6

User-perceptions Of Embedded Software Reliability

R.J. Kusters, R. van Solingen, J.J.M. Trienekens, H. Wijnands
Eindhoven University of Technology, Department of Technology Management
Address: P.O.Box 513, 5600 MB Eindhoven, The Netherlands, Tel.: +31 40 247 3703, Fax: +31 40 243 3612, E-mail: rku@tm.tue.nl

Abstract

Many researchers and practitioners have recognised that the perception of 'reliability' is largely influenced by personal view and application context. Depending on personal goals, interests and background, the interpretation of the reliability concept is different per individual. Industrial organisations try to fulfil user-needs regarding reliability but those reliability requirements are not as unambiguous as they should be. Therefore most industrial organisations fall back to a zero-defects approach, which implies a focus on fault detection instead of fulfilling user requirements. This in itself is not sufficient. In this paper, an approach is suggested by which the different user perceptions on reliability are modelled, and can be addressed. This Multi-party Chain model supports the explanation of the various views on product reliability of all users. Discussion among the parties involved is therefore enabled. Based on consensus the appropriate measures can be selected. The model should operationalise the basic relation between process and product, it enables the definition of metrics to evaluate process improvements, and links engineering activities to user requirements. The usefulness of the model is validated in a case-study.

Keywords:

Embedded Software, Reliability, Users, Multi-Party Chain Model, Software Process Improvement

D. Gritzalis (ed.), *Reliability, Quality and Safety of Software-Intensive Systems*

1 INTRODUCTION

An information system, especially a complex embedded system, does not end up being reliable just by accident. Often much effort and resources are spend to achieve reliability. The developers, often of many disciplines, each contribute in the development of a reliable product. In their design and implementation, reliability is a self-evident ambition which forms the basis of good engineering practices. As such, reliability can be seen as the result of a large number of explicit and implicit control activities, and measures, often deeply implanted in the professional engineering knowledge and actions of the experts involved.

In practice however, reliability is not always easy to achieve. Two common problems will be mentioned. Firstly, building functionality within plan and budget is often given higher priority than achieving quality attributes such as reliability. Secondly, reliability of the complex systems of today must be achieved by combining the experience and knowledge of the disciplines involved. Combining disciplines means co-operation of experts from different fields, probably causing communication difficulties. Human failures become more likely and can easily lead to less reliable or even unsafe systems.

In this paper we present an approach which puts user expectations on product reliability as being the ultimate requirement to be fulfilled. An information system should either fulfil all requirements as desired by all parties involved or it should be made clear that some of these desires cannot be fulfilled within the existing constraints. This statement entails at least two types of problem:

1. Requirements are not always made explicit by the people involved, but they do expect the system to adhere to them none the less.
2. We are most of the time dealing with many different involved parties. Differences between users occur related to for example a persons job, responsibility and their specific use of the system. Also individual preferences exist based on intellectual skills, background or interests. The result of these differences is that 'reliability' is being defined differently among the different users.

As a result we have to recognise that reliability is a *multi-dimensional* concept, that it means many things for many people. This concept is explored further in this paper. First we will take a look at the existing definitions of reliability. In section 4 we will develop the multi-dimensional concept of reliability. Based on this, in section 5 a control model for defining and implementing reliability is presented. This model is developed further in section 6 where the multi-party chain approach is presented. This approach supports tackling the problem of identifying and controlling reliability. It is based on modelling the organisational chain for a product life-cycle and relating the different parties, roles and requirements to explicit actions to be taken during development. Our approach explicitly defines reliability requirements for all users, and facilitates communication of those reliability requirements among all parties involved. Therefore it seems highly applicable for application in multi-disciplinary development teams.

A data model to support this model is also presented in section 6. In section 7, by means of a case study, the usefulness of the approach is illustrated. The paper ends with some conclusions and our plans for further research.

2 RELIABILITY CONCEPTS

In this section we will present two definitions of software reliability taken from the two most accepted standards: the ISO 9126 standard and the IEEE Software Engineering Standards Collection (IEEE, 1994).

2.1 ISO 9126

As reliability is one of the main characteristics of 'quality' according to the ISO 9126 (ISO, 1991) standard, it seems wise to look to this standard in more detail. The ISO 9126 standard sub-divides software reliability into three sub-characteristics:

1. *Maturity*: attributes of software that bear on the frequency of failure by faults in the software
2. *Fault tolerance*: attributes of software that bear on its ability to maintain a specified level of performance in case of software faults or of infringement of its specified interface
3. *Recoverability*: attributes of software that bear on the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.

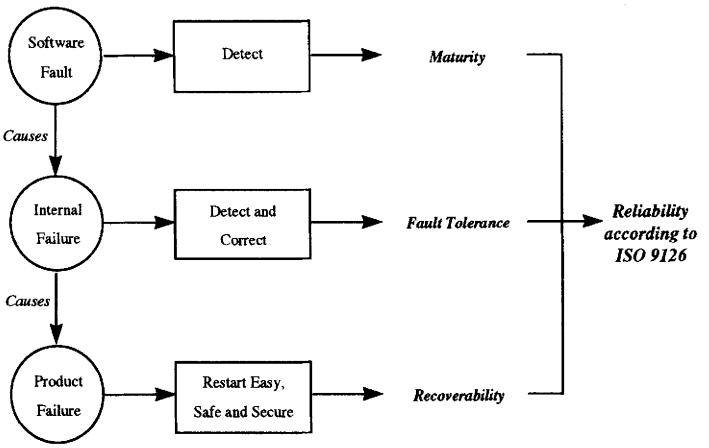

Figure 1 An interpretation of reliability according to ISO 9126.

Summarising those three sub-characteristics: Software must be operational to fulfil user requirements. That implies that design faults (bugs) must be taken out which results in *maturity*, because undetected faults may cause a product to fail. It also implies that a product should be *fault tolerant:* meaning that faults are caught in time by the product itself and failures are prevented. Finally, whenever a failure is occurring it is most important to get the product operational again. This is called *recoverability*.

2.2 IEEE Software Engineering Standards Collection

Software reliability is defined by the IEEE as *'the probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to, and use of, the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered.'*

2.3 Conclusion

Reliability according the ISO 9126 standard is emphasising *operation* of a product: an embedded product is not allowed to 'go down'. This definition implies that having a product operational automatically means that all reliability requirements are fulfilled, which is not obvious. Trying to keep the 'mean time to failure' as high as possible is a good thing, but it must be kept in mind that failures will occur. Therefore measures must be taken that make sure that product failure is foreseen and will not result in unsafe or insecure situations. This implies that concepts such as 'graceful deterioration' should also play an part in assessing reliability.

The IEEE definition adds to the ISO 9126 definition, in that it explicitly defines 'use' as being a cause for failure. It is more limited in that it only looks at the 'mean time to failure', while ignoring the 'mean time to repair', which is just as important an aspect of reliability.

Both definitions are made for engineers by engineers and provide specifications in engineering terms that can be used as part of a design contract. In doing so, they both provide a vital service. Developing information systems, especially complex embedded systems, will usually take place on the basis of a set of well worked out requirements.

However, the very nature of these solid technical definitions also provided a problem. The greatest objection to this type of definition is that principals and potential users are not supported in the formulation of their reliability requirements (see e.g. (Kaposi and Kitchenham, 1987). It is not clear how an organisation can determine the level and type of reliability required that is relevant in its specific situation and in which language this can be formulated (Trienekens, 1992). This implies the need for a wider look at reliability than is provided by either ISO or IEEE. This idea is explored in the next section.

3 RELIABILITY: A WIDER OUTLOOK

If we look up the word 'reliability' in normal English dictionaries, we usually find a meaning that has connotations with ideas such as 'trust' and 'confidence'. It will be clear that those (technical) standards have translated the concept of reliability into a definition which, although

verifiable, has lost the more emotional concepts *trust* or *confidence*. The relationship between the different types of use the system will see and the consequences this will have for its required level of reliability on the one hand and the formalisation of the notion reliability in the ISO and IEEE definitions is not evident anymore and will certainly be difficult to understand for the people (users) involved.

This illustrates the difficulty for developers to fulfil a common stated product requirement that a product must be reliable. It seems logical that developers tend to translate a reliability requirement into: *defect-free*. Many efforts are being spend to reduce the number of defects in software to an acceptable level, before the product is put into use. We find examples in recent literature that claim the possibility to *estimate software reliability* based on statistical models (see e.g. Fenton and Pfleeger, 1996). Those approaches may be well applicable for fault and failure estimation under experimental conditions, but don't address the human perception of reliability. After introducing a system that rarely fails and completely satisfies the agreed upon reliability requirements, many users still start complaining that the product is not what they expected, while others are highly satisfied.

Different parties have different ways of looking at a product from the point of view of reliability. It might theoretically be feasible to translate all these different 'visions on reliability' into one common definition which can then be used to communicate between all those parties involved. This seems the objective of the above discussed standards. In practice however, we deem this to be not feasible. Even if this common terminology could be developed, it is unlikely that it would provide a sufficient basis for discussion with the various parties involved, even if they were willing to adopt it (which seems unlikely). We will therefore take the approach in which explicit account is being taken of locally defined assessments of reliability.

In this paper we assume that product reliability is related to its *use*. This brings us to the conclusion that to create a reliable product one must first define the function it is intended for, and collect all local definitions of its reliability. The required level of reliability can now be derived from the characteristics of the situation within which the system has to function (Heemstra et.al., 1995). Another approach is to assume that product use is related to the responsibility of the users involved. Different parties, i.e. stake-holders such as buyers, users, project principals, developers etc., have different requirements regarding reliability. Reliability appears to be a multidimensional concept, both in theory and in practice. As a consequence different concepts are used to express reliability, different measures are carried out to determine and/or to establish reliability, and different standards are used to validate and to evaluate the reliability of a product.

This multi-dimensional concept of reliability, a described above, is not intended to displace the more traditional reliability concepts of ISO and IEEE. These will still be needed to provide accurate specifications for development engineers. However we do claim that a wider look at reliability is needed, if one wants to be able to define the type and level of reliability required in such a way that users both understand and agree to what is being proposed. We will take this premise as a starting point when designing a control model for implementing reliability.

4 A CONTROL MODEL FOR IMPLEMENTING RELIABILITY

For product development various activities are needed. A typical breakdown of activities for the development of embedded software systems is:

- initiation
- specification
- design
- implementation

The development activities produce the system and thereby will determine the system characteristics, including its reliability. Further phases in the life cycle, such as operation, maintenance and disposal will also change the system reliability in time. For simplicity, we will not discuss this and focus on the development phase of the life cycle, since those phases are assumed to have a higher impact.

Within the development process, several activities can be noted that specifically contribute to the reliability of the system. If we make this activities explicit they can be seen as control measures on all activities within the development process. We can depict this in a control model (figure 2).

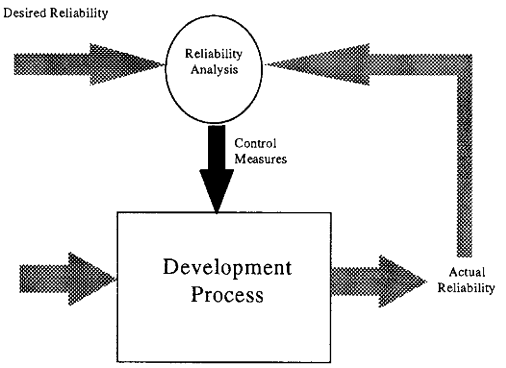

Figure 2 reliability control model.

In this model the flows of information and corresponding processes are:

- *desired reliability* is the reliability as stated or intended by the different users, the law or by ethics.
- *control measures* are all required activities that should take place in the development process in order that the actual reliability will equal the desired reliability. These control measures are selected on the basis of *a reliability analysis.*
- *actual reliability* is the reliability of the end product or the intermediate results of the development process, as perceived by the actual user of that product.

The goal of any development project is to create a system that conforms to a pre-specified level of reliability. Depending on both the type of reliability required as well as the level required a certain amount of measures are selected to result in the required product reliability. In order to select the right mix of measures from the numerous possible options, and to achieve a closed control loop we need to be able to assess the result obtained (actual reliability) to be able to determine if the proper mix of measures was selected in order to satisfy the required reliability, which has to be stated explicitly. More in detail:

1. Assessment of *actual reliability.* Product use is the ultimate reliability check, but during development specific data collection appears to be applicable to give a representation of product reliability. Keeping in mind that control measures will be taken, it seems logical to include data collection activities to track whether those measures give their intended effect.
2. Selection of *control measures* during the development processes. To select the appropriate measures to fulfil reliability requirements, a supporting method is required.
3. Make the *desired reliability* explicit while taking into account the multi-dimensional nature of the concept of reliability as was discussed in the previous section.

The one thing to be kept in mind is that whenever a product must be reliable, measures are required to result in the required level of reliability. Selection of measures is a difficult topic, and is done in practice mostly implicit and/or based on experience. What is required in both practice and science is a technique to describe the user-perceived reliability requirements for their specific context of use. Whenever this description is available measures can be selected with a clear goal, and evaluation (read: measurement) can be implemented. Such a methodology is not found in literature or practice, but will be described in the next chapter: the Multi-Party Chain model.

5 MULTI-PARTY CHAIN MODEL

Measures are considered to be activities/actions that aim to increase the reliability of an embedded software product. Both measures as well as the effect they will have will tend to be expressed in an engineers' language. Given this, communication between stake-holders is essential in order to achieve mutual understanding and consensus between the different parties involved which has to lead ultimately to agreed upon operational measures and metrics to realise and quantify the reliability of products.

To enable this communication the different interpretations of reliability have to be understood. Given a specific product, if it is known which party specified a specific requirement for this product and especially for which reason, then a basis for discussion has been created. If this specification is available it becomes possible to translate different interpretations and it becomes possible to negotiate and discuss those specified requirements. In the end, this enables us to link specific measures (in engineering terminology) to requirements of the different parties involved as expressed in their local terminology.

In this chapter we will present a model to facilitate the identification of all parties involved and of their concerns with reliability. This model, the 'multi-party chain' model (MPC), can be used in a specific situation to identify concerns and to facilitate communication among the parties involved.

The different parties that play a role in determining and realising reliable products, their views on what reliability should be and their mutual dependencies can be represented in a "multi-party chain".

In a data model of this multi-party chain (MPC) the main entity types and their interrelations are represented. The data model supports the identification of the different parties involved, their interrelations, their roles, their requirements etc. The focus of the data model is on the determination, the realisation and the evaluation of reliability in embedded software products by taken the right measures. By taking a specific measure, one aims at increasing one (or more) aspect of reliability. The key question is therefor which set of measures suit best to the set of requirements for a specific product.

The data model that represents this approach will be shown in Figure 6. We will not impose this model upon the reader directly, but we will transfer the control model of the previous section stepwise according to the points 1-3, into the MPC-model.

5.1 Assess the actual reliability of the output from the development process

The first step to take is to evaluate the actual reliability of the product. Therefor we need to perform data collection on that product. Theory on software metrics has learned that metrics on the total development process are required in order to gain insight into actual reliability of a product. Figure 3 shows the data model of the development process. For simplicity we have defined that a development consists of a unique set of actions taken in a specific order. Those actions (measures) are taken on the product, the process, the resources and on the project management. Evaluating the actual reliability of a product will consist of data collection on all of those entities.

The main entity types in the Figure 3 data-model and their definitions are:

- *Measure*: Activity that influences the type and/or degree of reliability that is attained in a given product
- *Project management*: Those activities that guide the execution of the project
- *Resources*: Those resources that are used during project development and use
- *Product*: An imbedded software product seen as part of a larger system
- *Process*: The process of developing, maintaining and managing the product

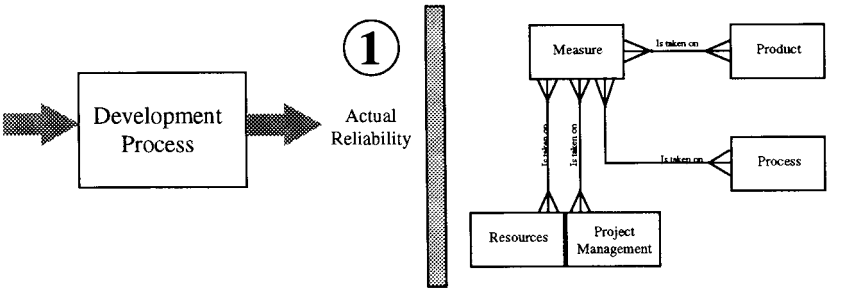

Figure 3 Data model of the development process.

5.2 Select control measures to fulfil reliability requirements

The second step to take is the definition of control measures that aim at fulfilling specific reliability requirements. Those requirements can be very diverse like: all software faults in the user-interface have been removed, or the most critical part of the system is fault tolerant. A certain reliability analysis has to be executed on which the measures can be selected. Figure 4 shows the data model in which control measures to fulfil reliability requirements are opposed upon the development process. Note that the data-model of Figure 3 is included in this figure.

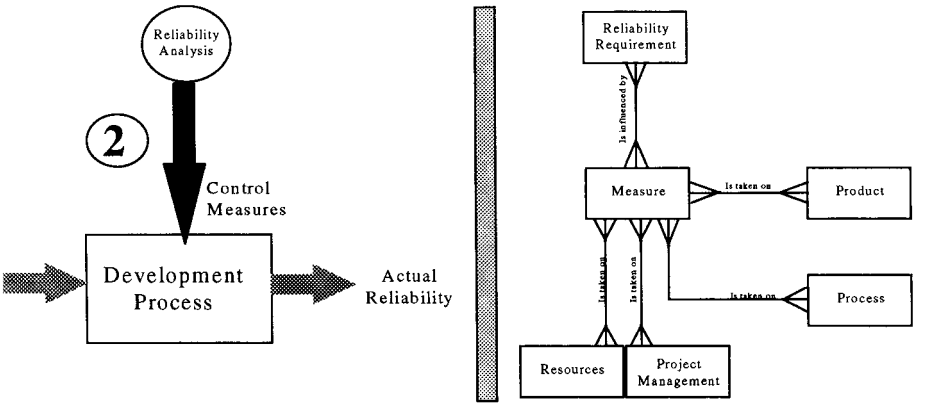

Figure 4 Data model of selection mechanism to fulfil reliability requirements.

The new entity type in this data-model and its definition is:

- *Requirement*: A statement regarding the type and degree of reliability that is required from the product as seen from the point of view of a party in a given role

Requirements have to be expressed in terms of a specification. Examples of reliability requirements of users: 'the system may not fail (go down) more than two times a month', 'when the system fails (goes down) an engineer must be able to fix it within one hour', or 'I will not have my customers annoyed by this system'. The first requirement can be expressed (in 'engineering language') into the reliability aspect "availability". The second requirement can be expressed into the reliability aspect "recoverability". The last requirement will need further discussion and clarification in order to find out what is really required, so that this requirement can be translated into requirements in the 'engineering language'. Consequently, measures and metrics (pertaining to project management, process, resource, and/or product) will have to be determined.

5.3 Make the desired reliability explicit

The final step to take is to make the desired reliability of the system explicit. As described before we will apply an approach in which all local definitions of all users are captured. Therefore a modelling technique is required which models all users and their view on reliability. As described before we focus on parties, underlying roles and related reliability requirements. Figure 5 shows the data-model by which we describe the chain of parties involved. In this way it becomes possible to make reliability requirements explicit and it is possible to trace where they are originating from, based on roles and parties.

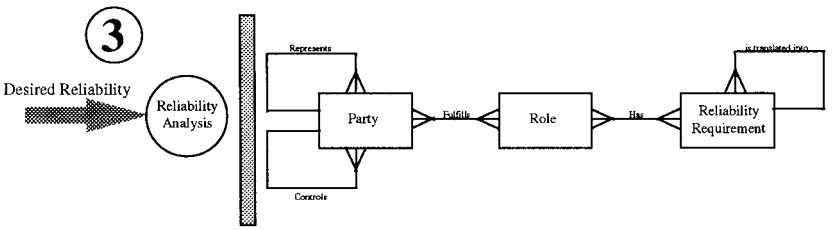

Figure 5 Data model to make reliability requirements explicit from the context of use.

This is a 'chain' model, because we assume that a number of customer-supplier relationships exist between the different parties. An important relationship is the 'representation' relationship. Take for instance a system for controlling a fuel pump. Some relevant parties are customers, fuel station managers, oil company managers, oil company purchasing officials, and vendor representatives. We see here a chain of representation. The fuel station manager represents the interests of the customer. The oil company represents the interests of both the customer and the fuel station manager. The oil company purchasing official represents al of

those. Also, The oil company purchasing official influences (and is influences in his turn by) the vendor representative. To chart this area a 'chain' approach is taken.

The main entity types in this part of the MPC data-model and their definitions are:

- *Party*: An identifiable person, group of persons of organisation who has a legitimate interest in the degree of reliability of the product
- *Role*: An area of responsibility of the party determining a view on the type and degree of reliability required
- *Requirement*: A statement regarding the type and degree of reliability that is required from the product as seen from the point of view of a party in a given role

From the 'chain' point of view (i.e. parties and their links, their roles and their requirements) the most interesting entity types are respectively: the entity type 'Party', the entity type 'Role' and the entity type 'Requirement'. Based on an identification and specification of these three entities measures can be determined and carried out. Consequently the effectiveness of these measures can be validated and evaluated within the previously determined 'party-context' (i.e. party, view, role, responsibility, goal etc.). We will discuss the three entity types 'party', 'role' and 'requirement' in more detail below.

A *'party'* was previously defined as an identifiable person, group of persons of an organisation who have a legitimate interest in the degree of reliability of the product. This 'legitimate interest' may come to light in one or more phases of the product life cycle. Examples of parties are: principals, customers and users (of different types, e.g. employees from marketing, purchasing, production etc.), project managers, engineers, maintenance people. Parties can also be stake-holders that are not directly involved in one or more specific phase of the life cycle, for instance governmental parties that develop and control policies, rules and constraints for reliability of branch organisations that set their specific requirements.

A *'role'* was defined as an area of responsibility of a party that determines which view exists on the type and degree of reliability required. A party can have more than one role. Example from Schlumberger, an producer of petrol dispensing machinery: The Schlumberger OPCO (the national sales and service organisation) has the responsibility to represent Schlumberger towards customers and vice versa (customers towards Schlumberger), but also to represent the national government of the country it is servicing and their policies towards Schlumberger. So the OPCO has several roles to play in communication on reliability.

Relationship types relevant to the chain approach are:

- Parties may have different responsibilities. This is represented by the different 'roles' that a party may assume. This relationship type allows an inventory of reliability requirements which are relevant to this party through the intermediate entity type 'role'.
- Parties can represent, both officially and unofficially, other parties.
- Finally parties may control and/or influence other parties. The last two relationship types allow us to build the multi-party chain by identifying the links between the different parties.
- Roles may be linked to several parties which in that case would be expected to have identical reliability requirements.
- A role may be concerned about one or more reliability requirements.

- A reliability requirement can be linked to several roles. In that case these roles, from a different motivating background, are expected to agree as to the required degree/type of reliability.
- Some reliability requirements can be 'translated' into another definition of reliability. This enables communication and agreement between different parties.
- Requirements can be influenced by measures/actions.

5.4 The Multi-Party Chain Data Model

If we combine the steps taken before into the overall Multi-Party Chain model we get the data model as represented in Figure 6.

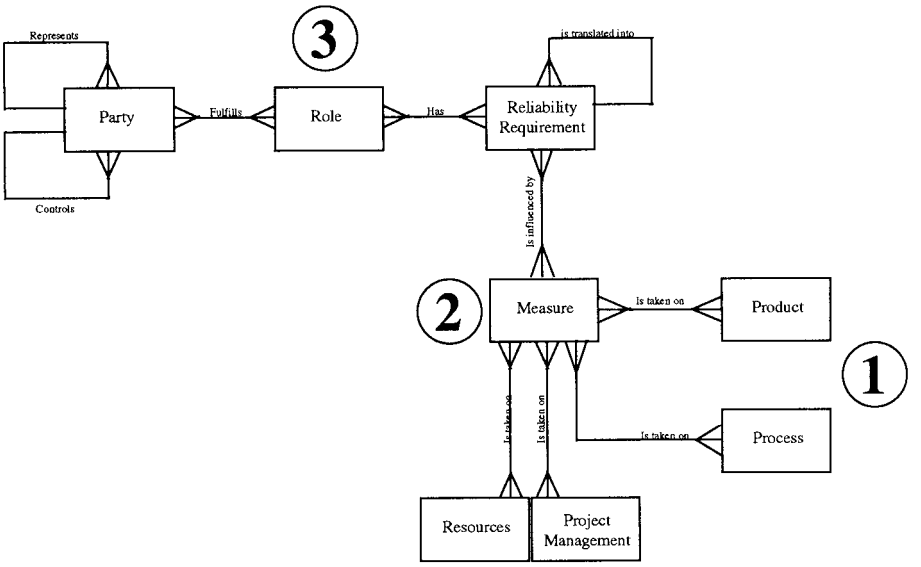

Figure 6 Data model of the Multi-Party Chain.

6 USING THE MPC MODEL

The MPC model can serve different objectives. The model can be used initially to fill in the entities and attributes for a specific embedded software product in a specific organisation, i.e. for positioning the different parties and the different types of requirements that can be distinguished. But it can also be applied for identifying the different types of measures that are executed, for investigating differences in opinions etc.

Based on such a 'case-study' the MPC data model can be used for answering questions such as:

- how to identify gaps between reliability interpretation, and how to bridge them (e.g. between 'fitness for use' interpretations and 'conformance to spec' interpretations; this to enable communication in the area of reliability
- how to distinguish measures, metrics, standards (e.g. ISO9001, usage of test scripts, the gathering of measurement data, informal reviews of developers), and how to store and supply this information in an user friendly way
- how to identify and express generic descriptions of reliability measures and specific descriptions of measures and how to make distinctions
- how to make distinctions between 'internal' parties (e.g. developers) and 'external' parties (e.g. customers)

7 CASE-STUDY: FIELD-TESTS AT SCHLUMBERGER RPS

The Multi-Party Chain approach has been applied to make a model of the Schlumberger RPS situation for a specific product. In this section the Schlumberger RPS MPC-model will be used to demonstrate the way in which this model can be applied.

Schlumberger is a technology driven company with more than 60 years of experience in international operations comprised of approximately 50,000 employees of 100 nationalities from six continents. Schlumberger Retail Petroleum Systems is world-wide leader in equipment and service for self-service petrol stations. The products of RPS are Dispensers, Point of Sales, Electronic Funds Transfer equipment, Back-Office and Forecourt Controllers. These are real-time embedded systems. RPS systems must be operational for 24 hours per day, on every day of the year. This creates a high demand on product reliability and product availability (Solingen et al. 1995).

A specific customer is demanding that each new product is first installed in a pilot installation where it has to be operational for a period of time before the final role-out can proceed. Those pilot installations are called: "Field-tests" (Solingen & Uijtregt, 1996), and the question is why this customer is demanding those tests, and whether an alternative solution can be offered that fulfils the same underlying requirements.

The first step to be taken is to identify the roles that originate to the demand for a field-test. Result: The MPC-model shows that a representative of an oil company has two roles:

- Buying systems that fulfil the oil-company requirements, customer requirements, and fuel-station requirements
- Deliver (fuel) products to fuel-stations in time

It is clear that the request for a field-test mainly addresses the first role of the representative of an oil-company. The difficulty for that specific role is that it has the responsibility to address requirements of many other parties, therefor this person will demand a test (validation of requirements) that is able to check all of the implicit and explicit requirements: practice. This shows the reason behind the demand for a field-test. Based on this conclusion it is possible to offer the person with that specific role an alternative solution which fulfils his needs. For

example showing that person the same product already operational for some years on an other fuel-station. Note: In real-life this request came for a product that was in it's first release to the market, so no operational systems were available, which declares the emphasis of that role for a field-test.

But before making the conclusions to early, it is better to focus on the underlying reliability requirements of the representative of an oil-company. In this example this means focusing on the first role. The underlying reliability requirements that are found using the MPC-model are:

- Availability
- Maturity
- No (known) failures
- Support in case of failure
- Recoverability
- All implicit reliability requirements are addressed

Again, it is shown that very diverse reliability requirements are included in this role, which declares why this person will ask for a field-test, since it seems difficult to address those different requirements together in one measure. The overall requirement of a product seems to be that it is continuously operational, which is being addressed by four of the above reliability requirements: availability, maturity, no failures, and recoverability. For a representative of an oil-company: it just has to work. It has to be 'good'. One can think of an alternative measures to address these issues, like lab-tests, formal proof, measurement results, or proven operation on a fuel-station in practice.

Two other requirements are still left: support in case of failure, and fulfilling implicit requirements. From the role of the representative the support issue is clear: a fuel-station must be available, so whenever it fails fast support must be guaranteed. One can think of contracts that guarantee support with penalties when those are not kept, but testing this kind of support in field-test is not suitable, because support is mostly important in situations where there is no special attention for a fuel-station, and during a field-test there is special attention because it is a test-case. So it can be concluded that a field-test is not addressing this requirement.

The second requirement left, is about implicit requirements which really show the need for testing in the field, in order to verify those implicit requirements. But, it does not imply that the product needs to be tested at a station of the customer. Involvement of end-users during design, or the intensive enhancement of the user-interface in an external usability-lab may be a sufficient measure to address this reliability requirement of the oil-company representative.

An alternative solution to field-testing might therefor be showing that the product is already operating in practice with success, by for example showing successful operation of the system at the station of competitors of that oil-company, or by testing a product on a Schlumberger test-station.

8 CONCLUSIONS

It appears that a general definition of reliability to which all people: developers, users, managers, etc., agree is not available. It has been concluded that the interpretation and view on

reliability is influenced by the responsibility, or role of a person. Those different views of reliability rightfully exist and cannot and should not be ignored. In this paper a multi-party chain model was presented. It can be used to chart the various views on product reliability in such a way that discussion among the various parties involved is enabled. A number of different scenario's for dealing with reliability can be distinguished by using the model. Also, the definition of reliability issues for different roles can be supported. This will enable the definition of metrics for reliability issues and eventually the formation of a link between engineering actions and customers requirements.

Further research within the context of the Spirits project (see appendix) will be aimed at populating the data model with more case material, at further empirical research into the uses of the model, and at developing a series of connected tools that support this approach.

9 REFERENCES

Fenton, N.E. and Pfleeger, S.L., *Software Metrics, a rigorous and practical approach*, Thomson Computer Press, 1996.

Heemstra, F.J., Kusters, R.J. en Trienekens, J.J.M., From Quality Requirement Factor to Quality Factor: an end-user based method, *Proceedings of the 6th European Software Cost Modelling Meeting*, Kerkrade, May 1995, pp. 18.1 - 18.19.

IEEE Standards Collection: Software Engineering, Institute of Electrical and Electronic Engineers, 1994.

ISO/IES 9126, *Information Technology - Software Product Evaluation - Quality Characteristics and guidelines for their use,* International Organisation of Standardisation, 1991.

Kaposi A. and Kitchenham B., The architecture of system quality, *Software Engineering Journal*, 1987.

Solingen, R.v., Latum F.v., Oivo M. and Berghout E.W., Application of Software Measurement at Schlumberger RPS: Towards enhancing GQM, *Proceedings of the sixth European Software Cost Modelling Conference (ESCOM)*, 1995.

Solingen, R.v., SPIRITS: Software Process Improvement in embedded Information Technology environments, *Proceedings of the 2nd ENCRESS Conference*, 1996.

Solingen, R.v. and Uijtregt S.v., Field-Testing Embedded Software Products, *Submitted to the 3rd ENCRESS Conference 1997.*

Trienekens J.J.M. Quality Management in Software Production, A Customer Oriented Approach. In: *Proceedings of the IFIP 5.7 Working Conference on Integration in Production Management* (ed. J.C. Wortmann ed.), North-Holland, Elsevier Amsterdam, The Netherlands, 1992.

10 ABOUT THE SPIRITS PROJECT

'SPIRITS' is an acronym which stands for Software Process Improvement in embedded IT environments. SPIRITS is a research project of Schlumberger Retail Petroleum Systems (RPS) which is executed in co-operation with two research institutes in the Netherlands. SPIRITS

will develop concepts and methods for process improvement to accomplish high and quantifiable reliability of embedded products. The main objectives are the design of:

- methods for process improvement in embedded systems development;
- methods for measurement and evaluation of the effectiveness of process improvement activities on the reliability of embedded products.

The practical application of the concepts and methods is validated in several case studies within Schlumberger RPS. The set of instruments will be based on practical experiences and will be scientifically founded and validated in a number of pilot projects.

11 ABOUT THE AUTHORS

- Rob Kusters is Associate Professor at Eindhoven University of Technology involved in software project management, risk management and software measurement
- Rini van Solingen is Quality Engineer at Schlumberger Retail Petroleum Systems involved in software quality measurement, and in parallel researcher at Eindhoven University of Technology involved in embedded software reliability, software measurement and software process improvement
- Jos Trienekens is Associate Professor at Eindhoven University of Technology involved in software quality, software management, and software process improvement
- Hans Wijnands is Dependability Expert at TNO-TPD in Delft. He is involved as external consultant in projects regarding reliability, maintainability, security and safety of embedded systems