



UNIVERSITY OF JYVÄSKYLÄ
INFORMATION TECHNOLOGY RESEARCH
INSTITUTE

Publications of the Information Technology Research
Institute 15/2004

Evaluation of Software Modernization Estimation Methods Using NIMSAD Meta Framework

Jussi Koskinen
Heikki Lintinen
Henna Sivula
Tero Tilus

University of Jyväskylä
Information Technology Research Institute
Publications of the Information Technology Research Institute 15/2004

EVALUATION OF SOFTWARE MODERNIZATION ESTIMATION
METHODS USING NIMSAD META FRAMEWORK

EVALUATION OF SOFTWARE MODERNIZATION ESTIMATION
METHODS USING NIMSAD META FRAMEWORK

Jussi Koskinen, Heikki Lintinen, Henna Sivula, Tero
Tilus

University of Jyväskylä
Information Technology Research Institute
Publications of the Information Technology Research
Institute 15/2004

Jyväskylä 2004

ISBN 951-39-1859-9

ISSN 1236-1615

Editor in charge: Jari Veijalainen

Editor: Tero Tilus

Layout: Tero Tilus

Cover: Tapani Artomaa

Press: Jyväskylän Yliopistopaino, Jyväskylä 2004

ISBN 951-39-1859-9

ISSN 1236-1615

Abstract

This report charts and compares some of the most promising methods and approaches available for 1) estimating the profitability of software modernizations and for 2) supporting the actual modernizations. Thus, we're concerned both with providing arguments for informed decisions regarding software modernizations, and charting effective technical possibilities to support the actual modernizations. In this report the focus is on the first objective. Profitability is affected by benefits, risks, and costs. There exists multiple approaches for evaluating these issues, including strategy selection of legacy system evolution and modernizations. We shall provide a comparison of 12 well-known approaches. The sample includes 6 general approaches suitable for strategic-level decision support, 2 risk evaluation based approaches, and 4 approaches for cost estimation. The evaluation is organized based on NIMSAD meta framework for evaluating methodologies. Reliable estimation of (long-term strategic-level) benefits especially appears to be difficult. Risk-based evaluation approaches seem practical bases for process improvements. Cost estimation has its peculiarities on software maintenance field, which should be taken into account. Empirical validation of most of these models, unfortunately, is either relatively weak or non-existent. We feel that an appropriate process development strategy includes improvement of the existing expert decision making (regarding software modernizations) iteratively in cooperation with software industry including the explication of risks and possibilities involved with modernizations. We will also briefly survey 10 main branches of approaches for actual modernizations.

Contents

1	INTRODUCTION	1
2	ESTIMATION OF THE PROFITABILITY OF MODERNIZATIONS.....	3
3	EVALUATION FRAMEWORK.....	7
4	COMPARED APPROACHES	11
5	COMPARISON	13
5.1	METHODOLOGY CONTEXT	13
5.1.1	<i>Use situation.....</i>	13
5.1.2	<i>Start for methodology use.....</i>	15
5.1.3	<i>Customers and problem owners</i>	16
5.1.4	<i>Context description.....</i>	17
5.1.5	<i>Culture and politics of methodology use</i>	18
5.1.6	<i>Risks in describing context</i>	19
5.1.7	<i>Risks of methodology.....</i>	20
5.2	METHODOLOGY USER.....	22
5.2.1	<i>Users' motives and values.....</i>	22
5.2.2	<i>Needed abstract reasoning.....</i>	23
5.2.3	<i>Needed skills.....</i>	23
5.3	METHODOLOGY	25
5.3.1	<i>Problem situation and problem boundaries</i>	25
5.3.2	<i>Diagnosis of situation.....</i>	27
5.3.3	<i>Prognoses.....</i>	31
5.3.4	<i>Problem defining.....</i>	33
5.3.5	<i>Deriving notional systems</i>	34
5.3.6	<i>Design</i>	34
5.3.7	<i>Implementation.....</i>	35
5.4	EVALUATION	37
5.4.1	<i>Internal evaluation</i>	37
5.4.2	<i>External evaluation</i>	38
5.5	SUMMARY.....	39
6	SUPPORT TECHNIQUES FOR MODERNIZATIONS	45
6.1	MODERNIZATION APPROACHES.....	45
6.1.1	<i>Integration.....</i>	45
6.1.2	<i>Wrapping.....</i>	46
6.1.3	<i>Migration.....</i>	47
6.1.4	<i>Language conversion</i>	48
6.1.5	<i>Data conversion</i>	49
6.1.6	<i>User interface renovation.....</i>	49

6.2	GENERAL SUPPORTING APPROACHES.....	50
6.2.1	<i>Reengineering</i>	50
6.2.2	<i>Refactoring and restructuring</i>	50
6.2.3	<i>Reverse engineering</i>	51
6.2.4	<i>Designing for maintainability</i>	51
7	CONCLUSIONS	53
	REFERENCES	55
	APPENDIX 1: SUMMARIZING TABLE.....	61

1 Introduction

It has earlier been estimated that software maintenance often consumes 50-75% of the total software lifecycle resources (as reported, *e.g.*, by Sommerville, 1995, p. 660), based on the studies performed by Lientz & Swanson (1980) and McKee (1984). Other evaluations at that period (Port, 1988; Huff, 1990) are in line with these results. It is stated more recently (Seacord *et al.*, 2003) that maintaining software and managing its evolution nowadays can represent more than 90% of its total cost, which is coined as “legacy crisis”. This trend has been identified also *e.g.* by Moad (1990), Eastwood (1993), and Erlikh (2000). According to Sommerville (2000) there are at least 250 billion lines of source code being maintained, and that number is increasing. According to Müller *et al.* (1994) the amount of code being maintained doubles in size every 7 years. Thus the importance of software maintenance and software evolution can hardly be over-emphasized.

According to Lehman’s first law (Lehman *et al.*, 1998) software must be continually adapted or it will become progressively less satisfactory in “real-world” environments. Software maintenance and modernizations help to keep it up-to-date and in use. Modernizations generally refer to large-scale changes which help to extend its lifetime. Many legacy systems have been very large investments and they contain invaluable business logic and knowledge. Thus, the modernization/renewal of these legacy applications (instead of a complete rewrite), is often potentially desirable option and its justification is an important issue.

Large modernizations are due to either pressures to adaptive, perfective or preventive maintenance (cf. Chapin *et al.*, 2000). Adaptive maintenance aims at updating the software to meet the new technical requirements. Enhance (or perfective) maintenance deals with changing user needs. Preventive maintenance aims at improving the system for the benefit of its future maintainability. Generally modernizations mean large changes to the system, typically due to the major changes in the technical context (*e.g.* user interface, operating system, programming language, system architecture) or major changes in business processes of the organization using the software. These sort of large modernizations are generally both technically (and humanly) hardest, and economically most critical maintenance situations. The typical degradation of software quality due to made changes (the ripple effect) is one of the main reasons favoring either rewrite or reengineering.

Large administrative and business applications are our main focus. These systems support the business processes of the organizations that use them. They are typically connected to their environment in complex ways. Thus they are E-type, “real-world” systems in the classification of Lehman *et al.* (1998). They may be “backend” systems. We’re here especially interested in software products which are aimed at supporting central business processes, and potentially to compete successfully in volatile software markets in long-term. Especially in this case changes of user requirements and evolution of the technical environment is rather a rule than an exception. The profitability of extending system lifetime depends on *e.g.* the system itself, the way that requirements change and of the selected/available maintenance/evolution strategies.

In this report we shall survey and compare some of the promising methods and approaches available for: 1) estimating the profitability of software modernizations, and for 2) supporting the actual modernizations. We have earlier published a wider and less focused analysis of the area (Koskinen *et al.*, 2003a). In this report we shall focus on the evaluation and comparison of the software modernization estimation methods by applying NIMSAD meta framework (Jayaratna, 1994) for methodology evaluation.

2 Estimation of the profitability of modernizations

In the following sections we shall compare some central methods for evaluating: 1) software maintenance/modernization strategies and benefits, 2) related risks, and 3) related costs. First we will provide a general discussion of the topic. The strategic options are discussed also *e.g.* by Bennett *et al.* (1999).

The software user (*i.e.* client) organization (in principle) has the following options:

- 1) Continuing using the currently used software product.
- 2) Replacement - purchasing a competing product (if such exists on markets).

The software supplier (in principle) has the following options:

- 1) Termination of system maintenance (*e.g.* warrants, corrections of errors, upgrades, or user support).
- 2) Complete rewrite of the system.
- 3) Modernization in required extent (this may include *e.g.* reengineering, and/or reverse engineering).

The software supplier (*i.e.* the organization producing the software system) and the user organization may have different notions, and possibly conflicting interests, on the issues of modernizations. For the software supplier, the decision may include pricing the product. For each organization it would (in principle) be sufficient to receive a reliable ROI (*Return on Investment*) -ratio for each option. In principle, the determination of the order of the alternative options would be sufficient (although relative values may serve better as support for argumentation). Different alternatives (with *e.g.* varying probabilities or expected economical value) may be analyzed *e.g.* by using decision trees, which is a simple and well-established technique.

ROI reduces the effects of multiple underlying decision criteria into a single variable. It is calculated (at the basic level) based on probable costs and benefits of an investment. The reliability of the ROI-calculations depends on the degree that relevant factors are taken into account while determining it. ROI may also serve as an argument for a made decision. Thus, the order of a decision and its argumentation is not fixed.

In case of software user organization, the easiest factor to determine is the actual purchase price of a software product. Generally the benefits are much harder to determine. Long-term effects are important, but their determination is generally obviously harder. Costs typically include also other factors than product price, *e.g.* there may be needs to personnel training, changes of organization's business processes *etc.* Wrong strategic choices may lead to a "dead end", *e.g.* because of the end of support for old technical solutions *etc.* In addition to these issues there usually are also other strategic-, politic-, legislative-, and regulatory considerations complicating the actual decision making and potentially the determination of reliable ROI-estimate. Thus, in "real-world" environments the ROI-estimates reflect expectations, and should be accompanied with confidence levels. In practice, it may be convenient to use simplified calculations, by *e.g.* modeling the benefits via saved work-effort (since the determination of the expected value of this variable is relatively uncomplicated).

Making of software modernization decisions is a process within some organizational context. "Real world" decision making in business organizations often have to be made based on "bounded rationality" (Simon, 1983). Besides that there exists multiple (and possibly conflicting) decision criteria, the certainty, completeness, and availability of useful information (as a basis for the decision) is often limited.

Especially organizational group decision making is complex, see *e.g.* (March, 1981). There may exist political and human reasons behind the decisions. The made decisions are not necessarily correct, and the evaluation of their correctness may be difficult or impossible. Thus, the nature of this process should be taken into account while planning its support (DeSanctis & Gallupe, 1987). There may also be unnamed, unstructured or intuition-based factors underlying the actual process.

Often it is not sufficient to consider modernization at system level. Instead, our hypothesis (based on preliminary empirical data), is that the relative benefits of the options should be determined at a more fine-grained, subsystem level. In ideal case the system architecture would be modular in such fashion that new functionalities could be "plugged in", or an old one replaced, without interfering rest of the system. Then the replacement decision could be based largely directly on technical quality metrics of the components, and mappings to the user requirements that a component fulfills (while assuming this metrics and traceability information available), and customer satisfaction.

In ideal (theoretical) situation there would probably exist a decision support system taking into account all the relevant decision criterias, producing the ROI-estimate, and explicating the argumentation (in style of expert support). Our current hypothesis is that, in case of (legacy system) modernization decision making, the right agenda is to iteratively enhance expert decision making support with feedback from software industry, by paying attention to the possible risks and potential benefits of modernizations. The current state of the process in

organizations should be charted, and then improved. Plausible risks should be identified, significant options identified and their relative importance estimated.

Our hypothesis (based on the preliminary empirical data), is that complete rewrite often is not (at least) an economically feasible option in these cases. Instead, there is often a need for wrapping the old functionalities inside a newer “facade”. Also even complete rewrite may (and in principle should) reuse relevant earlier system documentation.

Other main methodological and empirical observations made at this point of the project are as follows. Gathering information about the evolution patterns of “real-world” systems, as suggested, by Kemerer & Slaughter (1999) is valuable. According to Kitchenham *et al.* (2002) criterias for successful empirical software engineering research include the following: 1) a large data source of programs and their versions, 2) willingness of a good commercial partner to participate in the research project, and 3) highly disciplined research approach with desire to expand the previous research.

Jørgensen (1995), has found out that the explanatory power of even the best of the current modeling techniques (for software maintenance/modernization effort estimation) is not sufficient. Experts generally perform better. Thus it is more effective to focus on gathering empirical data on the area than trying to optimize the models technically (with likely only marginal possible improvements). Thus, a viable strategy, in present situation, is to iteratively enhance support for expert decision making with empirical feedback from the industry. In ideal case there would exist sufficient amount of available (longitudinal) data within the specific organization on similar projects in order to calibrate the applied models properly.

In principle (in this case), gathering longitudinal metrics information, information about large system portfolios, and forming large project databases appear to be promising research approaches. There is a clear need to gather empirical data on both actual system portfolios of software developing organizations (including such attributes as average system lifetimes, change-history and reasons for large-scale modernizations), as well as actual criteria for modernization decisions of their customers.

We aim at gathering both versatile quantitative and qualitative data using questionnaires (concerning system portfolios, and performed modernization projects, focusing on organizations producing software) and semi-structured interviews (concerning decision criterias, focusing on organizations using software). Interviews potentially enable capturing such decision criterias which would not necessarily otherwise be identified. We’ve planned to later expand this query also to cover other organizations (by also performing a large-scale, phone-based information gathering).

One of our current hypotheses (based on the gathered empirical data) is that the sufficient complexity of actual decision making greatly varies depending on the actual situation. *E.g.* changes in regulations, and legislation, or ending the support of outdated platform or operating system, may force to major modernizations,

with no need to specific further evaluations. Likewise, in case that there are only few software suppliers on markets, the availability or non-existence of specific aspects of the competing systems may clearly determine the choice. We will continue extending the theory formation and collection of empirical data in parallel.

One of our earlier hypotheses has been that in case of complex “real-world”, large-scale business/administrative systems, also intuition is used as a basis for decisions. Our preliminary empirical data suggests that if this a case, it is not easily admitted (although sometimes it is difficult to pin-point the actual expert decision making criterias). Thus, this area deserves further analysis.

One underlying basic problem in software engineering field (which affects *e.g.* effort estimations), is the lack of really reliable basic-level metrics. *E.g.* source lines of code is not an optimal metric, since the length of program lines depends on the used programming language, programming style, and amount of comments. Function points are another option, but (as noted) also they have their weaknesses. Other possibilities include, *e.g.*, number of tokens, statements, procedures, or modules. Complexity is one of the main maintenance cost drivers (Banker *et al.*, 1993), but metrics for it, such as cyclomatic complexity (see *e.g.* Gill & Kemerer, 1991) may produce misleading results (*e.g.* data complexity is ignored). Using multiple metrics in conjunction reduces this problem in some degree (Kafura & Reddy, 1987), but it may easily be too costly to gather such information. Because of these problems, it is a good idea to collect versatile system data, including also non-technical, more general level, estimates of system attributes and expert judgments.

3 Evaluation framework

We shall compare the most applicable of the currently widely available methods by using NIMSAD framework (Jayaratna, 1994). NIMSAD (*Normative Information Model-based Systems Analysis and Design*) is a meta framework for evaluating methodologies, especially information system development methodologies. Methodology is defined as an explicit way of structuring (rationalizing) thinking and action, involving both critical and creative thinking (Jayaratna, 1994, p. xi). In our interpretation we will apply this framework for more general situation of problem solving, and the related approaches and methodologies. This is in line with the general nature of the framework and the fact that Jayaratna also emphasizes the concepts of problem solver, problem situation, and problem solving process. One special kind of support for problem solving is organizational decision support, which is our main target while related to software modernizations. We will use the terms methodology and approach interchangeably. It should, however, be noted that most of the approaches for cost/benefit/risk estimation of software modernizations, are not methodologies in a strict traditional sense of information system development. Most notably, their focus and amount of applied notational conventions differ.

NIMSAD's good sides are the following: 1) it has a wide scope, 2) it is not restricted to evaluation of any particular category of methodologies, 3) it is practical - it has been used in several "real-life" cases, and 4) it considers different use situations. Weaknesses mentioned by Forsell *et al.* (1999) (who have applied it in case of evaluating methodologies for software component reuse) include the following: 1) the methodology user is separated from the methodology context, 2) viewpoint of what epistemology and ontology are differ from commonly accepted viewpoints, 3) there exists some ambiguous uses of concepts. Thus, the mentioned problems are mainly conceptual, and should be avoidable while being identified.

According to NIMSAD, methodologies are evaluated through four elements, which are: the Methodology Context, the Methodology User, the Methodology, and finally Evaluation; the way the methodology evaluates the other three elements. Jayaratna defines an extensive set of questions. Avison & Fitzgerald (1995) have summed up those questions. We've used their list of questions as a starting point in our analysis. This approach has also been followed by Forsell *et al.* (1999).

The covered contexts of this meta-framework, the elements, and the related questions are represented in Table 1, as represented by Forsell *et al.* (1999) based on Avison & Fitzgerald (1995). The last column of the table represents the needed re-interpretations of the framework for the properly focused evaluation of general problem solving approaches (including approaches for software modernization estimation). The represented interpretations will be applied in the following sections. M stands here for methodology.

NIMSAD elements as represented in (Forsell <i>et al.</i>, 1999)	Questions	Applicability and the needed re-interpretations of the NIMSAD elements for the proper evaluation of general problem solving approaches
Methodology Context		
Use situation	What kind of situations does the M suit?	OK, but the scope is generally more restricted
Start for M use	Which incidents initiate the use of the M?	OK
Customers and problem owners	Who are the customers and problem owners?	OK
Context description	How is the context described?	OK, but the role of context is less salient
Culture and politics of M use	What is the culture and politics of M use?	OK, but the possible importance of this element is generally not explicated
Risks in describing context	What risks does the M identify when describing context?	OK, but the role of the context is altogether limited
Risks of methodology	What are the risks in using the M?	OK
Methodology User		
Users motives and values	What are the users' motives and values	OK
Needed abstract reasoning	What level of abstract reasoning is required from the user of the M?	OK
Needed skills	What skills does the user of the M need to accomplish tasks required in M use?	OK
Methodology		
Problem situation and boundaries	How does the M help in understanding the particular situation and boundary setting?	OK, but is generally not deemed as very important
Diagnosis of situation	How does the M user diagnose what kind of system is needed?	OK, but is sort of present state analysis and includes as a central element the preconditions for the method application
Prognoses for system	How the M user make a prognosis for the system to be built?	<i>Prognoses for problem solution</i> , answers to the questions: where we want to go, what are the goals?
Problem defining	How the M user define problems which need to be solved?	OK, but in most cases trivial (and could be reduced to the issues already covered in the previous element)
Deriving notional systems	How you get systems which need to be described?	OK, but the process for describing the target state is usually not needed (and not supported)
Design (originally: logical and physical design separated)	Is this phase done? How the M user implements this phase?	<i>Design</i> , there usually is no abstract (or meta-level) design, and the role of design is altogether restricted (consisting of sub-phases producing input for the next element)
Implementing the design	Is this phase described? What is included in it?	<i>Implementation</i> , the actual problem solving process
Evaluation		
Evaluation (originally: cases before/during/after intervention separated)	How are the other elements (presented by NIMSAD) evaluated?	<i>Evaluation (external, internal)</i> , these M:s usually do not support internal "reflective" or "longitudinal" (self)evaluation

Table 1. The NIMSAD framework and its interpretation in our case.

The methodology context means the situation where the methodology is intended to be used and what is considered as important in the situation. Usually, the context is the organization whose problems are to be solved by the software being developed. In these sort of (often numerical models) the context is not necessarily described by the methodology. However, there typically are notes included regarding the issues which should be taken into account. The problem solving process is supported by the methodology which, in turn, is used by the methodology user. So, the methodology user is the problem solver or decision maker. It is necessary to know what guides his decisions, what kind of abstract thinking is required from him, how well he has to know the methodology he utilizes, and how he can acquire the necessary skills. These things are included in the evaluation of the methodology user element.

From the methodology element we would like to know how it supports the problem-solving process. A methodology guides and assists the methodology user in seeing the problem in certain way, in asking relevant questions, and in overcoming any new problems that arise. Thus, regarding the methodology element, we evaluate how the methodology supports the problem-solving process, that is: problem formulation, possible design of the solution, and actual problem solving, *i.e.* implementation of the solution. According to Forsell *et al.* (1999) general problem formulation consists of understanding the situation, performing the diagnosis, defining the prognosis outline, defining problems, and possibly deriving notional systems (describing the target state). The methods of our focus area basically assume that the problem is more or less trivially definable, and thus don't include specific support for that part of the process. The design phase contains the possibly explicated description of the planning of the solution. Since these sort of models are mainly not concerned with planning the problem solving, the "design phase" is for most of the methods almost irrelevant.

Empirical validation is especially important in case of methods intended to be used related to actual industrial software production. We have divided evaluation (in this sub-context) into two parts: 1) *internal evaluation* refers to reflective evaluation, *i.e.* to the evaluation based on the suggestions included in the methodology itself (including calibration by the users, as well as long-term iterative model development based on user feedback), and 2) *external evaluation* gathers all other type of evaluations possibly made (*e.g.* proofs, tests, simulations, use of expertise, field experiences, case studies, user feedback, and industrial references).

4 Compared approaches

Our evaluation includes the following approaches, which can be categorized based on their main purposes as represented. In the following we will use, for the sake of brevity, acronyms of the approaches or methodologies. The acronyms, however, are not in all cases used by the original inventors of the approaches. In case of some of the methodologies, for some of the questions posed in NIMSAD framework, the answer could not be found from the available materials. These cases are marked with N/A (question is not applicable or needed information is not available).

Modernization strategies and benefits:

- WMU (*Warrants, Maintenance, Upgrade*, Sahin & Zahedi, 2001b) is a model for choosing appropriate maintenance strategies based on aspired customer satisfaction level and their effects on it.
- SABA (Bennett *et al.*, 1999), is a high-level framework for planning the evolution and migration of legacy systems taking into account both organizational and technical issues.
- Renaissance (Warren & Ransom, 2002; Ransom *et al.*, 1998) is a method for iteratively evaluating legacy systems, from technical, business, and organizational perspectives.
- VDM (*Value-based Decision Model*, Visaggio, 2000) is a method and decision model for determining suitable software renewal processes at component-level based on the technical and economic qualities of those components.
- SRRT (*Economic Model to Software Rewriting and Replacement Times*, Chan *et al.*, 1996), is a formal model for determining optimal software rewrite and replacement timings based on versatile metrics data.
- RPP (*Reengineering Planning Process*, Sneed, 1995b) is a process model for estimating costs and benefits of reengineering.

Modernization risk management:

- RPFA (*Reengineering Project Failure Analysis*, Bergey *et al.*, 1999) is basically a check-list of potential problems related to reengineering projects, and of the corresponding appropriate technical and other means to react to the situation.
- RMM (*Risk-Managed Modernization*, Seacord *et al.*, 2003) is a new, general software modernization management approach taking risks (and both technological and business objectives) explicitly into account.

Modernization costs:

- COCOMO II (*Constructive Cost Model II*, Boehm *et al.*, 2000), is an established and relatively widely used general method for software effort and cost estimation, including many extensions for different kind of software and evaluation situations.
- FPA (*Function Point Analysis*, Albrecht, 1979; Albrecht & Gaffney, 1983) is an established and relatively widely used general method for software effort and cost estimation, having many variants for different kind of software.
- Softcalc (Sneed, 1995a) is a model and tool for estimating costs of incoming maintenance requests, developed based on COCOMO and FPA.
- EMEE (*Early Maintenance Effort Estimation*, De Lucia *et al.*, 2002;2001) is a new approach for quick maintenance effort estimation before starting the actual maintenance.

Most of these and many other approaches have been reviewed shortly earlier by Koskinen *et al.* (2003a). The most interesting of the relatively recent approaches which are not included in these reports are those represented by:

- Sneed (1999), for reengineering risk estimation (similar to RPFA),
- Swanson & Dans (2000), for software lifetime considerations, and
- Verhoef (2002), for system portfolio considerations.

5 Comparison

5.1 Methodology Context

5.1.1 Use situation

WMU: Support for analyzing warranty-, maintenance-, and upgrade decisions for software packages under different market conditions. Baseline for different practices/development policies based on customer satisfaction and aiming at long-term profit. Plan horizon is software upgrade cycle.

SABA: A novel two-phase decision model is presented to assist organizations in making decisions about legacy systems. The hardness of the decision problem is addressed. Legacy problem has (in past) been tackled only at the technical level, where it is seen as expensive, unproven, heavily staff dependent, not integrated with existing plans, and lacking cost-benefit justification. The objectives of the model are varied:

- 1) Provison of an analytical method to help organizations decide what to do with their legacy systems.
- 2) Based on the outcome of the previous point, help for deriving a transition plan from current to new practice.
- 3) Provison of an iterative method that can produce “first cut” results fast, and then successively better approximations.
- 4) Production of information from the various stakeholders (at various levels, regarding the appropriate target system).
- 5) Provison of a multidisciplinary approach (drawing from management theory, information systems, and software engineering).
- 6) Provison of a framework in which various technologies for legacy systems can be placed to meet a defined need.
- 7) Encouragement of support for complete software lifecycle.
- 8) Provison of means to analyze the future implications of software choices.

Renaissance: A method to support system evolution by first recovering a stable basis using reengineering, and subsequently continuously improving the system

by a stream of incremental changes. The approach integrates successfully with different project management processes.

VDM: The decision of how to rejuvenate an aging legacy system. The methodology can be used in the following three situations:

- 1) Determination of the technical and economic qualities of software system components,
- 2) Identification of the most suitable renewal process to be applied for each component.
- 3) Monitoring software system quality decay.

SRRT: Solution to the problem of timing software replacement/rewriting. The model assumes a finite fixed planning horizon. If rewriting speed is assumed to be linear to effort, optimal timings can be given in closed-form. The authors consider the effects of both linear and convex rewriting speed. The approach explicitly models the software degradation process and considers a more general problem scenario than the earlier similar models.

RPP: Project justification when selecting between legacy system reengineering, redeveloping and doing nothing at all. RPP is a general framework for estimating costs and benefits and also gives few advise for contracting.

RPFA: Highlighting some of the most important reasons for failures in reengineering efforts. Improvement of the reengineering state of the practice, evaluation of reengineering projects, and characterization of reengineering initiatives. Basis to start the planning of any migration effort. It is stated that one of the CMU's SEI's primary functions is transition of software technology. An important area is the migration of legacy systems to a more desirable target system (especially to a product line).

RMM: Clearly focuses on legacy system modernizations. Shows how to implement a successful modernization strategy. Describes a *risk-managed, incremental* approach, that encompasses changes in 1) software technologies, 2) engineering processes, and 3) business practices. Different architectures are also discussed relatively widely.

COCOMO II: Software cost estimation, especially determination of probable costs of planned large-scale software development projects. Addresses modern software processes and construction techniques. Estimates for required effort (man months) and required time (schedule) of a software development project. Determination of costs is then straightforward based on the determined required effort. There are many extensions planned or already defined, including the following of which most are widely discussed in Boehm *et al.* (2000).

- Object point data.
- Application point data.
- Phase schedule and effort estimation model (COPSEMO).
- Dynamic COCOMO.
- RAD (Rapid Application Development) schedule estimation model (CORADMO).

- Commercial-Off-The-Shelf (COTS) integration estimation model (COCOTS).
- Quality estimation (COQUALMO).
- Productivity estimation (COPROMO).
- Risk assessment (expert COCOMO).

FPA: Estimation of software development costs of administrative (and other types of) software systems. FPA has been developed further to multitude of versions and variants, including Reifer's version and 3D function points (Whitmire, 1995) for real-time systems, Symon's version, Feature points (*Jones, 1986), Full Function Points (*Garmus, 1996), Function Weight, Function Bang and Mk II Function Points Analysis (*Garmus, 1996). There has also been some recent attempts to use it in maintenance cost determination, including (Ahn, 2003).

Softcalc: Estimation of the efforts of incoming individual software maintenance requests as they arrive.

EMEE: Quick effort estimation before the operative phase of maintenance. Method is targeted at massive maintenance tasks (e.g. Y2K, Euro-conversion), which often require early (rough) estimates to support first resourcing decisions.

5.1.2 Start for methodology use

WMU: Need for determining appropriate strategies regarding software warranties, maintenance, and upgrades, insufficient customer satisfaction towards the software product supplied, or addressing following critical questions:

- "How often to maintain or upgrade the software?"
- "Whether to immediately respond to change opportunities?"
- "How to respond to technological obsolescence, and what is the measurement of the loss induced?"
- "Impacts of supplier's revenue structure, market volatility, and system quality on software-change decisions?"

SABA: It is stated that one of the major difficulties related to legacy systems is trying to decide rationally among very different options, ranging from discarding the old software completely, through reverse engineering, to freezing it, or outsourcing. Emphasized problems with legacy systems include: large scale, old age, obsolete languages, lack of consistent documentation, poor management of data, degraded structure, and support team which is reliant on experts.

Renaissance: A legacy system has reached a phase where continuing to maintain the system is expensive and may be ineffective in accommodating necessary changes.

VDM: The aging of a legacy system, *i.e.* its economic returns or quality are lower than expected, and the problem owner has to decide how to rejuvenate the system.

SRRT: The effort required to service maintenance requests on a software system increases as it ages. Thus it may be economical to replace it with a rewritten one.

RPP: Need for decision support in estimating whether reengineering is cost-effective and preferable to other alternatives.

RPPFA: Reengineering efforts are replete with examples of failures (which are similar to the traditional problems of IS-projects; meeting not requirements, schedule, and budget).

RMM: A critical choice for organizations is: whether to completely replace older systems, or incrementally incorporate new technologies into them. Many businesses choose the latter course, seeking to maximize their existing investment while adapting to a rapidly evolving technology.

COCOMO II: It is stated that competitive advantage increasingly depends on developing software for smart, tailorable products and services, and on the ability to develop and adapt these products and services more rapidly than competitors. These trends highlight the need for strong capabilities to accurately estimate software costs. Need for making financial decisions, setting project budgets and schedules, negotiation of tradeoffs, planning of maintenance or upgrade of legacy products, and deciding of where to implement process improvement.

FPA: Need to obtain an estimate of software development effort.

Softcalc: Need to know the effort required to perform the software maintenance task at hand.

EMEE: Need for an estimate before the operative phase of maintenance.

5.1.3 Customers and problem owners

WMU: Software suppliers, software change managers (also the needs of software user organization may be taken into account).

SABA: Software suppliers, engineers, and other stakeholders (also the needs of software user organization may be taken into account).

Renaissance: Software suppliers, software change managers.

VDM: Software suppliers, software change managers.

SRRT: Software suppliers, information system managers.

RPP: Software suppliers, software maintainers.

RPPFA: Software suppliers, technical or reengineering managers, project managers.

RMM: Software suppliers, technical or software change managers (also the needs of software user organization may be taken into account).

COCOMO II: Software suppliers, project managers.

FPA: Software suppliers, project managers.

Softcalc: Software suppliers, software maintainers.

EMEE: Software suppliers, software maintainers.

5.1.4 Context description

WMU: It is stated that the phenomenal expansion of the software markets requires substantial resources to monitoring products and managing their change.

SABA: It is stated that the represented model covers the often neglected business strategy modelling from a top-down perspective, involving many stakeholders. There exists the Organizational Scenarios Tool (OST) as one of the model's main components.

Renaissance: Organizations have their own established processes and tools. To be successful, the method should be sufficiently flexible so that it can support this variety.

VDM: With components' quality and economic values it is possible to monitor software system quality decay. The decision model must be specialized to the company and adapted to the targets set each time a renewal project is planned.

SRRT: It is stated that software suppliers incur huge maintenance cost because of: 1) a volatile user environment, and 2) deteriorating software maintainability.

RPP: It is stated that no one is keen on reengineering and it is usually a compromise, because new system is too expensive, old system is unmaintainable and commercial package is not available. Nevertheless, valid arguments of reengineering are needed in justification of project to managers and customers.

RPFA: It is stated that the purpose of the approach is high-level reengineering risk analysis in an enterprise-wide context.

RMM: The user organization.

COCOMO II: The approach includes its calibration within its user organization.

FPA: N/A. Description of the context of solution (the estimate) is implicitly defined by the use situation.

Softcalc: N/A. Context of the solution is implicitly defined by the use situation and therefore Softcalc doesn't pay attention to describing it.

EMEE: N/A. Context of the solution is implicitly defined by the use situation.

5.1.5 Culture and politics of methodology use

WMU: The model could be used as a decision tool. A software change manager could use the model to identify the policy type that best fits a system. The framework provides managers with the following two vehicles for decision making:

- 1) A guideline for the data needed to monitor software change process, the ability to manage the change by experimenting with data and various policies, the ability to set action guidelines, such that only unusual cases could be managed on an exceptional basis.
- 2) Ability to categorize the portfolio of the company's software systems based on the policies they require and have a formal framework to connect the nature of the market with the actions required (internal and external strategies) in order to remain competitive.

SABA: It is stated that the analysis stage seems to be very effective in pruning out solutions that are technically not viable or that do not fit with company strategy. By producing prioritised candidate solutions, the final phase of the model application can focus effort and time on a detailed exploration of just these to recommend a solution.

Renaissance: Framework is well-defined and easy to follow. Its use requires long term goals.

VDM: With software components' quality and economic values it is possible to monitor software system quality decay. The decision model must be specialized to the company and adapted to the targets set each time a renewal project is planned.

SRRT: It is assumed that the necessary input data is available. It is stated that rewriting a software system should be done in conjunction with proper control over the quality of the new software system.

RPP: A policy of using software measurement program is required.

RPFA: It is stated that the report will have succeeded if it raises the general awareness of the potential problems that are most likely to occur in real reengineering efforts. It is expected that readers attempting a reengineering effort will recognize potential hazards from among the represented real cases and will be able to redirect their efforts to avoid most of them. It is stated that the model could be used as a decision tool.

RMM: It is stated that risks must be managed throughout the modernization effort. It is also stated that plans must be well-considered and plausible, and that

the possibilities to modernization should be identified and used in an extent which is not too risky. The authors also underline the importance of creatively finding a better way to do things (*e.g.* by considering the underlying principles) rather than trying to improve the way things have been done in the past (related to modernizations).

COCOMO II: The approach fits well together with long-term process development. Existence of statistics within the organization on earlier projects is beneficial for the model's use.

FPA: Not explicitly stated. FPA assumes disciplined work culture (use of evaluation criteria) and a policy to collect metrics required to calibrate the applied productivity table.

Softcalc: Not explicitly presented. Implicitly Softcalc requires the organization to have a policy to collect the metrics needed to calibrate the method.

EMEE: Not explicitly stated. Implicitly EMEE assumes the organization to have basic culture in understanding and use of statistical tools and a policy to collect the metrics required to calibrate EMEE.

5.1.6 Risks in describing context

WMU: Following three assumptions are made and mentioned.

- 1) Probabilities of warranty, maintenance, and upgrade opportunities do not change within the upgrade lifecycle under study.
- 2) Once a customer's goodwill is lost due to lack of action, it cannot be recovered by a later action.
- 3) A hierarchy of thresholds from warranty, to maintenance, to upgrade is assumed.

SABA: Although the problem is technical, there is generally no adequate decision method to help organizations decide on the investment strategy for the legacy system. Different options (for migrating the legacy system) may apply to different parts of the software, each with a different risk.

Renaissance: N/A.

VDM: N/A.

SRRT: The model is well-defined and the most detailed one dealing with optimal rewriting policies. It is more realistic than the one on which it is partly based on (Gode *et al.*, 1990), by taking into account the user environment and the schedule of rewriting. However, it's rather evident (although the article does not address it) that this model still doesn't take into account all factors necessary to determine the optimal replacement policy. The model *e.g.* does not consider the severity of changing requirements (*e.g.* technological obsolescence and changing regulations).

RPP: N/A.

RPPFA: It is stated that the failure modes and examples are represented not to denigrate or second-guess the organizations, but rather to provide awareness that the failures are not at all uncommon, and that there is a hope for a better way of doing business.

RMM: It is stated that modernization can be a cost-effective option, but modernizing is daunting in that it requires careful analysis of options, stakeholder interests, and multi-faceted tradeoffs, including technical, programmatic, and organizational considerations. Modernization should be approached with respect for the magnitude of the effort and a plan.

COCOMO II: Calibration of the model to meet the needs and peculiarities of the organization applying it is assumed.

FPA: N/A (FPA doesn't describe the context of the solution).

Softcalc: N/A.

EMEE: N/A.

5.1.7 Risks of methodology

WMU: Possible risks are not explicated in the article. The details of the model are represented in other articles (Sahin & Zahedi, 2000; 2001) published earlier. The model requires relatively large amount of input information. The authors have developed a software package, which was used to explore profit and policy patterns under various market conditions. It is unclear whether this package is required in applying the model (assumed not to be needed).

SABA: Detailed information gathering can extend over several months.

Renaissance: The overhead of adopting Renaissance for the first few projects is high. Overhead for managing small projects is high. The approach helps risk reduction and cost distribution.

VDM: The risk of assessing the business value and the costs for each of the components incorrectly or incompletely is stated explicitly.

SRRT: The represented model extends and specifies the model of Gode *et al.* (1990) and earlier work; Chan *et al.* (1994). The model assumes availability of relatively large input data set.

RPP: We feel that correctness and accuracy of the model depend on metrics program and analysts selected for software analysis.

RPFA: The following risk issues exist, although they are not explicated in the methodology. The check-list content may or may not be trivial depending on who is using it. The details of the use of the framework are not described (implicitly assumed to be self-evident). It is stated that, in some cases, the represented examples may be combinations of two or more different situations, or the “facts” may have been altered in significant ways.

RMM: The actual risk-assessment part is relatively small. The cost estimation is covered only shortly and based on other methods (FPA, COCOMO).

COCOMO II: The reliability of the approach is based on: 1) empirical project database included, and 2) judgment from experts who are part of the organization applying the model. Judgment of an expert or experts (the ones who know the situation in the organization) is used in determining the values characterizing the complexity of software development in the particular case as compared to average (nominal) case. Thus sufficient level of expertise is critical for reliable results.

FPA: The greatest single risk factor is the methodology user. Using function point analysis requires expertise in software development. It involves expert judgment and thus has weaknesses common with other similar methodologies. It also requires disciplined approach to work. Another point of failure is the regulation to guide the users. Organizations using FPA must either develop criteria for determining required weights and factors such as whether a particular entry is simple, average or complex, or use “standard” criteria such as IFPUG’s “Function Point Counting Practices Manual”. Organization failing to (develop and) implement working criteria will risk the reliability of results. It should also be kept in mind that the basic FPA variant is mainly meant only for evaluation of administrative systems. Kitchenham (1997) gives a good summary of the problems of using function points.

Softcalc: Softcalc assumes the availability of relatively versatile input data set (see ‘Implementing the designs’). It’s quite likely that impact domain analysis required by Softcalc isn’t readily available even if wide variety of maintenance data had been collected. Impact domain analysis must be performed for the previous maintenance tasks to obtain a maintenance task database needed to calibrate maintenance productivity curve(s). That might be a laborious task. In his article (1995a) Sneed gives an example, but no actual empirical validation of any kind. In fact he presents validation as “what is needed”. The risk of missing validation can be minimized by using established estimation method as the basis and not neglecting proper calibration. Softcalc is partly based on the ideas of FPA and COCOMO.

EMEE: The scope of the approach is limited to large scale maintenance projects on very large software systems. The approach isn’t widely used.

5.2 Methodology User

5.2.1 Users' motives and values

WMU: The developed framework is based on the assumption that the main business strategy of a software supplier is long-term profit maximization. Warranties, maintenance, and upgrades are aimed at increasing system quality, and thus keeping customer satisfaction at sufficient level.

SABA: The decision about what to do with the software tends to be initiated by engineers hoping for a tool-based solution, but as appreciation of the project scope becomes clearer, it is essential to call upon many different roles (stakeholders) within the organization to frame the decision.

Renaissance: Continued maintenance of a legacy system can be ineffective, but replacing the system is risky and the costs can be very high. The user needs a method to find the best solution to legacy software modernization and subsequently prevent the legacy phenomena from reoccurring. The problem owner must be prone to change and continuous improvement.

VDM: The user needs to find the best (the most cost-effective) solution for legacy system modernization.

SRRT: Presented examples of goals of information system manager include: 1) minimization of the total application maintenance effort over a planning horizon of 20 years by deciding to rewrite the software system with a superior technology platform and 2) to use the opportunity to impose strict quality control during the rewriting to ensure a high development quality for the new software system.

RPP: Reengineering and redeveloping projects need valid arguments when suggested to managers or customers. Proper planning is a one of the prerequisites for success in a large reengineering project.

RPFA: It is assumed that the users are concerned with general, organization-wide risk analysis and long term process improvement.

RMM: It is assumed that the users are rational, sufficiently creative, and concerned with managing software risks.

COCOMO II: It is assumed that the users are rational and concerned with long-term planning and cost estimation.

FPA: Common to all the effort estimation methods.

Softcalc: Softcalc's motivation is common with all the other effort estimation methods: the need to assign to a task minimal resources with which it can be completed in time and satisfying the requirements. And they all have efficiency as common implicit value.

EMEE: Common to all the effort estimation methods.

5.2.2 Needed abstract reasoning

WMU: The model is stated to be relatively simple for its user - a menu of 8 well-defined policy options to choose from. Application of the model is simple.

SABA: The wide evaluation process covers a wide spectrum of criterias.

Renaissance: Wide scope.

VDM: Includes quantification of components' quality and business value.

SRRT: Results are received via relatively simple calculations.

RPP: A simple process model to use.

RPFA: The use of the check-list is straight-forward.

RMM: Complex.

COCOMO II: The output form of the expert judgments is simple, but their determination relies on expertise on similar earlier projects, and capabilities to compare the situations and scale their differences.

FPA: The level of abstract reasoning needed when using FPA is comparable to Softcalc. Applying given rules requires more expertise than it requires reasoning.

Softcalc: Softcalc is relatively straightforward and using it doesn't require high level abstract reasoning.

EMEE: EMEE can be completely automated. In theory EMEE doesn't require user involvement at all and therefore no abstract reasoning of any kind.

5.2.3 Needed skills

WMU: On one hand, the actual use of the methodology is simple - a strategy is chosen based on customer satisfaction index. On the other hand, there are number of issues which would need to be known in order to make optimal decisions. The capability maturity of the organization applying the model might not be at such level that all the (implicitly assumed) information (including versatile set of factors related to technology, software implementation, cost structure, market situation and customer expectations) would be available without (additional long-term) process development (and gathering of metrics data, and forming of statistics).

SABA: The evaluation requires both economic, and technical expertise, and capabilities to judge the relative goodness of different options. The extent of information capture must be adapted to the solution routes identified, which can be done by expert software engineers. Application domain expertise is essential during the analysis phase to assess impact on and by the domain.

Renaissance: Thorough understanding of the company's business, external environment and application is needed when constructing context models of the system. The method defines a set of expert roles, which should be available when using the method. These roles are: application business expert, legacy functional expert and legacy implementation expert. Skills in using various metrics are needed.

VDM: The application of the model needs both technical and economic expertise. The user needs a thorough understanding of the legacy system. Also the user needs some experience on using metrics to measure software/component quality and size, and methods to calculate the business value of a business function (*e.g. Net Present Value*).

SRRT: The methodology in itself is simple, but it assumes very extensive input data set to be available whose gathering may be elaborate and or it might not be explicitly modelled.

RPP: Methodology requires knowledge of organizational/project values and goals in estimating benefits of a project and knowledge of software metrics in selecting proper software attributes. Business analysts are needed to assess business value of legacy system.

RPPFA: Understanding of the situation in the organization where reengineering is applied is required.

RMM: Application of the approach requires wide knowledge related to the ways that the organization operates.

COCOMO II: The model provides tables from which the empirically derived effort multipliers corresponding to the relative estimates provided by expert(s) can easily be fetched and then used as an input to the formulas included in the model. Thus, the application of the model is technically easy, but extensive expertise and experience on software cost estimation and features characterizing the software development within the supplier's organization are essential for receiving reliable expert input for the model.

FPA: Expertise in software development and disciplined approach to implementation of FPA are needed. The user has to be able to pull the required parameters out from usually rather incomplete description of the software system to be built. Disciplined working method is required to ensure the reliability of results.

Softcalc: User is required to have enough experience in software maintenance to be able to determine the actual impact domain of the maintenance task at hand and express it using metrics used in underlying estimation method. In addition the skills required by underlying (development) effort estimation method are needed.

EMEE: Calibration, when properly done, requires knowledge in statistics (regression analysis) and a tool to perform the analysis with.

5.3 Methodology

5.3.1 Problem situation and problem boundaries

WMU: The article lists the following general suggestions based on the made research:

- It is suggested that by maintenance actions the software supplier attempts to preserve its market share, whereas by the upgrade action, the supplier tries to add new customers and increase its market share.
- Corrections of errors are assumed not to increase customer satisfaction above that of a fault-free system.
- Very high customer satisfaction may beneficially trigger technology diffusion, learning, and so-called network externalities.
- Suppliers operating in high volatile markets have a higher optimal average return than others (this being true for all revenue structures, customer response patterns and technological structures tested in this research), due to large possibilities to upgrade the system.
- Supplier incurs a relatively higher loss due to technological decay in a more volatile market.
- Higher quality reduces the extent of loss due to technological obsolescence.
- Customer satisfaction, quality and market volatility have strong interactions.
- The proposed baseline policies are suggested to be robust under various market conditions, and cover almost all available options.
- There are also numerous other observations mentioned.

SABA: The general conclusions of the model development include the following:

- A substantial review of legacy systems has shown that technical solutions alone are unsatisfactory.
- A mechanism is needed so that all stakeholders in an organization can contribute to the decision process.

Renaissance: Four key requirements for a method to support system evolution are explicated:

- The method should support incremental evolution.
- Where appropriate, the method should emphasize reengineering, rather than system replacement.
- The method should prevent the legacy phenomena from reoccurring.

- It should be possible to customize the method to particular organizations and projects.

VDM: Visaggio refers to Sneed (1995b), Ransom *et al.* (1998) and Favaro & Pfleeger (1997) in order to understand the problem situation.

SRRT: The article lists the following general suggestions based on the made research:

- Complete rewrite of large applications should be avoided.
- Programming staff should be organized by application (to increase *familiarity*).
- *Rewriting schedule* should be compressed (to minimize *double maintenance*).
- Strict *quality control* should be imposed to maintenance (in order to achieve *low quality deterioration rate*).
- *Inferior current platform* implies *earlier replacement* and *compressed rewriting schedule*.
- Maintenance staff not being *familiar* with the existing system implies earlier replacement.
- Greater *functional complexity*, good *rewriting effectiveness* or poor *maintenance quality* imply compressed *rewriting schedule*.
- With higher *rate of maintenance requests*, rewrite appears later and replacement earlier.
- Better *initial quality* or poor *rewriting effectiveness* imply more relaxed *rewriting schedule*.
- *Potential savings from rewriting* don't come from one single feature alone, but from better *platform*, *quality* and *familiarity* and stringent *maintenance procedure* together.

RPP: Functional reengineering (adaptations and enhancements) is advised to be kept distinct from technical reengineering. Reason for separation is that test data constructed for software development phase can be used in functionality testing after reengineering.

RPPFA: The approach is based on check-lists published earlier and represented here in form of a 10-point framework. It can be used to probe the relevant involved management and technical issues.

RMM: N/A.

COCOMO II: The book representing the approach lists also suggestions based on the large experience of the authors.

FPA: N/A.

Softcalc: N/A.

EMEE: N/A.

5.3.2 Diagnosis of situation

WMU: There is a suggestion to note the (possibly available) specific information related to:

- *Customer satisfaction index* (for a system).
- *Implementation quality* (quantified by the probability of arising warranty opportunity via e.g. records of customer complaints and calls).
- *Volatility of market* (characterized by degree of competition, product infancy, number of competitors, lack of dominance by one or more competitors).
- *Supplier's ability to respond to market volatility*.
- *Supplier's cost structure* (unit or total costs of each warranty-, maintenance- or upgrade action for a system or for systems of similar type).
- *Supplier's revenue structure* (expanding or contracting markets, business cycles, economic trends, marketing strategy, name recognition).
- *Customer expectations* (regarding enhancements and added functionalities, measured by average warranty decay, maintenance decay, and upgrade decay via e.g. customer survey).
- *Technological decay* (obsolescence).

SABA: In the first phase of the model the OST (which is stated to be a disciplined approach) is used to generating scenarios for the organization's future (which may be more or less radical). The core of the OST-method is a matrix. The columns (scenarios) are: *status quo*, automate (IT substitutes human effort), informate (IT augments human effort), transform (IT restructures a set of tasks or processes), and possible others. The nine rows of the matrix represent criteria for assessing the scenarios. The criterias are as follows:

- 1) Boundary (the unit of analysis).
- 2) Vision (global summary of the unit).
- 3) Logic (rationale for vision).
- 4) Structure (of the organization).
- 5) Roles (organizational roles of people).
- 6) View of information (resource analysis).
- 7) Costs (major costs, both financial and nonfinancial).
- 8) Benefits (both financial and nonfinancial).
- 9) Risks (major sources of risk).

Renaissance: The initial state is an aged legacy system, which then needs to be reengineered or replaced. The legacy system is assessed through the following parameters:

- Business value assessment: 1) where the system is used in the process, 2) the relationships between the system and other systems, 3) the costs and changes required if the system was no longer available, 4) defects and problems with the existing system.
- External environment assessment: 1) hardware (vendor/supplier rating, maintenance costs, failure rate, age, ability to perform function, performance), 2) support software (licence costs, frequency of fixes/patches, quality of support personnel).

- Organisational infrastructure (type of organisation and system users, technical maturity of the organization, training procedures in the organization, skill levels of system support, organizational attitude to change).
- Application assessment (complexity, data, documentation, external dependencies, legality, maintenance record, size, security).
- Test bed.

VDM: The current situation is diagnosed through components' quality and economic scores. The following data is required for calculating quality score:

- *System components* (elementary unit taken as the basis of for quality assessment and for the decision on the renewal process to be applied).
- *Component quality* (each component is associated with a set of metrics).
- *Baseline* for each metric (the limit value considered satisfactory for particular metric and the quantitative quality target of the organization).
- *Weights* (express the importance of a metric and, thus, the organization's quality policy).

The following data is required for calculating economic score:

- *Business functions* (each component implements a business function or a part of it).
- *Component size* (organization can choose which size metric to apply, e.g. FP, LOC).
- *Business value* of each business function (often used method for calculating business value is the *Net Present Value*, but organizations can and must choose another method if business value can not be quantified directly).
- *Cs_{ass}* (cost price of the software system).
- *Cs_{man}* (cost of maintenance).
- *Cs_{adm}* (cost of operation of the system).
- *Cs_{ast}* (cost of assistance in the use of the system).
- *Cs_{dif}* (cost of diffusion of knowledge of the package to maintainers, managers, assistants and users).
- *Cs_{plt}* (cost price and management costs of the hardware and software platforms necessary for using, managing and maintaining the software system).
- *Cs_{org}* (cost of coping with bad or incomplete functioning of the software system).

SRRT: The approach assumes the availability of the following data sets as input. Model functions are the following:

- The *speed of the rewriting team* (function of *team size*).
- *Functional complexity* of the existing (and new) software system at time *t*.
- *Code quality* of the existing (and new) software system at time *t*.
- *Maintenance productivity* on the existing (and new) software system at time *t* (function of *complexity* and *quality*).

Model parameters are the following:

- *Effort* required to develop a function point equivalent of code with the *existing* technology platform; it reflects the structuredness of the existing technology platform.
- *Effort* required to develop a function point equivalent of code with the *new* technology platform; it reflects the structuredness of the new technology platform.
- *Marginal effort* required to deal with the *functional complexity* of the *existing* software system; it reflects staff familiarity with the existing software system.
- *Marginal effort* required to deal with the *deteriorating code quality* of the *existing* software system; it reflects staff familiarity with the existing software system.
- *Marginal effort* required to deal with the *functional complexity* of the *new* software system; it reflects staff familiarity with the new software system.
- *Marginal effort* required to deal with the *deteriorating code quality* of the *new* software system; it reflects staff familiarity with the new software system.
- *Code quality* of the *existing* software system when it became operational; it reflects the control imposed on code quality during the development of the existing software system.
- *Code quality* of the *new* software system when it becomes operational; it reflects the control imposed on code quality during the development of the new software system.
- *Deterioration rate of code quality* of the *existing* software system; it reflects the control imposed on code quality during the maintenance of the existing software system.
- *Deterioration rate of code quality* of the *new* software system; it reflects the control imposed on code quality during the maintenance of the new software system.
- *Functional complexity* of the *existing* software system when it became operational; it reflects the complexity of the functional domain of the software system.
- *Average complexity of each maintenance request*.
- *Average rate of arrival of requests*; it reflects the volatility of the business environment.

RPP: The methodology requires a software measurement program. Data from following attributes is needed: system's quality and business value, maintenance productivity, maintenance costs and project costs.

RPFA: The framework describes a structure and context for exploring reengineering decision analysis and disciplined approaches to system evolution based on expert judgment of the situation within the organization related to these questions. List of reasons why reengineering efforts fail is the following:

- 1) The organization inadvertently adopts a flawed or incomplete reengineering strategy (e.g. for transition, environment integration or process).

- 2) The organization makes inappropriate use of outside consultants and outside contractors (e.g. giving up control to them, not getting the one that the organization is paying for, time and material contracts often don't conserve time and material).
- 3) The work force is tied to old technologies with inadequate training programs (e.g. middle managers "see the trees, but not the forest", aging work force would rather not learn much, culture dependent on maintaining *status quo*).
- 4) Organization does not have its legacy system under control (e.g. guesses substitute for historical data, inadequate change processes, informality of all system management processes).
- 5) There is too little elicitation and validation of requirements (e.g. unsatisfactory baseline requirements, failing to recognize a large requirements delta).
- 6) Software architecture is not a primary reengineering consideration (e.g. combining subsystems into "federations", architecture is "in the eye of the beholder").
- 7) There is no notion of a separate and distinct reengineering process (e.g. a generic "paper-driven" process, non-existence of a process to use metrics).
- 8) There is inadequate planning or inadequate resolve to follow the plans (e.g. assigning the planning to a committee without clear leadership and empowerment, lack of focused technical management oversight and control, lack of direct control over semi-autonomous business units).
- 9) Management lacks long-term commitment (e.g. managing to one's expected lifetime in a position, disengaging from work).
- 10) Management predetermines technical decisions (e.g. firing the manager as a solution, assuming that management edicts can fix price, schedule, and function, saving money now without considering increases in future maintenance costs).

RMM: Assessment of the following issues at each step of modernization:

- 1) What can go wrong.
- 2) What risks are important.
- 3) Representation of strategies to deal with those risks.

COCOMO II: The approach requires as input: estimates (on a 5-point scale) for the following 17 post-architecture effort multipliers:

- 1) RELY (*Required Software Reliability*).
- 2) DATA (*Database Size*).
- 3) CPLX (*Product Complexity*).
- 4) RUSE (*Developed for Reusability*).
- 5) DOCU (*Documentation Match to Life-Cycle Needs*).
- 6) TIME (*Execution Time Constraint*).
- 7) STOR (*Main Storage Constraint*).
- 8) PVOL (*Platform Volatility*).
- 9) ACAP (*Analyst Capability*).
- 10) PCAP (*Programmer Capability*).
- 11) PCON (*Personnel Continuity*).
- 12) APEX (*Applications Experience*).

- 13) PLEX (*Platform Experience*).
- 14) LTEX (*Language and Tool Experience*).
- 15) TOOL (*Use of Software Tools*).
- 16) SITE (*Multisite Development*).

FPA: In principle, the data required is likely to be known early in the evolution of a project. Albrecht's original version requires the count of the following operations:

- User inputs.
- User outputs.
- User inquiries.
- Files (or storage entities in general, such as tables in relational database).
- External interfaces.
- 3D function points count also algorithms and state transitions.

Softcalc: The use of Softcalc is initiated by the result of diagnosis of the situation and therefore it's assumed to be known. Availability of the following, relatively versatile, input data set is assumed:

- Size of the *impact domain* of the maintenance task using selected size metric (FP, LOC, ...).
- *Complexity metrics* (a set of them covering different aspects of code *i.e.* data, control, interface..., the metrics are translated to a complexity factor).
- Internal and external *quality characteristics* (external: *reliability, security, integrity, usability, time efficiency* and *space efficiency*; internal: *portability, flexibility, readability, modularity, data independence, interoperability*) which are rated with expert judgment and translated to a quality factor.
- *Project influence factor* (defined with the means of underlying estimation method).

EMEE: Following input data is required:

- *Number of components* in a "work-packet" (incremental part of software in massive maintenance process).
- At least one *dimensional metric* (LOC was used).
- At least one *structural metric* (McCabe's cyclomatic complexity was used).

EMEE doesn't work without calibration and therefore the application of the model requires a sample set of maintenance projects and needed metrics from them: input data and actual realized effort. Code auditing tools are needed in order to obtain the required metrics.

5.3.3 Prognoses

WMU: The approach provides *thresholds* for (strategically, optimally) profitably initiating actions, related to *warranties* (W), *maintenance* (M), and *upgrades* (U). There are 8 different baseline policies described. They are recommendations to

start WMU-actions or their combinations based on baseline thresholds (in order to increase customer satisfaction or to keep it at the current level). Also maximum expected return (using a discount rate of zero) and (non-discounted) average returns are reported.

SABA: There are many options for the software modernization available. Following migration strategies are listed:

- 1) Discard (throwing all the software away and starting again from scratch).
- 2) Wrap (identification of legacy software interfaces, and encapsulation of them by an object which replaces the interface of the caller).
- 3) Outsource (letting an outside specialist organization to offer the service, which is deemed as not part of the core business).
- 4) Freeze (not performing further work on the legacy system).
- 5) Carry on (continuing maintaining the system for another period).
- 6) Reverse engineering (restructuring, recovering reusable artifacts, redocumentation, design recovery *etc.*).

For each scenario, the technical implications and possibilities are identified within the Technology Scenarios Tool (TST), for existing software assets. TST first produces a set of potential solutions (by identifying solution routes such as *e.g.* freezing the system), as well as technology and support portfolio (produced by a stage called information capture, including gathering such input information as: source codes, data maps, flow diagrams, and tools like: crossreference listers, call graph generators, and data analysis tools). Then preferred solutions are first identified (by a stage called analysis) and then detailed (including costs, risks, tool investment, staffing, expertise, and more technical information about the software itself). The outcome is a prioritized set of solutions for the legacy system.

Renaissance: The desired result is an evolvable system which supports continuous evolution.

VDM: Produced results are: classification of system components according to their technical and economic qualities and the selection of the best possible renewal process for each component. Other results include qualitative and economic thresholds for each component, and best candidates for modernization.

SRRT: Produced results are the following (it is assumed that replacement benefits are derived based on the saved person-hours):

- Optimal *time for starting rewriting*.
- Optimal *time for replacing* the existing software.
- Optimal *size of the rewriting team*.

RPP: N/A.

RPPFA: The framework lists technical solutions, related activities, practices, and work products which may be used to solve or alleviate the identified problems for each category.

RMM: RMM integrates software engineering concepts with an organized understanding of the information systems technologies that define the range of

possible solutions. Lists of current standards and available products supporting various facets of legacy system modernization are provided.

COCOMO II: Produced results are: required effort in the project (measured in man-months), required costs, and required calendar time. It is up to the model user to decide whether these estimates reflect a desirable outcome as compared to the resources (labor force, budget, schedule) available to the project.

FPA: Simply the required effort.

Softcalc: Simply the required effort.

EMEE: Simply the required effort.

5.3.4 Problem defining

WMU: The problem which the approach solves is simply the above-mentioned strategy-selection. The situation is characterized by parameter values.

SABA: The problem to be solved is fixed (selection of the best migration strategy for a legacy system).

Renaissance: The problem is the transformation of an aged legacy system into an evolveable system.

VDM: The problem is the aging of a legacy system. The legacy system is vital to the organizations business function but its quality or economic returns are lower than expected.

SRRT: The problem which the approach solves is simply the determination of the values of the above-mentioned three output variables. The initial situation is characterized by input parameter values.

RPP: The problem to be solved with the process is to find valid arguments on behalf/against of modernization.

RPFA: The problem simply comprises the identification of risks related to reengineering projects.

RMM: Guidance at abstract level.

COCOMO II: The application of the models is technically straightforward. There exists different levels of granularity for quick and more elaborate and precise evaluation as well as for the different kinds of software development situations (described as model extensions).

FPA: N/A.

Softcalc: N/A.

EMEE: N/A.

5.3.5 Deriving notional systems

In these sort of models the description of the target state is mainly trivial (selection of the variable to be optimized or there is only one assumed way to optimization). Note that the term *notional*, which is used by Jayaratna, refers to speculative or imaginary future (systems), not to *notations*.

5.3.6 Design

WMU: N/A. Baseline policies are simply selected based on the input information provided by the model user.

SABA: The process is multi-phased and iterative. Thus this phase is covered, unlike in most of the other comparable approaches. The phases form a flow (OST-use, TST-use, and its sub-phases). What is considered to be design of the solution and what the actual solution-process is a matter of opinion.

Renaissance: In *Plan Evolution* –phase of the process framework there are three key activities which constitute more abstract design: *method calibration*, *system assessment* and *evolution strategy development*. The *Plan evolution project* – activity of the *Implement*-phase constitutes the less abstract design of the solution.

VDM: N/A.

SRRT: N/A.

RPP: N/A.

RPFA: N/A. The included list of potential risks is simply checked out.

RMM: N/A.

COCOMO II: The approach includes the selection of the appropriate model variants to be applied.

FPA: N/A.

Softcalc: N/A.

EMEE: N/A.

5.3.7 Implementation

WMU: Model's two axis are *upgrade cycle* and *customer satisfaction index*. Their relation is characterized with a picture. The form of the needed (numerical) input and output information is simple. The following example is provided: assuming a maximum (customer satisfaction) index value of 100, and given the technological, maintenance, and upgrade decays of 2%, 6%, and 20%, respectively, the maintenance baseline is $M = 100 - 6 - 2 + 1 = 93$, and the upgrade baseline is $U = 100 - 20 - 2 + 1 = 79$.

SABA: There exists the above-described OST- and TST tools.

Renaissance: Following phases of the model are related to this issue:

- The *Implement*-phase of the process framework has two key activities for implementing the designs: *Design, transform and test system* and *Prepare environment*.
- *Deliver phase (Migrate data, Install system, Train operators)* entirely relates to this.
- *Deploy and use phase (Changeover system, Evaluate system, Document environment changes)* are part of implementing the designs, and entirely relate to this.

VDM: The process is as follows:

- Definition of the system components.
- Calculation of a *quality score* for each component.
- Calculation of an *economic score* for each component.
- Choosing of an appropriate renewal process for business critical components according to both quality and economic scores and the aims (e.g. costs to be reduced and improvable characteristics) of the renewal process.

SRRT: An example illustrating the application of the model is provided in case of deriving optimal software replacement policies for a COBOL software system. Input and output parameters are listed, while the actual applied formulas are only referred to.

RPP: The process model includes the following five steps:

- *Project justification:* analyzing software, maintenance process, business value of software and calculating *return on investment*.
- *Portfolio analysis:* estimating the need of reengineering according technical quality and business value.
- *Cost estimation:* calculating the cost of reengineering by assessing cost of components to be reengineered.
- *Cost-benefit analysis:* comparing the estimated costs to benefits of reengineering, redeveloping and doing nothing.
- *Contracting:* including task definition and effort distribution.

RPFA: The represented risk-list is checked out manually, and corresponding solutions, examples, and explanations identified.

RMM: The application process consists of the following phases:

- Portfolio analysis.
- Identification of stakeholders.
- Understanding requirements.
- Creating the business case.
- Understanding the legacy system and existing technologies.
- Evaluating the technology.
- Defining target architecture and modernization strategy.
- Reconciling the strategy.
- Estimating the needed resources.

COCOMO II: The model includes its implementation (a software package). Estimates for required effort are received from the represented formulas. For software maintenance the size of the project is normally obtained via applying either one of the two following equations:

- $MCF = (Size\ Added + Size\ Modified) / Base\ Code\ Size$, or
- $Size_M = (Size\ Added + Size\ Modified) \times MAF$, where
- $MAF = 1 + (SU/100 \times UNFM)$,
- MCF is *Maintenance Change Factor*,
- MAF is *Maintenance Adjustment Factor*,
- SU is *Software Understanding*, and
- $UNFM$ is *Programmer Unfamiliarity*.

FPA: Implementation has the following four phases:

- Counting operations.
- Weighting the counts with expert-judged *complexity factors* and summing up the counts. *Complexity factors* are determined very much similarly as in COCOMO.
- Multiplying the result by *value adjustment factor* derived by evaluating given list of development cost drivers.
- Translating the resulting *FP count* to *effort* with productivity table.

Softcalc: Softcalc is carried out in the following seven steps:

- *Size*, *complexity* and *quality* of software are determined by automated audit.
- The *impact domain* of the software affected by the planned maintenance action is circumscribed.
- The size of the domain is measured with two or more of the following size metrics: LOC, statements, FP, data-points, object-points.
- The size measure is adjusted by a *complexity factor* (from the first step).
- The size measure is adjusted by the *external* and *internal quality factors*. The latter (which reflects the software maintainability) is also obtained automatically from a code audit.
- The size measure is adjusted by a *productivity influence factor* depending on the estimation method used.
- The *adjusted size measure* is transposed into *maintenance effort* by means of a productivity table.

EMEE: The actual use of EMEE consists of substituting metrics in model formula and calculating effort.

5.4 Evaluation

5.4.1 Internal evaluation

WMU: No specifications for really reflective evaluation.

SABA: Iteration may be used within the organization applying the model both to refine the solutions, and to explore the consequences of the model for the future of the organization. It has also been stated that the work is based on earlier works by Foster (1993), Sneed (1995b), and Renaissance project (Ransom *et al.*, 1998), which reflects gradual method development.

Renaissance: The method is customized to take into account the particular project and organizational factors. The feedback after each increment can be exploited and this also gives a chance for reflective evaluation. The method has no external validation but the authors have received a great deal of feedback and refinement suggestions from industrial partners of the Renaissance project.

VDM: There exists longitudinal data gathering and method development phase at the background.

SRRT: N/A.

RPP: N/A.

RPFA: N/A.

RMM: In principle, incremental modification.

COCOMO II: The model is based on empirical data, and it has a long development history, experiences have also been gained from the relatively wide long-term use of the older project database applied in COCOMO's first version. As noted earlier, calibration is assumed to be performed within the organization applying the model.

FPA: N/A.

Softcalc: Although strongly emphasizes proper calibration of maintenance productivity tables, Softcalc doesn't evaluate itself in any way.

EMEE: Nothing in addition to calibration.

5.4.2 External evaluation

WMU: The model is based on mathematical analysis of the possible (basic forms of) functions depicting the strategic situations. The model is “tested” with 3840 base line scenarios producing 26880 optimal policies (these were used in a sensitivity analysis). Almost all of these, in turn, can be reduced to the represented 8 baseline policies. There is no mention of the possible actual empirical evaluations. Possible relation to real industry-level practice is unknown. The form of the functions is seemingly based on other earlier empirical studies performed by others and by logical deduction.

SABA: The individual components of the model are based on synthesis of existing work from software engineering, and from information systems. It is stated that the concept of scenario-based methods has a long and well tested history. OST has been empirically tested in a number of domains. TST represents the formalization of experience in undertaking projects on very large industrial-scale legacy systems. An example of the use of TST is provided based on consultancy carried out by members of the project team prior to the start of the SABA project. Empirical evaluation of the model as a whole is stated to be the next step of the model development.

Renaissance: Evolution strategy selection was proven useful.

VDM: The method is based on a “real-world” renewal project on a banking system consisting of 653 programs and 1.5 million instructions observed over about 2 years.

SRRT: The model is mainly based on analytical and simulation solutions. The article gives an illustrative example of the model use with imaginary parameter values. The results deduced from the model have not been validated empirically, but model parameters and assumptions are based on field data. Constant maintenance request arrival rate is supported by field data from 10 applications over a 7-year period. Assumptions concerning *system growth* (Lehman *et al.*, 1998), *maintainability* (*Gibson & Senn, Jones; Kafura & Reddy, 1987), *complexity* increase (*Swanson & Beath), and *maintenance productivity* (*Chan & Ho) are taken from the results of the separate research projects received by the mentioned researchers.

RPP: Testing or validation are not mentioned. The process is developed by the author based on his accumulated 25 years of experience on software engineering.

RPFA: The model is based on empirical data. The represented suggestions are based on a variety of experiences over many years related to reengineering projects both government and industrial organizations. The example situations are taken from real experiences known first hand to authors or related directly to them. There are either no quantitative results or they are not published. The results are not published in any refereed scientific forum.

RMM: The book (Seacord *et al.*, 2003) describes an approach developed in CMU's SEI based on methods and techniques described in (Wallnau *et al.*, 2001). The book is recommended by emer. prof. M. Lehman (who is one of the most reknown scientists in the field of software maintenance/evolution) as a *must* for all people directly involved in such evolution. The method is illustrated via a large retail supply system case study ("real-world legacy system modernization effort, a COBOL-coded, nearly 2 MLOC, system, developed over 30 years, being replaced with a system based on J2EE architecture).

COCOMO II: COCOMO is a traditional (Boehm, 1981) and perhaps the best known method for estimating software costs. COCOMO II (2000) is an updated version of the classic model. Both versions of COCOMO are based on empirical data about real software development and maintenance projects. This data forms a project database. In COCOMO II's case there are 161 industrial reference points, *i.e.* actual projects, whose data is stored into the database. The represented formulas for output variables are based on this data. Boehm has earlier stated that the average error of the so-called intermediary model of COCOMO (*i.e.* the year 1981 version) is "only" 20%. It can be assumed that the newer version should produce at least as good results. The model is (*i.e.* should be) calibrated within the organization applying it. Boehm and his co-authors represent a specific (sub) method for this based on Bayesian models.

FPA: The approach is based on a strong set of empirical data. Due it's age and wide distribution it has been a target for many scientific studies *e.g.* Dolado (1997), Lee *et al.* (1998), Yau & Tsoi (1998), Locan (2000). Reliability and generalizability of FPA, however, are questionable, see *e.g.* Kitchenham (1997); Abran & Robillard (1993); Abran *et al.* (2002).

Softcalc: Sneed gives an example, but no actual empirical validation of any kind. In fact he presents validation as "what is needed".

EMEE: The presented model was validated against data collected from a large Y2K remediation project. There were 40 KLOC of components, and 15 KLOC of them were modified. The programs were written in COBOL/CICS, PL/1, JCL, and Assembler. A leave-one-out cross-validation was performed. Average prediction error was 47% with 10% sample, 42% with 30% sample and 35% with full data.

5.5 Summary

We have summarized the differences between the reviewed methods for estimating modernization efforts in Tables 2a, and 2b (provided as single table in appendix 1). All approaches focus on software, and are intended to be used by software supplier organizations.

Generalized NIMSAD elements	WMU	SABA	Renaissance	VDM	SRRT	RPP
M Context						
Use situation	Warranty, maintenance, upgrade decision making	Legacy transition strategies, organizational & technical analysis	Basis for reengineering & incremental improvement, integrates with project management	Component-level techn. & econ. analysis	Timing of replacement & rewriting	Cost & benefit analysis of evolution strategies
Start for M use	Strategically optimal customer satisfaction?	Evolution & transition strategies?	Evolution of degrading software?	Rejuvenation strategies of aged software?	Profitability of rewrite?	Profitability of reengineering?
Customers <i>etc.</i>	Sw. change managers	Sw. engineers	Sw. change managers	Sw. change managers	IS managers	Sw. maintainers
Context descr.	Sw. change control, product monitoring	Includes org. business strategy modelling	Org. context: processes & tools	Sw. quality control, renewal project	Volatile user environment, deteriorated legacy sw.	Deteriorated legacy sw., unavailability of proper tools
Culture <i>etc.</i>	Decision making support, change monitoring	Business strategy driven prioritization	Long term goal setting	Specialization to the organization and project	Availability of the input data required, sw. quality control	Sw. measurement program policy required
Risks (context)	Stable upgrade lifecycle assumed	Sub-systems may differ	N/A	N/A	Incompleteness of the decision criteria	N/A
Risks (M) (see also Evaluation)	Model details presented elsewhere	Information gathering may extend several months	High adoption overhead, well-defined and easy to follow, helps in risk reduction	Inaccurate assessment of component's business value and cost	Availability of reliable input data	Reliance on the quality of metrics program & analysts' expertise
M User						
Motives/values	Long-term profit maximization, optimiz. of customer satisfaction	Both technical and other considerations should be covered	Proneness to change & continuous improvement, prevention of new legacy problems	Rationality	Long-term cost estimation, sw. quality control	Argumentation, planning
Abstract reason.	Limited	Wide, including judging options	Wide	Estimation of sw. components	Limited, the model's use is simple	Limited, the process model is simple
Needed skills	Simple strategy selection & possibly prior elaborate metrics program developments	Economic & technical expertise & planning of information gathering	Expertise on application domain & legacy system's functional & technical issues	Components' quality & business value quantifications	Limited, but possibly requires metrics gathering process development	Knowledge on organizational & project goals & software metrics & business value estimation

M						
Problem situat.	About 10 suggestions of the nature of the problem	Some suggestions	4 suggested requirements	Other sources are referred for this	About 10 characterizations of the problem	Functional and technical reengineering to be separated
Diagnosis	8 factors (about 20 subfactors)	OST (4 main scenarios and 9 criterias)	4 categories of factors (about 30 subfactors)	Quality score (4 subcategories) & economic score (10 subcategories)	4 functions and 13 parameters	5 factors
Prognoses	Recommendations of maintenance actions (& expected ROI)	6 listed migration strategies, TST: prioritized set of solutions	Goal is sw. which can be continuously evolved	Classification of components, and their optimal renewal process	Time for starting rewrite and replace, size of the rewrite team	N/A
Problem def.	Fixed, characterized by parameter values	Fixed	Fixed, general	Fixed	3 specific subproblems, characterized by parameter values	Fixed, general argumentation
Notional syst.	N/A	N/A	N/A	N/A	N/A	N/A
Design	N/A	Multi-phased (OST, TST) iterative process	Method calibration, system assessment, evolution strategy development, evolution project planning	N/A	N/A	N/A
Implementation	Simple calculations	OST- and TST tools	3 phases (and 8 subphases)	4 phases	Application of the represented formulas	Project justification, portfolio analysis, cost estimation, cost-benefit analysis, contracting
Evaluation						
Internal eval.	N/A	Method has been developed gradually, iteration may be used to explore the consequences	Feedback after each increment can be used, the model is customized for the project	Longitudinal data gathering during method development	N/A	N/A
External eval.	Mathematical analysis of function forms (3840 base line scenarios)	Synthesis of earlier works, OST: empirically tested, TST: based on experience, model: to be tested	Feedback from industrial partners	Observations of a real-world renewal project (653 programs)	Based on analytical and simulation solutions, model parameters and functions are based on various earlier empirical field data studies	Based on the author's 25 years of sw. engineering experience

Table 2a. Summary of methodologies using generalized NIMSAD framework
(to be continued)

Generalized NIMSAD elements	RPFA	RMM	COCOMO II	FPA	Softcalc	EMEE
M Context						
Use situation	Reengineering risk analysis	Risk-managed & incremental modernization strategy, sw. architectures covered	Cost estimation for large-scale projects	Cost estimation	Maintenance request effort estimation	Quick & early effort estimation of maintenance tasks
Start for M use	Risks of reengineering?	Strategy to reuse the old and adapt to changing technologies?	Project budget, schedule, and maintenance plan?	Software development effort?	Effort of the current maintenance task?	Early estimate for maintenance task effort?
Customers <i>etc.</i>	Technical & reeng. & project managers	Technical & sw. change managers	Project managers	Project managers	Sw. maintainers	Sw. maintainers
Context descr.	Enterprise-wide context	Organization	Rapid development & tailoring of sw. products	Use situation	Use situation	Use situation
Culture <i>etc.</i>	Improvement of risk awareness	Disciplined risk management, creative problem solving	Long-term process improvement, system metrics & statistics	Disciplined work culture, metrics collection policy	Metrics collection policy	Understanding of statistical tools, metrics collection policy
Risks (context)	Proper process improvement attitude is required	Complex, multi-faceted problem & varying effort magnitude	Calibration is required	N/A	N/A	N/A
Risks (M)	Represented examples may be combinations or may have been altered	Risk assessment & cost estimation parts are limited, based on FPA & COCOMO	Sufficient level of expert judgment is critical	Discipline and expertise assumed	Availability of relatively versatile, reliable input required, model details?, laborious impact domain analysis, based on FPA & COCOMO	Limited scope & limited prior use
M User						
Motives/values	General, organization-wide risk analysis and process improvement	Rationality, creative problem solving, risk management	Rationality, long-term planning, & project estimation	Rationality, long-term planning & project estimation	Like in case of FPA, and COCOMO	Like in case of FPA, and COCOMO
Abstract reason.	Very limited	Complex	Comparison of projects and judging complexity factors	Limited	Limited	Very limited
Needed skills	General organizational & situational awareness	Wide knowledge of the organization	Model use is simple, but thorough expertise on project cost estimation and the project's characteristics is required	Disciplined approach to the gathering of the required input based on somewhat vague definitions	Experience on software maintenance, impact domain determination, general cost estimation knowledge	Knowledge on statistics

M						
Problem situat.	The checklist characterizes the situation	Example case study is described	Various suggestions	N/A	N/A	N/A
Diagnosis	10 factors	Problems, risks, strategies	17 effort multipliers	5-7 factors	About 20 factors	At least 3 factors
Prognoses	Lists of relevant activities, practices, technical solutions and tools	Lists of current standards and available products	Required effort, costs, calendar time	Required effort	Required effort	Required effort
Problem def.	Fixed, 10 subproblems	Guided at abstract level	3 main subproblems, modes of “granularity”, many extensions	N/A	N/A	N/A
Notional syst.	N/A	N/A	N/A	N/A	N/A	N/A
Design	N/A	N/A	Selection of model variants	N/A	N/A	N/A
Implementation	Check-out of the represented risk-list	9 phases	Application of the represented formulas, expert judgments	4 phases	7 phases	Application of the represented formulas
Evaluation						
Internal eval.	N/A	Incremental modernization makes it possible in principle	The method has long development history, calibration is required	N/A	Calibration is emphasized, but reflective evaluation is not explicated	Calibration
External eval.	Variety of experiences over many years	Illustrated with a “real-world” case study	Based on 161 industrial projects	Strong set of empirical data, various studies	None	Empirical validation, with a 40 KLOC project, minimum prediction error was 35%

Table 2b. Summary of methodologies using generalized NIMSAD framework (continued).

Our experiences of the use of NIMSAD include the following observations: we have shown the results such that ‘Risks of methodology’ include also those limitations of empirical validation which are represented in ‘Evaluation’ -part, since these two elements are otherwise typically, and unavoidably partly overlapping in content. Also ‘Needed skills’ -part overlaps with ‘Risks’ if skills are not sufficient and the methodology is still applied. ‘Use situation’ and ‘Start for methodology use’ are also partly overlapping in content, as well as ‘Culture and politics of methodology use’ and ‘Users motives and values’. Also, the contents of the ‘Use situation’ and ‘User motives and values’ -elements overlap if users adequately know when to apply a method. For the analysis of ‘Problem situation’ there generally are no strict sub-methods available, instead there are characterizations of the problem area and its general nature. In practice, ‘Prognoses’ depicts the main results (output) of the methodology in these cases. In many cases ‘Problem definition’, ‘Design’, and ‘Implementation’ merge into one phase.

6 Support techniques for modernizations

Large “real-world” applications are complex and thus their maintenance is a non-trivial task. While the decision to modernize has first been made, there is a need to perform the task by using technically sound and practical methods and tools. In this chapter we shall chart and characterize the main options available (due to limited resources allocated to this task our analysis is only a preliminary survey).

6.1 Modernization approaches

6.1.1 Integration

Integration as a form of modernization aims at replacing possibly outdated access protocols and components implementing the data transfer by their standard counterparts. For example XML and CGI can be used as aids in software integration. These issues are discussed *e.g.* by Robertson (1997) and Comella-Dorda *et al.* (2000).

Use of Extensible Markup Language (XML) has expanded from its origin in document processing and together with http-protocol now provides a solution for data integration (Comella-Dorda *et al.*, 2000). Wide tool support and general acceptance distinguish XML-http-platform from home-brewed data formats and communication protocols. Diminishing, but still apparent, risk factor is the current evolving situation. XML technologies and tools are not quite mature yet and XML itself is still evolving (XML 1.0 became W3C recommendation 6 October 2000 and XML 1.1 is at stage of candidate recommendation writing).

Key component in XML-integration is the XML server which is a mediator between legacy system and external systems (possibly another organizations) communicating with them. Most of the commercial XML servers support variety of interfaces and are therefore easily integrated with most usual legacy applications. With tailored legacy software having proprietary nonstandard interfaces or possibly no interfacing at all, the situation is quite different. Then the

risks are higher and the costs of integrating legacy system with XML-server must be carefully estimated.

Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers (*e.g.* Web-servers). CGI integration is a common way to provide web-access to existing resources. When considering using CGI integration it's worthy to remember that web-platform has it's roots in sharing information among researchers (<http://www.w3.org/History.html>) and it also currently provides best value and lowest risks when used in situations similar to the original (information publishing/retrieval) purpose, such as replacing proprietary information retrieval client with web-platform and standard web-browser.

Integration together with (object oriented) wrapping is useful for decomposing and decoupling large legacy systems into separate components (Matjaz *et al.*, 2000). Dividing legacy systems into components permits conquering them incrementally.

Robertson (1997) suggests that a significant success factor when integrating legacy systems to object oriented environment is the availability and appropriate use of reflective capabilities (meta objects) of the integrating OO environment.

6.1.2 Wrapping

Modernization by wrapping means building a new 'facade' which encapsulates the legacy resource and provides up-to-date and standard interfacing to the wrapped part of the system. Wrapping can be performed practically at any level. Common levels discussed are user interface (sometimes called screen scraping, Comella-Dorda *et al.*, 2000), business logic (Robertson, 1997) and components or enterprise resources in general (Matjaz *et al.*, 2000; Comella-Dorda *et al.*, 2000). Wrapping and encapsulation are discussed by also by Sneed (2000).

Wrapping and integration (discussed earlier) are closely related. Comella-Dorda *et al.* (2000) classify XML integration as a form of data wrapping and CGI integration as a tool to implement user interface wrapping and user interface renovation.

Encapsulation of (parts of) legacy system into objects, components or services is in general a very flexible way of modernization (Comella-Dorda *et al.*, 2000). But as usual, flexibility doesn't come without the price tag and adds the risk of inappropriate design and implementation of the modernization. Wrapping resources to objects or components using distributed object or enterprise component -related technologies like CORBA, EJB, DCOM and Java RMI is an established way of "conquering" legacy systems. There are several advantages to this approach for wrapping legacy business logic. With limited effort the advantages of component-based systems are supported. Wrappers are genuine components in the chosen environment and thus subject to all the management facilities available on the application server. Wrapping provides a "roadmap" to

substitute the old system (if that is the aim) incrementally and resource intensive “big-bang” replacement is avoided.

6.1.3 Migration

Migration means transforming legacy systems to new environments without having to completely redevelop them. These issues are especially discussed by Brodie & Stonebraker (1995), Bisbal *et al.* (1997) and Seacord *et al.* (2003). Here we shall refer to Bisbal *et al.* (1997) and Wu *et al.* (1997).

The goal of migrating software to new environments is to improve information systems’ maintainability and adaptability to new business requirements, while retaining the functionality of existing information systems. A migration project should cause as little disruption to the current business environment as possible. Considering the scale, complexity and risk of failure in migration projects, a need for well-defined, detailed and easy-to-implement methodologies exist. Few such methodologies have been developed:

- *Big Bang –approach* (Cold Turkey). This is a ‘from a scratch’ approach to migrating legacy systems. The system is completely redeveloped to a new hardware platform using a modern architecture, tools and databases. The main weakness of this approach is a great risk of failure.
- *Forward Migration Method* (Database First). The legacy data is first migrated to the new environment. After the data migration, the rest of the applications and interfaces are incrementally migrated. This approach involves the use of a Forward Gateway, which enables the legacy applications access the migrated data in the new environment.
- *Reverse Migration Method* (Database Last). Reverse Migration Method is an approach, in which the legacy applications are first gradually migrated to the new environment but the legacy data remains in the original platform. Legacy data migration is the last phase of the migration process. In this approach the applications in the new environment use a Reverse Gateway to access legacy data.
- *Composite Database –approach* (e.g. Chicken Little -methodology). This approach uses both Forward and Reverse Gateways. The target applications are gradually rebuilt with modern tools and technology. During migration process the target system access legacy data through reverse gateway and legacy applications access target data through forward gateway. Sometimes data is duplicated across both databases, which causes problems with data integrity. For this reason a *co-ordinator* is used. Co-ordinator intercepts all requests and determines what updates have to be made in both databases.
- *The Butterfly Methodology* (Gateway free –approach). Objective of this methodology is to migrate a mission-critical legacy system to a target system in a simple, fast and safe way. When using the Butterfly Methodology the target system will not be in production while the legacy system is being migrated. According to this approach a gateway is not needed while migrating legacy systems. The Butterfly Methodology consists of the following six phases: 1) prepare for migration, 2)

understand the semantics of the legacy system and develop the target data schema(s), 3) build up a Sample Datastore, based upon the Target Sample Data, in the target system, 4) migrate all the components (except data) of the legacy system to the target architecture, 5) gradually migrate the legacy data into the target system and train users in target system, and 6) cut-over to the completed target system.

6.1.4 Language conversion

Language conversion is a one way to modernize a legacy system. It is a considerable option for example when support of the original language is not available anymore. New language can also be adapted to use by wrapping (discussed earlier). This issue is discussed for example by Harsu (2003) and partly also by Systä (2001;2000). The related terminology is as follows:

- *Compilation* is used in transforming high-level programs into low-level ones.
- Transformation into opposite direction, from low-level to high-level, can be done with *decompilation*.
- *Conversion* can be used when source and target programs are at the same level.
- In *translation* any-level program is transformed into any-level program.
- In *source-to-source translation* transformation is done from high-level to high-level program.

Translation can be implemented with statement-by-statement approach, especially when source and target languages share the same programming paradigm and structures of languages are close to each other. Another approach is to translate via abstraction and reimplementation. It is useful, when target language is more structural than source language. Also reengineering (to be discussed later) may be reasonable to improve the quality, maintainability and reusability.

The source-to-source translator, tool to perform translation, has much in common with compiler. Both of them transform programs in another language, perform lexical analysis and parsing and may need preprocessor to transform the source code into more appropriate form for translator. Unlike compilers, converters usually expect that source program is error-free. Reason for this is that translation is usually performed in a major reengineering effort in organization and errors are already found in compilation and in long time use. Also code generation phase is different, since compilers and translators have different level target language.

One of the problems in language conversion is that all structures of the original language don't have direct counterparts in target language. In these cases, problem is handled by not providing any translation or by providing complicated translation. Editing structures before or after translation can mitigate the problem, but accurate documentation of deficiencies is needed.

Languages enabling conditional compilation may be a problem in language conversion. Translation of the directives for conditional compilation from source

language to target language raises problem in parsing, because source code is sequence of fragments instead of being continuous stream of tokens of legal program. The solution for this problem is multi-branch parsing.

Names of identifiers are tried to keep unchanged in language conversion. It is not always possible because reserved words differ from language to another. Program comments should be transferred from source to target code to make it easier to understand and maintain. Comments are usually stored to the nodes of the parse tree or syntax tree (representing the source program).

According to Harsu (2003), almost all of the source-to-source translators have deficiencies. When selecting or designing a language converter, requirements and goals of source-to-source translation should be prioritized: all of them are very difficult or impossible to achieve at same time. Main goal to converter may be that it creates syntactically and semantically correct programs. Readability and maintainability may include in subsidiary goals. One solution used in some converters to shortcomings is to provide rude translation automatically and finishing of the process is done under the control of user.

6.1.5 Data conversion

This topic area relates to data reengineering and *e.g.* the use of database gateways. These issues are discussed by Bianchi *et al.* (2003), Seacord *et al.* (2003), and Comella-Dorda *et al.* (2000). Data conversion is required, when a modern system has to access legacy database. Approaches for implementing data conversion are *e.g.* *database gateways* and *XML integration*. XML integration is discussed in more detail in one of the previous sub-sections. A database gateway translates between two or more data access protocols. The benefits of using a database gateway are improved connectivity, a possibility for remote access, and support for integration of legacy data with modern systems. Normally database gateways are used to translate vendor specific protocols into one of the standard protocols (*e.g.* ODBC, JDBC, ODMG). *Bridges* are gateways that translate one standard protocol into another.

6.1.6 User interface renovation

Modernizing the user interface (UI) is a common way to address usability issues in legacy systems. It can also be effective in increasing user satisfaction through polishing the appearance of the system. However, from maintainer's point of view the UI-renovated system is just as (or more) inflexible and difficult to maintain than it previously was. Applicability of this approach is clearly questionable when there's also other objectives than usability and user satisfaction issues alone.

User interface renovation can be implemented by *e.g.* CGI integration (described earlier) or user interface wrapping (Comella-Dorda *et al.*, 2000). Common technique to implement the latter is called screen scraping. It consists of wrapping the old (usually text-based) interface with new (graphical one). This technique can

be extended easily, enabling one new user interface to wrap several legacy systems. From the perspective of the legacy system itself the new (graphical) interface is indistinguishable from the function end user (entering text on a screen).

6.2 General supporting approaches

6.2.1 Reengineering

Reengineering is part of preventive maintenance, enabling system operation or supporting flexible future changes, thus being most effective in long-term. According to Lehman's (1998) second law, the complexity of software systems increases as they evolve (unless additional effort, such as reengineering, is directed to reducing it). Our hypothesis (based on empirical observations), is that effort should be directed also to preventive maintenance (especially in case of long system lifetimes), since otherwise the system evolution may lead to a "dead end", while maintainability is reduced to a minimum, and unexpected external change pressures are likely to appear (sooner or later). Reengineering is discussed e.g. by Arnold (1993).

There exists the question of customer-need -based business process reengineering pressures versus technical-need- based reengineering pressures. Teng *et al.* (1998) believe that the profitability of reengineering is considerably better while there exists profound and drastic (business) needs of change. Our hypothesis is that long-term improvements in system quality should be justifiable in case that system lifetime is long. Since reengineering benefits appear in long-term, whereas acute pressures for corrective maintenance reflect directly from customer reactions, it may be difficult to justify shifting maintenance resources to reengineering. Thus, there is a need to justify it economically.

6.2.2 Refactoring and restructuring

For several years, expert-level object programmers have employed a growing collection of techniques to improve the structural integrity and performance of legacy software. The techniques referred to as refactoring have remained in the domain of experts because no attempt has previously been made to transcribe the lore into a form that all developer could use. These techniques are discussed especially by Fowler *et al.* (1999), whose approach focuses on improving the design of existing code at technical level by providing a collected list of more than 70 proven and potentially useful refactorings. It is stated that with a proper training a skilled system designer can take a bad design and rework it into well-designed, robust code. It is stated that refactoring steps are simple, and while these steps may seem elementary, the cumulative effect of such small changes can radically improve the design. Refactoring is a proven way to prevent software decay. The refactorings and related illustrative examples are written in Java, but the ideas are applicable to any object-oriented programming language. Griswold & Notkin (1993) have developed the theory of program restructurings.

Restructurings could be useful if they could be automated and if they would not introduce as side-effects deteriorated program efficiency or maintainability. Restructurings could also include *e.g.* identification and elimination of redundant code and identification and use of program invariants.

6.2.3 Reverse engineering

This area includes actual reverse engineering and such related techniques as impact analysis, program slicing, and program visualization. Reverse engineering means the identification of a (software) system's components and their interrelations and creation of representations of the system in another form or at a higher level of abstraction. Modernization may be supported by reverse engineering tools, which are compared *e.g.* by Bellay & Gall (1997). Tool support also affects maintenance costs. Our hypothesis is that program comprehension is an essential aspect in order to make changes safely (without further deteriorating system quality during the process). Thus the support tools should provide the information needed in typical cases (Koskinen *et al.*, 2003a). The area is discussed *e.g.* in Koskinen (2000) (reverse engineering techniques), and Arnold & Bohner (1996) (impact analysis).

Basically, all reverse engineering tools are based on automated analysis of source code, which in turn may be static (*i.e.* compile-time) or dynamic (*i.e.* run-time). The received information is typically stored into abstract syntax tree or similar data structure. The formed or identified structures include program chunks, clichés, call graphs, system dependency graphs, program dependency graphs, control dependency graphs, definition use –relations, cross-references, class hierarchies, message diagrams, module dependency graphs, block diagrams, and program slices. Also, software metrics may be calculated automatically related to the use of this kind of tools. Metrics include numeric data characterizing *e.g.* the different quality aspects of software (*e.g.* software complexity, data complexity, code change frequency, code age, reachability, coupling, and cohesion).

6.2.4 Designing for maintainability

The issue of designing for maintainability is discussed most notably by Smith (1999) in his book. This issue is different from other discussed ones, since it helps to overcome maintenance problems (only) in long-run. The area is a relatively understudied one. In principle, program code should be structured, modular and *e.g.* portability interfaces well-defined. Mixing business logic and technical implementation should be avoided. In case of legacy code these ideals often are not met. In principle it would also be desirable to extensively reuse the existing code. There exists also the well-known problems related to software reuse (abstractness of software, delocalized features, needs to adapt components *etc.*).

Smith's approach aims at improving a program's capacity for altering code to fit changing requirements and for detecting and correcting errors. His approach emphasizes program comprehension and the underlying psychology of

programming. According to him traditional approaches do not adequately address maintenance issues. The book lists principles, which can be applied to a variety of situations and lead to new insights into software maintenance. The developed themes are supported by applied and basic research. Maintenance issues are presented, discussed and addresses in a problem solving format. Stating maintenance goals and the criteria for meeting them create a context for investigating the process and uncovering the principles that govern it. Covered issues include: 1) definition of maintenance problems, 2) selecting an approach for solving the problems, 3) modelling the data interpretation, 4) awareness of the nature of basic human information processing, 5) naming, 6) abbreviations and mnemonics, 7) language and how it is used, 8) maintenance strategies, 9) programming errors, and 10) structures for effective maintenance.

7 Conclusions

This report has compared methods for software modernization estimation and support. The main results of the analysis of modernization estimation methods have been collected into the represented Tables 2a, and 2b. The approaches support the selection of software modernization strategies, approximation of the possible benefits, identification of potential risks, and determination of the related costs. The approaches were the following:

- WMU focuses on maintenance strategy selection based on the aspired level of customer satisfaction.
- SABA is a high-level framework for planning the evolution and migration of legacy systems taking into account both organizational and technical issues.
- Renaissance is a method for iteratively evaluating legacy systems, from various perspectives.
- VDM is a method and decision model for determining suitable component renewal processes.
- SRRT is formal model for determining optimal software rewrite and replacement policies.
- RPP is a process model for estimating costs and benefits of reengineering.
- RPFA is a check-list of potential problems related to reengineering projects, and of the corresponding appropriate technical and other means to react to the situation.
- RMM is a new, general software modernization management approach, taking risks (and both technological and business objectives) explicitly into account.
- COCOMO II is an established and relatively widely used general method for software effort and cost estimation, including many extensions for different kind of software and evaluation situations.
- FPA is an established and relatively widely used general method for software effort and cost estimation, having many variants for different kind of software.
- Softcalc is a model and tool for estimating costs of incoming maintenance requests, developed based on COCOMO and FPA.
- EMEE is a new approach for quick maintenance effort estimation before starting the actual maintenance.

SABA, Renaissance, and RMM best take into consideration both technical and organizational issues. SABA offers evaluation means for legacy migration and evolution. Renaissance supports iterative evaluation, which is positive, but real empirical validation is missing. RMM's cost evaluation part is very limited and based on earlier theory formation. Both RPFA and RMM focus on risks. The general scope and extent of RMM is much wider. WMU focuses on maintenance strategy selection, and deals adequately with that problem. SRRT best focuses on software replacement timing, but has many practical limitations and very tough requirements. VDM's peculiarity is component-level evaluation, which is a reasonable choice. COCOMO II and FPA are general methods for cost estimation. COCOMO II is well validated. Softcalc and EMEE build on that general estimation theory and focus on maintenance, but have little or no empirical validation.

The NIMSAD framework has served well by enhancing systematic discussion of the various methods and approaches available (and having diverse focus areas, and different goals). NIMSAD has been generalized by us to analyze general problem solving approaches. Software modernization decision making is part of general problem solving. This has required minor reinterpretations of some of NIMSAD's elements. The generalized framework has served adequately, although there are some earlier mentioned elements whose information contents easily overlap.

We have also summarized 10 approaches for improving the process of actual software modernizations. These approaches may be adopted while decision of the possible modernization (or maintenance strategy) has first been made. This part has not yet been a focus area (in this report).

The most promising options for further advances include the following:

- Iterative long-term improvement of expert decision making related to modernizations by systematically enhancing the awareness of important issues, risks, and possibilities, by collecting feedback, and possibly introducing an approach best suited to the particular organizational and technical characteristics of software suppliers.
- Identification of evolution patterns (and change history) based on large system portfolios.
- Gathering of (both quantitative and qualitative) data related to similar projects and reflecting on them.
- Stepwise long-term improvement of the existing maintenance processes.

References

- 1) Abran, A. & Robillard, P. (1993). "Reliability of function points productivity model for enhancement projects (a field study)". *Conference on Software Maintenance 1993*, 80-97. IEEE Computer Society Press.
- 2) Abran, A., Silva, I. & Primera, L. (2002). "Field studies using functional size measurement in building estimation models for software maintenance". *Journal of Software Maintenance and Evolution: Research and Practice* **14**, 31-64.
- 3) Ahn, Y., Suh, J., Kim, S. & Kim, H. (2003). "The software maintenance project effort estimation model based on function points". *Journal of Software Maintenance and Evolution: Research and Practice* **15** (2), 71-85.
- 4) Albrecht (1979). "*Measuring Application Development Productivity*". Proceedings of the IBM Applications Development Symposium, 83-92.
- 5) Albrecht, A. & Gaffney, J. (1983). "Software function, source lines of code, and development effort prediction: a software science validation". *IEEE Transactions on Software Engineering* **SE-9** (6), 639-648.
- 6) Arnold, R. (1993). "*Software Reengineering (IEEE Computer Society Press Tutorial)*". IEEE Computer Society, 675 p.
- 7) Arnold, R. & Bohner, S. (Eds.) (1996). "*Software Change Impact Analysis*". Wiley-IEEE Press, 392 p.
- 8) Avison, D. & Fitzgerald, G. (1995). "*Information Systems Development: Methodologies, Techniques and Tools*" (2nd ed.). McGraw-Hill International.
- 9) Banker, R., Datar, S., Kemerer, C. & Zweig, D. (1993). "Software complexity and maintenance costs". *Communications of the ACM* **36** (11), 81-94.
- 10) Bellay, B. & Gall, H. (1997). "A comparison of four reverse engineering tools". *Proceedings of the 4th Working Conference on Reverse Engineering*, 2-11. IEEE Computer Soc.
- 11) Bennett, K., Ramage, M. & Munro, M. (1999). "Decision model for legacy systems". *IEEE Proc.-Softw.* **146** (3), 153-159.
- 12) Bergey, J. (1998). "*System Evolution Checklists Based on an Enterprise Framework*". Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, White paper.
- 13) Bergey, J., Northrop, L. & Smith, D. (1997). "*Enterprise Framework for the Disciplined Evolution of Legacy Systems*". Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Report: CMU/SEI-97-TR-007.

- 14) Bergey, J., Smith, D., Tilley, S., Weiderman, N. & Woods, S. (1999). "Why reengineering projects fail". Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Report: CMU/SEI-99-TR-010.
- 15) Bianchi, A., Caivano, D., Marengo, V. & Visaggio, G. (2003). "Iterative reengineering of legacy systems". *IEEE Transactions on Software Engineering* **29** (3), 225-241.
- 16) Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R. & O'Sullivan, D. (1997). "An overview of legacy information system migration". *Asia Pacific Software Engineering Conference, 1997 (APSEC'97)*, 529-530.
- 17) Boehm, B. (1981). "Software Engineering Economics", Prentice Hall.
- 18) Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B., Steece, B., Brown, A., Chulani, S. & Abts, C. (2000). "Software Cost Estimation with COCOMO II". Prentice Hall, 502 p.
- 19) Brodie, M. & Stonebraker, M. (1995). "Migrating Legacy Systems: Gateways, Interfaces & The Incremental Approach". Morgan Kaufmann, 210 p.
- 20) Chan, T., Chung, S. & Ho, T. (1994). "Timing of software replacement". In: DeGross, J., Huff, S. & Munro, M. (Eds.) *Proceedings of the Fifteenth International Conference on Information Systems*, 291-307.
- 21) Chan, T., Chung, S. & Ho, T. (1996). "An economic model to estimate software rewriting and replacement times". *IEEE Transaction on Software Engineering* **22** (8), 580-598.
- 22) Chapin, N., Hale, J., Khan, K., Ramil, J. & Tan, W.-G. (2001). "Types of software evolution and software maintenance". *Journal of Software Maintenance and Evolution: Research and Practice* **13** (1), 3-30.
- 23) Comella-Dorda, S., Wallnau, K., Seacord, R. & Robert, J. (2000). "A survey of black-box modernization approaches for information systems". *Proceedings of the International Conference on Software Maintenance - 2000*, 173-183. IEEE Computer Soc.
- 24) De Lucia, A., Pannella, A., Pompella, E. & Stefanucci, S. (2001). "Assessing massive maintenance processes: an empirical study". *Proceedings of the IEEE International Conference on Software Maintenance*, 451-458. IEEE Computer Soc.
- 25) De Lucia, A., Di Penta, M., Stefanucci, S. & Venturi, G. (2002). "Early effort estimation of massive maintenance processes". *Proceedings of the International Conference on Software Maintenance - 2002*, 234-237. IEEE Computer Soc.
- 26) DeSanctis, G. & Gallupe, R. (1987). "A foundation for the study of group decision support systems". *Management Science* **33** (5), 589-609.
- 27) Dolado, J. (1997). "A study of the relationships among Albrecht and Mark II Function Points, Lines of Code 4GL and effort". *Journal Systems Software* **37**, 161-173.
- 28) Eastwood, A. (1993). "Firm fires shots at legacy systems". *Computing Canada* **19** (2), p. 17.
- 29) Erlikh, L. (2000). "Leveraging legacy system dollars for E-business". *(IEEE) IT Pro*, May/June 2000, 17-23.
- 30) Forsell, M., Halttunen, V. & Ahonen, J. (1999). "Evaluation of component-based software development methodologies". Penjam, J. (Ed.) *Proceedings*

- of the Fenno-Ugric Symposium on Software Technology (FUSST'99). Tallinn Technical Univ., Tallinn, Estonia.*
- 31) Foster, J. (1993). “*Cost Factors in Software Maintenance*”. Ph.D thesis. Computer Science Dept., Univ. of Durham, UK.
 - 32) Fowler, M., Beck, K., Brant, J., Opdyke, W. & Roberts, D. (1999). “*Refactoring: Improving the Design of Existing Code*”. Addison-Wesley, 431 p.
 - 33) Furey, S. (1997). “Why we should use function points”. *IEEE Software* **14** (2), 28-31.
 - 34) Gill, G. & Kemerer, C. (1991). “Cyclomatic complexity density and software maintenance productivity”. *IEEE Transactions on Software Engineering* **17** (12), 1284-1288.
 - 35) Gode, D., Barua, A. & Mukhopadhyay, T. (1990). “On the economics of the software replacement problem”. In: DeGross, J., Alavi, M. & Oppelland, H. (Eds.) *Proceedings of the Eleventh International Conference on Information Systems*, 159-170.
 - 36) Griswold, W. & Notkin, D. (1993). “Automated assistance for program restructuring”. *ACM Transactions on Software Engineering and Methodology* **2** (3), 228-269.
 - 37) Harsu, M. (2003). “*Ohjelmien ylläpito ja uudistaminen*” (in Finnish). Talentum, 292 p.
 - 38) Huff, S. (1990). “Information systems maintenance”. *The Business Quarterly* **55**, 30-32.
 - 39) Jayaratna, N. (1994). “*Understanding and Evaluating Methodologies: NIMSAD: A Systemic Framework*” (Information Systems, Management and Strategy Series). McGraw-Hill.
 - 40) Jørgensen (1995). “Experience with the accuracy of software maintenance task effort prediction models”. *IEEE Transactions on Software Engineering* **21** (8), 674-681.
 - 41) Kafura, D. & Reddy, G. (1987). “The use of software complexity metrics in software maintenance”. *IEEE Transactions on Software Engineering* **SE-13** (3), 335-343.
 - 42) Kemerer, C. & Slaughter, S. (1999). “An empirical approach to studying software evolution”. *IEEE Transactions on Software Engineering* **25** (4), 493-509.
 - 43) Kitchenham, B. (1997). “The problems with function points”. *IEEE Software* **14** (2), 28-31.
 - 44) Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K. & Rosenberg, J. (2002). “Preliminary guidelines for empirical research in software engineering”. *IEEE Transactions on Software Engineering* **28** (8), 721-734.
 - 45) Koskinen, J., Lahtonen, H. & Tilus, T. (2003a). “*Software Maintenance Cost Estimation and Modernization Support*”. University of Jyväskylä, Information Technology Research Institute, ELTIS-project, Technical report, 60 p.
 - 46) Koskinen, J., Salminen, A. & Paakki, J. (2003b). “Hypertext support for information needs of software maintainers”. *Journal of Software Maintenance and Evolution: Research and Practice*.

- 47) Lee, A., Cheng, C. & Balakrishnan, J. (1998). "Software development cost estimation: Integrating neural networks with cluster analysis". *Information & Management* **34**, 1-9.
- 48) Lehman, M., Perry, D. & Ramil, J. (1998). "Implications of evolution metrics on software maintenance". *Proceedings of the International Conference on Software Maintenance - 1998*, 208-217. IEEE Computer Soc.
- 49) Lientz, B.P. & Swanson, E. (1980). "*Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*". Addison-Wesley: Reading, MA, 214 p.
- 50) Locan, C. (2000). "An empirical analysis of function point adjustment factors". *Information and Software Technology* **42**, 649-660.
- 51) March, J. (1981). "Decision making perspective". In: *Perspectives on Organization Design and Behaviour*, 205-244. van de Ven, A. & Joyce, W. (eds.). John-Wiley.
- 52) Matjaz, B. et al. (2000). "*Integrating Legacy Systems in Distributed Object Architecture*". *ACM SIGSOFT Software Engineering Notes* **25** (2).
- 53) McKee, J. (1984). "Maintenance as a function of design". *Proceedings of the AFIPS National Computer Conference*, 187-193.
- 54) Moad, J. (1990). "Maintaining the competitive edge". *Datamation* 61-62, 64, 66.
- 55) Müller, H., Wong, K. & Tilley, S. (1994). "Understanding software systems using reverse engineering technology". *The 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences Proceedings (ACFAS)*, **26** (4), 41-48.
- 56) Port, O. (1988). "The software trap – automate or else". *Business Week* **3051** (9), 142-154.
- 57) Ransom, J., Sommerville, I. & Warren, I. (1998). "A method for assessing legacy systems for evolution". *Software Maintenance and Reengineering, Proceedings of the Second Euromicro Conference*, 128-134.
- 58) Robertson, P. (1997). "Integrating legacy systems with modern corporate applications". *Communications of the ACM* **40** (5), 39-46.
- 59) Sahin, I. & Zahedi, F. (2000). "Optimal policies under risk for changing software systems based on customer satisfaction". *European Journal of Operational Research* **123** (1), 175-194.
- 60) Sahin, I. & Zahedi, F. (2001a). "Control limit policies for warranty, maintenance, and upgrade of software systems". *IIE Transactions* **33** (9), 729-745.
- 61) Sahin, I. & Zahedi, M. (2001b). "Policy analysis for warranty, maintenance, and upgrade of software systems". *Journal of Software Maintenance: Research and Practice* **13**, 469-493.
- 62) Seacord, R., Plakosh, D. & Lewis, G. (2003). "*Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*" (SEI Series in Software Engineering). Addison-Wesley.
- 63) Simon, H. (1983). "Models of bounded rationality". *Behavioral Economics and Business Organization* (vol 1-2). The MIT Press.
- 64) Smith, D.D. (1999). "*Designing Maintainable Software*". Springer Verlag, 169 p.

- 65) Sneed, H. (1995a). "Estimating the costs of software maintenance tasks". *Proceedings of the International Conference on Software Maintenance - 1995*, 168-181. IEEE Computer Soc. Press.
- 66) Sneed, H. (1995b). "Planning the reengineering of legacy systems". *IEEE Software* **12** (1), 24-34.
- 67) Sneed, H. (1999). "Risks involved in reengineering projects". *Proceedings of the IEEE Sixth Working Conference on Reverse Engineering*, 204-211. IEEE Computer Soc.
- 68) Sneed, H. (2000). "Encapsulation of legacy software: a technique for reusing legacy software components". *Annals of Software Engineering* **9**, 293-313.
- 69) Sommerville, I. (1995). "*Software Engineering*" (5th ed.). Addison-Wesley.
- 70) Sommerville, I. (2000). "*Software Engineering*" (6th ed.). Addison-Wesley.
- 71) Swanson, E. & Dans, E. (2000). "System life expectancy and the maintenance effort: exploring their equilibration". *MIS Quarterly* **24** (2), 277-297.
- 72) Systä, T. (2000). "*Static and Dynamic Reverse Engineering Techniques for Java Software Systems*" (Ph.D. thesis). Department of Computer Science and Information Sciences, University of Tampere, 233 p.
- 73) Systä, T., Koskimies, K. & Müller, H. (2001). "Shimba - an environment for reverse engineering Java software systems". *Software - Practice and Experience* **31**, 371-394.
- 74) Teng, J., Jeong, S. & Grover, V. (1998). "Profiling successful reengineering projects". *Communications of the ACM* **41** (6), 96-102.
- 75) Verhoef, C. (2002). "Quantitative IT portfolio management". *Science of Computer Programming* **45**, 1-96.
- 76) Visaggio (2000). "Value-based decision model for renewal processes in software maintenance". *Annals of Software Engineering* **9**, 215-233.
- 77) Wallnau, K., Hissam, S. & Seacord, R. (2001). "*Building Systems from Commercial Components*". Addison-Wesley.
- 78) Warren, I. & Ransom, J. (2002). "Renaissance: a method to support software system evolution". *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02)*, 415-420. IEEE Computer Soc.
- 79) Whitmire (1995) "*Introduction to 3D Function Points*", Software Development, 43-53.
- 80) Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D. & Richardson, R. (1997). "*Legacy System Migration: A Legacy Data Migration Engine*". *Proceedings of the 17th International Database Conference (DATASEM'97)*, 129-138.
- 81) Yau, C. & Tsoi, H. (1998). "Modelling the probabilistic behaviour of function point analysis". *Information and Software Technology* **40**, 59-68.

Appendix 1: Summarizing table

This table (see next page) contains the differences between the reviewed methods for estimating modernization efforts presented in section 5.5, in Tables 2a, and 2b as single table.

Generalized NIMSAD elements	WMU	SABA	Renaissance	VDM	SRRT	RPP	RPFA	RMM	COCOMO II	FPA	Softcalc	EMEE
M Context												
Use situation	Warranty, maintenance, upgrade decision making	Legacy transition strategies, organizational & technical analysis	Basis for reengineering & incremental improvement, integrates with project management	Component-level techn. & econ. analysis	Timing of replacement & rewriting	Cost & benefit analysis of evolution strategies	Reengineering risk analysis	Risk-managed & incremental modernization strategy, sw. architectures covered	Cost estimation for large-scale projects	Cost estimation	Maintenance request effort estimation	Quick & early effort estimation of maintenance tasks
Start for M use	Strategically optimal customer satisfaction?	Evolution & transition strategies?	Evolution of degrading software?	Rejuvenation strategies of aged software?	Profitability of rewrite?	Profitability of reengineering?	Risks of reengineering?	Strategy to reuse the old and adapt to changing technologies?	Project budget, schedule, and maintenance plan?	Software development effort?	Effort of the current maintenance task?	Early estimate for maintenance task effort?
Customers etc.	Sw. change managers	Sw. engineers	Sw. change managers	Sw. change managers	IS managers	Sw. maintainers	Technical & reeng. & project managers	Technical & sw. change managers	Project managers	Project managers	Sw. maintainers	Sw. maintainers
Context descr.	Sw. change control, product monitoring	Includes org. business strategy modeling	Org. context: processes & tools	Sw. quality control, renewal project	Volatile user environment, deteriorated legacy sw.	Deteriorated legacy sw., unavailability of proper tools	Enterprise-wide context	Organization	Rapid development & tailoring of sw. products	Use situation	Use situation	Use situation
Culture etc.	Decision making support, change monitoring	Business strategy driven prioritization	Long term goal setting	Specialization to the organization and project	Availability of the input data required, sw. quality control	Sw. measurement program policy required	Improvement of risk awareness	Disciplined risk management, creative problem solving	Long-term process improvement, system metrics & statistics	Disciplined work culture, metrics collection policy	Metrics collection policy	Understanding of statistical tools, metrics collection policy
Risks (context)	Stable upgrade lifecycle assumed	Sub-systems may differ	N/A	N/A	Incompleteness of the decision criteria	N/A	Proper process improvement attitude is required	Complex, multi-faceted problem & varying effort magnitude	Calibration is required	N/A	N/A	N/A
Risks (M) (see also Evaluation)	Model details presented elsewhere	Information gathering may extend several months	High adoption overhead, well-defined and easy to follow, helps in risk reduction	Inaccurate assessment of component's business value and cost	Availability of reliable input data	Reliance on the quality of metrics program & analysts' expertise	Represented examples may be combinations or may have been altered	Risk assessment & cost estimation parts are limited, based on FPA & COCOMO	Sufficient level of expert judgment is critical	Discipline and expertise assumed	Versatile and reliable input required, laborious impact domain analysis, based on FPA & COCOMO	Limited scope & limited prior use
M User												
Motives/values	Long-term profit maximization, optimiz. of customer satisfaction	Both technical and other considerations should be covered	Proneness to change & contin. improvement, preventing new legacy problems	Rationality	Long-term cost estimation, sw. quality control	Argumentation, planning	General, organization-wide risk analysis and process improvement	Rationality, creative problem solving, risk management	Rationality, long-term planning, & project estimation	Rationality, long-term planning & project estimation	Like in case of FPA, and COCOMO	Like in case of FPA, and COCOMO
Abstract reason.	Limited	Wide, including judging options	Wide	Estimation of sw. components	Limited, the model's use is simple	Limited, the process model is simple	Very limited	Complex	Comparison of projects and judging complexity factors	Limited	Limited	Very limited
Needed skills	Simple strategy selection & possibly prior elaborate metrics program developments	Economic & technical expertise & planning of information gathering	Expertise on application domain & legacy system's functional & technical issues	Components' quality & business value quantifications	Limited, but possibly requires metrics gathering process development	Knowledge on organizational & project goals & software metrics & business value estimation	General organizational & situational awareness	Wide knowledge of the organization	Model use is simple, but expertise on project cost estimation and the project's characteristics is required	Disciplined approach to the gathering of the required input based on somewhat vague definitions	Experience on software maintenance, impact domain determination, general cost estimation knowledge	Knowledge on statistics
M												
Problem situat.	About 10 suggestions of the nature of the problem	Some suggestions	4 suggested requirements	Other sources are referred for this	About 10 characterizations of the problem	Functional and technical reengineering to be separated	The checklist characterizes the situation	Example case study is described	Various suggestions	N/A	N/A	N/A
Diagnosis	8 factors (about 20 subfactors)	OST (4 main scenarios and 9 criterias)	4 categories of factors (about 30 subfactors)	Quality score (4 subcategories) & economic score (10 subcateg.)	4 functions and 13 parameters	5 factors	10 factors	Problems, risks, strategies	17 effort multipliers	5-7 factors	About 20 factors	At least 3 factors
Prognoses	Recommendations of maintenance actions (& expected ROI)	6 listed migration strategies, TST: prioritized set of solutions	Goal is sw. which can be continuously evolved	Classification of components, and their optimal renewal process	Time for starting rewrite and replace, size of the rewrite team	N/A	Lists of relevant activities, practices, techn. solutions and tools	Lists of current standards and available products	Required effort, costs, calendar time	Required effort	Required effort	Required effort
Problem def.	Fixed, characterized by parameter values	Fixed	Fixed, general	Fixed	3 specific subproblems, characterized by parameter values	Fixed, general argumentation	Fixed, 10 subproblems	Guided at abstract level	3 main subproblems, modes of "granularity", many extensions	N/A	N/A	N/A
Notional syst.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Design	N/A	Multi-phased (OST, TST) iterative process	Method calibr., system assessment, evolution strat. devel., evolution project planning	N/A	N/A	N/A	N/A	N/A	Selection of model variants	N/A	N/A	N/A
Implementation	Simple calculations	OST- and TST tools	3 phases (and 8 subphases)	4 phases	Application of the represented formulas	Proj. justification, portfolio anal., cost estimation, cost-benefit analysis, contracting	Check-out of the represented risk-list	9 phases	Application of the represented formulas, expert judgments	4 phases	7 phases	Application of the represented formulas
Evaluation												
Internal eval.	N/A	Method has been developed gradually, iteration may be used to explore the consequences	Feedback after each increment can be used, the model is customized for the project	Longitudinal data gathering during method development	N/A	N/A	N/A	Incremental modernization makes it possible in principle	The method has long development history, calibration is required	N/A	Calibration is emphasized, but reflective evaluation is not explicated	Calibration
External eval.	Mathematical analysis of function forms (3840 base line scenarios)	Synthesis of earlier works, OST: empirically tested, TST: based on experience, model: to be tested	Feedback from industrial partners	Observations of a real-world renewal project (653 programs)	Based on analytical and simulation solutions, model parameters and functions are based on various earlier empirical field data studies	Based on the author's 25 years of sw. engineering experience	Variety of experiences over many years	Illustrated with a "real-world" case study	Based on 161 industrial projects	Strong set of empirical data, various studies	None	Empirical validation, with a 40 KLOC project, minimum prediction error was 35%