Method for Robustness Analysis and Technology Forecasting of Software Based Systems

– A method proposal

Göran Calås
Summary

When designing software products with a long-term perspective, the software system will have to be modified for each product release.

Software Engineering has traditionally dealt with the problem of designing software that is economical to maintain, and easy to re-use; by using a number of strategies and method principles. Some of these principles are applied in the Robustness Analysis method once introduced as a part of Objective Systems AB’s software engineering method Objectory and the CASE tool OrySE. Rational’s Unified Process model RUP, has inherited Objectory’s robustness analysis method, which is now incorporated in the method activity called Object Analysis.

The guiding principle in Objectory’s and the RUP’s Robustness Analysis method description is to localize future system changes to well defined places in the software. This principle is known as: “Separation of Concerns”. In RUP and Objectory, this means that an architecture concept is applied, where analysed Use-Cases are assumed to be implemented using three different objects and class stereotypes. The stereotypes are called: Boundary Classes; Control Classes, and Entity Classes. In the more traditional domain object analysis, the modelled world is limited to consist of only one object stereotype, the Domain Objects. A Domain Object Analysis model, has not as rich description language as a Robustness Analysis model.

A driving force for investing the extra effort required by a Robustness Analysis is that the value of the analysis model increases. It becomes a better complete description of the system being developed. The extra Robustness Analysis effort is expected to give a return in a long-term perspective, when future system releases are made due to new requirements, or software maintenance.

The hypothesis put forward in this dissertation is that it is possible to do better software analysis, by first study how the software product, or software system, is expected to grow, evolve and develop in the future. Such a futuristic study is called Technology Forecasting, a term taken from a non-software oriented method TRIZ (Theory for Inventive Problem Solving). In Technology Forecasting, different techniques are applied to build a vision of potential evolution trends for the software investigated. When a future evolution trend and sketch for the software has been identified, this information serves as an input to the Robustness Analysis method. Robustness Analysis can then be extended to cover potential future requirements and technology paradigm shifts.

The concept of combining robustness analysis with technology forecasting, is expected to result in an improved software design, by making it possible to “design software products of tomorrow, already today”.
In this dissertation, some popular system and software engineering methods are analysed, concerning their abilities to contribute to and to perform Robustness Analysis and Technology Forecasting.

Due to short-comings in investigated popular methods, a new method: Robustness Analysis with Technology Forecasting (RATF) has been proposed and briefly investigated.

To verify the proposed RATF method, it has been analysed concerning its theoretical benefits. To further evaluate the RATF method performance a case study was with the purpose to evaluate practical usage of the RATF method, and its performance. The case study tested RATF against the investigated popular methods. Unfortunate this case study could not be completed before for the project deadline. Hence the case study had to be omitted from this dissertation.

The dissertation results are four research papers:

- A research survey
- A method description of proposed RATF method
- An analysis of proposed RATF method
- A definition of software evolution laws

Comment due to publishing decision (added 2004-08-23)

The four papers described above were originally included in this dissertation. Upon recommendation from my examiner, we decided to remove all of the above research papers, but the research survey. The omitted research papers will now be sent to international software research conferences for publication during year 2005. This publishing strategy is expected to give better impact for the research results achieved.

This means that only the research survey paper is included in this dissertation.
Preface

Methods for Robustness Analysis of software based systems, aims to analyse and correct potential weaknesses of a future software design. The proposed method for “Technology Forecasting”, aims to predict a software system’s future evolution path, thus guiding software architects and engineer in further product management.

Robustness analysis, attempts to answer the questions:

• “How well does this software system adjust to potential future requirement changes?”

• “How does the system need to change to fulfil potential future requirements?”

Technology forecasting, tries to answer the questions:

• “Assume an existing system: How would such a system typically evolve, to become a potentially better system?” This implies that the system may evolve in multiple directions or dimensions, simultaneously.

• Assuming evolution over a very long time period: “How would an ideal software system look like?”

• Assuming an ideal system: “What evolution paths and requirement forces are most likely, to guide the current system to reach the ideal system characteristics?”

Robustness analysis and technology forecasting, are expected to complement each other, by providing two approaches for handling uncertainties in potential future requirements on software design and development. Designing for reuse is complicated, if one does not know the future requirements. As there is a trade-off between cost for extensive analysis and design of a reusable system, and the cost of re-work due to requirement changes; one have to be pragmatically, and find the right ambition level.

This paper aims to describe a new method, using a combination of “investigation of deficiencies in an analysis model”, and “prediction and invention of future technological evolution paths of a software system”. This method is expected to give advantages in prediction and comprehension of software technology evolution, by enabling design of more “future proof” systems.
## CONTENTS

Summary ............................................................................................................................ 2
Preface ............................................................................................................................... 4
1 Introduction ................................................................................................................. 6  
  1.1 Document overview ............................................................................................. 6  
  1.2 Guide to the reader ............................................................................................... 7  
  1.3 Background .......................................................................................................... 8  
2 Theory ......................................................................................................................... 8  
  2.1 Definitions ............................................................................................................. 9  
  2.2 The hypotheses ..................................................................................................... 9  
3 Problem description ..................................................................................................... 9  
4 Project goals ................................................................................................................ 9  
5 Methods ...................................................................................................................... 10  
6 Theory ....................................................................................................................... 10  
7 Models ....................................................................................................................... 11  
8 Simulations ................................................................................................................. 11  
9 Measurements ............................................................................................................. 11  
10 Results ..................................................................................................................... 11  
11 Conclusions .............................................................................................................. 12  
12 Acknowledgements .................................................................................................. 12  
13 References ............................................................................................................... 13  
14 Appendix - Attached research paper ...................................................................... 15
1 Introduction

1.1 Document overview

This dissertation on the Robustness Analysis and Technology Forecasting (RATF) method consists of the following papers.

[1] Robustness Analysis and Technology Forecasting
   – A research survey of popular methods
   This is a research survey on application of Robustness Analysis and Technology Forecasting methods in software engineering. It investigates some popular non software oriented methods and software engineering methods, and their application in software engineering.

[2] Robustness Analysis and Technology Forecasting
   – Analysis of the RATF method
   This is a proposal of a new software engineering method the RATF method. The RATF method itself is documented in a separate method description, see step 3 below. The method combines Robustness Analysis with Technology Forecasting. It is a synthesis of promising methods and techniques as indicated by the research survey, in step 1 above. The RATF method is evaluated in this paper.

[3] Robustness Analysis and Technology Forecasting
   – Description of the RATF method
   This is a description of the new software engineering method. The purpose of the document is to work as a practical description for software engineers.

[4] Robustness Analysis and Technology Forecasting
   – Method evaluation as a case study
   This paper presents practical experiences from usage of the new RATF method.
   As the project had to finish before its deadline 2004-08-02, the case study has been omitted from this dissertation.

   – A method proposal
   That is this dissertation file, which is a collection document for the research papers written. It also report results from the dissertation project.

   This was the first paper written on the subject, by the author. It translates System Evolution Laws, from the TRIZ method, to suit the software engineering domain.

© G. Calás, 2004. All rights reserved.
1.2 Guide to the reader

Chapter 1.1 specifies papers included in this dissertation.

The goal of this dissertation was to: perform research in software engineering methods; and produce a number of publishable research papers.

The produced papers are to follow formatting guidelines for research articles to be submitted to IEEE software engineering conferences. Since different conferences have different requirements on format, the final formatting has not been done. Following formatting work remains for all papers produced: perform peer review; adjust page size and column placement; alter headings formats and numbering; and proof read all papers. It might be surprising that peer review and proof reading remains, but these activities requires help from other people than the author. This dissertation was to be produced by the author alone.

Research papers produced has the following internal dependencies, marked as dashed arrows. The Analysis of the RATF method is built on top of its description and the research survey. Suggested reading sequence is: [5] → [1] → [2] → [3] → [6].

![Figure 1: Dependencies between dissertation papers](image)

© G. Calás, 2004. All rights reserved.
1.3 Background

I became interested in the technology forecasting subject in 1998 when I was preparing a lecture on innovation technology and patents for Ericsson Telecom AB. I had earlier worked as a software engineer, systems engineer, project manager, product manager, systems architect, and strategic system manager within the company; when I came in touch with the non-software oriented innovation method, called TRIZ (Theory for Inventive Problem Solving). I recognized the potential immediately but it was not adapted for software problems. As analogies between technical engineering domains and software engineering are appealing to the mind, it almost became an obsession try to find out IF; and if possible, HOW; software engineering and software architecture design may benefit from methods like TRIZ. If the TRIZ method should be usable, it needed to: be adapted from to world of physics to the world of software engineering; be practical; applicable; feasible; and economical to use.

Often the most genius solution requires radically less implementation effort than the first chosen design solution. More genius solutions are also often smaller, faster and require less resource footprint than past generation solutions.

Then, what prevents us from producing those genius solutions from the beginning? Do we really have to redesign software, rewrite, and modify software to make it evolve through system generations? Or is it possible to save the money and skip to the next generation software system at once?

This was the clue of my detective story; which at least is partially solved in this dissertation project.

2 Theory

This dissertation studied theoretical effects of combining robustness analysis with technology forecasting, with the purpose to reach a better software development result.

Software technology forecasting explores knowledge about future software systems, and future software technology generations. The dissertation concludes that it is possible, and a common practice to predict future evolution in software engineering, see [2]. One example is design for re-use.

- It is easy to predict the future!

- The difficult part is to figure out all possible evolution paths for the future.
  And, then there is that tricky part of figuring out which of the evolution paths that are most likely in our future.

- And then, since forecasting can be quite time consuming, there is always a risk; that the future has already sneaked by; while we are busy figuring it out.

Theories are suggested in the research papers [1][2][3][6] that are included in this dissertation.
2.1 Definitions
Terminology and definitions are presented in each paper.

2.2 The hypotheses
The following hypotheses are expected to provide a better method on software robustness analysis and software technology forecasting.

i) **Robustness Analysis** is the process of analysing an existing system in the view of future potential environmental change. The purpose of the analysis is to be able to project how the system needs to change, to adapt to future system needs.

ii) **Software technology forecasting** is the process of predicting how a software system may evolve, assuming it evolves to increased ideality in any direction.

iii) By combining robustness analysis with technology forecasting, software engineering results improve

The research question for this study has been:
“Is it possible to apply Technology Forecasting and Robustness Analysis, as integrated process in Software Engineering?”

And if so,
“How can we apply these processes?”

And
“Can the steps be documented as method?”

And
“How well does such a method perform, compared to traditional methods?”

3 Problem description
Traditional software development methods does not provide means for predicting how software is expected to change in the future, and how to improve software design to incorporate tomorrow’s design solutions already today.

4 Project goals
The project goals where:
1. Write scientific article with a survey on the state of art in mentioned domain.
2. Perform own research investigation and evaluate hypothesis.
3. Write scientific article on investigation results.
4. If identified method is proven useful, document method guidelines as a booklet for software engineers.

1 This last step was not a part of the research question stated in the project description.
5 Methods
In completing this research study, the following books on research and scientific writing have been of great help [8][9][10][11].

6 Theory
The theories suggested and evaluated by this dissertation are found and explained in four separate papers [1][2][3][6].

If the theory behind the RATF method is proven to be right, and The RATF Method is proven successful, it might challenge the common perception of software engineering as described in Kruchten’s book [7] on RUP, where he states the following.

<table>
<thead>
<tr>
<th>Wrong Assumption 2: We Can Get the Design Right on Paper Before Proceeding</th>
</tr>
</thead>
<tbody>
<tr>
<td>The second step of the sequential process assumes that we can confirm that our</td>
</tr>
<tr>
<td>design is the right solution to the problem. By “right” we imply all the obvious</td>
</tr>
<tr>
<td>qualities: correctness, efficiency, feasibility, and so on.</td>
</tr>
<tr>
<td>With complete requirements tracing, formal derivation methods, automated proof,</td>
</tr>
<tr>
<td>generator techniques, and design simulation, some of these qualities can be achieved.</td>
</tr>
<tr>
<td>However, few of these techniques are readily available to practitioners, and many of</td>
</tr>
<tr>
<td>them require that you begin with a formal definition of the problem.</td>
</tr>
<tr>
<td>You can accumulate pages and pages of design documentation and hundreds of</td>
</tr>
<tr>
<td>blueprints and spend weeks in reviews, only to discover, late in the process, that the</td>
</tr>
<tr>
<td>design has major flaws that cause serious breakdowns.</td>
</tr>
</tbody>
</table>

Software engineering has not reached the level of other engineering disciplines (and perhaps never will) because the underlying “theories” are weak and poorly understood, and the heuristics are crude.

At times it more closely resembles a branch of psychology, philosophy, or art than engineering. Relatively straightforward laws of physics underlie the design of a bridge, but there is no strict equivalent in software design. Software is “soft” in this respect.

Reprint from [7], page 55.

The marked paragraph indicates that software engineering has a long way to go to become mature. But by incorporating innovative modelling and a knowledge base on engineering parameters for software, and technical effects for system conflict resolution, it might be possible to model software to become more innovative and future proof.

The third paragraph from the top, mentions the need of a problem definition, a definition phase included in the RATF method [3].
The last paragraph refers to the use of laws of physics. In both the method description [3], the research survey [1] and the earlier evaluation of software evolution laws [6], new laws for software evolution and principles for solving innovation problems, are explained.

7 Models
No models have been used.

8 Simulations
A case study to compare results of applying the RATF method for a few software engineering problems, with results from application of studied software methods (as covered in the research survey); could not be completed in time for this dissertation. It is recommended to perform such a case study before the RATF method is made commercially available. The case study was not mandatory in the project plan.

9 Measurements
The research survey [1] contains a comparison of software engineering methods, and their abilities to support robustness analysis, and technology forecasting.

10 Results
The project goals with project results are listed below.

1. Write scientific article with a survey on the state of art in mentioned domain.
   - A research survey [1] is produced.

2. Perform own research investigation and evaluate hypothesis.
   - The research survey [1] included an investigation on Robustness Analysis and Technology Forecasting in software and system engineering methods.
   - The method analysis [2] evaluates The RATF Method, proposed, from a theoretical point of view.
   - A case study [4] has been partially performed, but it was not finished in time for the project deadline. The case study was not a part of the project description [12]. Hence it is omitted. The purpose of the case study was to evaluate The RATF Method proposed, on an experimental basis.

3. Write scientific article on investigation results.
   - The method analysis [2] has written as a scientific article.
   - To be able to publish The RATF method as described in [3], it has been written as a scientific article.

4. If identified method is proven useful, document method guidelines as a booklet for software engineers.
As the case study [4] was not completed, The RATF Method has not been completely verified, as case study experiments are missing. The purpose of the experiments where to compare development results, using traditional software engineering methods, and The RATF method, proposed by this dissertation.

A method handbook should be completed when case studies, of the RATF method, have been completed. After the case studies, the RATF method should be corrected, for any deficiencies found. An interesting alternative to a method handbook on paper, is to develop a method plug-in to the Rational Unified Process (RUP) documentation in HTML format.

11 Conclusions

The new method for Robustness Analysis with Technology Forecasting, proposed in this dissertation has clear advantages when designing innovative large scale software systems, with long-term product plans. Example applications are architectures, highly innovative software solutions, product families, and software product-lines.

Future work indicated:

- Is to complete a case study to evaluate The RATF Method on an experimental basis and to get benchmark figures that compares the performance with other methods.
- Refine and document the RATF method as handbook on paper, and as an online extension to the RUP web page based documentation.
- Improve technology forecasting modelling support suing UML diagram notation.

12 Acknowledgements

I would like to thank: my fiancé Desirée, my mother Anita, and my son David; for your support and motivation.
13 References

The documents [1], [2], [3], [5], [6], and [12] are included in this dissertation.

  *Robustness Analysis and Technology Forecasting*  
  *— A research survey of popular methods*,  
  University of Trollhättan / Uddevalla, Department of Technology, Sweden.  
  **This paper is attached as appendix A.**

  *Robustness Analysis and Technology Forecasting*  
  *— Analysis of the RATF method*,  
  University of Trollhättan / Uddevalla, Department of Technology, Sweden.  
  **Due to a publishing decision it was removed from this dissertation.**

  *Robustness Analysis and Technology Forecasting*  
  *— Description of the RATF method*,  
  University of Trollhättan / Uddevalla, Department of Technology, Sweden.  
  **Due to a publishing decision it was removed from this dissertation.**

  *Robustness Analysis and Technology Forecasting (RATF)*  
  *— Method evaluation as a case study*  
  University of Trollhättan / Uddevalla, Department of Technology, Sweden.  
  **The above paper could not be completed on time for the hand in of the dissertation, 2004-08-02. So, it has been omitted from this dissertation.**

[5] **That is this dissertation file!**

  *Software System Evolution Theories.*  
  Högskolan i Trollhättan och Uddevalla, Trollhättan, Sweden  
  **Due to a publishing decision it was removed from this dissertation.**

  *The Rational Unified Process; An Introduction,*  
  Addison-Wesley, USA  
  ISBN 0201604590

  *What is this thing called Science? Third edition,*  
  Open University Press, Buckingham, UK  
  ISBN 0-335-20109-1
_Scientists must write;_
_A guide to better writing for scientists, engineers and students_,
Routledge, London, UK,

_Academic Writing;_
_A university writing course_,
Studentlitteratur, Lund, Sweden.
ISBN 91-44-00409-5

_Rapporter och uppsatser_,
Studentlitteratur, Lund, Sweden,
ISBN 91-44-00417-6

_Project description:_
_Method for robustness analysis and technology forecasting of software based systems_,
University of Trollhättan / Uddevalla, Department of Technology, Sweden.
14 Appendix A - Attached research paper

The following pages contain the research paper, and should be treated as a separate document, with its own references. The pages are identical with [1].
Robustness Analysis and Technology Forecasting in Software Engineering – A research survey of popular methods

Göran Calås
Ulfstorps gård 2; SE-555 94; Jönköping; SWEDEN
goran.calas@home.se
Phone: +46-36-162909

Abstract

The talent to know how to design tomorrow’s software products already today, gives a tremendous competitive edge.

To design future proof and robust software systems require extensive knowledge about future: evolution of system environment; requirements; and technology.

In this paper, some popular software engineering methods are introduced and evaluated concerning their abilities to guide Robustness Analysis and Technology Forecasting in software engineering.

By combining the methods, it is assumed that the software engineer will get a better method efficiency to build change robust software, as estimated future system characteristics are considered. The Technology Forecasting method feeds the Robustness Analysis method with information about future system demands, thus improving overall analysis results.

This evaluation of the popular methods, indicate a gap in providing guidelines on how to:
• Predict wanted and beneficial software evolution.
• Integrate knowledge and prediction about future software systems’ evolution during software analysis, preferably robustness analysis.

The focus for the study has been on technical aspects of software evolutions, rather than psychological, intellectual, social and cultural factors.

1 Introduction

1.1 Background

The purpose of Robustness Analysis and Analysis-Object Modelling is to: develop a model that leaves the system tolerant to changes; and to gain further understanding of system demands.

Design of re-usable and robust software systems require knowledge about potential evolution of system environment and requirements. This kind of knowledge is often recognized as experience, and it is rarely expressed formally. The experiences often represent tacit knowledge in software engineering that cannot be expressed. It is kind of knowledge that software engineers learn through their professional practice.

Such knowledge is vital for designing software products that need to be well equipped to withstand future evolution of system requirements. Reasons for evolution of requirements are changes in: system environment; system usage; internal construction technology used; and software components, external and internal.

Prediction knowledge about future software evolution and requirements is especially applied during a certain software analysis phase, denoted Robustness Analysis. When new software is introduced, it starts to influence its surrounding environment. Finally this results in new requirements on the software itself. Therefore it is normally an iterative task to build high quality reusable software solutions and components.

1.2 A problem in need of a solution

Assume that we knew how a software product will be developed and evolve in the future. If we just knew every evolution step from an immature software system, to the final ideal software solution. Then it would have been much easier to develop the proper software solution from the beginning.

Technology Forecasting can help us predicts a system’s vision and provide knowledge about future:
• System evolution
• Requirements
• Product features
The system vision may then serve as an important input to the Robustness Analysis. During Robustness Analysis, software is analysed, and re-modelled to make it more robust towards future requirements changes and system evolution. The Robustness of a software system indicates how quick and easy it is to adapt it to fulfil new requirements, either automatically or through a limited amount of manual operation.

1.3 Methods

Methods in software engineering vary from being extremely formal to non-formal. More formal methods for software analysis, as described by the early Objectory process, and later in Rational’s Unified Process (RUP) model include an analysis step called Robustness Analysis. This step is now contained in RUP’s Analysis step, without calling it Robustness Analysis. The purpose of Robustness Analysis is to achieve an improved analysis that takes in account likely future changes in system environment, and re-models the system to better cope with these changes. The technique to solve the problem is usually to localize future software changes to a limited space of the software. This follows the software engineering principle behind “separation of concerns”.

The contribution of adding extra work effort on doing robustness analysis is expected to result in better system properties and long term value, by improving system features and reducing maintenance effort.

Less formal approaches to software development makes use of other ways to handle system changes. Extreme Programming, for example, focus on reducing the effort introducing software changes by making every change as cheap as possible, by reduction of formalism. One could say that changes happen when requested.

Studies of software evolution, from a maintenance perspective, reveals how software systems decay during their life cycles. But these studies do not give a proactive recommendation for how to design more robust software. Nor, do these studies show how to develop software from one generation to another.

1.4 Previous research

Currently there exist a number of methods and techniques for developing software. Most methods deal with software development as a single event, while other like RUP and the earlier Objectory method describe how to extend an existing system.

When developing software and maintaining software through its life cycles, and through software system generation, one have to deal with other problems than when specifying and designing a single software program.

Svetinovic and Godfrey [21], gives a good overview of problems related to software technology forecasting, taking into considerations non-technical aspects that influence software evolution such as: psychological; intellectual; social and cultural factors. They also contribute with insights on evolution towards both increasing, and decreasing variations of functional and quality based attributes among a population of software that is a product-line or product family.


Study of software design and architectural patterns, enables the software engineer to design better software, by relying on past successful design experiences. One can see design patterns as a kind of recipe book for software engineers. Design patterns facilitate the design of flexible and efficient systems, by re-using design experience thus taking into consideration future likely system requirement changes, e.g. [3].

Two books have been written on Robustness Analysis related to Use-Case modelling, the RUP method [8] and the Objectory method [10]. The current RUP on-line documentation has captured most of the past method descriptions to guide Rational Rose and Rational XDE software tool users in software analysis.

In RUP’s Business Analysis method, the artefact Vision and Business Vision captures essential aspects for the evolution of a software product-line, which plays a similar role as technology forecasting.

The innovation problem solution technique TRIZ, and its methods, e.g. Algorithm for Inventive Problem Solving (ARIZ) a technique for conflict analysis, intensification and resolution synthesis, supports technology forecasting and innovative thinking, supports technology forecasting of physical systems. In general TRIZ and ARIZ can be applied to software engineering problems as proven in [3][4][16][18][19][3][20].
1.5 Current software engineering practice

There is a tendency among software engineers to forget to make a complete software analysis work. Introduction of object oriented methods, and wide interpretations of methods like eXtreme Programming (XP) and rapid prototyping, increase the belief on a seamless transition from analysis, to design and further on to implementation.

*A fool with a tool
... is still a fool.*

What is too often forgotten is that software engineering and certainly software analysis work is an intellectual process, where different views are processed and transformed. There is no one-to-one relationship between problem and solution. Well performed intellectual labour results in good solutions. But if software engineers too often jumps to conclusions on implementation, from over specified requirements on design and implementation level, the implementation results becomes too rigid. So, if software robustness is wanted, one better concentrate on software analysis.

*The solution was perfect.
But it was not the right problem.*

If the analysis of the problem is systematically done and the problem completely understood, then there is a good foundation for further development. This does not mean that we have to follow an old waterfall oriented software development model. It merely means that software analysis should be kept at a reasonable level. Analyse as much as is economically reasonable, without jumping to design conclusions too early, especially when developing large-scale software, based on costly software architectures and third party products.

It is often economically motivated perform a deeper analysis of the problem history and expected future evolution, e.g. a robustness analysis with a certain objective in mind. If a faulty requirement is detected late, the work to correct it multiplies, through every development stage from analysis, to design, and on to delivery, and in service operation.

It is extremely costly to correct faults in software that exists due to misunderstood or poorly understood problems, as this results in incorrectly directed requirements statements.

1.6 Gaps in previous papers and methods

Execution of a robustness analysis, gives a good grasp of the problem situation. A simple domain object model does not give room for in-dept re-modelling of the problem space and system analysis model.

Not even the famous RUP methods include guidelines on how to do a deeper robustness analysis, taking into account system evolution aspects.

Lehman [15] has studied software evolution and come up with a set of software system evolution laws. But these laws are only applied on software maintenance problems, not on prediction of future evolution.

Design patterns and architecture patterns captures experience on how to build successful systems [3]. Design patterns can be used as software modelling abstractions to even help us to predict different software evolution paths. Assume we have a system where we apply two different architecture patterns, one at the time, e.g. Pipes-and-filters, and Model-view-control. The result is then two different system evolution directions for the system in mind.

The use of design patterns together with formal methods is rare in the popular methods investigated, but there are exceptions [6]. Though design patterns provide useful heuristics and are of great help, there is a common misconception that patterns guarantee reusable software and higher productivity [23].

Remarkable none of the popular methods investigated, combines systematic software technology forecasting with analysis and design for future evolution and demands [21].

Papers written on the method application of TRIZ on software problems, indicates that the potential for pure application of TRIZ on software problems, are low, since software problems often are considered to be of the type: understand the requirements and code as quick as possible. To make full advantage of principles and ideas of TRIZ, the method has to be translated and adapted for software engineering. A reason for this is that the laws of physics are not directly applicable for software engineering.

1.7 Introduction to research

With this as a background, a few new questions arise:

- Is it possible to predict the evolution of software units for the future?
- And if so, how can we prosper upon such knowledge to make better software solutions?
- And is to be recognized as state of the art and best practice, among software and system oriented processes?
This paper evaluates some popular software and system engineering methods, and best practises, concerning their abilities to perform robustness analysis and to contribute to technology forecasting of software systems. As technology forecasting is a relatively new discipline for software engineering, methods with origins outside the software business have been introduced.

2 Problem

2.1 Description

To design the product of tomorrow, already today, and to write perfect software from the beginning, is a dream for many software engineers. Software engineering is usually seen as an iterative process where we have to analyse, design, implement, correct, and re-write the software multiple times.

To be first with a technologically mature product on the market would give a tremendous market advantage.

Though the software engineer profession is said to be quite new and immature, it has still be around for at least 50 years at the time of writing. Software engineers have become better at using formal methods, and tools. Quality assured software development processes are acknowledged by most software engineers, though not as often put into practice.

Then why is it so difficult to develop the right software during the first development increments, or even first product releases?

It is a generally accepted that most types of software installations cannot be 100% specified before they are designed. Once the software has been installed, it will start influence its system environment, thus changing the demands upon itself. This is one of the major motives behind iterative software engineering.

Still, some software projects develop much more successful software from the beginning, than others. The use of design patterns, as explained in [3], is gives the software engineer a tool to build more successful software, by reusing design experience as heuristics from earlier successful software design projects.

If a software program is followed through out its lifecycle, it starts evolving to survive new emerging requirements. According to Leman [8], it is necessary to maintain software, with a repertoire of counter actions, to avoid that software degenerates, becomes useless, and too costly to maintain.

2.2 Problem definition

This leads us to the problem definition of this research survey.

1. Which popular methods for software engineering address robustness analysis with technology forecasting, or prediction of system evolution.
2. How do these methods and concepts predict future changes in system environment?
3. How do they combine robustness analysis and technology forecasting?
4. How may these methods contribute to an improved software engineering method that integrates technology forecasting and robustness analysis, to let software developers develop software that meets future demands today.

3 Method

3.1 Definitions

- Coad/Yordon Structured Analysis for Real-time systems, is an old method for analysis of real-time systems.
- Design Patterns, are reusable earlier proven successful abstract design solution, applicable to concrete problems.
- Objectory (OrySE), was Objective System’s software development process.
- Occam’s razor, meaning the simplest method out of two, that have similar results, is the best method.
- RUP, is Rational’s Unified Process model.
- Robustness Analysis, also denoted Analysis-Object Model in Objectory, and Analysis Model in RUP. It is the process to develop a model that leaves the system tolerant to changes. This is possible by analysing future potential system changes, to reveal the system’s drawbacks and imperfections. Popular methods use design heuristics to model the system tolerant to changes.
- Software Evolution, is the change of essential properties of a software product family over the time [1].
- Technology Forecasting, is the process of predicting future evolutionary or revolutionary changes in a technological system. This is done by studying existing system generations, and to identify current system maturity in terms of technology evolution, history and expected evolution steps. Intensive study of system conflicts and their, may also reveal more, or less, likely system evolution leaps. Knowledge of concepts for these evolution leaps lets the software engineer introduce inventions in the software system studied.
• Trends of evolution / Prediction Tree. Published prediction trees and trends of evolution have appeared in over 21 books on TRIZ, and they also appear in the Invention Machine Labs software. These trend-curves and prediction trees are applied to the objects and actions in the functional statement, in order to forecast next-generation designs. These trends are also expressed in over six dozen "standard solution" formats, which are expressed symbolically using Substance-Field Analysis.

• TRIZ, the Russian acronym “Teorijz Rezhenija Izobretatel’skich Zadach” means the “Theory for Inventive Problem Solving”, also denoted TIPS. TRIS is a method usually applied on non-software engineering problems, like mechanics, physics, chemistry, and electronics.

• XP (eXtreme Programming) is a concept of developing software by focusing on source code development, and reduction of work overhead caused by unnecessary formalism. XP is similar to the software development concept, evolutionary software prototyping.

3.2 Search methods

Material for analysis has been collected from: software engineering books; www.google.se web search tool; RUP on-line documentation, www.icee.org, www.research.ibm.com, patent databases; www.triz-journ.com; and NEC Research web site CiteSeer http://citeseer.ist.psu.edu/. The following are some examples of search keywords and phrases: robustness analysis; evolution patterns; design patterns; architecture patterns; software evolution; TRIZ; TRIZ software; UML;

3.3 Selection of evaluated methods and concepts

In this study some popular software engineering methods have been selected for deeper analysis. As software engineering trends have changed back and forth the last years. The old methods: Coad Yourdon’s Structured Real-time Analysis, and the Objectory method have been selected for evaluation. The reason is that these methods where considered to be rather strong for software analysis. Other methods like ROOM, a real-time oriented software engineering method; has been omitted from this study. The focus in ROOM is on real-time aspects and embedded systems, not on robustness analysis and technology forecasting.

Specific military standard oriented methods have also been omitted.

The methods and concepts selected are:

• Objectory method (ORYSE), Robustness Analysis. Objectory was the first method to introduce robustness analysis, though it was called Analysis Object Modelling.

• Rational Unified Process (RUP) Robustness Analysis method [12]. RUP includes Robustness Analysis, but name it Analysis Modelling. It inherits its way of performing software analysis from Objectory, among other methods. RUP is considered the mainstream de-facto standard for many software engineers.

• Jan Bosch’s methods described in [1]. Bosch describes hot to evolve product-line architecture using Darwinism. The expectation on the method was to study how technology forecasting and robustness analysis, or similar, where integrated in Bosch’s concept.

• Coad / Yourdon structured analysis of real-time (SART) systems. This method is merely used as a reference, as it is quite old by the time of writing. When object oriented methods where introduced in around 1993, the Yourdon SART method, kept its standpoint that analysis and design was two separate activities. The software analysis phase of Yourdon SART was built on a clear separation of external system interfaces, boundary, data entities, and control or function oriented process abstractions.

• ICONIX Process, Use Case Drive Object Modelling – A 99% Fat-Free Approach [20]. This method concept is studied, because it is new, and it created as a reaction on over complicated methods like RUP. The objective behind ICONIX has been to include what is needed but nothing else, still Robustness Analysis is a part of ICONIX.

• TRIZ applied on software as described in [3]. The method TRIZ comes from non-software oriented engineering disciplines, but has lately been applied on software. There is no complete adaptation of TRIZ for software engineering, but the attempts made gives an insight on how to do technology forecasting.

• Principles behind eXtreme Programming (XP). The software development concept XP, was selected as it represents a way of building software with a minimized amount of administrative documentation. As the application of XP is different among organizations and groups, it is difficult to refer to the way of working as a method standard. So, the XP concept has been omitted from further studies in this paper.
• Architecture Trade-off Analysis Method (ATAM) [4]. The software architecture engineering method ATAM, describes how to develop an iterative architecture using quality attributes and evolutionary principles to design the best fitted architecture, for product-lines. ATAM was studied concerning its way of integrating technology forecasting abilities with software analysis.

• Attribute-Based Software Evolution [21]. This is a concept, similar to ATAM, which has taken Software Technology Forecasting in consideration, when designing and evolving software quality attributes.

3.4 Data collection methods

Data collection has been made by reading method descriptions, other papers, and testing of methods. The author’s professional experience has also resulted in observations made and reflections on software engineering, and method usage. The author has professional experiences from: 17 years in software engineering business; 3 years as teacher in Systems Architecture Work, and Development with Design Patterns.

3.5 Evaluation method

The study is made as a combination of qualitative comparisons of software engineering methods and concepts for technology forecasting and robustness analysis. When data is collected and rated according to a questionnaire form, it can be seen as a quantitative measurement and rating of each method’s capability to perform robustness analysis and software technology forecasting. But this is not the main intention with the study. The purpose is to evaluate assets and best practices of each method, to make it possible to combine the results in practical software engineering projects.

So, each method has then been evaluated and studied concerning its ability to provide useful methods, techniques, principles and concepts for robustness analysis and software technology forecasting.

4 Results

4.1 Evaluation Questionnaire

For each of the following investigated methods and concepts, the following questions where asked:
1. Where in the process are Software Technology Forecasting activities located, if any?
2. Where is Robustness Analysis located, if any?
3. How is technology forecasting supported? That is, how are future trends of evolution estimated?
4. How is robustness analysis supported? That is, how are analysis models stressed to manage future likely system changes?
5. What diagram types are used, what notation is used, and what are the purposes of each diagram type?
6. How does the method contribute to a more robust system?
7. How is the future software evolution forecasted?
8. General discussion on each method’s advantage, and disadvantages.
9. Summary

4.2 Objectionry

The Objectionry 3.3, process book was studied. The author also had previous professional experiences from the usage of Objectionry, and its tool, OrySE; which have been considered in this evaluation. Objectionry and RUP shares most of their ways of performing robustness analysis. Objective Systems SF AB’s business process re-engineering method, Objectionry Business Engineering (BE), played a similar role as Business Case analysis in RUP.

So when the evaluation is similar, the reader is forwarded to the evaluation results of RUP, later in this paper.

1. Technology Forecasting is not addressed. See also RUP evaluation.
2. Robustness Analysis is documented in an Analysis-Object Model. Each Analysis-Object is documented in both an Analysis Survey, where Analysis Objects are briefly described and placed in Packages. An Analysis-Object Description documents the objects in detail. A difference between RUP and Objectionry, is that Objectionry performs the analysis using objects, and RUP uses classes. Robustness Analysis was only described in the special version of the modelling tool called OrySE.
3. Though Technology Forecasting is not supported. See also the evaluation of RUP.
4. Robustness analysis is supported as in RUP but no vision document exists. See also RUP evaluation.
5. See RUP evaluation, but without a Business Analysis Model.
6. See RUP evaluation.
7. Software evolution is not included in Objectionry.
8. The advantage of Objectionry is that it was less sophisticated than RUP, thus demanding less administrative work. At the same time the studied Objectionry version gives more detailed instruction to the novice software engineer. This is mainly because it is a predecessor of RUP, which also count as Objectionry’s drawback.
9. RUP is a superset, including Objectionry, and more.
4.3 Rational Unified Process

1. Technology Forecasting is not addressed by the method as such, but estimation and analysis of future conditions are treated in Business Process Analysis, where information is collected for storage in software.

2. Robustness Analysis is documented as an Analysis Model, in the RUP elaboration phase, and it is updated during construction phase. The Analysis Model is optional in RUP.

3. Though Technology Forecasting is not addressed, future software evolution requirements are indirectly considered during Business Process Analysis, and Stakeholder analysis. The artefacts Business Vision, Vision and Stakeholder Requests, document primary current and long-term needs.

4. Robustness analysis is supported by analysing each Business Use Case and Use Case to find which object that collaborates in the realization of the Use Case. The separation of Boundary Classes that is external interfaces; Control Classes modelling functionality and algorithmic knowledge; and Entity Classes modelling information structures, follows the GRASP design pattern, Control Object, and the principle of Separation of Concerns. This principle, itself does not necessary contributes to a more robust system, but in combination with good knowledge about stakeholder requests and system vision, future system change can be anticipated.

5. The diagram types used are: Business Analysis Mode, and Analysis Model. In the Analysis Model, three different class stereotypes are used: boundary classes, control classes, and entity classes. In the Business Analysis Model, which models overall business behaviour, the stereotypes are: Business Systems; Business Workers; Business Entities; and Business Events. The Business Analysis Model analyses the overall purpose of the software system, in its business process context. The Analysis Model gives guidance when finding classes. The stereotyping results in a robust object model because changes to the model tend to affect only a specific area.

6. The method contributes to a more robust system by enforcing the separation of concerns principle on analysis objects. It supports division of complex tasks, by splitting control classes into two classes; e.g. a Task Performer class and a Queue Handler class. In Use Cases where multiple actors are involved, the control class can be split to model each of the actors’ behaviour. Restrictions on communication associations and dependency relationships between Boundary, Control, and Entity Classes, enforce the architecture principle, where the presentation layer, depends on logic, and information layers; and the logic layer uses and depends on the information layer, only. This contributes to lose couplings in the software.

7. Software evolution is not forecasted in RUP. But, future expectations are captured in stakeholder requests, still these future likely demands are not analysed, nor modelled.

8. The advantage of RUP in the context of software technology forecasting and robustness analysis, is on robustness modelling. RUP is considered a competent and relatively complete software engineering process. System scope and system environment is considered at an early stage. The weakness of RUP is that one often needs an increment planning wizard to plan the software increments, meaning that RUP is costly to work with as the administrative tasks can easily consume most of the work, if RUP is not restricted through tailoring. The risk of analysis paralysis is high.

9. RUP’s contributes with means to analyse system related background problems through stakeholder analysis, and business process analysis. Its robustness analysis principles are easy to adopt. Software technology forecasting is not modelled at all.

4.4 Jan Bosch’s method

This following evaluation is based on the methods and principles describes Bosch’s book [2]. Bosch’s method share similarities with the ATAM method.
1. Bosch’s book does not describe a pure process or method, instead he discusses ways of: Designing Software Architectures; and working with Software Product Lines. In his method descriptions, he suggests using quality attributes as means to reach robust software architectures, instead of expertise of software systems architects which express tacit knowledge. His method proposal is very similar to ATAM as explained in [11]. The Bosch does not directly address the issue of Robustness Analysis. Technology forecasting is not directly addresses either, but different design and architecture patterns are used as tools to transform the system, and then to measure Predicted QA values. Predicted QA values are based on Quality attribute prediction, which is based on an Impact analysis of the actual software architecture in a Scenario Profile described by several Scenarios.

2. Robustness analysis is not covered.

3. Technology forecasting is not directly addressed, but evolutions of Software Product Lines are covered. Evolution is seen to take place after a product has been developed and deployed. In the creation process of product lines, the following trends are anticipated: evolution of an existing set of products into a product line; replacement of an existing set of products with a software product line; evolutionary and revolutionary evolution of new software product line; and finally, development of a new software product line. Three kinds of software evolution within a product line are recognised: architectural evolution, component evolution, and product evolution.

Concerning scenarios for evolution of product lines, Bosch has identified a number of demands that results in evolution of product-lines: new product line; introduction of new product; adding new features; extend standard support; new version of infrastructure; and improvement of quality attribute.

4. Robustness analysis is not supported. Measurement of quality attributes may indicate level of robustness. Bosch’s method descriptions aim to guide development of architectures, where robustness aspects are covered. Good product line architectures are able to outlive future changes in environment and technological evolution.

5. No specific diagram types are used for description of evolution or robustness analysis.

6. The method contributes to evolving product line architectures, by measuring quality attributes and following up attribute values, through software development increments.

7. Future evolution is not forecasted, but design patterns and architecture patterns are used for system transition of product line architectures. In theory, it is possible to first set up and measure quality attributes and then apply different system transition techniques (design and architecture patterns) to evolve the product line. Scenario’s acts as an evaluation morale scope for evaluation of the quality attributes for product lines architectures.

8. The advantage of the method is that it operates on architecture level, and product line architecture level. There is a strong asset to be able to measure software architecture quality attributes, and to track its evolution to a more ideal. Robustness analysis, and technology forecasting is not really addressed by the method.

9. Measurement of Quality attribute using Scenarios seems to be a promising technique.

4.5 Coad / Yourdon SART

The analysis of Coad / Yourdon’s method for Structured Analysis of Real-Time systems, is based on the author’s earlier experiences for development using the method for both development of object-oriented software, and traditional software developed in C and Pascal. Yourdon SART is considered a non-object-oriented method.

1. Yourdon SART, does not include technology forecasting.

2. The method does not include traditional robustness analysis, but the analysis phase of the method bears similarities to robustness analysis.

3. Technology forecasting is not supported by the method.

4. Robustness analysis is supported by the analysis phase which is kept completely implementation free. Still the analysis phase have a clear separation between interfacing information, parallel executing processes “bubbles”, and internal data storage. The result is a modelling effect similar to what is achieved in RUP, and Objector’s Robustness Diagrams, where boundary objects, control objects and entity objects are used.
5. Diagrams are described using process bubbles, drawn as a circle, with information processing capabilities. Inputs and outputs to/from a processing bubble are described as files, and drawn as rectangles. Internal information, similar to entity objects, act as information structures. Internal information structures are pictured as a laying rectangle with no right/left borders. All information structures are described using the syntax description language, Extended Backus Naur/Normal Form (EBNF). Software hierarchies are described by zooming inside other processing bubbles.

6. According to the method description, it strength to view the software from three different viewpoints: function structure, data structures, and execution flow/sequences. The motivation is that this allows information structures to be modified, independent of software flow.

7. Future software evolution is not forecasted, but it is common that the analysis phase is extended beyond what is later implemented.

8. The method’s advantage for software engineering, is that it is lean from an administrative view-point, and that it allow for advanced real-time processing descriptions, where process abstraction is strong, and information and stimulus may be described to be either single shot, or continuous.

9. Though Yourdon SART is quite old, it still holds assets and properties that remain, even in modern methods and processes like RUP. An interesting fact about Yourdon SART, is that the analysis phase is said to be 100% implementation independent. That means that the developer may reuse the analysis model and design implement the solution in software, hardware, or a combination, e.g. using VHDL to implement functionality in a Field Processor Gateway Array (FPGA) circuit.

4.6 ICONIX Process

This process is a reduced version of RUP, where administrative labour has been reduced. To further increase its efficiency, the ICONIX process has reduced the number of diagrams to be used. Certainly, a process like ICONIX will work for most projects, but there will still be projects that demands more extensive modelling, e.g. state diagrams, for some reason. So the ICONIX Process is not a general purpose software engineering process, like RUP. Well RUP does not cover modelling of real-time systems, and systems that combines hardware and software.

1. There is no technology forecasting included in the ICONIX Process.

2. Robustness Analysis is located directly after each use case has been modelled, as each use case is realized using a robustness analysis model. This is similar to RUP.

3. Technology Forecasting is not supported.

4. Robustness Analysis is treated the same way in the ICONIX Process as in RUP, but control classes are always given the same name as its Use Case, and entity and other control classes are given concrete design names that will be used during implementation. One, could say that the ICONIX Process drives the Robustness Analysis activity one step further towards design, as design objects are referred to during analysis. This makes the analysis bound to its intended design environment.

5. The ICONIX Process uses the same UML diagrams that are used in RUP, but limits these to Use Case Model, Robustness Diagram, Sequence Diagram, and to visualize static views Domain Models, and Class Diagrams are used. State diagrams are not included.

6. The process contributes to robustness in a similar fashion as RUP, but does not include the Business Process modelling step, which means that stakeholder analysis might get weaker.

7. Future software evolution is not considered at all.

8. The advantage of this process, method is that administrative tasks and not-so-important diagrams are omitted to fit lean software engineering. The process also takes some short cuts in its modelling steps, e.g. state diagrams are omitted. The drawback is that the process does not fit general problems, and that the adaptation of RUP is something that is done once for each system category to be developed.

9. One could say that the ICONIX Process is a tailored version of RUP suitable for small software projects, still requiring robustness analysis.

4.7 TRIZ

TRIZ is a general purpose tool for working with problems requiring inventions, or innovations in any engineering field, where physics, mechanics, chemistry and electronics are used for solving problems. That means that TRIZ is not built for software engineering problems. Still TRIZ methods have proven useful for software engineering problems; though no real TRIZ for software exists. TRIZ includes methods for: analysis of system conflicts; ARIZ, a method for intensification of system conflicts; and a method for managing knowledge about how to resolve system conflicts, handling of system engineering parameters, similar to ATAM’s software quality attributes.

An earlier unpublished paper [3] written by the author of this paper, translates and evaluates TRIZ
system evolution laws, to function in software engineering. This enables software engineers to predict possible evolution of software systems.

Earlier studies on TRIZ and software [5] have described how to apply parts of TRIZ on software development problems, and software engineering.

This evaluation of TRIZ is based on [23] but not limited to that description. TRIZ course documentation from year 2000: TRIZ Methodology for Innovative Engineering; Three-Day Hands-On Workshop, with Victor R. Fey from the TRIZ Group; is used as the main documentation of TRIZ.

TRIZ processes are organized as follows:

1. TRIZ has a wide number of supporting methods, like AFTER-96 [12]. Most methods of TRIZ, are in one way or another involved in technology forecasting.

2. Robustness Analysis is not directly covered by TRIZ, but TRIZ instead, analyses the problem situation, and contributes to a problem formulation.

3. TRIZ’s Problem Formulation method and earlier phases are seen as analysis; it is first during Contradiction Analysing, when future technology evolution is modelled. Other concepts that contribute to technology forecasting, are: Ideal Design; System Modelling, with Substance-Field Analysis; Patterns of Evolution. Also engineering knowledge tools like: Physical Effects and Phenomena; 39 Engineering Parameters, which are similar to architecture quality attributes; and 40 principles innovation; contributes to technology forecasting and creation of innovations.

4. The analysis of contradictions, and the use of the tool ARIZ, helps engineers “intensify system conflicts”; thus revealing and solving inner system conflicts. In TRIZ papers, it has been recommended to combine TRIZ with the Taguchi method, to ensure satisfied customers.

5. The diagram types used in TRIZ are:
   - Diagrams to do problem analysis and to visualize relationships between Harmful and Useful functions.
   - Substance-Field Models, a diagram to model a Minimal Technological System. Such a diagram often start by describing the relationship between three nodes called: the Object, that we want to affect; the Tool, which is used to affect the Object; and Energy, that represents engineering fields as Mechanical, Thermal, Electrical, Magnetic and Chemical fields. The field represents communication mechanisms between the Tools, which influence the Objects. In computer science, this would be similar to communication protocol or access method.

To analyse Harmful and Useful functions, the following notations are used:
- Useful function
- Harmful function
- ----> Eliminates
- = = > Causes
- -------> Is required for
In Substance Field models, the following notation is used:

- \( D \) A Su-field
- --- Unspecified (inter)action
- ---> Specified action
- === Transition from Su-field structure to desirable structure
- ~~~~ Harmful, unsatisfactorily action
- \( \rightarrow F \) Output field
- \( F \rightarrow \) Input field
- \( S' \) Modified substance
- \( F' \) Modified field

6. The method contributes to a more robust system, by performing a deep technical analysis of the primary functions to be performed. This lets the engineer understand the primary function of the system, and how the system will evolve. Then during design, it is possible to anticipate future evolution.

7. Future evolution is forecasted using analysed primary system functions to specify an Ideal system. Then system conflicts are analysed in detail and resolved using a knowledge base of known Technical Effects. These effects let the designer chose how to best solve the contradictions, without introducing new harmful effects. Thus, the engineer is able to invent a new system generation. Also, technology forecasting takes place during analysis of system maturity, and by the use of known evolution trends in technical systems, the Patterns of Evolution. By applying these evolution patterns on the actual system being studied, it is possible to foresee how the system may evolve in several dimensions. Then it is up to the engineer to identify which evolution trends that are most likely, and fruitful, concerning the system’s driving forces.

8. The TRIZ method has an obvious advantage for modelling of innovations and for doing technology forecasting. But it is not adapted for software engineering problems, and it is questioned that software engineering really requires level innovations. Most software innovations appear as level 1 and level 2 innovations, according to TRIZ innovation level scale [3].

Table 1: Levels of inventiveness in general

<table>
<thead>
<tr>
<th>Level</th>
<th>Degree of inventiveness / Description</th>
<th>% of solutions</th>
<th>Source of knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apparent solution</td>
<td>32%</td>
<td>Personal</td>
</tr>
<tr>
<td></td>
<td>Improvements localised in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a single sub-system of the</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>technological system that</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>do not resolve any system</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>conflicts.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Minor improvement</td>
<td>45%</td>
<td>Within company</td>
</tr>
<tr>
<td></td>
<td>System conflicts have been</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>resolved in similar systems /</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>applications.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Major improvement</td>
<td>18%</td>
<td>Within industry</td>
</tr>
<tr>
<td></td>
<td>System conflicts are resolved</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>within one discipline, e.g.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mechanical, chemical, physics. One</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>element of the system can</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>be completely changed, maybe</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with potential change of other</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>elements.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>New concept</td>
<td>&gt;4%</td>
<td>Outside industry</td>
</tr>
<tr>
<td></td>
<td>Development of new system.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System conflicts are resolved by</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interdisciplinary approaches. E.g.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mechanical problems are solved by</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>chemical or electrical engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>techniques. The development concept</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>may be applied to many problems of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lower level.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Discovery</td>
<td>&lt;1%</td>
<td>All that is knowable</td>
</tr>
<tr>
<td></td>
<td>Pioneering invention which usually</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>requires creation of a new</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>engineering discipline.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many researchers expect that TRIZ has a really high potential to enlighten software development, as for example described in [5]. The disadvantage of TRIZ is that the TRIZ methods are somewhat unstructured, and focus on small scale problems, rather than management of system complexity which is one of the biggest issues for software engineers. As mentioned, there is no generally accepted and published method for the application of TRIZ in software engineering.

9. The summary is that TRIZ may fill a gap in current software engineering methods, as it contributes with innovative thinking, technology forecasting techniques, and methods for systematic system conflict resolution. TRIZ is a promising method for cross breeding with existing software engineering methods.
4.8 Architecture Trade-off Analysis Method (ATAM)

The work procedure suggested for ATAM is as follows:
1. Present of ATAM
2. Present business drivers
3. Present the architecture
4. Identify architectural approaches
5. Generate the Quality Attribute Utility Tree
6. Analyse the architectural approaches
7. Brainstorm and prioritise scenarios
8. Analyse the architectural approaches
9. Present and report the results

For each of the following investigated methods and concepts, the following questions where asked:
1. Technology forecasting as such is not included in ATAM, but ATAM provides tools to evaluate investigate risk themes, based on evaluation of architecture and business needs and scenarios.
2. ATAM does not include a robustness analysis method, but the method guides an iterative architecture improvement spiral, that results in more robust systems and architectures.
3. Future trends of evolution are investigated by first stating the needs in a Quality Attribute Utility Tree, and then iteratively evaluate, and refine the system architecture to support the needs stated.

Table 2: Example of a Utility Tree

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Q Attribute Refinement</th>
<th>Q Attribute Scenario</th>
<th>Priority / Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Data latency</td>
<td>Minimize latency on DB to 10 ms</td>
<td>Medium / Low</td>
</tr>
<tr>
<td></td>
<td>Throughput</td>
<td>Deliver real-time video</td>
<td>High / Low</td>
</tr>
<tr>
<td>Modifiability</td>
<td>New product changes</td>
<td>Low / High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change of platform</td>
<td>Add CORBA in less than 20 man weeks.</td>
<td>High / High</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Launch on Linux in less than 6 months.</td>
<td>High / High</td>
</tr>
<tr>
<td>Availability</td>
<td>HW failure</td>
<td>Low / High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RPC failure</td>
<td>Low / Low</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>Data confidentiality</td>
<td>Authorizati on works in 99.98% of time</td>
<td>Low / High</td>
</tr>
<tr>
<td></td>
<td>Data integrity</td>
<td>High / Medium</td>
<td></td>
</tr>
</tbody>
</table>

4. The major goals of ATAM are to: elicit and refine precise statements of quality attribute requirements together with architectural design decisions; and to evaluate architectural design decisions to determine if they address the quality attribute requirements. During this iterative work-flow, robustness is achieved.
5. Architecture goals are modelled using Attribute Utility Trees, and Scenario cards, that describe standpoint scenarios for evaluation of architectures.
6. ATAM helps software engineers to explore software system architectures, and to assess the architecture, using a trade-off analysis. During the trade-off analysis, architecture oriented requirements from stakeholder and experts are evaluated. The evaluation is made based on: experience; scenarios; and models.
7. By using applying each evaluation scenario, it is possible to evaluate future architectures, before they are implemented. So, ATAM guides dry-run of a likely evolution of software, and systems architectures.
8. The big advantage of ATAM is that it is built on estimated existing and future scenarios that describes the morale for analysis of software and architectures. As software qualities are measured using the Utility Tree, the evaluation is based on facts. The major disadvantage is that ATAM mostly focus on pure architecture evaluation.
9. ATAM is a method for architecture evaluation, where the evaluation results rate architecture solutions, and indicate architectural risks. ATAM looks promising for doing fact based evaluation of software evolution, using scenarios and software quality attributes.

4.9 Attribute-Based Software Evolution, as described in [21]


1. The method starts of with technology forecasting, by defining the qualities that the core product line will have to support in the future. Then TRIZ Laws of Technological Systems Evolution is applied, to forecast the evolution of product lines.
2. There is no robustness analysis involved in the process, but quality and functional attributes are measured and followed up through the application of evolutionary patterns.
3. The technology forecasting follows the following procedure:
   a. Represent every product as a set of functional and quality attributes.
b. Find the intersection of these sets. The architecture that supports these goals represents the core product line.

c. Use evolution patterns (actually TRIZ’s Laws of technological system evolution), to estimate the conflicts that can arise from the need to support future goals of the already existing products.

d. Refine the core architecture to accommodate possible future goals of already existing products.

e. Use evolution stories for products similar to envisioned products to estimate possible future conflicts. Also use evolution patterns to estimate possible future goals based on the already existing set of goals.

f. Decide if it is feasible to support envisioned goals using the proposed product line, and if it is, then re-factor the architecture of the common assets to form the core of the product line and to optimise the common attributes.

4. Robustness is not supported.

5. No specific diagrams have been proposed.

6. The method contributes to a more robust system in the same fashion as ATAM.

7. See point 3, above.

8. The advantage of the method is that it combines evolutionary patterns to design better product-line architectures. This is a step missing in ATAM. The use of Functional Attributes and Quality Attributes lets software engineers define target Quality Goals for the future product-line’s architecture. Quality Goals (QG) are classified for: Non-system QG which includes Business QG, and Project Management QG; as well as System QG with Architecture QG, and Design QG. Design QG is further classified as Run-Time QG and Non-Run-Time QG. The disadvantages of the method are that it only focuses on evolution of product-lines, and not other software systems, and that it does not include robustness analysis, though its overall consequence might be a more robust system. Compared to ATAM, evaluation scenarios are missing.

9. The method combines goal setting for quality and functional attributes to capture the future system’s qualities demands on its product line architecture. Then the method gives some guidance on how to evolve the existing product-line architecture, by applying TRIZ System Evolution Laws.

5 Discussion

5.1 Validity of research

Though the survey only covers some popular methods that can be used for robustness analysis, or software technology forecasting, much can be said about the current state of the art in this field.

Among other alternative methods, that where not included in the detail study, it was found at an early stage that these would not contributed to a discussion on the topic for this paper, that is the combination of, or intersection between Robustness Analysis and Technology Forecasting.

Then, there is a gap between research, software engineering methods, and even industrial application of methods, among professional software engineers. So, methods are theories, and software engineering practices often more ad-hoc execution of theoretical methods.

From this one can conclude that a survey study like this; only can cover a fraction of the state of the art in software engineering practice.

Still, the methods to be evaluated, demonstrate a width in difference to cover the most important aspect of current software engineering methods.

With this as a background, and the survey made in this paper, it can be stated that:

- There is no software engineering method that integrates technology forecasting with robustness analysis.
- The studied methods can be classified according to the three following categories
  - Focus on heuristics, techniques, principles, or patterns to achieve change robust software
  - Use of quality attributes as improvement indicators, to enable design iteration towards better software architecture, or product line architectures.
  - Analysis, modelling with engineering knowledge tools and heuristics, and resolution of system conflicts for inventive problems, with the purpose to invent tomorrow’s system solution.
- None of the investigate methods and concepts, except The Attribute-Based Software Evolution, as described in [21].
5.2 Technology Forecasting

Very little is written about technology forecasting in software engineering, and there are not many methods for predicting how software will evolve.

Meir Lehman, [8] has identified a number of software evolution laws that most often are applicable for maintenance of software, these laws are presented as a courtesy to the reader in appendix A.

From the method TRIZ, which has lately been applied on software engineering problems, provides a drawing technique called evolution trees. Evolution trees are a kind of mind maps that describes possible evolution paths. Description of evolution trees, are presented in appendix D.

The TRIZ method supports Technology Forecasting, using its Laws for Evolution of Technical Systems, among many techniques. These evolution laws are specified in appendix B.

The evolution laws where not applicable for software engineering, as they where originally intended for physical systems. The TRIZ evolution laws have recently been transcribed and briefly evaluated in [3] for software engineering.

A summary of the transcribed TRIZ evolution laws are presented in appendix C, Software evolution laws. A better name for these laws within the context of software engineering; would be to call them: Software evolution patterns.

To further evaluate software technology forecasting using the TRIZ method, the AFTER-96 method [12] has been tested, see appendix D.

5.3 Robustness Analysis

The methods studied shows two ways of analysing and modelling a system’s robustness armour.

First, the RUP, ICONIX Process, and Objectory process applies heuristics, and software engineering principles like: separation of concerns; high cohesion; and low coupling; to achieve maintainable software.

Secondly, methods like ATAM, and Attribute-Based Software Evolution, and Bosch’s approach to software engineering; makes use of Quality Attributes and Function Attributes. These attributes are used for measuring maturity and requirement fulfilment of: software, architecture, and product line architectures. To further improve evaluation of quality attributes, ATAM introduces evaluation scenarios. The evaluation scenarios capture existing and future stakeholder needs.

6 Summary

1. Seven software engineering methods and one general purpose system engineering method TRIZ was evaluated concerning their abilities to combine: software technology forecasting, that is prediction of potential software evolution; with robustness analysis, which is a method to model a system tolerant to changes.

2. Among the studied methods, the TRIZ method was found to be outstanding for performing software technology forecasting, but TRIZ cannot be expected to be useful by software engineers directly, since it is not adapted to software engineering problem space. So, there is no TRIZ methodology for software engineering, though some studies have applied TRIZ on software problems.

3. It is recognized that assets from TRIZ technology forecasting, and contradiction modelling and resolution, ought to be combined with RUP’s robustness modelling and, ATAM’s way of setting evaluation scenarios for architecture evaluation based on incremental architecture modelling and quality attributes for maturity estimation and product line requirement specifications.

4. The author has as a result of this research survey, initiated a study to synthesise and evaluate a new method that combines assets from TRIZ, RUP, and ATAM, to achieve a method that integrates technology forecasting and robustness analysis.

7 Conclusion

There is no single software engineering method or process concept that completely combines software technology forecasting with robustness analysis. But many of the methods evaluated contributes with techniques and principles, that potentially could be combined and give systematically technology forecasting with comprehensively robustness analysis.

8 Further research

It is expected to be valuable to synthesise a method that combines the best parts of the studied methods. As a consequence, such method will need to be scientifically evaluated.

Currently there is no method that combines: the usage of quality attributes; scenarios; and robustness analysis diagrams; with technology forecasting methods as described in TRIZ. So, the author has initiated such a study.
References


*Attribute-Based Software Evolution: Patterns and Product Line Forecasting,*
Software Architecture Group, University of Waterloo, CANADA.

*TRIZ for Perl-Programming,*
TRIZCON2001, The Altshuller Institute, 
http://www.triz-journal.com/archives/2001/05/a/index.htm

*Systematic Innovation; An introduction to TRIZ (Theory of Inventive Problem Solving)*
CRC Press – SZT. Lucie Press, US. 
ISBN 157444111-6

*Uncommon Software Patterns: The Top Ten Misconceptions,*
Appendix A – Lehman’s Laws

This is a summary of Lehman’s laws on software evolution concerning life-cycle evolution and software maintenance.

Lehman’s eight laws on software evolution are quoted from [3] as follows:

1. *Continuing change* – I 1974
   E-type systems must be continually adapted else they become progressively less satisfactory in use.

2. *Increasing complexity* – II 1974
   As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it.

   Global E-type system evolution processes are self-regulating.

   Unless feedback mechanisms are appropriately adjusted, average effective global activity rate in an evolution E-type system tends to remain constant over product lifetime.

   In general, the incremental growth and long term growth rate of E-type systems tend to decline.

   The functional capability of E-type systems must be continually increased to maintain user satisfaction over the system lifetime.

   Unless rigorously adapted to take into account changes in the operational environment, the quality of E-type systems will appear to be declining.

   E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

Appendix B – TRIZ System Evolution Laws

The following table specifies the original TRIZ evolution laws.

<table>
<thead>
<tr>
<th>No</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Evolution in stages</td>
</tr>
<tr>
<td>ii</td>
<td>Evolution towards increased ideality</td>
</tr>
<tr>
<td>iii</td>
<td>Non-uniform development of system elements</td>
</tr>
<tr>
<td>vi</td>
<td>Evolution toward increased dynamism and controllability</td>
</tr>
<tr>
<td>v</td>
<td>Law of increasing flexibility</td>
</tr>
<tr>
<td>vi</td>
<td>Increased complexity, then simplification through reduction</td>
</tr>
<tr>
<td>iv</td>
<td>Law of transition to a higher-level system</td>
</tr>
<tr>
<td>iiv</td>
<td>Evolution with matching and mismatching components</td>
</tr>
<tr>
<td>xi</td>
<td>Evolution towards micro-level and increased use of fields.</td>
</tr>
<tr>
<td>vii</td>
<td>Law of transition to micro-level</td>
</tr>
<tr>
<td>x</td>
<td>Evolution towards decreased human involvement</td>
</tr>
<tr>
<td>xi</td>
<td>Law of shortening of energy flow path</td>
</tr>
</tbody>
</table>

G. Calás, 2004. All rights reserved.
Appendix C – Software Evolution Laws

This part is a reprinted from an unpublished paper [3] by the author, based on TRIZ system evolution laws.

**Table 4: Software evolution laws**

<table>
<thead>
<tr>
<th>Software System Evolution Law</th>
<th>Examples of expected observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evolution in Stages &lt;i&gt;</strong></td>
<td>1. Software system follows the evolution path from pregnancy to decline.</td>
</tr>
<tr>
<td></td>
<td>2. A new product line built enabled by new technology is not likely to replace an existing product that</td>
</tr>
<tr>
<td></td>
<td>has reached its adolescence.</td>
</tr>
<tr>
<td></td>
<td>3. The product shift will take place after some time of crisis, when the previous product has entered</td>
</tr>
<tr>
<td></td>
<td>the decline stage.</td>
</tr>
<tr>
<td></td>
<td>4. Initial development from pregnancy to childhood, is made as unofficial project with a small</td>
</tr>
<tr>
<td></td>
<td>enthusiastic team.</td>
</tr>
<tr>
<td></td>
<td>5. The levels of inventions are highest during pregnancy to birth, with a second peak at the beginning</td>
</tr>
<tr>
<td></td>
<td>of the products adolescence phase.</td>
</tr>
<tr>
<td></td>
<td>6. The numbers of inventions are highest at the end of maturity.</td>
</tr>
<tr>
<td></td>
<td>7. Previous software, system architectures, and platform generations, tend to block newer generation</td>
</tr>
<tr>
<td></td>
<td>software products which are trying to introduce newer technology.</td>
</tr>
<tr>
<td><strong>Evolution towards increased ideality &lt;ii&gt;</strong></td>
<td>1. Increased ideality, as the ratio between the total numbers of functions provided and costs.</td>
</tr>
<tr>
<td></td>
<td>2. Software that evolves according to one of the three typical paths of evolution, towards increased</td>
</tr>
<tr>
<td></td>
<td>ideality, see figure 3.</td>
</tr>
<tr>
<td><strong>Non-uniform development of system elements &lt;iii&gt;</strong></td>
<td>1. Some components evolve faster than others.</td>
</tr>
<tr>
<td></td>
<td>2. Some components become bottlenecks and prevent further evolution of system performance.</td>
</tr>
<tr>
<td></td>
<td>3. Programmers tend to optimise the wrong components.</td>
</tr>
<tr>
<td></td>
<td>4. Performance profiling of software using execution profilers like gprof, indicate different</td>
</tr>
<tr>
<td></td>
<td>performance bottle necks for each software system generation and release.</td>
</tr>
<tr>
<td><strong>Evolution towards increased dynamism and controllability &lt;iv&gt;</strong></td>
<td>1. Increasing possibilities to configure the software from release to release.</td>
</tr>
<tr>
<td></td>
<td>2. The system is expected to have more operational modes in later versions, than in previous ones.</td>
</tr>
<tr>
<td></td>
<td>3. Software objects tend to become more dynamical in later releases, than earlier software releases.</td>
</tr>
<tr>
<td><strong>Evolution towards increased complexity, then simplification &lt;v&gt;</strong></td>
<td>1. The number of functions increases in the system, which increases the complexity.</td>
</tr>
<tr>
<td></td>
<td>2. The increased complexity, eventually becomes simplified, encapsulated, or both.</td>
</tr>
<tr>
<td></td>
<td>3. The software components, replaced complex instructions</td>
</tr>
<tr>
<td></td>
<td>4. Software components evolve into a single powerful component.</td>
</tr>
<tr>
<td><strong>Evolution with matching and mismatching components &lt;vi&gt;</strong></td>
<td>1. Continuous evolution towards symmetry for some system parts.</td>
</tr>
<tr>
<td></td>
<td>2. Continuous evolution towards asymmetry for some system parts.</td>
</tr>
<tr>
<td></td>
<td>3. Asymmetrical design is found to give surprisingly better performance qualities, when applied.</td>
</tr>
<tr>
<td><strong>Evolution towards decreased human involvement &lt;vii&gt;</strong></td>
<td>1. Newer system releases reduce human tasks with automatic processes.</td>
</tr>
<tr>
<td></td>
<td>2. Wizard functions are introduced to reduce tedious human tasks.</td>
</tr>
<tr>
<td></td>
<td>3. Software installation becomes more and more automated.</td>
</tr>
<tr>
<td><strong>Law of transition to micro-level &lt;viii&gt;</strong></td>
<td>1. Systems evolve towards increasing degree of fragmentation, of inner elements and sub systems.</td>
</tr>
<tr>
<td></td>
<td>2. Object granularity increases.</td>
</tr>
<tr>
<td></td>
<td>3. Independent processing units becomes smaller.</td>
</tr>
<tr>
<td></td>
<td>4. Processing goes from batch oriented to stream oriented.</td>
</tr>
<tr>
<td><strong>Law of increasing flexibility &lt;ix&gt;</strong></td>
<td>1. Systems become capable to adapt to changing environmental conditions.</td>
</tr>
<tr>
<td></td>
<td>2. Systems evolve to become capable of adapting performance capacity.</td>
</tr>
<tr>
<td></td>
<td>3. The evolution goes from non-dynamic system to a system with multiple states, to a system with</td>
</tr>
<tr>
<td></td>
<td>variable number of states.</td>
</tr>
<tr>
<td><strong>Law of transition to a higher-level system &lt;x&gt;</strong></td>
<td>1. Subsystems are moved to super-ordinate systems.</td>
</tr>
<tr>
<td></td>
<td>2. Super-ordinate systems replace sub-ordinate systems completely.</td>
</tr>
<tr>
<td></td>
<td>3. Subsystems merge into new systems.</td>
</tr>
<tr>
<td></td>
<td>4. When sub-systems merge into new systems, some or some parts of the sub-systems can be removed.</td>
</tr>
<tr>
<td><strong>Law of shortening of data flow and transformation path. &lt;xi&gt;</strong></td>
<td>1. Protocol stacks evolve from containing multiple layers, to be performed in a single optimised step.</td>
</tr>
<tr>
<td></td>
<td>2. Less number of fine granulated objects</td>
</tr>
<tr>
<td></td>
<td>3. Shorter paths for data transition</td>
</tr>
<tr>
<td></td>
<td>4. Less number of protocol layers when accessing databases.</td>
</tr>
<tr>
<td></td>
<td>5. More storage of data in local memory, than in common persistent memory.</td>
</tr>
</tbody>
</table>
Appendix D – A Case Study Using the TRIZ AFTER-96 method

This is a description of a part of the TRIZ method and its diagram for analysing and describing system evolution. It is assumed that these principles are mostly valid for software engineering as well, though this is not yet proven.

AFTER stands for Algorithm for Forecasting Technology-Evolution Roadmaps.

The method is described below, and now tested using a real-life software problem, that is an export/import function for a geographical information system. This demonstrates how to apply the AFTER-96 method on software problems.

9.1 Phase 1: Definition and Analysis

Determine basis for forecasting

What is the primary function?
- To interchange information and states with other systems in different space, semantics and time.

What does this function, or system consists of?
- Import, export of data, a data container for target format, and a container for source format, and a set of rules (could be software program steps) on how to transform data, in each direction, and finally a user/usage interface.

What part do we want to apply technology forecasting to?
- The export function.

What are the criteria for the selection?
- E.g. optimal speed of export function.

Examples of criteria based questions:

a) What parts of the software system are deficient?
b) What parts of the software are candidates for improvement?
c) What part of the software, after it is improved, will contribute to the product’s becoming a “Market-breaker”?

- A redefinition of the system functional statement is: a specified part of the system states and events, are to be exported, that is it shall synchronize a domain of the external system’s information states and events.

The above statement is broken down in a functional statement of the form, “Subject Verb Object”.

Apply TRIZ Substance Field analysis for forecasting

In TRIZ, the Subject is referred to as Substance 2, or S2, that is “a specified part of the system states and events” ---- synchronize ----> S1, that is “a domain of the external system’s information states and events”.

Substance 2 is the active substance, where changes first appear, and Substance 1 the passive. The purpose of this function is to control Field 1, or F1, denoted “synchronize”, that is the synchronization data flow.

The analysis of the function connected to Field 1, can proceed as the item for technology forecasting has been defined.

9.2 Phase 2: Operations

The following “Technology Forecasting Operators (TFO)” can be applied to objects and actions under examination.

The Four Relationship Curves Operator

Determine where the current product is on the S-Curve of evolution for each of its functions. Each function will have a different S-Curve. This information can be used to determine product introduction strategies.

An analysis of the software problem studied, indicates that the system is to be considered mature, according to the S-curves.

Circular Evolutionary-Patterns Diagram

The software for our case study is rated on the seven scales indicated by the circular evolution patterns diagram:

- Dimensionality:
  Not applicable

- System multiplicity:
  The software is a bi-system with function and anti-function. A single process is occupied. Multiple instances may exist.
  Thread safety is not ensured.

- Waste of substance:
  Not applicable.

- Dynamic capability:
  Format cannot be changed dynamically, nor the number of processing nodes, nor allocation of execution to multiple hardware nodes.
  Executed in batch mode.
Function, system, and probabilities:
This aspect could not be analysed…

Degree of system voidness:
Not applicable.

The “Four Parts” operator

Divide the system into a “Four Parts” system, consisting of:

a) ENGINE - The source of energy.
   That is analogous to the information in the system.

b) TRANSMISSION - The part that connects the energy source with the output.
   Analogy in software could be the file system, or a communication protocol for database export format, e.g. bulk copy format, RPC, or XML.

c) LIMBS - The part or parts that do the work, or handle the function.
   In software this is the machinery that picks information from the database and converts it to the export format. So, one could imagine that the Engine, the information drives the information translation functions, that are the Limbs.

d) CONTROL(S) - The part or parts that control the way limbs conducts to make for a productive operation, and provide consistency and reliability to the actions of the limbs.
   In software this is similar to a control interface to a class, or software module. In our example, the Control, corresponds to the interface to the import and export mechanism, e.g. an application programmers interface (API), or a component interface.

The “Four Stages” operator

There are four stages that can be used to assess what stage the current system parts being forecasted are in.

1) Synthesis:
   Finding the right combination of parts
2) Improving:
   Improving the parts
3) Dynamization:
   Making the system dynamic
4) Self-development:
   Providing the system with the capability of self-development.

In our case, the system that we are investigating is in stage 2), thus emerging stage 3).

The Scale and Scope Operator

Scale for our system is class, or software component level.
Transformations of database information to and from external formats are necessary, since different system providers have chosen to maintain and develop systems with incompatible information structures. Coordination is seen as a non-economical option.

Function, Phenomena, Form Operator

Compiler research technology could be used for transformation of database contents. Serialization of data could also be a useful technique.

The Ideal Final Result Operator

All data bases shall have an automatic synchronisation mechanism that synchronizes database contents between databases of incompatible formats, instantaneously.

Trends of Evolution/Prediction Tree Operator

The standard solutions and trends of evolution could not be applied on our software problem, because the method is not adapted for software problems. By instead using the evolution patterns from this paper, see Appendix C – Software Evolution Laws; it is possible to predict possible evolution. Potential evolution predictions describe

- Evolution in stages:
  The import/export function is not expected to do any leaps in evolution stage, for the moment.
- Evolution towards increase ideality:
  Increasing ideality would mean that a data-fusion and connectivity feature was included in the database itself, thus eliminating the need for import and export functions.
- Non-uniform evolution of system elements:
  One can expect that the import function will not evolve at the same pace as the export function.
- Evolution towards increased dynamism and controllability:
  It will most likely be possible to build an export only, an import only version of the software. The software will have multiple application programmer interfaces, maybe even an SNMP interface.
Evolution towards increased complexity, then simplification: The complexity of the export/import function will increase as new information structures will be added. Later the system will become less complex, when it becomes necessary to introduce a database-to-XML component to support the import/export function.

Evolution towards matching and mismatching components: The import and export functions are already mismatching components.

Evolution towards decreased human involvement: Automatic import and export can be introduced as a side effect, when the service is needed by another software.

Law of transition to micro-level: The syntax description of each information element to be translated, may have its own specification, translation rule, and trigger describing when the information is to be transformed.

Law of increasing flexibility: The translation work can be made incrementally. And information that changes in the database, could immediately be translated and inserted in the target information structure, thus increasing performance and data translation efficiency.

Law of transition to a higher level system: The general purpose import/export feature described is a candidate for a new standard component among database engine developers and software houses.

Law of shortening of data flow and transformation path: This means that translation of data should not have to pass lots of new information protocols, like XML, bulk copy formats, and similar. A direct bridge between different data sources will be available, allowing us to compile an optimised synchronization gateway between different database and information system providers.

Alternative Systems Operator

In other engineering fields, e.g. electrical power systems use a transformer to convert electricity with one characteristic to another. Car manufacturers use a gearbox to convert transmission forces and rotation. So, the software’s user interface could be inspired by a gear box, or a power supply network.

Application of Technical Function Operators

The above mentioned predictions could be further combined and analysed to identify which of the evolution paths that would be the most fruitful and prosperous evolution. As this requires complete analysis of the software, this step has been omitted.

Still one could predict that the simple import and export function that we started with, will change to include features like:

- Flexible deployment of export, import, or both functions for one hardware node.
- Incremental translation of information back and fourth to our database, where only affected information structures have to be recalculated.
- Multiple application programmer interfaces will be introduced to allow the function to spread.
- A translation scheme that describes how to convert the information during export, and import will be introduced.
- Functions for data synchronization, i.e. data fusion, of databases will become a commodity in commercial databases.

9.3 Phase 3: Planning and Implementation

When the forecasted software system evolution is described, it is necessary to lay out time plans for the introduction of the features described. Market situation and economic aspects plays a significant role in this. The time plan needs to cover both short term, as well as long term planning of product family roadmaps.

During implementation, many new systems conflicts will have to be solved to enable a transition to the next forecasted innovation level. Both TRIZ and its conflict resolution method ARIZ could be useful tools during the realization of forecasted software product evolution.