Review of thesis’s objectives. ....                                   | 144 |



## Glossary

| Acronym         | Description   |
|-----------------|---|
| <b>AADL</b>     | Architecture Analysis and Design Language                       |
| <b>AE</b>       | Architectural Element   |
| <b>AIS</b>      | Actual Impact Set   |
| <b>AND</b>      | Alpha-numeric display   |
| <b>APES</b>     | Adaptable Project Estimation System                             |
| <b>APES-CIE</b> | Adaptable Project Estimation System – Change Impact Estimations |
| <b>API</b>      | Application's Program Interface                                 |
| <b>AR</b>       | Architectural Rational  |
| <b>AREL</b>     | Architecture Rationale and Element Linkage                      |
| <b>ASW</b>      | Application Software  |
| <b>ATEX</b>     | ATmosphere EXplosibles <sup>1</sup>                             |
| <b>BBN</b>      | Bayesian Belief Networks  |
| <b>CART</b>     | Classification and Regression Trees                             |
| <b>CF</b>       | Certainty Factor  |
| <b>CIA</b>      | Change Impact Analysis  |
| <b>CIE</b>      | Change Impact Estimations                                       |
| <b>CIP-IPF</b>  | Class Interactions Prediction/Impact Prediction Filters         |
| <b>CIS</b>      | Change Impact Set   |
| <b>CM</b>       | Carrier Module  |
| <b>COCOMO</b>   | Constructive Cost Model   |
| <b>CON</b>      | Control   |
| <b>COTS</b>     | Commercial Off-The-Shelf  |
| <b>CPT</b>      | Conditional probability tables                                  |
| <b>DAG</b>      | Directed Acyclic graph  |
| <b>DIS</b>      | Discovered Impact Set   |
| <b>DJF</b>      | Design Justification File                                       |
| <b>DM</b>       | Descent Module  |
| <b>DR</b>       | Design Rationale  |
| <b>DRL</b>      | Design Rationale Language                                       |
| <b>ECSS</b>     | European Cooperation for Space Standardization.                 |

---

<sup>1</sup> This is a the acronym in French

| Acronym       | Description                                       |
|---------------|---|
| <b>EIS</b>    | Estimated Impact Set                              |
| <b>ESA</b>    | European Space Agency                             |
| <b>ESTEC</b>  | European Space Research and Technology Centre     |
| <b>FAT</b>    | Factory Acceptance Test                           |
| <b>FPIS</b>   | False Positive Impact Set                         |
| <b>GNC</b>    | Guidance, Navigation and Control                  |
| <b>GUI</b>    | Graphical User Interface                          |
| <b>HMC</b>    | Hidden Markov Chains                              |
| <b>HW</b>     | Hardware  |
| <b>IMU</b>    | Inertial Measurement Unit                         |
| <b>KLOC</b>   | Miles of Lines of code                            |
| <b>IBIS</b>   | Issue-Based Information Systems                   |
| <b>IDE</b>    | Integrated Desktop Environment                    |
| <b>IEEE</b>   | Institute of Electrical and Electronics Engineers |
| <b>IG</b>     | Influence Graph                                   |
| <b>LOC</b>    | Lines of Code                                     |
| <b>MMI</b>    | Man-Machine Interface                             |
| <b>NASA</b>   | National Aeronautics & Space Administration       |
| <b>OBSW</b>   | On-board Software                                 |
| <b>OSAT</b>   | On-Site Assessment and Training.                  |
| <b>QDC</b>    | Questions, Options and Criteria                   |
| <b>RUP</b>    | Rational Unified Process                          |
| <b>SEURAT</b> | Software Using RAtionale                          |
| <b>SIS</b>    | Starting Impact Set                               |
| <b>SLO</b>    | Software Lifetime Object                          |
| <b>SRD</b>    | Software Requirements Document                    |
| <b>SW</b>     | Software  |
| <b>TCL/TK</b> | Tool Command Language/Tool Kit                    |
| <b>UCM</b>    | Use Case Models                                   |
| <b>UML</b>    | Unified Modelling language                        |
| <b>4GL</b>    | Fourth Generation Languages                       |
| <b>3D</b>     | Three Dimensional                                 |

## **1 INTRODUCTION**

In the last decades, software has become the fuel of modern society, as it is being used in all sectors, from industry to services, from the public to the private, from local areas to worldwide networks (Samad, et al., 2011). This widespread use of the software in almost any human activity is forcing software development to face increasingly complex challenges. The ability to deal with a high level of complexity in a flexible way makes software an essential and increasing part of so many products and services in the market. But this flexibility of the software makes software project management especially difficult. Accurately predicting the costs and resources that will be required for a project from its conception till its final delivery is a very complex task (Kruchten, 2000).

In addition, through the whole lifecycle of the software projects, it becomes necessary to make decisions in order to modify characteristics of the project such as its scope, its design, the team, or the time required. In this scenario, to have precise estimations throughout the whole project lifecycle in order to support decision-making becomes a key element for success (Leung, et al., 2002).

These multiple and recurrent decisions lead to changes in the code and the design, to accommodate to new requirements that are identified as crucial, other nice-to-have features to be implemented, flaws that are identified in late stages of the project, etc... These changes usually generate a cascade of additional, usually unexpected, changes. This is the so-called "ripple effect" (Bilal, 2006).

Considering their impact for the outcome of the project, ideally, estimation models and techniques should allow predicting the consequences of these changes and of this "ripple effect". The possibility of "what-if" analysis is a must for successful project execution (Bohner, et al., 1996).

And these estimations should be based on the new design, since the source code will not be available until the later phases of the project and re-done each time that the need for making new decisions arises.

Current estimation methods are based on models for project execution characterization. External attributes of interest (cost, schedule, effort, budgeting, and quality) are related to "internal" system attributes (structure, behavior, data management) (Angelis, et al., 2001). And, different types of external attributes lead to different types of models for estimation: quality, cost, and risk models. These models are based on quantifiable attributes of the project, and their corresponding relationships and dependencies (Laird, et al., 2006)

The main problem related to current estimation techniques is that estimation methods must be able to deal with uncertainties. For instance, estimations on effort based on lines of code done at the beginning of a project are indeed an estimation of an estimation; based on an estimation of the lines of code required, we estimate the final effort associated to a development. As a matter of fact, we don't know, prior to a given development, what would the final number of requirements be, neither the lines of code, the number of classes of the design, what would the components of the team be, nor the required changes in the design. Therefore, most initial estimations are subject to failure. Exceptions to this rule are the cases in which a new development is very similar to an existing previous one: based on the similarities between both developments, we can assume, with a high degree of accuracy, the values of most of the variables that are used to provide estimations. Nevertheless, in most cases, there are evidences of tremendous deviations between the original estimations and the final, real values.

An additional source of noise in software estimations is created by the decisions that affect changes in the design and the code. There are numerous situations during the lifetime of a project in which decisions have to be made. These decisions affect the design and the final code, and have multiple effects on other elements or pieces of the software, even those that are apparently unrelated. These decisions are not always made after an accurate change impact and estimation analysis, and in most cases the causal relationships among the elements involved as well as the considered rationale and the alternatives are not explicitly detailed (or even analyzed). Projects end up with an associated network of decisions that are mutually inter-related.

This thesis focuses on these problems: first, the need to give accurate estimations to drive the decision process; second, the need to identify and perform a mental mapping of the elements involved in the design decisions; and third, the need to maintain this “network” of decisions in such a way that it is being shared by the stakeholders within a project.

In order to tackle these problems, we will use Bayesian Belief Networks (in short, BBN) for representing the main concepts related to a given project, their causal relationships, as well as the associated conditional probabilities. BBNs are well-defined analysis techniques based on probability calculus that have been used for estimations in multiple areas (Kjærulff, et al., 2005). The main advantage of using BBN for project estimations and measuring change impact is that they allow the estimations to be based on uncertainty and incompleteness of the input parameters. In addition, BBN techniques allow software engineers to use an explicit representation of the causal relationships between the relevant project attributes. BBN estimations also allow us further refinement once these parameters are known.

The estimation and change impact problems are related to three different areas: change impact analysis techniques allow to determine the software artifacts involved in change before the change is made; software estimations and, in particular, the use of Bayesian Belief Networks for software estimations provide the basis for estimating in advance the effort, risk, and/or quality associated; and design rationale gives us the required background for analyzing the mental process associated to software changes. In the following chapters we will provide an overview of the state of the art in these techniques.

## 2 METHODOLOGY FOR THIS THESIS

For this thesis, we wanted to follow a practical approach that allowed us to develop a model for change estimation based on current needs in real developments; a model that can be evaluated from a qualitative perspective by its potential users.

In order to test this model, developing a tool to be used in real projects was required, so that we could get the feedback from those who have used the tool. The different steps followed are shown in Figure 2-1, and this thesis is structured accordingly.

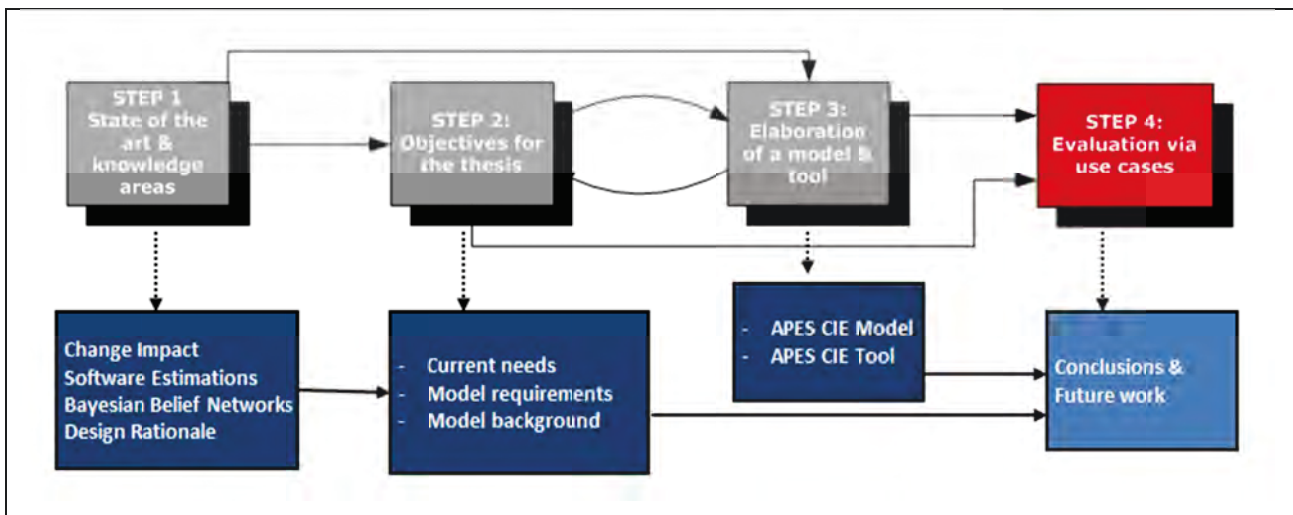


Figure 2-1: Methodology for this thesis

- In a first step we performed an identification of the state of the art and the main knowledge areas involved in change impact estimations. We analyzed four different knowledge areas related to the problems found in change impact estimations. The outcome of this first task is described in Chapter 3 of this thesis, where we provide a description of the state of the art of change impact analysis, software estimations and design rationale. It describes BBN technology and its underlying mathematics. It also discusses those studies that have been more relevant for the outcome of this thesis.
- In a second step, we identified the objectives for the model: its possible inheritance from previous models, the key problems found in change estimations as well as the

main requirements for the model to be developed, considering the current needs. The conclusions obtained in this step are detailed in Chapter 4.

- Based on the requirements for the model and the tool, we developed a model for change impact estimations and built a tool that supported it. Both the model and the tool stem directly from its requirements (obtained at step 2), and the conclusions from Step 1. Chapter 5 provides a description of the model and Chapter 6 provides a description of the tool, based on a simple case.
- A fourth step consisted of the evaluation of the tool in real projects. The tool was applied to two different use cases: a development for embedded critical software and a development of a robot for the gas & oil industry. A qualitative evaluation was performed by the main stakeholders involved in the project. Results of this experience are shown in Chapter 7, meanwhile the conclusions as well as the future work are detailed in Chapters 8 and 9, respectively
- Finally, Chapter 10 of this thesis includes the bibliography

### 3 STATE OF THE ART

When we performed the analysis of the state of the art, we realized that there were four areas of research connected to Change Impact estimations:

- Whenever a change is to be made in the software, a first task to be accomplished is an estimation of the software and documentation artifacts affected by the proposed change. For this, **software change impact analysis** or just “impact analysis” is the discipline oriented to estimate what will be affected in software and the related documentation if a proposed software change is made (Shawn 1996). An analysis of the state of the art of this topic is given in Section 3.1. This section provides the understanding on those techniques that are in use for analysing the impact of a change in an existing design in terms of: artifacts affected and how they will be affected.
- Once these elements have been identified, we are in the position to compute estimations on global software attributes (such as cost, or effort). For this, software metric estimations try to provide global “attributes” of a development before the actual development is made. The focus here is not in the software artifacts affected by the change, but the key, global attributes (such as effort, quality and risk) needed for decision making. As we will see in the following sections, **software estimations** is a very wide area of knowledge in which multiple techniques have been developed, from those based on lines of code to learning-oriented techniques (Boehm, 1981). This topic is discussed in Section 3.2. This section provides us an insight into the different estimation methods oriented to provide in-advance figures for the effort, risk, and quality associated to a given development, as well as to evaluate the impact of a change in terms of effort, risk or quality.
- The technique we use for software estimations is based on **Bayesian Belief Networks**. We will discuss this technique, the reasons behind its use, as well as the use for change impact estimations in Section 3.3. As we outlined in the introduction of this thesis, BBNs are well defined analysis techniques based on probability calculus



that have been used for estimations in multiple areas (Korb, et al., 2004). And they are especially oriented towards estimations when there is a high degree of uncertainty. They can be used to estimate both the probability of artifacts to be effectively part of the change, as well as to estimate global attributes as, for instance, was proposed by Tang et al. (Tang, et al., 2006).

- Finally, these estimations ultimately serve for decision making. In most cases there will be multiple alternatives for the implementation of a change, and one of them will be the one considered best according to the input information. In others, a single change will enforce a cascade of multiple decisions, some of which will be considered valid, some others will be discarded. This is the core object of study for design rationale. **Design Rationale** is a discipline devoted to study the reasons behind decisions made while designing (Burge, 2005). As we will see in Section 3.4, design rationale provides many benefits to an existing development: first of all, it provides an explicit assessment on the alternatives being evaluated and the reasons behind any design decision, leading to a better design support. Moreover, they improve the communication of the team, ease the learning on tackling with design problems, and allow a better maintenance and documentation (Lee, 1997).

### **3.1 STATE OF THE ART OF CHANGE IMPACT ANALYSIS**

Impact analysis predicts the parts of the software system that can be affected by changes in the system. Understanding the nature of changes and measuring their impact (in term of those elements affected) is a process known as Change Impact Analysis (or CIA, for short). The use of CIA techniques has various advantages: impact analysis information can be used to make design decisions during the design process, identifying the expected changes, planning them, and identifying the effects of such changes before they are actually implemented. To summarize, CIA has the following advantages:

- It serves to foresee changes and their impact in new or evolving software resulting in more robust components

- It can significantly reduce the cost of developing new software, since the later changes or unexpected problems are dealt with, the more expensive (in terms of money, time and other resources such as human resources) they become
- As a side effect, it assists project managers in the suitability of proposed changes

Experience in the last decades has shown that software changes are inherent to any software development, and these unexpected changes come from the very beginning of the development till the maintenance of the software. Moreover, as software processes have become more and more reliable, the industry has shifted from generating new software to reusing software as much as possible. Thus, “a major problem is that small changes can ripple through software to cause major unintended impacts elsewhere” (Bilal, 2006).

From the multiple definitions on change impact we have found (Pleeger, 1991), (RADC, 1986) we will use the one from (Shawn, 1996) that defines impact analysis as “identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change”, because it emphasizes the fact that CIA is by itself an estimation, since the actual changes are not known till the change is accomplished.

### **3.1.1 KEY ELEMENTS AND CONCEPTS IN THE IMPACT ANALYSIS PROCESS**

A first concept of paramount importance for CIA is **traceability**. For impact analysis, we take the definition of traceability from (Shawn, 1996) as “**the ability to trace between artifacts generated and modified during the software product lifecycle**”. Much of the literature about software development focuses on traceability of requirements, but we will use this broader definition, that involves any existing artifact involved in the software production. In particular, we are interested in predictive impact analysis (Kama, 2013), that is; estimations on change impact to be performed before changes are implemented.

The motivation behind the impact analysis activity is to identify software artifacts (i.e., requirement, design, class and test artifacts) that are potentially affected by a change. The change can be in the form of addition, removal or modification of new or existing software artifacts. Once we have information on potentially affected software artifacts, effective

planning can be made on what actions will be undertaken with respect to the changes (Kama, 2013).

The so-called “ripple effect” is another important concept, which was defined in (Stevens, et al., 1974) as “the effect caused by making a small change to a system which affects many other parts of a system”. Typical consequences of the ripple effect affect code, data and documentation. However, the ripple effect can affect any artifact involved in the software production: requirements, design elements, environmental elements, and management aspects such as costs, schedule or training.

If we analyze the way software is produced, we find that, after a first stage in which some documentation (requirements, initial design documents) is made, a model of the system is produced. At some point in the development programmers start to develop code.

In an ideal situation, design and requirements will never change, but the real scenario is different: late discoveries in the project force the design to be changed, and this also affects requirements and (in some cases) adding new requirements that were unexpected at first. As time for changes lowers, development of code increases to cover the new functionalities, or, in some cases, to fix some deficiencies of the original design that are discovered when coding. Decisions need to be taken in order to determine the changes required. Since there is not a complete view of the impact of those changes, changes are finally implemented in parallel at several levels (requirements, design, code, tests), making design and documents obsolete and/or incomplete, since they don't reflect accurately the underlying code. Modern development methodologies, like the RUP (Kruchten, 2000), describe in a precise way this problem. They provide a set of guidelines and practices to cope with this challenge. But it requires a huge effort from the development team to synchronize documents, models and code, and it is especially difficult to perform this task during the whole lifecycle of a given project. Moreover, the reasons that lead to changes are usually not documented, and tend to be forgotten, which makes learning from previous decisions difficult.

(Kama, 2013) and (Shawn, 1996) define the impact analysis as a process that generates and modifies Software Lifetime Objects or SLO,s. SLO,s are composed of any possible object involved in the development cycle: requirements, system descriptions, classes or

packages of the design, modules of the source code, test specification, test procedures, test reports, resources being used for the development, etc...

The impact analysis process is a process in which the SLOs affected (to be generated and/or modified) are determined. This process consists of three different steps that are shown in Figure 3-1.

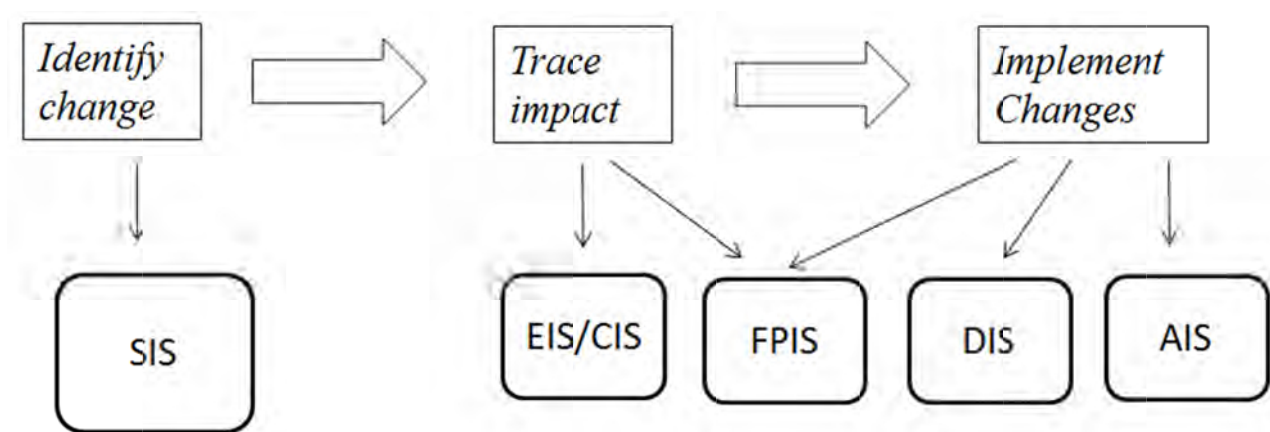
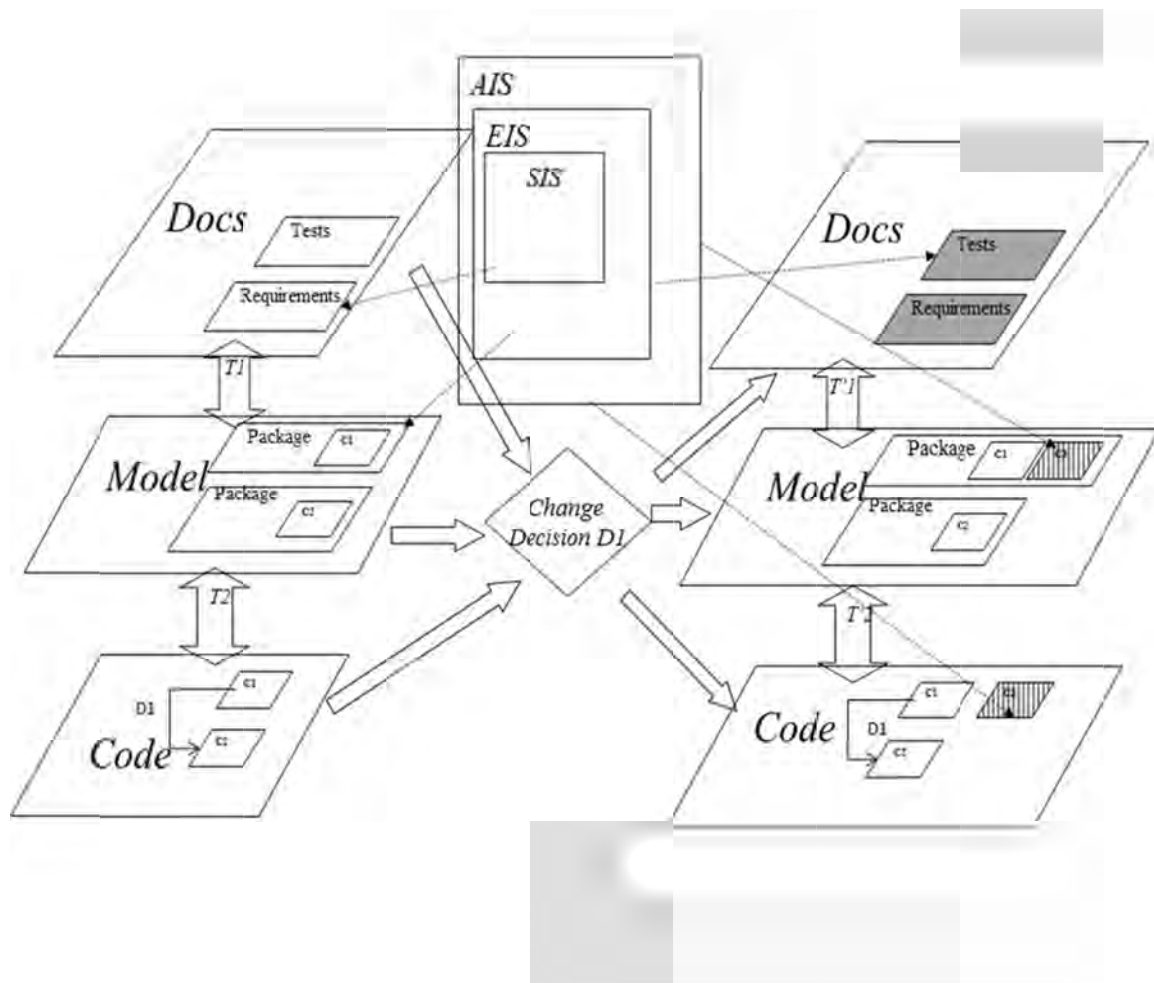


Figure 3-1: Steps in change impact and associated sets of SLOs.

1. The first step for change impact consists of identifying the change specification; the set of impacted artifacts that are thought to be affected. This initial set is the starting impact set (SIS) (Shawn, 1996), (Arnold, et al., 1993).
2. The second step is to trace the potential impact, the "ripple effect", identifying additional elements that are to be affected. The Estimated Impact Set (EIS) or Change Impact Set (CIS) is the set of objects estimated to be affected (Kama, 2013) (Shawn, 1996) after this analysis is performed. False impacted elements, originally found in the SIS can be identified; this is the so-called false Positive Impact Set (FPIS), which is the overestimated impact set in the CIS.
3. A third step involves the implementation of the requested changes. Once this is accomplished, the Actual Impact Set (AIS) is identified, a set that contains of SLOs actually modified as a result of the change. A new set of SLOs to be implemented is also identified, that forms the Discovered Impact Set (DIS).

For the identification of the impacted SLOs, two different approaches can be used: dependency analysis or traceability analysis (Kama, 2013):

- Dependency analysis is based on the analysis of the program, and studies the relations among the different SLOs inside the program (functions, variables, methods, packages, etc.). This is performed exploring the structure of the code. Obviously, this analysis cannot be applied to the initial phases of the project in which there is no generated code.
- Traceability analysis is the analysis of relations between software artifacts among different phases, and therefore it can be applied from the beginning of the design. The identification of the change implications is performed among different workflows of phases of the development (requirements, validation, and code) and therefore the estimation is based on a traceability analysis.



**Figure 3-2 Change Impact analysis and its associated SLOs**

In Figure 3-2, we can see that a proposed change involved initial changes in requirements (SIS). After a change impact analysis, it is determined that this requirement will imply changes in a package of the design, and therefore both requirements and this package form the Estimated impact set (EIS or CIS). To do so, we count on dependency analysis (D1, in the figure) and traceability analysis (T1, T2). During the implementation of the change, a new class has to be generated, which in turn requires modifications to the tests. This forms part of a set of the discovered impact set (DIS); that is, the set of new elements that are to be generated as a result of the change. The final set containing all the SLOs modified or generated becomes the Actual Impact Set (AIS).

Figure 3-2 shows a typical process for a change decision, in which new elements are involved during the process, and therefore each new set contains the elements from the previous one; that is, SIS is contained in EIS, which in turn is contained in AIS. In the figure, the False Positive Impact Set (FPIS), which is the overestimated impact set in the CIS, is empty. But this is not always the case. The combination of artifacts in CIS with artifacts in DIS and the elimination of FPIS artifacts in that combination should be in the AIS results; namely:

$$\text{AIS} = \text{CIS} + \text{DIS} - \text{FPIS} \text{ (Kama, 2013)}$$

This process is performed iteratively, and the accuracy of the CIA can be determined by using metrics to evaluate the closeness between CIS and AIS. Metrics for the accuracy of change impact are based on the final identification of the differences between CIS and AIS (Kama, 2013).

### **3.1.2 TECHNIQUES FOR CHANGE IMPACT IDENTIFICATION**

Two main categories of impact analysis techniques exist: the static analysis technique and the dynamic analysis technique.

- **Static analysis.** The static analysis technique is based on the analysis of the static information from the program obtained from software artifacts (requirements, design, testing, documentation, etc...). (Kama, 2013) Identifies three main static analysis techniques, namely the Use Case Maps technique, the requirements interdependency technique and the class interactions prediction with impact prediction filters.
- **The UCM Model technique** is based on Use Case Models (Hassine, et al., 2005) to define the impact on a high-level definition of the system based on use-cases and requirements. Although it has the advantage that it can provide estimations from the very beginning of the life-cycle (specification), this technique does not use software artifacts related to other workflows (coding, testing), and therefore it cannot be considered complete.

- **The requirements interdependency technique** (Y. Li, et al., 2008) assumes that the requirements and the class artifacts are completely developed prior to implementing the technique. This technique uses traceability links detection between the requirements artifacts and the class artifacts. Impact of changes can be analyzed by navigating to the class artifacts that are traced to a requirement. Once again, this technique cannot be considered complete, since some requirements do not map directly to implemented classes and there is no possibility to include design artifacts in the process.
- **The Class Interactions Prediction with Impact Prediction Filters (CIP-IPF)** technique (Basri, et al., 2015) follows a very similar approach to the requirements interdependency, but with a different requirements interaction traceability technique, that provides the possibility to use design artifacts analysis in the process.
- **Dynamic Analysis.** The dynamic analysis technique takes into account program dynamic information and is based on the execution of code. There are mainly two techniques used on method level analysis: Influence mechanism and Path impact. While the Influence mechanism technique uses an Influence Graph (IG), the Path impact technique considers the whole path DAG and uses class artifacts as the source of analysis, assuming that class artifacts are completely developed. Both techniques are limited in terms of the traceability process (from requirements to code, from requirements to design), having the first also limitations on scalability (i.e. when the number of classes is too high, this technique becomes unusable). The main flaw for dynamic analysis techniques is that they cannot be used in the absence of code, and therefore are useless in the early stages of the development. Since we are interested in methods that can be used from the very beginning of the software conception, in this thesis, we will focus on static analysis techniques, introducing the use of techniques based on probability calculus, in particular based on BBNs, to determine the probable impact of changes on the different elements. As we will see in the next section, these methods are specifically useful in the presence of uncertainty.



### **3.1.3 CONCLUSIONS ON IMPACT ANALYSIS**

The main flaw of the analyzed techniques is that none of the previously analyzed impact analysis techniques support all the important elements for impact analysis implementation throughout the whole lifecycle of a development (specification, design, implementation, validation and maintenance), and therefore are valid exclusively for a given period of the development. Dynamic analysis techniques are not applicable when there is no underlying code, while static analysis techniques focus on high level requirements, use cases and/or design elements that can be traced to classes, which is clearly insufficient for a complete analysis of the impact.

- What we think that would be necessary is a system that can be used from the beginning of the development, providing a model that can be based on all kind of artifacts produced in the different workflows of the development process (note that the meaning of the term workflow that we are using corresponds to the notion of workflow of the Unified Process (Kruchten, 2000)).
- Any system for change analysis shall be able to use both dependencies obtained automatically or based on expert’s judgment.
- This change analysis is in the majority of the cases is performed in the presence of uncertainty, and therefore has a probabilistic nature. The consequences of the changes are not known “a-priori”; for a given change, we estimate – based on existing information at the moment the analysis is made – which artifacts will be affected, what goals will be achieved, what the effort required will be, and how long the implementation of the change will take, based on the existing resources. But the conclusions of the analysis are to be reviewed periodically, and will eventually change, not only in the number of SLOs impacted (i.e. differences between CIS and AIS), but also in terms of the effects achieved.
- Finally, the whole development process can be seen as a chain of change decisions that are committed in order to achieve a particular goal; some of these decisions are dependent on other previous decisions, and some of them performed independently.

Each of these change decisions has a causal dependency between its root causes and its consequences.

We will come back to these ideas later, when we analyze our Bayesian model for change impact.

### **3.2 STATE OF THE ART OF SOFTWARE ESTIMATIONS**

This chapter discusses the types of techniques used for software estimations, the existing problems related to this area, as well as the state of the estimations for change impact.

#### **3.2.1 TYPES OF TECHNIQUES USED FOR ESTIMATIONS**

Since the start of the formal analysis of software development, there have been many approaches to estimate several metrics (such as quality, risk or cost) taking into account data coming from different phases (analysis, design, or coding) of software engineering (Gilb T, 1976) (Boehm, 1981). In fact software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code (these are the classic software “metrics”) to models that help predict software resource requirements and software quality.

Depending on the attribute of the software to be estimated, software estimations can be categorized into:

- **Effort estimations:** methods oriented towards “a priori” determination of cost and schedule parameters for a given project
- **Quality estimations:** intended to predict the number of failures that will be produced in the development. Formally, a failure is an observed deviation of the operational system behavior from specified or expected behavior (Fenton, et al., 2002). Faults can be classified according to their importance or simply the mitigation action, such as, for instance:
  - Faults that had already been corrected;

- Faults that will need to be corrected;
- Faults that require no action (i.e. not treated as a fault); and
- Temporary faults, (e.g. due to installation problems)
- By contrast, **risk estimations** tend to predict the probability that a given parameter of the project (mainly costs or schedule) could be exceeded.

In addition, depending on the scope of the estimation, we find two main categories of estimations:

- **Project estimations:** (that is the traditional approach for estimations) provide estimations of a project as a whole, from its conception till its final development. The estimation is periodically updated, as soon as some uncertainties from the initial phases are being clarified. Project estimations tend to estimate global attributes of the project, considered as a whole.
- **Change Impact Estimations,** in contrast, are oriented towards the use of the estimations in predicting the impact of changes in the software during the whole development process (including maintenance). Given a change to be implemented, the estimation tries to determine the impact of that change on a given attribute (such as effort, quality, and risk). These estimations are traditionally performed during the maintenance phase of the development.

However, both estimations (project estimations and change impact estimations) can be considered complementary; in fact, a project can be considered as the outcome of a sum of changes performed incrementally.

This thesis focuses on this last type of estimations, the so-called “Change Impact Estimations” (Bohner, et al., 1996). Formally, a simplified description of the problem of measurement in Software Engineering consists of: given a project, a set of input attributes, a set of intermediate attributes, a set of output attributes (measurements), a set of relations among input attributes, intermediate attributes and measurements, obtain a model that can predict the values of the output attributes as a function of the values of the other two kinds of attributes. This model will be used in future project estimation processes, by providing the

values of the input (and possibly intermediate) attributes, to obtain the values of the output attributes. So, usual questions to use any model are:

- What are the relevant attributes?
- How is complexity of projects computed and where does it come from?
- How do the values of some attributes affect others?
- What is the impact of changes of values on some attributes?
- How do we represent those attributes?

We will now discuss several approaches that address some of these questions. We have used the word “attributes”, though there have been all kinds of names given to the same concept. Examples are decisions (usually represented by input or intermediate features), variables, features, factors, or parameters. Also, some of these attributes come from project measurements and are usually referred to metrics. In some papers, those metrics are used as the desired output. However, usually they are used to infer the values of the project measurement estimates (most often cost, quality and risk) as intermediate or even input attributes.

Tang et al. (Tang, et al., 2006) divide the software metrics area into two large components: the component concerned with defining the actual measures (in other words, the numerical “metrics”); and the component concerned with how we collect, manage and use the measurements. While software metrics have been dominated by the first component (definition of specific measurements and models) much recent work has focused on the second component (collecting, managing and using metrics in practice). Although there have been improvements, the approaches to quality and resource prediction have remained fundamentally unchanged since the 1980’s. The key driver is some type of measurement of size/complexity and variations determined by regression approaches are explained by other measurement variables.

For the numerical metrics, (Fenton, et al., 2000) provides a classification of the software metrics that is commonly accepted. This classification describes succinctly what traditionally are considered as “internal” and “external” attributes.

The most common and first metric to be used was the Lines of Code (LOC or KLOC for thousands of lines of code) metric. It was, and still is, the basis for the measurement of programming productivity (LOC per programmer/month) (Fenton, et al., 2000).

Further studies proposed regression-based models for module defect density (number of defects per KLOC) in terms of module size measured in KLOC. The need for more discriminating measures was evident during the 70,s with the increasing diversity of different programming languages (Fenton, et al., 2000). Measurements for software complexity and measurements of functional size were developed.

Boehm et al (Boehm, et al., 2000) survey the main approaches for metrics estimations and defined five kinds of techniques:

- **Model-based:** they are parametric techniques, as SLIM, COCOMO (Boehm, et al., 1995), Checkpoint, or SEER (Basha, et al., 2010). They rely on models represented in a variety of formalisms (as functions, distributions, or knowledge bases) that depend on some parameters and are able to produce project estimations.
- **Expertise-based:** they are based on experts' judgments. Examples are the Delphi approach or the hierarchical decomposition of Work Breakdown Structure (Leung, et al., 2002) They have the advantage of incorporating the knowledge of experts, and the disadvantages that they are biased by the experts that defined them (thus, sometimes, they are domain dependent), and also the estimation models usually are hard to obtain.
- **Learning-Oriented Techniques:** the creation of the estimation model is posed as an inductive task, and machine-learning techniques are used to automatically generate the models from data. Examples of employed techniques are analogy (Case-Based Reasoning), or neural networks, though several other techniques could have been used as model-based regression (M5 or CART) (Quinlan, 1992). The advantage of these techniques is that they alleviate the knowledge acquisition task, and the main disadvantage is that

many instances of correct (little noise, no missing data,...) pairs <project, metrics values> are needed while usually very few examples are available.

- **Dynamics-Based Techniques:** they assume that software project metrics change over the system development cycle. Thus, metrics can be defined in terms of formal models such as differential equations (Madachy, 1994). They are good for planning and control, but particularly difficult to calibrate.
- **Regression-Based Techniques:** like OLS (Weisberg, 1985): they have been the most widely used ones and pose the task as the learning-oriented ones (in fact, one could merge them together): starting from data of <project, metric values> generate a regression model (usually as a linear function of the known variables). They obtain good results when there are lots of data, no data is missing, there are no outliers, variables are uncorrelated, and the understandability of variables in the model is important. However, these conditions are seldom met, specially the three first conditions.
- **Composite Techniques:** they combine two or more of the previous techniques. For instance, the Bayesian approach uses a causal model defined by the experts that can be initially injected with estimations on conditional probabilities generated from previous projects data (Fuentetaja, et al., 2013).

Other potential classification criteria could have been the kind of software metric a given approach focuses on. Thus, there are approaches for cost estimation, quality estimation, and risk estimation and so on.

Also, there are approaches that try to estimate several of those metrics in parallel. There are other surveys on cost estimation that present a similar decomposition as (Molokken, et al., 2003). They even compare different cost estimation metrics as in (Gray, et al., 1997), or of only one kind as the analysis of machine learning in (Mair, et al., 2000). Finally, other surveys focus on a specific context as software development within the 4GL framework using space data (from NASA) (Morgan, et al., 2003)

This thesis will focus on this last type of techniques, the so called “composite techniques” in particular those based on Bayesian Belief Networks. In the following sections, we will

provide the rationale for this decision, identifying the weaknesses of the traditional estimations models, their current flaws and the new approach based on BBNs.

### **3.2.2 ESTIMATIONS FOR CHANGE IMPACT**

One of the first works on measuring the ripple effect produced by changes in software is due to Yau and Collofello (Yau, et al., 1980). This measurement results from the study of the control flow of the program and how some changes on one variable affect other parts of the software until no more source is found to be affected. Obviously, this computation is significantly laborious so that different approaches have been proposed in order to make it more explicit and easy to compute. In 2001, Blac suggested to break down the structure of programs into their natural constituents (modules) in order to study more accurately the ripple effect (Blac, 2001) though restricting the attention to procedural programs (i.e., those adhering to the imperative paradigm). So far, the ripple effect is computed as the scope of a change in one variable and how it ripples either within the same module (intra-module change propagation) or among modules (inter-module change propagation). Thus, the computation of the ripple effect provides also an additional measure on the program's complexity, in more than one sense.

Fortunately, some steps have been taken towards generalizing this measure to object-oriented programming (such as C++ and Java). At the same time, it has been also observed that changes (either internal as bugs, replacement of tools, etc.) or external (economic constraints, human resources, etc.) shall be traced back to as early stages in the development process as possible. Hence, it has been recently suggested to use architecture design decisions as a starting point for the change impact analysis (Hassan, et al., 2008). In this case, graphs are used to describe all the dependencies among objects. Since, in the mentioned work, authors proposed to use AADL (Feiler, et al., 2012), being ADDL a model-based language that allows enriching UML with more model-based design elements, these objects are components, connectors and configurations as they are used in the architecture design. From these graphs, a model is proposed to track the change impact by providing a detailed examination of the consequences of changes in the system. This mechanism

consists of defining evolution operations that are used later on to trace the impact of changes resulting from evolutions. The evolution operations are implemented as rules that specify what actions shall be taken on when some conditions are met. Thus, various scenarios can be simulated and the impact of different changes can be followed across networks of any complexity.

### **3.2.3 WEAKNESSES OF THE TRADITIONAL MODEL FOR SOFTWARE ESTIMATIONS**

Although software metrics have been developed for more than 30 years, most companies are not using them as a key driver. Obviously, there should be deficiencies on the existing estimation models that have prevented their acceptance. For instance, some models only estimate a specific metric, as cost, while not supporting the estimation of other metrics, as risk. Also, most techniques require knowing attributes that are unknown until very late in the development, as LOC. As depicted by Fenton et al (Fenton, et al., 2000) what would be required is to have simple estimation methods based on existing metrics to perform management decision-support. They should combine the different aspects of software development and testing and enable managers to make many different kinds of predictions, assessments and tradeoffs during the software lifecycle. These metrics are currently based on elements that are evaluated, because they are not known a priori (Fenton, et al., 1999). Fenton et al (Fenton, et al., 2000) present an extensive critique of this approach, because of the impossibility for existing models to predict defects accurately using size and complexity metrics alone. In particular, the authors mention some problems as:

- Some metrics are based on attributes that describe very poorly the reasons why some metric value is high or low. Examples are size-based attributes or no static complexity ones (as cyclomatic complexity). The knowledge about the causal relationship among those attributes cannot be easily incorporated into those models.
- Some methods (as machine learning or regression based) require a huge amount of precise (little noise) historical data. This data is hard to obtain and sometimes



describes projects that are highly different than the future ones (the inductive assumption fails).

- Knowledge about design decisions or initial constraints (such as available resources) is difficult to incorporate into those models or explicitly using them for metrics estimations. They assume a static environment and perfect knowledge of all relevant attributes of the project, so that there is no uncertainty on the inputs/outputs of the models.

Explicit management of uncertainty is needed for this task, since there are many inputs that are not known a priori, or are only partially known. Fenton et al (Fenton, et al., 2000) identify the need for software estimations to be able to predict under uncertainty, and to be able to be refined during the lifetime of a project, as the degree of certainty increases.

### **3.2.4 HANDLING UNCERTAINTY**

According to (Smith, 1992), any automatic system modeling processes that are probabilistic in nature shall:

- Compute the best course of action
- Provide an explanation why that is the best course of action
- Provide a framework that can be critically appraised and modified

As discussed in (Pearl, 1998), the very first systems with these goals in mind were rule-based systems. This sort of system consists of both a knowledge base and an inference system. While the former consists of a collection of facts stored according to some predefined knowledge representation system (such as propositional logic, first-order predicate logic, frames, etc.), the latter contains a set or dictionary of rules such as “A implies B”.

- At each iteration a rule is picked for execution if and only if the current contents of the knowledge base satisfy the premises of the selected rule. Executing a rule means committing the changes in the knowledge base as specified in the consequent of the

rule. In order to ensure a best course of action, rule-based systems can be iterated either in a forward or backward direction. The explanation results naturally from the concatenation of all the rules selected so far. However, rule-based systems suffer from very serious drawbacks. For instance, they cannot handle exceptions appropriately, while reasoning with exceptions is an important, even crucial, factor in automated decision systems.

- The first idea to tackle with exceptions is uncertainties. One can try to represent exceptions with numbers representing how well a given fact is known. This can be achieved by modifying the dictionary of rules to distinguish among different levels of uncertainty, such as in the rule “A with certainty  $x$  implies B with certainty  $f(x)$ ” where  $f(x)$  is a function to be provided as well. While uncertainties can be used to derive the value of other uncertainties from one assertion to the next, they have some significant drawbacks. Most importantly:
  - They do not stand for probabilities. While this is apparent from the definition, it has some important consequences, including the fact that uncertainty measures cannot be operated (e.g., aggregated) among them in a coherent and precise way
  - It is not possible to compute uncertainties incrementally, i.e., it is not feasible to compute the impact in the uncertainty from one observation and after assimilating this new value, to re-compute it again considering the impact of a new fact (characterized by its own uncertainty). The reason is that, as stated above, there is no way to operate uncertainties jointly in a coherent way, since they are not probabilities

In fact, this distinction serves to classify automatic decision systems in one of the following categories: either extensional system (such as MYCIN (Shortliffe, 1976) or R1 (McDermott, 1984)) or intentional systems. The extensional systems usually mirror relationships in the form of rules such as (A implies B) and maybe decorated with an amount  $x$  which stands for a certainty factor (CF), reflecting both beliefs and disbeliefs in such a way that they are not necessarily related to each other –hence, leading easily to contradictory or counter-intuitive results, as stated before. This sort of assertions shall be read as “if A is found, then B can

be asserted with certainty  $x$ ” regardless the way A was inferred or obtained. This way of reasoning imposes some locality that leads to some counter-intuitive and contradictory results, mainly appearing as problems in bidirectional inferences, retracting conclusions and also improper treatment of correlated evidence.

Consider for instance, the following example proposed in (Jensen, 1996): if C can be derived both from A and B with different levels of certainty, what can be said about the certainty of C? The main problem is that rules are not the right mean for reasoning under uncertainty because rules are context-independent while coherent reasoning is sensitive to the context. In this example, the problem is that no clue is given about the dependency between A and B.

In contrast, intentional systems use the probability theory to take into account all observations that shall be factored out before asserting something new. However, this can be a very laborious task. Belief networks<sup>2</sup> provide an efficient way to encode and propagate the probabilities among different observations or evidences, for example using hierarchical modeling. In fact, a large number of systems have been built under this paradigm including applications in Agriculture, Computer Vision, Computing, Information Processing, Medicine and so many others just to mention a few (Jensen, 1996).

### **3.3 BAYESIAN BELIEF NETWORKS**

As stated in (Smith, 1992), algorithms for both modeling and propagating evidence (or beliefs) shall:

- Identify the objectives of the analysis
- Identify the viable decisions
- Quantify any uncertainty in the problem
- Quantify the costs of taking any viable decision

---

<sup>2</sup> They are also known as Bayesian Belief Networks, Bayesian Networks and other alternatives shortening Networks by Nets.

Probably, one of the easiest paradigms to propagate probabilities instead of certainty factors are Decision Trees, see (Gamerman, 1997). Decision Trees provide a pictorial view of a given problem that can be used to compute the expected pay-off associated with a sequence of actions according to the basic rules of statistics. Hence, the sequence leading to the maximum expected pay-off can be systematically found. However, Decision Trees have some serious drawbacks. In real-world problems it is far from easy to enumerate all the feasible combinations of the variables taking part in the problem at hand. Also, decision trees often compute the final outcome under the assumption that probabilities are independently distributed. This is not usually the case, so that other techniques are often used to compute the final probabilities associated with different courses of action. Therefore, other algorithms are usually better suited. More remarkably:

**Bayesian inference:** from a precisely defined model that establishes the causal relationships among different factors, probabilities can be propagated according to the Bayes' theorem. Models can be defined in various ways so that probabilities can be propagated according to different mathematical apparatus (introduced in the bibliography with the purpose of being both computationally efficient and as accurate as possible): generalized linear models, hierarchical models, dynamic linear models and dynamic generalized linear models.

**Markov chains:** can be used to predict the next state of a stochastic process just by observing the last state. Beyond its scientific interest (since many results can be proven describing the general behavior of many different types of dynamic systems), it is a very powerful technique that can be used in many different contexts. For example, it is possible to implement the so-called high-order Markov chains that result from considering the last  $n$  states instead of the only one. Also, Markov chains can be combined among them and even be used to describe partially observable states (such as the Hidden Markov Chains or HMC for short). Markov chains can be also used for stochastic simulations. Among others, the most common techniques are the Gibbs sampling and the Metropolis-Hasting algorithms (Geman, et al., 1984).

A common technique extending the basic behavior of decision trees are Influence Diagrams which are the basis of Causal Networks. Jensen (Jensen, 1996) describes causal networks as a set of variables and a set of directed links between variables. Variables may have a discrete (countable) or continuous domain. In any case, variables take values from a number of mutually exclusive states. These values support different types of evidence (Korb, et al., 2004):

- Specific evidence, which occurs once a variable is known to take on a specific value
- Negative evidence, when a variable is known not to be in a specific state
- Virtual evidence, which is just any probability distribution (either discrete or continuous).

Evidence may be transmitted among variables in three different ways:

- Serial connection (e.g.,  $A \rightarrow B \rightarrow C$ ),
- Diverging connection (e.g.,  $A \rightarrow B$ ,  $A \rightarrow C$ , ...,  $A \rightarrow E$ ), and
- Converging connection (e.g.,  $B \rightarrow A$ ,  $C \rightarrow A$ , ...  $E \rightarrow A$ ).

Figure 3-3, Figure 3-4 and Figure 3-5 illustrate the different sorts of connections:

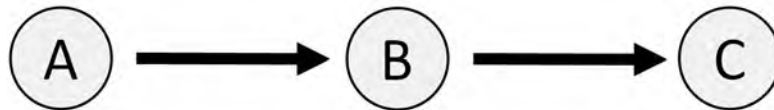


Figure 3-3: Serial connection:  $A \rightarrow B \rightarrow C$

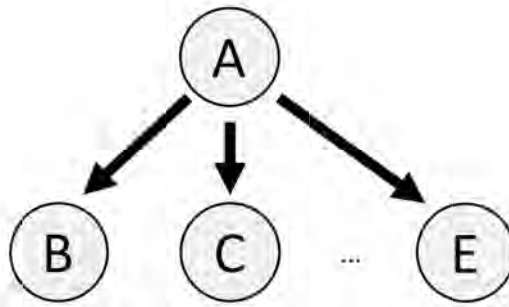


Figure 3-4: Diverging connection:  $A \rightarrow B, A \rightarrow C, \dots A \rightarrow E$

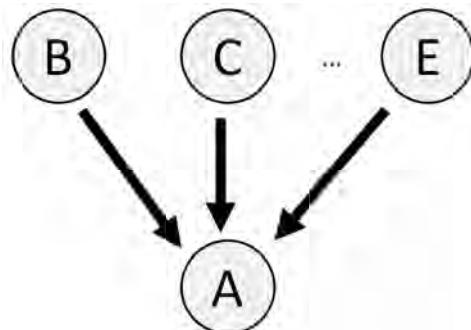


Figure 3-5: Converging connection:  $B \rightarrow A, C \rightarrow A, \dots E \rightarrow A$

From the previous types of propagation, the following definition follows (d-separability): two variables A and B in a causal network are d-separated if for all paths between A and B there is an intermediate variable V such that either:

- The connection is serial or diverging and the state of V is known, or
- The connection is converging and neither V nor any of Vs descendants have received evidence

This definition is of crucial importance since it turns out that when A and B are d-separated, it can be proven that the certainty of A has no impact on the certainty on B. From the definition of causal networks (whose main result consists of identifying d-separability as a property of human reasoning), when using probability calculus, the Bayesian networks result. In this case, relations are expressed as conditional probabilities. Indeed, the statement: "if B is true, the probability of event A is x" shall be read as  $P(A|B)=x$ . Contrary to common intuition, this does not mean that if B is known to be true, A shall happen with probability x –as it was suggested in the extensional systems discussed in the previous section. Indeed, it means that if B is true and everything else is known to be irrelevant for A, then  $P(A)=x$ . This way, the relations among different parameters can be modeled and inference can be applied.

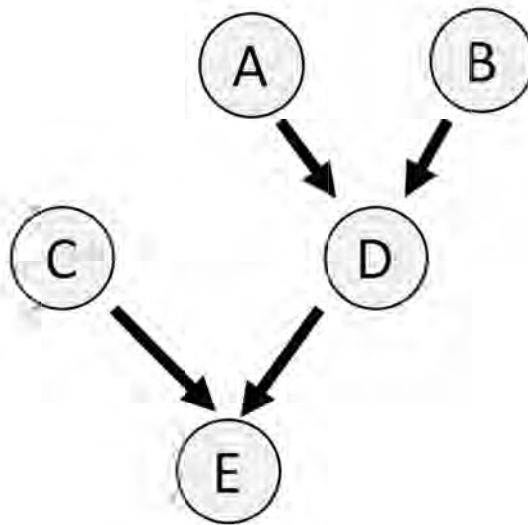


Figure 3-6: An example of a small BBN

For example, the preceding figure shows a small portion of the BBN shown in (Radlinski, et al., 2007) where nodes have the following meaning:

- A stands for the project duration;
- B means average number of people working full time in the project.

These variables serve to compute the total effort adjusted by Brooks factor in node D which, along with node C (that contains some metrics on process and people quality) serve to predict the total effective effort, in node E.

It is important to emphasize that D is said to d-separate A and B from C and E if D receives no evidence when computing the values for C and E, so that it has to be computed from A and B solely. If D would have diverging connection issuing from it, it would be to d-separate its parents (A and B) from its descendants (E and others) if D has received evidence so that its children are computed solely from D.

Therefore, conditional probabilities can be seen as the strength of links relating pairs of nodes. For instance, if A and B are parents of C, the probability  $p(C|A,B)$  shall be quantified, instead of just the probabilities  $p(C|A)$  and  $p(C|B)$ , which do not provide any clue on how to compute the probability of C, since interactions between A and B are not taken into account. Because there is no probabilistic approach for propagating probabilities with cycles, Bayesian networks are required not to contain any cycles.

In other words, they are depicted as Directed Acyclic Graphs (DAG)<sup>3</sup> where, for each variable A with parents B1, B2, Bn, there is a probability  $P(A|B1, B2, \dots, Bn)$  attached. Of course, root nodes (nodes with no parents) are just qualified with their “a priori” probability. For all the other nodes, the Conditional Probability Tables (or just CPTs for short) shall be defined which formalize the conditional probability of every node given its parents. From the given conditional probabilities, it is possible to compute the probability of a universe of variables with the chain rule which simply computes the product of all the conditional probabilities for every variable given the set of ancestors of each variable. Since probabilities can be conditioned upon any subset of variables, different types of reasoning are supported by Bayesian Networks. More precisely:

- Diagnostic reasoning can be performed from symptoms to causes.
- Predictive reasoning can be used to update the beliefs on some effects when information on new causes becomes available.

---

<sup>3</sup> Also known as polytrees



Also, a type of reasoning known as intercausal reasoning is possible. Consider for example the influence of two nodes A and B on a single effect, C. If after performing either diagnostic or predictive reasoning, new evidence is available on one of the causes, say A, then the probability on B can be accordingly updated. This reasoning can be used to understand, for instance, given a set of effects (e.g., metrics values) which causes influence them (e.g., design decisions), and select other values for the input variables to see their effects.

If the model built so far satisfies d-separability, then Kim and Pearl’s message passing algorithm for computing accurately the probability of each variable can be used. Otherwise (i.e., in the presence of d-connected paths), more laborious methods have to be applied which usually do not result in accurate values. However, if exact values are required, still some techniques can be applied like clustering the belief network.

Usually the domain knowledge is manually acquired from experts. This leads to the effect known as “knowledge bottleneck” since there might be no expert at all to interview or because the elicitation can become an awkward process. To make it even worse, knowledge elicitation is usually error prone, time consuming and a very expensive task. Instead, whenever possible it would be highly desirable to automate the knowledge acquisition by means of machine learning for either deriving the causal relationships or the Conditional Probability Tables (CPTs) governing them. From the principle of Common Cause it is possible to anticipate different types of relations among variables. Different methods can be used to automatically assist in the modeling of belief networks, including Path Models (Korb, et al., 2004) pag 153, Conditional independence learners whose main goal is identifying variables that do no affect each other, and the Pearl’s Network Construction Algorithm (Korb, et al., 2004) (page 38). The problem of automatically deriving the topology of a polytree is usually referred to as causal modeling.

On the other hand, CPTs can be either defined upon the results of elicitation from expert knowledge or by exploiting local information. This is typically known as statistical modeling.

Summarizing, BBNs present a set of advantages, with a significant impact in the context of change impact estimations. If an event is known to happen (a node takes a unique value), the BBN can be fed with probability 1.0 for that value. However, any probability distribution

can be used. This is, BBNs fairly generalize the behaviour of many other decision systems, which are often deterministic. For example, in the likely case of not knowing the probability of some input variables (also referred as decision variables), it is usually assumed that they are all equally likely though other scenarios can be defined as well.

Although the most typical reasoning approach is a straight application of the definition of conditional probabilities, which are updated according to the Bayes' Theorem, there are different ways of applying inference. Some of them are, but not necessarily limited to: variable elimination, mini-bucket elimination or clique propagation. In general, it is possible to run different inference algorithms over the same model.

Explanations can be easily generated. They result from the causal links that affected (up to a given probability which does not exceed a given threshold) the node under consideration. The usage of probabilities allows designers to carefully review the behavior of the BBN.

Since BBNs are fully probabilistic methods, other methods for estimating the a-priori probabilities (such as max-likelihood estimation) or learning the structure of the BBN (mainly based on Monte-Carlo procedures) are possible. In this regard, top-down inference (also known as predictive inference) can be seen as a generalization of Markov stochastic models. Indeed, there are generalizations of BBNs that can behave as Hidden Markov Models when reasoning in a top-down fashion while providing additional functionality if they are executed in bottom-up or combined mode. The same idea can be further generalized to the so-called Logical Markov Models.

### **3.3.1 CAUSAL MODELS AND BAYESIAN BELIEF NETWORKS**

One of the most important advantages we have mentioned of the BBN is that they allow describing causal relationships. Originally, reasoning models were thought as graphs, which consisted of nodes, which can take one among several values from a given range or domain, and arcs that stand for some sort of relationship between a pair of concepts or nodes. Propagation of values throughout the graph was mainly based on certainties and was driven by the inference rule of Modus Ponens. Thus, though they significantly relied on probabilistic computations, they did not obey the fundamental laws of statistics. Initially, this

was not a problem, since most "expert systems" were intentionally devoted to provide significant explanations of their conclusions ---mainly as a trace of rules followed throughout the model. Examples of this sort of "expert systems" are MYCIN (Shortliffe, 1976) and PROSPECTOR (Hart, et al., 1977).

This completely changed with the introduction of Bayesian Belief Networks. While Bayesian Belief Networks do retain the ability to produce explanations, they do adhere to a full probabilistic calculus, making the resulting explanations easier to understand and, more importantly, to debug and trace. From this perspective, it shall be clear that the most significant contribution of BBN is the propagation model suggested. In this case, models consist of acyclic graphs with nodes  $X_i$  which can take one among several values from a domain  $D_i$  (Kjærulff, et al., 2005). Each node can be connected to an arbitrary number of neighbors, setting up a causal dependency, which is characterized with a conditional probability. For example, if node  $X$  is connected to node  $Y$ , then  $Y$  is said to be conditioned by  $X$  or, equivalently, that  $X$  is the cause of  $Y$  with a given likelihood. Moreover, if  $Y$  is also connected to  $Z$ ,  $X$  affects  $Y$  and the ultimate values of  $Y$  do also propagate to  $Z$ .

In short, Bayesian Nets consist of (Kjærulff, et al., 2005):

1. A set of nodes that represent random variables, whose values can be known or have a given probability associated
2. A set of directed causal links represented as arcs between nodes which stand for concepts.
3. Each node contains a Conditional Probability Table (or CPTs for short) which state for every pair <cause, effect> its likelihood

Figure 3-7 shows a simple BBN consisting of four nodes, with its causal links and their corresponding Conditional probability tables

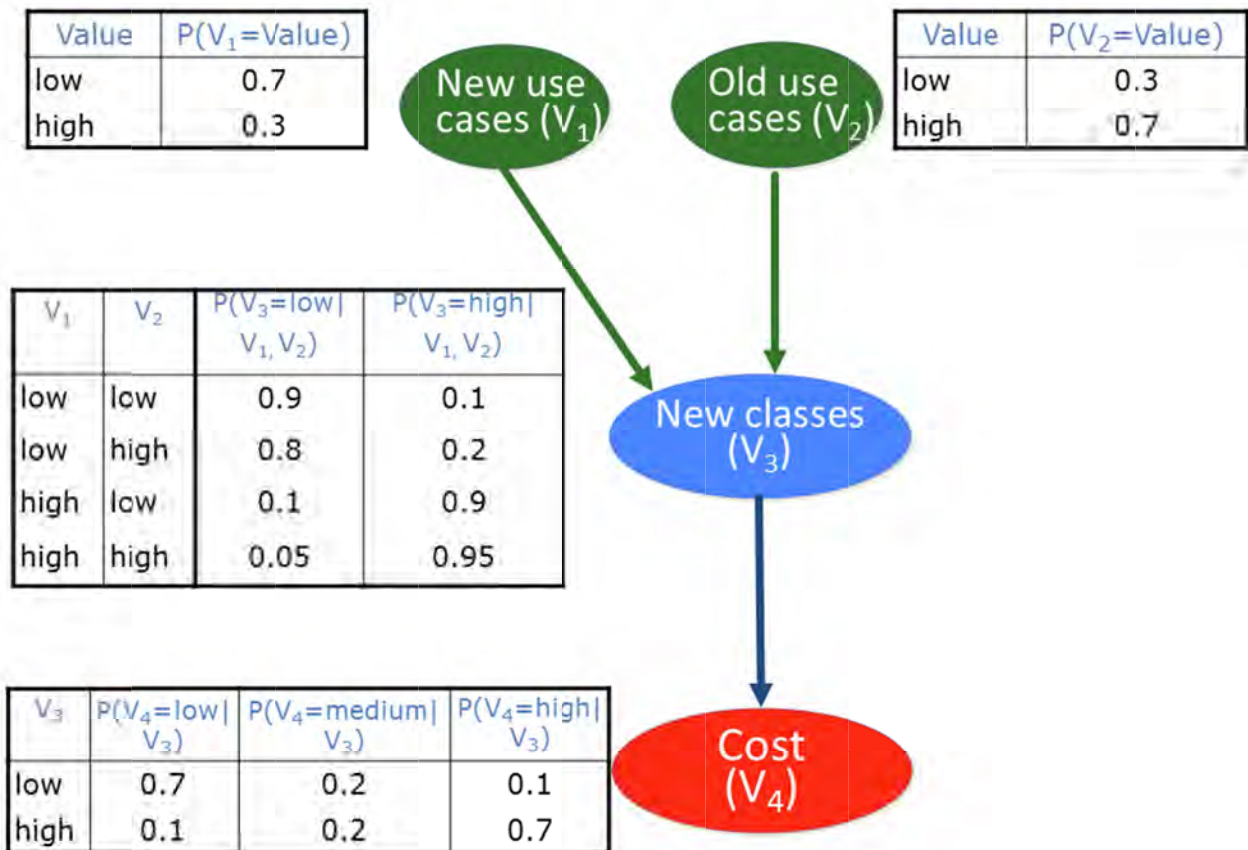


Figure 3-7: Simple BBN with four variables, their causal links and CPTs

Thus, using a BBN (which is expressed as a causal probabilistic Directed Acyclic Graph, DAG) for a particular application domain requires completing a set of tasks (Korb, et al., 2004) namely:

1. **Identification of the relevant variables**, which will form the nodes of the graph. In our case these variables will include all elements that affect the estimation of costs according to the selected metrics. This step also incorporates the definition of the variables values. Variables can have *discrete* or *continuous* values. If it is a discrete variable, the specific values have to be defined.

2. **Identification of the causal dependencies among variables.** If a variable X might affect the value of another variable Y, an edge is defined in the graph between the corresponding nodes, taking special care of not creating a cycle.
3. **Parameterization of the probabilistic information of the graph.** This step requires defining the prior probabilities for each root node in the graph, and the conditional probability tables (CPTs) associated with each non-root node that quantifies the relationships between nodes.

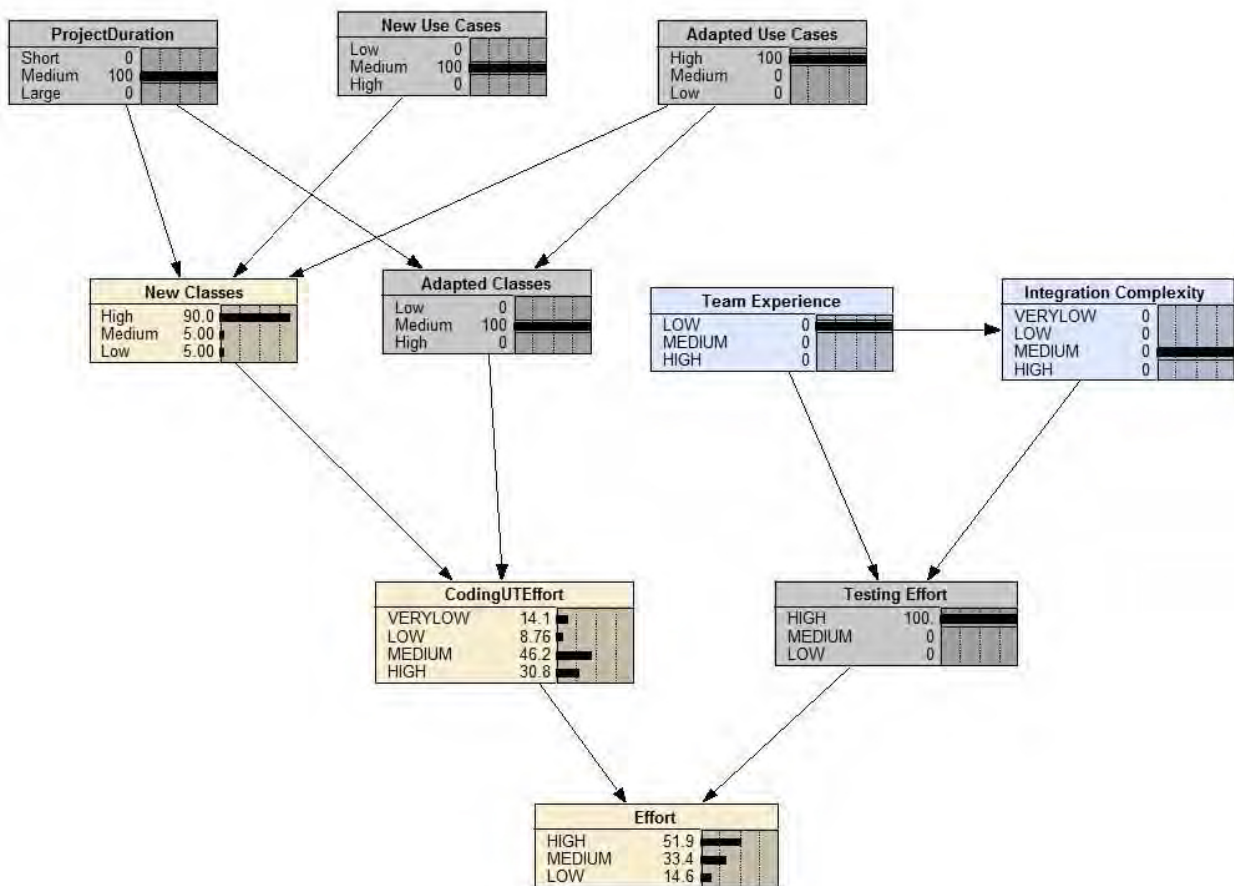


Figure 3-8: Example of a BBN as shown using Netica, a BBN tool

So for instance, Figure 3-8 above shows a direct causal relationship between “adapted use cases” and “new use cases” on a given design, indicating that there is a direct causal

relationship between the number of use cases to be implemented as new, and the number of use cases that need to be adapted, with respect to the new classes to be developed. Each of these nodes represents a different variable, which can have discrete or continuous values.

The advantage of describing a probabilistic argument via a BBN, compared to describing it via mathematical formulas and prose, is that the BBN represents the structure of the argument in an intuitive, graphical format. The main use of BBNs is in situations that require statistical inference — in addition to statements about the probabilities of events, the user knows some evidence, that is, some events that have actually been observed, and wishes to infer the probabilities of other events, which have not as yet been observed. Using probability calculus and Bayes theorem it is then possible to update the values of all the other probabilities in the BBN. This is called **propagation**. Bayesian analysis can be used for both 'forward' and 'backward' inference (Tang, et al., 2006)

Although the underlying theory (Bayesian probability) has been around for a long time, building and executing realistic BBN models has only been made possible because of recent algorithms and software tools that implement them.

Once we have defined a BBN, there are several ways of using it, but the three most used ones are:

1. **Prediction** (top-down): given the values to known variables, obtain the values for goal variables, by propagating values through DAG. In the case of this thesis, this scheme will be used, for instance, to estimate the given metrics from design decisions.
2. **Diagnosis** (bottom up): given the observed (or required) values of the goal variables (metrics in our case), obtain the most probable causes (design decisions, for instance)
3. **Combined approach**: it can be used to understand, for instance, given a set of effects (metrics values) which causes influence them (design decisions), and select other values for the input variables to see their effects.

### **3.3.2 BBN,S USED FOR CHANGE IMPACT ESTIMATIONS**

Other considerations regarding the usage of BBNs in software engineering management are the following:

- BBNs can be easily traced so that it is possible to build explanations for the resulting values (or a posteriori distributions). Handling design rationale is a key aspect in model design, since complex interactions among components can be early detected by understanding the hidden relations among those components.
- BBN's are unit-independent: Although the case of discrete (multimodal models)/continuous (gaussian models) variables is an important factor to take into account, BBNs are "unit-independent" as long as they just simply propagate measures along with their respective probability distributions. This way, the same critic factors can be measured in different units such as functional points, thousands of lines of code, etc.
- Different software/project metrics can be used to adjust themselves, e.g., the effort required to develop a project can be adjusted by process and personnel quality and this, in turn, can be automatically derived from other factors (instead of using the Brooks factor as in (Radlinski, et al., 2007), which might include professional background, physical mobility which, at the same time can be related to each other in some way and so on). In the case of model design, a change in the design of a hardware component has an impact on the control software that can be reflected by combining the BBN models of both designs.
- Users do not have to enter observations for all the "predictor variables" (or decision variables) since "a priori" distributions (either based on past data, or learned/updated by other modules, or simply handcrafted by authorized users) can also be satisfactorily used. This improves the ability of software engineers to focus on the relevant data.
- A representation for the change impact model is needed. A critical step in the construction of BBNs is to have some sort of automatic representation for

architecture and design decisions (such as in (Tang, et al., 2006)). It does not seem difficult to propose an extension of UML so that it would be possible to derive: a Directed Acyclic Graph along with “a priori” distribution probabilities for the root-nodes and CPTs for the non-root nodes. A few ideas follow:

- Information for the CPTs can be manually fed into the system by experienced users. Although this does not seem to be the right choice in the general case, it should be always possible to the final users to edit the conditional probability tables (CPTs) for either modifying the probabilities or changing its parameters (as in the case of Gaussian models where the mean and variances have to be specified). Another alternative consists on providing qualitative values for the probabilities (high, medium, low) and then have an automated way of translating those into actual probabilities. In general, it is expected that dependent variables shall be characterized with discrete values in a rather restricted domain.
- It is possible to use existing data from past software projects. However, it has the serious drawback that it requires a careful review of the design processes followed in those projects trying to understand what was done and more importantly, why, since these are the decision criteria to be modelled in the BBN. In some cases, it might be desirable to devise automatic means for automatically converting information from architecture/analysis/design artifacts (SLO,s) into BBNs
- The inter-dependencies among different factors estimating the effort are far from being well known and described in the specialized bibliography. Thus, causal discovery provides a nice opportunity for automatically discovering (or stressing) the importance of the correlation among different factors
- Moreover, either if the BBN structure has been automatically obtained or not, the probability distributions can be difficult to tabulate. In this regard, statistical modelling with a Gibbs sampling can be used to derive preliminary probability distributions from the careful study of past projects or as a result of interviews with software experts in the domain. It would be still highly desirable to automatically review and probably refine the existing data with automatic modules built with learning techniques



- Most causal dependencies to be modeled in a BBN are still highly domain-dependent so that experience might still be necessary. In order to make the analysis and design decisions independent from the people involved in the project and in order to allow others to better understand the development and to evaluate the impact of all decisions (along with other hypothetical decisions), learning would provide a significant added value. For example, it might be observed that some sorts of architectural decisions lead to a significant impact on the risk of a given component of a software system, so that the system could automatically suggest including this dependency in the future
- Learned data can be easily reused across a large number of designs. The systems to be conceived shall not be restricted to a particular application domain, but instead should allow users to refine their models to different application domains (such as payload data segments, control centers, space software, etc.) and it shall be expected that in many cases a large number of dependencies shall be appearing elsewhere within the same domain. Consider, for example, the case of “Shared Dependencies”, according to (Zhao, 1998), which would model the relation between antennae, centers and stations all across the European Space Agency (ESA) network. They would appear in most projects modeling the production chain of any mission. Or consider all the “telemetry constraints” to appear as constraint dependencies in a model for predicting the complexity of software components to be used in control centers
- Since BBNs contain the rationale of analysis and design decisions and they can be used for both predicting and diagnosing with regard to various levels of decisions, they could be used also for:
  - Automatic generation of documentation. Especially if an explanation engine is built along with the BBNs (not a human-readable explanation engine, but just some sort of info that aids in understanding why BBNs produce the values they do).

- To assist in the generation of test cases to be used in FATs, OSATs, etc. This seems quite straightforward from the purpose of this project since BBNs shall be used to estimate the complexity, expected number of errors, etc. of well-separated software components. The more complex or error prone, the more to be tested.
- Sensitivity analysis on the decision variables. It seems quite interesting to understand how stable or not are different decision variables or assumptions happening in a given software project. While BBNs are usually used for deriving answers in the form of probability distributions (so that for each allowable value, its probability is computed), they could also compute how much a given variable has to change in order to have a significant impact in another variable. This might be quite interesting since causes and effects are usually related in a non-linear fashion so that big changes in some causes do only slightly affect some effects while the contrary can happen as well for other variables.

### **3.3.3 THE AREL MODEL**

Tang et al. (Tang, et al., 2006) use BBNs to model and quantify the probability of the causal relationships between design decisions and design elements. For this purpose, they define the Architecture Rationale and Element Linkage (AREL) model to represent the causal relationships between architecture design elements and decisions. AREL exploits the idea of representing the causal relationships as arcs and objects as nodes. They form a DAG over which it is possible to propagate statistics with the aim of tracing change impact decisions back from the architectural design of software.

Tang et al. claim that it is highly desirable to automatically derive the design of BBNs to be used in the project estimation reasoning. They suggest that all design decisions can be seen as either:

- **Architecture Elements (AE)** (such as a requirement, a use case, a class or an implementation artifact), or as
- **Architecture Rationale (AR)** (e.g., design decisions, design constraints, trade-offs, risks, etc.) which are related as AE->AR->AE.

Both elements are linked to decisions. In addition, <ARs> contain additional information related to the rationale. Figure 3-9 shows the elements of the model and their relationships.

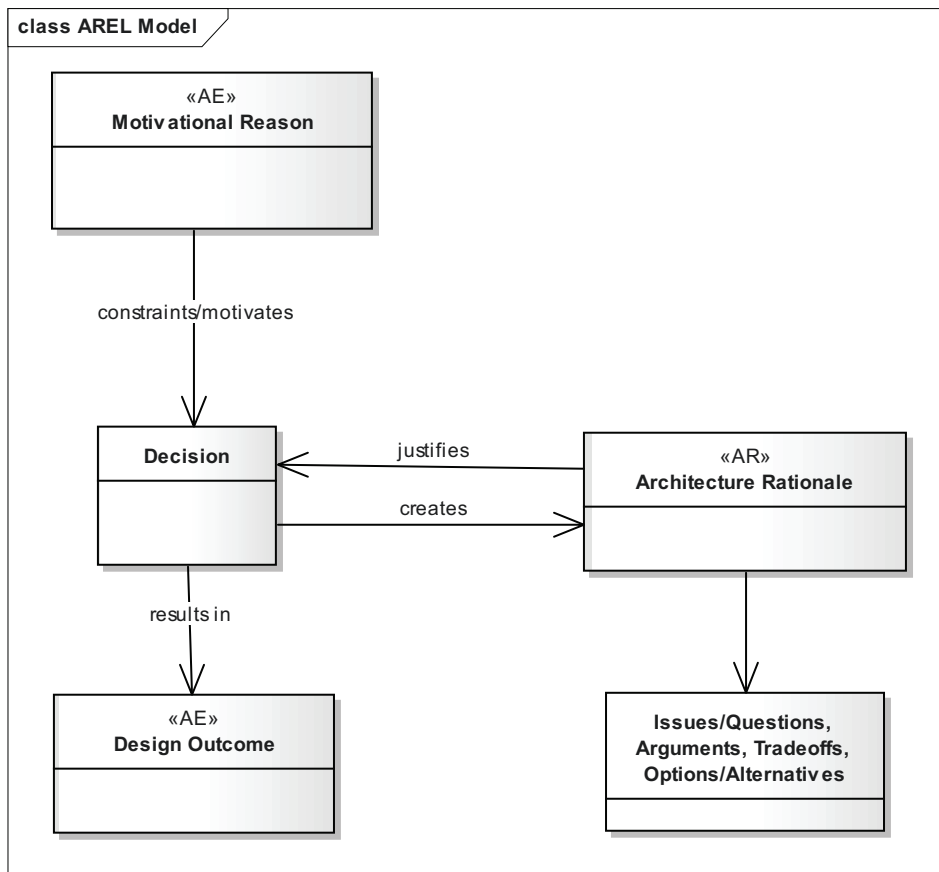


Figure 3-9: Elements and their relations in the AREL model.

Thus, an AE can be explained with regard to an AR, which, in turn, has an impact in another AE. An AE can be an input or an output or both when it is involved in two decisions. As an input, it can be a requirement, a use case, a class or an implementation artifact.

An AR represents a decision and contains the following elements from the design rationale:

- Issue of the decision
- Design constraints
- Assumptions
- Strengths and weaknesses of the design decision
- Trade-offs made for that decision
- Risks and non-risks in the design alternatives
- Quantitative justifications of a design alternative by costs, benefit and risks
- Any alternative design which has been discarded and its considerations

This model has three advantages: simplicity (only two kinds of concepts are represented), encapsulation (all design rationale is embedded within the set of Architectural Rationale ARs), and causal relationships (each AR acts as a connector among AEs and thus one can easily build BBNs). Also, this model can be easily integrated with UML. As a matter of fact, this model can be seen as a further generalization of that described in (Fenton, et al., 2002). This approach, though, does not automatically derive the conditional probabilities (CPTs) that are used in the non-root nodes of the BBN or the “a priori” probability distributions that are fed at the root nodes. Thus, CPTs and “a priori” probability distributions are specified by hand.

Decision variables are all discrete with a rather narrow domain. ARs can be “valid” or “invalid”, while AEs can be “volatile” (if they are subject to change) or “stable” (if they are not subject to change). The idea can be further generalized to arbitrarily larger nets. This reduction of the possible discrete values of AR,s and AE,s contributes to reducing the size of the corresponding CPT,s that are to be filled manually.

The authors stress the importance of the combined top-down/bottom-up reasoning approach to assist software engineers in the understanding of mutual influence between causes (or assumptions) and effects (or changes). That is, both as diagnostic reasoning or predictive reasoning. Likewise, Fenton et al (Fenton, et al., 2002) discuss some examples of these models of inference: whilst top-down can be used to predict the number of defects (or any

other parameter) from personnel (number, experience, etc.), bottom-up would be predicting the personnel (number, qualification,...) necessary to bound the number of defects below a given threshold.

Consistency checking is provided in the AREL via an automated consistency-checking tool that is used to detect the directed cycles in the underlying BBN. However, the tool relies on human intervention for the resolution of such anomalies.

AREL provides a way to link AR and AEs very intuitively for designers, linking UML elements by causal relationships. Moreover, by using AREL, design decisions and their rationale are recorded and traced. At the same time, the AREL model can be used for:

- Predictive reasoning (indicating design elements affected if one or more elements are to be changed)
- Diagnostic reasoning (identifying those causes that produced a given effect on an AE), as well as
- Combined reasoning (ripple effect, combining both predictive and diagnostic capabilities)

In that sense, it constitutes an innovative approach to the problem outlined in Section 3.1 (Change Impact Analysis) and it clearly represents a new approach to tackle change impact estimations. On the other hand, it has limitations on the workflows it can be applied to, since it limits itself to design elements and not elements related to other workflows (requirements, coding, testing, among them).

### **3.4 DESIGN RATIONALE**

Specifying rationale has always been considered important in Software Engineering (Schubanz, et al., 2012). Design rationale (DR) captures the knowledge and reasoning justifying the resulting design. This includes how a design satisfies functional and quality requirements, why certain designs are selected over their alternatives and what type of system behavior is expected under different environmental conditions (Tang, et al., 2005). In fact, any software development can be seen as a long sequence of design decisions (that in

turn implement *changes*), each having a different impact on the existing design or software artifacts. In a traditional development, developers and designers start with an initial set of requirements and implementation constraints, and they make a long sequence of design decisions with no clear statement of why they do things on a given way, and not another (Long Parnas, et al., 1986). Due to software’s mutability, design decisions are more likely to be changed during software development than other types of product development. (Burge, 2005).

### **3.4.1 DESIGN RATIONALE: SERVICES**

There is a common consensus on many aspects related to this area of knowledge: a first conclusion is that design rationale provides an evident added value to the development. On a survey conducted by Tang et al. (Tang, et al., 2005) around 85% percent of the respondents considered design rationale as something important for the design justification (setting its importance between 4 and 5 in a scale from 1 to 5). A similar percentage was obtained for the frequency of considering alternative architecture designs in their designs. Lee (Lee, 1997) discussed seven possible “services” that can be provided by any tool that provides design rationale, some of which pending of a deeper exploration, namely:

1. **Better design support:** If the rationale is correctly structured, it can help any designer to identify the different alternatives explored before the decision was actually taken, which in turn could serve to detect inconsistencies in the existing design, as well as supporting decision-making. As an example (Conklin, et al., 1991) report on the use of a DR tool at the NCR company that helped to identify several design omissions that would have cost three to six times more than the cost of capturing and constructing the rationales.
2. **Dependency management:** Any design can be viewed as the process of managing dependencies to yield a product that honors all dependencies among requirements and the components that implement them (Lee, 1997). Burge and Brown (Burge, et al., 2002) mention the possibility for the user to using rationale to verify that the design meets the requirements and the designer’s intent. Design rationale eases

traceability, since it defines the existing path between requirements, the design decisions taken, and the elements of the design produced as a result.

3. **Collaboration and Project management.** Explicitly assessing the rationale means that a common vocabulary and project memories are shared across the stakeholders involved in the project, which contributes to reduce the interaction among engineers and contractors, and helps to improve the design process (Lee, 1997). Conklin and Yamekovic (Conklin, et al., 1991) claim that, once the development team was trained in the use of a gIBIS, a design rationale tool, users had the impression that meetings were more productive.
4. **Reuse/redesign/extension support.** Reuse or redesign is improved in two different ways: on one hand, it serves as a repository of knowledge, and secondly, the decisions themselves that can be useful for future decisions in similar projects. Rationale can serve to identify those portions of the design that can be reused (Burge, 2001)
5. **Better maintenance support.** Because design rationales explain the design decisions made, they can also help maintain the design. Burge provides an example of use of DR techniques in order to perform corrective, perfective and enhance maintenance (Burge, 2001)
6. **Learning support:** design rationale contains important aspects of the know-how used in each development. This information is very helpful for system designers, and it is also possible for computational agents to learn from these decisions as Bracewell et al. have done with Dred2.0 (Bracewell, et al., 2009).
7. **Documentation support:** design rationale can be used to automatically generate documentation (i.e. not only what is designed and developed, but also why it was developed). That is, for the elements of the design, we do have information regarding why they were generated, and the reasons for these elements that lead to their generation or modification. This information can be used to provide in the documentation details about offering a picture of the history of the design and reasons for the design choices as well as a view of the final product. (Burge, 2001)

### 3.4.2 CAPTURE OF DATA

According to the degree of formality, the approaches that are used to represent design rationale can be divided into three main categories: informal, semiformal, or formal (Lee, 1997) (Burge, 2005).

A key issue on design rationale is the capture of data (Burge, 2005). Capture understood as the recording of the design rationale, either during or after designing. Lee (Lee, 1997) classifies the different methods as:

- **Informal:** data are gathered without any kind of structure for representation: descriptions in natural language, audio/video recordings, raw drawings, etc...
- **Semiformal:** in which only parts of the representation are computer readable, being the rest informal (unstructured text, raw data, etc...). The advantage of semi-formal representation is that they can support many computational services that are not possible in the informal representation, and can also lead to advantages in capturing rationale (Conklin, et al., 1991). Most DR systems fall into this category. A semiformal representation is the best trade-off if the primary services are to help people archive, retrieve, and examine the reasons for their decisions.
- **Formal:** In formal representations, objects and relations are defined as formal objects that the system can interpret and manipulate using formal operations. Information added becomes part of a Knowledge Base.

In general, there is a growing recognition of the role of maintaining and documenting the rationale for the technical decisions, but there is a plethora of methods and tools, and none of them have been adopted as standards. For instance, the IEEE 1471-2000 standard (IEEE, 2000) does not have any further ontology associated to it and it is not clear what types of specific information should be captured as design rationale (Tang, et al. 2005).

The key for the capture of data is the model and the usability, understood as the capability of the DR to provide useful information (Burge, 2005)



### 3.4.3 NOTATION

The way to use DR varies depending on representation format and contents. Since the seventies, there have been a plethora of different notations for design rationale. They all have in common that they are based on entity-relationship models due to the convenience of the use of node-link structures for computation as well as the view of design reasoning as a sort of argumentation. That can also be seen as structured activity that can be represented within a formal structure of nodes and links (Shum 1991). A notation particularly important, that was the basis for other notations that extended it, is IBIS (Issue-Based Information System). IBIS was developed during the 70,s by (Kunz, et al., 1970). IBIS uses a set of elements (nodes in IBIS terminology) (such as positions, arguments, and resolutions) as well as a set of relations that are used in a formal way to represent the rationale behind decisions. The IBIS notation was used by Conklin and Yamekovic (Conklin, et al., 1991). They developed two different tools (itIBIS and gIBIS) and tested them at NCR during the nineties. Nowadays, the Dred2.0 tool, based also on IBIS, is being used by a subsidiary of Rolls-Royce (Bracewell, et al., 2009).

The model proposed by IBIS had as its central element the *issue*. An issue is stated in the form of a controversial question, with different points of view: issues can be categorized as factual, deontic, explanatory, instrumental and conceptual (Noble, et al., 1998). For any given issue, there can be many different positions (person's responses to given issue), indicating agreement or disagreement with the issue. In other cases, issues can have different alternatives and each position can consider an optimal alternative. Positions are supported or opposed by arguments, against or in favor of a position.

Lee (Lee, 1989) extends IBIS and creates a new language, the Design Rationale Language, or DRL, "a language to provide a vocabulary for representing the qualitative aspects of decision making -- such as the issues raised, pro and con arguments advanced, and dependency relations among alternatives and constraints, that typically appear in a decision making process".

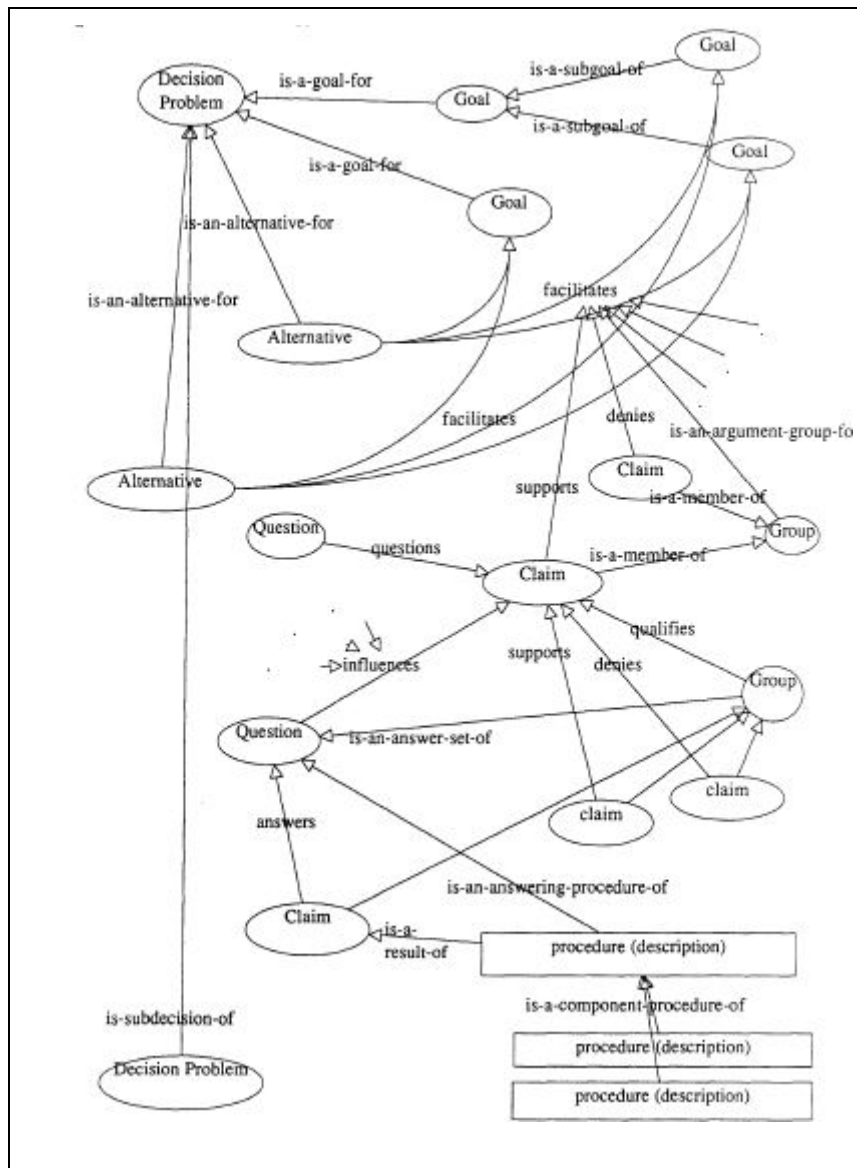


Figure 3-10: DRL ontology, as depicted in (Lee, 1989)

The fundamental objects of DRL are goals, alternatives, and claims. Alternatives represent the options to choose from, goals specify the properties of the ideal option, and claims constitute arguments relevant for choosing. Other objects are no less essential in a decision making, but either they are special cases of the above three (e.g. Decision Problem is a subclass of Goal) or they are useful in general (e.g. Group, Viewpoint) or they are auxiliary (e.g. Question, Procedure).

Finally, MacLean et al. propose a model based on Questions, Options and Criteria (QDC) (MacLean, et al., 1991) in which questions identify the main issues, options provide possible answers to questions, and criteria serve to compare and assess the options. The approach they propose is Design Space Analysis, which takes into account the justifications for each possible design, and reflects considerations such as consistency, models and analogies.

#### **3.4.4 RATIONALE FOR DESIGN: RATIONALE AND SEURAT**

Moreover, Burge and Brown (Burge, et al., 2008) describe the Software Using RATIONale (SEURAT) system. In line with the description of change impact analysis that we provided in Section 3.1 of this thesis, the SEURAT system is based on the belief that software development is, at its essence, a decision-making process.

A software development is therefore the final outcome derived from a set of design and implementation decisions that are taken during the whole project development. They highlight the fact that it is necessary to capture the rationale for decisions made, and the developer’s intent behind their decision choices, as well as their evaluation of their assumptions, requirements, quality attributes, and inter-decision dependencies.

The methodology outlined by Burge (Burge, 2005) and its tools can be applied to the different workflows of the software development (Requirements, Analysis, Design, Implementation, Testing and Maintenance). Moreover, Burge and Brown consider that these different workflows overlap in time.

Rationale involves therefore not only the design phase, but also the remaining phases of the software lifecycle. This rationale can be used for documentation, revision of designs, design reuse, validation, evaluation and, particularly, for maintenance.

Burge (Burge, 2005) discusses on methods for Design Rationale Representation, Design Rationale Capture and Design Rationale use, as well as on Software Design, Software Architecture and Software Maintenance. The final tool that emerges as a result of this work is the so-called SEURAT system. It defines the knowledge representation for the rationale, provides a semi-formal argumentation structure, and uses inference to detect errors in the rationale structure and content. SEURAT supports semantic inference via an argument

ontology that describes a hierarchy of reasons for making software decisions that contains several levels of abstraction.

SEURAT uses a representation of rationale that is an extension to the DRL representation used in SYBIL (Shum, 1991) called RATSpeak (Burge, et al., 2003). The system was conceived with the following requirements in mind:

- **Argumentation support:** a representation was needed to capture the different alternatives and the reasons for and against them.
- **Traceability to requirements:** the requirements met or those that drive a decision are explicitly represented.
- **Non-requirement arguments** for decisions are also represented.
- It was possible to use natural **language descriptions**.
- Questions for **particular decisions** are also explicitly represented.
- Finally, it was also possible to generate a **history of the decisions** taken using the system.

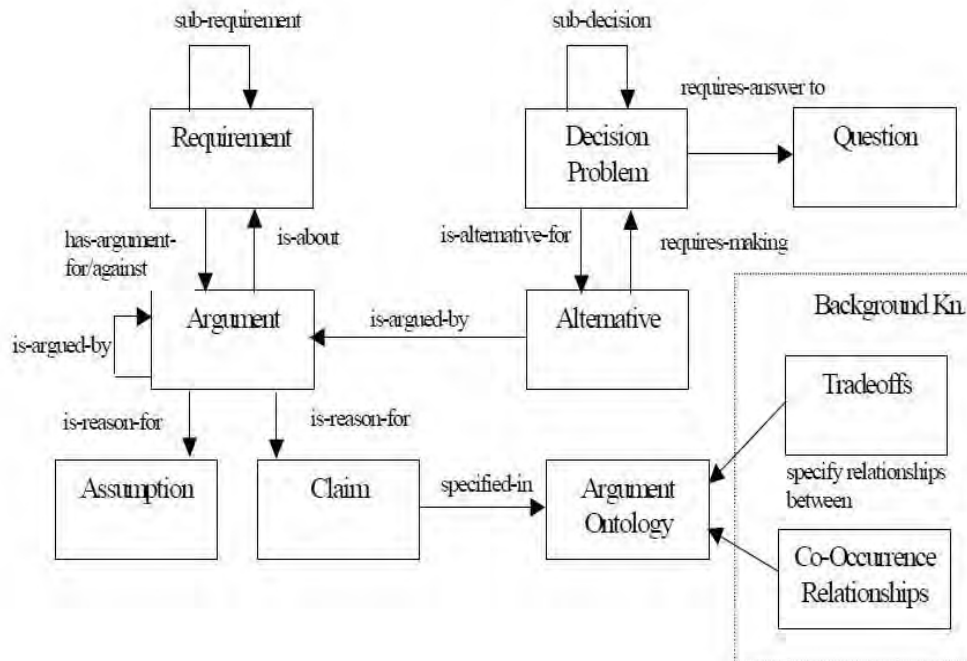


Figure 3-11: Seurat's model for argumentation's: RATSpeak, as shown in (Burge, 2005)

As can be seen in Figure 3-11 in SEURAT there are decision problems that are to be made as part of the development process. A decision problem answers a set of questions, and has a set of alternatives, each of them supported by a set of arguments. Arguments can be for and against the proposed alternatives, and they can satisfy, address or violate requirements or assumptions. Claims are in turn reasons why an alternative is good or bad. They are specified using argument's ontology. Ontology background knowledge gives relationships between different arguments in the argument ontology and is used to check the rationale for violations of the relationships. SEURAT was developed as an Eclipse (Clayberg, 2008) Plugin and provided all these capabilities in a set of Eclipse Views and Editors.

### 3.5 CONCLUSIONS: CURRENT FLAWS OF ESTIMATIONS AND EXISTING PROBLEMS FOR DESIGN RATIONALE

To summarize the conclusions obtained in this chapter, we find the following:

- The majority of the works on SW estimations focus on the estimations for quality, costs and risks; there are very few studies focused on estimations for change impact. Traditional estimation methods are rarely based on the change impact analysis. Instead, a set of attributes is used (team size, programming language, lines of code, etc...) for estimating effort, risk and quality.
- Traditional models of estimations are supported by incomplete models or estimation techniques: Traditional metrics approaches, such as COCOMO, often driven by regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use measurement to analyze and minimize risk.
- Most of the estimation techniques are not able to deal with uncertainty. During the earlier phases of the projects, most of the parameters (such as LOC, or code size) are unknown. These missing parameters for the model are in turn evaluated to be input into the estimation and therefore the final estimation relies also on evaluations, and not facts, therefore reducing its accuracy and reliability. For estimations to be taken into account as a valid tool, it has to be possible to perform them in the presence of uncertainty, and they need to be refined periodically, as uncertainty decreases.
- In the majority of the cases, software estimations lack causal modelling; so for instance estimations based on correlation between measurable software metrics or project parameters and the target estimates of interest do not reflect the causes of the correlation. The relationship between cause and effect can be easily misunderstood when using exclusively these methods and therefore wrong conclusions can be withdrawn. Knowing this cause-effect relationship would be required in order to take the right decisions. In other words, this view of a development as the result of a cascade of decisions is of paramount importance.
- Design changes determine variations in effort, risk and quality during the whole lifecycle of the project. The result of a project can be seen as a network of design decisions and changes; during the whole lifecycle of the project, different changes to the design model need to be undertaken. These design decisions have a strong

influence on the project estimates, causing a cascade of changes that affect the quality, the effort and the schedule. If the design decisions are not used to update the estimation models, they become incomplete.

- Extrapolation of results to other projects leads to errors. The particularization for a given scenario improves accuracy but leads to non-extrapolative estimators. In order to provide better accuracy, estimation models are traditionally done relying in the available historical data from past projects. This process leads to models that cannot be reliably used across different software application domains; that is, particularized models cannot be reused in projects that have different nature and characteristics, as it is usually the case in change impact scenarios. Once again, a simple difference in a decision change can have a very important effect, and therefore it is very difficult to apply similarities.
- Estimations should take into account the innovative character of most developments: some projects differ clearly from its predecessors either because they are in an innovative application area, they use a new technology, or they have special characteristics that affect their external parameters. Estimation tools need to have some way to add these characteristics into their models to be usable in these types of projects, that are, due to the novelty nature of software, very frequent. Environmental factors are most of the times not considered, and have a tremendous impact.
- There is a traditional resistance to record the rationale for decisions: there is common belief for developers to trace the rationale behind the decisions taken, the alternatives, and the pros and cons evaluated during the whole lifecycle. This is one of the most important obstacles to overcome.
- Change impact is tied to change due to design decisions, which in turn are tied to the rationale for these decisions, and these decisions in turn determine the traceability among SLOs.
- All these problems limit seriously the usefulness of the current estimation methods and tools for change impact.

The following conclusions can be extracted from these problems:

- Estimation methods should be complete, covering the design changes made during the whole lifecycle. An incomplete view of the estimations would lead to incorrect decisions.
- Current estimation approaches and change impact techniques mostly rely on source-code based input data. Being the current trend the use of model-development techniques, it would be required for estimation systems to rely on design elements or higher SLOs, rather than in code elements. For instance, using the model elements (e.g. UML Use Cases) rather than the code elements (e.g. Lines of Code) we can obtain a better accuracy of the estimation for the early phases. In contrast to the source code, the model is available from the early phases of the project, and therefore early estimations could be performed starting from the requirement models. This will partially mitigate the problem previously mentioned before. The earlier that decisions are taken in the project, the easier and less costly that these decisions are. Most risks could be mitigated or eliminated having an anticipated (and accurate) estimation.
- A formal reasoning of the causal relationship is also required. It is not enough to capture the correlation between different parameters. New approaches tend to capture the stochastic causal relations and to formally reason about them. This allows engineers exposing the most probable causes of the produced estimates and gives more precise instruments for project estimation and control.
- During the whole lifecycle, design is subject to changes: estimation models and techniques should allow us predicting the consequences of these changes and the “ripple effect”. The possibility of “what-if” analysis is a must for successful project execution. Again, these estimations should be based on the new design, since the source code will not be available until the later phases of the project.
- Estimation models should be contrasted against reality, without losing generality: past projects data can and will be used to test their accuracy, and reliability. This data can be used to tune the model. But measures should be taken in order to keep the



models useful for general use, being one of the most valuable characteristics of a model its adequacy to a wide range of project types.

- Two of the models analyzed seem particularly important: the AREL model from Tang, and the RATSpeak (implemented in SEURAT) model from Burge. The following is a comparative of both systems, and also an evaluation of the elements of them that can be used in our model.

### **3.6 AREL AND SEURAT: ANALOGIES AND DIFFERENCES**

By analyzing both systems (SEURAT and AREL) we reached the following conclusions:

- Both models and systems conceive the result of a development as the final outcome of a network of change decisions that are triggered by a rationale. A set of input elements (arguments, requirements, assumptions) are taken into account for a decision to be made, and the decision produces a set of new elements that represent the possible outcomes of this decision.
- The AREL model (Tang, et al., 2006) provided causal reasoning under uncertainty for change impact analysis based on BBNs; that is, it was targeting change impact estimations based on probability. On the other hand, SEURAT (Burge, et al., 2008) provides a model for capturing the rationale behind decisions.
- The model in SEURAT is much more powerful, in terms or representation of the rationale behind decisions, than the model in AREL. However, SEURAT lacks a probabilistic model for the elements that are subject to change (SLO,s that can or not take part of the implementation)
- Also, SEURAT's integration with Eclipse is a very strong point, since it allows its use by developers, easily integrating design decisions with the SLOs.
- Although SEURAT's model focused on Maintenance, it also allowed engineers its use throughout the whole lifecycle. Meanwhile, the AREL model was much more tied to the design. In fact, that is the main flaw we identified of the AREL model.

- Both of them consider the problem of data gathering as a key obstacle in design rationale; in general, developers don't see the benefit of gathering the information related to alternatives in the design, arguments on how a decision was made, etc.... They clearly recognize this problem as the main obstacle to be overcome by any tool of these characteristics.

## **4 OBJECTIVES OF THE THESIS**

The main objective of this thesis is the development of a change impact model based on BBNs. The main development took part in the framework of an ESA project: “Cost Complexity and Change Impact” for ESA-ESTEC as detailed in its final report (Ocón, 2010). The work on this project focused on the development of the so-called Adaptable Project Estimation System (APES) tool, a system designed to provide different types of estimations for software development projects in the aerospace sector. The project had two different areas:

- **APES for Project Estimations (APES-PE).** The first component of the project focused on traditional project estimations (like those commented in section 3.2 of this thesis). These estimations provide effort, quality and risk estimations of the project, given a project from which a set of attributes is known. The design of the topology of each BBN was suggested by a tool, Weka (Hall, et al., 2009), and reviewed by an expert in the field, that introduced changes. Once the BBN network is considered correct, CPTs are adjusted automatically based on existing data from previous projects developed by two different companies of the aerospace sector (Fuentetaja, et al., 2013).
- **APES for Change Impact Estimations (APES-CIE).** The second component, that is the subject of this thesis, focused on the estimations for change impact. Here we found that the approach had to be completely different. The network of interactions between SLOs (the change impact analysis indicated in Section 3.1) was the driver for the estimations and the underlying BBNs, and therefore the BBNs generated as a result were, for each new project, different in nature, and it was not possible from previous data from past projects not only to adjust the CPT,s, but even to generate a topology. Therefore, a totally different approach had to be taken for change impact, and this is the object of this thesis. The following sections outline the APES-CIE model and the rationale that lead to its ontology.

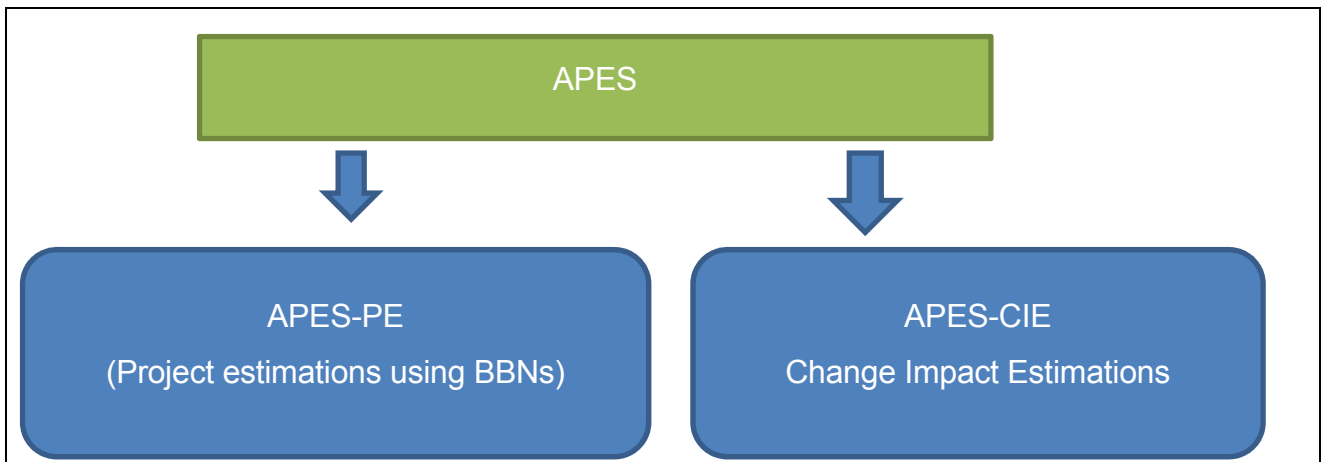


Figure 4-1: The APES project and its two main outcomes

The main objectives focused on current needs in project development, being the first one the need to determine the change impact throughout the lifetime of a project. As Kruchten points out (Kruchten, 2000), in any project there are several workflows that need to be managed in parallel in a coordinated manner: requirements elicitation, design, software construction, deployment, testing and maintenance. These workflows span through the entire lifetime and have strong dependencies among them. Therefore our first objective is:

**O1: to develop a model to provide reliable estimations in order to determine the change impact throughout the whole lifetime of a project.**

That is, our aim is to develop a model that will be valid not only for the design process, but for other processes as well as to related to software (or even hardware) development: requirements elicitation, testing, maintenance. The model is to be validated via the development of a system, the so-called APES-CIE (Adaptable Project Estimation System – Change Impact Estimation part), and the testing of the validity of this model throughout the use of APES-CIE tool in real developments.

In addition, this study has been promoted by the European Space Agency (ESA), so it has to focus on space software development and maintenance (for on board and on-ground software). As such, the model for change impact estimations and design rationale shall be in line with the ESA standards for SW development, also known as the European Cooperation

for Space Standardization, or ECSS standards (ECSS-E-40, 2009) (ECSS-E-40, 2003). Therefore a second objective was:

**O2: The APES-CIE Model shall be valid for the aerospace market**

In Sections 3.1 and 3.2 we have seen that one of the main problems of estimations and change impact is the uncertainty that is inherent to any development. Our model shall take this into consideration; that is, that change impact estimates will have to be done without a complete knowledge of the consequences of a change nor the right way to implement it, namely:

**O3: The APES-CIE model shall contemplate the inherent uncertainty associated to any development. This uncertainty shall be managed using statistical techniques.**

This means that, throughout the whole development lifecycle, the model will provide estimations for changes, since uncertainty is considered inherent to any development. And these estimations will be refined and adjusted periodically by the stakeholders.

As we have seen in section 3.3, BBN is a statistical technique, that is able to provide relevant estimations in presence of uncertainty, and therefore:

**O4: The statistical technique to be used by the APES-CIE model shall be based on Bayesian Belief networks**

Which means that the uncertainty associated to the elements involved in a change will be represented by variables with probabilities associated to them, and these variables will be linked in a Bayesian Belief Network. In other words, we will need to identify a valid underlying BBN model relating all the elements involved in change impact analysis.

One of the main conclusions on our section for change impact (Section 3.1.3) was that any development process can be seen as a chain of change decisions that can be linked to other decisions, that is:

**O5: Our change impact estimation model shall conceive the outcome of a software development as a result of a network of design decisions.**

Being decisions the key for taking actions that will modify the outcome of any development, our model conceives decisions are the key drivers of the change process. And our model shall be designed accordingly.

In section 3.4 we have seen the volatility of the decisions: the fact that, throughout the project development most decisions are changed. But decisions have causes and effects. Among these possible causes and effects we find the Software Lifetime Objects (SLOs) that we mentioned in Section 3.1: requirements, classes, documents, modules, etc. That is:

**O6: SLOs, among other elements, shall be linked to design decisions, either as inputs (causes for the decisions to be taken) or as outputs (effects of such decisions).**

Since we want the model to be valid for change impact analysis, we need to be able to identify the SLOs affected by a decision, or those SLOs that have some influence on taking that decision (that is, they are causes of the decision to be taken).

We need to determine new ways to link those elements related to design decisions (points in which a change is decided) required during project's development. By doing so, we contribute to determine the traceability of the SLOs.

In section 3.4 we have seen the importance of the rationale for design decisions. In general, each change is caused by a change decision, and this decision has an implicit rationale. Capturing the rationale provides advantages for any development since it provides multiple benefits: better design, team collaboration, support and maintenance. Moreover, change impact and design rationale are inter-related. The model must not only contain the decisions, but the rationale behind them; that is:

**O7: It shall be possible, by using the model, to capture the rationale for decisions**

Our model should enable users to explicit and declaratively express the main elements of the design rationale. It should also allow users to explicitly represent the relations among them.

When we discussed design rationale, we found the problem of data gathering as a key obstacle in design rationale. Development teams in general are reluctant to changes in the way they work, and they do not see a real advantage on capturing the rationale for a design, considering the additional time required to gather all this information. To minimize this problem, one of the key objectives will be to maximize its usability, in other terms:

**O8: The model shall be easy to use by developers**

For this purpose, the elements linked to the process of decision making for each decision shall be based on common and intuitive concepts for its users, and we shall create an ontology for design rationale that shall not be intrusive from the developer’s point of view. While the model will be based on BBNs in an “internal” model, we will try to find an “external” model (the one seen by the user) that can be easily understood.

In addition to all this objectives, in section 3.3 we identified both predictive and diagnostic reasoning as key advantages of BBNs, which leads us to the next objective.

**O9: The model shall take advantage of the predictive and diagnostic reasoning capabilities of BBNs.**

In section 3.1 we identified three steps for change impact analysis: identification of the change, tracing of the impact, and implementation of changes. We discussed the main techniques for change impact analysis: dependency analysis and traceability analysis. Although there are automated methods for both techniques, none of the automated methods provide estimations of the probabilities for SLOs to be involved in a change. Change impact analysis, at the end, is a process to be performed by humans (cannot rely on automated techniques). Also in sections 3.1 and 3.2 we identified that one of the problems for change impact estimations is that each change is performed under certain circumstances, and therefore this singularity of the change makes identifying the consequences of changes based on past, historical data very difficult. In other words:

**O10: Instead of being based on historical data, or automated techniques, the model shall be based on the result of change impact analysis made by users.**

Considering that for change impact, in the most general case, there is no precedent of an identical situation, the model will depend on the manual input from developers of the elements related to the rationale, as well as the evaluation of the probabilities associated to the effects that a potential decision will have. However, we will leave open the model to the automatically generated inputs based on some of the automated techniques discussed in Section 3.1.2

For this CIE model, the APES-CIE tool will serve as the test bench in which to validate the capabilities of the model, by applying the model to practical cases. In line with the objectives for the model, the tool has two main requirements associated;

- a. **The complexity of the BBN shall be hidden to the users.** Based on BBNs, our aim here is to be able to represent the causal relations of design decisions made during software development, in a way that can be easily understood by developers (in line with objective O8)
- b. **APES-CIE shall use the Eclipse IDE** as the environment for the tool: as in the case of SEURAT, our tool will be a plug-in for Eclipse. By doing so, we guarantee that developers can use from the IDE, therefore easing its adoption (in line with objective O8)

Note that these last requirements are exclusive of the tool, and do not apply to the model. The general characteristics of the APES-CIE tool are provided in Section 6.1. In the following section we will discuss the APES-CIE ontology.



## 5 THE APES-CIE MODEL

The APES-CIE model developed for change impact estimations was driven from several constraints, namely:

- **The need for simplicity:** A simple model is the key to facilitate the use of the model by the developers involved in a software project. Conklin and Yamekovic (Conklin, et al., 1991) state that, in order for decision rationale to be successfully adopted by the development teams, they need to have the perception that DR capture actually has a short-term payoff and that the cost of DR capture has been minimized. To make it usable, we have adopted those elements from existing notations that we considered representative, eliminating those that we thought that could be discarded. In that sense, APES-CIE model provides a semi-formal approach for Design Rationale that is simplified w.r.t. previous models such as IBIS or DRL.
- **The need for doing the system non-intrusive, or as less intrusive as possible.** As Burge and Brown state (Burge, et al., 2002) “*The more intrusive the capture process, the more designer resistance will be encountered.*”
- **The search for a notation in line with existing methods and standards in the aerospace sector.** Our study was oriented towards the use on the aerospace field, and therefore it had to be in line with existing methods and techniques already in use in the software development standards for the aerospace sector, in particular software development standards for the European space market ( ECSS-E-40, Part 1B 2003).
- **The need to adapt the notation for the use of BBNs.** Since BBNs consider exclusively cause and effect relationships among its nodes (Kjærulff, et al., 2005), it was necessary to perform a tailoring of existing DR notations for its use. A model of DR was created, and it had an underlying BBN model that serves for predictions.

## **5.1 APES-CIE MODEL: OVERVIEW**

The philosophy of the design rationale for change impact in our system conceives the development as a series of **trade-offs**. Each trade-off corresponds to the way designers respond to a specific question that requires **design decisions** to be made. Trade-offs can be, in fact, considered equivalent to the concept of “issues” from IBIS.

For instance, suppose that, at a given point in the development, developing a new feature to fulfill a requirement is needed. This feature can be implemented in many different, mutually exclusive, ways. For instance, it can be made by reusing existing software or doing it from scratch, or it can involve a chain of decisions that are not necessarily exclusive (i.e. a series of design decisions).

As in the case of the AREL model, each decision has an “status” attribute, that is a discrete variable that can have two possible values: either VALID or INVALID, that indicates that this decision has been taken (=VALID) or discarded (=INVALID). Since the design decision has an uncertainty associated to it, this attribute of the decision corresponds to a BBN variable.

But decisions have both causes that influence them (either positively or negatively), that are inputs to the decision, as well as outputs, in the form of consequences.

In the AREL model, both inputs and outputs were elements of the design (Design Elements). In the APES-CIE system, we have extended this concept to “decision element”. In APES-CIE, a “decision element” is any element that is involved in a decision, either as an input (cause) or as an output (consequence). Contrary to the AREL model, it is not restricted to design elements. Each decision element, in turn, can also have an equivalent discrete variable (“status”) that can have two different values.

The change impact evaluation is therefore conceived as a set of Trade-offs, each consisting of a set of decisions to be made. These decisions can be mutually exclusive (i.e. a decision made implies to discard the remaining decisions of that trade-off) or not (a trade-off involves various decisions that are independent from each other). Causes and its consequences for decisions are always “decision elements”. The set of possible decision elements is restricted to five different types that correspond to concepts traditionally used during the system’s

design in the aerospace field: requirements, assumptions, goals, environmental issues and design elements. Figure 5-1 shown in the next page) is a class diagram that describes our model.

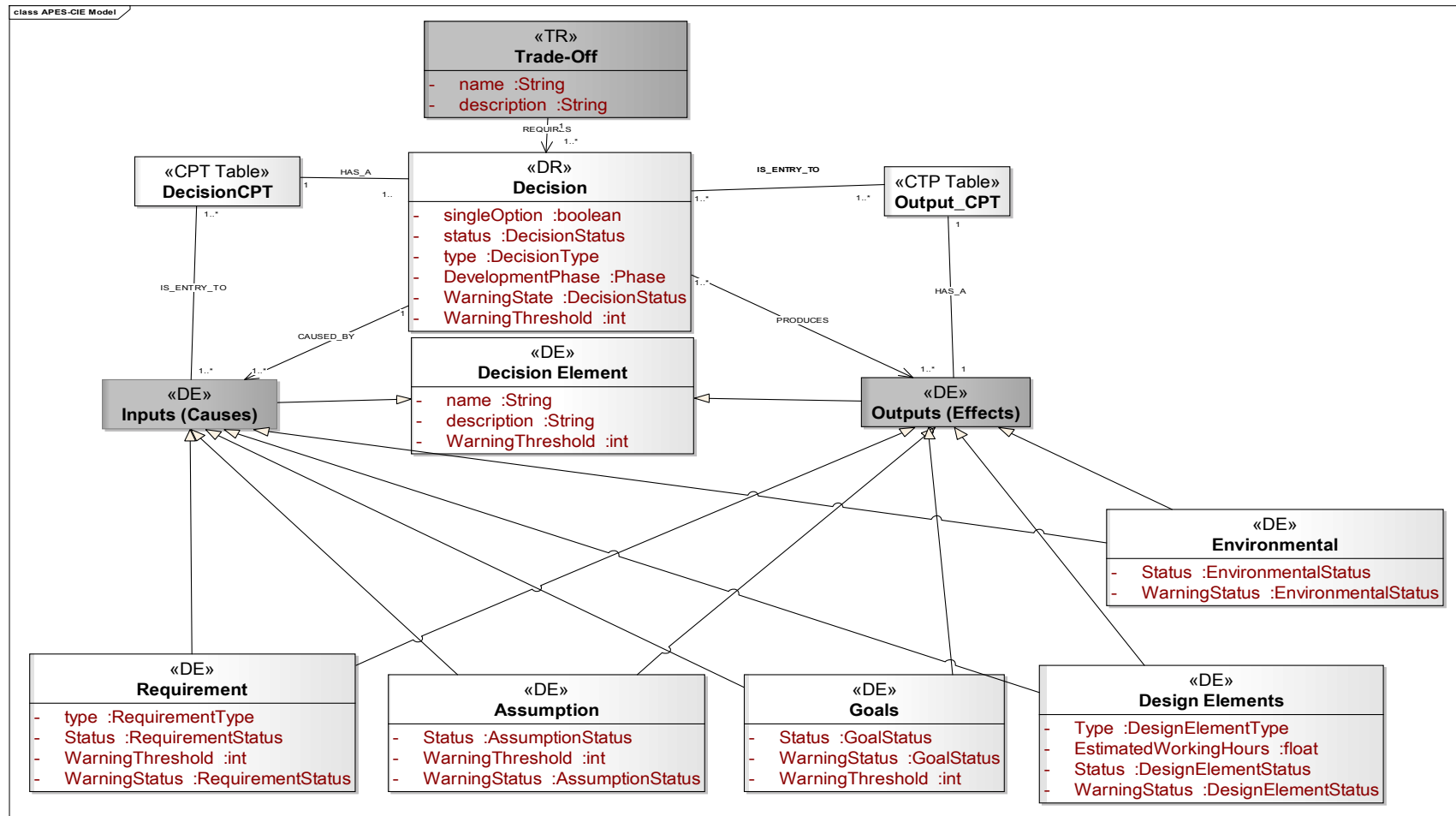


Figure 5-1: The CIE Model

As shown in Figure 5-1 causes and consequences inherit from a common class: “decision element”. This “decision element” has a set of common attributes (name and description). In addition, a set of classes inherit both from “inputs” and “outputs”: these are the different types of *decision elements*, namely: *requirements*, *assumptions*, *goals*, *design elements*, and *environmental*. *Decision elements* can play simultaneously the role of input to a decision (*cause*) and output of another, different decision (*consequence*).

Each project has a set of trade-offs associated to it, with its corresponding decisions and decision elements that form a single BBN. Variables of this BBN are the status of the decisions (VALID, INVALID) and the statuses of the decision elements. As we mentioned earlier, status for decision elements are discrete variables. They can have the following values:

- For requirements, status can be either STABLE (meaning the requirement is part of the baseline) or VOLATILE (meaning the requirement is not yet part of the baseline)
- Design elements can be STABLE or VOLATILE (stable meaning that they will take part of the final design, VOLATILE meaning that it is yet to be decided)
- Assumptions can be TRUE or FALSE, TRUE indicates that we know for sure the assumption is correct.
- Goals can be ACHIEVED or NOT ACHIEVED.

Each decision has an associated CPT based on the causes related to it (this is represented by the *CPT\_Decision* class in Figure 5-1). In turn, each decision element that is an output has a set of decisions for which it is a consequence, and therefore has an associated CPT whose entries are those decisions for which it is a consequence.

The following sections describe in detail each element of this model.

## **5.2 TRADE-OFFS**

Trade-offs can be thought of as the “issues” as defined in IBIS. They are “questions to be answered”. As mentioned before, trade-offs are used in two different cases:

- Either to describe a balance of factors all of which are not attainable at the same time, or
- A question that might imply a set of decisions, related to this trade-off.

Trade-offs serve as roots for a set of different decisions, that can be single choices or not. Trade-offs in space development are traditionally contained in the Design Justification File, together with decisions (ECSS-E-40, 2003). Trade-offs can be considered a common entry point for a question that needs a set of decisions to be made during development. So, for instance, the question “What type of database should we use for this project?” can have many different options (mySql, ORACLE, indexed files). Each trade-off has therefore a set of decisions associated to it. For simplicity, in our model, each decision belongs to a single trade-off.

A trade-off is identified in the model by its name (that shall be unique). In addition, trade-offs have a complementary description that is a text field.

## **5.3 DECISIONS**

These are design decisions that are to be made during the development process. As commented before, each decision in the model has its own “inputs” (possible causes - elements to be considered for the decision to be made) and “outputs” (all those consequences of the decision when it is made, design elements, requirements that are a consequence of the decision, goals that are accomplished or not if the decision is made, etc.). Decisions are always a child of a given trade-off, for which they are supposed to provide a (possibly partial) solution.

An internal relationship is established between tradeoffs and decisions. Decisions have the following attributes:

- Name: unique name that identifies the decisions

- Description: a text field to provide additional indications
- Status: can have two possible values: VALID or INVALID. This is a variable that belongs to the underlying BBN. Once its value is known, it can be set by the user either to VALID or INVALID. VALID will be represented by a 100% probability, meanwhile INVALID will be represented by a 0% probability. If the decision is yet to be taken, the system assigns automatically a probability for the status to be VALID, based on the propagation of probabilities of the underlying BBN.
- Type: single choice or multiple choice. Single choice indicates that this decision is identified as a mutually exclusive alternative for the trade-off. In this case they correspond to the concept of alternative (Lee, 1989) to the trade-off. Marking a decision as single option has an effect on the status of the remaining decisions of type “single choice” of the same trade-off. In fact, if the status value of one of them is set to VALID, the status of all the remaining decisions is set to INVALID.
- Development phase: this value indicates the phase of the development in which the decision is made (requirements, design, implementation, testing, and maintenance).
- Warning state: This is a flag that can be set by the user to indicate if the user has to be warned whenever the value for the status has reached a given status.
- Warning threshold: this is an integer in the range 0 to 100 that indicates the threshold to warn the user when the associated probability to its status has reached a given limit.

## **5.4 DECISION ELEMENTS**

Decision elements can act simultaneously as causes and consequences of a decision. Each decision element is identified by a name. Design elements can have also a textual description. In our model, clauses and consequences are categorized and are common elements that are present in any development, and they can be:

- Requirements
- Assumptions

- Design Elements
- Environmental issues
- Goals

All decisional elements share the following attributes:

- Name: a unique string that identifies univocally the decision element.
- Description: a text field that provides additional information.
- Status: this is a field that is filled only if the status of the decision element is known.

All decision elements can have only two possible statuses, so that:

- For Requirements, Environmental, and Design Elements, possible statuses are VOLATILE or STABLE
- For Assumptions, possible statuses are TRUE or FALSE
- For Goals, possible statuses are ACHIEVED or UNACHIEVED.

Therefore, for all decision elements, *status* is a discrete variable with two possible values. This discrete variable has an associated probability that is computed by the system based on the underlying Bayesian Belief Network. However, users can set the value of the status as a finding when the value of the variable is known.

Warning status and warning threshold are variables used by the user to request the system to raise a warning if the probability associated to the status of the element is below a given threshold. For instance, the user might consider important to be warned by the system if the probability for a particular design element to be STABLE (actually: to be part of the development) is lower than 50%. In such case, the user will set warning status to STABLE and warning threshold to 50%, and the system will raise an alarm when the probability is lower than 50%.

#### 5.4.1 REQUIREMENTS

In the aerospace field, design decisions are in the majority of the cases driven by the requirements that are contained in the so-called Software Requirements Document (SRD). Different requirements can be added or removed in successive versions of the Specification



Requirement Document, and also during most of the design phase requirements are subject to change. Requirement status (that, as already mentioned, can have two discrete values VALID or INVALID) is therefore involved in the decisions, either as input to a decision (meaning that the decision is taken due to the existence of a requirement) or as an output to the decision (meaning that the decision causes an extra requirement to be added to the system).

Requirements correspond directly with the traditional definition: “a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user” (ECSS-E-40, 2003). These are categorized in such document as requirements for: flight software, software reuse, man-machine interfaces, and real-time software. They can be either functional requirements or non-functional requirements. In sum, requirements have the following attributes:

- **Name and Description, warning status and warning threshold:** inherited from “Decision element”
- **Type:** indicating the requirement type: either Functional, non-functional
- **Requirement status** can have the same values as in (Tang, et al., 2006), either “VOLATILE” or “STABLE”. “VOLATILE” means that the requirement will not be part of the baseline; meanwhile stable means that it will be part of the baseline. In our model, requirements can be nested, meaning that a single requirement can have a set of associated sub-requirements.

#### 5.4.2 ASSUMPTIONS

Assumptions are “assessments that are considered as true” by the user. They can be considered as arguments that have influence on the decision. In fact, they correspond to the DRL concept of “claim”: “a claim asserted as relevant to decision making” (Lee, 1989). The reason that leads us to use a different term (“assumption” instead of “claim”) is the fact that this is the traditional term used in the aerospace sector. Assumptions are traditionally included in Software Requirement Documents and in the Software Detailed Design, as well as the Design Justification File (a document that contains the design decisions made during

the design process in space projects), all of them are documents associated to the ECSS Standards (ECSS-E-40, 2003). In our model, assumption status has two possible discrete values, either TRUE (the assumption is known to be correct) or FALSE (the assumption is known to be incorrect). As in the case of any decision element, the status underlying probabilities are computed automatically by the system, if they are not set to any of these two values by the user.

Therefore, assumptions have the following attributes:

- **Name and description, warning status and warning threshold:** inherited from “Decision element”
- **Assumption status:** either TRUE or FALSE

### **5.4.3 DESIGN ELEMENTS**

Design Elements are “basic components or building blocks in a design” (IEEE Glossary, 1990). They correspond to “Artifacts” as defined in the Unified Process (Kruchten, 2000). In general, they can be considered artifacts generated during the design process that are to be built during the lifecycle. There is a strong variability in the degree of size of complexity. So, for instance, a design element can be a subsystem, a class, a module, a function, a package, a document, etc. In the CIE, design elements have an important attribute that is the effort devoted to generate such artifact. This is used to allow the user to estimate the effort that is to be devoted to a particular task. As in the case of requirements, they have a “status” discrete variable whose values can be either STABLE or VOLATILE. STABLE means that they will definitely take part in the development, while VOLATILE means they will not be part of the development. Design elements can have child design elements associated to them. More importantly, design elements have their estimation on the working hours that are needed for their achievement. Design elements have the following attributes:

- **Name and description, warning status and warning threshold:** inherited from “Decision element” Type:
- **Status:** (already commented) STABLE or VOLATILE

- **Type:** it can be one of the following: class, package, subsystem, diagram, module, functional item, use case or other.
- **Estimated working hours:** this is the estimation on the working hours it will take the generation of the corresponding design element.

#### **5.4.4 GOALS**

The term “goal” is so generic that no definition of this term is found in the IEEE glossary (IEEE Glossary, 1990), nor in the ECSS SW Standards (ECSS-E-40, 2003). Instead, the term “goal” is a common term used in the Decision Representation Language (Shum, 1991) (Lee, 1989). In this last document, goals are defined as “a condition that one wants to satisfy in making a decision”. In our model, goals are objectives that we want to be achieved as a consequence of the decision made, and they can be restricted to the project, or they can be internal goals of the organization, that do not need to be declared explicitly. They can represent the fulfilment of a whole set of requirements, or they can represent “shadow” requirements: internal restrictions that we need to match, but not necessarily agreed with the customer (as requirements are). For instance, a goal can be “allow the development team for this project to be available for the next project before the end of the year” (shadow requirement) or it can represent a set of requirements, as “to have a user-friendly interface” (requirement). Goals can have related sub-goals. The probability associated to them indicates whether they are to be achieved or not. The attributes for a goal are the following:

- **Name and description, warning status and warning threshold:** inherited from “Decision element” Type:
- **Status:** ACHIEVED or UNACHIEVED.

#### **5.4.5 ENVIRONMENTAL**

Environmental elements refer to general issues not necessarily set as requirements, but that will need to be taken into account for a decision: environmental conditions in which the SW will have to run; organizational aspects; the team that we use; the availability of a given

asset (such as a simulator required for testing); or organizational aspects. These issues belong to the environment in which the project is developed, but are not part of the nucleus of the development as such. This type of element does not exist in the DRL model (Lee, 1989) nor in the ECSS Standards, which only mentions environmental conditions for the SW to run (ECSS-E-40, 2003). However, we have added them because we consider them part of the decision elements. The attributes for an environmental item are the following:

- **Name and description, warning status and warning threshold:** inherited from “Decision element”Type:
- **Status:** either STABLE or VOLATILE

## **6 THE APES CIE SYSTEM**

To demonstrate the usefulness and advantages of the model exposed in the previous section, the so-called APES-CIE tool was developed to allow us the definition of change decisions affecting SLO,s in real projects, and to ease the decision-making process under uncertainty.

To maximize the usability of the APES-CIE, the system, developed for ESTEC in the framework of the CCI contract, this tool was conceived as an extension of an existing IDE (Integrated Desktop Environment). From the IDE it is possible to link easily existing DLOs (source modules, code classes, requirements) to the design changes created from the system. The links between causes (the DLOs that produce the need for a change) and the effects of the decision (the DLOs produced as a result of the decision) was to be reflected internally in a Directed Acyclic Graph (DAG) along with “a priori” distribution probabilities for the root-nodes and CPTs for the non-root nodes. But these DAGs and CPTs would be hidden from the user, so that he or she does not have to burden with the details of the generated BBN network.

The IDE chosen for the implementation was Eclipse. The reason to develop APES-CIE within the Eclipse framework was the following:

- a. Nowadays, Eclipse has a widespread use as a generic IDE in multiple developments.
- b. Eclipse provides the possibility to develop in relatively easy way extensions to its functionalities in the form of plugins (Clayberg, et al., 2008).
- c. There is a current trend for UML tools to be integrated into the Eclipse’s IDE. Taking this trend into account, the advantage of this approach is that it eases the traceability from the code and design elements to the estimation model elements, since both models are generated in the same development environment.
- d. In addition, Eclipse eases the automatic generation of documentation, allowing developers and managers to work on the design and the BBN from the very beginning.

- e. Also the development of the APES tool as an Eclipse’s plug-in makes it independent of any UML design tool being used. The same design and coding IDE (Eclipse) shall therefore act as the Man-machine interface (MMI) for adding and modifying the corresponding DAGs.

This extension to Eclipse, developed “ad-hoc” for the APES Change Impact functionality allows a user to insert a cascade of change decisions that, in a similar way as in the AREL model, links these decisions (design rationale) to its causes and effects (design elements), being causes and effects SLOs of the development.

Internally, the APES-CIE tool creates the corresponding DAGs automatically. This reflects a representation of the mental process of decision making, based on probabilistic estimates from causes to effects.

For the propagation of probability to be performed, APES requires a BBN Engine, an external library. In our case, the external engine being used was Netica (Netica, 2008). Netica provides an API so that Bayesian propagation is performed by the system via calls to Netica’s software.

By embedding APES into an IDE, the system allows users to link elements from many different workflows (requirements, design, testing) of the design, and therefore it matches one of the key requirements that were needed: *the possibility to perform change estimations throughout the whole lifecycle, as well as to refine them as soon as the certainty increases*. Also the system can easily be enhanced taking advantage of future plug-ins developed for Eclipse.

Contrary to the other subset of the APES tool (APES-PE), APES-CIE is not based at all in historical data, since, as we stated previously, for decisions to be taken when evaluating a change we need to address situations that are, in the immense majority of the cases, singular and based on the characteristics of the project and its situation in the moment in which the decision has to be considered

All these capabilities provide a global greater flexibility of the tool with respect to future changes and improvements in BBN technologies.

## 6.1 GENERAL CHARACTERISTICS OF THE APES-CIE SYSTEM

The main requirements for the APES CIE system, that were its key drivers, are the following:

- Formalization of the DAG networks and elements. The tool uses intuitive, user-friendly notation. Elements of the graphics will correspond to the equivalent BBN elements (variables, states, causal relationships) so that it will be easy to familiarize users with the tool, even for users without previous knowledge of BBN technology.
- Clean and concise interfaces to the BBN API. The primitives for the communication to the BBN API are clearly identified, in order to make its future porting to a different BBN engine possible.
- Extensive configurability. If a single user might find a use for an item to be configurable, then that item will be configurable.
- Hierarchical configurability. Configuration items are configured at various levels, the one which applies being the most specific one. For example, the system may be globally configured not to show by default the corresponding CPTs, but a certain view item for a particular DAG may be individually configured to show the values.
- Flexible propagation. The user is able to select the moment at which the propagation of the values is performed.
- *"Everything is editable"* paradigm. It is possible from the MMI to modify all the values that have any impact in the computation of the Bayesian Network outcomes. For those values that need any additional textual value, it is possible for the user to introduce and modify it.
- Flexibility on the selection of displayed properties. The GUI shows the name, value and description for any parameter.
- Clear separation of computed and manually added values: it is possible from the GUI to identify, without any additional action, which values are computed and which values have been manually introduced. The MMI uses different colours for this.

- Mapping between DAGs and Eclipse is straightforward. There is a one-to-one relationship between Eclipse Projects and DAGs. Each project contains a single DAG that has all its network of decisions.
- Portability. The system, as Eclipse, is able to work in various operating systems, and has been tested as a minimum in Linux and Windows.
- Commercial off-the-shelf (COTS) software used by the tool is reduced to a minimum. The system shall not use any COTS product, except those required by the BBN engine (Netica) and Eclipse.

## **6.2 ASSUMPTIONS AND DEPENDENCIES**

During the design and implementation of the system the following quality attributes were prioritized:

- **Learnability and intuitiveness:** the interface is designed to allow a smooth learning curve.
- **Efficiency:** The tool is designed so that it does not consume many resources, in terms of CPU, RAM memory and disk.
- **Error logging:** Errors occurred during the use of the tool are logged. It is possible to perform a diagnosis of problems occurred using the tool by analysing its logs.
- **Simplicity:** Tool has been designed keeping in mind simplicity from both the usage and the software design point of view.
- **Defensive programming w.r.t. bad inputs:** The tool does not allow users to introduce data that could cause a malfunctioning of the tool, or that might cause it to provide an unexpected behaviour.
- **Forgiveness:** When introducing wrong values, and whenever possible, users are informed of the reasons why their input values were rejected.



- **Feedback:** It is possible for the user to modify the variables that affect the computation.
- **CPT data** (probabilistic estimates) will be input by the user, although this process will be made for the user as easy and intuitive as possible
- **Automatic gathering:** whenever possible, the tool will perform automatic propagation of all elements derived from the rationale, as described by the user.

### **6.3 OPERATING ENVIRONMENT**

The operating environment of the tool stems from the previous discussion of requirements, and from what we learned in Chapters 1 and 2 of this thesis. The tool runs as a plug-in within Eclipse. The tool is oriented towards its use from a desktop PC or a laptop allowing users change impact estimations based on change impact analysis elements defined by the user. It is the user that performs the change impact analysis, and manually defines the variables (SLOs) involved in the BBN, as well as its CPTs, being the tool in charge of the impact estimations for the given analysis.

A main goal of the tool therefore is to provide support for the elements that perform the change impact analysis depicted in Chapter 1. In this chapter we saw that during the whole lifecycle of a given project, design is subject to changes. These changes could eventually cause a tremendous effect on the evolution of the project that has been identified as “the ripple effect”. And each change had a set of SLOs associated, both as inputs and outputs.

In Chapter 2 we identified “design changes” or more generally “change” as the consequences of “design decisions”, being each “design decision” triggered by a design rationale. In this rationale we identified additional elements (such as “goals”, “alternatives”), in most cases not traditionally identified and written as part of the design, that played as a key role in the decision.

Moreover, we identified the need in SW projects to work under uncertainty, and the need to perform “what-if” analysis, that could help in the process of decision w.r.t different options.

These estimations on change impact have to be available from the very beginning of the project till the end of it.

From the point of view of change impact analysis it is of paramount importance to identify the set of SLOs involved in each decision, as well as the rationale for this. Therefore, it is necessary that all stakeholders participate in the development of the model; for each project, a network of changes is built progressively and each new change in the design implies adding new variables (SLOs that are causes, SLOs as outputs, design decisions and their associated rationale).

The fact that all developers participate in the design of the network implies also the following:

- There must be a single view for all users.
- This view should change dynamically with all the changes.
- This view should be easy to use, since all stakeholders will be involved.

For that purpose, we selected a *single-tree view* to show all the elements involved in the change impact analysis, so that *users do not have to deal with the complexity of graphs*. In other words, standard users should not have to deal with the BBN concepts: there must be a DAG behind the interface, but this DAG should be hidden to the users. BBN notions and vocabulary are avoided here. For instance, based on the information provided by the users in the tree view, the tool uses the Netica API to build internally a BBN, and performs automatically the compilation of the DAG network (a process necessary once the DAG is complete), without burdening the user with the low-level details of BBN estimations. The tool performs the compilation automatically as soon as there are sufficient elements as to perform it.

In Chapter 2 we saw the importance of the model behind the design rationale tools. By model, we mean a symbolic representation of the different elements involved in the estimation of the impact of a change.

Considering the need for “what-if” analysis, we included in the model the concept of “trade-offs”: as commented in Chapter 5.2, trade-offs correspond to the “issues” concept of DRL. In the case of the tool, trade-offs are used in two different cases:

- Either to describe a balance of factors all of which are not attainable at the same time, or
- A question that might imply a set of decisions, related to this trade-off.

This particular type of estimations has a special relationship with the Design Justification File (DJF), a document generated in ECSS standards that contains the design rationale. According to the ECSS standards document, the DJF shall contain “all the all significant trade-offs, feasibility analyses, make-or-buy decisions and supporting technical assessments”. It is conceived as an aid to justify the decisions taken, as well as an estimation tool for impact on software items. Considering that the relation between changed elements and design decisions, as well as its root causes, is supported by the tool, all the information that normally goes to the DJF can be handled by the tool. This also implies the existence of an internal database, which is hidden to the user.

To illustrate the capabilities of the tool, we are going to use an example. As we commented before, in APES-CIE, the main element for operating with change impact elements is a *tree view* of the rationale. There is a single tree view per project. This is the so-called “APES Change Impact View”, that can be seen from the list of views available in Eclipse, as shown in Figure 6-1.

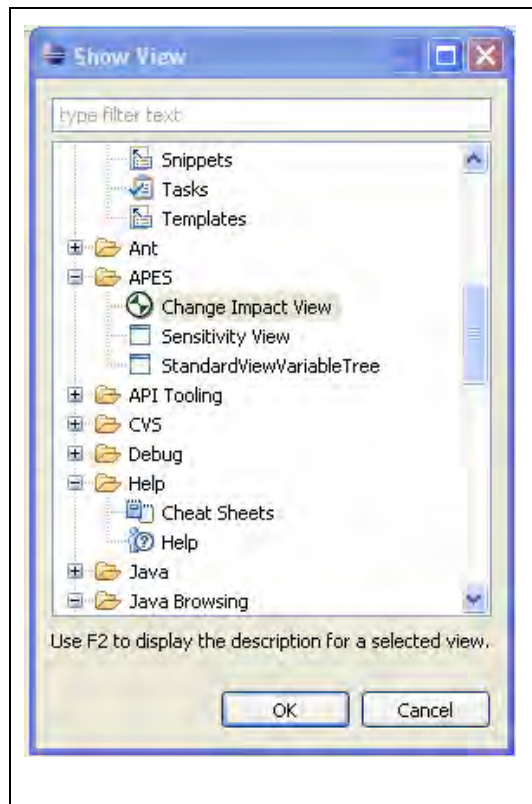


Figure 6-1: Selecting the Change Impact View in the Show View Dialog in Eclipse.

Eclipse is a multi-project tool. Therefore, the Change Impact View is to be selected for a particular project. Once the user selects the "Change Impact View" to be opened, a dialog is shown on the screen asking for the project for which the tree view will be shown, as can be seen in the following figure:

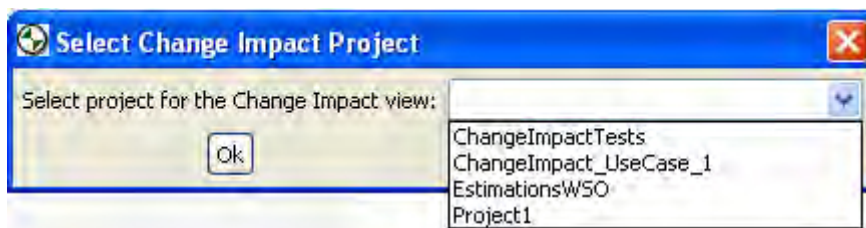


Figure 6-2: Selecting a project for the change impact view.

The list of projects corresponds to the list of projects that the user has in the Eclipse workspace (those that are shown in the Package explorer window of the Eclipse instance that is running). If there were no projects in the workspace, it would be not possible to open the change impact view for a project. It is understood that the change impact is to be opened for an existing project. Once a project is selected, the corresponding change impact view can be seen in the Eclipse

It is from this view that the change impact relative to all decisions for this project can be evaluated. The change impact view will contain all the trade-offs, decisions, causes for such decisions as well as consequences related to the project, together with its corresponding probabilities.

#### **6.4 EXAMPLE. DEVELOPMENT OF A MEMORY TOOL FOR A MISSION CONTROL CENTRE**

In this example, a project manager faces a design decisions with many alternatives: there is a need to extend the functionality of an existing ground control system to display the memory map of the spacecraft controlled from ground.

- Since this spacecraft is a new development, it has a set of requirements relative to the memory management that are to be accomplished. The “memory management” tool to be developed is part of the software to be delivered for the control centre of this particular spacecraft, and the control center software itself is a much larger development, whose interface is to be done using the Java programming language.
- For the memory management tool, there is a previous development, based on Microsoft Access that performed a very similar task for a previous spacecraft. So the possibility to develop this application based on an adaptation of the existing Microsoft Access tool is a serious alternative to be considered. But this previous development is much simpler, and would need to be customized and extended. Moreover, it is not clear whether the former team that developed the MsAccess tool will be available.

- The company has also available a set of Tcl-Tk programmers that have performed similar developments using the Tcl\_tk language. Using Tcl-tk code snippets from previous developments, it would be relatively easy to build the memory management tool for the spacecraft. The advantage with respect to the previous option is that it is considered more re-usable for the future; meanwhile, although costs would be higher than in the previous option, can be maintained in the same rough order of magnitude.
- Finally most ambitious alternative would be the development of the memory management tool using the Java Language. This would simplify the final delivery (avoiding having to add an extra programming language to the list of languages used in the SW to be delivered), provide better scalability and possibly better performances. But this seems to be the most expensive (in terms of man-hours) alternative, and also is subject to the availability of Java programmers during the construction of the SW.

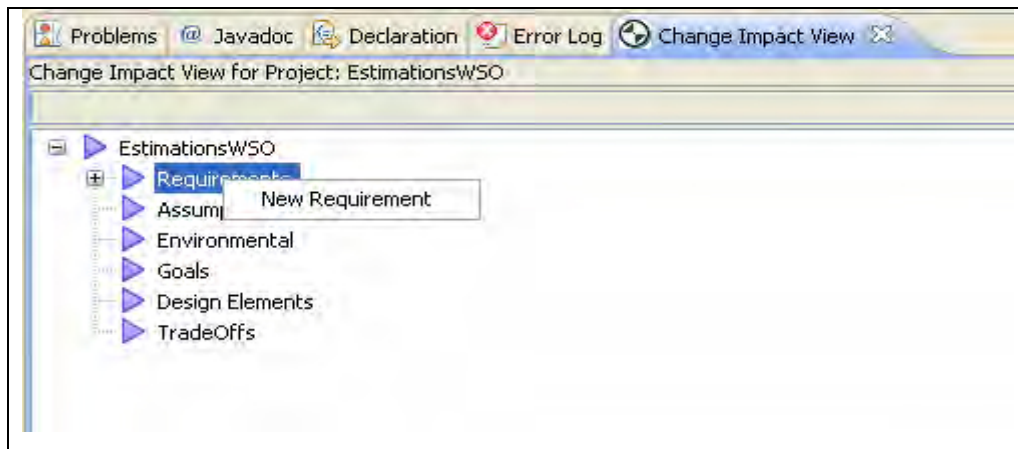
There is no requirement from the final customer w.r.t. which language shall be used for the memory tool. As soon as the system requirements are met, the company can choose the language for the implementation of the memory mapping tool.

The operating environment consists of four different steps, that are explained hereafter:

#### **6.4.1 STEP 1: ADDING DECISION ELEMENTS**

##### **6.4.1.1 STEP 1.1 ADDING REQUIREMENTS**

Decision elements are the first elements that are to be input by the user. These decision elements are the ones that will trigger the decisions. The first and most important elements to be added are the *system requirements*. In the “change Impact View” of Eclipse, there are four main entries: the first four are to add decisional elements (requirements, assumptions, environmental and goals), meanwhile the last one (trade-offs) will be used to add decisions. In the figure shown below we can see how the user can select the “Requirements” branch and add a requirement from there.



**Figure 6-3: First steps: adding a requirement to an empty change impact tree**

It is not necessary for a user to add all the requirements for the system. Users should only add those requirements that are considered to be either causes for decisions to be taken, or consequences of such decisions. However, in the case of requirements, it is particularly important to check that all requirements are to be met.

In addition, requirements can be added as "sub-requirements" of other particular requirements. Figure 5-4 shows the aspect that the requirements branch of the model tree will have once a whole set of requirements have been added by the user.



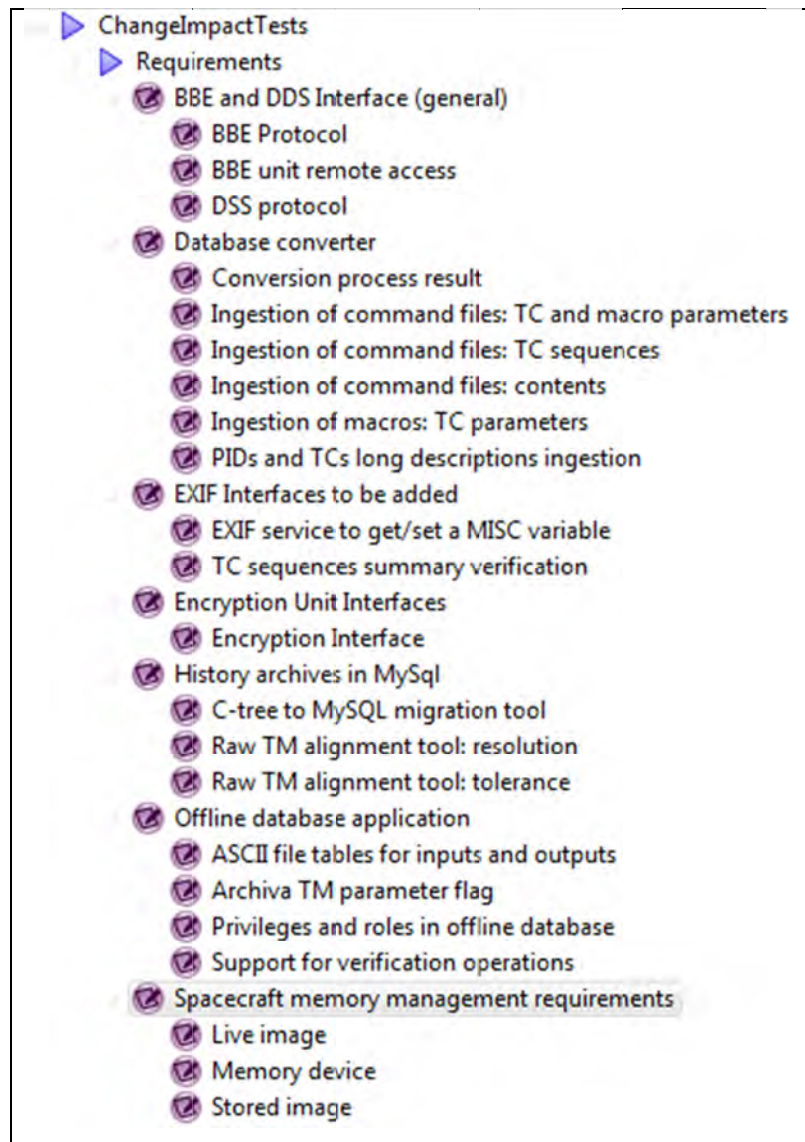
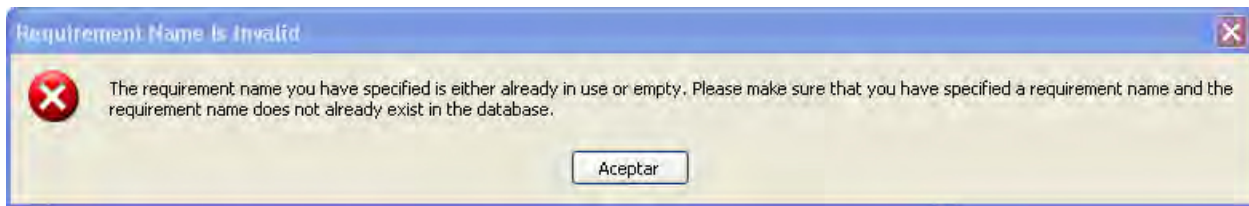


Figure 6-4: Partial view of the requirements in the Change Impact Tree

In Figure 6-4 only those requirements relative to the memory management tool have been added. Users can also add requirements to the change impact view later on, if they are considered relevant for the decisions to be made.

Note that, when saving a requirement, requirements names are used as a single identifier, for this element: the system will not allow the name to collide with existing elements. In such case the system shows a warning using a window like the one shown in Figure 5 3.





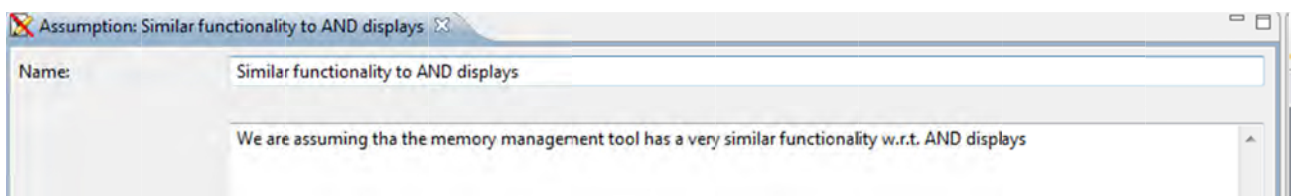
**Figure 6-5: Adding a requirement that already exists**

This applies not only for requirements, but also for any element to be added to the change impact (assumption, environmental, goal, tradeoff, decision). That is, change impact elements are identified by their name that should be unique for a given type of element; the system does not allow users to input two requirements with the same name, or change impact elements without a name, since they will be used to build the internal BBN.

#### **6.4.1.2 STEP 1.2 ADDING ASSUMPTIONS**

The next decision elements to be added are assumptions. In a similar way to requirements, assumptions can be added to the change impact tree by selecting the "Assumptions" branch (See Figure 6-3), and clicking with the right button of the mouse.

Once the user has clicked on the "New Assumption" option, a new tab will show up on the Eclipse's editor, showing the data to be edited of the new assumption.



**Figure 6-6: Adding an assumption**

As we did previously with requirements, we will add exclusively those assumptions that we think will affect future decisions. In this particular example, one of the most important assumptions is that we are assuming a very high degree of similarity of the Alphanumeric Displays of the spacecraft's memory (AND) with respect to previous developments. Should this assumption not be valid, it would affect seriously the effort that needs to be invested in adapting existing software assets. That is the reason why is should be taken into account.

Assumptions normally are considered inputs to decisions (as it is in this case), but shall also be considered outputs to decisions, since a given decision can invalidate an assumption, or reinforce its validity.

### 6.4.1.3 STEP 1.3: ADDING ENVIRONMENTAL ISSUES

In Chapter 5.4.5 we defined “environmental elements” as those that are not considered requirements, but that will need to be taken into account for a decision: environmental conditions in which the SW will have to run; organizational aspects; the team that we use; the availability of a given asset (such as a simulator required for testing); or organizational aspects. In our example, we have four important environmental issues to be taken into account, that are shown in

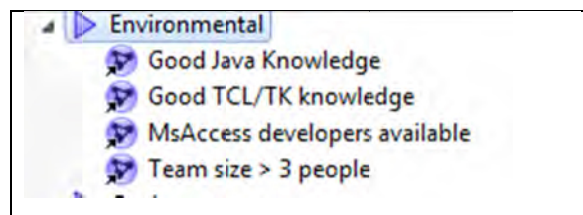


Figure 6-7: Environmental issues of the example

These environmental issues refer all to the characteristics of the team that will perform the work, that is: whether they will have *Good Java Knowledge* (that will have an impact if the development is to be done in Java), whether they will have *good Tcl/Tk knowledge* (also will have an impact, if the development is to be done in Tcl/Tk), whether *we will have more than three programmers for the development team* (which will be very important if the development had to be done from scratch), and whether *the former developers of the existing Microsoft Access will be available*.

Other environmental issues could refer, for instance, to problems with the project plan, unavailability of SW assets required, etc... As in the case of the assumptions, environmental issues are traditionally *causes* for decisions, however our model allows them to be *consequences* of decisions as well.

#### 6.4.1.4 STEP 1.4 ADDING GOALS

According to the definition we provided in Section 5.4.4 goals are "high level objectives that are important to be achieved". In this example, in the long term it is important for the company to have a system that can be reused for further developments, and this can be considered an internal goal. In addition, it is also an objective to finish the testing of the tool before the end of the year, moment in which the programmers associated will be also involved in another project. This requirement is not dictated by the project's schedule, but is an internal goal for the company that will ease the department to fulfill its objectives in the long term.

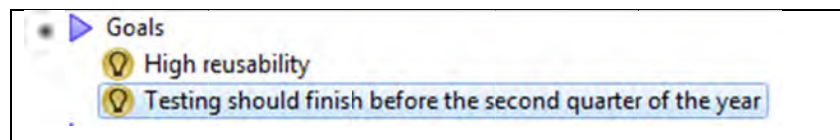


Figure 6-8: Goals for the example

Goals are typically consequences of decisions, and as such are outputs to decisions, but in some cases, they can also play the role of "inputs" (they can trigger a decision as well, being the cause of it).

#### 6.4.1.5 STEP 1.5 ADDING DESIGN ELEMENTS

The last type of decision elements to be added is "design elements": these are among others, classes, modules, diagrams, documents, or packages to be developed. In the most general case, they are outputs to decisions, but can also be inputs, as in the AREL model. This corresponds to the traditional concept of "design element" from the AREL model.

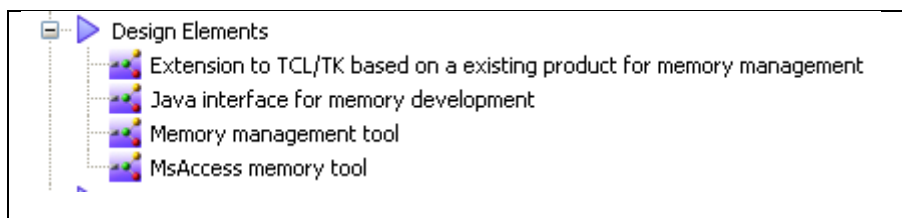


Figure 6-9: Example of design elements

In the example, we are considering as possible outcomes of our design decisions three different possibilities: either a SW tool based on an existing (and limited) Microsoft access development, we perform an extension to Tcl/Tk based on a existing product for memory

management, or we develop from scratch a Java GUI for the memory management. These are all the possible three components that will be part of the design.

Design Elements have a very important attribute to be filled: the working hours that are estimated for the development of the particular item. This datum will be used by the tool to compute the costs associated to a decision, as we will see later.

## **6.4.2 STEP 2: COMPLETING THE TREE: ADDING TRADE-OFFS AND DECISIONS**

### **6.4.2.1 STEP2.1 ADDING TRADE-OFFS**

The last root entry that we saw in Figure 6-3 was “Trade-offs”: that is the point from which trade-offs and their corresponding decisions are to be generated.

Trade-offs are a common root for a set of decisions associated to them. Until now, we have added all the decision elements that will take part of our decisions, but, since they are not part of the underlying BBN yet, (either as a cause of any decision or an effect of any decision), the probabilities for the “status” attributes described in the elements of the model are not being computed. In other words, no underlying BBN exists until the user adds trade-offs and their corresponding decisions. For this purpose, the corresponding decision needs to be added. And each decision is related exclusively to a single trade-off. Therefore, from the “Trade-offs” branch, we add a new trade-off which is “type of memory tool to be used” as shown in Figure 5-7. Once the user has clicked on the “New Trade-off” option, a new tab will show up on the Eclipse’s editor, showing the data to be edited of the new Trade-off item. In Figure 6-10 we have filled the data for our particular trade-off item:

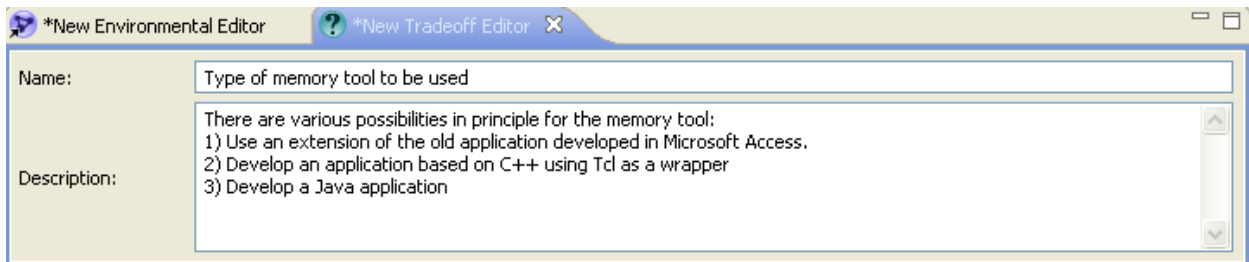


Figure 6-10: Trade-off for our example

This trade-off “*Type of memory tool to be used*” will have three different design decisions associated: either to what type of design element will have to be developed: either to develop a new Java application for memory management, to *use a TCL/TK existing editor*, or to *reuse the MsAccess application*. This is explained in the next section.

#### 6.4.2.2 STEP 2.2: ADDING DECISIONS, CAUSES AND EFFECTS

In previous sections of this thesis, we mentioned that the main idea for the change impact estimations was to tackle the “ripple effect” (i.e., what are the consequences of modifying this component, which part of the design gets affected by a given change). For this, the notion of *design decision* is central.

Design decisions are linked to the Trade-off that they are associated. In our example, we create a “*Develop new Java Application*” decision, hanging from “*Type of memory tool to be developed*”.

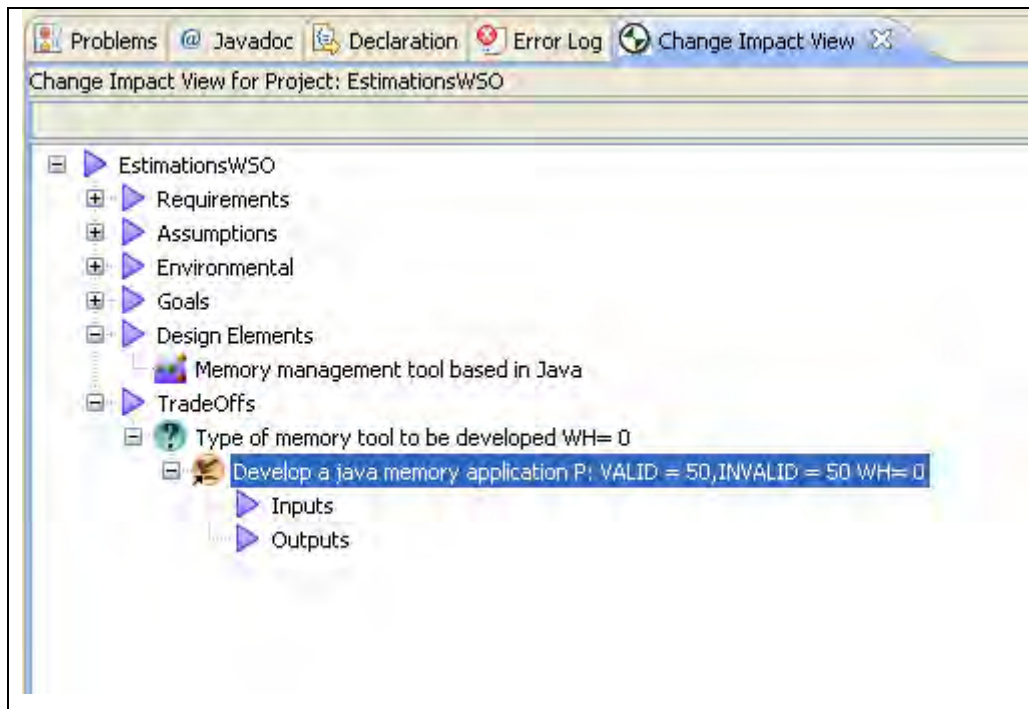


Figure 6-11: Decision inputs and output folders, generated automatically

As we can see in Figure 6-11, the decision to *Develop a new Java application* has a 50% of probability to be VALID, and the same probability to be INVALID, since the user has not indicated whether this decision is considered VALID or INVALID, and it has not any causes nor decisions associated to it. In principle, the APES tool *is agnostic w.r.t new decisions* (probabilities are equally distributed). Note also that at the right part of “*Type of memory tool to be developed*” there is a label “WH=0” that indicates the estimated working hours associated to this decision. This datum is generated based on the estimated costs of its design elements: since it has none, there are no working hours associated to the decision. At this point, the tool has internally generated a BBN; this BBN only has one node (the newly created decision), has compiled the internal BBN, and has propagated its probabilities. But design decisions for a project are triggered by its corresponding causes (i.e. decision elements that were evaluated in order to take the decision). These are considered “inputs” for the decision. In addition, decisions have the corresponding “outputs”, that is, consequences that come as a result of the decision taken (which in turn can be *decision elements* for further decisions). Both types of elements (*inputs* and *outputs*) need to be

added to the decision in order to provide accurate estimations. To complete the decision, we need to add those decisional elements that are inputs and outputs for the decision. We do this by adding the inputs (causes) and the outputs (consequences) for this decision, that are decision elements we added previously. By adding all the decisions and all input and output causes we end up having the following set of decisions associated to our trade-off, as shown in the following extract of the change impact tree (Figure 6-22)



**Figure 6-12: Initial tree for propagation**

The tree can be read as follows: for the “*type of memory tool to use*” trade-off there are three possible decisions to be taken, either to use an extension to an already existing MS access application, to develop the tool as an extension to a TCL/TK editor, or to develop it from scratch using Java. For each decision, the following considerations apply:

1. Decision elements that act as causes to take the decision to develop the memory tool as an extension to MS Access are: to *have available the team that developed the*



- former application* and the *similarity of the functionality to be developed* (AND displays). It will produce a *MS access memory tool* (design element) and will have an impact (negative) on the possible *reusability* in future space applications goal.
2. Decision to build a TCL/TK editor for alphanumeric displays will depend on whether the developer's team has a good knowledge of TCL/TK, and also on the similarity of the functionality to be developed with previous developments.
  3. Finally, in order to develop the application from scratch using the Java language, we must take into account whether we count on *a good java knowledge* of the team, the *similarity of the development w.r.t. previous applications*, whether we count on a *team of more than three people for the task*, and the *availability of a set of tables required for the development in Java*. It will produce a *java memory tool* and will have a (positive) impact on the reusability of the tool in future developments.

Note that the working hours for the decision and the issue have changed. This is because we have set the estimated working hours for each of the design elements to be produced for each decision. The Trade-off takes the working hours (WH=1200) for the worst case, considering that they have the same probability. Note also that the system is *not taking into account yet that these three decisions are mutually exclusive (sum of probabilities for each decisions associated to the issue is 1)*.

The complete change impact view is now shown in Figure 6-13



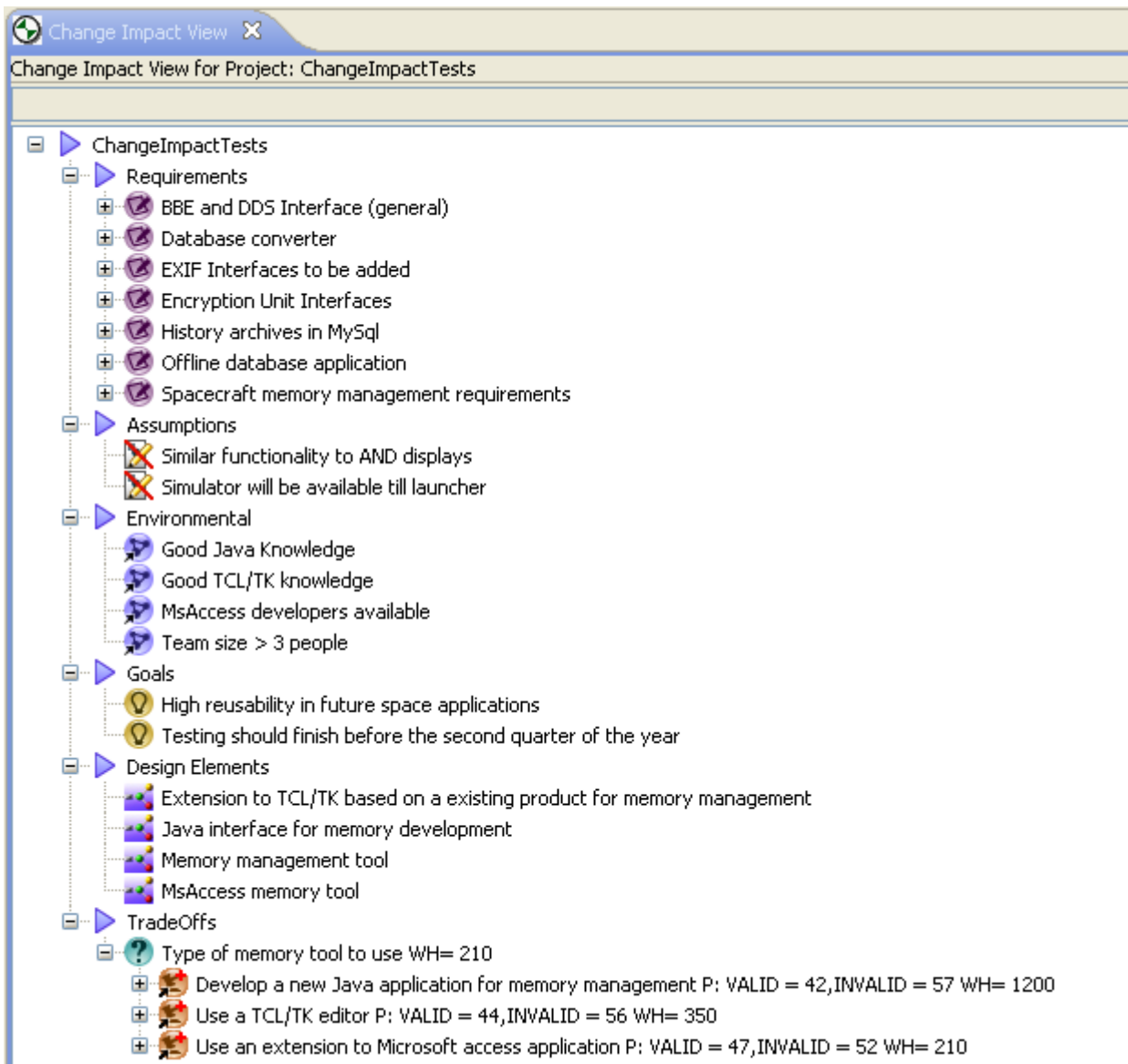










Figure 6-13: Decision Elements and its underlying network

At this point, the system is still "agnostic", and the probability for all these decisions is yet the same for all the possible decisions of the tree. Note that the system uses different icons for each type of element; the icons used are as follows:

-  This icon indicates that the element is a requirement. Examples of requirements in the figure are "Database converted" or "Offline database application".
-  This icon indicates that the element is an assumption. Examples of assumptions in the figure are "Simulator will be available till launch".
-  This icon indicates that the referred element is an environmental issue. An example of environmental items in the figure is: "Team size > 3 people".
-  This icon indicates that the referred element is a goal. Examples of goals in the figure are "Memory management tools should be reusable" or "Testing should finish before the end of this year".
-  This icon indicates that its associated element is a design element. Examples of design elements in the figure are "Java Interface for memory development" or "Memory management tool".
-  This icon indicates that this is a trade-off. There is a single trade-off in the figure, named "type of memory tool to use".
-  This icon indicates that this is a decision that is mutually exclusive w.r.t. other decisions that depend on the same trade-off. It was explained before that decisions that hang from a trade-off are not necessarily mutually exclusive. For those decisions that are not mutually exclusive, the icon for decision is somewhat different (has not the red plus sign, the icon used is ). In the figure, we can see the three decisions that we have added:
  - "Develop a new Java application",
  - "Use a Tcl/Tk editor", and
  - "Use extensions to Microsoft Access application".

It is from this point where a statistical analysis can be performed; this is the object of the next chapter.

### 6.4.3 STEP 3: STATISTICAL ANALYSIS: USING THE TREE VIEW

We are now at the point in which it is possible to see the propagation of probabilities for a decision, and perform a "what-if" analysis, to analyze different scenarios.

if we perform a review of the characteristics of the example, we saw previously that software architects have identified three different possibilities: The first option is the most risky (or at least that is what it seems), since it assumes that we will develop the system from scratch in Java. On the other hand, it will provide the developer's team with a valid asset that can be reused for future applications. This option is shown in the figure as "*Develop a new Java application*".

Another possibility is to develop an extension to a TCL/TK Editor. This is an internal software asset that the developers have, and could be easily customized for this particular spacecraft. This option is shown in Figure 6-13 as "use a TCL/TK editor". It would also provide a reusable tool, but its degree of integration is considered lower.

The third option would be to use an extension of an in-house development from the spacecraft's manufacturer. That will be in principle valid from the functional point of view, but it means to have two different systems running on-line. In addition, changes to this software will be required to be instantiated for the new spacecraft. This option is the one represented by the entry "*Use an extension to a Microsoft Access application*".

In Figure 6-13 we can see the working hours associated to each decision (labeled as WH=). Working hours for each decision are computed by adding all the working hours for the design elements that are outputs for the decision. So, for instance "*Develop a new Java application*" has 1200 hours assigned because we previously set 1200 as the hours required to develop the "Java Interfaces for memory development", "Use a TCL/TK editor has 350 hours, because "Extension to an existing TCL/TK application for memory development" has these hours assigned, and so on.

The working hours associated to the Trade-off "type of memory tool to use" are the working hours corresponding to the decision that has the highest probability (in this case, the system has selected "*Develop a new Java application*", although the three of them are equally

probable). Internally, as the users add inputs to the decisions, the tool builds and compiles automatically a BBN that holds the different elements. Figure 6-14 shows the BBN underlying this model. This BBN is generated internally and compiled by the system, and its probabilities are also automatically propagated by the system via calls to the Netica API, but it is not shown to the user, who does not need to have any knowledge on BBN technology.

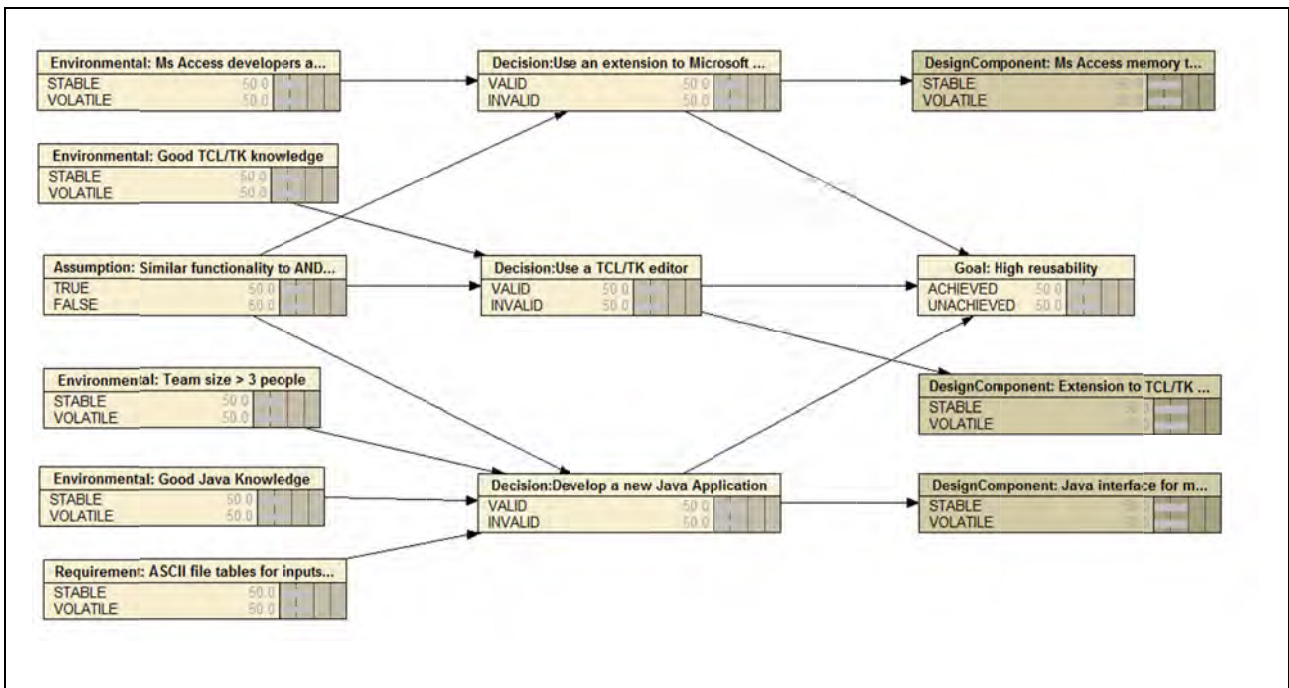


Figure 6-14: BBN Model for the “development of a memory tool” example.

We can see an equivalence w.r.t the tree view: “Use a tck/tk editor” is influenced by “Good Tcl/tk knowledge” and the “Similarity functionality to AND displays”, and produces “high reusability of the code” as well as a design component (extension to Tcl/tk existing SW). “Develop a new Java application” is influenced by the “Good Java Knowledge ” and the existence of “ASCII files for inputs”. It also depends on the availability of three developers to finish on time. Meanwhile, “Use an extension to Microsoft access” only depends on the similarity of the alphanumeric displays w.r.t. the previous applications and the availability of MS Access developers. Outputs for Java and MS Access are their corresponding memory modules, and all decisions have a different influence on a common goal which is “High reusability”. In the following sections this is analyzed in depth.

#### 6.4.3.1 STEP 3.1 ADJUSTING CPT’S FOR DECISION INPUTS

Here it is important to clarify what we mean by “probabilities” within APES-CIE. In previous sections we have seen how users insert decisions for a given trade-off. These decisions have “inputs” (causes) and “outputs” (effects) associated. Each of these “inputs” and “outputs” are elements that have a discrete attribute (status) with two different possible values. Although the users perspective is based on a tree of trade-offs and decisions, there is an underlying BBN that contains all the elements and decisions that are linked together. The steps that we will see in the following sections will allow users to fill the CPTs of this BBN in a user-friendly manner, as well as to set those “findings” (i.e. facts) that the user knows as certain. Based on this information, APES-CIE will propagate the probabilities for all the elements in the underlying BBN, and will indicate the probability for a decision to be taken, the probability for a design element to be part of the development, etc. These probabilities are not based on historical data; instead, they are based on the CPTs and the findings added by the user, which in turn depend on the plausibility that users concede to the possible outcome of a decision based on the status of its causes, as well as the possible outcome of an effect based on the decisions from which they depend.

Therefore, these probabilities indicate what users can expect based on their own beliefs, as well as those certainties that they have. They are not based on historical data, and shall be taken as estimations.

Since the system is in principle agnostic, the probabilities for all the elements that have been added are 0.5. Now it is time to adjust the conditional probability tables for the decisions, and the outcomes of these decisions. We will start by editing the probabilities for the decision “*Develop a new Java application*” and its values (VALID and INVALID) on each case by using the tool. The user edits the probabilities as shown in Figure 6-15

| Good Java Knowle... | Similar functionality to AND | Team size > 3 people | ASCII file tables for inputs and outputs | VALID | INVALID |
|---------------------|------------------------------|----------------------|--|-------|---------|
| STABLE              | TRUE                         | STABLE               | STABLE                                   | 100   | 0       |
| STABLE              | TRUE                         | STABLE               | VOLATILE                                 | 60    | 40      |
| STABLE              | TRUE                         | VOLATILE             | STABLE                                   | 85    | 15      |
| STABLE              | TRUE                         | VOLATILE             | VOLATILE                                 | 50    | 50      |
| STABLE              | FALSE                        | STABLE               | STABLE                                   | 45    | 55      |
| STABLE              | FALSE                        | STABLE               | VOLATILE                                 | 30    | 70      |
| STABLE              | FALSE                        | VOLATILE             | STABLE                                   | 40    | 60      |
| STABLE              | FALSE                        | VOLATILE             | VOLATILE                                 | 5     | 95      |
| VOLATILE            | TRUE                         | STABLE               | STABLE                                   | 40    | 60      |
| VOLATILE            | TRUE                         | STABLE               | VOLATILE                                 | 35    | 65      |
| VOLATILE            | TRUE                         | VOLATILE             | STABLE                                   | 10    | 90      |
| VOLATILE            | TRUE                         | VOLATILE             | VOLATILE                                 | 5     | 95      |
| VOLATILE            | FALSE                        | STABLE               | STABLE                                   | 20    | 80      |
| VOLATILE            | FALSE                        | STABLE               | VOLATILE                                 | 10    | 90      |
| VOLATILE            | FALSE                        | VOLATILE             | STABLE                                   | 10    | 90      |
| VOLATILE            | FALSE                        | VOLATILE             | VOLATILE                                 | 0     | 100     |

Figure 6-15: Editing a CPT for a decision: develop a new Java application

The meaning of the table shown on the upper table of Figure 6-15 is as follows:

- The decision “*Develop a new Java application*” shown on the tree under “Type of memory tool to use” is driven by the inputs to the decision.
  - *Good Java Knowledge*: is an environmental issue that is crucial for such development.
  - *Similar functionality to AND displays*: means that we share the experience with previous developments, and eases the development of a new application from scratch.
  - *Team size > 3 people*: is also important. It is important to determine that we could count on more than 3 people for the development.

Therefore our stakeholders have determined, among other knowledge, that:

- If *Good Java Knowledge* is guaranteed, there is a *Similar functionality to AND developments*, and *team size could be more than three people*, the probability of this decision to be valid is 95% (first row). This is the probability assigned by users based on their estimations.
- If *Good Java Knowledge* is guaranteed, there is a *Similar functionality to AND developments*, but the *team size cannot exceed three people*, the probability of this decision to be valid is 60% (second row). Users assign this probability, considering

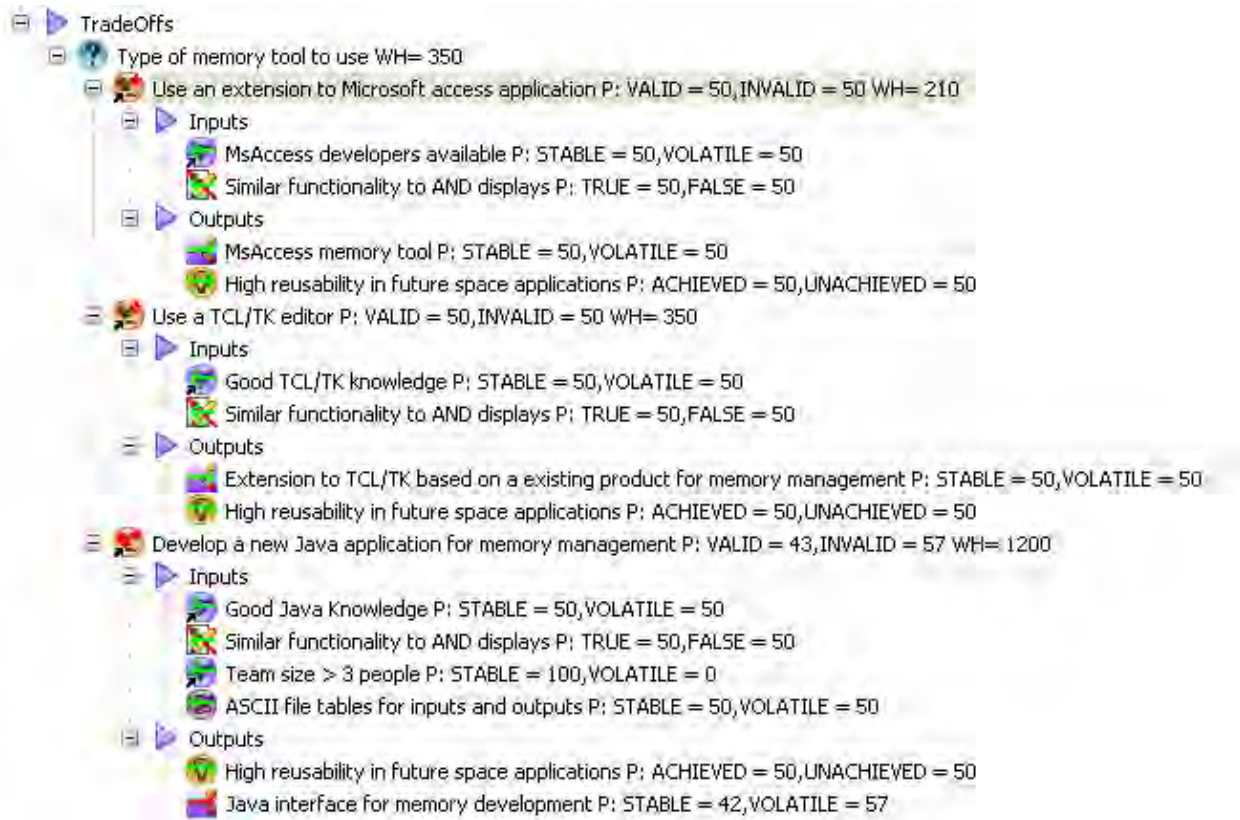
that without three persons in the team, it would be more difficult to develop this application in Java, since it would require more effort.

- If *Good Java Knowledge* is guaranteed, there is NOT a *Similar functionality to AND developments*, but *the team size can exceed three people*, the probability of this decision to be valid is 85% (third row). The users decide to assign this probability considering that most of the development will have to be done from scratch, there will be more than 3 people available and that the team has a good Java command.

These probabilities need to be setup by hand by the developers/project manager, by reaching an agreement on the influence of the different factors in the validity of the decision. The weight of each case is decided by those involved in the development. Therefore, it is an estimation reached by agreement from the stakeholders.

Also, obviously, the sum of probabilities (for VALID and INVALID) for a given row should be 1. Otherwise when saving the CPT the application will raise an error, and will not let us save the changes to the CPT.

Figure 6-16 shows the tree once the CPT for the decision "*Develop a new Java application for memory management*" has been modified according to Figure 6-15. The icon for such decision is marked now in red, showing that its probabilities have changed.



**Figure 6-16: Change impact tree view showing probabilities for a decision**

Note that the new probability for the Java decision depends on the newly CPT values added by users, as well as the probabilities of its inputs. Note also that working hours for the trade-off “type of memory tool to use” have changed to 350, while it was 1200 before. This is due to the fact that the most likely decision now is “use a TCL/TK Editor”. The tree sets automatically the working hours of a trade-off that has mutually exclusive decisions to the working hours of the decision that has a higher probability to be VALID.

Similar operations are done for the other decisions of this trade-off (use a TLC/TK editor and use an extension to a Microsoft Access application).



### 6.4.3.2 STEP 3.2 ADJUSTING CPT'S FOR DECISION'S OUTPUTS

Once we have setup the tables for the decisions, it is time to setup the tables for the outcomes of these decisions. These are the items that were indicated in the "Outputs" sections of the decision.

Some outcomes could be the result of several decisions, while others can depend on a single decision. In our case, "high reusability" is an output of the three decisions, and therefore depends on the results of the other decisions. Other outputs depend exclusively on a single decision, and for this kind of outputs the probability table is very simple.

So, for instance "Java interface for memory development" will only be STABLE whenever the decision "*Develop a new Java application*" is set as VALID. If we edit the conditional probability table for this output of the decision as shown in Figure 6-17.

| Develop a new Java... | STATIC | VOLATILE |
|-----------------------|--------|----------|
| VALID                 | 100    | 0        |
| INVALID               | 0      | 100      |

Figure 6-17: Editing an output for an output of a decision (outcome).

As we can see in the figure, for this particular development item, it should only be developed in case that the decision to *Develop a new Java application* is VALID. Therefore, the corresponding CPT is straightforward. By contrast, goal "Memory Management tool should be reusable" has three decisions that could cause the goal to be obtained. The corresponding CPT table is shown on Figure 6-18

| Develop a new... | Use a TCL/TK e... | Use an extensi... | ACHIEVED | UNACHIEVED |
|------------------|-------------------|-------------------|----------|------------|
| VALID            | VALID             | VALID             | 100      | 0          |
| VALID            | VALID             | INVALID           | 100      | 0          |
| VALID            | INVALID           | VALID             | 100      | 0          |
| VALID            | INVALID           | INVALID           | 100      | 0          |
| INVALID          | VALID             | VALID             | 0        | 100        |
| INVALID          | VALID             | INVALID           | 70       | 30         |
| INVALID          | INVALID           | VALID             | 10       | 90         |
| INVALID          | INVALID           | INVALID           | 0        | 100        |

**Figure 6-18: Editing the outcome based on those decisions that influence it**

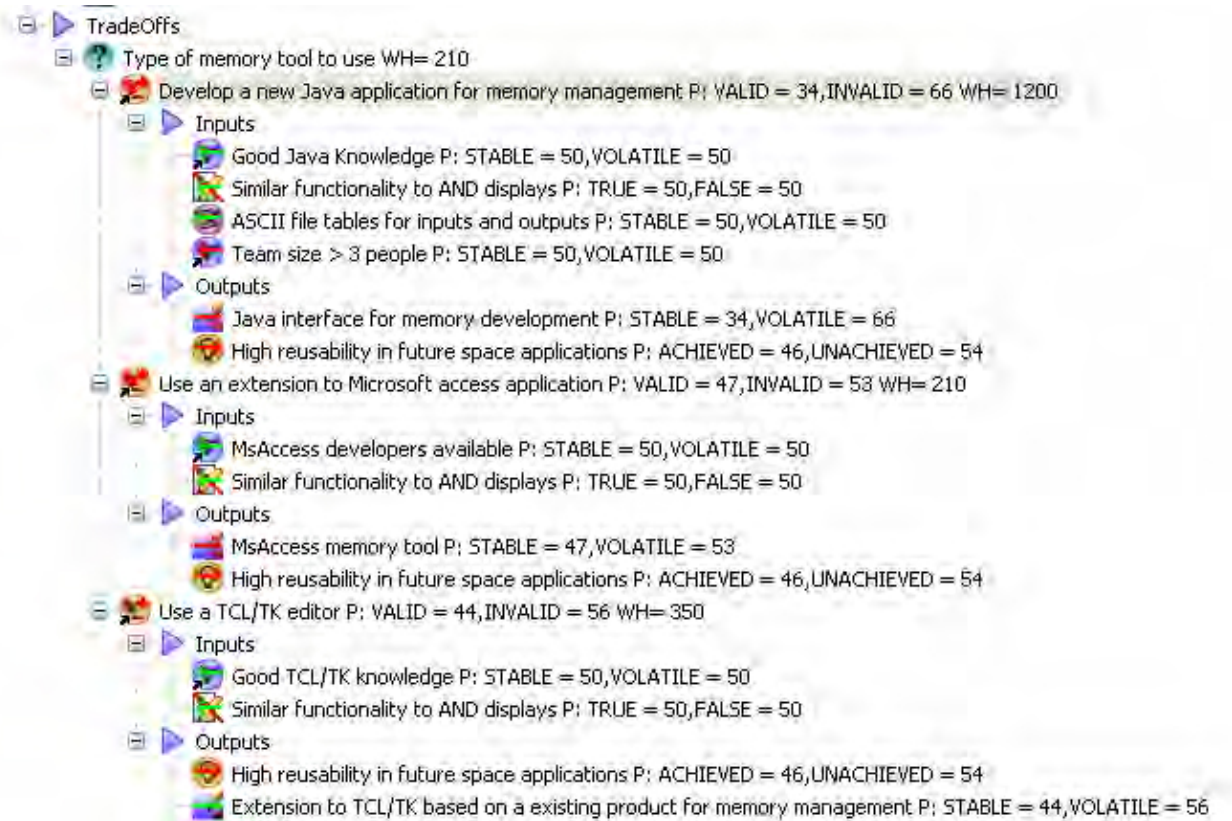
For this particular table, although we know that the decisions are mutually exclusive, we set the values “ACHIEVED” in the corresponding CPT for those combinations in which the decision to *Develop a new Java application* is set as VALID.

For the others (combinations in which only one of the decisions is set as VALID) we have assumed the following results:

- For the case in which we *Develop a new Java application*, we have set the goal to ACHIEVED in a 100%.
- For the case in which we *Use a TCL/TK Editor*, we have set the probability to achieve the *High reusability* goal to a 70%, since we consider that it could be discarded in the long term.
- For the case in which we *will use an extension to MsAccess application*, we have set the probability to reach the goal to a 10%. It is very unlikely that such application will be reused in further developments.

By saving the corresponding table, the tool will propagate the probability values.

In our example, once we have filled the corresponding CPT,s for all decisions and outputs, the tree has changed the probabilities as shown on the figure:



**Figure 6-19: Probability propagation once the CPTs have been filled**

At this point there are no “findings” or “facts”. That is, all probabilities are computed based on propagation from inputs based exclusively on the propagation of probability from the CPTs. We are assuming that we don’t know anything about the inputs that are triggering the decisions to be taken. Therefore, all decisions seem to have very similar probabilities. The decision to use an extension to the Microsoft Access application seems the most likely, but its associated probability (47%) is very similar to *Use a TCL/TK Editor (44%)* or *Develop a new Java application (34%)*. This is due to the fact that we haven’t put in place those facts (certainties) that the users might have about the causes that trigger the corresponding decisions. This is what we will do in the following section.

### 6.4.3.3 STEP 3.3 FORWARD PROPAGATION: PREDICTIVE REASONING

Now let us suppose the user wants to analyze what would happen in case we could rely on some facts. For instance, the decision to develop a new Java Interface was influenced by the possibility to have a team with more than three people. Let's suppose that we know this fact, and that we can rely on the fact that *Team size >3 people* will be STABLE. We can edit the corresponding entry, and set the value to STABLE manually, as shown in Figure-6-20

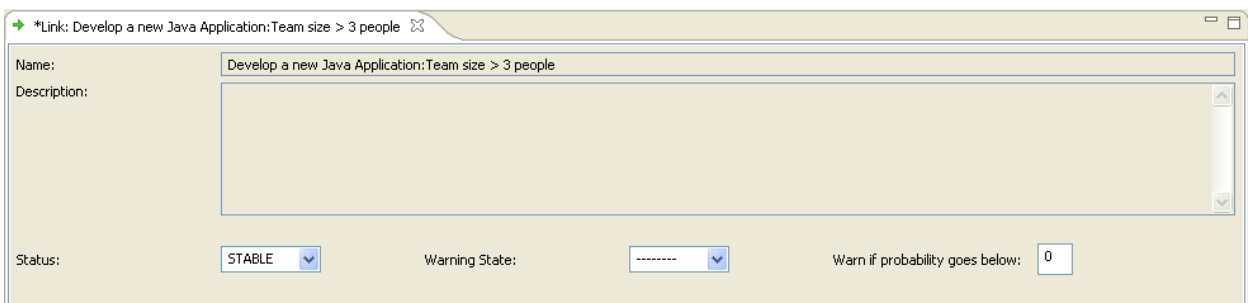


Figure-6-20: Setting a fact for an input once it is known.

Once we have done this, we can see that the probability to develop a Java application is higher now (42%), and the tree becomes as shown in Figure 6-21



Figure 6-21: Tree view probability propagation once a fact for an input is set

Note also that those elements that have changed its probability are now in red. That is:

- *High reusability in future space applications* has increased its probability to be ACHIEVED to 53% (before it was 46%).
- *Java interface for memory development* has increased also its probability to 42% (before it was 34%).
- The decision to *Develop a new Java application* has raised to 43% from 34%.

The working hours for the tradeoff have not changed, since the most probable decision is still "*Use a TCL/TK editor*". The opposite effect would be obtained in case the team size could not be higher than three people. That is, that would lower radically the probability of developing a Java application.

We see with this example how by setting the corresponding facts, users can determine the probabilities for each decision, which could help them take the right decision. In addition, the probabilities filled by the users contain the rationale for the decisions to be taken.

Similarly, users can set the decisions as facts; that is, they can set manually the status of a particular decision to VALID or INVALID. The corresponding probabilities for their effects will be propagated, and that would give them an idea of the results of the decisions taken. In our case, for instance, setting the decision to "use an extension to Ms Access application" will generate the results shown in Figure 6-22.



Figure 6-22: Setting a fact for a decision.

The automatic propagation of the probabilities allows users to see the results in the tree. In our case, we can see that, by taking the decision to develop the system in Microsoft Access, High reusability is very unlikely to be obtained. Also, the list of items to be developed is shown on the tree.



Taking into account that these three decisions are mutually exclusive, the system has set automatically the status of the remaining decisions to INVALID, and therefore all decisions have changed their probability, and so have been affected all its outputs and inputs. The only input that has not been affected is "*Team size > 3 people*" that was set as a finding (a fact) previously, and therefore has not changed its probability.

In addition, the working hours for the trade-off have not been modified, since the decision to be made is the same as before.

From the previous discussion, we can extract three conclusions:

- By adjusting facts to the inputs of the decisions, we can see the more plausible decision, and the probabilities for the different outcomes.
- By selecting a decision, we can see the probabilities associated of its outcomes (effects)
- When setting a decision that is a multiple choice as VALID, all other decisions from the same trade-off that depend on the same trade-off and are mutually exclusive are set to INVALID automatically.

#### **6.4.3.4 STEP 3.4 BACKWARD PROPAGATION: DIAGNOSTIC REASONING**

It is also possible to analyze the decision chain in the opposite direction; that is, from the effects that are desired, to the required decisions to be taken for them. An example of this follows. Let us suppose we leave the decisions as "unknown", by editing decision *Use an extension to Ms Application*, and setting its decision's status as unknown ("-----" as shown in Figure 6-23).

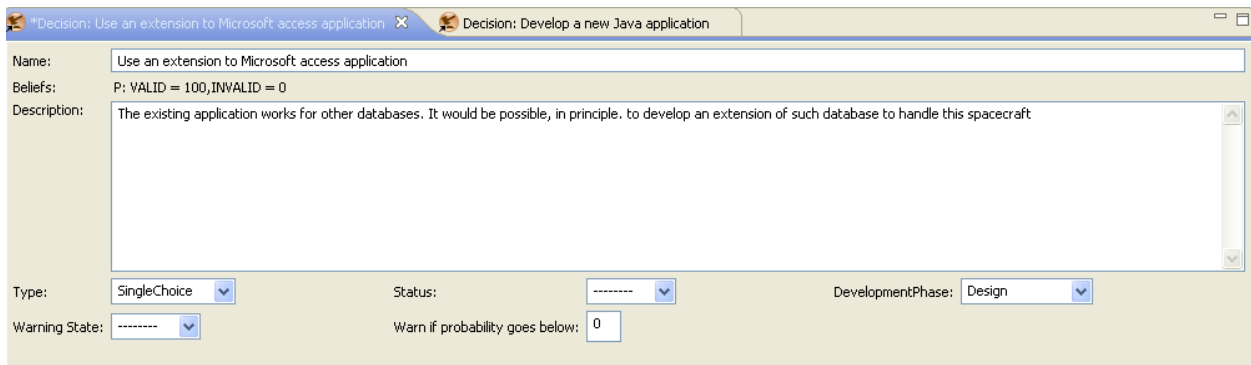


Figure 6-23: Reverting a decision fact to “unknown”.

Saving the corresponding tab, we now set the “status” value for the “high reusability goal” to ACHIEVED. Then, as shown in Figure 6-24, the corresponding probabilities change for the decisions.



Figure 6-24: Setting a fact for an output goal, analysis of decisions to reach that goal.

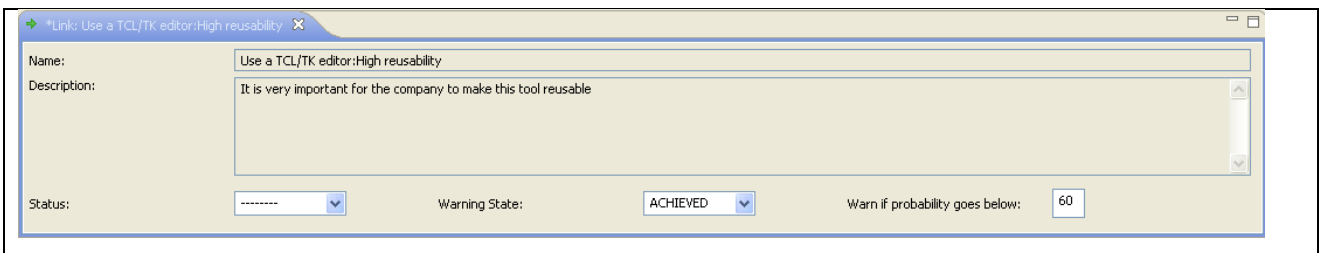
We now have setup the goal “high reusability” to be the most important one for our development. The probabilities for each decision have changed, clearly showing that the



decision to “Develop a new Java application” is the most likely to be implemented. All decisions have been affected, and all the inputs and outputs are now in red, indicating that their probability has changed. The working hours for the trade-off have been set to 1200, because the most probable decision is “Develop a new Java application”, that has a design element “java interface for memory development” that has a 1200 hours workload. Once again, the only input that has not been affected is the “Team size > 3 people”, because it was previously set as a fact and therefore its probability has not been modified. In this way user can also determine which the decision to be taken is in order to obtain a set of given results. Users can therefore analyze not only the effects of decisions, but also which are the right decisions to take in order to obtain a given result.

#### 6.4.3.5 USING THRESHOLD WARNINGS

The tool can also reason with a “warning” threshold. This warning threshold feature is provided for inputs, outputs and decisions that can be used to detect whether the probability for a given element goes below a given threshold. For instance, let us now modify the threshold for the link “High reusability” and, at the same time, set the STATUS as UNKOWN (“-----”), as shown in Figure 6-25



**Figure 6-25: Setting threshold for High reusability if probability is lower than 60%.**

By saving the corresponding values we can see what happens when we unset the “STATUS” to VALID and set the Warning State to ACHIEVED with a probability lower than 60%.

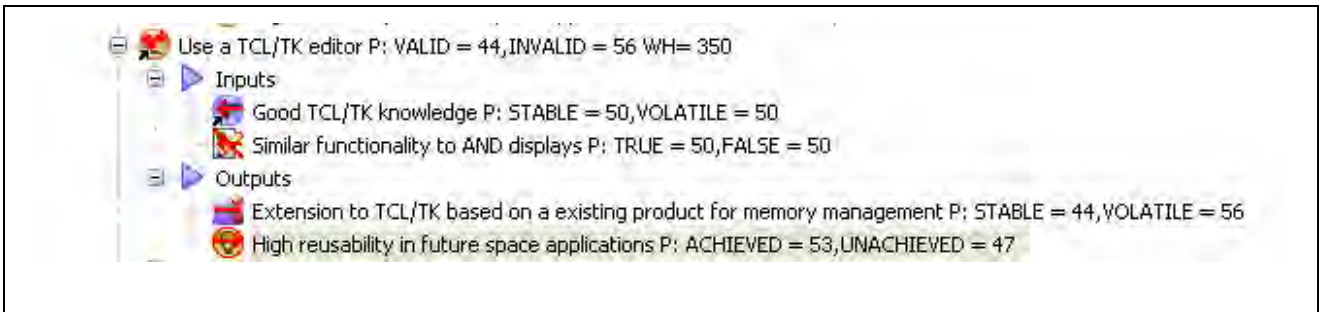


Figure 6-26: Probabilities affected and Warning for the High reusability.

We can see that the probabilities have changed now, because we unset the *High reusability* as a fact (and this implies that its probability will be computed again by the system), but also we can see that the *High reusability* is lower than 60 percent, and therefore the tool shows an icon that indicates that the probability is lower than what we expect for this decision.

## 6.5 APES-CIE FROM THE AEROSPACE'S METHODOLOGICAL PERSPECTIVE

APES-CIE was designed for the aerospace market, in particular having in mind the ECSS standards from ESA (ECSS-E-40, 2009). As we pointed out before, the APES-CIE system can be used from the very beginning of the development cycle for multiple purposes.

Figure 6-27 shows the different activities involved in the software development process

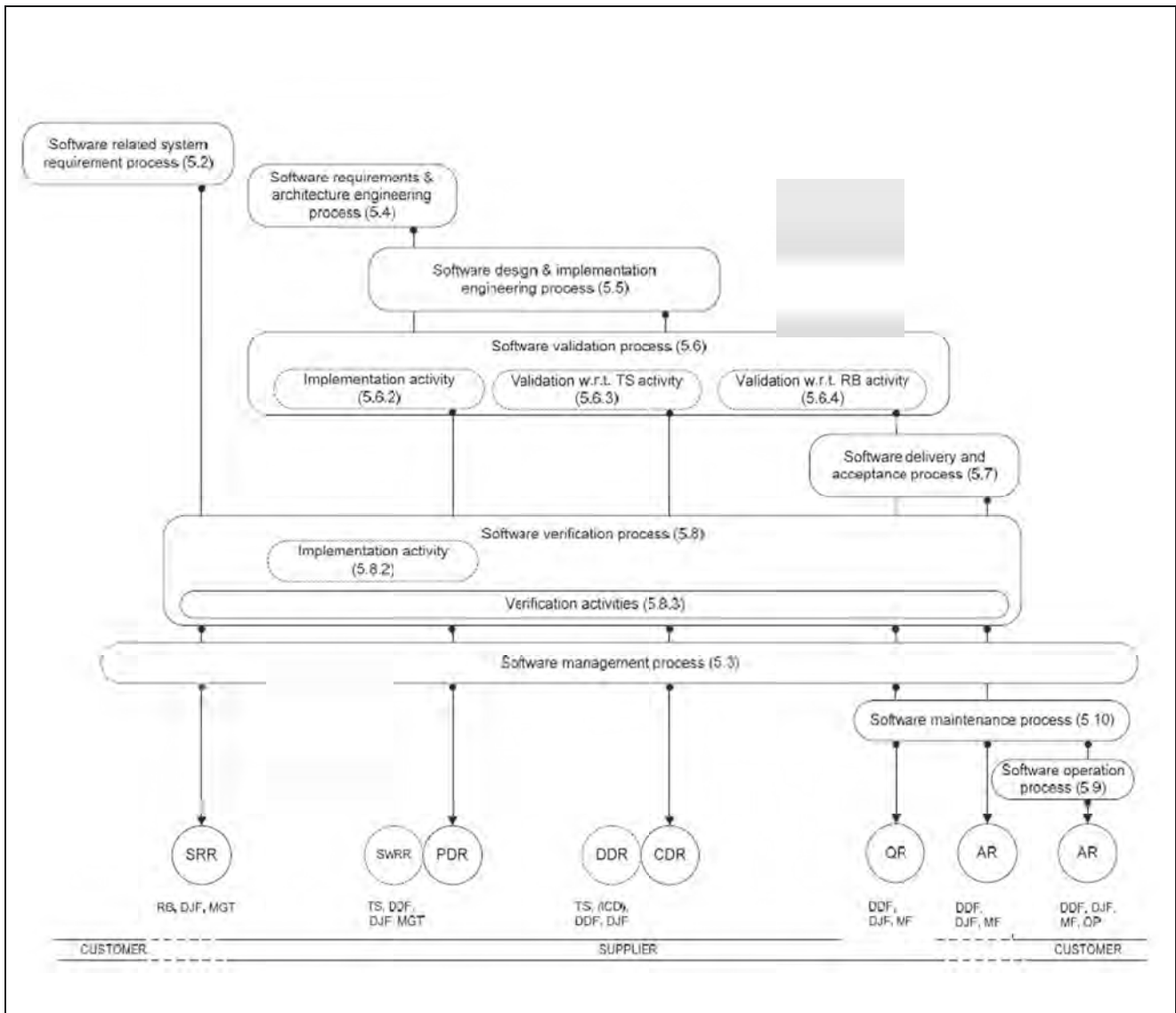


Figure 6-27: Software lifecycle process, as depicted in (ECSS-E-40, 2009)

The APES-CIE system has a direct application, at least, to the following processes:

- In the “software requirements and architecture engineering process” activity, ECSS standards explicitly state that “during this process, the result of all significant trade-offs, feasibility analyses, make-or-buy decisions and supporting technical assessments are documented in a design justification file (DJF)”. The immense majority of this information can be gathered by using APES-CIE. Moreover, APES-CIE will provide a much more detailed justification of the agreements and the reasons

behind the decisions taken. Backward and forward propagation can be used to identify key goals to be achieved, design elements involved, the root requirements behind a decision, and those assumptions that we made when we took them. This will allow taking the right decisions in advance, which in turn reduces costs and increases the quality of the final product.

- During the “software design and implementation engineering process”, the ECSS standards require to add into the DJF, all *“those important design choices, analysis and test data that show that the design meets all requirements”*. The results of this process are the input to the critical design review (CDR). The word “important” here is the key: not all decisions can be stored into the APES-CIE tool, because that would create a tree very difficult to manage. Instead, users can add those decisions that have greater impact to the tool, as well as their related elements, and this information can be gathered and stored into the tool. Moreover, users can add a priori decisions that they think that might have a great impact, and delete the items of information related to this decision later from the tool if it is considered too exhaustive.
- For the software validation process, part of the information from APES-CIE could be important to verify; that is, check that the technical specification and the requirements baseline functions and performances are correctly and completely implemented in the final product. It is during this process that we can verify and refine most of the estimations performed before, updating the CPTs, setting known facts, etc. The model provides a partial traceability from different elements of the software: from requirements to design elements, from design elements to tests, being it possible to trace back the outcome thanks to backward propagation.
- For the maintenance process, the main objective is *“to modify the existing software product while preserving its integrity”*. Although we haven’t had the opportunity to test it for maintenance, it is obvious that the information contained in the APES-CIE tool will be a very valuable asset for maintenance, since it will help to understand the reasons behind design decisions.

- An additional possible use of the tool is during meetings and reviews. As a mind mapping tool, it provides an overview of the main factors that lead to a decision, of the weight that these factors had in taking or discarding such decisions in the developer’s mind, and on the effects that were caused by them. From that perspective, it can be a very powerful communication tool.
- Questions on these potential capabilities were asked to stakeholders of different projects. The results are detailed in the next chapter of this thesis.

## **7 EVALUATION**

For the evaluation of the use cases, it was not necessary to evaluate the robustness of the BBN statistical propagation, since APES\_CIE used an already existing API from a BBN tool (Netica, 2008) in charge of the propagation of probabilities. Instead, our aim was to test the validity of the model for design rationale and change impact estimation, from the user's point of view. We wanted to know, among other issues, whether: the tool was easy to use; it provided an added value for design rationale; or users thought that it allowed to improve existing methods for change impact estimations.

Therefore, the evaluation of the model was conducted by using the tool in the development of two different projects. The first project was an on-board software development for a space mission: this use case is a development of critical software, which follows dense and strict procedures. For instance, code testing is structured in 4 different layers (unitary tests, numerical precision tests, integration tests, system tests) with a combined coverage of 100% of the code. Programming rules are very strict; each individual deviation needs to be justified, and the documentation of the project consists of dozens of different deliverable documents, each with its own different releases (User Manual, Software development plan, Interface Control Document, Design Justification File, Software Development Plan, Risk Register, etc..)

The second project to which the tool was applied is a development of a robot (a rover) for an oil & gas platform at sea. This is a very different project in which, although the software is also critical, the number of documents to be delivered, that is, the evidences to be provided, are fewer. The project has also a high dependency on the hardware; many decisions depend on the availability or the accuracy of a set of sensors and actuators for a given task. This case provides an added value: to investigate the validity of the tool when used for the integration of hardware and software.

The tool was shown and used for both projects, and after the relevant discussions, a questionnaire was filled in by the stakeholders of the project. This chapter is structured as follows:

- Section 7.1 details the characteristics of the first project (on-board software development) in which the tool was used, and focuses on some important issues & aspects
- Section 7.2 describes the characteristics of the second project for which the tool was used (rover for a gas & oil platform at sea). As in the previous case, we will concentrate on the results of the use of the tool. As we will see, the most important aspects referred to the integration between software and hardware.
- Finally, in section 7.3 we will comment the different elements of the questionnaire, and the answers provided by the stakeholders.

## **7.1 USE CASE 1: ON-BOARD SOFTWARE DEVELOPMENT**

The objective of this use case is the development of application software (ASW) for two different spacecraft of a space mission: a carrier module (CM) and a descent module (DM) that will control and guide a capsule to land on Mars.

Several subsystems of the application software will be shared between carrier and descent (e.g. thermal), meanwhile others (e.g. GNC) will not be shared, since the CM and DM requirements for these subsystems are totally different.

This mission combines high costs, very strict deadlines, together with a very high software criticality: any failure of the software could cause mission loss.

The mission faces multiple technical challenges. In addition, the team is composed of 7 developers including the project manager. A prime contractor provides the requirements for

the software, is responsible for the integration of the ASW into the remaining OBSW components, as well as for the system testing. Project’s duration is 28 months.

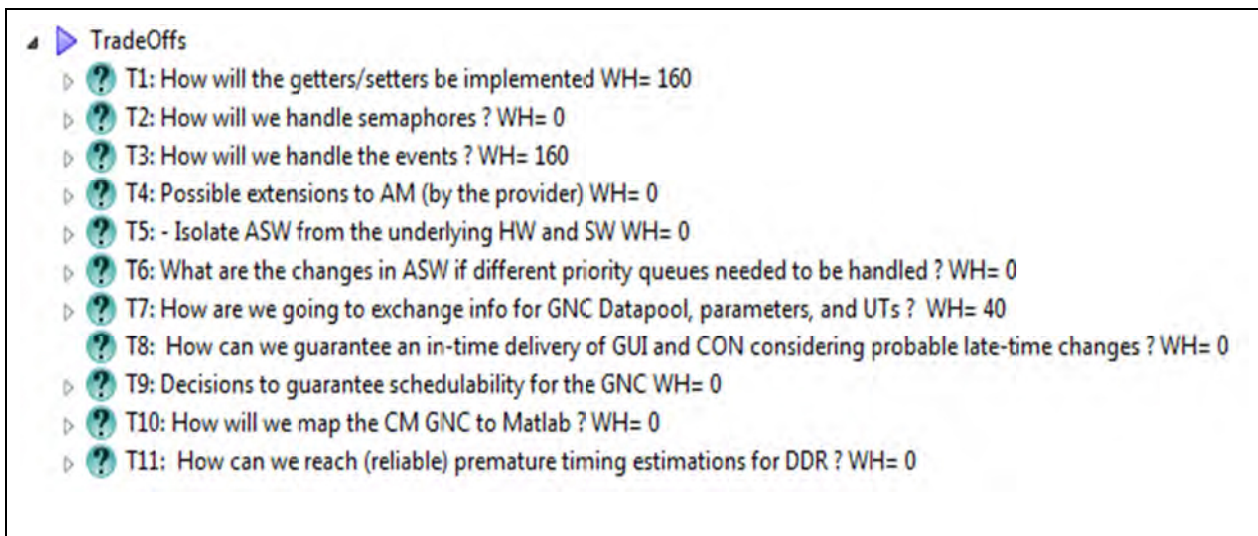
The APES-CIE tool was applied during the detailed design of the application software, trying to identify and assess those issues, and their related decisions, that will lead to a better design and will overcome existing problems.

One of the most important factors for the application software is that it has to be integrated within an existing framework. This framework provides different possibilities for the ASW, in order to perform its different tasks. In addition, in some cases, the application software itself will need to cover capabilities not provided as part of this framework. Therefore, most of the issues to be tackled are relative to the way the ASW will interface with the existing framework.

Another important issue for the project is how to interact with the different teams of different companies. This aspect refers not to the SW being developed, but instead to the SW assets related to the different workflows in which all those companies involved will have to interact (for instance, testing).

The main trade-offs identified in this case are depicted in Figure 7-1.





**Figure 7-1: Different Trade-offs to be assessed and identified.**

So, for instance, in Figure 7-1, Tradeoffs from 1 to 6 are related to design decisions on how to interact with the existing framework, and design decisions of the ASW itself. Tradeoffs 7, 8, 10 and 11 are related to the verification and validation activities and depend also on the interaction between different teams working for different companies.

A very interesting example of the possibilities for the APES-CIE is its use for a design decision oriented towards the schedulability of the Guidance, Navigation and Control subsystem (which is part of the application software). The referred framework included a concept named “subsystem manager”. The Guidance, Navigation and Control of the system is a subsystem manager, and as such, it had (in principle) a single task associated to it.

But, in the case of the Guidance, Navigation and Control, we needed to split it into three different tasks: a first task will collect data from the Inertial Measurement Unit (IMU) and the radar and compute the navigation. Meanwhile, a second task (with lower periodicity) had to accomplish the guidance (GUI) and control (CON) functions, in a control loop which has a lower frequency.

Therefore, we had four possible decisions.

- The first one was to run the Guidance, Navigation and Control (GNC) task in a single task (the very high frequency one), and every 10 or 20 times this task was active, calling the guidance and control functions (D1 in Figure 7-2).
- Another possibility was to split the GNC into two different “subsystem managers”, each with its own task (D2 in Figure 7-2). This had also disadvantages (lack of cohesion, since many data from the GNC will have to be shared among the two tasks).
- A third possibility was to split the work to be performed periodically for guidance (GUI) and Control (CON) into different steps, each step to be executed into a “time-slice” (high frequency control loop). In this case we will execute calls to guidance or control functions in the high control loop, but these functions will be divided into different steps. That is, it will execute successive “steps” of Control and Guidance, instead of the whole guidance and control algorithm in a single “tick” of the high control loop. That will possibly allow the high control loop to be schedulable (D3 in Figure 7-2). This option had also disadvantages, namely, that we would need additional data structures to collect the partial results for Control and Guidance obtained for each step.
- Finally, we have the option to extend a subsystem manager so that it is able to allow a single subsystem manager to have two different tasks running (Option D4 in Figure 7-2).

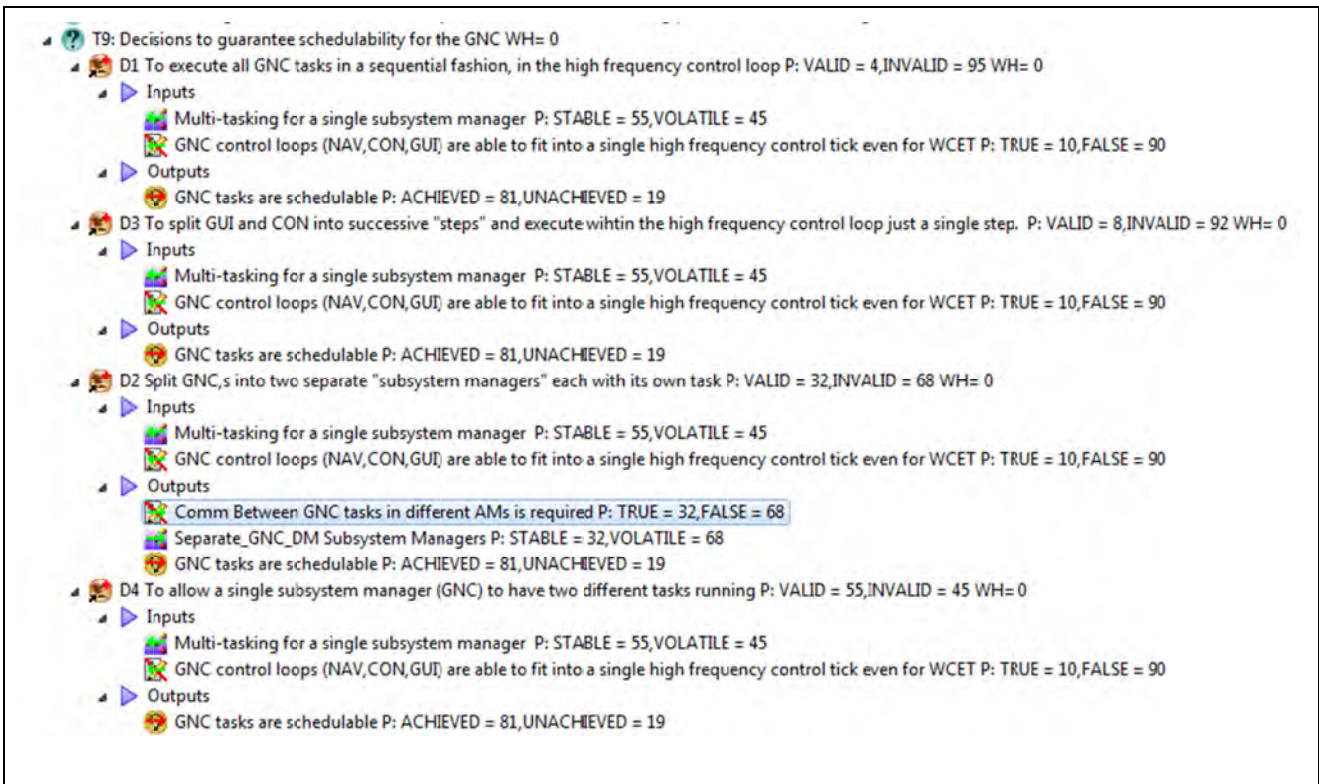


Figure 7-2: Decisions associated to GNC schedulability.

An initial analysis, based on previous developments, allowed us to determine that the probability for all GNC control loops to fit into the high frequency control loop was a 10%. Based on that, and based also on the corresponding CPTs for the decisions and their effects, assuming a 55% probability for multi-tasking to be possible in a single subsystem manager, the system pointed out D4 as the most probable option. After discussions with the contractor, it was identified that it was possible to have multitasking (two tasks) in a single subsystem manager, and therefore the right decision (D4) was taken. But the tool allowed us to make the proper balance related to the decisions. This serves as an example of the usage that we did to the tool within this project,

## 7.2 USE CASE 2: ROVER FOR OIL & GAS PLATFORM AT SEA

The objective of this project is the development of a medium-size rover able to read manometers and valves in an oil & gas platform located overseas. This rover has to be able

to negotiate stairs and overcome obstacles, smoothing out its movements when dealing with inclined ground and lifting the robot up when it needs to go over an obstacle. Perception sensors (cameras and 3D Laser) and the inertial measurement unit (IMU) allow the rover to determine its position and attitude, thanks to image acquisition, localization and navigation software. Image processing algorithms are used for the reading of manometers and valves. In addition, Methane sensors in the robot are used to alert of a possible explosion.

The project's team is composed of 8 different engineers, and development's time is around 30 months.

The development is subject to a competition among different companies all over the world, and once a year these companies have to pass a set of evaluation tests. The tests performed one year serve to demonstrate basic capabilities that will be used in next year's tests.

In autonomous and semi-autonomous modes, the system must autonomously detect identify anomalies, such as leaks, unforeseen obstacles, or out of limits measurements taken from manometers and valves.

Being a project not as critical as the previous one, this project involves both hardware and software, and it depends heavily on the right integration between both. In addition, there are multiple suppliers for: sensors, actuators, motors, batteries, etc. and it is necessary to evaluate multiple alternatives related to the hardware, as well as to perform the correct hardware design. There are multiple algorithms whose reliability depend on the accuracy and the availability of sensors and actuators.

These characteristics lead to a complex HW and SW design, with multiple inter-dependencies among the HW and SW assets. For instance, to pass a single navigation and vision test, the rover has to achieve multiple sub-goals, which in turn depend on a harmonic collaboration between HW and SW. Considering that the rover is designed from scratch for

this competition, it is very easy to miss the global picture about which decisions are more critical and what are the consequences for the different cases.

The tool was used after the first year of development, at a point in which the rover was already built but faced numerous difficulties. We tried to identify those trade-offs that were more critical and that were needed to be solved and assessed for the second year.

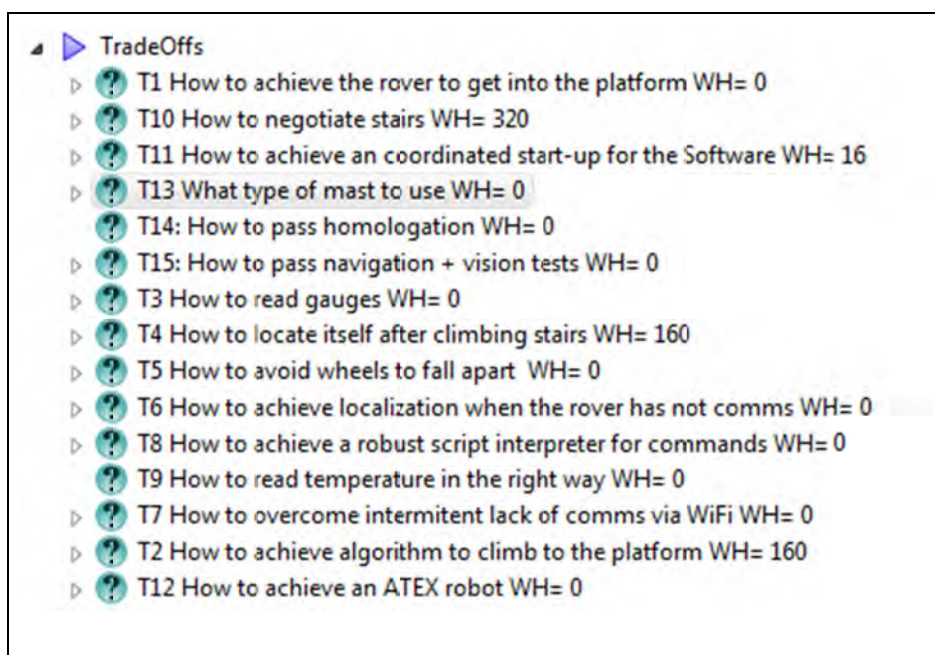


Figure 7-3: Trade-offs related to the rover for Oil & Gas.

Most of these trade-offs refer to decisions regarding the capability of the rover to perform basic tasks: negotiate stairs, climb into the platform, locate itself after climbing stairs, read gauges, or read the temperature properly. These basic skills will be also the input to other tests that require a combination of them, as, for instance, pass navigation and vision tests.

Other trade-offs refer to decisions yet to be taken due to the failure of a subsystem: for instance, during the tests the mast that was carrying the cameras failed, and therefore we had to decide for an alternative one. Finally, some of them were related to problems detected during tests. For instance, how to overcome lack of communications during the



tests, or how to achieve a coordinated start-up (that was non-deterministic and sometimes failed).

There was a common consensus from the team that the APES-CIE tool helped to identify the most critical issues and their inter-dependencies.

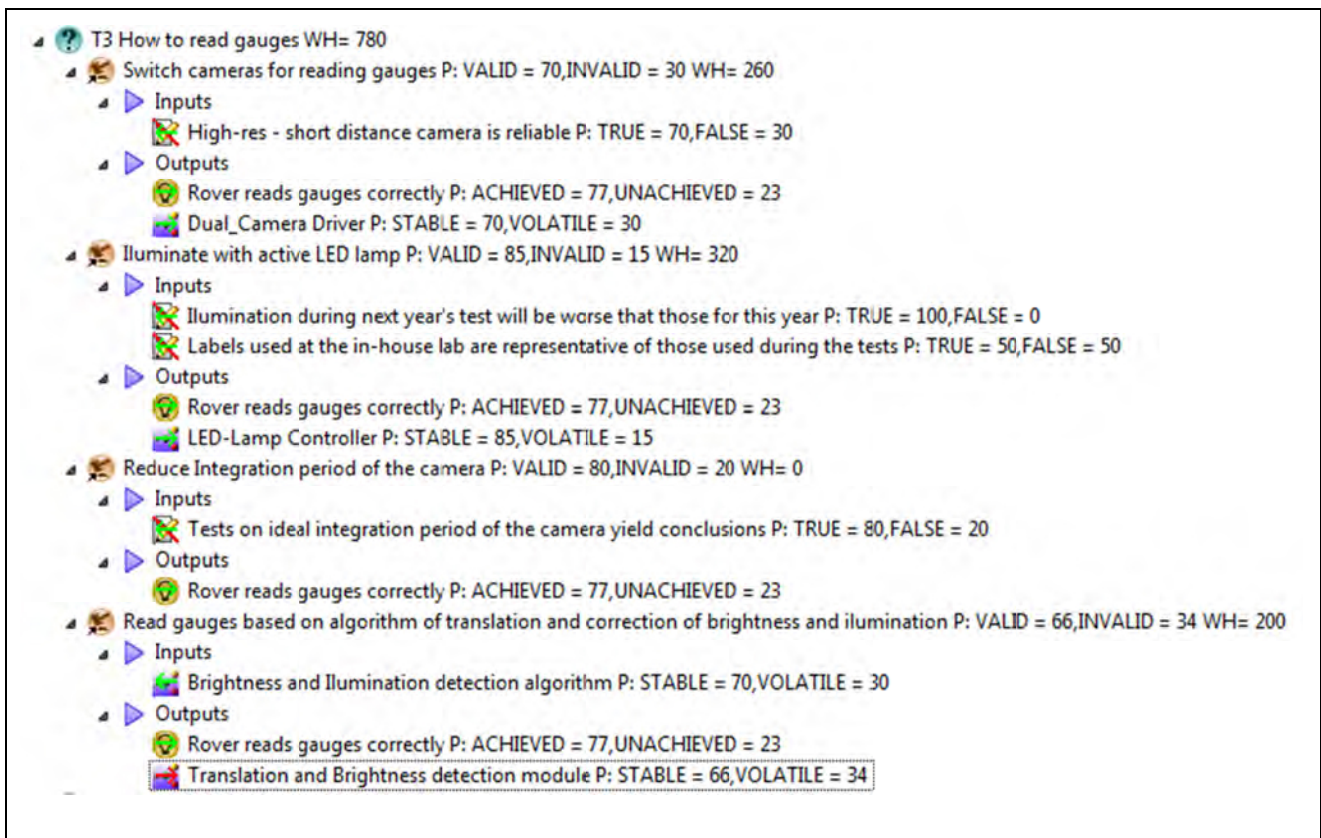


Figure 7-4: Decisions for reading gauges.

In Figure 7-4, we can see different decisions related to gauge reading, such as switching cameras, illuminate with active LED lamp, reduce the integration's period of the camera, and reading gauges based on translation and correction of brightness and illumination.

An important difference with respect to the previous case is that all these decisions are independent; that is, they are not linked to each other. Figure 7-4 shows the probabilities once the CPTs were adjusted and once the probabilities for its main inputs were determined (some depend on previous design decisions to be taken during the development).

Obviously, in this case, the main goal (Rover reads gauges correctly) is ensured by adopting all the decisions proposed to guarantee a higher accuracy, but not the high effort (around 780 hours) required to implement all the decision elements involved (780 hours). Also, some of the decisions depend on the availability of equipment/algorithms or even the reliability of some testing to be performed.

### **7.3 EVALUATION METHOD**

The evaluation method consisted of a questionnaire that was filled by the developer's teams of these projects. The team interviewed consisted of:

- For the first use case, a system engineer, with more than 20 years of experience, and a project developer, with 4 years of experience
- For the second use case, the project manager, with 14 years of experience and two developers, with 2 and 4 years of experience.

The questionnaire included a set of questions that had to be evaluated (in a scale from 1 to 5, being 1 = complete disagreement and 5 = complete agreement). In the following, we will comment the different questions and the results obtained.

#### **7.3.1 DESIGN SUPPORT**

For design support we were interested on two different types of questions:

- The questions about the use of the tool in the design process try to determine whether the tool can be useful when applied to this activity,
- Other questions refer to the completeness and validity of the model for design rationale (whether users think that any possible decision can be represented by the elements of the model, and the rationale decisions are correctly represented).

In the following, we will comment the results for the different question

**Q1: Are all aspects concerning issues (Trade-offs) correctly represented?**

Here we obtained 4.6 points on average. That is, a majority of the users fully agree with this conclusion. Some user pointed out that it would be better to have the possibility of sorting the items so that they were always placed in the same order (order in the presentation changes when probabilities are re-computed).

**Q2: Does the tool have a positive impact when used for system's design?**

The average value for this question was 4.8. One user that was working on use case 2 commented that it was probably worth to use the tool once the initial design has been made, to avoid having in the change impact model a very large number of decisions taken during the first stages of the project.

**Q3: Is the tool helpful for setting priorities (in decisions and activities to accomplish)?**

Here we obtained 5 points on average. That is, users fully agree with this conclusion.

**Q4: Is the tool helpful to identify inter-dependencies of decisions?**

The result obtained on average was 4.2 (partial agreement). Two of the users commented that there are cases in which an input (cause) for a decision is an output for another decision, and this could be difficult to visually detect looking at the tree. Users in this case need to open the corresponding CPT for that element. But, although the tree provides a simpler view, it hides the underlying network. This affects the visualization of the BBN, but not the underlying model.

**Q5: Is the tool helpful to identify causes of decisions?**

Here we obtained 4.8 points on average. That is, users almost fully agree with this conclusion. There is no comment on how the model can be improved.



**Q6: There is a benefit of gathering the information related to alternatives in the design, arguments**

Here we obtained 5 points on average. That is, users fully agree with this conclusion.

**Q7: Does the tools serve to determine the traceability among elements and their inter-dependencies?**

All users almost fully agree with this statement, with 4.8 points on average.

**Q8: All elements involved in a design decision can be represented in the model.**

This was a very important question, because it was addressed to determine the completeness of the model. Users fully agreed with this assessment (with average 5.0)

**Q9: The rationale for decisions is correctly represented.**

Also a very important question, addressed to determine the validity for representing design rationale. The obtained value was a 4.6 on average. Once again, there was no comment regarding missing elements for the rationale.

**Q10: The tool can help stakeholders to explain their motivations, and to reach an agreement.**

All users either fully agree or partially agree on this (with 4.8 points on average). Some users commented that, by using probabilities, it is possible to better explain the argumentation behind a decision, and that reaching an agreement on the probability tables was the key factor.

| Design Support |  | Average |
|----------------|--|---------|
| Q1             | All aspects concerning issues (Trade-offs) are correctly represented                             | 4.6     |
| Q2             | The tool has a positive impact when used for system's design                                     | 4.8     |
| Q3             | It is a helpful tool for setting priorities  | 5.0     |
| Q4             | It is helpful tool to identify inter-dependencies of decisions                                   | 4.2     |
| Q5             | It is helpful to identify causes   | 4.8     |
| Q6             | There is a benefit of gathering the information related to alternatives in the design, arguments | 5.0     |
| Q7             | The tools serves to determine the traceability among elements and their inter-dependencies       | 4.8     |
| Q8             | All elements involved in a design decision can be represented in the model                       | 5.0     |
| Q9             | The rationale for decisions is correctly represented   | 4.6     |
| Q10            | The tool can help stakeholders to explain their motivations, and to reach an agreement           | 4.8     |
|                | <b>Global Average on design</b>  | 4.8     |

**Table 7-1: Questions on design and their corresponding average values.**

### 7.3.2 EASE OF USE

A set of questions were oriented towards the ease of use; here we were trying to identify whether the tool was sufficiently simple as to be used by normal developers, as well as possible improvements on usability. Questions were as follows:

#### **Q11: The tool is easy to understand**

We obtained a partial agreement on this point (average 4.2). Some user commented the need to explain some of the acronyms used in the tool (WH, or CPT for instance). Others commented the need for an on-line help. Some user commented that the tool required “minor training”.

#### **Q12: The tool is easy to use.**

Here we obtained 4.4 points on average (almost complete agreement). There was a comment on a user that mentioned the need of “*high experience in project development to define the right probability values used for computation and decision making, even though this definition can be fine-tuned during several iterations*”. Some user commented also the

advantage of the tool to be integrated into an Integrated Desktop Environment such as Eclipse, a very familiar environment for developers.

**Q13: Data gathering does not take so much time.**

This received one of the lowest grades: 3.6. (almost partial agreement) An important comment related to this question was that *“Specification of all elements (Requirements, Goals, Assumption, Desing Particulars), that can affect final trade-offs decision is not an easy task; it is refined step by step during project phases from the very beginning due to tool flexibility. User certain experience is required for this and will be helpful.”*

**Q14: The tool provides relevant information in case of errors**

For this question we received a 3.8 on average. Some user commented that when there were facts added to the model that lead to contradictions (according to the CPTs), probabilities were not computed, and were shown as “-“, without an explanation of the conflict. That is, a CPT in which when a decision is VALID design element DE1 is STABLE, and I set decision VALID and DE1 as VOLATILE, the system failed to compute the probabilities (but there was no mention to the reason why this is happening).

**Q15: The argumentation model (reasons for decisions and causes) is easy to follow**

We had a 4.6 on average; a user commented the (already mentioned) issue that sometimes it is difficult to follow the tree, when there were chained decisions.

The average value obtained for this section is 4.1, we found here again the issue of the tree as a partial but not complete view of the model, the focus on the users regarding the need of expertise for filling the CPTs, as well as some minor possible refinements in the tool (explanation of acronyms, on-line help).

Table 7-2 shows the results for these questions

| Design Support                        |  | Average    |
|---------------------------------------|--|------------|
| Q11                                   | The tool is easy to understand   | 4.2        |
| Q12                                   | The tool is easy to use  | 4.4        |
| Q13                                   | Data gathering does not take much time                                       | 3.6        |
| Q14                                   | The tool provides relevant information in case of errors                     | 3.8        |
| Q15                                   | The argumentation model (reasons for decisions and causes) is easy to follow | 4.6        |
| <b>Global Average on this section</b> |  | <b>4.1</b> |

**Table 7-2: Questions on ease of use and their corresponding average values**

### 7.3.3 MAINTENANCE

Although the tool was not being used for maintenance in any of the two cases, we considered important to ask the users about the possible usage during this phase. There were two different questions regarding maintenance:

#### **Q16: The tool helps developers understand design decisions, thus improving future maintenance tasks**

Here we obtained a 4.8 on average. In particular one user commented that *“This could be true in the case of activities developing products where maintenance phase is of paramount importance”*. Here it is worth mentioning that both use cases were focusing on their specific project and not tied to a particular product to be maintained.

#### **Q17: The tool helps understanding the changes required for better analysis**

We got a 5.00 on average, with no discrepancy on this question.

The conclusion for maintenance is therefore that the tool is seen as valuable for maintenance activities, particularly for maintenance of products that have a larger maintenance period, and are subject to more changes and improvements.

Table 7-3 shows the results for this section.

| Maintenance                           |  | Average    |
|---------------------------------------|--|------------|
| Q16                                   | It helps developers understand design decisions, thus improving future maintenance tasks | 4.8        |
| Q17                                   | The tool helps understanding the changes required for better analysis                    | 5.0        |
| <b>Global Average on this section</b> |  | <b>4.9</b> |

Table 7-3: Questions on maintenance.

### 7.3.4 LEARNING SUPPORT

We asked the users whether they thought this could be useful to allow developers with less experience and know-how to learn from the design decisions modeled using the tool.

#### **Q18 The tool can be used to understand why decisions were made by new members of a project**

Here we obtained an almost complete agreement (4.8). We received a comment stating that *“Post-analysis could be very interesting to avoid wasting time in futures project. Storing the history (how the assumptions and probabilities were changing within the project) could also be interesting”* while others mentioned that *“the tool can provide a quick overview of major criticalities of a project for new members of the development team”*.

#### **Q19: Users can learn how to make future decisions based on the information present**

We received 4.8 points on this point. There was a comment of one user stating that it might be difficult in most occasions to find a situation in which the causes and effects will be the same. Anyhow he considered that having the rationale from the tool was a very valuable asset.

#### **Q20: Some decisions and their outcome can be extrapolated to external projects**

Here we received again a 4.4 (partially complete agreement). Some comments mentioned that this will only be possible for activities that have a strong similarity.

So as a general conclusion for this section, there is a recommendation to store the historical data for the decisions (not only the current picture), and there is a global consensus on the

possibilities to use this information for learning about the project as well as the extrapolation of decisions to other projects. But, this last issue will depend on very similar circumstances, which is not the common case.

Table 7-4 shows values for this section

| Learning support                      |  | Average |
|---------------------------------------|--|---------|
| Q18                                   | The tool can be used to understand why decisions were made by new members of a project | 4.8     |
| Q19                                   | Users can learn to make future decisions based on the information present              | 4.8     |
| Q20                                   | Some decisions and their outcome can be extrapolated to external projects,             | 4.4     |
| <b>Global Average on this section</b> |  | 4.7     |

Table 7-4: Questions on learning support

### 7.3.5 DOCUMENTATION

#### **Q21: The tool generates information that is relevant for projects' documentation**

For this we received an average value of 4.4. The main objection regarding this is the lack of possibilities for printing or exporting the information from the tool to another document.

#### **Q22: Using the tool it could be possible to reduce the volume or number of documents**

For this question we received 3.8 points on average. Some users mentioned that there is no possibility to reduce the amount of documents in some projects (since number and type of deliverables are agreed from the very beginning). Others mentioned that, although there is valuable information in the tool, it was not clear how to achieve this goal.

The main problem being identified for documenting is that it is impossible to export or import information into other formats (pdf document, csv files, etc...)

The global average for this section is the lowest of all sections, being a 4.1 mainly due to this fact.

| Documentation                         |   | Average |
|---------------------------------------|---|---------|
| Q21                                   | The tool generates information that is relevant for projects' documentation     | 4.4     |
| Q22                                   | Using the tool it could be possible to reduce the volume of number of documents | 3.8     |
| <b>Global Average on this section</b> |   | 4.1     |

Table 7-5: Questions on documentation

### 7.3.6 SENSITIVITY ANALYSIS

**Q23: Propagation of probabilities from causes to effects provides a helpful insight of the key issues in making decisions.**

With this question we tried to determine whether users find the forward propagation capability of the tool useful, and whether it gives them hints on those arguments (causes) that are more relevant to decisions. Although there was an almost complete agreement on average (4.8), some users also mentioned the need for an on-line help for some activities (for instance, in order to fill the CPTs).

**Q24: Propagation of probabilities from effects to causes serves to identify the most important elements that contribute in order to obtain an effect.**

We obtained the same result as before (4.8 on average).

**Q25: The tool serves to identify those elements that are critical to achieve our goals**

For this we obtained a complete agreement (5.0). The global average for this section is a 4.9, which means that users see a very good potential for both forward and backward propagation.

| Sensitivity Analysis                  |   | Average |
|---------------------------------------|---|---------|
| Q23                                   | Propagation of probabilities from causes to effects provides a helpful insight of the key issues in making decisions                            | 4.8     |
| Q24                                   | Propagation of probabilities from effects to causes serves to identify the most important elements that contribute in order to obtain an effect | 4.8     |
| Q25                                   | The tool serves to identify those elements that are critical to achieve our goals   | 5.0     |
| <b>Global Average on this section</b> |   | 4.9     |

**Table 7-6: Questions on sensitivity Analysis**

## 7.4 REVIEW OF THESIS'S OBJECTIVES

It is now the time to contrast the objectives we had for this thesis against the results obtained.

| Objective of the thesis  | Global evaluation  | Possible Improvement (s)   |
|--|--|--|
| O1: to develop a model to provide reliable estimations in order to determine the change impact throughout the whole lifetime of a project. | Since the estimation model is based on the change impact analysis performed by the user, the reliability of the model is based on the experience and the ability of the user when performing such analysis. However, forward and backward propagation is considered useful by the users (Q23, Q24) and, what is more important, the tool serves to identify those elements that are critical to achieve our goals (Q25). | Store the history of estimations and use learning techniques to determine accuracy and deviation |
| O2: The APES-CIE Model shall be valid for the aerospace market   | One of the projects used as a use case was an on-board software development. Being the qualitative evaluation performed by the developers involved in this project very positive, and based on the analysis we performed in Section 6.5, the model can be considered valid for this market.  | -  |



| Objective of the thesis  | Global evaluation   | Possible Improvement (s) |
|--|---|--------------------------|
| <p>O3: The APES-CIE model shall contemplate the inherent uncertainty associated to any development. This uncertainty shall be managed using statistical techniques.</p>        | <p>This objective has been achieved; the model allows users to perform what-if and backward analyses and there is a very positive feedback from the users regarding propagation of probabilities (Q23, Q24). In addition, users consider that the tool has a very positive impact for system’s design (Q2). It can be argued that the “probabilities” are, in the majority of the cases, based on impressions from the user and not based on historical data.</p> | <p>-</p>                 |
| <p>O4: The statistical technique to be used by the APES-CIE model shall be based on Bayesian Belief networks</p>   | <p>That is the way within which the model has been designed.</p>  |                          |
| <p>O5: Our change impact estimation model shall conceive the outcome of a software development as a result of a network of design decisions.</p>                               | <p>The main elements of the ontology are design decisions – see Section 5.1</p>   | <p>-</p>                 |
| <p>O6: SLOs, among other elements, shall be linked to design decisions, either as inputs (causes for the decisions to be taken) or as outputs (effects of such decisions).</p> | <p>The ontology considers SLOs as decision elements linked to decisions as inputs or outputs.</p>   |                          |
| <p>O7: It shall be possible to capture the rationale for decisions by using the model,</p>   | <p>The information regarding the decision taken is contained in the model. There is no suggestion from any evaluator regarding elements not contained in the model. There are very positive answers from the evaluators regarding this point (Q4, Q5, Q8).</p>  |                          |

| Objective of the thesis   | Global evaluation  | Possible Improvement (s)  |
|---|--|---|
| O8: The model shall be easy to use by developers  | Q11, Q12 and Q15 induce to think that the results are very positive in this respect.   | Changes in the way the information is presented by the tool, but not changes of the model are required (for instance, to use a network view instead of a tree view) |
| O9: The model shall take advantage of the predictive and diagnostic reasoning capabilities of BBNs.   | Users considered forward and backward propagation a very useful feature (Q22, Q23).  |   |
| O10: Instead of being based on historical data, or automated techniques, the model shall be based on the results of change impact analysis made by users. | This was a conclusion from the state of the art that has driven how the tool works. The tool provides a useful insight of the key decisions, but is based on (and limited by) the user's perspective | Adding automatically information from change impact techniques  |

**Table 7-7: Review of thesis's objectives.**

## **8 CONCLUSIONS**

Change Impact analysis and change impact estimations are a difficult area of knowledge for software development. Although there have been software estimations based on probability calculus for several decades, most of the software development companies still rely on the heuristics from their experts, and not on statistics. The main reason for this is the uncertainty inherent to software changes (the so-called ripple effect: a single change could have multiple effects in different workflows of the software development process) and the unavailability of most of the important variables that are used for estimations until an advanced phase of the development (for instance, LOC). But software changes are inherent to any software development, and unexpected changes come from the very beginning of the development till the maintenance of the software. In fact, any software development can be seen as a sum of changes, some of them being performed in parallel workflows. In this context, BBNs are well defined analysis techniques based on probability calculus that have been used for estimations in multiple areas, that allow estimations under uncertainty and incompleteness of the input parameters.

The conclusions obtained in this thesis can be summarized as follows:

### **C1 Change impact analysis, design rationale and statistical techniques can be combined to handle the uncertainty inherent to any development process**

In the “State of the art” chapter, we identified four knowledge areas of research connected to change impact estimations: change impact analysis (CIA), that predicts the parts of the software system that can be affected by changes in the system; software metric estimations (SE), that try to provide global “attributes” of a development before the actual development is made; Bayesian Belief Networks (BBN), the modelling and statistical technique that we decide to use for change impact estimations; and design rationale models (DR), oriented to capture the knowledge and reasoning justifying the resulting design. We identified that in the majority of the cases, software estimations lack causal modelling, which is an area traditionally covered by DR and CIA. Design changes determine variations in effort, risk and quality during the whole lifecycle of the project. Change impact is tied to change due to

design decisions, which in turn are tied to the rationale for these decisions, and these decisions in turn determine the traceability among SLOs.

**C2: The singularity of change impact makes its estimation more difficult than other kinds of estimations: historic data is not as valuable due to the particularities of any change**

We also identified that the particularities of any software development make it difficult to extrapolate previous results obtained in other projects to the changes to be performed on a given project. Each project has its own, particular “network” of changes, and that traceability among different SLO,s is the driver for this network. We learned that, although there are multiple software tools for helping in the construction of such network, these tools do not provide the complete set of relations that may exist among the different SLO,s.

**C3: Change impact is driven by decisions, which in turn depend on design rationale**

From this point of view, change impact analysis and design rationale are complementary areas of knowledge. Our model provides the rationale behind the changes, so that it can support not only the decision-making process, but the design justification, as well as the change impact analysis.

**C4: Our model is based on previous models, and can be considered an evolution based on them**

From the state of the art, two particular methods and techniques of particular relevance for this thesis were found: the AREL model from Tang, and RATSpeak DRL (implemented in SEURAT) from Burge. None of them satisfied completely our expectations: meanwhile the AREL model focused on the design workflow, the design rationale model for SEURAT, that was able to represent in a much more accurate way the rationale behind decisions, lacked a probabilistic model for the uncertainty regarding SLO;s that are subject to change. We combined ideas and models from both, using a different perspective, to create APES-CIE.

**C5: Common and intuitive concepts (requirements, design elements, goals, assumptions, and environmental aspects) can be combined to create an ontology for design rationale that is easy to use, and understand.**

The APES-CIE model was created having the methodology for space development in mind. APES-CIE uses common concepts of Design Rationale as well as software development standards for the aerospace field to create a model that is intuitive, and easy to use by developers. In this model, for each design decision, represented by trade-offs, users analyze a possible set of design decisions. These design decisions can be mutually exclusive or not. Inputs to the decisions are causes that lead to the decision to be taken, and these are also common concepts used by developers in the development process: requirements, assumptions, design elements, or environmental issues. Outputs to a decision are consequences on elements that can be affected if the decision is taken: design elements built, requirements fulfilled, goals achieved, or assessments that can be assumed to be true. The model allows stakeholders to build the network of elements involved in a change. This network is built manually by the users that are those that have the knowledge, avoiding the need for them to learn the underlying techniques being used. In essence, the tool can be seen as a mind-mapping tool for the design rationale, a tool that is directly connected to the way that some deliverables in the space industry are produced (such as the design justification file). Once users have identified the inputs (i.e. causes) and the outputs (i.e. consequences) for a decision, they will have to assign probabilities for the decisions (based on inputs) and outputs of decisions (based on decisions) by filling conditional probability tables (CPTs). The initial setting of the tool for the CPTs is to provide equal probability for all cases. Users must adjust the corresponding CPTs for the inputs to a decision as well as the output for a decision based on their knowledge. Users can also set those facts that they know, fixing the values for: requirements that are known to be part of the development, assessments that are known to be true, decisions that are enforced, or environmental conditions (for instance, having to work with a given number of developers, necessarily). As CPTs are being adjusted, and facts are set, the model is able to propagate automatically the

probabilities for the effects (forward propagation) as well as the probabilities for the causes (backward propagation) allowing both predictive and diagnostic reasoning.

**C6: APES-CIE provides some advantages with respect to previous models: ease of use, trade-offs to associate decisions, estimations of effort based on the most probable decision, which combined with forward and backward propagation allows for better estimations of changes in projects**

Our model can be considered an extension to the AREL model, since it spans through all the workflows associated to project developments, including requirements, assumptions, environmental issues and goals, that are not present in AREL. It also includes the notion of trade-off, with a set of associated design decisions (something that is also not present in AREL). Moreover, the model has an underlying effort estimation based on the most probable path of decisions, something that cannot be found in AREL. Finally, the model was embedded in such a way that there is no need from the user's point of view to have previous knowledge of the underlying probability-based techniques. All these issues represent clear advantages with respect to AREL. With respect to SEURAT, Seurat does not use BBN technology, and therefore our model has an underlying probabilistic model of change impact that can work even in the presence of uncertainty, with forward and backward propagation. Therefore our model takes benefit of the advantages of BBN techniques. In addition, our model for design rationale is much simplified with respect to DRL, focusing on the ontology being used for the space development, an ontology that can be used also in other development environments, particularly control systems. The APES-CIE model, in that sense is extended w.r.t AREL, but is much less generic than SEURAT, being particularized to the way that software is produced in the space industry. The model allows also for the user to provide estimates of the effort required for the design elements (a.k.a. artifacts), estimating also the effort required for each decision, considering the most probable option to be selected.

**C7: The APES-CIE Model can be used as a mind-mapping tool**

Our model combines both ideas, mind mapping and BBN technology, and is aimed to:

- Help the user to organize its ideas on the different elements involved in a design.

- Determine the impact of the design decisions on the deliverables of the project, the requirements, the goals as well as the design items
- Help the user on which decision to take for obtaining the desirable effects.
- In addition, the tree view eases the determination of the traceability for the different elements, since it is very easy then to determine the relations between design elements and requirements, and vice-versa.

For space applications, the information filled in the tool can be the majority of the information contained in the Design Justification File of the ECCS standards. These capabilities provide the user with a powerful set of tools for global estimations on a project, based on BBN software. This is the ultimate goal for the APES tool. This is possible thanks to the Bayesian Belief Network technology, which is used by the tool, and is conveniently “hidden” to the user, so that the user does not need to have knowledge of this technology in order to use it.

**C8: Evaluation via use cases shows that the model is easy and intuitive to use, but the tool lacks some capabilities that would ease its usage and adoption by development teams**

From the evaluation phase, we have identified the main pros and cons of the tool and the model from the user’s perspective. We have seen that, although the tool is considered easy to use and intuitive, and users agree on the model for design rationale, there are some aspects of the tool (tree view, acronyms, lack of on-line help, export and import capabilities of data, lack of historical data storage) that need to be refined. But all these aspects do not refer to the model itself, nor to the validity of the concept (using Bayesian belief network technology as a support for design rationale and change impact estimation). That’s why we will continue improving both the model and the tool in the future.

## 9 FUTURE WORK

There are many possible areas of future work that can be explored. APES-CIE is a Change Impact Estimation model that provides at the same time estimations on changes, a tree view of the Change Impact Sets of a given project, as well as the underlying design rationale. As such, there are four possible areas of improvement:

- As already commented, the most important feature of these models to be successful is its ease of use. As was already pointed out in the review of thesis’s objectives, the tool that we have developed still has some usability aspects that can be improved: for instance, system requirements are normally stored into a database, using a requirements tool. A possible improvement could allow these requirements to be directly imported into the tool. The same applies to the design elements that are normally contained in the UML tool used for design. This first area of improvement would focus on *importing into the model those decision elements that already exists, without the need to replicate them (basically, design elements and requirements)*. Other mentioned improvements by users are: an on-line help to guide the users, the possibility to expand the tree view so that all related decisions can be seen at a glance, or the storage of historical data to see the evolution of the probabilities and the causes for decisions.
- A second, possible improvement could be in *modifications and extensions to the CIE model*. We have mentioned that we used the AREL model as a starting point, in which decisions have two possible values: either VALID or INVALID. But also in the example provided we have seen that there are decisions that are *mutually exclusive*. A further refinement of the model will extend the possible statuses of a decision, to be not only VALID or INVALID, but also consider each alternative decision as part of its statuses (f.i., in the example of Section 6.4, the “*memory tool to use*” trade-off will contain one single decision, with 4 possible statuses: “Java”, “MsAccess”, “Tcl/tk”, or None). This will affect only mutually exclusive decisions, and not those decisions that



are not related to each other. Other improvements could also be possible, like for instance the identification of patterns in design decisions.

- A third improvement area would focus on *possible connections of the model with the outcome of change impact analysis tools*, like those mentioned in Section 3.1. Some of these tools analyze existing code or SLO,s and provide the traceability in an automated way. Their outputs could be used to build the DAG associated to decisional elements and its related decisions automatically.
- Finally, the last area of improvement that we foresee is the automatic generation of project documentation based on the existing CIE model. By doing so, documents like the risk register, the Design Justification File, or the cost justification analysis that is traditionally provided in change contract’s notices, could be partially generated from the information stored in the APES system.

## 10 BIBLIOGRAPHY

- Angelis, L., Stamelos, I. and Morisio, M. 2001.** Building a Software Cost Estimation Model Based on Categorical Data. 2001.
- Arnold, R.S: and Bohner, S.A. 1993.** Impact Analysis – Towards a Framework for Comparison. Los Alamitos, Calif : IEEE CS Press, 1993, pp. 292-301.
- Basha, S and Dhavachelvan, P. 2010.** Analysis of Empirical Software Effort Estimation models. 2010, pp. 68-77.
- Basri, Sufyan, Kama, Nazri and Ibrahim, Roslina. 2015.** A Novel Effort Estimation Approach for Requirement Changes during Software Development. 2015, pp. 237-252.
- Bilal. 2006.** "Computing Ripple Effect for Object Oriented Software". 2006.
- Blac, S.E. 2001.** *Computation of Ripple Effect Measures for Software. PhD Thesis,*. London, UK : London South Bank University,, 2001.
- Boehm, B. 1981.** *Software Engineering Economics.* New-York : Prentice-Hall, 1981.
- Boehm, B., Abts, C. and Chulani, S. 2000.** Software Development Cost Estimation Approaches: A Survey. 2000, pp. 177-205.
- Boehm, Barry, et al. 1995.** Cost Models for Future Software Live Cycle Processes: COCOMO 2.0. *Annals of Software Engineering Special Volume on Software Process and Product Measurement.* Amsterdam, The Netherlands, : J.C. Baltzer AG, Science Publishers, 1995, pp. 45-60.
- Bohner, S.A. and Arnold, R.S. 1996.** *Software Change Impact Analysis.* IEEE Computer Society Tutorial : IEEE Computer Society Press, 1996.
- Bracewell, R, et al. 2009.** Dred 2.0: A method and tool for capture and communication of design knowlede deliberated in the creation of technical products. 2009, pp. 6-223, 6-234.
- Burge, J. and Brown, D.C. 2002.** Discovering a research Agenda for using design rationale in Software Maintenance. 2002.
- Burge, J. E. and Brown, David C. 2008.** *SEURAT: Integrated Rationale Management.* Leipzig, Germany. : ACM 978-1-60558-079-1/08/05., 2008.
- Burge, J. E. 2005.** Software Engineering Using design RATionale. 2005.

- Burge, Janet E. and Brown, David C. 2003.** Rationale Support for Maintenance of Large Scale Systems. 2003.
- Burge, Janet E. 2001.** Design Rationale for Software Maintenance. 2001.
- Clayberg, Eric and Rubel, Dan. 2008.** *Eclipse Plugins (3rd Edition)*. s.l. : ISBN-13: 978-0321553461, 2008.
- Clayberg, Eric. 2008.** *Eclipse Plug-ins (3rd Edition)*. 2008. ISBN-13: 978-0321553461.
- Conklin, Jeffrey E. and K.C., Burgess Yakemovic. 1991.** A Process-Oriented Approach to Design Rationale. 1991, pp. 357-391.
- ECSS-E-40, Part 1B. 2003.** *European Cooperation for Space Standardization (ECSS). Space Engineering. Software – Part 1: Principles and Requirements*. 2003.
- ECSS-E-40, Part 1C. 2009.** *European Cooperation for Space Standardization (ECSS). Space Engineering. Software – Part 1C*. Noordwijk, The Netherlands : s.n., 2009.
- Feiler, Peter H. and Gluch, David P. 2012.** *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. s.l. : Addison-Wesley Professional., 2012.
- Fenton, N. E. and Ohlsson, N. 2002.** Quantitative Analysis of Faults and Failures in a Complex Software System. 2002.
- Fenton, N.E. and Neil, M. 2000.** Software Metrics: Roadmap. 2000.
- Fenton, Norman and Neil, Martin. 1999.** Software Metrics and Risk FESMA 99. 1999.
- Fuentetaja, R., et al. 2013.** Multi-step Generation of Bayesian Networks Models for Software Projects. 2013.
- Gamerman, Dani. 1997.** *Markov Chain Monte Carlo, Stochastic Simulation for Bayesian Inference*. London (UK) : Chapman & Hall, 1997.
- Geman, S. and Geman, D. 1984.** Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images: 609-628. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12. 1984.
- Gilb T. 1976.** *Software Metrics*. s.l. : Chartwell-Bratt, 1976.
- Gray, Andrew R. and MacDonell, Stephen G. 1997.** A comparison of techniques for developing predictive models of soft-ware metrics. 1997, pp. 425-437.

- Hall, M., et al. 2009.** The Weka data mining software: an update. 2009.
- Hart, P.E: and Duda., R.O. 1977.** PROSPECTOR. A computer based consultation system for Mineral Exploration. 1977.
- Hassan, Mohamed Oussama and Basson, Henri. 2008.** "Tracing Software Architecture Change Using Graph Formalisms in Distributed System". 2008, pp. 1-6.
- Hassine, J., et al. 2005.** Change Impact Analysis for Requirement Evolution Using Use Case Maps. Washington, U.S.A. : s.n., 2005.
- IEEE Glossary. 1990.** *IEEE Standard Glossary of Software Engineering Terminology. September, the 28th.* s.l. : ISBN 1-55937-067-X., 1990.
- IEEE. 2000.** *IEEE Recommended Practice for Architectural Description of Software Intensive Systems.* s.l. : Software Engineering Standards Committee of the IEEE Computer Society, 2000.
- Jensen, Finn V. 1996.** *An Introduction to Bayesian Networks.* London (UK) : UCL Press, 1996.
- Kama. 2013.** Change Impact Analysis for the Software Development Phase: State-of-the-art. 2013, pp. Vol. 7, No. 2.
- Kjærulff, Uffe B. and Madsen, Anders L. 2005.** *Probabilistic Networks — An Introduction to Bayesian Networks and Influence Diagrams.* 2005.
- Korb, Kevin B. and Nicholson, Ann E. 2004.** Bayesian Artificial Intelligence. Florida (USA) : Chapman & Hall/CRC, 2004.
- Kruchten, P. 2000.** *The Rational Unified Process: An Introduction.* s.l. : Addison-Wesley ISBN 0-201-70710-1, 2000.
- Kunz, Werner and Rittel, Horst. 1970.** *ISSUES AS ELEMENTS OF INFORMATION SYSTEMS.* 1970.
- Laird, Linda M. and Brennan, M. Carol. 2006.** *Software Measurement and Estimation: A Practical Approach.* s.l. : Wiley Interscience, 2006.
- Lee, J. 1997.** *Design Rationale Systems: Understanding the issues.* 1997, Vol. IEEE Expert, pp. 78-85.

- . 1989. Decision Representation Language (DRL) and its support Environment. 1989, Vols. MIT AI Lab, Working Paper no 325, August.
- Leung, H. and Fan , Z. 2002.** Software Cost Estimations. 2002.
- Long Parnas, D and Clements, P. C. 1986.** A Rationale Design process: how and why to fake it. 1986, pp. 251-256.
- MacLean, Allan, et al. 1991.** Questions, Options and Criteria, Elements of Design Space Analysis. 1991, pp. 201-250.
- Madachy, R.J. 1994.** *A software Project Dynamics Model for Process Cost, Schedule and Risk Assessment*. s.l. : Ph.D. dissertation, University of Southern California, 1994.
- Mair, Carolyn, et al. 2000.** An investigation of machine learning based prediction systems,. 2000, pp. 23-29.
- McDermott, J. 1984.** R1 re-visited: 4 years in the trenches. 1984, pp. 21-32. .
- Molokken, K. and Jorgensen, M. 2003.** A review of software surveys on software effort estimation. 2003, pp. 223-230.
- Morgan, Jeanette N. and Peeples, A. 2003.** A Software Development Cost Estimation Model for Higher Level Language Environments. 2003.
- Netica. 2008.** *Netica-J Manual*. Vancouver, BC, : Norsys Software Corp., 2008.
- Noble, D. and Horst, W.J. Rittel. 1998.** Issue-Based Information Systems for Design. 1998, pp. 275-286.
- Ocón, J. 2010.** *Cost, Complexity and Change Impac Final Report (Deliverable of the CCI Project)*. Madrid, Spain : s.n., 2010.
- Pearl, Judea. 1998.** *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco (USA) : Morgan Kaufmann, 1998.
- Pleeger, S.L. 1991.** *Software Engineering: the production of quality software*. New York : MacMillan Publishing Company, 1991.
- Quinlan, J.R. 1992.** Learning with continuous classes. *Proceedings AI'92*. Singapore. : Adams & Sterling Eds., 1992, pp. 343-348.
- RADC. 1986.** "*Automated Lifecycle Impact Analysis System*". Rome : Rome Laboratories, 1986. Tech Report RAD-TR-86-197.

- Radlinski, L., et al. 2007.** Improved Decision-Making for Software Managers Using Bayesian Networks,. 2007, pp. pp. 13-19.
- Samad, T. and Annaswamy, A. M. 2011.** *The Impact of Control Technology*. s.l. : IEEE Control Systems Society, 2011.
- Schubanz, M., Pleuss, A. and Botterweck, G. 2012.** *Modeling Rationale over Time to support Product Line Evolution Planning*. Leipzig, Germany : s.n., 2012.
- Shawn. 1996.** *Software Change Impact Analysis*. Los Alamitos, California., : IEEE Computer Society Press, 1996.
- Shortliffe, E. H. 1976.** *Computer-based medical consultation: MYCIN*. New York (USA) : Elsevier, 1976.
- Shum, S. 1991.** *Cognitive Dimensions of Design Rationale. People and Computers VI*. Cambridge : Cambridge University Press, 1991.
- Smith, J. Q. 1992.** *Decision Analysis. A Bayesian Approach*. Bristol (UK) : Chapman & Hall., 1992.
- Stevens, W., Meyers, G. and Constantine, L. 1974.** *Structured Design, J. VOI 13 NO 2*. s.l. : IBM Systems, 1974.
- Tang, A., et al. 2005.** A Survey of the Use and Documentation of Architecture Design. 2005.
- Tang, Antony, et al. 2006.** Using Bayesian belief networks for change impact analysis in architecture design. 2006, pp. 127–148.
- Weisberg, S. 1985.** *Applied Linear Regression*. New York : John Wiley and Sons, 1985.
- Y. Li, J. Li, Yang, Y. and Mingshu, L. 2008.** Requirement-centric Traceability for Change Impact Analysis: A Case Study in Making Globally Distributed Software Development a Success Story. 2008.
- Yau, S. S. and Collofello, J.S. 1980.** Some Stability Measures for Software Maintenance. 1980, pp. 542-552.
- Zhao, J. 1998.** Assessing the Complexity of Software Architectures. 1998, pp. 163-166.