

Value function deployment in a software process

P. Pulli & M. Heikkinen

VTT Electronics, P.O. Box 1100, SF-90571 Oulu, Finland

ABSTRACT

Quality Function Deployment (QFD) is widely accepted as a value function to guide product development to meet customer needs. We propose a complimentary value function, Risk Function Deployment (RFD), to guide through the uncertainties and risks associated with the product development. RFD makes use of QFD-style matrix-based correlation techniques in risk analysis and prioritization to capture product risk-elements and their probabilities and losses, and to detect risky parts of the product. RFD also supports risk management by evaluating the effect of individual risk mitigation activities. According to our experiences, the benefit of RFD lies in visualizing the complementary sides of decision making, benefits and risks, and by providing a framework for balancing them.

The role of value functions, such as QFD and RFD in the software development process is discussed. Our point of view is from the development of embedded real-time systems. We believe that value functions are able to increase the visibility of the evolving software product, reduce change penalty, and bring closer discipline specialists. Use of value functions enables new management approaches to software development process that are more concurrent and goal-directed rather than activity-based and serialised. Management issues of concurrent goal-oriented software development processes are not well understood and thus open interesting new avenues for research.

INTRODUCTION

Much of the recent research in software process improvement is related to software process maturity movement ignited by Carnegie-Mellon University Software Engineering Institute followed by similar activities in Europe, such

as Esprit/Bootstrap project. We acknowledge these activities with high respects. However, we see these approaches good for catching-up generic software process when the starting point is fairly low. However, maturity movement with its generic process assessment does not give sufficient direction for companies aiming further.

We believe - especially in the domain of embedded real-time systems - that the way to improve the software process is to customise each instance of the software process according the product under development. This means that the underlying generic software development process must be *instantiated* and *run* bearing in mind the goal(s) of the product to successfully enter the marketplace.

Traditionally **instantiation** of the process is carried out by project planning, i.e. definition of work breakdown structure (WBS) leading to a set of *activities*. Then a project schedule is drawn that describes a plan to run this set of activities concurrently taking into account the resources planned to be available. The point of our critic is that this kind of project plan is just an intelligent guess how things could go, not a real instantiation of a project.

Running a project is trying to keep deviations to the project plan as small as possible. Typically, progress monitoring is based on percentage activities completed. Most of the projects run this way show relative good progress in completing the early activities where the completion of the activities is easy since the ambition level can be adjusted. The going gets tougher later when work is built upon results of earlier activities. Typically, in real-time software development projects this leads often to huge testing and integration problems - many times 50 percent of project effort is spent on integration testing. The point of our critic is that project managers tend to monitor percentage activities completed rather than how far the product under development is from the goal.

A natural consequence of the guess-based process instantiation and percentage based activity monitoring is the need to change project plans frequently late in the project when the project manager no longer can hide the deviations. The longer the project manager tries to hide deviations, the longer he/she is ignoring signals from the process that could be used to guide the process towards its goals. A new project plan is typically created by killing some of the less important activities and concentrating effort on delayed essential activities. The result of this kind of projects is often a handicapped product entering the market too late.

We are looking for improved techniques for both process instantiation and running the process. We have directed our interest to

- Goal-directed software development process models [1] which support

instantiation based on market and product characteristics.

- Value functions, since they have potential to give insight on how far the product under development is from the goal(s).

GOAL-DIRECTED PROCESS

A goal-directed approach views the process instantiation as setting of high-level explicit goals and defining the policy to decompose those goals further into explicit sub-goals. Running a goal-directed project is maintaining the dependencies of sub-goals on goals, and changing, deleting or defining new goals. The essential mechanisms are:

Defining goals and rationales: The designer working in an a goal-directed process environment sees a list of goals to achieve. Also the rationales for these goals are explicit, at least how they relate to higher level goals.

Allocation of goal tree to concurrent sub-goal trees: Decomposition of development task into smaller, lower-level (less abstract) development tasks to development teams or individuals. The purpose is to find sub-goal trees with high internal cohesion and relative low mutual coupling.

Pattern-matching a sub-goal tree for reuse: Lookup of existing fully elaborated sub-goals if (parts of) the current sub-goal can be substituted by reusable components. At the lowest level this could be matching against the programming language statements

Validation of a Sub-Goal: Checking the consistency of a state of a sub-goal tree.

Verification between Sub-Goals: Ensuring that the interfaces between concurrently proceeding sub-trees match so that they fill the upper level goals.

Storing Intermediate States of a Goal Tree: Recording the route how the goal tree has been built.

Redoing a Sub-Goal: The purpose of redoing mechanism [2] is to support re-running the process with different goals, i.e. making changes to the product.

QUALITY FUNCTION DEPLOYMENT

Goal-directed process needs a reliable set of value functions for pruning sub-goal candidates at any level of goal tree. Otherwise the benefits of the process model are difficult to access, since without sufficient control information, the process is in danger of getting lost in the multitude of design

choices.

Quality Function Deployment (QFD) [3], [4] was developed in Japan in 1970's and 1980's [5] as an integrated set of quality tools and techniques. Essential idea in QFD is to bring the voice of the customer in the development process. QFD uses variety of matrices to examine in detail the interaction of various dimensions such as function, cost, customer demands, raw materials, etc. Matrices are prepared by a team effort, so they have the potential to bridge the expertise of different individuals or groups. From the decision making point of view, the QFD matrices present a roadmap for the actual decision network [6] of the project team. Early experiments of employing QFD in a goal-directed software process has been reported in [7].

We have experimented with QFD especially in early parts of software projects, especially for deriving the customer needs and relating them to planned product features. The QFD path (Z0-Z1-A1-F2) ranks the alternatives according to their benefits, Fig. 1.

- Z-0 Matrix:** The purpose of the Z-0 matrix is to derive priorities to all stakeholder of the system.
- Z-1 Matrix:** The purpose of the Z-1 matrix is to analyze and give priorities to user needs and requirements.
- A-1 Matrix:** The matrix contains the user needs and plausible technical features of the software fulfilling the needs. As a result the matrix produces information that help to identify the technical features that produce the greatest user satisfaction.
- F-2 Matrix:** The matrix contains the weighted technical features (derived from A-1 matrix) and different alternatives for implementing the features. The idea is to estimate how well each implementation alternative is able to satisfy each technical requirement. As a result each alternative gets a weight that can be used as a guideline in the implementation.

The good news about QFD is that it works but the bad news about QFD is that it works "backwards". This means that it is more a measurement tool than actually a design tool. We have used QFD as a generic decision making tool in the design process in a *generate and test* manner, e.g. when there is a need to develop a set of alternative solutions and to select the most promising as the basis of the implementation.

RISK FUNCTION DEPLOYMENT

We are working with a complimentary value function, Risk Function Deployment (RFD), to guide through the uncertainties and risks associated

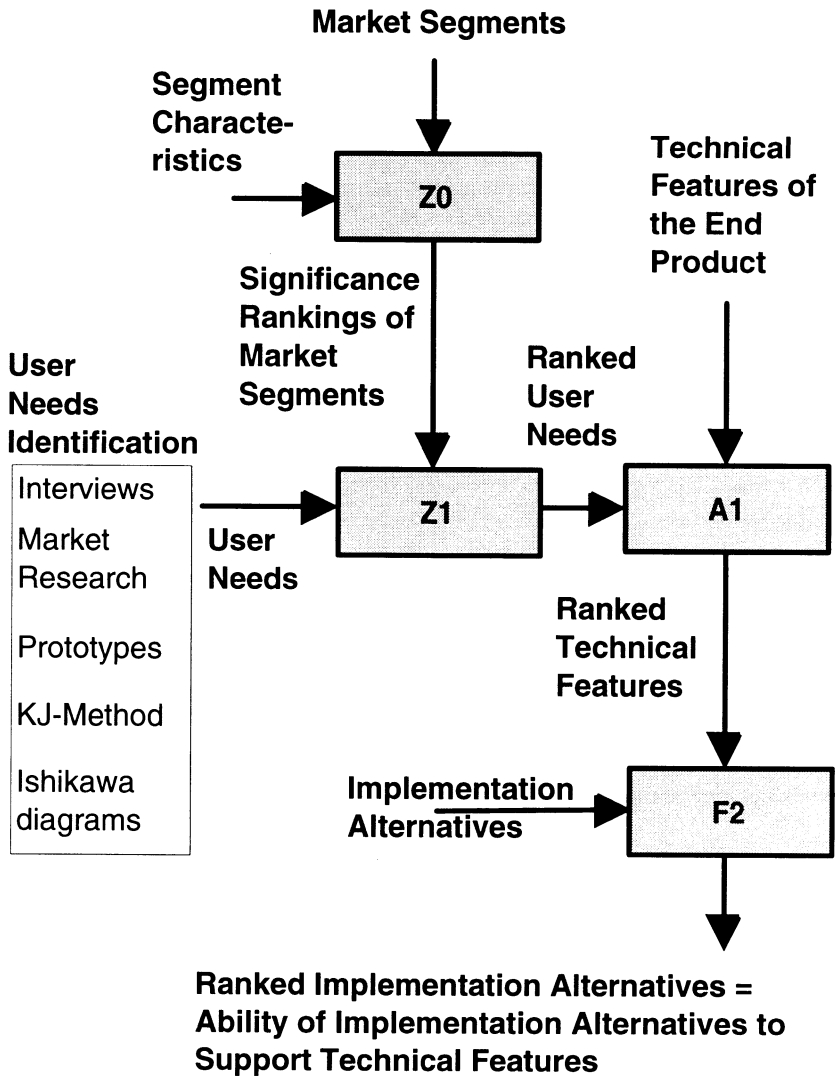


Figure 1: The QFD decision support path for comparing the benefits of different implementation alternatives.

480 Software Quality Management

with the product development. What we hope to gain by RFD is a major reduction of search space for the *generate and test* type process inherent when applying QFD in a goal-directed process.

The RFD we propose [8] is based on QFD-style matrix-based correlation techniques in risk analysis and prioritization to capture product risk elements and their probabilities and losses, and to detect risky parts of the product. RFD also supports risk management by evaluating the effect of individual risk mitigation activities. The RFD path (Z0-Z1-A1-R1L-R3) analyses and ranks the product risks and identifies the critical parts/features of the product, Fig. 2.

R-1L Matrix: The matrix contains the user needs and possible product risks. Product risks are identified with check lists, by developing an Ishikawa diagram [9], by analyzing complaints, and/or by testing the system. Matrix indicates possible loss that each risk can cause. As a result each product risk gets a weight that can be used as a multipliers in the next matrix (R-3 matrix).

R-3 Matrix: The matrix contains the technical requirements (components) and weighted product risks (derived from R-1L matrix). The idea is to indicate probabilities of product risks and to get a ranked list of most probable product risks. The matrix also produces a ranked list of most probable technical requirements (components) that rises the probability of risk occurrences.

FURTHER RESEARCH

We believe that value functions are able to increase the visibility of the evolving software product since they relate all major decisions to anticipated customer needs. This brings closer discipline specialists since they have a level playground where the effects of their decisions can be connected to the overall product attractiveness to the ultimate decision maker - the customer. However, our experience is that QFD and RFD are not enough - other forms for process/product visualising are highly desirable.

Management issues of concurrent goal-oriented software development processes are not well understood and thus open interesting new avenues for research. We are seeking co-operation for a joint European research in communication patterns inside a highly concurrent, goal-directed development process and distributed value functions in decision making process.

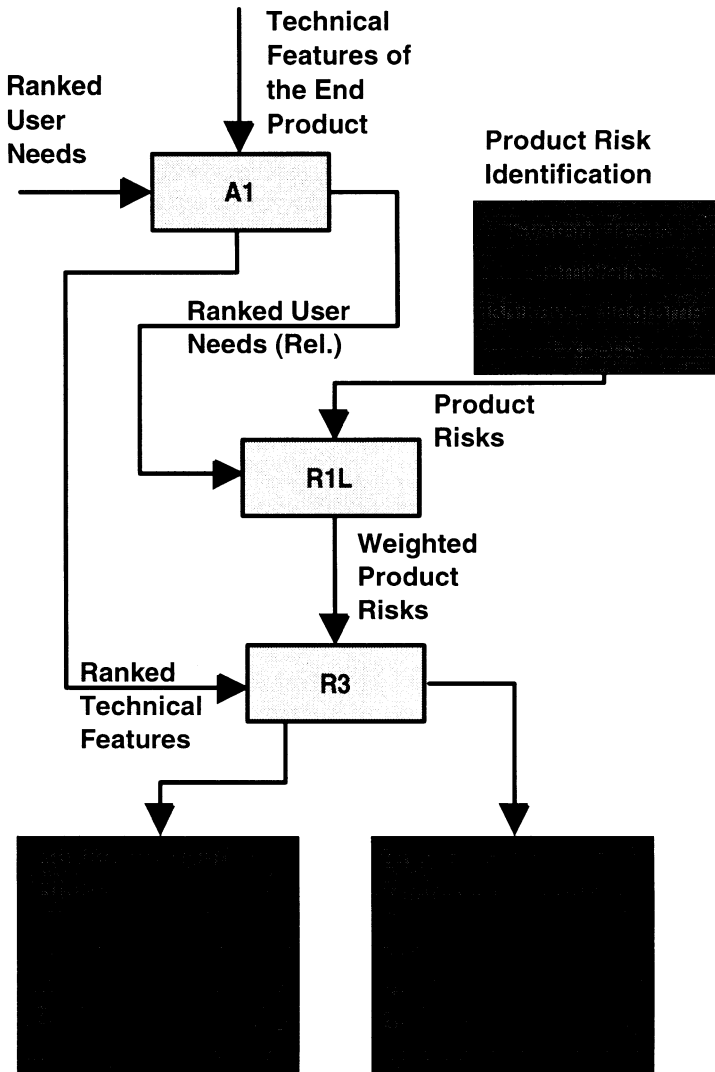


Figure 2: The RFD decision support path for deriving ranked lists of product risks and critical parts/features of the product.

ACKNOWLEDGMENTS

The work presented in this paper has been carried out in the IPTES project [10]. The objective of the IPTES (Incremental Prototyping Technology for Embedded Real-Time Systems) project is to develop methodologies, tools, and theoretical results on distributed prototyping of real-time systems. IPTES is partially funded by the European Communities under the ESPRIT program, project no. EP5570 and the Technical Development Centre of Finland (TEKES).

The IPTES consortium is formed by IFAD (Denmark), VTT (Finland), MARI (UK), CEA/LETI (France), ENEA (Italy), Synergie (France), Universidad Politécnica de Madrid (Spain), Telefónica I+D (Spain), Politecnico di Milano (Italy), and Rautaruukki (Finland).

References

- [1] I. Thomas. The Software Process as a Goal-Directed Activity. Proceedings of the 5th International Software Process Workshop, Kennebunkport, Maine, USA, 137–139, October 10–13 1990.
- [2] T. Katayama, M. Suzuki. Mechanisms for Software Process Dynamics. Proceedings of the 5th International Software Process Workshop, Kennebunkport, Maine, USA, 78–81, October 10–13 1990.
- [3] R. E. Zultner. Software Quality (Function) Deployment – Applying QFD to Software. In 13th Rocky Mountain Quality Conference, 1989. 11 pages.
- [4] B. King. Better Designs in Half the Time: Implementing QFD Quality Function Deployment in America. Goal/QPC, Methuen, Massachusetts, 1989. 564 pages.
- [5] Y. Akao. Quality Function Deployment. Integrating Customer Requirements into Product Design. Productivity Press, Cambridge, Massachusetts Norwalk, Connecticut, 1990. 369 pages.
- [6] B. Curtis. Models of Iteration in Software Development. Proc. 3rd International Software Process Workshop, Breckenridge, Colorado, 53–56, 17–19 November 1987.
- [7] R. Thackeray, G. van Treeck. Applying Quality Function Deployment for Software Product Development. Journal of Engineering Design, 389–410, Vol. 1, no 4 1990.



- [8] P. Pulli, T. Vataja, M. Heikkinen: Extending Quality Function Deployment to Risk Function Deployment. CE&CALS Washington'93 Conference, 241-253, June 28 - 1 July 1993.
- [9] K. Ozeki. Handbook of Quality Tools: The Japanese Approach. Productivity Press, Cambridge, Massachusetts, 1990.
- [10] P. Pulli, M. Heikkinen: Concurrent Engineering for Real-Time Systems. IEEE Software, 39-44, Vol. 10, no 6 1993.