Team Understanding

A Successor of the Process Improvement Paradigm?

Bengt Lennartsson, BLn@IDA.LiU.se
Department of Computer and Information Science
Linköpings Universitet, S-581 83 LINKÖPING, Sweden

Ulf Cederling, Ulf.Cederling@masda.hv.se
Department of Mathematics, Statistics, and Computer Science
Växjö University, S-351 95 VÄXJÖ, Sweden

Abstract: This paper is based on studies of industrial development projects over a period of more than a decade. The findings are in terms of recognizable patterns rather than in terms of numbers. The most important conclusion is that understanding the application domain problem is one of the most important success factors in the development of complex systems. The next conclusion is that, in most cases, knowledge and experience from many disciplines will be needed to find an acceptable solution. So, solving the application domain problem will require team understanding and team learning. It is described how the Team Learning Paradigm for system development can be built from experience in other fields, and how methods and ideas from different disciplines can be combined to a new model for system development.
Section 2 presents the contributions from the areas of System Architecture and System Development Models. Section 3 compares the Incremental Improvement paradigm with the Technology Shift phenomenon; the *Kaizen* and the *Kairyo* strategies in Japanese. In Section 4 the general requirements on education in society are discussed and also how these are related to system development. In Section 5 the pieces are put together to the Team Learning Strategy for system design and development. Related work in Section 6, and Conclusions in Section 7 complete the paper. As ideas and experience from many different areas have been used, there are almost 30 references.

Keywords: Team learning, system development, system development model, system architecture, software engineering, process improvement, quality assurance.

1. Introduction

The conclusions and observations presented in this paper are based on studies of many industrial development projects over a period of a couple of decades. The findings are mainly of a qualitative character, where the empirical evidence is in terms of recognizable patterns. The conclusions can be verified (or falsified) informally, as they are presented to experienced and skilled people and compared to their own conclusions. One impression is, without giving any numbers even here, that a majority of the system architects and designers in industry have reached very similar conclusions, while academic people tend to make observations supporting the benefits of their own theory, their own development model or method, or their own set of tools.

In most development projects there are many success factors. In order to reach the goal you have to understand the problem **and** know and organize your resources **and** make an appropriate choice of methods and tools **and**, in the long run, improve your skills and your development process. The scale eventually determining *Failure* or *Success* will in most cases in particular consider your weakest points; a weakness in one feature can not normally be compensated by excellence in another area. *A chain is only as strong as its weakest link*. So, while this paper focuses on just one aspect of system development, team learning and understanding, it does not claim that the other aspects are unimportant. **All** capabilities are required. However, some of the others have been in focus for several years, while the team learning aspect has, in general, been neglected. Hence this paper aims at achieving a better balance in the interest in the different success factors involved in complex system design and development.

Another implicit message in the paper is that most of the characteristics of the design and development of complex *software* systems are valid for complex system development in general. The software design and VLSI design areas, for instance, have much to learn from each other, and both would benefit from being regarded as the same engineering discipline: system design.

1.1. Research methods

There has been some demand, see Basili [1], Tichy [2], and Fenton [3], on more empirical research in software engineering. The most common interpretation of *empirical*, however, has been in terms of controlled experiments where you can assign numbers to the quantities. However, a wider scope for the empirical observations has also been discussed, *e.g.* Adrion [4], and Glass [5]. In many other disciplines purely qualitative empirical methods are common and well established [6].

The conclusions in this paper are based on observations of many kinds of industrial system development projects. Sometimes the observations have been by means of case studies, that is by mere observation and analysis of the activities, the documents, and other material, by interviewing key persons in the project, etc. This type of research is purely observational; you do not as a researcher introduce new methods, models, or tools to be used in the project for test and verification. In large projects the observational approach is normally the only method possible. You are not allowed to perform any controlled experiments or even influence the selection of methods or tools, because the projects are too big and too important. It is out of question to replicate activities and compare results. It may be possible to study some of the issues in lab situations. However, the large-scale aspects can not. Problems due to system size or to the complexity of the system or the development organization are not scalable. In disci-

plines like astronomy, weather forecasting, national and global economy it has been widely accepted for long time that controlled full scale experiments are impossible. The research on models, methods and tools for the development of complex systems may be of the same character.

Another method is the action research approach. Here you do not perform any controlled experiments either, but you have managed to convince the project leaders and the project management to use some new method, tool, organization, etc., and your objective is to study the consequences of the introduction of this new component. The conclusions about the team learning aspects of system design and development in this paper are based on an action research effort, where the Problem-Based Learning model has been used in the development of a training program for industrial system design engineers.

In science and engineering it is sometimes difficult to achieve acceptance and appreciation for empirical research where you can not specify your observations in numbers. Nevertheless, you can make relevant interesting observations, define and test models and hypotheses, and develop and verify/falsify theories in general. Maybe the most important results in science are of a qualitative character. Charles Darwin's work [7] is an excellent example.

1.2. Outline of the rest of the paper

This paper presents how methods and ideas from different disciplines can be combined to a new model for system development. Section 2 presents the contributions from the areas of System Architecture and System Development Models. Section 3 compares the Incremental Improvement paradigm with the Technology Shift phenomenon; the *Kaizen* and the *Kairyo* strategies in Japanese. In Section 4 the general requirements on education in the society are discussed and also how these are related to system development. In Section 5 the pieces are put together to the Team Learning Strategy for system design and development. Related work in Section 6, and Conclusions in Section 7 complete the paper. As ideas and experience from many different areas have been used, there are almost 30 references.

2. System Architectures and System Development Models

The means for handling complexity in system development projects are *System Architecture* regarding the system to be developed, and *Development Model* for the organization and management of the development task. In both cases the divide-and conquer strategy is applied. The architecture presents the complete system as a number of components and subsystems, where each component can be considered separately to some degree [8]. In the same way, the development model divides the total development effort into separate phases or activities [9]. It is outside the scope of this paper to give a survey of the two areas. The references do, however, and they contain in turn a number of references.

The point to make here is that both the architectural approach and the development model view primarily consider *the deliverables:* the architecture deals with the technical dependences and relations between the components to be delivered, and the usual development model with the management of the deliverables and of the process where they are ”produced”. In both cases there is a *product-oriented* view. The task is to *pro-*

duce the deliverables: the requirements and design documents, the code modules and configurations, the test specifications, the user and maintenance documents, etc.

Of course a system architecture and a development model are necessary, but the question is whether they, in their traditional sense, really are the cornerstones of system development. Are they alone the most important success factors?

3. Incremental Improvements versus Technology Shifts

The architectural and the development models are product oriented. However, ”it is *the process* where the deliverables are produced that is important”, it is often claimed by people from the process improvement and quality management communities. ”The means to improve the overall result is to improve this process.”

There are two mechanisms for improvements (in terms of quality, productivity, functionality, return on investments, etc.). The continuous paradigm: the Japanese *Kaizen* (improvement) philosophy [10], the quality assurance strategy in the same spirit TQM [11], and the process centered views CMM [12], [13], ISO 9000[14], etc., where in a systematic way you define the roles and responsibilities, learn from your own experience, and make (small) improvements in all the components of your business. That is, you are taking small steps within the current structure and the current technology.

The rest of the world tried to copy the success in the Japanese photo, car, and consumer electronics industry in the seventies and eighties. The *Kaizen* philosophy has an impact as illustrated in Figure 1(a).

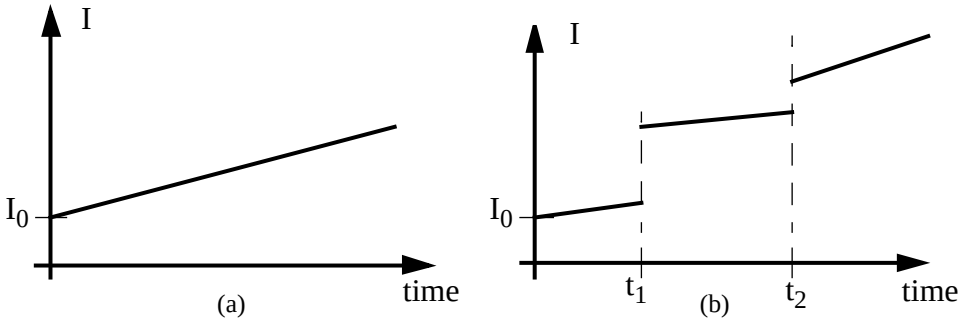

Figure 1 Performance index, I, of some kind, versus time.
(a) for continuous improvements, *Kaizen*, and
(b) with discontinuities, *e.g.* due to technology shifts, *Kairyo*, at times t_1 and t_2.

However, there are also other mechanisms and strategies for improvements. Instead of taking small steps within the existing structure you can make more drastic changes, *Kairyo*, Figure 1(b). You can switch to a new technology, a new design or assembly method, a new organization of the work, or a new team with better skills and knowledge. Quite often such new technologies have been developed and exploited in new high-tech companies. Examples of such shifts are: from vacuum tubes to semiconductors, from mechanical to electronics-based calculators, from discrete components to (V)LSI technology, from LP to CD records, from assembly level programming to application oriented modeling tools.

If you build your strategy upon the continuous improvement model, Figure 1(a), you will achieve robustness and stability within your current structure and organiza-

tion, but you may lose flexibility with respect to new technologies. On the other hand, if you focus on readiness to adopt new technologies, you may pay by not detecting possible improvement within the current structure. To quote from Druffel [15], former director of the Software Engineering Institute:

> *As we progress in our understanding of our process and evolve measures, let us not forget a lesson learned from improvement efforts in other businesses. A narrow focus on continuous process improvement can make us oblivious to fundamental changes that would yield enormous improvement. We must ensure that measurement does not impede acceptance of paradigm shifts the research community proffers.*

To balance the two strategies of robustness and improvements within the current framework on one hand, and flexibility and ability to exploit new technologies on the other, is one of the most important strategic decisions within an organization.

However, both the quality management movement and the process improvement one are in principle production oriented. They have inherited the models and the concepts mainly from the manufacturing and assembly plant environments. Design and development are viewed as rather mechanical activities controlled by rules, guidelines, and handbooks. Concepts like problem understanding and creativity are not in focus.

4. The New Abilities Required

Let us just add one more component before we put the pieces together in the next section. It is a very general observation that a professional education today does, in general, not last for the of one's whole life. Regardless of profession you need to learn new concepts, new methods, how to use new tools, etc., several times during your career. This applies in particular to engineering. The situation for a person graduating from an engineering program today is quite different from the situation a generation ago. In short, the new situation implies:

1. ABILITY TO LEARN is more important than static knowledge about facts.
2. ABILITY TO COMMUNICATE AND COOPERATE are more important than individual brilliance.
3. ABILITY TO UNDERSTAND TOTALITY is more important than narrow knowledge about details.
4. ABILITY TO ACT AND ASK QUESTIONS is more important than ability to follow detailed instructions.

Just a brief motivation for each point:

1. As everything, especially the technology, changes very fast, it is no longer possible in an organization to rely upon a central function for planning and executing continuing education and training for the engineering staff - the time delay from the detection of the need to the execution would simply be to long. Hence the responsibility to follow the trends and to identify the actual learning needs must be given to the individual design engineer or to the design teams.
2. Today almost every design task is such that knowledge and experience from many different areas are needed to find an acceptable solution. One individual engineer, regardless how smart, can not find the solution him/herself.

3. The integration of material from different domains is generally needed. In the design team there must be persons able to bridge the gaps between the different disciplines.
4. The traditional model where instructions are propagated downwards in the development organization is today inadequate for two reasons. Firstly, the understanding of the problem to solve evolves in parallel with the design. That is, nobody is able to write in advance design specifications that will survive. Secondly, even if there were people able to grasp and describe everything initially, many things would change during the course of the work. The demand on flexibility will require design and development engineers thinking by themselves and asking questions when needed, rather than just reading possibly outdated documents.

The new requirements on abilities are one of the reasons why Problem-Based Learning [16] and Project-Organized Curricula [17] are getting more and more attention [18], at schools as well as at universities. The new requirements on skills and behavior patterns on the design and development engineers in industry are following the same trend as the general demand in the society.

The list of ”abilities” is in terms of attitudes and behavior rather than about knowledge about facts. The attitudes required are along the same line as you can read in the best-seller by Covey [19]. In order to change attitudes and behavior patterns, however, other means than traditional lectures and classes are needed. Team learning is such an approach.

5. Team Learning as a Strategy for Design and Development

”What determines whether a complex development project will succeed or not is whether it is possible to establish an understanding and a shared opinion about what to do and how in the initial group of persons defining the project. The rest is often fairly straight forward”, Schefström [20].

The importance of the understanding in the very first phase is also verified by others. *”Most safety-critical software errors can be traced to errors in the requirements - that is, to misunderstandings about what the software should do”*, Leveson [21].

So, there is some evidence that the development of an understanding of what to do may be equally important as the ability to implement something once it is specified. This is not surprising if you accept that system development is an intellectual design activity rather than a mechanical production task. This understanding will in most cases take place at the team level. The situation, or the problem, is so complex that knowledge, experiences, and skills from many different areas, will be needed for its solution. Successful system design and development require successful team learning and team understanding. The problem solving capability exists when an appropriate team is working together in the development of the shared understanding.

At the Center for Organizational Learning, MIT Sloan School of Management, team learning in industrial projects has been studied for a long time. In [22] and [23] Senge presents his five disciplines: ***Systems thinking***, ***Personal mastery***, ***Mental models***, ***Building shared visions***, and ***Team learning***. He claims that all five, as an ensemble, are required. The team understanding is also a key component in the new system development model at Microsoft [24]. The team aspects have gained some attention in

the general management and quality engineering literature; see for instance Smith and Reinertsen [25].

In a complex system development project there are many teams, many problems, and many design alternatives. However, for each activity, there are always two steps: the mental one to understand what to do, and the production one to package the result of your understanding so that it is made available to other parties; see Figure 2.

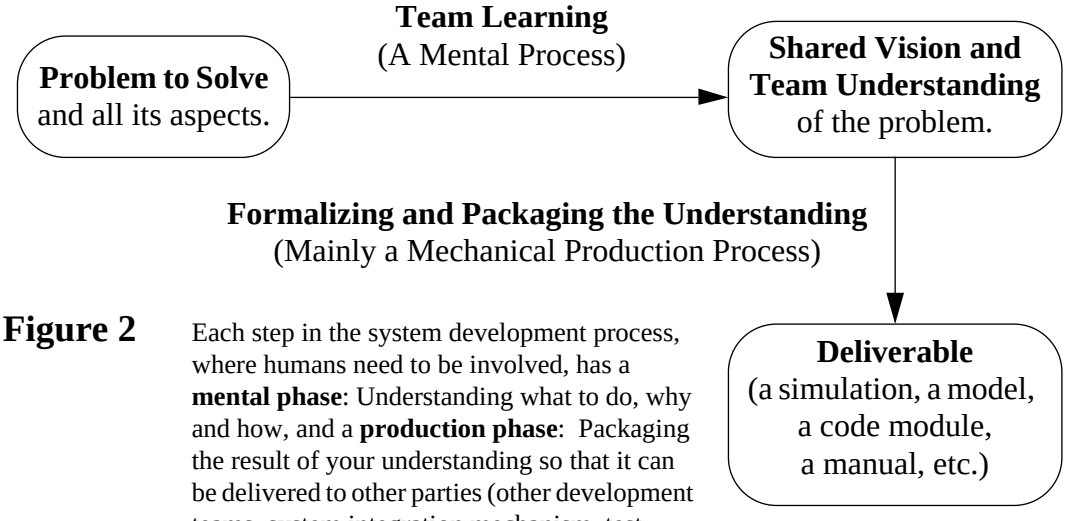

Figure 2 Each step in the system development process, where humans need to be involved, has a **mental phase**: Understanding what to do, why and how, and a **production phase**: Packaging the result of your understanding so that it can be delivered to other parties (other development teams, system integration mechanism, test facilities, customers, etc.)

In the team understanding paradigm for system design and development, the team learning is put into the center and made the main concern. The question is how to apply the paradigm to the development of complex systems in the real world. The total development effort must be distributed to many different teams. So, there are now three immediate questions: How big is a team? How do you manage and coordinate many teams? How do you initiate and support the team learning process?

5.1. Team Size

The question of optimal team size has been studied in many contexts, and a presentation of all these aspects is outside the scope of this paper. Just two remarks:

- Initial disagreement and various backgrounds in a team will add tension and new dimensions to the shared understanding. Agreement and similar backgrounds within the team will narrow the scope. With less than five persons in a team there will not be a sufficient amount of ”interface dynamite”.
- A team acts as a team only if there are established relations between each pair of persons in the team. With more than eight persons in the team it will be too expensive to establish and maintain all these relations. Eight is an upper limit.

In each design and development team there should be five to seven persons, if necessary up to eight can be acceptable. The exact limits are not important here, just the implication that in every complex real world development project you need to have many teams, and the obvious question is now: How to coordinate and assign tasks to all these teams.

5.2. Team Coordination

One simple way to initiate a large project is to define a number of subtasks, specify their contents, and assign people to each one. Thus you have an initial structure of the project organization. Are there really any alternatives to this hierarchical distribution of work and responsibility? The contract between the customer and the vendor must contain definitions of what to deliver and when. The allocation of resources to the different units must be based on specifications of the work to be done.

The question of team coordination manifests the difficulty with the introduction of the new model. Implicit in the team understanding paradigm is the assumption that the team input is not a formal unambiguous document defining exactly what to do. The definition of the relations and interfaces to other teams is part of the task; it has not been possible to do this in advance. Hence, some method other than the tradition hierarchical decomposition is needed for the coordination of the total effort.

The old model works in the case where you already have the knowledge needed to do the decomposition into subtasks. If you have developed similar systems before, this can be possible, provided you develop the new one in a similar way. However, if technology shifts are involved, learning and new understanding will be necessary. As learning, in the sense of finding solutions to complex problems, is best organized in teams of five to eight persons, you may have to start up many parallel teams to be able to gather the understanding needed in time. So, you have to distribute responsibility for the learning instead of for the development among the teams.

The understanding has to be coordinated and integrated. Tools for modeling, simulation and animation are excellent vehicles for communication, within as well as between the teams, of ideas about intended system behavior and system structure. Such tools may take the part of the roles requirements engineering and formal specifications have in the traditional model. In later steps, where the implementation of the shared vision established has started, there must be means for the integration of the contributions from the different teams.

One such approach is the *Daily Build Strategy* introduced within Microsoft [24]. The idea is that each team, five implementors, five testers and one coach, has the task of implementing one specific feature in the next release of a product. Each day a new internal version of the product is built, and at least once or twice a week each team has to submit their component for integration. This way the evolving interfaces and interactions are tested and verified incrementally, as well as the behavior of the total system. Exactly the same strategy, for exactly the same reason, was successfully used in a smaller scale for a group of 30-50 engineers within Satt Control AB in the eighties [26].

5.3. Team Learning Support

There are two main sources of experience in the team learning area. One is from using groups or teams in ordinary university curricula, and the other is the use of teams in industrial training programs. In the first case there are different models of how the team is used.

In the Problem-Based Learning (PBL) model [16], in particular as it has been executed in health care programs, there are variants of the operational guidelines with seven or nine steps in the cycle. The starting point is always ”a realistic situation” used

for brain-storming and associations. After the brainstorming there is structuring, definition of problems, and definition of learning needs. Then there is a phase with literature studies, etc. That is, in the PBL method in its normal setting, it is assumed that the knowledge exists and is available *e.g.* in the library. The initial realistic situation is designed by the teaching staff for its educational purpose, and the students are not required to eventually ”solve the problem”. The examination is also normally taken individually according to the knowledge goals defined for the period. The student group is called ’tutorial group’. In the Project Model [17], the students have to solve or investigate a real situation, and the task for the team is to develop and present a solution. It is normally up to the students to organize the work and to make assignments for the members of the project group. For both PBL and Project Organized programs, there are many variants. However, they share the basic view: learning and not teaching is in focus, the use of the group in the learning process, and problems in the real world are starting points for the learning.

There is also experience from industrial training. In particular Senge’s model [23] can be applied directly. We have used a combination of PBL and the Senge model for the training of design engineers in industry [18]. Initially the PBL Cases, or the Microworlds in Senge’s terminology, can be designed for training purposes, but rather soon the Cases or the Microworlds can be unified with the real situation, and thus the model is applied to the system development task itself and not to artificial training problems.

The basic PBL idea is that you first try to understand the problem, then look around for solutions. Hence, if you are considering introduction of the team learning strategy into your organization you have to understand the needs first, and then you can bake your own pie from the ingredients readily available. Senge’s Fieldbook [23] is an invaluable source to dig into at this stage. The team is not just the optimal mechanism for problem solving. It is also the cornerstone of organizational learning. Senge:

> *It can’t be stressed too much that team learning is a* ***team skill****. A group of talented individual learners will not necessarily produce a learning team, any more than a group of talented athletes will produce a great sports team. Learning teams learn how to learn together. Unless the teams can learn, the organization can’t.*

6. Related Work

The idea that learning and understanding are important components in system development is not new. However, in general the implications on the system development model are neglected even if there are exceptions.

The organizational learning is one objective behind Boehm’s spiral model [9]. However, it is about collections of data rather than about learning and understanding at the team level. The Experience Factory at the Software Engineering Laboratory developed by Basili and others [27] is definitely a model for learning. However, the responsibilities for system development on one the hand and the responsibility for collection of data and experiences on the other have been separated. Hence, the learning achieved in one project can normally not be applied until later projects. The Experience Factory model has been used successfully not only for incremental process improvements, but also for the evaluation of technology shifts, Figure 1(b).

At the Software Engineering Institute the Capability Maturity Model (CMM) effort has received much attention [12]. CMM is very organization and production oriented. Understanding and learning are based on collection of data from the ”production process”, and not as a mental activity aiming at better understanding of the application domain problems. In the follow-up, the Personal Software Process (PSP) is defined [28]. Here learning in the software domain is the main concern, and only at the individual level.

There are approaches to product development in general with relevance to the ideas in this paper. Concurrent engineering and the shortening of development time in general [25] mean releasing the constraints available in the waterfall model. In particular the Wavefront Method [29] has been looked upon in relation to the team understanding approach.

The study at Microsoft by Cusumano and Selby [24] has already been mentioned. Few organizations can directly copy the Microsoft model. However, we can all learn something from it. The experiences from Senge and others at MIT Learning Center are highly relevant for anyone interested in system design and development in general.

7. Conclusions and Future Work

So, the main conclusion from many development projects in industry is that the major difficulty is to achieve a shared vision and a general understanding of what to do, why, and how. The development of this shared vision and understanding has to take place in appropriately composed teams of, ideally, five to seven persons. This understanding of the application domain problem has very little overlap with the learning mentioned in process improvement literature, where you only focus on tuning “the production plant”, *i.e.* the parameters of the development organization.

The development of mechanisms and support for this team understanding is one of the most important success factors in an organization. *The rate at which organizations learn may become the only sustainable source of competitive advantage*. Ray Stata of Analog Devices, in [22].

Senge’s statement that it is just at the team level the learning can take place is difficult to verify. However, our observations are not in conflict with this statement. In our ordinary university curricula the group approach is beneficial for most of the students, but there are a few who prefer to learn individually. In a complex development project there may be room and need for deep technical specialists in some cases, even if they do not feel comfortable in the team discussions.

In the future we will try to integrate the team learning approach better with ideas about system architecture, where the system architecture is regarded as a framework where the different contributions of shared visions and team understandings can be communicated in the organization and reused in future system families [30]. The initial understanding constitutes the backbone of the architecture, and as further learning takes place, the architecture evolves and becomes more and more decorated. However, it would be a mistake to think of the architecture in this sense as a data structure or as a repository, where everything is formalized and can be retrieved when needed. The application domain understanding resides in the minds of those who have developed it. The communication of this understanding to other teams or other persons is a social process and not just a transfer of data or the delivery of a document.

The main difficulty when applying the Team Understanding Model in complex and large development projects is the coordination and integration of the partially autonomous activities. The model from Microsoft works when you are building new releases of existing systems, and the same model was useful at Satt Control in a smaller scale, where it was rather easy for the system architect to communicate his initial vision to the rest of the team. CASE studies of similar processes in other organizations would be of great value.

In our experience, there are very important national and cultural differences in the attitudes to the use and the coordination of autonomous teams. A team aiming at developing an understanding of such aspects needs participants from many different disciplines indeed.

8. Acknowledgments

The list of all very helpful persons in industry is too long to be included, so just a few will be mentioned explicitly. Thanks to Roland Ekinge and others within Whirlpool, we have been able to understand how the Team Learning ideas and the Problem-Based Learning Model can be combined into successful training programs in an industrial environment. In the implementation this combination Kristina Davidson's contributions have been very important. Ulf Olsson and others at CelsiusTech Systems have given us access to all their experience from the development of the architecture for their ship system family. Hilding Elmqvist *et. al.* at Satt Control in the eighties have shared with us their experience from the Daily Build strategy. Göran Östlund at the Association of Swedish Engineering Industries has for many years been a very stimulating and supportive discussion partner.

Jesper Andersson has suggested many improvements and given valuable remarks, and Ivan Rankin has contributed by proofreading our English.

The study at CelsiusTech Systems has been supported by NUTEK under contract P1339-2.

References:

[1] Victor R. Basili, Richard W. Selby, David H. Hutchens: Experimentation in Software Engineering. *IEEE Transaction on Software Engineering*. Vol. SE-12. No 7. July 1986. pp 733-743.

[2] Walter Tichy *et. al.*: Experimental Evaluation in Computer Science: A Quantitative Study. *Journal of Software and Systems*. Jan. 1995.

[3] Norman Fenton, Robert L. Glass: Science and Substance: A Challenge to Software Engineers. *IEEE Software*. Vol. 11. No 4. July 1994. pp 86-95.

[4] W.R. Adrion: Research Methodology in Software Engineering. Summary of the Dagstuhl Workshop on Future Directions in Software Engineering. *Software Eng. Notes*. Vol. 18. No 1. Jan. 1993. pp 36-37.

[5] Robert L. Glass: The Software Research Crisis. *IEEE Software*. Vol. 11. No 6. Nov. 1994. pp 42-49.

[6] Robert K. Yin: *Case Study Research - Design and Methods*. Sage Publications, Inc. Newbury Par, CA, 1989. ISBN 0-8039-3471-8.

[7] Charles Darwin: *Origin of the Species by Means of Natural Selection, or the Preservations of Favoured Races in the Struggle for Life*. 1859.

[8] *IEEE Software*. Special Issue on Software Architecture. Vol. 12. No 6. Nov. 1995.

[9] Barry Boehm: A Spiral Model of Software Development and Enhancement. *Computer.* Vol. 21. No 5. May 1988. pp. 61-72.

[10] Masaaki. Imai: *Kaizen: The Key to Japan's Competetive Success*. McGraw-Hill Publishing Company. New York. N.Y. 1986.

[11] Joseph M. Juran: *Juran on Quality by Design: the new steps for planning quality into goods and services*. The Free Press 1992. ISBN 0-02-916683-7.

[12] Marc C. Paulk *et. al.*: Capability Maturity Model, Version 1.1. *IEEE Software*. Vol. 10. No 4. July 1993. pp 18-27.

[13] Mark C. Paulk *et.al.* eds: *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley Publishing Company 1995. The SEI Series in Software Engineering. ISBN 0-201-54664-7.

[14] Östen Oskarsson, Robert L. Glass: *An ISO 9000 Approach to Building Quality Software*. PrenticeHall, 1996. ISBN 0-13-228925-3.

[15] Larry Druffel: Professionalism and the Software Business. *IEEE Software*. Vol. 11. No 4. July 1994. Page 8.

[16] Mark A. Albanese, Susan Mitchell: Problem-based Learning: A Review of Literature on Its Outcomes and Implementation Issues. *Academic Medicine*. Vol. 68. No 1. Jan. 1993. pp 52-81.

[17] Finn Kjersdam, Stig Enemark: *The Aalborg Experiment - Project Innovation in University Education*. Aalborg University Press 1994. ISBN 87-7307-480-2.

[18] Bengt Lennartsson, Kristina Davidson: Experiences from Direct Application of the PBL Method in Industrial Training and Organization Development. *Proceedings of The University of Maastricht's 20th Anniversary Conference: Placing the Student at the Centre*. Maastricht, The Netherlands, Nov. 20-22, 1996.

[19] Stephen R. Covey: *The 7 Habits of highly effective people*. Simon & Schuster, 1992. ISBN 0-671-71117-2.

[20] Dick Schefström. Luleå University of Technology, Sweden. Personal communication 1991.

[21] Nancy Leveson: *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company. 1995. ISBN 0-201-11972-2.

[22] Peter M. Senge: *The Fifth Discipline - The Art & Practice of The Learning Organization*. Currency Doubleday, 1990. ISBN 0-385-26095-4.

[23] Peter M. Senge: *The Fifth Discipline Fieldbook - Strategies for Building a Learning Organization*. Currency Doubleday, 1994. ISBN 0-385-47256-0

[24] Michael A. Cusumano, Richard W. Selby: *Microsoft Secrets - How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. The FREE Press 1995. ISBN 0-02-874048-3.

[25] Preston G. Smith, Donald G. Reinertsen: *Developing Products in Half the Time*. Van Nostrand Reinhold 1991. ISBN 0-442-00243-2.

[26] Bengt Lennartsson: *Software System Design - experiences from projects in Swedish industry (In Swedish)*. Mekanresultat 88005, November 1988.
ISBN 91-524-0986-4.

[27] Victor Basili, Scott Green: Software Process Evolution at the SEL. *IEEE Software*. July 1994. pp 58-66.

[28] Watts S. Humphrey: Using a Defined and Measured Personal Software Process. *IEEE Software*. May 1996. pp 77-88.

[29] Lars Philipson, Anders Ardö, Kenny Ranerup: Design of a Microprocessor and its Software as a Seven Week Class Project. In *Proceedings of the 1989 VLSI Education Conference and Exposition*. Santa Clara, Clifornia. July 1989. pp 3-12.

[30] Ulf Cederling, Bengt Lennartsson: A System Family of Integrated, Distributed Embedded System Products and its Architecture. *Proceedings of the International Workshop on Development and Evolution of Software Architectures for Product Families*. Madrid, Spain, Nov. 18-19, 1996.