Scenario-based Analysis of Non-Functional Requirements[1]

Alistair G. Sutcliffe and Shailey Minocha

Centre for HCI Design, School of Informatics, City University
Northampton Square, London EC1V 0HB, UK
Tel: +44-171-477-8411
Fax: +44-171-477-8859
e-mail: a.g.sutcliffe@city.ac.uk

1. Introduction

Scenarios have been advocated as an effective means of acquiring and validating requirements as they capture examples and real world experiences that users can understand [Potts et al. 1994]. Although scenarios have appeared in many diverse forms [Carroll 1995], most authors have used them for behavioural analysis, e.g. scenario scripts [Potts et al. 1994] and use case scenarios [Jacobson et al. 1992, Graham 1996, Cockburn 1995]. More wider ranging views of scenarios as context settings for understanding socio-technical systems have been proposed by Eason et al. [Eason et al. 1996] and Kyng [Kyng 1995], but these too focus on user activity and system functionality, rather than system qualities also generally referred to as non-functional requirements. This paper explores the role scenarios could play in addressing non-functional requirements and proposes a method for scenario generation and analysis for the purpose.

Taxonomies of non-functional requirements (hereafter, NFRs) such as security, accuracy, performance, cost, have been proposed by several authors [Roman 1985, McCall et al. 1977, Boehm et al. 1978, Pohl 1996]. NFRs express constraints or conditions that need to be satisfied by functional requirements and design solutions [Myloupoulos et al.1992]. One approach to specifying quality requirements is the *Requirements-Properties Matrix* [Boehm et al. 1978] that identifies the functional requirements implied by NFRs. The WinWin framework of Boehm & In [Boehm & In 1996] links quality requirements to stakeholders and is supported by a tool that helps analyse conflicts and trade-offs between NFRs. However, their approach lacks a systematic method for analysis and resolution of NFRs through specific usage contexts of the system. Moreover, NFRs are invariably informally and vaguely stated in requirements specifications, so an improved means of framing the initial quality question is necessary.

In this paper, we propose an analysis method that describes scenario templates for NFRs, with heuristics for scenario generation, elaboration and validation. The paper is constructed in four sections. First the method is outlined, then the case study used to illustrate the method is introduced. This is followed by examples of the NFR scenario templates and the scenario based analysis is described via the case study. The paper concludes with a brief discussion.

[1] This research has been funded by the European Commission ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project.

2. Scenario-based Analysis Method

The method follows from our previous work by advocating a technique combination approach to requirements engineering [Sutcliffe 1997]. Whereas functional requirements are refined into specifications, NFRs can only be either satisfied by a design or assessed with an implemented system (e.g. performance efficiency). Some NFRs have a close tie with functional specification and are gradually refined into components of the requirements specification, leading some to argue that the term is redundant. We shall not enter into that debate here. As with all requirements, the sooner they can be captured, specified and validated, the better. Accordingly scenarios are proposed as a method for early exploration and validation of NFRs. Unfortunately scenarios alone are of little use, as they have to be tested against some vision of the intended system, be it a system architecture, design or a prototype. Three representations (see Figure 1) are proposed to investigate the relationships between requirements, designed artefacts and design decisions:

(i) Scenarios that express the system context and are used to capture the NFR.

(ii) A system model that is assessed using scenarios. The system model may be either an architecture, a technical design, a prototype, or an implementation.

(iii) Design rationale that enables arguments for or against design options to be studied. We use the *QOC* notation [Maclean et al. 1991] as this expresses a design problem as a *question*, functional requirements as *options* and non-functional requirements as *criteria*. However, design rationale is presented in a tabular format to facilitate comparisons as in Quality Function Deployment approaches (e.g. House of Quality [Hauser & Clausing 1988]).

The method steps are summarised in Figure 2. First the NFRs are identified and categorised. Rather than creating a new taxonomy we adapt the revised McCall's quality model [Bowen et al. 1985] and the Requirements Specification Model [Pohl 1996]. Decomposing NFRs into quality criteria is useful for determining metrics to assess how well the requirement is satisfied in a design; however, metrics are not the main focus of this paper, so this step is not elaborated. Taxonomies of metrics suitable for NFR-like properties can be found in [Bowen et al. 1985]. The next stage is to create contextual and usage scenarios for each NFR. The scenarios are then assessed against the design model to examine trade off options which are recorded in design rationale. This facilitates investigation of different functional requirements manifest as versions of a system architecture, storyboard mock ups or early prototypes. Scenarios are 'dry run' against different views of the required system to establish which trade offs are more satisfactory in addressing the non functional requirements. Design rationale allows the option space to be investigated and provides a summary for a range of decisions.

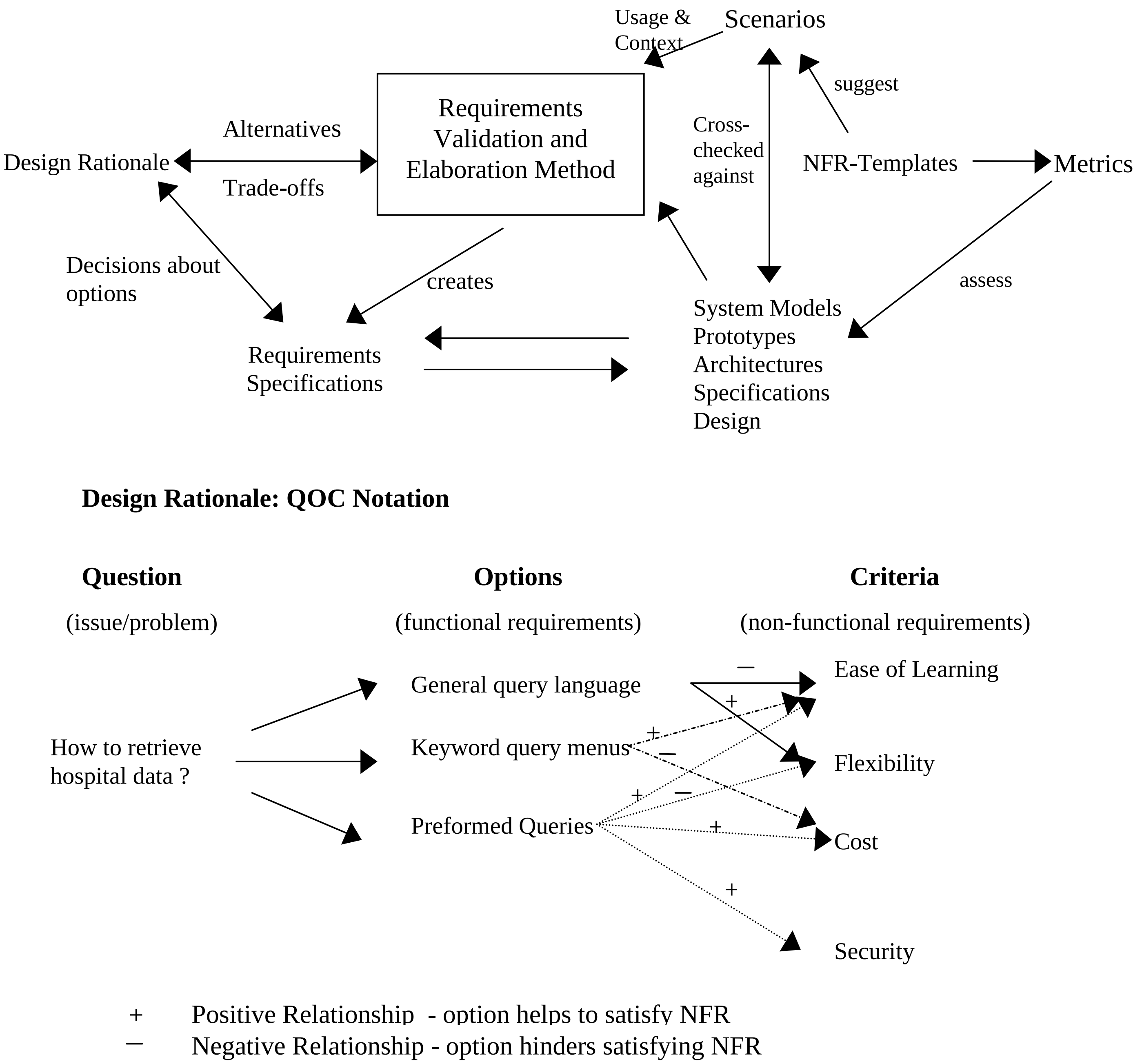

Figure 1 Representations used in the method and example of a design rationale diagram in QOC format. Options and criteria may be cast in a decision table to compare relationships.

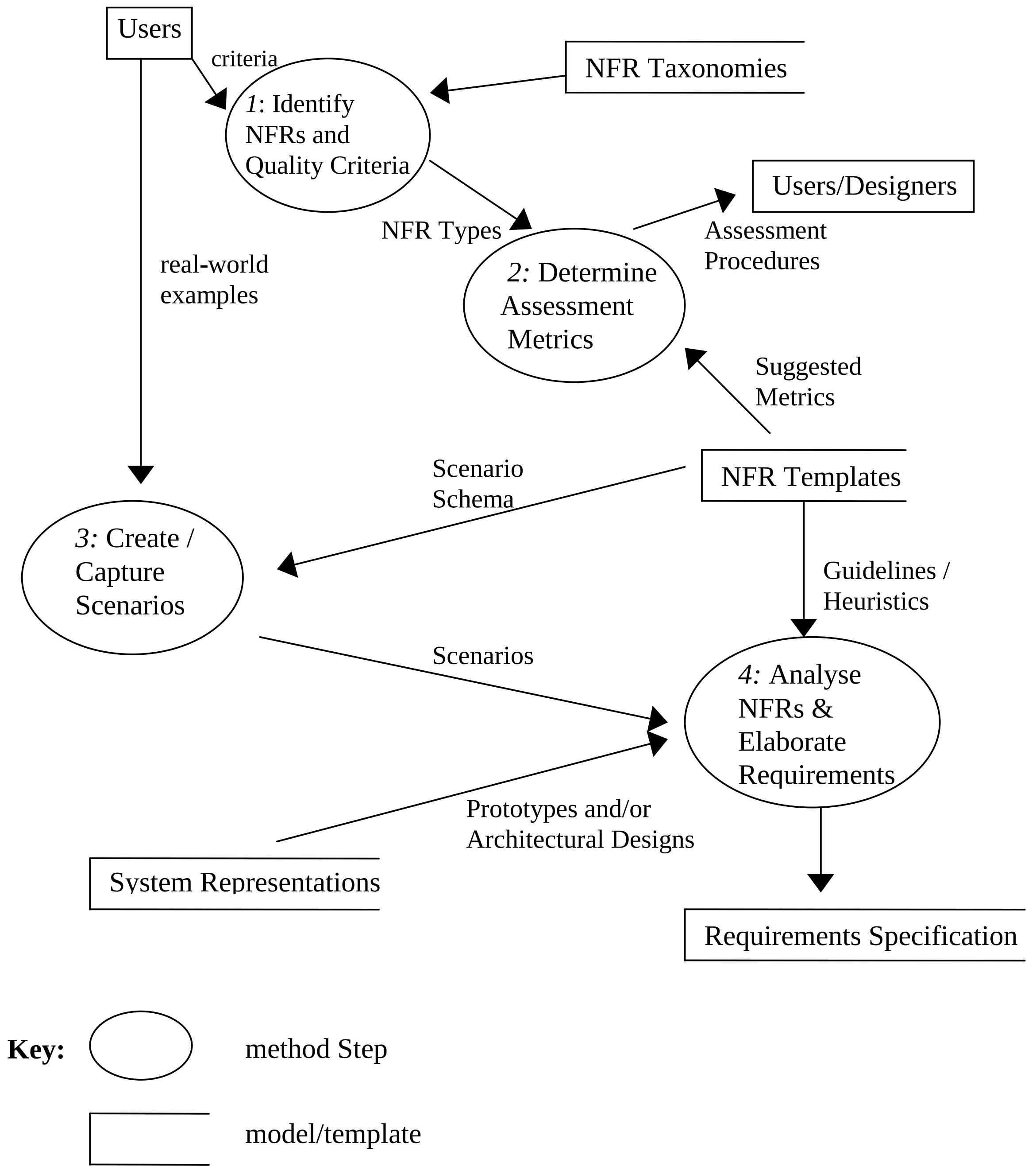

Figure 2 NFR Analysis Method, expressed in DFD format

The method contains a template for each high level NFR with embedded heuristics for scenario creation, validation and benchmark assessment by metrics. The templates, which are described in more detail in section 4, give process guidance as well as scenario knowledge representation schemas. Templates for each high level type of NFR describe the necessary contents of a scenario that may be either generated or captured, an example scenario, validation guidelines for using the scenario in conjunction with other representations, and metrics for quality assessment. As many NFRs interact, choosing one solution may frequently adversely affect another property. To help tracing possible knock-on effects the method provides a comparison matrix (see Table 1) as an aid memoir to trace interactions between NFRs. When interactions are found, these can be reflected in scenarios as a record of the problem or entered into the design rationale as synergistic or conflicting criteria. Although NFRs can be explored by scenario analysis many can only be finally validated by system testing. In this case scenarios become test scripts that advise how measurements should be collected, e.g. for usability, the scenario is a script of user-system interaction which is assessed for errors and operational timings.

Before describing the templates and use of the method in more depth, the case study application is summarised.

3. Case Study Application

HIPPOCRATES is a Windows-based Executive Information System that was developed in the Greek Ministry of Health. It collects, distils and presents critical information on hospitals, doctors and patients throughout the country to support the decision making process of health planners and executives of the Ministry.

The primary use of the system is to assist in health planning and monitoring. The system stakeholders are 6 health planners who are the primary end-users, about 25 departmental managers and administrative staff of the Ministry who make specific inquiries such as hospital staff vacancies, patients, bed occupancies, etc., circa 30 users in private enterprises who need information to satisfy the needs of their marketing departments, and up to 100 other users outside the Ministry, e.g. local and international independent consultants.

The system is composed of two sub-systems: the Data Entry that allows ministry personnel to enter and edit data, and the Information Retrieval (IR) sub-system. The IR sub-system contains health data grouped into four sub-systems; covering functional (general statistics on ward resources and their utilisation), personnel, financial, and patient data information. The structure of the new system is based on the legacy system. Further details of the system can be found in [Markis 1984].

3.1 Impact of non functional requirements

Even though the requirements were explicitly stated in the tender documents and specified in detail for the legacy system, the system failed principally because non-functional requirements were not met. The reasons included many process problems such as failure to adopt a systematic approach to requirements analysis, poor communication, socio-political problems between the customers and suppliers and finally poor appreciation by the developers of the critical nature of usability and performance non-functional requirements. A complete diagnosis of this case history can be found in [Sutcliffe et al. 1997].

A total of 27 usability problems were identified by user interface evaluation; 11 of these were attributable to poor feedback and cues, that is, problems which could have been cured by application of user interface design guidelines such as ISO 9241. Five requirements had simply not been implemented, while 11 requirements were only partially satisfied because of poor user interface design. These problems were a mix of failure in functional specification and lack of attention to usability as a non-functional requirement.

The development team faced various problems during the data transfer, where data could not be automatically transferred to the new system because of incompatibilities. Some data had to be re-transferred and part of the code had to be rewritten. The developers could not take full advantage of the new RDBMS system because the Ministry analysts disagreed with changing the underlying data structure and the database design.

Performance problems were discovered when the real data was loaded in the system during a prototype test. The problem was finally solved by fine tuning the system by creating indexes on database elements and creating an update facility for intermediary summary tables. A first delivery for testing took place in the beginning of January 1996, two months after the initial deadline. The Ministry's analysts assessed the system and reported the following problems in detail: poor system performance; poor interface design and missing information retrieval functions.

To encompass the changes, the developers needed more time and a new deadline was set for June 1996. Another prototype version of the system was delivered to the Ministry for testing in June 1996. The final operational system was delivered in the middle of July 1996, eight months late. At the current time of writing the system has not been finally accepted. The developers dealt with only minor problems leaving the major ones untackled and the Ministry continues to use the legacy system.

Non-fulfilment of non-functional requirements was the main cause of this system's failure, however, we argue that many implications were not captured in the reported history. Only NFRs that came to the customers' attention were cited as problems and these were found too late in the development process. Many remain undiscovered. In the next section, we shall illustrate how the hidden implications of NFRs can be uncovered for this application.

4. Scenario Templates for NFRs

Scenarios may either be generated as a stimulus for requirements analysis or captured as system histories and contextual descriptions taken from the real world. The value of generating scenarios is twofold. First they help explain what a non functional requirement means to a user, and thereby promote discussion and requirements validation. Secondly, they act as definition for the NFR because they provide an operational setting against which the NFR can be assessed when the system is built. In this manner scenarios act as conformance documents to supplement metrics for quality and performance criteria. Each template is composed of general recommendations for the types of representations required, a scenario schema that describes the necessary information that should be represented and illustrated with an example, guidelines for use of the scenario in requirements validation and metrics for quality assessment. A subset of NFR scenario templates we have produced are illustrated in the following section, for details of the complete set see [Sutcliffe et al. 1997]. The scenario templates are illustrated by the case study.

Each template aggregates advice on process and representation for requirements analysis. Inevitably the depth of advice has to be sacrificed for breadth of coverage. Our objective is to provide advice for analysing and validating requirements rather than giving a complete design solution. Design advice for many NFRs implicates research subjects in their own right such as system safety, security, usability engineering and so on. The end point for our method is to point the developer towards the appropriate literature that may be studied for in depth consideration of design issues.

4.1 Usability

Representations: Requirements analysis and validation for usability necessitates a prototype, or at least a storyboard mock up of the user interface as well as scenarios. Specifications as state charts or state transition diagrams may be used for more limited validation.

Scenario schema: one or more scripts of the user interaction with the system, expressed in terms of actions or event sequences with exceptions or errors. A description of the users' characteristics, the system environment and its operational context, and the user-system interface itself.

Scenario

Script: The user is trying to find information on hospital districts. The user accesses the health institution selection screen (see Figure 3) and tries to find a field for entering the hospital name that they want to retrieve. However, the institution drop-down list box does not provide the facility for entering the institution name. A specific institution can only be selected by browsing through the whole list of institutions. This is time consuming and annoys the user.

User description: Ministry officials prepare reports for the Minister. The officials are used to PC Windows interfaces and are experienced in use of the old legacy system, but have no experience of the new system apart from a brief training session. The system is used on-demand for ad hoc data retrieval by ministry officials and other users in their offices.

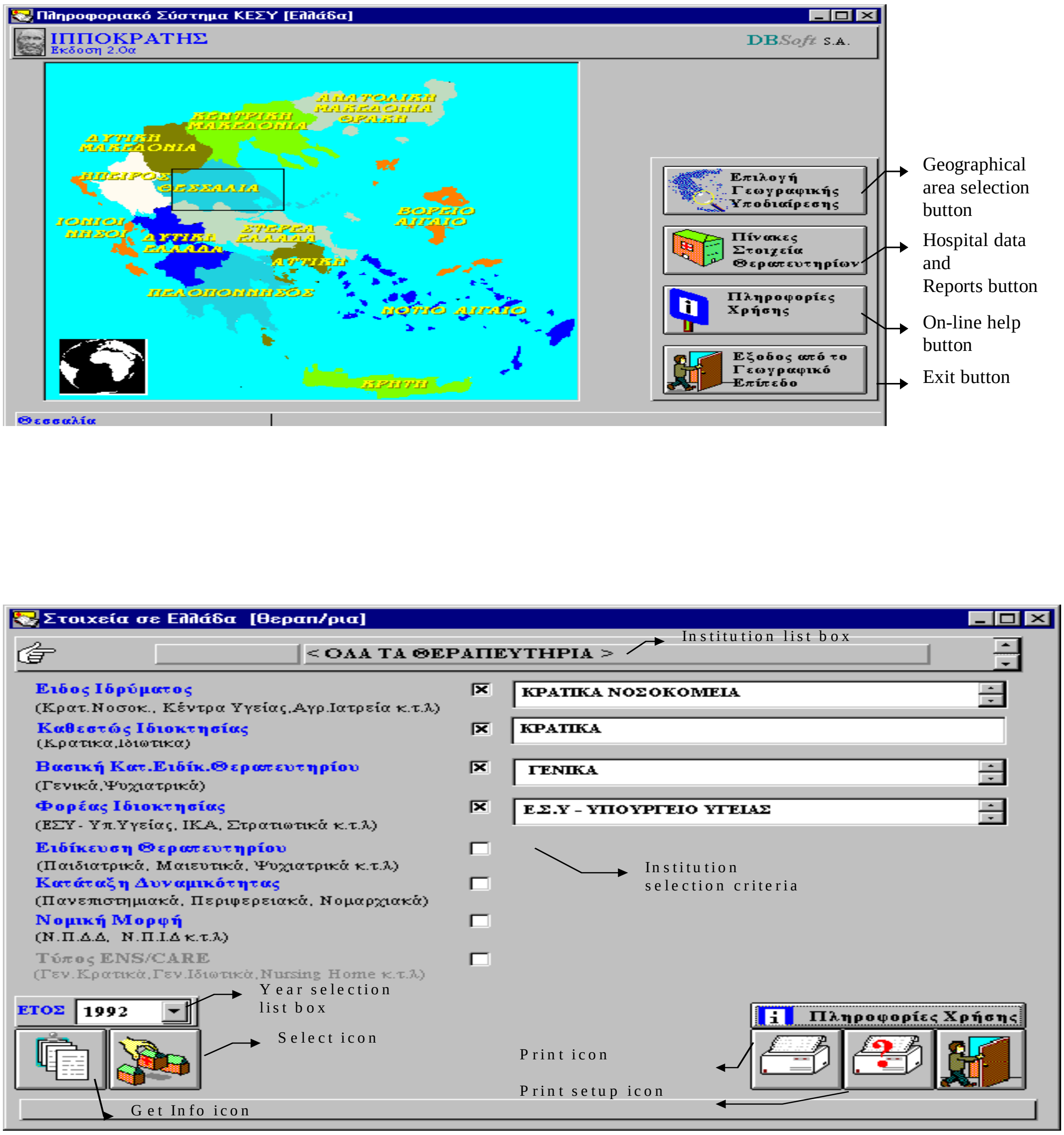

Figure 3 (a) Hippocrates User Interface: Map for geographical area specification in data retrieval.; (b) Query screen for selecting the type of institution within a specific geographical area.

System description: The search facility provides a cascading set of menus so the users can narrow the query or drill down in a search. However, search terms are preset in the menus which can make the system inflexible.

Metrics: User errors, task completion times, task performance (in this case finding the appropriate information), training time.

Validation guidelines: Testing usability requires several scenarios to be generated because different aspects of the user interface should be covered. One starting point is to generate scenarios from description of the user's task, or imagine a typical 'day in the life' of system , and then create scripts for normal actions, exceptions and errors. Step through the scenario script asking questions about whether the user interface effectively supports the interaction, e.g. is it clear what to do next, can the system feedback be interpreted ? Simple checklist heuristics can be used to assess the user interface [see Nielsen 1993] or more complex methods for diagnosing usability problems may be employed, for further details see [Sutcliffe et al. 1995].

4.2 Reliability

Representations: Reliability requires either an implemented system which can be tested, however, as this is rarely available at the requirements engineering stage, recourse has to be made to similar systems and their reliability history. Simulations may be used for some reliability testing, e.g. module interfaces.

Scenario schema: Description of the system failure and its impact in terms of system operation and recovery; expected frequency of failure based on forecasts and historical evidence. Historical evidence and/or assumptions behind a reliability forecast.

Scenario

Expected failure: The data retrieval system fails because of a power supply interruption or because of data corruption in the database.

Impact: The system in inaccessible for several hours while power supply is reconnected; corrupt databases may take longer to correct as backups will have to be restored and recent data re-entered. Loss of the system hinders handling ad hoc queries and preparing reports by the Minister's staff. Loss of up to 3 hours is tolerable, beyond that system failure has a severe impact as workloads back-up and decision making is disrupted.

Expected reliability and rationale: Reliability of 10^{-3} working hours is forecast based on previous history of power supply problems. Working hours are defined as availability during the normal working day 8.00 am to 6.00 pm. Database corruption failure is forecast as 10^{-4} working hours based on previous history modulated by the anticipated increase in data entry volume.

Assessment metrics: Mean time between failures; failure per unit time, degrees of failure (severe. tolerable, etc.). Software complexity metrics, e.g. Function points, McCabe metrics, may be used as predictors of reliability, but their track record is not much better than simple lines of code [Fenton & Pfleeger 1997].

Validation guidelines: The scenario is assessed for the severity of failure and its impact on operation of the socio-technical system. If no recovery or manual working is possible then the impact will be severe. Assessment should also consider the time scale of impact and safety aspects, e.g. system failure in fly by wire avionic systems is immediately fatal and non-recoverable. The reliability of the forecast should be judged from the evidence presented and the method of calculation. Design solutions are in the realm of software engineering. Application of modular design, sound software engineering methods, safety kernels and formal specification should be investigated. Further design advice can be found in [Fenton & Pfleeger 1997].

4.3 Reusability

Representations: Specifications of the reusable components, details of the process of reuse and description of the future application for reuse should, ideally, be available. For reusability, the main problem is anticipating the future reuse contexts. When reuse is limited to the same domain, (evolutionary reuse) this might be reasonably tractable, but for more wide ranging reuse, generating sufficient scenarios to cover several different reuse contexts can be difficult.

Scenario schema: Description of the anticipated reuse context in terms of the applications and method of reuse (e.g. configuration, customisation, design by reuse); description of reuse library; context and constraints; information about the organisations which are the source and destination of reusable components, management of the reuse process, legal and copyright issues.

Scenario

Anticipated reuse: The database schema will be reused for the new decision support system as will menu driven 'drill down' information searching facilities. Reuse method: customisation of existing schema and code, reuse by evolution. Reuse library: database schema for hospital personnel, equipment and patients, financial information organised by hospital department. Contextual information: schema and processes will be reused within the same organisation and the same application - hospital administration database and decision support system, however, the users will change so the searching facilities will need to be tailored. This may impact on the database schema. The clients wish to reuse the database schema and encourage reuse of existing information search functions.

Assessment metrics: % modules in new application reused, % line of code reused, % reuse library reused on new projects, frequency of module reuse, customisation effort in developing application with reuse as a % of expected time for development from scratch.

Validation guidelines: A range of potential target applications for reuse should be assessed to ensure that the reusable components are designed to the correct level of abstraction. In addition, the method of reuse should be investigated to validate the design for the reuse approach being adopted, e.g. reuser designers may be prepared to customise a system but not to design from low level components. This raises difficult problems of anticipating different future reuse contexts, so it is advisable to generate several different scenarios to give better coverage. Legal issues, copyright and management of the reuse process are investigated for critical success factors, for instance, lack of management incentives and small reuse component libraries are known barriers to reuse uptake. For further details see [Prieto-Diaz 1991].

4.4 Portability

Representation: This NFR needs a design architecture to have been created to assess platform and operating environment implications. Ideally a technical design or implementation should be present, and detailed description of the software/hardware interfaces. Portability is related to inter-operability and may also be expressed as an implementation constraint.

Scenario schema: Description of the current application system and its dependencies on machine, compiler and operating system; description of the target operating environment that the system should be ported to, with expected constraints.

Scenario

Current application: The hospital administration database and decision support system is written in COBOL and uses an ORACLE database on a VAX minicomputer. The system should be portable because new installations are expected in different parts of the ministry as well as at other user sites. Target operating environment: PC workstations with MS-Windows-95 operating environment and Oracle database. In addition the system should be portable to UNIX machines with an OSF user interface. Expected constraints: COBOL code will not be completely compatible across platforms or compilers, user interface libraries will be platform specific.

Assessment metrics: Number of platforms the system can be ported to, effort and cost required for each system porting.

Validation guidelines: The scenario is investigated by taking the current system description and target software environments, then tracing the dependencies between the current system and its operating environment. This helps to identify the components and interfaces that will need modification. Architecture diagrams can be used to identify components, but program language, user interface, network communication and database dependencies should also be assessed. Design issues pertain to system interfaces and message/data interchange formats. Operating system resources, timing and synchronisation dependencies are also important. Advice should be sought in standards (e.g. CORBA) and manufacturers' specifications of software and hardware environments which usually give compliance instructions.

4.5 Security

Representations: A system architecture is necessary for assessing this NFR, while an implemented system is needed for testing. Scenarios play a key role is presenting both the context and interaction between the system and its environment. Security, like safety, poses a 20-20 foresight problem for scenario generation. It is difficult, apriori, to anticipate all the possible circumstances under which security might be compromised. Histories taken from previous systems, case studies and different security models can provide material to stimulate imagination.

Schema: The scenario should capture the confidential information or resource to be protected, the potential threat, agents responsible for the threat and their motivation, expected frequency of threats, consequences of security breaches, damage limitation and counter-measures.

Scenario: Patient information is sensitive and confidential. Unauthorised insurance brokers, unscrupulous journalists and criminals might try to access patient data to check insurance applications, embarrass public figures, or harass/blackmail individuals. Threats: may be widespread but the overall frequency will depend on the effectiveness of the system security. Consequences: of security breach are severe. Hospitals and the Ministry could be sued for failure to maintain confidentiality, in extreme cases patients lives could be endangered.

Metrics: Mean time between break-ins, % break-ins foiled, cost of loss, insurance claims.

Validation heuristics: Security should be analysed by considering the possible types of threats, their cause or motivation, opportunities whereby threats may become attacks, and the potential consequences of attacks on the availability, integrity and confidentiality of assets. Abilities of the system to detect attacks before they are successful and trace perpetrators should be investigated. Protection policies that should be considered are: encrypting the data so even if the system security is breached reading the data is not possible within a key; making the system secure from outside access by firewalls; password security; access logs to trace possible perpetrators; and back-ups to guard against possible data loss. Further guidance is given by [Pfleeger 1997].

4.6 Scalability

Representations: Case histories of similar systems can provide material for scenarios, and a system architecture is the minimal set of requirements for validation. Scenarios can propose future visions of large scale systems. Scalability shares the problem with reuse of imagining the future system. Increase in scale might effect different aspects of the system over an uncertain time scale.

Schema: Scalability has several dimensions: number of sub-systems and their distribution, increases in volume of transactions, data volumes, complexity of processing; number of users and their diversity. The future demands of increasing scale on the system should be described along with the rationale for possible increases.

Scenario: The system might have to be expanded to deal with more hospitals and different Ministry offices; more transactions may be expected as the private health sector expands, hence data volumes will increase to deal with additional patients, hospital equipment and personnel. New users may be expected as health information is made available to individual researchers, non government organisations, EU and international agencies, as well as hospital administrators. The expected increases are 50% increase in inquiry volumes over a 5 year period, with 10% more users and distribution into 5 new sites per annum. New data types and hence data schema will be necessary as the system has to deal with health service privatisation. The database schema needs to be scaleable without comprehensive re-design.

Metrics: Cost of unit increases in transactions, data volumes, number of users and installations.

Validation heuristics: Increases in scale should be quantified where possible. The impact of scaling on system resources and assumptions should be questioned. Capacities for databases, processor resources, and peripheral devices (VDUs, disc, printers) will all be impacted by increasing transaction volumes or users. Software design implications should be investigated when different user groups might have to be handled. Few precise sources of advice on design for scalability are available, however, this NFR is related to portability, so references on standards and architectures should be consulted [Kazman et al. 1996, Kazman et al. 1998].

4.7 Performance-efficiency

Representation: Performance can be validated either by building a simulation of the required system, or by testing an implementation. Some performance assessment may be possible on system architectures, but generally a detailed technical design is necessary.

Schema: Volumes of transactions or input events; expected transaction throughput; distribution of volumes over time, peaks and troughs in loading; critical loading in weekly, monthly and annual cycles; response times for interactive and batch work; expected search times for databases; reasons for variation in workload patterns.

Scenario: The Hippocrates system is expected to deal with 4-500 hospitals throughout Greece with summary data on up to 15 million patients. Volumes for hospital personnel are 25- 50,000. As Hippocrates is a management information system it does not have a daily transaction volume; however, data retrieval requests are expected to be in the range of 200- 2,000 /day from users throughout Greece. Daily peaks are expected in mid morning (10-11 am) and later in the afternoon (4-5pm); no weekly pattern is expected, but end of month and financial year-end peaks are expected when Ministers require reports. Further peaks might occur if an election is called or health stories are current in the press. Response times are expected to be < 0.2 seconds for interactive dialogues, and search requests are to be satisfied within 2 minutes maximum, 15 seconds average.

Metrics: Transaction volumes per day/week/month, however, simple volumes disguise differences between transactions, so it is advisable to measure volumes by transaction type. Response times for system loading with transaction type mixes.

Validation heuristics: Volumes should be checked against capacities to deliver the necessary performance. Particular attention should be paid to peaks in volumes when the system's resources will be stretched. Measures for smoothing volume fluctuations should be investigated as should means of accessing additional resources for peak loads. Design for performance is a matter of software optimisation and ensuring adequate processor, memory, etc. resources. Assessment, however, can be complex so further advice on performance evaluation for databases see [Connolly 1996] and for software systems [Fenton 1994].

5. Conclusions and Discussion

Use of the templates demonstrated that nearly all the non-functional requirements we have considered had implications for the Hippocrates system. Application of the method could have prevented the problems that did occur by making the designers aware of usability as a key quality concern of the users. The usability template points towards design advice in ISO standards and other sources. The designers would have benefited from an explicit statement of the expected performance so they could have anticipated problems with the database. However, we also believe that application of the method has exposed several problems that remain to be discovered. These have not been detected to date because the system has not gone live and encountered possible problems with security, scaling, etc. These potential problems were discovered by use of templates and the analysis process prompted by scenarios.

The templates we proposes improve McCall's definitions of NFRs as software product qualities using a hierarchy of factors, criteria and metrics. The template's metrics and scenarios can be applied to the Goal-Question-Metric (GQM) technique of [Basili & Rombach, 1988] to measure quality criteria. For example, scenarios provide the context for goals to identify quality goals, as well as the questions to determine if the goals can be met by measurable criteria suggested in the template.

The framework analyses NFRs as potentially synergistic or conflicting goals to be satisficed during the development process, as do [Mylopoulos et al. 1992] and [Chung et al. 1994,1995] but these analyses require a specific domain model and cover only a subset of NFRs. We have provided more general heuristics and analytic guidance, which complements Chung's NFR modelling approach to facilitate model building for specific NFR types. The Software Architecture Analysis Method [Kazman et al. 1996] proposes scenarios to evaluate software architectures to determine how the required system will satisfy desired NFRs. However, little guidance is given about how to capture or generate scenarios and the focus on architecture limits analysis to later stages in development. Our research addresses the scenario generation problem as well as facilitating resolution of NFRs as early as possible in the system life cycle. In the future we will continue to integrate out work with previous taxonomies and methods as well as extending the coverage to further NFR types. In addition industrial validation of the templates is planned to assess their utility, improve their design and as investigate templates as a means of delivering methodical advice into industry.

Acknowledgements

This research is funded by the European Commission ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project. The authors wish to thank other members of the CREWS project for their comments and advice.

References

[Basili & Rombach 1988] V. R. Basili and H. D. Rombach, 'The TAME project: Towards improvement-oriented software environments', *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 758-773, 1988.

[Boehm et al. 1978] B. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod and M. J. Merritt, *Characteristics of Software Quality*, North-Holland Publishing Co. 1978.

[Boehm & In 1996] B. Boehm and Hoh In, 'Identifying Quality-Requirement Conflicts', *IEEE Software*, pp. 25-35, March 1996.

[Bowen 1985] T. P. Bowen, G. B. Wigle and J. T. Tsai, 'Specifications of Software Quality Attributes: Software Quality Evaluation Guidebook', RADC TR-85-37, Vol. III, *US Rome Air Development Center Report* D182-11678-3, 1985.

[Carroll 1995] J. M. Carroll, 'The Scenario Perspective on System Development', in John M. Carroll (Ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, pp.1-17, John Wiley & Sons, 1995.

[Chung et al. 1994] L. Chung, Brian A. Nixon and Eric Yu, 'Using Quality Requirements to Systematically develop Quality Software', *Proc. Fourth International Conference on Software Quality*, VA, USA, Oct. 3-5, 1994.

[Chung et al. 1995] L. Chung, Brian A. Nixon and Eric Yu, 'Using Non-functional Requirements to Systematically support Change', *Proc. Second International Conference on Requirements Engineering*, pp. 132-139, 1995.

[Cockburn 1995] A. Cockburn, 'Structuring Use Cases with Goals', Web Address: http://members.aol.com/acockburn/papers/usecase.htm.

[Connolly 1996] Thomas M. Connolly, *Database Systems: A practical approach to Design, Implementation and Management*, Addison-Wesley, 1996.

[Eason et al. 1996] K. Eason, S. Harker and W. Olphert, 'Representing Socio-Technical Options in the Development of New Forms of Work Organisation', *European Journal of Work and Organizational Psychology*, vol. 5, no. 3, pp. 399-420, 1996.

[Fenton 1991] N. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, 1991.

[Graham 1996] I. Graham, 'Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis', *Object-Oriented Systems*, vol. 3, pp. 123-142, 1996.

[Hauser & Clausing 1988] J. R. Hauser and D. Clausing, 'The House of Quality', *Harvard Business Review*, May-June 1988, pp. 63-73.

[Jacobson et al. 1992] I. Jacobson, M. Christerson, P. Jonsson, G. Oevergaard, *'Object-Oriented Software Engineering: A Use-Case Driven Approach'*, Addison-Wesley, 1992.

[Kazman et al. 1996] R. Kazman, G. Abowd, L. Bass and P. Clements, 'Scenario-based Analysis of Software Architecture', *IEEE Software*, vol. 13, no. 6, pp. 47-55, Nov. 1996.

[Kazman et al. 1998] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson and J. Carriere, 'The Architecture Trade-off Analysis Method', Submitted to *International Conference on Software Engineering*, 1998.

[Kyng 1995] M. Kyng, 'Creating Contexts for Design', in John M. Carroll (Ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, pp. 85-107, John Wiley & Sons, 1995.

[Leveson 1995] N. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[Maclean et al. 1991] A. Maclean, R. M. Young, V. M. E. Belotti and T. P. Moran, 'Questions, Options and Criteria: Elements of design space analysis, *Human Computer Interaction*, vol. 6, nos. 3 & 4, Special Issue on Design Rationale, J. M. Carroll and T. P. Moran (Editors).

[Markis 1984] P. Markis, 'HIPPOCRATES: An on-line Interactive Health Data Retrieval and Processing System', *Medical Informatics*, vol. 9, no. 2, pp. 79-91, 1984.

[McCall et al. 1977] J. A. McCall, P. K. Richards and G. F. Walters, 'Factors in Software Quality', RADC TR-77-369, 1977. vol. I, II, III, *US Rome Air Development Center Reports* NTIS AD/A-049 014, 015, 055, 1977.

[Myloupoulos et al.1992] J. Mylopoulos, L. Chung and Brian A. Nixon, 'Representing and Using Non-functional Requirements: A Process-Oriented Approach', *IEEE Trans. On Software Engineering, Special issue on Knowledge Representation and Reasoning in Software Development*, vol. 18, no. 6, June 1992, pp. 483-497.

[Nielsen 1993] J. Nielsen , *Usability Engineering*, Academic Press, 1993.

[Pfleeger 1997] C. P. Pfleeger, *Security in Computing*, Prentice Hall, Inc., Second Edition,1997.

[Fenton & Pfleeger 1997] Norman E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, International Thomson Computer Press, 1997.

[Pohl 1996] K. Pohl, *Process Centered Requirements Engineering*, John Wiley & Sons Ltd., 1996.

[Potts et al. 1994] C. Potts, K. Takahashi and A. I. Anton, ‘Inquiry-based Requirements Analysis’, *IEEE Software*, vol. 11, no. 2, pp. 21-32, 1994

[Prieto-Diaz 1993] R. Prieto-Diaz, ‘Implementing faceted classification for software reuse’, *Communications of the ACM*, vol. 34 , no. 5, pp. 88-97, 1991.

[Roman 1985] G.-C. Roman, ‘A Taxonomy of Current Issues in Requirements Engineering’, *IEEE Computer*, pp.14-22, 1985.

[Sutcliffe et al. 1995] A. G. Sutcliffe, M. Ryan, M. V. Springett and A. Doubleday, ‘A Model Mismatch Analysis: Towards a Deeper Explanation of User’s Usability Problems’, *Technical Report, Centre for HCI Design, City University, London, HCID/95/9*, 1995.

[Sutcliffe 1997] A.G. Sutcliffe, 'A Technique Combination Approach to Requirements Engineering', *Proceedings 3rd IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, pp. 65-74, 1997.

[Sutcliffe et al. 1997] A.G. Sutcliffe, A. Economou and P. Markis, ‘Tracing Requirements Errors to Problems in the Requirements Engineering Process’, *in press*.