Feasibility Study Inputs based on Requirements Engineering

Robert Pergl

Department of Information Engineering,
Faculty of Economics and Management,
Czech University of Life Sciences,
Prague, Czech Republic
pergl@pef.czu.cz

Abstract. Theoretically, every software project can be successful if it has unlimited resources and does not care about the profit. Because this is not true in practice, the feasibility study is an important step in the software project initial phase. To achieve a valuable analysis, it is important to identify crucial aspects related to the feasibility. Most of the aspects come from the software product requirements.
An original method for identifying and documenting inputs to a feasibility study is presented in this paper. The method takes the requirements on the software product and provides a structured quantified framework for analysis of the requirements' impacts on the project infrastructure needs. The method formalism is based on the theoretical background of systems theory and modelling.

Key words: feasibility study, requirements engineering, software engineering, information systems development, managing software projects

1 Introduction

The *feasibility study* (or the *analysis of alternatives*[1]) is used to justify a project. It compares the various implementation alternatives based on their economic, technical and operational feasibility [2]. The steps of creating a feasibility study are [2]:

1. Determine *implementation alternatives*.
2. Assess the *economic feasibility* for each alternative. The basic question is "How well will the software product pay for itself?" This is decided by performing a cost/benefit analysis.
3. Assess the *technical feasibility* for each alternative. The basic question is "Is it possible to build the software system?". The set of feasible technologies is usually the intersection of the following main aspects:

[1] In many corners of the software and other system development universes, the term, "cost/benefit analysis" or CBA, has been supplanted by "analysis of alternative", itself a term encompassing not only economic feasibility (and, hence, also the practice of CBA) but technical and operational feasibility, as well.

J. Barjis, M.M. Narasipuram, G. Rabadi, J. Ralyté, and P. Plebani (Eds.):
CAiSE 2010 Workshop EOMAS'10, Hammamet, Tunisia, pp. 121-132, 2010.

- Requirements on the technology.
- Available licences and their costs.
- Abilities of the developers and maintainers to master the technologies.
- Maturity of the technology, its support.
- Technologies to cooperate with / integrate with.

4. Assess the *operational feasibility* for each alternative. The main question is "Is it possible to maintain and support this application once it is in production?"
5. *Choose an alternative.*

We will let aside the operational feasibility which is not so tightly related to the requirements engineering compared to the remaining feasibilities (economic and technical). The requirements analysis implies many input parameters to the feasibility study and the feasibility study implies input parameters to the Define Infrastructure stage [2] (Figure 1).

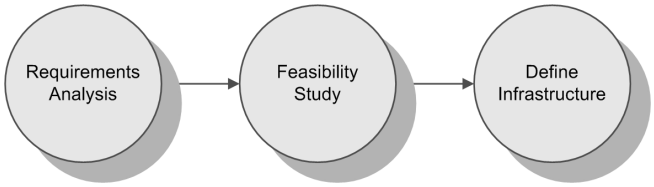

Fig. 1. Data flow from requirements analysis to the Define Infrastructure stage

In an ideal (but a rather rare) case, the project infrastructure fits all the project needs: the process is fine-tuned, all the necessary technologies are available and ready and all the team members have all the needed skills and knowledge. However, typically something is missing in this mosaic. For the project to be successful, a certain *adaptation* to the implied requirements must take place. The role of the method presented in this contribution is to help to identify and quantify such adaptation needs.

The starting point for the method is the demand that the transformation of the requirements to the feasibility study (further denoted as **RFST**, requirements-feasibility study transformation) should have the following attributes:

- *structured*, so that the logic of the transformation is easy to comprehended,
- *recordable*, so that the conclusions may be verified and/or used for increasing the evaluation accuracy in the future,
- *traceable*, so that the conclusions may be analysed and audited.

Here follows an original method based on the analogy between the software project and the systems theory and modelling that provides a quantified structured framework for this transformation.

2 The Method's Formal Background

The method is based on the analogy between systems theory and software project. A formal model of a general system consists generally of inputs, outputs, inner elements and relations [6]. Inputs are divided into endogenous ones, which are inputs crucial for the system model and exogenous, which are other inputs that must be taken into account. The analogy between the general system model and a software engineering project is shown in the Table 1[2].

[2] Not all the terms in the table are important for the goal of the contribution, however we find the analogy very inspiring and thus we wanted to present it in its fullness

Systems modelling term	**Software project analogy**	**Set symbol**
endogenic inputs (crucial inputs interesting for the modelling)	explicit software product requirements	I_S
exogenic inputs (other inputs)	external conditions both predictable and unpredictable (environment)	I_E
inputs (union of endogenic and exogenic inputs)	all external factors and impacts influencing the project	$I = I_S \cup I_E$
outputs	– software product and its parametres, – technology environment for running the product, – documentation and other artefacts, – trainings	O
inner elements	– team (project roles), – subcontractors, – tools (both development and supporting), – artefacts (code and documentation)	P
inner relations (relations between inner elements)	– process, – project management, – intra-team communication, – subcontractors communication.	R_I
relations from inside to outside	information to the customer	R_{IO}
relations from outside inside	information about the requirements changes	R_{OI}
relations from inside to outside and to inside	cooperation requests to customer from the team	R_{IOI}
relations from outside to inside and outside	team responds to immediate customer requests for cooperation	R_{OIO}
relations (union)	all relations	$R = R_{OIO} \cup R_{IOI} \cup R_{OI} \cup R_{IO} \cup R_I$

Table 1: Analogy between the general system model and a software engineering project

Inner elements may be based on the concrete purpose and goal of the model

- objects,
- classes.

Objects are chosen in the case when it is necessary to model the project system in a detailed level of single objects: documents, team roles, etc. In general methodological models, classes will be usually used. Each element then represents a whole class, not an individual object: e.g. "programmer" represents all the programmers; We do not care about structure and dynamics of objects inside the class. For the purpose of this paper, we will assume that all the inner elements are classes.

Now we can speak about *software project management* from the perspective of systems modelling. Input-output mappings constitute functional requirements. The inputs are given (the customer specifies the requirements). The outputs are implied by the inputs. This implication is methodologically not trivial at all, however we may assume that the outputs are created according to some best-practices and their characteristics are thus given. Inputs and outputs are thus out of our management attention here, while the other elements are the core of software process management:

- inner elements,
- relations between inner elements,
- relations from inside to outside.

Managing the software project thus means managing those elements. It may be also comfortable to work with groups of those elements:

Definition 1. ***Project factor*** *is*

- *An inner element.*
- *A relation between inner elements,*
- *A relations from inside to outside,*
- *Any sub graph of a graph consisting of a set of nodes P and a set of edges $R_I \cup R_{IO}$.*

The set of project factors will be denoted C.

If we suppose, that the goal of the project is to achieve the relation between the inputs and outputs, then the project management means fine-tuning the project factors. This happens based on the inputs and represents an *adaptation* of the project system according to inputs in such a way, that the project system achieves its goal in an optimal way, i.e. with minimum resources (time and costs).

Generally, there are two possibilities how to detect an adaptation need:

1. Adaptation *ex post*, that is based on past. The adaptation driver is discrepancy between outputs and inputs. This type of adaptation is used in the Adaptive Software Development (ASD) methodology [3].
2. Adaptation *ex ante*, that is considering the future. This type of adaptation is performed based on prediction about the needs of structure changes. This is the type of adaptation that is relevant to the RFST.

3 The Method

3.1 Definitions

The above formal model as a general software project model may be used for further research by introducing appropriate relations and factors in the system model. The method for supporting the RFST is based on introducing the relation representing demands on adaptation for the project system:

Definition 2. ***Demand*** *$dem(a, s)$ of the input a for the project factor s, where $a \in I$ and $s \in C$ is a mapping $I \times C$ onto an ordinal scale $\langle 0, 10 \rangle$. 0 means, that the input a does not require the adaptation of the factor s. The higher the value, the higher the adaptation needs.*

As for the relations between the inner elements, substitutability has been chosen to be included in the method. Demands for adaptation of one factor may be mitigated by substitution of another factor. An example may be providing training to team members instead of hiring a new needed expert role (or vice versa). Substitutability is defined as:

Definition 3. ***Substitution*** *of project factors s_1 and s_2 $sub(s_1, s_2)$, where $s_1, s_2 \in C$, is a mapping $C \times C$ onto an ordinal scale $\langle 0, 10 \rangle$. The substitution represents a possibility of substituting a demand for project factor s_1 by a substitute s_2. In case where substitution is not possible, the function has value 0. The higher value, the higher substitutability. The value 10 means perfect substitutes.*

Individual demands for factor adaptation are added and we get the total demand:

Definition 4. ***Difference*** *of the factor s_j, where $j = 1, \ldots, n$ is the value of the function $dif(s_j)$, that assigns a non-negative whole number to every project factor $s_j \in C$:*

$$dif(s_j) = \sum_{i=1}^{m} dem(a_i, s_j) \tag{1}$$

where a_i is input, n is the number of project factors and m is the number of inputs.

A factor may be substituted by substitutes, which is covered by the following definition:

Definition 5. ***Total substitution of the project factor*** *s_j, where $j = 1, \ldots, m$ is the value of the function $csub(s_j)$, that assigns a non-negative whole number to every project factor $s_j \in C$:*

$$csub(s_j) = \sum_{k=1, k \neq j}^{n} sub(s_j, s_k), \tag{2}$$

where n is the number of project factors.

The resulting demands for factor adaptation may be thus mitigated by inner substitution relations. We get the resulting difference of the factor:

Definition 6. ***Resulting difference of the project factor*** b_j *is the function*

$$vdif(b_j) = max(0, dif(b_j) - csub(b_j)) \tag{3}$$

The resulting difference represents overall clean demands for factor adaptation. Factors with highest values are the most crucial topics for the RFST, however also high differences mitigated by high substitutions will result in an action (ensuring the substitution works).

3.2 Inputs and Factors Selection

The next step in the method construction is to select appropriate inputs and project factors. The sets I and C are naturally very large. For practical applications it is necessary to specify a subset of the inputs $I_2 \subset I$ and a subset of the project factors $C_2 \subset C$. Ideal attributes of those sets should be:

- completeness,
- independence,
- minimalism.

In practice, it is very hard to achieve perfection in all those parameters and we make a balance between completeness and model comprehensibility and manageability, for the time complexity of processing all demands according to the definition is $\theta(|I_2| \times |C_2|)$.

During the method development, 32 inputs and 57 factors were included in the method. Software product quality characteristics and subcharacteristics according to ISO/IEC 9126-1 [5] and additional aspects were selected as the inputs. The factors were divided into the following categories[3]:

- human resources (the team),
- the management process,
- the artefacts,
- software and hardware tools,
- communication and collaboration.

3.3 The Evaluation

The process of evaluating the inputs for RFST using the method is as follows:

1. *Requirements gathering.* Requirements gathering by ordinary methods (interviews, questionnaires, etc.) [4].
2. *Structuring requirements.* Informal requirements are transformed to the method's inputs.

[3] The list and description of the inputs and the factors is out of scope of this paper. Please contact the author of the contribution if interested.

3. *Requirements analysis.* This step means identification and quantification of demand functions. For all the pairs of inputs and factors, we analyse whether the input implies some sort of adaptation of the project infrastructure. For factors that are not present in the project infrastructure yet, the adaptation means the adoption of this factor. The demand value then represents the complexity of the factor implementation.
4. *Difference function evaluation* according to the Equation 1.
5. *Substitution functions and total substitutions evaluation.* For the factors with high differences, the high adaptation demand may be mitigated by identifying some substitution relations. Substitutions for each factor are then summed according to the Equation 2.
6. *Resulting differences evaluation* according to the Equation 3.
7. *Results interpretation.* Non-zero resulting differences represents overall clean demands for factor adaptation and are thus a topic for the RFST. Generally, the higher the value, the higher the overall adaptation needs. Factors with highest values are the most crucial topics for the RFST, however also high differences mitigated by high substitutions need attention (ensuring the substitution takes place). For a further discussion about results interpretation see the next section.

3.4 Results Interpretation

The quantification of the demand and substitution values is performed based on expert estimations of the method's user. The correctness of the values and thus the correctness of the results depends on the ability of the user to estimate the adaptation needs and to assign ordinal numbers to them.

If the method's user sticks to the recommended ordinal scale $\langle 0, 10\rangle$, the values of the resulting differences lie in the interval $\langle 0, 10m\rangle$, where m is the number of inputs. The upper limit represents the situation when all the inputs imply the maximum demand.

For each specific input set it is necessary to define certain results interpretation ranges having an adequate message. For example for the set of 32 inputs the resulting differences are in the range $\langle 0, 320\rangle$ and we may decide to define the following ranges categories:

1. The range $\langle 0, 50\rangle$ means "Neglectable adaptation need".
2. The range $\langle 51, 250\rangle$ means "Necessary to adapt".
3. The range $\langle 251, 320\rangle$ means "Too high adaptation needs".

When interpreting the values, it is necessary to keep in mind that the resulting difference is a scalar value, while the demands have in nature also various qualitative characteristics, as well. Those qualitative characters are not expressed in the calculation. Thus situations may occur, when two demands for adaptation may even neutralise each other. The resulting difference value represents just a sum of all the demands. Its high value represents the message "this factor needs an attention" and must be interpreted correctly according to the nature of demands that contribute to it.

4 A Practical Example

Let us illustrate the whole concept on a small example. Let us imagine that we need to develop an information system for a cattle farm. The information system should be used to administrate information about the cattle, the information about the veterinary inspections and the lactation data.

Let us suppose that we chose the quality characteristics according to ISO/IEC 9126-1 [5] as inputs I_2. The norm defines six characteristics:

- functionality,
- reliability,
- usability,
- efficiency,
- maintainability,
- portability.

As for the project factors C_2, let us suppose, we use the following factors in several categories:

- team characteristics:
  - qualification,
  - personal stability,
  - personal commitment,
- roles:
  - team leader,
  - analyst,
  - developer,
  - tester,
  - technical writer,
  - subject matter expert.
- process:
  - development process flexibility,
  - risk management,
  - quality assurance.

We gather requirements using suitable methods, structure them, map them to the inputs and evaluate the demands. For simplicity of the example, let us assume that we learnt that the information system must be highly *reliable* and this makes demands for our *team qualification*, on the *tester role* and also makes the *quality assurance* process a crucial one. The project is large and complex and the schedule is tight. It makes demands for the *team commitment*. Unfortunately, it looks like the team commitment is not high and needs increasing, fortunately, the team is at least *stable* and the personality of the *team leader* makes him a team authority. Users request the possibility of remote lactation data gathering. This will be solved by porting the solution to mobile devices. This *portability* of the solution will require more *team qualification* and will enhance demands for the *tester role* and overall *quality assurance*.

First, we fill in the demands table Table 2. Only rows and columns with at least one non-zero demand are shown.

		Inputs a			$dif(s)$
		reliability	functionality	portability	
Factors s	team qualification	6	2	5	13
	commitment	0	8	0	8
	tester role	3	0	8	11
	quality assurance	8	0	5	13

Table 2: Demands analysis $dem(a, s)$

Next, we quantify the substitution functions like:
sub(commitment, personal stability) = 2
sub(commitment, team leader role) = 3

By incorporating these substitutions we obtain a table with resulting differences (Table 3).

		Total difference $dif(s)$	**Total substitution** $csub(s)$	**Resulting difference** $vdif(s)$
Factors s	team qualification	13	0	13
	commitment	8	5	3
	tester role	11	0	11
	quality assurance	13	0	13

Table 3: Resulting differences

Now we can interpret the results. The analysis shows us, that the most crucial areas that will require adaptation is team qualification and quality assurance. There is a high demand for the tester role. Increased personal commitment will be required, but it may be partly mitigated by substitutes.

This output provides a structured input into the feasibility study elaboration. The inputs should be further analysed, the necessary actions formulated and evaluated from the three feasibility perspectives described in section 1.

5 Discussion and Conclusions

The feasibility study is a crucial input to decision about the go/not-go for a project. In the case of software projects, a quality feasibility study has a great

importance, because the success rate of the software projects is far from 100% (Standish Group CHAOS reports). One of the input sources for the feasibility study are the resulting software product requirements.

The presented method provides a way how to quantify the impact of user requirements and other software project aspects on the project infrastructure. The demands for infrastructure changes resulting from the requirements represent required adaptation that should take place for the project to be successful. The presented method represents one possible approach to this issue and meets the stated criteria: to be structured, recordable and traceable. It is based on the analogy between the system modelling and a software project and formalizes the software project as a system with inputs, outputs and inner structure which is represented by project factors crucial in the project management.

The presented example should provide the reader with a feeling how the method works and how it can be used. In this simple example, the conclusions may be made just with common sense, but in real situations, typically tens of inputs and factors will be involved and the conclusions will not be that apparent.

The selection of appropriate inputs and project factors sets is an import step of the method. The balance between the accuracy and the manageable number of elements should be maintained.

The method does not automate the process, but helps to cope with the analysis in a structured, manageable way that simplifies discussion, reasoning and makes decision making process a recordable, traceable action with better reuse options. The method also inspires the user to think about possible relations between the requirements and the infrastructure, which serves as a kind of checklist for not forgetting some important issue.

The economic aspects of the analysis are not covered in the method, however they should be taken into account during the analysis: e.g. when deciding about the possible substitutions. Enhancement of the method in this area is one of the topics of the further development.

The method is now being further developed and tested in practice with the support of the IZMAN project (see Acknowledgement).

Acknowledgement

The research was supported by the Ministry of Education, Youth and Sports of the Czech Republic (Grant No. 2C06004 Information and knowledge management IZMAN).

References

1. Beck K., Andres C.: Extreme Programming Explained: Embrace Change (2nd Edition), Addison-Wesley Professional, (1981)
2. Ambler S.W.: Process Patterns: Building Large-Scale Systems Using Object Technology, Cambridge University Press, (1998)

3. Highsmith III J.A.: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing Company, (1999)
4. Hull M.E.C., Jackson K., Dick J.: Requirements Engineering, Springer, (2004)
5. ISO/IEC IS 9126-1 Information Technology - Software product Quality Part 1: Quality model
6. Skyttner L.: General Systems Theory, World Scientific Publishing Company, (2001)
7. Standish Group International: CHAOS Report 2007, (2007)