On the System of Systems Approach to the Development of Everyday Life Applications

Mika Myller

Helsinki 15th September 2005

Master of Science Thesis

UNIVERSITY OF HELSINKI

Department of Computer Science

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section Faculty of Science	Laitos – Institution – Department Department of Computer Science

Tekijä – Författare – Author
Mika Myller

Työn nimi – Arbetets titel – Title
On the System of Systems Approach to the Development of Everyday Life Applications

Oppiaine – Läroämne – Subject
Computer Science

Työn laji – Arbetets art – Level Master of Science Thesis	Aika – Datum – Month and year September 15, 2005	Sivumäärä – Sidoantal – Number of pages 91

Tiivistelmä – Referat – Abstract

This research project studies compatibility of *system of systems engineering methodologies* with the ARKI research group's proposal for co-design philosophy and approach and their applicability to the engineering of the application development environment of the ARKI research group.

Based on the literature and observations this thesis shows that the application development environment of the ARKI research group can be considered as a system of systems having the characteristics of complex systems. Thereby system of systems engineering methodology, complex systems engineering, is suggested preferable to other approaches to engineer the application development environment of the ARKI research group. In addition compatibility of complex systems engineering with the ARKI research group's proposal for co-design is justified. Further, this research suggests that the complex systems engineering approach used in combination with the practice-centered co-design approach might benefit in general the development of everyday life applications.

The thesis has been produced under the umbrella of the ARKI research group, at the Media Lab of the University of Art and Design Helsinki.

ACM Computing Classification System (1998): C.5.0 [Computer System Implementation]: General; D.2.9 [Management]: Programming teams, Software development models; D.2.11 [Software Architectures]: Domain-specific architectures; H.1.1 [Systems and Information Theory]: General systems theory; H.1.2 [User/Machine Systems]: Human factors; H.4.3 [Communication Applications]: Bulletin boards; J.7 [Computers in Other Systems]: Consumer products; General terms: Design, Experimentation

Avainsanat – Nyckelord
co-design, complex systems, everyday life, systems engineering, system of systems, usability

Säilytyspaikka – Förvaringställe – Where deposited
Library of the Department of Computer Science C-2005-

Muita tietoja – Övriga uppgifter – Additional information

Acknowledgements

I especially would like to thank Kari-Hans Kommonen, research group leader and my supervisor, for taking me under the umbrella of the ARKI research group and giving me the grand opportunity to study this interesting subject and trusting that I will manage to pull this through. Without his inspiring visions and encouragement I would never have come up with this subject or managed to write a thesis about it.

I thank my supervisors at the University of Helsinki, Inkeri Verkamo and Juha Gustafsson, for their patience with my ”forever chancing research subject”, and efforts in pulling me from “philosophical spheres” toward the more solid ground of computer science. They are not to blame if my feet are flying in the air and head is still in clouds…

I thank all the members of the ARKI research group, especially Iina Oilinki, Taina Rajanti and Kirsti Lehtimäki for commenting and helping me with English, and Elizabeth Bly for proofreading my thesis. Everything that is correct came from them; all the mistakes are mine.

I thank Miguel Lara Alday for good laughs and support through some difficult times, which took place at the same time with this project.

I thank Anna Kurvinen for understanding and backing me while I was finishing this thesis.

Finally, the greatest thanks go to my daughter Iida for reminding me what is truly important in life and keeping me sane through this frantic episode.

At the end (and for the beginning), I want to cite two authors to express my gratitude for the words that lead my efforts:

> "To define is to kill. To suggest is to create." – Stéphane Mallarmé
>
> "Everything is simpler than you think and at the same time more complex than you imagine." – Johann Wolfgang von Goethe

Contents

1 INTRODUCTION 1

2 OVERVIEW OF CO-DESIGN 4

2.1 CO-DESIGN – A DESIGN STRATEGY FOR A DIGITAL SOCIETY 4
2.1.1 Design 4
2.1.2 Rationale for Co-design 5
2.1.3 Co-design Approach 7
2.1.4 Building Blocks and Applications 8
2.2 CO-DESIGN AND META-DESIGN 11

3 ENGINEERING SYSTEMS OF SYSTEMS 15

3.1 SYSTEMS, SYSTEMS OF SYSTEMS AND COMPLEX SYSTEMS 15
3.1.1 Complexity, Emergence and Complex Systems 19
3.2 TRADITIONAL SYSTEMS ENGINEERING 24
3.2.1 The Practice of the Traditional Systems Engineering 24
3.2.2 Boundary Conditions for Traditional Systems Engineering 25
3.3 AGILE METHODOLOGIES 29
3.4 SYSTEM OF SYSTEMS ENGINEERING 32
3.4.1 Implications for System of Systems Engineering 33
3.4.2 Proposal for a Methodology 34
3.5 COMPLEX SYSTEMS ENGINEERING 36
3.5.1 Complex Systems Engineering Methodology 38
3.5.2 Tactics for Complex Systems Engineering 47
3.6 COMPARISON OF SYSTEMS ENGINEERING APPROACHES 54

4 APPLICATION DEVELOPMENT ENVIRONMENT OF ARKI 56

4.1 PRACTICAL APPLICATION DEVELOPMENT CASE: VOICE NOTES 56
4.1.1 The Architecture of Voice Notes 57
4.1.2 Voice Notes Application as a System of Systems 58
4.2 CHARACTERISTICS OF APPLICATION DEVELOPMENT IN ARKI 62
4.3 APPLICABILITY OF SYSTEMS ENGINEERING METHODOLOGIES 67
4.3.1 Applicability of Traditional Systems Engineering 68
4.3.2 Preference of System of Systems Engineering 69
4.4 PREFERENCE OF SYSTEM OF SYSTEMS METHODOLOGIES 70
4.4.1 Applying Complex Systems Engineering Tactics 71
4.4.2 Implications for Application Developers 75
4.5 SUMMARY 77

5 DISCUSSION 79

6 CONCLUSIONS 84

REFERENCES 86

APPENDICES

APPENDIX 1 ZOPE – WEB APPLICATION SERVER

1 Introduction

In everyday life people face diverse digital products and applications: online booking systems, online banks, online stores, emails, word processors, instant messengers, file managers, mobile phones, digital calendars, digital cameras... The usability of these products and applications is traditionally approached through an HCI (human computer interaction) stance, where an example situation is "a person using an interactive graphics program" [ACM92]. When the best of breeds of the products mentioned above are examined in the example HCI situation, they can be considered usable. However, integration and interoperability difficulties that people encounter when trying to use a set of these products to achieve their various goals, suggest that there is need for a more comprehensive approach in order to improve the usability of these products.

We in the ARKI research group at the Media Lab of the University of Art and Design Helsinki have approached this challenge through a *co-design* philosophy. We aim at elaborating a dynamic design methodology, where the multitude of surrounding products and systems are taken into account and designed to be an open system that is continuously adapted to the practices of everyday life uses. During our research we have implemented and intend to implement several applications in order to experiment with our concepts. Gradually we have begun to consider that we should more consciously engineer our own application development environment toward an open and adaptable system that better meets our co-design goals. The problem that has emerged is how one should approach engineering such a system.

The challenge is that the *Application Development Environment of the ARKI research group* (ADEA) is composed of our and third parties' systems (products and applications) that are built more or less independent development tracks. Furthermore, the ADEA includes potentially all possible systems that are found interesting at some phase of the research. Thus, it is reasonable to argue that rather than a single system the ADEA is a *system of systems*: a system that is composed of independent systems that are managed separately [GDM05]. Or, even a *complex system*: a system of system having characteristics of an *ecosystem*[1] [NoK04]. The literature presents approaches to the engineering and management of large complex systems of systems such as air and space operation centers and port security systems (see Bar03, CoK03, Kea03, NoK04) but applicability of these approaches to engineering small systems of systems, such as the ADEA, has not been studied.

This research set out to study the applicability of the system of systems engineering approaches to engineering the ADEA and the compatibility of these approaches with our proposal for co-design. However, it is important to notice that this research is not co-design and does not include applying co-design. The intent of this research is to support our co-design research by approaching the application development from its philosophical perspective and building an insight to guide the engineering of the ADEA. Based on literature and personal observations this thesis shows that the ADEA can be considered a complex system and complex system engineering can be applied to engineering it. Further, this research suggests that the complex systems engineering approach used in combination with the practice-centered co-design approach might benefit in general the development of systems of everyday life.

Computer science has traditionally approached systems engineering and advancing information technology knowledge from technical perspective. But although every systems engineering problem has technical aspects, in the case of systems of systems just as important, and some might argue more important, are the contextual, human, organizational, policy, and political system dimensions that shape the decision space and feasible solutions for the technical system problems [Kea03]. Therefore, instead of addressing system of systems engineering primarily from the technical perspective, this thesis approaches its research subject from a wider contextual perspective. This necessarily entails philosophical and methodological problems with the consequence that the technical perspective is less predominant. Nevertheless this thesis aims to contribute to advancing information technology knowledge.

[1] Here the term ecosystem stands for the complex of a community of independent agents and its environment functioning as an *ecological unit* (a unit of living organisms and their environment).

This thesis is organized into six chapters. In the following chapter, I present briefly the ARKI research group's proposal for *co-design* approach and philosophy, its main concepts and its relation to other usability approaches in order to build understanding of the relevance of system of systems engineering perspective to this research. In the third chapter, I view systems engineering from the system of systems perspective and review systems engineering methodologies. In the fourth chapter, given the materials presented, I view application development and the engineering of the ARKI research group's application development environment from system of systems engineering perspective. In the fifth chapter, I discuss the relevance of this research for everyday application development. Finally, in the sixth chapter I present the conclusions and results of the research and suggest directions for future research.

2 Overview of Co-design

In this chapter I briefly present the ARKI research group's proposal for *co-design*[1] and view its relation to other usability concerned design approaches. The aim of this presentation is only to be illustrative, not exhaustive. The purpose is to give enough information to understand the background and design philosophical constraints set for this study. The intent is neither to assess the ARKI research group's proposal for co-design, its methodology or terminology, nor justify them. They are given as the "boundary conditions" and to contradict them is out of the scope of this thesis. In the following I refer to the ARKI research group's proposal for co-design as our proposal for co-design.

2.1 Co-design – a Design Strategy for a Digital Society

The starting point of this thesis is *co-design*, a proposal for a design strategy for a digital society. At the moment our proposal for co-design is rather a philosophy than a formal methodology – even its name is still arguable. It is inevitably work in progress, and therefore giving finite description is impossible. However, there are design approaches that relate to co-design and having a look at those helps to get a picture of our proposal for co-design. But before going to relations of co-design with other design approaches, I approach co-design by presenting some of the understanding it is based on.

2.1.1 Design

To understand co-design it is good to revisit the concept of *design* to free it from its exclusive use in relation to designer identity. However, proving the fundamental nature of design goes far beyond the purpose of this research, and my aim is only to

[1] The information of the ARKI group's proposal for co-design presented in here is based on presentations and documents by group leader Kari-Hans Kommonen [Kom01, Kom03a, Kom03b, Kom03c] and notes by researcher Andrea Botero Cabrera and other group members [Bot04a] as well as on discussions with Kari-Hans Kommonen. A more comprehensive discussion of the background of the ARKI research group's proposal for co-design is presented in Botero Cabrera et al., *Codesigning Visions, Uses, and Applications* [Bot03].

present some ideas that support rethinking the concept of design and understanding our approach to design.

Herbert Simon offers a good starting point to reconsider design. He argues in *Sciences of the Artificial* [Sim69, p. 111] that:

> "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones. The intellectual activity that produces material artifacts is no different fundamentally from the one that prescribes remedies for a sick patient or the one that devises a new sales plan for a company or a social welfare policy for a state. Design, so construed, is the core of all professional training; it is the principal mark that distinguishes the professions from the sciences. Schools of engineering, as well as schools of architecture, business, education, law, and medicine, are all centrally concerned with the process of design."

However, design is not limited only to practice of design in profession. Design can be considered as a basic, fundamental human characteristic. People design in their everyday life all the time; aims to shape one's life are an example of this. One can even argue as Nelson and Stolterman do in their book *The Design Way*: "Humans did not discover fire – they designed it" [NeS03, p. 9]. Hence, it is somewhat justified to consider design, an intentional act to change or improve an existing situation into preferred one, as Rachel Strickland, an architect and videographer, has noted: "a fundamental element of our species adaptation" (as quoted by Tom Moran[1]).

2.1.2 Rationale for Co-design

The understanding of design as a fundamental human characteristic forms a postulate for our proposal for co-design. But where it fundamentally emerges is the understanding that new technology and products can only become successful after people integrate them into their own lives (e.g. mobile phones and SMS, email, web blogs). In this way, people always perform the last steps of the design process and only through the new practices they introduce into the social system (families, communities, networks of friends), the products will become interesting to other people. On the other hand, having a role in social systems the products of digital technology influence people's ability to design their practices. That, on the other

[1] http://www.cityofsound.com/blog/2002/08/tom_moran_on_ev.html [18.4.2005]

hand, depends on the products' capabilities to be incorporated into *the complex of products and people* and its environment functioning as an *ecological* unit – into the *ecosystem* of the digital environment of everyday life.

We believe that some of the most severe barriers for usability and usefulness of new products rise from the lack of attention to *ecosystemic* issues. The system of digital products of everyday life is more complicated than the system of traditional products in the industrial society: each product is a component of an ecosystem of activities and relationships with other products and actors (see Figure 1). The traditional development model, in which systems are designed exclusively by experts in isolated developments and design is considered being finished at the factory, provides products that are usually designed to be fairly "closed" (as opposite of open) with little opportunity for creating new functionality. This leads to interoperability and system integration problems that people must face and solve alone with little help from the producers. However, the diversity of individual needs in the consumer market makes it difficult or impossible to survey a sufficiently comprehensive user feedback, let alone to take it into full account and do customizations and integrations that people need. Therefore, people need to be able to do final adaptation or design by themselves.

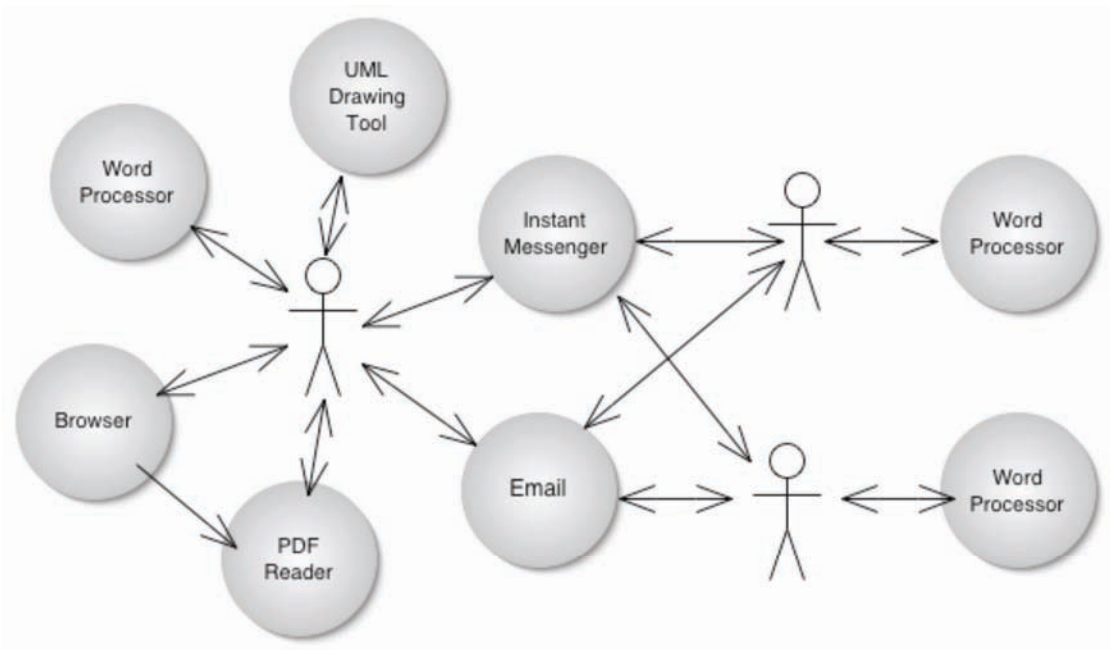

Figure 1. In order to have my master's thesis written I had to copy and paste text and images from various sources to a word processor. In order to get my master's thesis commented and finished I had to copy and paste several fragments of text to emails and instant messages and send several versions of the thesis to my colleagues (and my colleagues back to me) by email. To make the suggested changes to my thesis I had to copy and paste again text and images from various sources to have them modified. Thus, my colleagues and I "integrated" these various products into "Master's thesis writing and commenting system".

2.1.3 Co-design Approach

Co-design aims at tackling the above-mentioned ecosystemic problems by empowering the members of the social systems to design better and more functional and interesting systems. Our proposal is to enable social innovation by moving from a product oriented design process to a user practice centered dynamic design process. Instead of producing final static solutions to abstracted users, we aim to design the technology together with people toward an open system that is continuously adapted to the practices of everyday life uses. In order to achieve this we seek to involve more stakeholders (real people) than just users in a *shared* project, which is started by reflecting everyday practices instead of focusing only on products. During the project we aim to facilitate collaborative design by showing to stakeholders in a dialogical way the emerging possibilities and limitations of the technology. We hope to achieve this through identifying appropriate *design interfaces*, the set of tools, methods and practices that facilitate design activity (e.g. a *shared language*), for the different *design layers*, stages and stakeholders of the project, during the co-design process. The ecosystemic view is embedded into our approach: the social system and its design (e.g. the practices of community) are considered as an integral part of the process and its outcomes as the final digital artifacts, *applications* and *building blocks*, which I explain in the next chapter.

2.1.4 Building Blocks and Applications

Instead of the term *product* we rather use the terms *application* or *building block* depending on the context. A very simple view of the relation of applications and building blocks is that applications are composed of building blocks. However, thinking of an application as a traditional product does not give an accurate description of the concept of an application in our terminology but while the concept of a building block is explained in more detail the simplified definition of application serves the purpose.

Building blocks can be considered to range from the *lower design layers' technological components* (hardware and software) that are close to developers to the *upper design layers' elements* (application concepts) that are close to the end-users and designers. The purpose of defining and describing building blocks is to enable stakeholders in other *design layers* to understand what various components and elements are, what services they provide and under what conditions, and how they can be interacted with. For the software designers, to consider systems as layers is common practice. The challenge however is in extending communication to people who do not understand each other's components and language for describing the functionality that the components offer. Thus, describing building blocks is about defining appropriate *design interfaces* in order to (co-)design applications.

Applications on the other hand, we understand first as things that people do, *practices*, and only second as products. Therefore our concept of application differs in some extent from the concept of application (text processors, browsers, calendars etc.) used in information technology industry. On the other hand it is used more in its original meaning: a digital product can be used or applied to one or more things. Hence, it can have (or be part of) one or more *applications*. To understand the difference one might think of an application as an application of applications, a *meta-application*. As the concept suggests, a meta-application is an application that consist of other applications or products (compare to a *meta-system* that is a system of systems). But rather than composition of products, applications are for us essentially practices, in which they differ from traditional applications and products that are physical objects.

For example, my favorite application is "sharing photos of my daughter with my parents". The practice is more essential than the composition of this application. There are several possible compositions, *designs*, which consist of several traditional applications and products or rather building blocks. One of them consists of a digital camera, a personal computer, an image viewer, and email (see Figure 2). However, perhaps I could have a family album on my home server where I could put the photos. Whenever I add these photos, my parents will be informed automatically about the new photos in the album. This composition would enable realizing other similar applications with little effort (see Figure 3). Therefore the concept of application in our terminology is better understood as a practice rather than a product, and practices are essentially design opportunities: they reveal need for new designs.

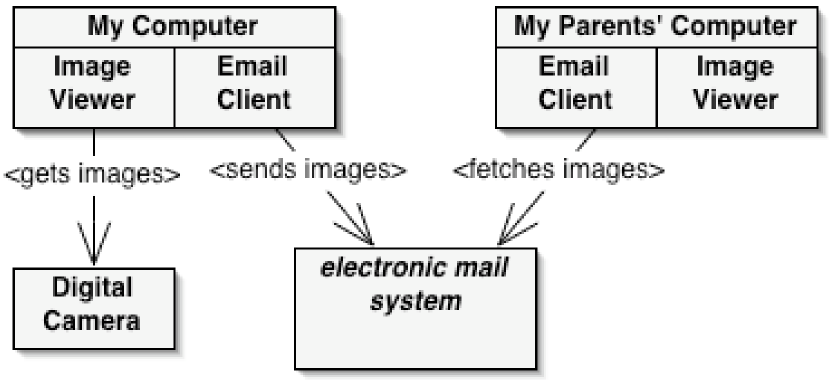

Figure 2. The application of "sharing photos of my daughter with my parents".

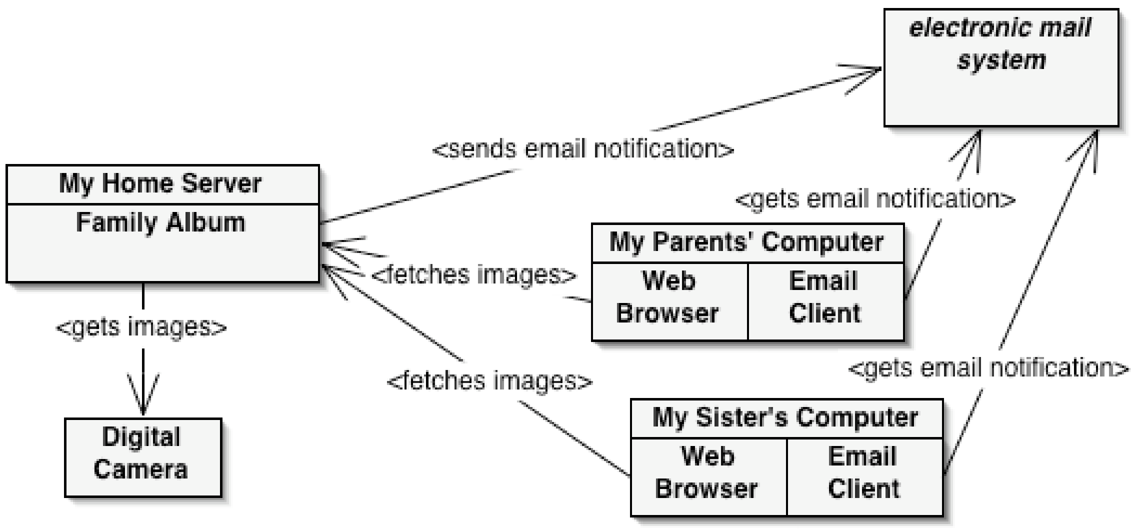

Figure 3. The application of "sharing photos of my daughter with my parents" realized using a home server and the application of "sharing photos of my daughter with my sister" which is realized using same building blocks with little more effort.

In summary, when our everyday life digital products and applications are examined separately, their usability can be considered high in general. They support well our simple tasks, e.g. as writing a party invitation. However, when one considers practices, e.g. organizing a party, there seems to be possibility to improve their usability. The problem, however, is not that people have to integrate products as meta-applications, systems of systems of everyday life, although occasionally this may suggest need for a product that better supports the needs. The problem is rather that there are not enough possibilities for people easily to compose independent products to systems of systems (see Figure 4). People are forced to act as system integrators and adaptors in their daily life in order to achieve their goals. For example if you want to go to aerobics you may be able to book the time online but the reservation does not go to your digital calendar unless you copy it from the display and neither does the reminder of the last day to return books that you borrowed from a public library nor your flight reservation. Or if you have to travel with public transport from your home in Tampere to your office in Helsinki, Tampere's local route guide system does not interoperate with the train guide system and neither does train guide system with Helsinki's local route guide system. You are forced to act as a system integrator and connect systems to each other by copying data from one system to another system. These examples suggest that there is need for a more comprehensive view on development of applications, products, and systems, which we encounter in our everyday life.

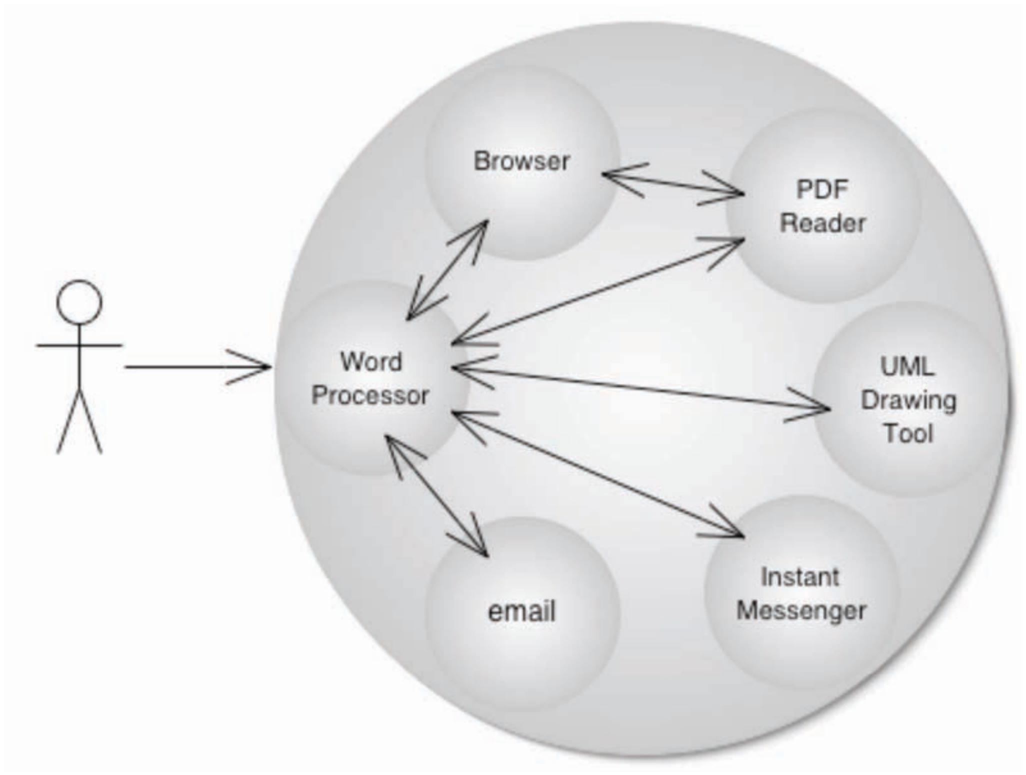

Figure 4. An improved version of my "Master's Thesis Commenting System" from my perspective. Instead that I operate as a "systems integrator" the products offers capabilities to be integrated with each other.

2.2 Co-design and Meta-design

Our proposal for co-design has many interesting points of convergence with the framework of *meta-design* by Fischer et al. [FiG04]. It might help to understand co-design by comparing it and meta-design and other "human-centered design" approaches such as *user-centered design* [NoD86] and *participatory design* [ScN93] because unlike co-design, these approaches are comprehensively documented in literature. Prior to comparing these approaches there are two concepts that need to be defined. They can be considered as "two basic stages of all design processes": *design time* and *use time* [FiG04]. At the design time, systems developers create environments and tools, as well as in conventional design approaches they create complete systems for the world-as-imagined [FiG04]. At the use time, users use the system to fill their needs but only to the extent that those needs are anticipated at the design time unless some modifications are made to the system [FiG04].

The similarities and differences between the above-mentioned “human-centered design” approaches are exposed by two aspects: 1) their understanding of people and 2) their focus on activities during “the two basic stages of all design process”, design time and use time. User-centered approaches can be considered limiting their understanding to people mainly as *users* (instead of e.g. as the members of the society). Therefore, they naturally place users in the central role and have a lot of focus on usability activities and processes taking place at design time in the systems' original development processes. The other approaches consider people more in their everyday context and not just as task-centric users. These *people-centered design* approaches seek to involve people (“users”) more deeply in the *collaborative design* process as *co-designers* or *co-developers* by empowering them to propose and generate design alternatives. By making work, technologies, and social institutions more responsive to people's needs, people-centered approaches also support more than user-centered design approaches diverse ways of thinking, planning and acting. However, both co-design and meta-design differ from user-centered design and participatory design approaches (whether done for users, by users, or with users) in their emphasis and aim to support people to evolve systems themselves.

Unlike user-centered design and participatory design approaches which place users mainly in reactive role, co-design and meta-design aim to provide people with means to be proactive. User-centered design and participatory design approaches focus on system development at design time and in both approaches developers and people (users) are brought together to envision the context of use only in design time. From the perspective of co-design and meta-design this is inadequate. Despite the best user-centered and participatory design efforts at the design time, systems and people are having difficulties when they need to adapt systems to new needs, account for changing tasks and practices, deal with subjects and contexts that increasingly blur professional and private life, couple with socio-technical environment in which they live, and incorporate new technologies [HeK91].

As a response to the above-mentioned challenges co-design and meta-design aim to create *open systems* that can be modified by their users and evolve at use time, hence supporting more complex interactions rather than linear or iterative processes. Because open systems should allow significant modifications when the need arises, both approaches set supporting the evolution that takes place through modifications as a "first class design activity". Bonnie Nardi advocates eloquently this call for open, evolvable systems in her book *A Small Matter of Programming* [Nar93, p. 3]:

> "We have only scratched the surface of what would be possible if end users could freely program their own applications. [...] As has been shown time and again, no matter how much designers and programmers try to anticipate and provide for what users will need, the effort always falls short because it is impossible to know in advance what may be needed. [...] End users should have the ability to create customizations, extensions, and applications..."

In summary, both co-design and meta-design aim at defining and creating a social and technical environment in which collaborative design can take place. Further, both approaches extend the traditional notion of system design beyond the original development of system to include a co-adaptive process between users and system, in which users become co-designers. However, there are also some differences between co-design and meta-design, aspects that are missing or are not clearly articulated in the framework of meta-design approach [FiG04].

One of the differences between co-design and meta-design is that we believe that people may have more to give if they were involved prior to deciding what systems or products will be developed. Therefore, we do not try to enable people to take advantage of new technology only in collaborative design but rather a *"collaborative innovation"* process by involving people to innovate before any decisions about a system or a product to be designed are made. This aspect is missing or is not well presented in the current framework of meta-design although meta-design also emphasizes the importance of social-innovation.

Another difference between co-design and meta-design is domain. Meta-design has concentrated to design *expert systems* (or rather *systems for experts*) of organizations [Fis01, Fis05] that can be considered very centralized and specialized, and having clear boundaries. But the "system" of our interest, the digital environment of everyday life, is very heterogeneous and has unclear boundaries. For the research problem of this study, engineering the ARKI research group's application development environment, this is the most interesting difference, which I show below by viewing the process model presented by Fischer et al. [FiO02].

In order to manage the development of large evolving expert systems and information repositories, Fischer et al. have presented a process model called *Seeding, Evolutionary growth, and Reseeding* (SER) [FiO02]. They justify the model by arguing that systems that evolve over a sustained time span must continually alternate between periods of activity and unplanned evolutions and period of deliberate (re)structuring and enhancement. In short, instead of building complete and closed systems, the SER model advocates building *seeds* that can be evolved over time through the contributions of people. During the *evolutionary growth* phase, people focus on solving a specific problem and creating problem-specific information rather than on creating re-usable information. In the *reseeding* phase the information gathered during evolutionary growth is organized, formalized, and generalized deliberately and in a centralized manner.

The SER process model is evidently feasible and useful in the development of complex socio-technical systems such as expert systems. But although the ARKI research group's application development environment is a sort of expert system, the SER model is only partly applicable to our purposes (e.g. developing one of our online collaboration environments), and therefore it is not quite adequate for us. The reason for this is that the characteristics of systems and the domain that the SER model is designed for differ from our domain and system. Rather than being a single system, the ARKI research group's application development environment as well as the domain of our interest (the digital environment of everyday life) is a *system of systems* and as such its development would probably benefit from applying the system of systems engineering methodologies suggested in the literature [Bar03, CoK03, Kea03, KSM03, NoK04]. In the next chapter, I review systems engineering from the perspective of these methodologies.

3 Engineering Systems of Systems

In this chapter I review engineering systems of systems. First, to facilitate discussion I define the concepts of *system*, *system of systems* and a specific kind of system of systems, *complex system*. After that, I briefly present the *traditional systems engineering* approach and the boundary conditions that have been recognized for applying it. These boundary conditions together with the characteristics of the systems of systems suggest the need for an alternative approach to engineer systems of systems. In the remainder of this chapter I review emerging *system of systems engineering* methodologies that attempt to address the shortcomings of traditional systems engineering in addressing system of systems problems.

3.1 Systems, Systems of Systems and Complex Systems

So far I have laid the concept of *system* on common understanding of what systems are. In order to facilitate discussion on single systems and systems of systems I now give a more accurate definition of the *system* and the *system of systems*. A single system is [GDM05]:

> "a combination of dependent elements operating together to accomplish a single common goal. The system cannot be expected to operate in the designed manner without its components and the components serve no useful purpose when separated from the system." See Figure 5.

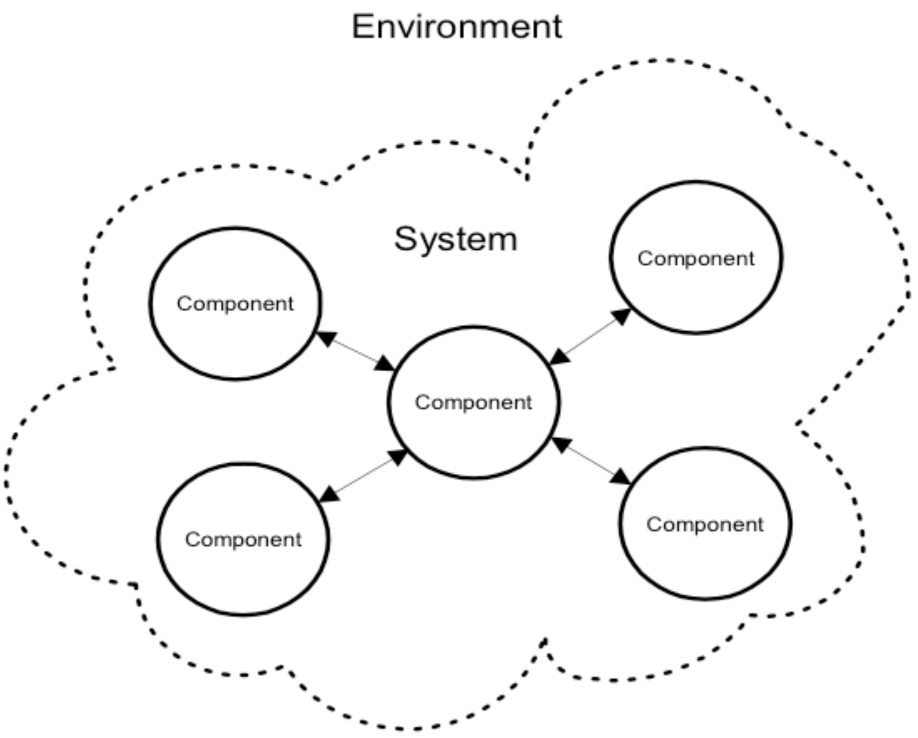

Figure 5. A single system, its components, and the environment. [GDM05]

A system of systems is [GDM05]:

> **1**: “a system built from independent systems that are managed separately from the larger system”;
>
> **2**: “a subset of systems”.

When composed to a system of systems, the *component systems* produce some utility that is greater than the sum of the individual component systems, e.g. produces some functionality that did not exist before or improves the usability of existing functionality. But when separated – unlike system components – the *components of system of systems* that are systems by the definition still serve some useful purpose. As single entities the component systems interact with both the environment and each other (see Figure 6). [GDM05]

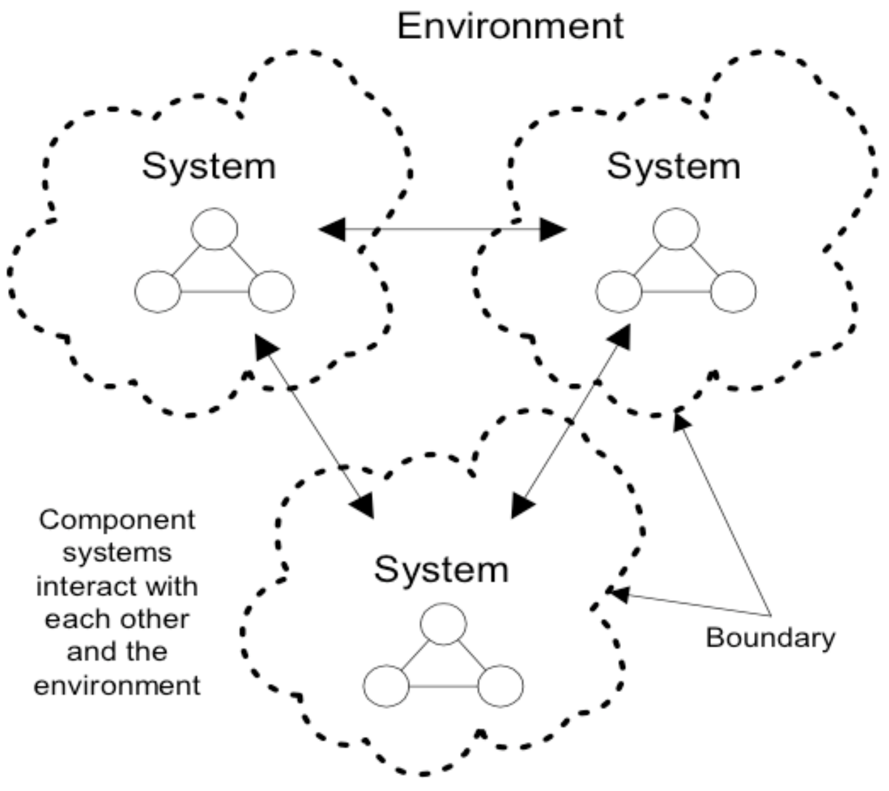
Environment
System
System
System
Component systems interact with each other and the environment
Boundary

Figure 6. A system of systems and the environment. [GDM05]

Where it is obvious based on the context, I will use the term *system* for a single system but when there is possibility for confusion, I will use the terms *single system* or *stand-alone system*. However, it should be clear that single systems and systems of systems are both *systems* although their characteristics differ. In order to distinguish very large and complex but monolithic systems from true systems of systems Maier has defined five characteristics of systems of systems, which I view in the case of the Internet [Mai96]:

The first characteristic of system of systems is the *operational independence of the elements*. If the system is decomposed, each element (component system) can still perform independently of the others. The elements of the system of systems are independent and useful in their own right. For example, the Internet is composed of computers and computer networks, which may continue their operation if the Internet were decomposed.

The second characteristic is the *managerial independence of the elements*. Each component system has its own purpose independent of the other component systems and they are managed separately for that purpose. The component systems are acquired separately and integrated. After integration, they maintain a constant operational existence independent of the system of systems. Again, the elements of the Internet fulfill this characteristic. They are acquired separately and after integration, connecting them to the Internet, they maintain their operational independence.

The third characteristic is *evolutionary development*. The system of systems is not fully formed or finished. Its development and existence is evolutionary with functions and purposes added, removed, and modified as experiences are gathered. It continually evolves as needs change and newer technologies become available. For example, computers and computer networks offering new services are constantly added into the Internet, modified and in some cases removed.

The fourth characteristic is *emergent behavior*. The system of systems performs functions and carries out purposes that are not possible by any of the individual systems operating alone. The reason for developing the system of systems is to obtain this unique behavior. In the case of the Internet, the World Wide Web is one of the examples of emergent behavior.

The fifth characteristic is *geographic distribution*. In many cases individual component systems are distributed over large geographic areas. They can readily exchange only information and not substantial quantities of mass or energy. In the case of the Internet, this is self-evident. The Internet is spread around the World and the computers can only exchange information and neither mass nor energy.

In order to be considered a true system of systems, a system should have all or a majority of these characteristics [Mai96]. However, to understand system of systems problems these characteristics are not sufficient. To have understanding of the special characteristics of system of systems problems the concepts of *complexity*, *emergence* and *complex systems* are essential. In the following, I study these concepts in detail.

3.1.1 Complexity, Emergence and Complex Systems

In this chapter, I define the concepts of *complexity*, *emergence*, and *complex systems*. They are needed to understand system of systems problems and systems engineering approaches proposed to address the problems. I start from a simple definition of complexity and give a preparatory definition for a complex system. Then I review the concept of emergence and return to complexity in order to render it to a useful engineering concept. Finally, I define the characteristics of complex systems more precisely from the perspective of developing an engineering approach.

First, the term "complexity" does not mean "difficult to understand", although something that is complex may be understood only with considerable effort [NoK04, p. 11]. Merriam-Webster Online[1] offers the following definition for the term "complex":

> **1**: a whole made up of complicated or interrelated parts

To explain the difference between simple and complex systems, the term "interrelated" is essential. To understand the behavior of a complex system we must understand not only the behavior of the parts but how they act together to form the behavior of the whole [Bar97, p. 1].

Also the term "complicated" is important. The parts of a complex system are often complex systems themselves [Bar97, p. 5]. However, this is not the only possibility. We can describe a system composed of simple parts where the collective behavior is complex. This is called *emergent complexity* [Bar97, p. 5]. Any system formed out of atoms is an example. The idea of emergent complexity is that many simple parts interact in such a way that the behavior of the whole is complex[2]. [Bar97, p. 5]

Emergence, considering a collection of elements and the properties of the collective behavior of these elements, is one of the two approaches to organizing the properties

[1] http://www.webster.com [1.3.2005]
[2] We can describe also a system composed of complex parts where the collective behavior is simple. This is called *emergent simplicity*. An illustrating example is a planet orbiting around a star. The behavior of the planet is quite simple, even if the planet is the Earth that has many complex systems upon it. [Bar97, p. 5]

of complex systems [Bar97, pp. 5, 10]. The second approach begins from understanding the relationship of systems to their descriptions. For this approach, the central issue is to define qualitatively what we mean by *complexity*. It aims at answering what we mean when we say that a system is complex and how to identify complexity of one system and to compare it with the complexity of another system. [Bar97, p. 6]

Bar-Yam [Bar97, p. 12] suggests that

> "Loosely speaking, the complexity of a system is the amount of information needed in order to describe it. The complexity depends on the level of detail required in the description."

Norman and Kuras [NoK04, pp. 11–12] argue that although this definition takes the concept close to a useful understanding for engineering, and borrows in an attractive way from Shannon's information theory [Sha48], it also seems arbitrary in some ways, as it suggests that a collection becomes more complex when measured with more precision. For example, if one calculates all the possible arrangements of books in the office, the number of discernible possibilities is different depending on the precision of the ruler used. However, it is still the same room and the same set of books. Arguably, the complexity should be the same. It should not depend on the measuring method. The counter argument is that the use of a different ruler is equivalent to using a different scale; hence, finding that the complexity at different scales is different should not be surprising [NoK04, p. 12].

There are also other interesting views of complexity. One aspect to contemplate is the difference between the actual number of possibilities and the number of useful possibilities [NoK04, p. 12]. Another view of complexity is Turchin's theory known as Metasystem Transition Theory [Tur77, Tur95], which describes an evolutionary process that generates higher levels of complexity and hierarchical control in system structure and function. However, whatever model is used to understand complexity, rendering "complexity" into a useful engineering concept requires metrics [NoK04, p. 12].

Norman and Kuras [NoK04, p. 13] suggest that measures of complexity and *intricacy* may serve as good metrics to understand the relative merits of a system, and may be useful for relative comparisons. The term "intricacy" is often considered synonymous with the term "complexity". However, there is a reason to argue that they are *not* synonyms. There is a difference, which an example may help to understand. Norman and Kuras [NoK04, pp. 12–13] use a board game called *Mousetrap* played by children. In the game, players move colored mice that act as playing pieces around a board. In doing so they build a Rube-Goldberg mousetrap which one player ends up using to capture the other player's mouse, thus winning the game. The advertising copy reads as follows[1]:

> "Construct a crazy mice-catchin' contraption piece by piece as you race your mice around the track! Once it's built, turn the crank...that kicks the marble...that rolls down the chute...and sets off a zany chain reaction that just might trap a pesky mouse!"

The bizarre mouse-catching device is *intricate*, "difficult to understand". However, it is not *complex*. First, it has only one possible configuration, and it results in only one behavior. Second, it does not interact at all with its environment. Each piece is carefully fitted onto the previous structure, which sets up the conditions for the subsequent structure, and it assembles the same way each time. [NoK04, p. 13]

Norman and Kuras [NoK04, p. 13] ground their argument of the usefulness of measures of complexity and intricacy on the mathematical properties of *complexity* and *intricacy*. They argue that the mathematical properties of complexity and intricacy can be shown to relate to specific mathematical characteristics. It appears that intricacy relates to the number of axes of characterization, "the absolute volume of a hyperspace defined by axes", whereas complexity relates to "the volume reachable within this hyperspace" [NoK04, p. 13]. For example, the hyperspace of the mousetrap device has many axes; yet the mousetrap device has a narrow extent along each axis, forming a narrow volume of reachability within this hyperspace [NoK04, p. 13]. However, Norman and Kuras leave the detailed mathematical treatment of the subject in a subsequent publication and approach the rendering of the "complexity" to a useful engineering concept by reviewing formulations of complexity in the discipline of architecture.

[1] http://www.areyougame.com [2.3.2005]

Christopher Alexander's architectural patterns offer models for considering complexity and emergence in the architecture [Ale79]. Emerging from the repeated application of the principles, Alexander writes about spaces, homes, town and cities, which are "alive". Norman and Kuras write about Alexander's concept of "alive", his patterns, and their meaning for the complex systems [NoK04, p. 13]:

> "His concept of 'alive' is a reflection of the interactions among the components in the environment and the people, and the support the environment affords to the repeated patterns and events, which make up the peoples' experiences minute-to-minute and day-to-day. He recognizes that there are both patterns formed at higher levels from bottoms-up application of patterns, and there are explicit patterns applied at higher levels – and in this he hints at multiscale analysis."[1]

Fundamentally, Alexander is talking about the relations among interacting entities, and the results of those relations [NoK04, p. 13]. His notion of complexity seems to align the notion of "*order*". In the informal sense *order* is often associated with organization as well as with the actions or other forms of direction that lead to the organization [NoK04, p. 13–14]. In that sense, order is not simply a passive thing, a state property, but it is dynamic. It combines both form and function. Thus, by focusing on the *relationships of things*, not just the state of the things as a result of the relationships, we can understand the reasons for the organization and perhaps understand the implications to change, and even infer or deduce state elsewhere, which may be out of view [NoK04, p. 14].

Formally, the order of a system is a measure [NoK04, p. 14]. The measure is the set of all specific and instant relationships among the parts of a system. The "relationships" as defined by Norman and Kuras are: "patterns in attributes, where attributes define the parts of a system (and sets of 'values' define attributes)" [NoK04, p. 14]. Now, it is possible to infer or deduce the specific values of an attribute, a part of a system, based on other attribute values because those attribute values collectively form relationships, i.e. patterns. [NoK04, p. 14]

Compared with other given characterizations of complexity, Norman and Kuras' characterization of complexity seems to provide the most useful characterization of

[1] About applying multiscale analysis see *Duality* in Chapter 3.5.1.

complexity for engineering purposes. It manages to characterize things in active way, which is more than appropriate for entities that are active, systems.

Finally, having an understanding of complexity and how it might be measured, we can review what makes a system a *complex system*. In the following are the characteristics of complex systems presented in the literature [Bar97, Hey95, Hol95, Kau93], which must be considered when developing an engineering approach [NoK04, p. 15, Nor04, p. 5]:

- The structure and behavior of a complex system is not deducible from the structure and behavior of its component parts.

- The elements of a complex system can change in response to imposed "pressures" from neighboring elements (consequently leading to reciprocal and transitive implications).

- A complex system has a large number of useful potential compositions of its elements.

- Given a steady influx of energy (raw resources), a complex system increases its own complexity.

- A complex system is characterized by the presence of independent change agents.

Engineered systems of systems can be seen having these characteristics [NoK04]. Furthermore, systems of systems can be viewed as having the characteristics of ecosystems [NoK04 pp. 1, 17]. Rather than being top-down designs, complex systems are bottom-up constructions. They evolve through *co-evolution* [Nor04]. The concept of *co-evolution*, on the other hand, comes originally from biology. In the nature species and organisms live in an environment and form an ecosystem of interdependent organisms and species. Co-evolution occurs when these organisms and species, agents, living in close relationship adapt to changes caused by each other. The relationships can be predator-prey relationships or more like a symbiosis, the main thing is that the agents affect each other causing "development pressure". The change in a complex system is introduced very much like in an ecosystem. Change causes local "pressures" among juxtaposed elements against which the system responds [Nor04].

Thus, with the *complex system* we mean a system that fulfills the characteristics listed above, a system that is active and "alive", and cannot be isolated or studied separately from its environment. For characterization and comparison of a complex system, the balance of *complexity* and *intricacy* might serve as a measure, whereas other corollaries of these measurements might be a measurement of the rate at which the system adapts to required or desired change [NoK04, p. 15].

Given the definitions of a single system, a system of systems and a complex system, I proceed to reviewing traditional systems engineering and reasons for its shortcomings in dealing with systems of systems.

3.2 Traditional Systems Engineering

Systems engineering, as any engineering discipline, is the subject of ongoing discussion, research, and debate. There are several standards and models produced by various organizations, e.g. INCOSE Systems Engineering Handbook [INC04], IEEE Standard 1220-1998 [IEE98], CMMI®[1], but there is no single commonly agreed standard or model on how systems should be engineered. However, it is possible to identify the main features of most commonly applied approaches, and to give an abstract presentation of *traditional systems engineering* (TSE).

3.2.1 The Practice of the Traditional Systems Engineering

TSE decomposes distinct functions of the engineering process into individual phases that can be performed sequentially, each phase building on the previous one. The most familiar process models of this approach are *waterfall model* [Roy70] and *spiral model* [Boe88]. Most systems engineering approaches are variations of these two

[1] http://www.sei.cmu.edu/cmmi/cmmi.html [3.9.2005]

basic process models. Both processes proceed through a well-defined series of stages, specifying requirements, developing designs, and implementing and testing those designs to comply with or to satisfy exactly the specified requirements. The waterfall approach tends to go through this process only one time. It aims at getting everything right the first time by building systematically from coarse to fine granularity in design and implementation. The spiral approach proceeds through the process many times. These are called *iterative cycles*. In each cycle a more complete design or implementation is created, problems from the past cycles are corrected, and new previously ignored details are added.

Whether the process model is the waterfall model or the spiral model, the practice of TSE can be considered as the application of a series of *linear transformations* moving from the statements of the requirements to a preliminary design, a final design, actual development, then testing and delivering [NoK04, p. 7]. In the waterfall model this happens only once, whereas in the spiral model this happens in each cycle. The other fundamental characteristic that unifies different TSE approaches is the aim to understand the position of a system within the environment, isolate the system under study from the environment, and then treat the environment as a constant [NoK04, p. 8].

3.2.2 Boundary Conditions for Traditional Systems Engineering

TSE has been and is an effective and proven discipline for addressing the problems of single and even very technically complex systems. The Manhattan project (the atomic bomb) and the Space program (the conquest of the Moon) stand as the hallmarks of TSE approach to engineering large projects [Bar03], and several smaller and less known but equally successful projects stand as the proof of its solid practices. However, to have a successful, or at least a low risk outcome, there are some absolutely required characteristics, *boundary conditions*, for applying TSE [NoK04, p. 9]. Failing to have any of these boundary conditions raises the risks of TSE projects dramatically, and it is unlikely that other mitigation strategies can be found to reduce the risks introduced [NoK04, p. 9]:

1) The specific desired outcome must be known *a priori*, and it must be clear and unambiguous (implied in this is that the boundaries of the system, and thus responsibility, are clear and known);

2) There must be a single, common manager who is able to make decisions about allocating available resources to ensure completion;

3) Change is introduced and managed centrally;

4) There must be interchangeable resources (that is money, people, time, etc.), which can be applied and reallocated as needed.

Keating et al. [KSM03] have also presented a very similar list of conditions but not as strict as the boundary conditions presented above. Their six primary conditions suggest only that a system of system engineering methodology may be *preferable* to TSE approaches if any or all of the following conditions are present:

1) *Turbulent environmental conditions* – the environment for systems engineering effort is highly dynamic, uncertain, and rapidly changing.

2) *Ill-defined problem conditions* – the circumstances and conditions surrounding the problem are in dispute, not readily accessible, or lack sufficient consensus for initial problem definition.

3) *Contextual dominance* – the technical aspects are overshadowed by the context within which the problem system is embedded. Success will be as much determined by adequately addressing the contextual problem drivers as the technical problem drivers.

4) *Uncertain approach* – the path of progression on how "best" to proceed with systems engineering effort is indeterminate. Standard processes for systems engineering are either failing or highly suspect for adequately addressing the situation.

5) *Ambiguous expectations and objectives* – the ability to establish measures of success or system objectives for the systems engineering effort are vague. This may be a result of inadequate understanding, hidden motives, or lack of technical competence to proceed with a systems engineering effort.

6) *Excessive complexity* – the boundaries of the system are such that its complexity is beyond capabilities of TSE. To proceed requires significant simplification of objectives.

In general, the emerging systems of systems problems are recognized to stretch the boundaries of TSE [Kea03, Bar03, NoK04]. Despite the success of the Manhattan project and the Space program, the reality is that most large engineering projects, which generally continue to follow the TSE paradigm, are much less satisfactory [Bar03]. The reason is that inherent to the paradigm there are several assumptions that are questionable in the present systems of systems environments [Bar03]:

1) Substantially new technology will be used.
2) The new technology to be used is based upon a clear understanding of the basic principles or equations that govern the system.
3) The goal of the project and its specific objectives and specifications are clearly understood.
4) Based upon the specifications, a design will be implemented and consequently the project or mission will be accomplished.

First, in the case of system of systems it is unlikely that only new technology will be used and the system of systems can be changed from the old to a new one over a very short time [Bar03]. The engineers have to deal most likely with *legacy* systems. Second, it is unlikely that there is a clear understanding of the basic principles and equations that govern the system of systems. The non-linear dynamics of engineered systems of systems, which are also considered having the characteristics of *complex systems* [Bar03, CoK03, NoK04], are difficult to understand (see Chapter 3.1.1). Furthermore, although technical aspects are important, in the case of system of systems just as important, or one might argue more important, are the *contextual* *issues*: human, organizational, policy, and political system dimensions that will ultimately change the decision space and feasible solutions for technical system problems [Kea03]. This has been recognized repeatedly in the socio-technical literature (e.g. TaF93, Kea01). Third, although the overall goal of the system of systems project might be clear in succinct form, the specific objectives are most likely ill-defined, unclear and unambiguous [Bar03, Kea03, NoK04]. Fourth, because of the long-term maintenance of systems of systems and pressures addressed to their evolution, one cannot consider their development to be completed [NoK04]. Thus, the assumption that based upon the specifications, a design will be implemented and consequently the project or mission will be accomplished is incorrect.

In summary, systems of systems stretch the boundaries of traditional systems engineering in three important areas:

First, TSE has not been developed to address the high levels of ambiguity and uncertainty encountered in system of systems engineering. TSE has difficulties to adequately respond to ill-structured problems with constantly shifting requirements. This is a problem because in system of system environments it is naïve to think that problem definitions and requirements will be isolated from shifts and pressures stemming from highly dynamic and turbulent development and operational environments. [Kea03]

Second, although TSE does not ignore *contextual influences* (human, organizational, policy, and political system dimensions) on system problem formulation, analysis, and resolution, it certainly places the context in the background. In contrast, the problems of system of systems are evolving in ways that suggest contextual aspects must be moved to the foreground. Practitioners have recognized that system of systems problems cannot be artificially separated from their context, the circumstances and conditions within which they are embedded because the context can both constrain and overshadow technical analysis in determining system solution success. [Kea03]

Third, TSE has been successful at deploying "complete" system solutions especially through iterative development processes. However, pressures on system of systems design and deployment dictate that *partial systems solutions* must be deployed and iterated after deployment. This is contrary to the linear nature of TSE approach that aims to complete design followed by complete implementation. [Kea03]

Given the presentation of the traditional systems engineering and its limitation related to engineering systems of systems I review below first less traditional systems engineering approaches, *agile methodologies*, from the perspective of engineering system of systems and later the emerging system of systems engineering methodologies.

3.3 Agile Methodologies

There are a growing number of agile methodologies and a number of agile practices attempting to offer methods to utilize leading edge technologies, respond to erratic requirement changes, and deliver products quickly. This they achieve through adaptability, which is gained by incremental (small release, rapid cycles) and iterative processes that embody evolutionary features. The most known set of agile methodologies includes Lean Development (LD), ASD (Adaptive Software Development), Scrum, XP (eXtreme Programming), Crystal methodologies (Crystal Clear, Crystal Orange, Crystal Orange Web)[1], FDD (Feature Driven Development) and DSDM (Dynamic Systems Development Method) [Hig02]. Occasionally also Free/Open Source Software Development (F/OSS) is included into the agile methodologies because of its similarities with the other agile methodologies (see, e.g Abr02 and Fow03). Thus, the F/OSS approach can be considered as a variant of the multifaceted agile methodologies. Table 1 shows how F/OSS places itself between the agile methodologies and TSE.

[1] Only Crystal Clear, Crystal Orange and Crystal Orange Web are constructed [Coc02] and the first two of these have been experimented in practice [Abr02].

Home-ground area	Agile Methodologies	F/OSS Development	TSE
Developers	Agile, knowledgeable, collocated, and collaborative	Geographically distributed, collaborative, knowledgeable and agile teams	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Dedicated, knowledgeable, collaborative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Largely emergent; rapid change, commonly owned, continually evolving – "never" finalized	Knowable early; largely stable
Architecture	Designed for current requirements	Open, designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Inexpensive	Expensive
Size	Smaller teams and products	Larger dispersed teams and smaller products	Larger teams and products
Primary objective	Rapid value	Challenging problem	High assurance

Table 1. Home ground for agile and TSE [Boe02], augmented with Free/Open Source Software column (F/OSS) [Abr02]. (In the original table instead of TSE was used the term *plan-driven methods*)

From the system of systems perspective, the most of the above listed agile methodologies are not applicable. Although Crystal Orange, FDD, F/OSS, ASD and DSDM are claimed to be capable of scaling up to projects having 100 developers [Abr02] and Scrum even up to 300 [Lar04, p. 130], XP, Scrum and Crystal Clear and Crystal Orange are suitable only for small or medium sized collocated teams. XP, Scrum and Crystal methodologies do not scale well to projects having distributed geographically [Abr02]. XP scales for two teams working on related projects with limited interaction [Bec99] and it lacks management practices in general [Abr02]. Scrum is not for large, complex team structures, however, small isolated team on a large project could make use of some elements but then the interfaces between the smaller sub-teams must be clearly defined [RiJ00]. The Crystal methodologies are restricted to address only collocated teams, thus they do not support distributed development at all [Coc02]. DSDM is applied to large projects lasting 23 years and it scales if the system can be split into the components that can be developed in small teams having 2–6 members but in order to apply the DSDM approach the time used to the development should be constrained [Sta97]. Thus, DSDM does not appear suitable for system of systems engineering that "never" ends. FDD is claimed to be "worthy of serious consideration by any software development organization that needs to deliver quality, business-critical software systems on time" [PaF02, p. xxiii] but it covers only design and implementation and not the whole life-cycle of the system [Abr02].

From the above listed variants of the agile methodologies, LD, ASD, and the F/OSS are the most promising methodologies from the system of systems perspective. LD has been used successfully on a number of large telecommunications projects in Europe [Hig02]. LD is especially interesting as the most strategic-oriented agile methodology. It advocates wide adoption of agile methodologies by strategic selling at senior levels within the organization. ASD, on the other hand, is interesting because of no built-in limitations for its application and the adoption of complex systems theory to engineer large systems [Hig02] whereas F/OSS is interesting because the F/OSS projects can be considered complex systems [Kuw00]. However, ASD and F/OSS are more development philosophies than methods *per se* [Abr02]. On the other hand, as the following review of the system of system engineering exposes, the methodologies proposed for engineering especially system of systems are themselves rather philosophies or embryonic methodologies than well-defined set of methods and practices. Nevertheless, agile methodologies are rather addressed engineering single systems than systems of systems. In general only small organizations may select a single agile methodology and customize it to their needs. For larger organizations, one or more agile methodologies in combination with TSE methodologies have to be retailed [Hig02, p. 365]. TSE, on the other hand, has its boundary conditions, which suggests that there is a need for a methodology that especially addresses system of system engineering problems.

3.4 System of Systems Engineering

The concept of *system of systems engineering* (SoSE) has received considerable attention in the literature but unfortunately the state of the literature is "a rather fragmented collection of seemingly disparate perspectives on the associated phenomena" [Kea03]. The shortcomings of the SoSE literature are lack of universally accepted definition of system of systems and addressing SoSE primarily as an information technology issue that has a broad objective of "*getting everything to work together*" [Kea03]. For the latter is given proof terms such as "interoperability", "platform integration", "systems architecture", and "information intensive", that have emerged to capture the information dimension of these new systems of systems [Kea03]. The narrow focus of SoSE dialogue on information technology and technical problem solving is a problem because in the case of a system of systems just as important are the contextual dimensions that ultimately shape the decision space and feasible solutions for the technical system problems [Kea03]. Furthermore, dominance of information technology in definition of system of systems concepts relates directly to the absence of "any in-depth advancement of fundamental principles, underlying theory, accepted methodologies, or body of empirical work that would constitute foundations for a discipline" [Kea03]. Evidently, SoSE is in the embryonic stages of development, which is further affirmed by the following list of issues that SoSE research should produce [Kea03]:

1) *SoSE philosophy* to capture different level of thinking inherent in the system of systems approach.
2) *Methodologies* that provide guidance and direction for the structuring and achievement of SoSE initiatives.
3) *Processes* that provide methods for specific aspects of SoSE.
4) *Techniques* that enhance knowledge and advance practice through specific tools to support SoSE efforts.

Since SoSE is in early stages of development there are naturally only few comprehensive approaches to develop its concept, foundations, research directions, and practice implications. In the following, I review first implications of SoSE for systems engineering practitioners as suggested by Keating et al. [Kea03] and their proposal for a SoSE methodology [KSM03]. Later, I review *complex systems* *engineering* [Bar03, NoK04], which is an approach that is based on the *complex systems* theory and takes rather different perspective on SoSE than the methodology by Keating et al.

3.4.1 Implications for System of Systems Engineering

Keating et al. [Kea03] have provided the following initial guidance and implications for SoSE practitioners.

First, *proceed with the assumption that the initial problem definition is always incorrect and suspect*. A system of systems is created to solve a problem or fulfill a mission. The development is burdened with a considerable amount of uncertainty and ambiguity in the system environment, boundaries, and stakeholders' interests. Therefore, instead of considering that one has perfect knowledge at the start, one should design the used processes to permit continual questioning and reframing of problems and missions.

Second, *building system transformation capability is more important than initial deployment* because the initial deployment of a system of systems is always going to be incorrect. Therefore, system of systems must be engineered to have inherent capabilities for flexibility, rapid identification of systematic failure issues, and system reconfiguration deployment. System of systems engineers must recognize when the speed of deployment outweighs solution completeness. Implied in this is that transformation capability and iteration are the highest priority.

Third, *bringing context in the foreground and technical solution to the background* determines success in the system of systems environment. Context involves issues that are likely to impact the approach, design decisions, and deployment of systems of systems. These may involve organizational, structural, resource allocation, procedural, policy, or political issues. System of systems engineers should consider these influences in the foreground and not to relegate them to background "noise", because inability to solve these can doom even the "best technical solution".

Fourth, *effectiveness in system of system engineering environments is determined first as a function of systems worldview*, or philosophy, which is critical in determining success in system of systems environments [Kea03]. Although arguing primary of worldview or philosophy might be met with skepticism, one only needs to look at the systems engineering tools and techniques that have not generated the level of success promised in the system of systems environments [Bar03, Kea03, NoK04]. This suggests that there must be something beyond tools and techniques that will generate success in addressing complex systems problems: systemic perspective, worldview or philosophy [Kea03]. The systemic perspective, worldview, has to be embedded as the fundamental approach to solving system of systems problems. It guides thinking, decision-making, acting and interpretation of what is done and how it is done [Kea03]. Thus, one should take care that the team members have a sufficient worldview to achieve high performance, that appropriate training, education and development is planned to bring individuals and the team to a sufficient level of maturity for success, and that there is compatibility between the supporting philosophy and the system of systems engineering approach [Kea03].

3.4.2 Proposal for a Methodology

Evidently, the above-presented implications for systems engineering can operate only as guidance not as a methodology. In addition Keating et al. [KSM03] have provided an example of an initiative SoSE methodology. However, this methodology addresses only *existing system transformation* while leaving out the other contexts, *new system design*, *system operation and maintenance*, and *evaluation and evolution*, that SoSE should address [Kea03]. Thus, the methodology suggested by Keating et al. cannot be considered generic as such. Further, the following presentation is generalized by replacing the terms “port” and “port security system” with the term “system”.

The SoSE methodology proposed by Keating et al. [KSM03] (SoSEM) for existing system transformation contains following iterative phases:

Phase I – *System Problem Definition and Environment*

- Conduct an exploration of the system environment to articulate the context and system problem.
- Produce a framing of the problem as well as the operating environment and context.

Phase II – *Model Current System*

- Conduct a SoSE analysis to model the existing system to and identify systemic issues.
- Produce a description of the current system as it is operating.

Phase III – *Integrate System Requirements Definition and Conceptual Design*

- Define high-level requirements and develop a conceptual design for a system that would be operationally compatible and integrated with other system related initiatives and activities being pursued on local or other relevant levels (e.g. federal, state).

Phase IV – *Analysis for System Deployment*

- Identify and assess systemic issues, barriers, and opportunities for deployment of the integrated system including a prioritized listing for enhanced system operation.
- Provide an assessment of the gaps between current and conceptually ideal systems in order to lay the foundation for a transformation strategy based on gaps and priorities.

There are three aspects that differentiate this approach as a SoSE approach rather than TSE approach [KSM03]. The first aspect is acceptance of the lack of problem understanding. It is not assumed that the precise specification, approach, and problem are sufficiently understood to detail specifics: "The first phase of the effort was an attempt to bring sufficient structure and order to the problem system such that further work would be fruitful " [KSM03]. The second aspect is that the approach is tailored to the initial situation and flexible enough that emergent conditions can modify the approach. Because the SoSE methodology must be top-down and provide increasing levels of resolution and detail as new understanding of the problem emerges, the approach provides an increasing clarity and definition as the system investigation proceeds and is adjustable [KSM03]. The third aspect is that the approach places systems engineers in a position of being methodological experts and not necessarily technical experts:

> "Although sufficient knowledge of the subject area must be developed, the role of the systems engineer is to initially bring structure and order to the problem system. Thus, the effort can proceed with expert technical knowledge infused as appropriate for the effort. Thus the systems engineering role encompasses methodology specification and execution in pursuit of the SoSE solution." [KSM03, p. 6]

A rather different perspective to SoSE is provided by Bar-Yam [Bar03, Bar05] and Norman and Kuras [NoK04, Nor04]. In following, I present their methodology.

3.5 Complex Systems Engineering

The *complex systems engineering* (CSE) approach rises from the understanding that the *traditional system engineering* (TSE) does not scale up or manages poorly with systems having characteristics of an *ecosystem*, i.e. complex systems [Bar03, Bar05, NoK04]. *Enterprises*, collections of independent organizations that are loosely associated to achieve something in common, seem to be such systems [NoK04]. In general, systems that fulfill the definition of complex systems seem to violate the boundary conditions for applying TSE approach [NoK04]. TSE seems to apply to fairly simple[1] applications and products, which are under the complete control and management of a single party [NoK04]. Based on these differences, it seems reasonable to conclude that there is a need to apply a system engineering approach, which acknowledges the difference between *complex systems* and more traditional developments to which TSE can be applied [NoK04]. This approach is suggested to be CSE [Bar03, NoK04].

The most fundamental concept for CSE is (*co-*)*evolution*, which is the primary mechanism of change within complex systems (see Chapter 3.1.1). The evolution provides a conceptual framework in which to understand how repetitive incremental

[1] Despite the superficial complexity of the Manhattan project and the Space Program, the tasks that they were striving to achieve were relatively simple compared to, e.g. the problems of air and traffic control [Bar03].

change can safely produce both rapid innovation and increase overall complexity [Bar02]. Repetitive incremental change – embodying features of evolutionary processes – is not new in systems engineering (see e.g. spiral development [Boe88], Adaptive Software Development, eXtreme Programming and Free/Open Source Software development, and agile methodologies in general [Fow03, Hig02]). However, CSE aims at taking even better advantage of the promise of evolutionary methods by "a deliberate and accelerated mimicry of the processes that drive emergence and natural evolution" [Bar03]. The key differences between the evolutionary approach of CSE and the strategies of other evolutionary methodologies are: 1) an emphasis on parallel *competitive* development teams and 2) the importance of creating an ongoing *fielded* implementation strategy where coexistence of multiple types of components is possible [Bar05].

In other words, CSE is *not* a new or renewed attention to detail like the traditional evolutionary-oriented system engineering approaches; it is an attention to overall coherence that permits diversity in the systems environment [Bar03, Bar05, NoK04]. This is exposed by the most essential difference between TSE and CSE, the acknowledgement of *autonomous agents*, in the case of software intensive systems: independent development tracks. [NoK04]. CSE acknowledges the presence of active, independent agents as important elements of systems of systems, whereas from TSE perspective they (or rather their autonomy) is considered precisely the effectors, which must be eliminated to apply TSE and have everything under complete and centralized control. CSE's approach is different. It augments a set of tools for addressing the presence of the autonomous agents. It applies "selective pressures" to *the aggregate of interacting agents* allowing them to manage their response and their own changes. Through this approach, it aims at addressing the overall coherence without a direct and immediate attention to detail.

In summary, Table 2 contrasts differences between artifacts that TSE and CSE produce. The term *product* is used to identify the outcome of TSE and the term *enterprise* is used to identify the outcome of complex systems engineering. (For a more comprehensive analysis of simple and complex products, see Hob98). Given the motivation and objectives of CSE, I review in following the key concepts and the emerging methodology of CSE.

TSE	CSE
Products are reproducible.	No two enterprises are alike.
Products are realized to meet pre-conceived specifications.	Enterprises continually evolve so as to increase their own complexity.
Products have well-defined boundaries.	Enterprises have ambiguous boundaries.
Unwanted possibilities are removed during the realizations of products.	New possibilities are constantly assessed for utility and feasibility in the evolution of an enterprise.
External agents integrate products.	Enterprises are self-integrating and re-integrating.
Development always ends for each instance of product realization.	Enterprise development never ends – enterprises evolve.
Product development ends when unwanted possibilities are removed and sources of internal friction (competition for resources, differing interpretations of the same inputs, etc.) are removed.	Enterprises depend on both internal cooperation and internal competition to stimulate their evolution.

Table 2 Comparing TSE and CSE [NoK04, p. 18].

3.5.1 Complex Systems Engineering Methodology

The first answer for succeeding with complex engineering projects, which the field of complexity provides, is limiting the complexity of objectives as much as possible [Bar03]. However, simplifying is not always possible because the necessary or desired core function is itself highly complex. Recognizing what gives rise to complexity helps to understand this. The complexity of engineered system is related to its task [Bar03]. The complexity of a task, on the other hand, can be quantified as the number of possible wrong ways per every right way. The more likely the system performs a wrong action, the more complex the task. In order for a system to perform a task it must be able to perform the right action. As a rule, this also means that the number of possible actions that the system can perform and select between must be at least this number. This is known as the "Law of Requisite Variety" [Ash56] that relates the complexity of a task to the complexity of a system that can perform the task. When the complexity of the system rises too high and simplification will no longer work, using an evolutionary engineering approach becomes essential [Bar03, Bar05].

In general, the goal of systems engineering can be considered being to increase the order of system and the available complexity that is needed to perform the tasks addressed to the system. However, there is a practical upper limit to the degree to which this can be done successfully through pre-specification followed by implementation, i.e. TSE. To engineer a system beyond this limit one needs to combine several related activities into a single continuous *regimen* of engineering and development. Here a regimen is distinguished from a *recipe* that can be understood as shorthand for the cumulative processes of TSE. Recipes are tightly and precisely scripted sequences of steps to yield reproducible outcomes such as specific kind of dishes (or software system) whereas regimens are looser formulations of more generalized steps that can be used and combined in various ways to yield many different instances of generalized outcomes such as better health and life quality (or an air operations center). The argued advantage of CSE over TSE in engineering complex systems is based on this difference: whereas TSE is a practice of direct impact and effects, CSE tends to be indirect. CSE drives change in the focus from implementing the solution designed according to requirements to resolving or reducing "selective pressures" acting on the present system elements[1]. CSE does this seeking to bring together independent, disparate organizations and entities and providing them with a sense of "pressure" that they feel, and a set of processes that can be used to resolve this pressure. It targets to provide incentives for the partnerships needed and to compel the engagement of their respective resources to accomplish the integration without resorting to arguments over whose money is being spent, or whether "interoperability" or "integration" is a "requirement" they have. [NoK04]

To understand the CSE approach and how it differs from TSE, the following key concepts that CSE employs may be contrasted to the practices of TSE [Bar03, Bar05]:

[1] In the context provided by CSE, the elements of system (single systems) can be built using the traditional engineering approaches [Bar03, NoK04].

- Focus on creating an environment and process rather than a product

- Continually build on what already exists

- Individual components must be modifiable *in situ*

- Operational systems include multiple versions of functional components

- Utilize multiple parallel development processes

- Evaluate experimentally *in situ*

- Gradually increase utilization of more effective components

- Effective solutions to specific problems cannot be anticipated in advance

- Complex systems are “integrated” continuously

These key concepts capture the “paradigm shift” from a “complete system specification” to the creation of an environment where evolutionary change can take place. This requires understanding complex systems as populations rather than as rigid assemblies of unique components, and encouraging, safeguarding and monitoring multiple parallel efforts exploring experimentally improvements to the system. [Bar05]

Norman and Kuras [NoK04] have elaborated in some aspects the ideas presented in the above listed key concepts. Their work have not been published and reviewed, so in the following I acquiesce in giving a summary of “the elements of CSE regimen” that they have proposed to capture the methodology of CSE [NoK04]. The elements include *developmental environment*, *outcome spaces*, *rewards*, *developmental precepts*, *judging*, *continuous characterization*, *safety regulations*, and *duality*. After introducing the elements I present them in a combination.

Developmental Environment

The overall regimen of CSE aims at creating and managing an environment in which multiple autonomous agents can interact to explore the utility and practicality of creating, modifying or disregarding existing relationships [NoK04, p. 19] and in which a process of innovation and creative change can take place [Bar03]. This *developmental environment* can be understood as either a separate and distinct environment in which complex systems develop and operate or an overall ecosystem that includes the system and the environment [NoK04, p. 20]. Thereby, human beings (designers, engineers, users) and technology (computers, communication devices, networks) as interactive agents, and even the process of creating systems components (design, implementation, training) should all be understood to be parts of the system itself [Bar03].

As such, the focus of this activity is the completeness of the environment relative to supporting the more focused activities that occur in it [NoK04, p. 20]. Within this environment it is possible for traditional systems engineering of software and hardware components to occur [Bar03]. However, the focus of TSE efforts is to change the parts of the systems rather than change the system as whole.

Because the developmental environment must be available to all the independent agents, attention to the environment must be explicit and continuous. Thus, nurturing and managing the developmental environment so that it can evolve itself, rises as the single most important activity underpinning the deliberate development of complex systems. [NoK04, p. 20]

Outcome Spaces

CSE tends to identify *outcome spaces* instead of *specific outcomes*. An outcome space is distinguished from the many specific outcomes that comprise it. When specific outcomes are sought or meant to be exactly reproducible, one should use TSE to achieve them. Depending on whether the outcome can be realized by individual autonomous agents by themselves or achieved by autonomous agents collectively but not individually, competition or cooperation should be encouraged. For sustained development complex systems need both (more about competition and cooperation in Chapter 3.5.2.) [Bar03, NoK04, pp. 20–21]

In order to explore possibilities, the diversity of entities and components, all specific outcomes in the outcome space should be viewed as acceptable as any other without there being strong preferences for any of them. However, this does not mean that outcomes cannot be identified as unwanted. One possibility to do this is to partition outcome spaces into wanted and unwanted sub-spaces. [Bar03, NoK04, pp. 20–21]

Rewards

The decisions of autonomous agents (independent development tracks) determine the utility and the practicality of existing and new relationships within the complex system. *Rewards* are directed to shape their decision-making processes and to motivate the autonomous agents to make decisions that cause the complex system to enter the desired outcome spaces. Rewards, however, should not be dependent on specific processes of the autonomous agents, unless the specific outcome space is the adopting of a common process. First, because the autonomous agents could view insisting on a common process as too invasive. Second, because insisting a common process might stifle innovation and variety needed for evolution. [NoK04, p. 21]

In general, rewards can be considered as the energy flowing through a complex system. In the case of software intensive system, rewards are almost always associated to the money flowing through the entire system development environment. However, there are also other forms of rewards. The essential characteristic of a reward is that it motivates, and as long as people are involved, the list of possible rewards is as long as the list of factors that motivate people. [NoK04, p. 22]

One should also consider the possibility to distribute rewards extra-contractually. It offers motivation to autonomous agents to “keep their eye on” the whole complex system, even if they are not engaged directly. This helps in avoiding stagnation, because innovations, which the autonomous agents bring to the whole system, set up potential of new approaches and influences. [NoK04, p. 22]

Developmental Precepts

Developmental precepts are easy to confuse with rewards, however they are quite different. Rewards promise gain for achieving outcome spaces whereas developmental precepts constitutes “the rules of the game”. They stimulate contextual discovery and interaction among agents e.g. by establishing constraints on how autonomous agents achieve outcome spaces, or how they interact. They do not specify outcomes or even outcome spaces. They shape autonomous decision-making leading to specific outcomes. For example, in some specific cases a developmental precept might claim that two or more component systems must be delivered at the same time in order to ensure that the integration of those systems is not left to the end-users to accomplish. The specifics of which systems to be delivered and how they need to be interconnected would be left to the system providers to resolve in their own best interests. However, the “global” outcome of more integrated systems would also be accomplished, even though the specifics of how and when were never explicitly formulated in advance at any “global” level. [NoK04, p. 22–24]

Judging

Judging associates specific outcomes *achieved* with autonomous agents, and assigns rewards to them accordingly. It requires human judgment. The most important characteristic of judging is that because rewards are established *prior* to realization of desired outcomes and judging is based on actual outcomes achieved, judging is based solely on what actually happens, not on what will happen. [NoK04, p. 24]

Judging for rewards that are associated with outcomes that can be directly attained by autonomous agents is quite straightforward. Judging for rewards that are associated with specific outcomes that are in outcome spaces that can only be associated collectively with autonomous agents is more demanding. For example, if a complex system achieves to reduce the use of resources, its *footprint*, although some of the components have increased their footprints, judging requires identification of the responsible agents and the apportionment of the reward. [NoK04, p. 24]

Continuous Characterization

The *characterizations* of outcome spaces and rewards are initially represented only with succinct labels. This maximizes opportunities for autonomous agents to interpret the characterizations inconsistently as well as to shape the evolution of the complex system. To the extent that consistency matters, outcome spaces, rewards and especially *the current condition of the complex system* will benefit from continuous and progressively more detailed and complete characterizations. However, because the autonomous agents are acting independently, consistency can never be guaranteed in complex system development. Therefore, the characterizations cannot be made too detailed. Nevertheless, characterization refinement can become less than cost effective or even counter-productive, if the details obscure rather than illuminate essential coherence that is desired and achieved. Furthermore, although consistency tends to accelerate complex system evolution, it accelerates the system evolution in narrower directions, i.e. directions explicitly identified and characterized. Therefore, it is necessary to add new apparent or attractive outcome spaces to the characterizations even with initially limited detail, because it is unlikely that the new possibilities will be explored even though available unless they are registered. In the extreme, this might result in stagnation in overall complex system behavior. [NoK04, p. 25]

Safety Regulations

Safety regulations are aimed at preserving the developmental environment [NoK04, p. 26]. They preserve the *stability* of a complex system. For example safety regulations define how new components are introduced to the system. The overall purpose of such safety regulations would be to protect other developments and components when a new component is introduced to the system. Safety regulations might also be applied to the retirement of no-longer-used runtime components, which on the other hand can operate as tools (backup systems with override capability of the new component systems) in the safety regulation of a system [Bar03, NoK04, pp. 26-27].

Safety regulations should also be directed at the detection of the continued presence of cooperation and competition since they are both necessary for the operation of any complex system. In general, safety regulations are about avoiding “collapse” and “stagnation” in overall complex system behavior. [NoK04, p. 27]

Duality

Duality is the explicit recognition that complex systems “development time” cannot be fully separated from its “run time”. Multiscale analysis (see Figure 7) might be applied to both distinguish between and to understand the relationships among separate scales in a complex system during its “development time” and “run time”. These scales include the system components (software, hardware), their developers (groups of people as independent development tracks), and the human operators of the system. [NoK04, pp. 27–28]

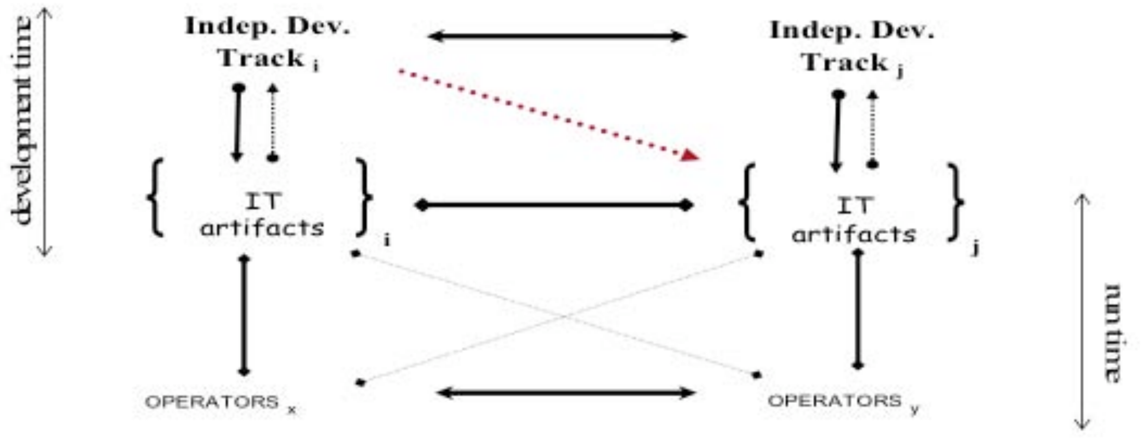

Figure 7. Multiscale analysis applied to the relationships among separate scales in a complex system. [NoK04, p. 28]

In Figure 7, the independent development tracks (autonomous agents) that create the IT artifacts (the components of a software intensive system) interact with one another as well as with their creations. These interactions are frequently identified as occurring during “development time”. The IT artifacts interact also with one another. Such interactions can occur during development time, but most of the time they are thought of as an essential element of the “run time” of the system. However, in almost every case during “run time” there are also interactions between the system components and their human operators. Moreover, these human operators often interact with one other directly without intermediate technology. [NoK04, p. 28]

Complex systems engineering aims at taking into account not only run time but also development time interactions. In order to do this, it has to define outcome spaces at least at three distinct scales, two corresponding to “run time” and one corresponding to “development time”. [NoK04, p. 28]. Growing attention to user-developer interaction during development time can be considered as an implicit recognition of this multi-scale reality [NoK04, p. 28]. Furthermore, growing attention to system development by users during run time (see Chapter 2.2) can also be seen as the recognition of need to take into account these multiple interactions.

Running the Regimen of Complex Systems Engineering

The approach of complex systems engineering unfolds in various ways. That is because it is a regimen not a recipe. In the following, is a summary of the view of Norman and Kuras [NoK04, pp. 29–30]:

Early on, one should clearly formulate desired outcome spaces as well as publish rewards available to autonomous agents. Whereas the actual phases and trajectory of complex system are determined collectively by autonomous agents using traditional systems engineering approaches, recognizing specific outcomes when they occur is a primary task of the complex system engineer.

Once recognized, desirable outcomes are rewarded. This does not happen automatically but human judgment is required. This judgment must remain the prerogative of those responsible of the emergent complex system. Once judgments are made the rewards must be restated along with the restatements of desired outcomes spaces. This formulation of desirable outcome spaces should never stop. The desirable outcome spaces should be restated along with rewards, as new outcomes occur.

The complex system engineer is responsible for managing the overall developmental environment that mixes operational and developmental contexts. Attention on the developmental aspects of this mixed environment includes specifying, operating, maintaining and modifying an infrastructure that supports interactions among autonomous agents and their creations, specifying and enforcing developmental precepts intended to stimulate discovery and interaction among the autonomous agents, and the specification and application of safety regulations.

Complex systems operate continuously. They change constantly and become more complex. Therefore, a complex system engineer characterizes continuously the complex system and its outcome spaces, emphasizing aspects that are associated with the order of the system. Such system and outcome space characterizations become the assignment of responsibility for changes in the complex system's order. In turn, the assignments become the basis for judging, which assigns rewards to the appropriate autonomous agents.

Given the presentation of the methodology of CSE, I briefly present some tactics that Norman and Kuras [NoK04] have suggested as an initiative proposal for the practice of CSE.

3.5.2 Tactics for Complex Systems Engineering

A free market system, the economy, is an example of an evolutionary complex system [Bar05] (about the economy as an evolving complex system, see e.g. AAP88, ADL97). Thus, market economies may be mined for some suggestions for tactics and strategies for CSE [NoK04]. The most relevant of tactics are those which make technical change easier, permit organizations to collaborate, and trim the environment selecting "success" and punishing "failure" [NoK04]. Table 3 shows a set of tactics that Norman and Kuras have harvested from commercial practices and maps them to the aspects of CSE regimen as they propose [NoK04, p. 31]. In the following, I explain the tactics briefly.

	Developmental environment	Outcome spaces	Rewards	Development precepts	Judging	Continuous characterization	Safety regulations	Duality	Independent agents
Half-life Separation	●	●		●			●	●	
Playgrounds	●	●	●	●	●	●	●	●	●
Collaborative Environments	●	●		●		●			●
Partnerships									●
Developers Networks	●	●				●	●	●	●
Branding		●	●	●	●			●	
Co-opetition	●								●
Leveraging others' Investments			●	●		●		●	●
Respect Ricebowls	●	●		●					●
Opportunistic Approach	●	●				●	●	●	●
Advertising and Discovery	●	●	●	●		●		●	●
Value-add business models			●		●	●			●
Experience for test				●	●	●	●	●	

Table 3. Commercial Practices mapped to the Elements of CSE Regimen. [NoK04]

Half-life separation

Layering systems based on functionality is familiar for system architects. However, despite of this principle, systems might still be tightly-bound into monolithic entities, which limit the "pulse" of the evolution to move at the pace of the slowest changing element. In order to increase the rate of evolution, one must shorten the generation time (pulse time, spiral time, etc.). Therefore, in addition to separating based on functionality, one should separate based on “half-life”, likely rate of change. [NoK04, p. 32]

Playgrounds

Play and games serve an important function for humans in preparing for life. They are not merely for killing time. Humans, as natural pattern recognizers and problem solvers, learn and innovate through experimentation. This can be seen everyday among children playing in *playgrounds* where they constantly innovate and learn through interaction. [NoK04, p. 32]

The concept of *playground* tries to answer to the problem of introducing and recognizing “goodness” in the complex system environment. The concept suggests that instead of large, carefully-scripted demonstrations with centrally chosen participants and known answers, one should create a place where technology could be experimented safely and connected to the process that delivers new elements into the field, and where it could provide its potential qualitative edge. The reason for the “potential” rather than for the “actual” edge is that experiments should be possible result also in negative findings. [NoK04, p. 33]

It is also important to remember that the essence of complex systems is that change is constant, unplanned, and unpredictable in its complete effect. In the complex systems environments no element stands alone. Each element supplies partial context back to elements around it. Thus, any change in any element causes a change in context to all elements, which juxtapose the changed element. As a result, change flows to neighboring elements, which respond causing further change in their neighboring elements. Generally, the effects and pressures brought by any change are difficult or even impossible to predict. Playground aims at answering to this problem as well. Instead of developing new things assuming that all other elements it may impact or influence have themselves remained static, it is possible to developed them in a playground where they are connected to each other. [NoK04 p. 33]

Finally, the concept of playground aims at solving contradiction in welcoming change. The organizational process is for the keepers of the “Systems of Record” to invite an innovator or new-capability provider into the fold. However, this puts the identification and valuing of innovation into the hands that are least likely to welcome it, because by definition innovation's appearance is disruptive. Playground aims at providing a place where innovation can be elaborated without disturbing real-world working processes until it is ready to be introduced. [NoK04, pp. 33–34]

In summary, if an environment provides a place to play and innovate, it will support the evolution of system at a rate faster that would occur otherwise. If there is also a process for guiding and managing evolution, then the system can be developed based on demonstrated value rather than future promises of value. However, the most important advantage of playground is that developing capabilities through discovery does not require the level of detail and *a priori* planning that a pure engineering approach requires. [NoK04, p. 35]

Collaborative Environments

Collaboration allows people to concentrate on what they do best and how they add value rather than expending resources on incidental aspects, which do not discriminate their offerings from others. Additionally, the better that what one has produced fits into a bigger whole, one is able to collaborate with others more easily and the risk of integration is reduced. Therefore, it is important to create environments where collaboration can take place. [NoK04, p. 35]

Partnerships

Partnerships help to increase the utility of what the partners produce and offer, to spread development risk and to improve understanding of true need. Successful partnerships seek to reduce overlap, and come to rely on what others provide. Therefore, the ability to form and sustain partnerships is important in complex system environments. [NoK04, p. 35]

Developers Networks

To speed up the evolution of system, one must shorten the generation time. This suggests shortening the feedback cycle and lowering the amount of code that must be written by increasing reuse of useful code. This is best achieved by getting developers to use a specific platform and thereby interact with each other. An illustrating example is Microsoft Windows™ platform. Microsoft's strategy was to create opportunities for others. They were not building all the functionality themselves, nor were they trying to make a killing on a developer tool *per se*. They did not even try to get a piece of action for all the functionality developed using their tools. Instead, they sowed the seeds with their tools, and took advantage of the multitude of applications built on top of their platform. An important factor in this success story was recognition that it was necessary to lower the knowledge barriers to developing sophisticated applications and that integration and interoperability required many developers loyal to the platform. In order to win developers loyalty Microsoft introduced Microsoft Developer's Network (MSDN) that provided attractive and compelling developer tools and environments. [NoK04 p. 35–36]

Branding

Brand suggests that the product satisfy specific qualities. Without it, the capability of the product must be subjected to a process of evaluation. Further, if there is not a certification that the product fulfills specific qualities, there is risk that the product has to be reworked before it qualifies, e.g. it can be integrated to the existing system. [NoK04, p. 36–37]

Co-opetition

The term *co-opetition* was coined by Adam Brandenburger and Barry Nalebuff [BrN05]. It offers a theory of creating and capturing value which contains fundamental duality. Whereas creating value is an inherently cooperative process, capturing value is inherently competitive. In order to create value, people cannot act in isolation. They have to recognize their interdependence. But along with creating value, there is the sharing of the value which happens through competition. Co-opetition makes these both possible by allowing independent parties to cooperate on those elements and aspects which transcend their individual ability to control, while preserving their ability to compete on demonstrated value in their space. [BrN05]

Leveraging Others' Investments

Partnerships, collaborations, and other instances of cooperation all attempt to use the investments that others have made for their own benefit. Leveraging others' investments directly benefits oneself and indirectly others. For example, a project α leverages with a small amount of money the development budget of joint project β for good behavior on project β's part. Project β manages to build and deploy desired functionality. Both benefit, and the others (who may not have invested anything) are also able to use the services built by the project β. [NoK04, p. 38]

Respect Ricebowls

People place great value in that which they are personally involved in and responsible for. These interests are often described as "ricebowls". The thought or impression that the others will impose themselves on the independent agents in ways and manners, which they view are inappropriate, causes resistance to cooperation. Because cooperation has great value, aspects that interfere with it cause "innovation drag". Therefore technical approaches which tend to respect "ricebowls" are worth considering in order to remove some of the hesitations for forming cooperative partnerships. Examples of technical approaches that respect "ricebowls" include current development in web services. Web technologies expose functionality with the minimum requirements for homogeneity. They offer a "virtual homogeneity" within a heterogeneous world. In this way, they offer potentiality for other independent agents to offer their services to others. This permits new associations and relations to be exploited and thereby supports innovation and the rise of technical structures and approaches which permit the agility needed: assembly moves out toward the end-users further blurring the difference between development and run time.
[NoK04, p. 38]

Opportunistic Approach

Restricting oneself to deliver complete solutions before fielding limits the "pulse" of the evolution to apparent "complete" sets, and hence slows the evolution of the system. However, if one would treat logical sets of users as a unit, and involve them in managing the identification and introduction of functionality and change, then one might be able to be more responsive. [NoK04, pp. 38–39]

Advertising and Discovery

Finding useful capabilities and functionality offered by third parties is not trivial. However, for the development environment and the operational environments, achieving transparency for effective advertising and discovery is critical. Advertising and discovery technologies can operate as a key enabler to create opportunities for *small world phenomenon* to emerge. The small world phenomenon is the hypothesis that everyone in the world can be reached through a short chain of social acquaintances. This creates possibilities to discover new relations in areas not previously explored. [NoK04, p. 39]

Value-add Business Models

The users complain continuously that they do not get "a vote" in what is built for them.[1] This is primarily due to the business models employed today. The dominant business model used is an *employer-contractor*. The employer presents requirements, and then various potential contractors propose how they will produce the functionality desired. The market place is selling and buying the engineering hours and a certain process which can be argued to reduce risks. It is not a demonstration of specific achievement but a promise well told. Success is not directly related to the usefulness of produced outcomes. [NoK04, p. 39]

An alternative for the employer-contractor model could be a *by-use payment* model complemented with an assumption that there is no *a priori* assumption of the undesirability of redundant functionality. Under such model money would flow to those who produce demonstrated utility to the user. This would cause the market to shift to understanding and satisfying real needs rather than the sale of engineering hours. [NoK04, p. 39]

[1] This may apply in the business-to-business market where often managers instead of end-users make purchasing decisions but in the business-to-consumer market this does not hold. Consumers can "vote" by not buying something. On the other hand the customers, people, are often brought into the design process after it has already decided what is build for them, and the products are loaded with functionality that the users do not need but are forced to pay for.

Experience for Test

Full-coverage testing of complex systems is very difficult if not impossible. Rather than on relying traditional approaches, one might collect and catalog things when they go wrong in the field and analyze them. Obviously there is need to develop testing approaches to have some sense of belief about the systems before fielding. In addition, one should test the infrastructure to failure in order to know the performance boundaries of the system. Then the system should be monitored in the field to see if the limits are approached. This makes possible to intervene before, not only after, problems emerge. [NoK04, p. 40]

3.6 Comparison of Systems Engineering Approaches

In summary, I compare some aspects of the above-presented systems engineering approaches.

Focus

The primary focus of TSE as well as agile methodologies is a *single system*, whereas the focus of SoSEM and CSE is a *system of systems*. Furthermore, the focus of CSE is on creating an environment and process rather than a product. In this CSE differs from SoSEM, which focuses on the product just as TSE. On the other hand, SoSE methodologies are not adaptable to the development of single systems, whose development should be approached using more or less traditional systems engineering methodologies.

Expectation

TSE is expected to provide a *solution*, whereas SoSEM and CSE as well as agile methodologies are expected to provide *initial response*. SoSEM, CSE and agile methodologies provide the "solution" as the system evolves. This is due to the fact that TSE provides a solution to a defined problem, whereas agile methodologies, SoSEM and CSE consider problems as emergent.

Approach and Its Characteristics

The approach of TSE is *process* whereas the approach of SoSEM and CSE is *methodology*. The approaches of agile methodologies range from process to methodology. However, the approach of TSE and agile methodologies as well as the approach of SoSEM is *direct* and *linear*. They build on subsequent phases. In addition TSE and SoSEM are *top-down* approaches. In these aspects SoSEM is close to TSE. CSE, on the other hand, can be considered *indirect* and *non-linear*, and its approach, as well as the approach of agile methodologies to actual problem solution, is *bottom-up* although system evolution is guided top-down. In the top-down guidance CSE and SoSEM resemble each other. Other points of convergence between the system of systems methodologies can be found in modeling: whereas SoSEM *models* an existing system, CSE *characterizes* it, and whereas SoSEM defines the "ideal" model of system, CSE defines "outcome spaces". On the other hand *analysis* and *requirement specification* are common to all systems engineering approaches. However, SoSEM and CSE differ from TSE and agile approaches that instead of technical dominance contextual influences are central to the analysis.

Autonomous agents

The most essential difference between the methodologies culminates in relation to autonomous agents (development tracks). TSE does not accept autonomy of agents. In order to succeed TSE requires everything to be under complete and central control. The perspective of SoSEM to the autonomous agents is somewhat unclear. Correa and Keating [CoK03] have viewed systems of systems as *artificial complex (adaptive) systems* but the top-down approach [KSM03] that they propose does not count that development tracks building the component systems would autonomously provide right solutions. CSE, on the other hand, as well as agile methodologies counts on that self-organizing development tracks will autonomously deliver desired outcomes.

Given the presentation of systems of systems and systems engineering methodologies I will proceed to application development and the application development environment of the ARKI research group.

4 Application Development Environment of ARKI

In this chapter, I analyze application development in the ARKI research group from system of systems engineering perspective and show that system of systems engineering provides valuable insight to our application development. As an example, I use an actual development case, *Voice Notes*. After that, I analyze the characteristics of the application development environment of the ARKI research group and applicability of system of systems engineering methodologies to its overall development. Based on this analysis I argue that in our case complex systems engineering is preferable to other approaches. In addition, I propose some applications of complex systems engineering tactics. Finally, I present implications of system of systems perspective to single systems developers.

Before proceeding I define two concepts in order to facilitate discussion. With the concept of *Application Development Environment of ARKI* (ADEA) I mean people, software, and hardware involved into the design, development and operation of applications, whereas with the concept of *application* I mean a *single system* or *system of systems* that is composed of the components of ADEA. At the moment the overall development of ADEA is not in any project's explicit interest or set of requirements but the projects of the ARKI research group are concerned about developing separate applications. The development of the ADEA toward an open, adaptable and interoperable (co-design compatible) system that better supports separate application developments is the ARKI research group's private endeavor and so far there has not been any formal engineering approach to manage its overall development.

4.1 Practical Application Development Case: Voice Notes

The case of voice message sharing application[1], *Voice Notes*, offers an example of co-design process but also an example of an emerging system of systems thinking in everyday application development. The origin of Voice Notes application lies in the

[1] Voice Notes is rather a *platform* or a part of platform that enables Voice Notes *applications* but in order to facilitate discussion I refer here to the Voice Notes as a single application.

long-term co-design relationship with one of our collaboration communities. From previous probes on diaries and sharing, we formulated a development theme of "remembering and reminding" (see ARKI Thematic Publication 1, 2004 [Bot04b]) that we explored in a workshop with some of the members of the community. We brought to the workshop also an interest in developing an audio blog. However, our scenario about it did not produce any enthusiasm, as it combined audio notes with radio and television, which were felt to be too public as interfaces for private notes. Only after some twists and turns, the idea of sharable audio notes got its final shape: the community members came up with a need to offer means to leave a voice message with a phone and being able to listen, rename, comment and share it with others in the web. The actual implementation of Voice Notes, on the other hand, was not such a success story. However, during the implementation, considerations that finally lead to this research got their shape.

The Voice Notes was originally developed or rather hacked up through a couple of iterations from a demo to a prototype without applying consciously any methodology or approach. Therefore, the purpose is not to prove applicability of any systems engineering approach empirically. However, during the development and afterwards I came to think that there might be lessons learned. From my perspective the system of systems engineering approach seemed to offer a valuable view on what we are aiming at and to help us better to understand and manage the development of applications in the ARKI research group's co-design projects.

4.1.1 The Architecture of Voice Notes

The development of the Voice Notes was rather systems integration than creation of something totally new from scratch. However, from independent elements emerged something new. In that the development of Voice Notes resembled more system of systems engineering than traditional systems engineering.

In Figure 8 the architecture of the second version of Voice Notes is presented at a general level. The first version was composed of an Answering Service, an email server (Mail Boxer), and a Web User Interface with an embedded QuickTime Player to view and listen to messages, *voicenotes*. The second version also included a

Document Generator component that generated documents from emails that contained audio file, and some functionality to edit and comment these documents. The main functionality was built on top of the Zope (Z object publishing environment), which is an open-source web application server (see Appendix 1), and Zope CMF (Content Management Framework), which is an open-source extension for Zope (Zope product). In addition the functionality of the online environment of community (customizations made to Zope CMF) was utilized. Together these form a part of ArkiWorks application platform.

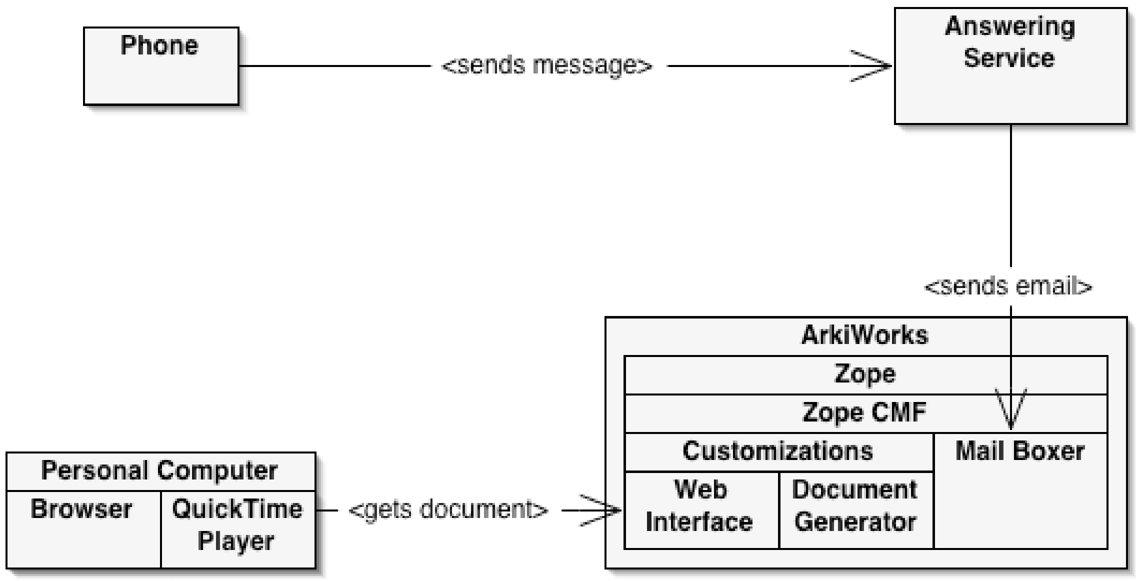

Figure 8 The Architecture of Voice Notes Application. The user leaves a voice message to the Answering Service. The Answering Service sends the message as an email attachment. The Mail Boxer receives the email and calls Document Generator, which generates a document from the email. The same or another user listens the voice message embedded in the document, re-titles the document, writes a description of the content or comments the message.

4.1.2 Voice Notes Application as a System of Systems

What is noticeable in the Voice Notes is its character as a system. Evidently Voice Notes is not a large complex system of systems. However, it is more a system of systems than a single system and for an everyday application its technical complexity is rather high although manageable using traditional systems engineering approaches. Furthermore, one may argue that the Voice Notes is not a system composed of systems but a system that is composed of components. However, this view does not give an accurate description of the Voice Notes, which an analysis based on Maier's [Mai96] five principal characteristics of systems of systems exposes.

If we take look at the elements of Voice Notes (see Figure 8), the Phone, the Answering Service, the ArkiWorks, the Personal Computer with the QuickTime Player and the Browser are clearly systems in their own right. They are *operationally and managerially independent* and the Phone, the Answering Service, the ArkiWorks and the personal Computer are even *geographically separated*. They can be and are *operated independently*. They are developed for their own purposes and not for the purposes of the Voice Notes. The only component-like parts are the Web Interface, the Document Generator, and the document editing and commenting components (Customizations) that are essentially extensions of Zope and Zope CMF.

The status of Zope, Zope CMF and Mail Boxer is somewhat controversial. They are open-source software and in a way they are also operationally and managerially independent although instances of them are operated and managed locally. However, their independence arises from the fact that they may or may not satisfy the needs of the Voice Notes as such and there is neither financial nor other means to force them to be developed for certain purposes. They support the Voice Notes perfectly or not at all or something between. (And actually they did not: a few long known bugs were encountered, and some workarounds were needed.)

It seems reasonable to argue that at least the first, the second and the fifth characteristic of systems of systems [Mai96] are present to some extent in Voice Notes: operational and managerial independence of elements and geographic distribution (the Phone, the Answering Service, the ArkiWorks, and the Personal Computer). Furthermore, the fourth characteristic, *emergent behavior* is fulfilled since the Voice Notes performs functions and carries out purposes that do not reside in any single element of the Voice Notes. The third characteristic, *evolutionary development*, is also fulfilled since functions and purposes were added, removed, and modified, and development will still continue as experiences are gathered and new needs are recognized.

Furthermore, not only the composition of Voice Notes but also its development suggests that the system of systems provides a valuable perspective for us. Initial guidance and implications for system of system engineering provided by Keating et al. [Kea03] (see Chapter 3.4.1) gives support for this argument:

First, *proceed with the assumption that the initial problem definition is always incorrect and suspect* [Kea03]. As systems of systems are created to solve a problem or fulfill a mission, the Voice Notes was created to carry out an experiment, which was burdened with a considerable amount of uncertainty and ambiguity in the system environment, boundaries, and stakeholders' interests. In the case of Voice Notes requirements where changing all the time during development and even after the trial it was not sure, which were the stakeholders' (in this case the users') actual interests although they were sure that they will come up with them (see ARKI Thematic Publication 2, 2005 [Leh05a]). Furthermore, there was uncertainty related to the application platform (Zope and Zope CMF). It was not certain that the application was realizable as planned. Finally, we found out that there existed a "well documented" bug that we had to overcome with a workaround. Therefore, in order to succeed with experiments such as the Voice Notes, instead of considering that one has perfect knowledge at the start, one should design the used processes to permit continual questioning and reframing of problems and missions [Kea03].

Second, *building system transformation capability is more important than initial deployment* because initial deployment of a system of systems is always going to be wrong [Kea03]. In the case of the Voice Notes, it was crucial to have the minimal level of "partial solution" deployed to satisfy stakeholders' primary needs and to get the trial ongoing. To succeed we have to consider possible strategies for migration from one version to another. Therefore, speed of deployment and engineering capabilities for flexibility and system reconfiguration deployment were considered far more important than the solution's completeness. This resembles implications of system of system engineering [Kea03]. Also rapid identification of systems failures [Kea03] rose as an important aspect. For one of the participants a wrong version of the QuickTime Player was installed because of her relatively old computer system. Rapid identification of failure and reconfiguration of her computer system enabled her to participate in the trial.

Third, *bringing context in the foreground and technical solution to the background* determine success in the system of systems environment [Kea03]. We faced some contextual issues in a small scale for example when we had to solve the participants' concerns about cost of using the Voice Notes. In order to solve this we had to find out the organization's politics and policies about paying participants' phone calls. Inability to solve these would have doomed even the "best technical solution": the participants were not willing to pay for the use of Voice Notes yet; only later they wanted to have the Voice Notes for their own use and they were ready to pay for its use. Another issue that demonstrates the importance of contextual issues is that although the community members were excited about the Voice Notes and the co-design process was a success the first trial did not go as expected. The members of the community hardly used the web interface. They were excited about phones that they were carrying to all places and about possibility to make a note without pair of glasses, pen and paper, and in almost any light, and especially that they did not have to remember to send the note by email to others. The problem culminated to the usage of computer. The members of the community had mainly slow Internet connections and they were not willing to pay more for faster connections because they used the Internet only for reading their emails once or twice a week. They did not have a *practice* were checking voice notes could have been embedded. Again, the best technical solution can be doomed by contextual influences. One cannot expect to change people's practices over night. It is possible if contextual issues support the change but people cannot be expected to redesign their lives all over only because of some technical gadget. Therefore, the context should be brought to the foreground.

Fourth, *effectiveness in system of system engineering environments is determined first as a function of worldview*, or philosophy, which is critical in determining success in system of systems environments [Kea03]. This not only holds in the case of Voice Notes but it is also the most important point of convergence with systems of systems and our case. It opens a view from application development to the development of our development environment and *vice versa*. When the development of Voice Notes was considered, it was necessary to see the Voice Notes role in the "big picture", in the relation to our co-design efforts. The Voice Notes was not only seen as a separate application or an experiment, but something that would become an integrated tool of the community's collaboration environment and its practices and our ArkiWorks application platform. Therefore it was considered important that it would be possible to handle voicenotes as any other contents (documents) of the collaboration environment without need to switch between applications. The Voice Notes was not just a one-time application but it became a part of the application platform *and* the application development environment of the ARKI group (ADEA). For example, Voice Notes was used as a platform for other applications (e.g. *Vaakku* [Leh05b], a soft toy that can be used to send audio messages to parents' e-mail) and one of the solutions that were implemented in it will be developed further as a building block for other applications.

In summary, in the ADEA application development tracks do not create only applications but also the developmental and operational environment for themselves and other applications. In this, the ADEA resembles a system of systems having the characteristics of complex systems. In following, I analyze in more detail the characteristics of ADEA.

4.2 Characteristics of Application Development in ARKI

The ARKI research group, having 10–15 people, is the size of a small or medium firm. Apparently, even if the research partners, companies and communities, were included, as an organization it would still not be an enterprise, not even a small one. But the application development environment of the ARKI research group (ADEA) can be considered an enterprise in small. The key issue is not the size, but the characteristics.

Application development tracks in the ARKI are usually independent having one or two designers working on them. The development tracks of the research projects share a common development environment with each other but they are responsible only for the research project, seldom for the whole group. The reason for this is that software development is financed by the research projects, not directly by the research group. Thus, the ARKI can be considered to have several internal independent development tracks. See Table 4.

Development Track	**Project**
Online collaboration environment: ARKI	ADIK, Encompas, Mediaspaces...
Dissemination environment: Encompas	Encompas
Online community environment: Aktiiviset seniorit ry	ADIK
Voice message sharing application: Voice Notes	ADIK
Media sharing basket: Kori (MediaFolder)	Encompas
Video editor: Cutter	Mediaspaces
Community calendar	ADIK
Home server (planned)	ADIK, Encompas
Family blog (planned)	Encompas
Audiovisual web blog, e.g. mlab.tv	Mediaspaces
ArkiWorks application platform	A joint effort of several projects: ADIK, Encompas, Mediaspaces…

Table 4. The development tracks of the ARKI Research Group. (The table is not comprehensive but only illustrative.)

If external digital products and applications that are utilized in the ADEA are also included, the number of developmental tracks multiplies. See Table 5.

Product	**Vendor**
Mac OS X Server *operating system*	http://www.apple.com
Subversion Version control system	http://subversion.tigris.org
Python *programming language*	http://www.python.org
Zope *web application server*	http://www.zope.org
CMF *content management extension for Zope*	http://www.zope.org/Products/ CMF/index.html
Plone content management extension for Zope	http://plone.org
exUserFolder *authentication extension for Zope*	http://sourceforge.net/projects/exuserfolder/
MySQL *relational database*	http://www.mysql.com
Mail Boxer *emailing extension for Zope*	http://www.zope.org/Members/mjablonski/ MailBoxer
ZWiki Wiki for Zope	http://zwiki.org
Answering Service	Telecom Operator τ
QuickTime Player *media player*	http://www.apple.com
Browsers (Firefox, Mozilla, Safari, Internet Explorer...)	Different vendors
Phones (Nokia 7710, Nokia 6630, Nokia 3330...)	Different vendors
Multimedia Messaging Server	Research partner α
Authentication Server	Research partner β

Table 5. Third parties' products in the ADEA. (The table is not comprehensive but only illustrative.)

Some of the above-listed development tracks build on each other (e.g. Zope on Python, Zope extensions on Zope, and we build on the top of all them). However, in most cases the products are not built for the same purpose and they do not share the same conceptual basis (this is especially true between our and third parties' products) or funding which could be directed to solve problems. They can be considered analogous to independent organizations inside an enterprise. For further affirmation for the enterprise and complex system characteristics of the ADEA one can compare the characteristics of the ADEA to the characteristics of complex systems presented in Chapter 3.1.1:

The structure and behavior of a complex system is not deducible from the structure and behavior of its component parts.

The behavior of ADEA is not well known and its desired change is even less known. There have been many past projects that have left their traces to the ADEA and many ongoing projects that change the behavior of the ADEA according to their needs. Not until through this research the overall development of the ADEA has been considered. Previously there have been only independent application developments without a clear connection, and unfortunately because of inadequate understanding of objectives and lack of technical competence, even development of similar applications (e.g. online environments) have forked into separate versions that have unique and intricate implementations. Understanding the behavior of ADEA is hindered more by the fact that the ADEA is in constant change due to new projects, which stretch its boundaries and force to take advantage of its capabilities to limit and explore new ones in order to fulfill missions and experiments that projects should conduct. Even if one had the complete architecture of the ADEA, the presence of autonomous agents (independent development tracks) would make it impossible to infer its behavior: necessary changes are made to applications as required because fulfilling missions is the first priority. Therefore, the ADEA emerges rather than is consciously developed. For these reasons, I believe that at the moment no one has full understanding of the ADEA. On the other hand, by re-architecting, re-designing and re-factoring, and through innovation the complexity of the components of the ADEA could be reduced and the structure and behavior of ADEA could be made more understandable. However, because of the objectives set for the ADEA (open system that is constantly adapted) and the organizational structure of ARKI (development driven by the independent research projects), it is unlikely that the complexity of the ADEA as whole could be reduced so that its structure and behavior could be deduced from its component parts, although reducing the complexity (or rather intricacy) of its parts would be possible.

The elements of a complex system can change in response to imposed "pressures" from neighboring elements (consequently leading to reciprocal and transitive implications)

Independently introduced applications cause direct "pressure" on those applications which perform similar tasks, or which could potentially act in concert with these introduced applications. For example, changes in Media editor application or in Voice Notes cause "pressures" to online environments acting in cooperation with them. Another example is introducing a new collaboration environment or web site version for some of the collaboration communities. This causes social and political "pressures" to develop the older collaboration environments and web sites of other collaboration communities in order to keep these research partners content. On the other hand, this would not be a problem if all the online environments were based on the same source code. But as it was stated above for historical reasons all the online environments have unique implementation to an extent that they can be considered having their "own life". Naturally, the most obvious answer to this problem would be re-architecting, re-design and re-factoring of the online environments. However, this suggestion does not take into account contextual issues and therefore it is based on fallacy that the change could happen in a very short time. Eventually unifying the source base of these online environments has to be addressed through the disciplined inquiry and rigor of more or less traditional systems engineering because the intricate characteristics of current implementations halt the evolution, but until then they are "legacy systems" that we just have to deal with. However, when we have brought together these forked online environment versions, we still must face the fundamental "pressures", which are imposed on us as our collaboration communities develop and change their everyday life practices and demand the systems to be adapted to their needs. This concerns all of our applications.

A complex system has a large number of useful potential arrangements of its elements.

As the development tracks of ARKI (Table 4) shows the elements of ADEA provided by third parties (Table 5) have already many useful arrangements. However, there are not as many useful arrangements of products provided by us as we wish for. The reason to the situation is that we have not yet produced many applications (current projects have lasted approximately for one and half year). However, in the future there should be more systems of systems such as Voice Notes. Furthermore, these applications will be integrated with each other, e.g. Voice Notes with Community calendar and Family blog. This gives us reason to believe that the number of useful arrangements of ADEA will increase, which leads us to the next characteristic.

Given a steady influx of energy (raw resources), a complex system increases its own complexity.
The tasks of ADEA are numerous and in flux due to new and previous projects. The projects are enabled by raw resources (money) flowing into the system. The projects introduce new applications and thereby increase the number of possible connections and relations of the elements (complexity) of ADEA.

A complex system is characterized by the presence of independent agents.
In the case of ADEA the independent agents are best understood first as people and second as independent development tracks formed by people. The ARKI research group cannot be viewed as a monolithic entity moving in a pre-specified direction. Rather the ARKI research group is a loosely coupled collection of individuals, which aim to achieve something common. These people, designers and researchers, work more or less independently on different applications and in collaboration with research partners forming development tracks with them (see Table 4). Further, the ADEA includes third parties' digital products and applications (see Table 5). Producers of them can also be considered as independent agents of ADEA. Their decisions (that are independent of us) affect through artifacts that they produce and we import to the ADEA.

Given the presentation of the characteristics of our development environment, I proceed to analyze the suitability of traditional systems engineering and system of systems engineering to develop it.

4.3 Applicability of Systems Engineering Methodologies

In Chapter 3.2.2 I presented four *boundary conditions* for applying traditional systems engineering (TSE) methodology and six *primary conditions* that suggest a system of systems engineering (SoSE) methodology preferable to TSE. In the following I analyze first if the application development environment of the ARKI research group (ADEA) fulfills the boundary conditions for applying TSE successfully and then if the ADEA fulfills some of the primary conditions to consider using a SoSE methodology.

4.3.1 Applicability of Traditional Systems Engineering

The four boundary conditions for applying TSE successfully are according to Norman and Kuras:

1) The specific desired outcome must be known a priori, and it must be clear and unambiguous.
2) There must be a single, common manager who is able to make decisions about allocating available resources to ensure completion.
3) Change is introduced and managed centrally.
4) There must be interchangeable resources, which can be applied and reallocated as needed.

First, in the case of the ADEA neither the desired outcome nor the boundaries of the system are very clear. The ADEA is a research "project" and its boundaries as well as its outcomes are constantly in change. The outcomes and boundaries are clarified gradually but not known *a priori*.

Second, in the case of ADEA there is not a single, common manager who could make decisions about allocating available resources. Inside the ARKI research group this is possible to the extent that projects allow it but "parts" of the ADEA are also scattered outside the ARKI. There is not a way, for example, to force financially or by other means the developers of Zope products to develop their products into directions that ensure completion of our projects.

Third, inside the ARKI it is possible to introduce and manage change centrally to some extent but also inside the ARKI the projects must move forward and independent development tracks should be able to introduce changes and bring new products into the ADEA according to their needs, e.g. new Zope products. ("Should" because of the interdependencies of products this is not currently always possible.) This way change is introduced in many places and even outside the ARKI.

Fourth, in the case of the ADEA there are not resources that could be applied directly or reallocated as needed to develop the ADEA. The development of the ADEA can happen only gradually by independent development tracks driven by the needs of the research projects.

Based on this analysis, it is reasonable to believe that applying TSE approaches to the ADEA involves risks and the outcome might not be as successful as desired. Further, this encourages analysis of advantages of system of systems engineering to approach the development of ADEA.

4.3.2 Preference of System of Systems Engineering

The six primary conditions that suggest a system of system engineering (SoSE) methodology preferable to TSE are according to Keating et al.:

1) *Turbulent environmental conditions*
2) *Ill-defined problem conditions*
3) *Contextual dominance*
4) *Uncertain approach*
5) *Ambiguous expectations and objectives*
6) *Excessive complexity*

First, the ADEA as an environment for systems engineering effort is considerably dynamic, uncertain and rapidly changing, because of independent development tracks. Second, the turbulence in environmental conditions is further increased by ill-defined problem conditions. There is no sufficient consensus for initial problem definition, functional or qualitative requirements of the ADEA. Third, in the development of ADEA contextual issues overshadow the technical aspects. Technologies enable almost everything but the success of engineering the ADEA is primarily determined by adequately addressing the contextual problem drivers (human, organizational, policy and political constraints). Fourth, the path of progression on how to "best" proceed with the development of ADEA is indeterminate. Standard systems engineering processes are at least highly suspect for adequately addressing the development of ADEA (see Chapter 4.3.1). Fifth, because of our inadequate understanding of objectives and lack of technical competence to proceed with the development of ADEA, we are not able to establish measures of success or objectives for the systems engineering or at least our objectives, e.g. "an open system that is continuously adapted to everyday life uses", are vague from the traditional systems engineering perspective. Sixth, the boundaries of ADEA are such that its complexity is beyond pre-specification and centrally managed development, and thus beyond capabilities of TSE. We cannot either proceed by simplifying our objectives: from the TSE perspective they are too vague and changing them would baffle our endeavor.

Fulfilling all the six primary conditions in some extent it seems reasonable to conclude that we should seriously consider using a system of systems engineering methodology to the development of ADEA. In the next chapter I view the preference of proposed system of systems engineering methodologies.

4.4 Preference of System of Systems Methodologies

The ADEA is a developmental environment in a real sense. Therefore, complex systems engineering, which emphasizes creating and nurturing *developmental environment*, seems more than appropriate for our purposes. However, the most important issue that makes CSE preferable to the SoSE methodology by Keating et al. is the presence of autonomous agents. The ADEA emerges from the efforts of independent development tracks. As system of systems engineers we can only *indirectly* bring order to the ADEA. We should do this by continuous characterizations of outcome spaces, stating rewards that can be in our case only indirect (desirability of that what is produced for users and us), defining developmental precepts and safety regulations from the co-design perspective, and judging outcomes. As an *initial response* to engineering the ADEA, in following I propose how CSE tactics proposed by Norman and Kuras might be applied in the case of ADEA.

4.4.1 Applying Complex Systems Engineering Tactics

In this chapter, I review briefly applying CSE tactics to the development of ADEA. The tactics have not been applied yet, and thus the purpose is only to illuminate how these tactics may unfold in the case of the ADEA.

Half-life Separation

Half-life separation is the closest of the CSE tactics to software design principles. It is applicable as such and we should apply it whether the CSE approach is used or not. In practice this means following solid software design principles (e.g. Bus96) in order to support modifiability and re-usability. New components should be backward compatible in order to support components changing in lower pace. For example, the ArkiWorks application platform should be designed so that it is composed of replaceable components.

Another architectural issue that should be concerned is *data independence*. It could be considered to constitute a sub-practice of half-life separation. The data should be considered as a "component" that never changes. Therefore, the data architecture should be designed so that the "raw" data (including metadata) is independent of the applications used to access it. No application should claim ownership of the data but it should be considered as the property of users, not applications. This is feasible using layered object-oriented (component) architectures that hide the data layer and at the same time enable application specific data types (objects). On the other hand, achieving data independence in general might require co-opetition (see below).

Technical Approaches which Respect "Ricebowls"

The Voice Notes application is an example of using a technical approach that respects "ricebowls". The Voice Notes uses a web service (answering service) to explore new associations and relations without interfering to the operation of the telecom organization. Technical approaches which respect "ricebowls" are something that we should discuss with our research partners in order to facilitate systems development but they are something that we could also consider in the case of, e.g. peer-to-peer networks of home servers providing services for other households.

Advertising and Discovery

Advertising and discovery are quite well applied in the Zope community[1]. There are lists of available Zope products[2] and means to introduce new ones. However, there is a problem in comparing the products and choosing the most appropriate one. Better *social recommendation systems* could ease finding potential products but finding the most suitable product is not the only problem. After finding the most useful capabilities and functionalities one is encountered with integration problems. These problems could be reduced and discovery facilitated by having a *test bed* in our environment where the study of these products is easily and safely done.

Collaborative Environments

In the ARKI, collaboration takes many forms already at the moment. There is collaboration between the ARKI group and practice-partners (communities) and research partners (companies) and collaboration between developers. The ARKI is a collaboration environment. However, there is need to communicate more clearly how independent developments fit into the "big picture" in order to reduce integration risk. In addition, collaboration with developers outside the ARKI (Zope community) should be considered because it might reduce overlap and improve understanding of our true need. However, this would require understanding the practices of the Free/Open Source Software community and involvement in the Zope community's everyday work (see e.g. Tuo02). We could also consider establishing a Finnish Zope group, a collaborative environment, for sharing information and solving problems.

Partnerships

The collaborations with developers outside the ARKI (Zope community) could also take a form of partnership. In this way it could be possible to improve understanding of the goals of the ADEA (i.e. the objectives of co-design), spread development risk and reduce overlap.

[1] http://zope.org [17.5.2005]
[2] http://www.zope.org/Products/ [17.5.2005]

Developers Networks

We are already connected to a developer network because of our earlier decisions. We have chosen a specific programming language, Python, because of its "programmer centered design", and a specific application platform, Zope, because it is mainly implemented in Python. The Zope company, on the other hand, has been successful in creating developers networks (community) and getting developers to use a specific platform. We have benefited from all that. Thus far we have only been receiving but perhaps we should consider also contributing in order to create partnerships. This would require productizing our applications.

Co-opetition

In the ARKI research group applying co-opetition is not applicable because of our limited resources but in the Zope community this already is "applied". Zope unites independent parties (developers) to cooperate on those elements and aspects, which transcend their individual ability to control (Zope platform) while preserving their ability to compete on demonstrated value in their space (diverse Zope extensions).

Leveraging Others' Investments

The ADEA is built mainly on others' investments. However, we could probably manage to leverage others' investments and to build even less ourselves. In order to succeed we should define precisely what kind of functionality and behavior we are ready to reward.

Branding

The development of co-design philosophy could lead to branding products as "Co-design Compatible". "Co-design Compatible" or "Ready for Co-design" stickers would ensure that the product is open, adaptable, interoperable, and easy to integrate with other "Co-design Compatible" products. At the moment this is more an idealistic goal than applicable practice because applying this tactic would require specifying *exactly* the requirements that qualify a product as "Co-design Compatible".

Opportunistic Approach

Our approach is already opportunistic. One of the basic ideas of the co-design philosophy is to involve users in managing the identification and introduction of functionality and change. Therefore, we work in close relationship with our collaboration communities and develop applications based on almost daily feedback.

Permitting "value-add" business models

The idea of co-design is to give users "a vote" in what is built for them. The research projects and software developments are given directions based on the user feedback. Researchers and designers "get paid" for composing something that the users are ready to use. Hence, success is directly related to the usefulness of that which is produced. However, co-design differs from the suggestion by Norman and Kuras [NoK04]: co-design is a design approach that aims at solving desirability of functionality *a priori* although it leaves room for discovery.

Experience for Testing

We prefer experience for testing for practical reasons. Naturally applications are black-box tested offline as well as some usability tests are carried out. But we are not able to perform full-coverage testing because the users' environments (computers and software) are heterogeneous and our resources are limited. On the other hand we are interested in experiences that people have when using the products in real life situations. Therefore, in general applications are tested in the field. Instead of attempting to come as close to full-coverage testing as possible, we prepare to solve problems when they occur.

One unique characteristic of the ADEA is that it is mainly built "on the field". It is a combination of production and development environment. Therefore, some changes cannot be tested before delivering them. In order to have safer and less stressful (more efficient, effective, satisfactory, i.e. usable) development environment for developers, the production and the development environments should be separated in the extent it is possible. At least changes should be possible to experiment safely before introducing them. Taking into use an issue (a.k.a. bug) collector and error logs for reporting, cataloging and monitoring failures (instead of just mailing lists) would further improve our capabilities to recover when things go wrong.

Playgrounds

The concept of *playground* captures the goal of our efforts: the main purpose of the ADEA is to be a playground, where our experiments can take place. But it is also supposed to evolve to a prototype of "playground" where people can innovate and discover new ways to take advantage of technology and adapt it to the practices of everyday life uses. The applications of CSE tactics above are initial steps on a path transforming the current ADEA to a playground. However, they are not enough. Because the ADEA emerges as a result of application developments, which form a system of systems, in following I review in addition to above-presented tactics system of systems engineering implications for application developers.

4.4.2 Implications for Application Developers

To chart system of systems thinking implications for application development one could take a look at the system of systems engineering implications for single system developers presented in Defense Acquisition Guidebook (DAG) by the United States Department of Defense [Uni05]. Rephrased it says that systems or applications should not be developed as stand-alone systems, but as parts of larger meta-systems delivering unique and encompassing capabilities. Developers should be aware of the distinguishing system of systems engineering attributes that might apply to their system and the possible impact on their system architecture. DAG also presents a list of questions (rephrased below) that developers should ask themselves to address system of systems concerns, capitalize on system of systems capability pay-offs, and effectively meet the design and development requirements of current and future system of systems:

1) Will the experimental capabilities of the ADEA improve if the ARKI research group incorporates my system into the portfolio of existing and planned systems of systems?

2) What additional capabilities and behavior could my system deliver within the context of existing and planned systems of systems?

3) Which are the most valuable capabilities that other systems can provide to my system if it becomes a part of existing and planned systems of systems?

4) To which systems of systems can my system contribute the most value?

5) Are there system of systems capabilities, behavior, and requirements that my system must address to become part of the existing and planned system of systems?

6) Am I designing my system so that it can be easily integrated with other systems?

7) Does my system have an adaptable and open architecture to enable future reconfiguration and integration into a system of systems?

8) Have the system of systems interface requirements been adequately defined and documented in the specification of my system?

9) Has my project developed and documented interface control requirements for external functional and physical interfaces?

10) Has my project identified and established conformance testing or certification mechanisms to assure that standards used by external interfaces conform to the prescribed interface specifications?

11) Has my project verified the external functional interface specifications to ensure that the functional and performance requirements for such interfaces are satisfied?

12) Does my system fully comply with external interface requirements identified through the ADEA Integration and Development process and its accompanying documents and architectures?

13) Have I established rigorous interface design and management based on conformance and verification of standards at upper layers as well as at the application, transport, network, physical, media and data link communication layers?

Evidently this list of questions can be used only as a guide. We do not have defined interface specifications, integration and development processes, or system architecture. We are only at the beginning of our endeavor. Eventually interfaces, processes and architectures are to be defined but they (the order of the system) will emerge only gradually through *research-in-action*, establishing knowledge concurrent with application and problem resolution.

4.5 Summary

In the case of ADEA, we do not have the luxury of doing research and then moving to action. Therefore system of systems engineering, especially CSE offers us valuable perspective. We must establish knowledge concurrent with application and problem resolution. Literature and observations that I have presented suggest that to succeed in developing the ADEA toward an open, adaptable and interoperable system of systems requires something beyond tools and techniques. To succeed, it is important to inculcate the "systems perspective". Fundamentally, it is a question of compatibility between the supporting philosophy and the system of systems engineering approach. In our case the supporting philosophy is co-design and the most promising candidate for a system of systems engineering approach is the complex systems engineering. However, obviously this is not enough for engineering purposes. The "model" of the ADEA that I have presented is rather a *qualitative* than *quantitative* model and as such, it has little use for engineering. In order to measure our success we need metrics and for metrics we need something quantifiable. This requires a *reductionistic* approach, the decomposition of system into its parts in order to isolate variables that uniquely determine the state of the system. However, the reductionistic approach, especially in the case of modeling of complex systems, is not without potential for distortion, errors, and *oversimplification* [Dea04]. The system model is always dependent on knowledge and viewpoint of the person who models the system, and therefore there is no “perfect” *true model* of any system [Dea04]. However, as more knowledge of the diverse actions and interactions of the parts that comprise the system are gained, it is possible to develop a more accurate *approximation* of the true nature of the system [Dea04]. However, because the system model must also address the needs of the individuals who express interest or concern for the performance of the system under model it is probably necessary to develop several models to capture the complexity of the system [Dea04]. This concerns also ADEA. Thus, the “model” that I have presented should be considered as a first iteration model, whose primarily purpose was only to argue that it is reasonable to view the ADEA as a complex system on the way toward more quantifiable models that captures evolutionary and multiscale reality of the ADEA as a complex system.

5 Discussion

At the first sight complex systems engineering (CSE) seems to concern totally different kind of systems than what we are interested in our co-design research: our concern is digital products that people use in their everyday life whereas CSE is concerned how to engineer and manage the *enterprise*. But if one takes a look at the ADEA or the digital environment where people live today, the similarity becomes apparent. We are surrounded by a large collection of independent and loosely associated products, *representatives* of organizations, which we try to integrate to our "systems of systems" in order to manage our everyday life and social relationships.

Understanding the digital products, software, as representatives or *representations* of organizations that produce them is especially feasible because of the almost infinitely flexible nature of software and the existence of computer networks, which makes it possible to update software continuously, thus enabling the products to represent the state of the organizations instantly (see Figure 9).

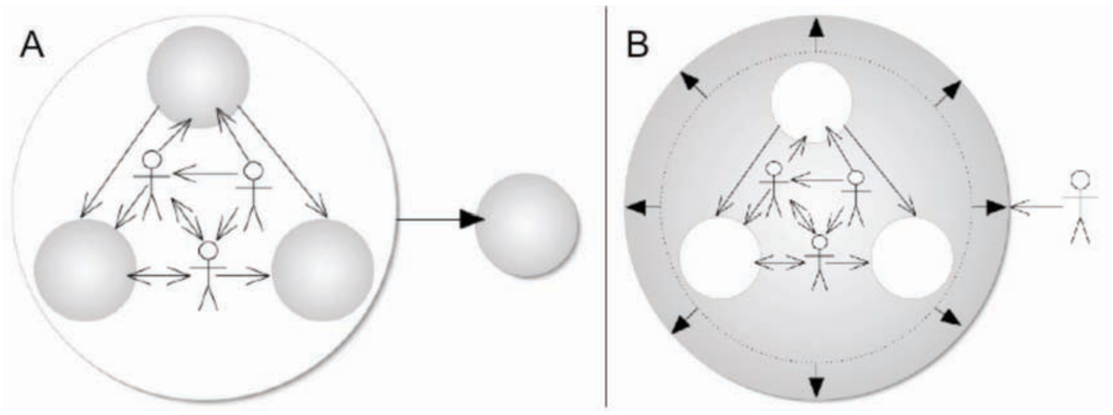

Figure 9. "A" presents a common view of product development where an organization (people and systems) produces a product (e.g. a word processor). "B" presents a view where the product represents the organization that produces it to the user.

Based on the view that the products represent the organizations that produce them, the system of products exposes not only as a system of systems but as a system of loosely coupled organizations (see Figure 10).

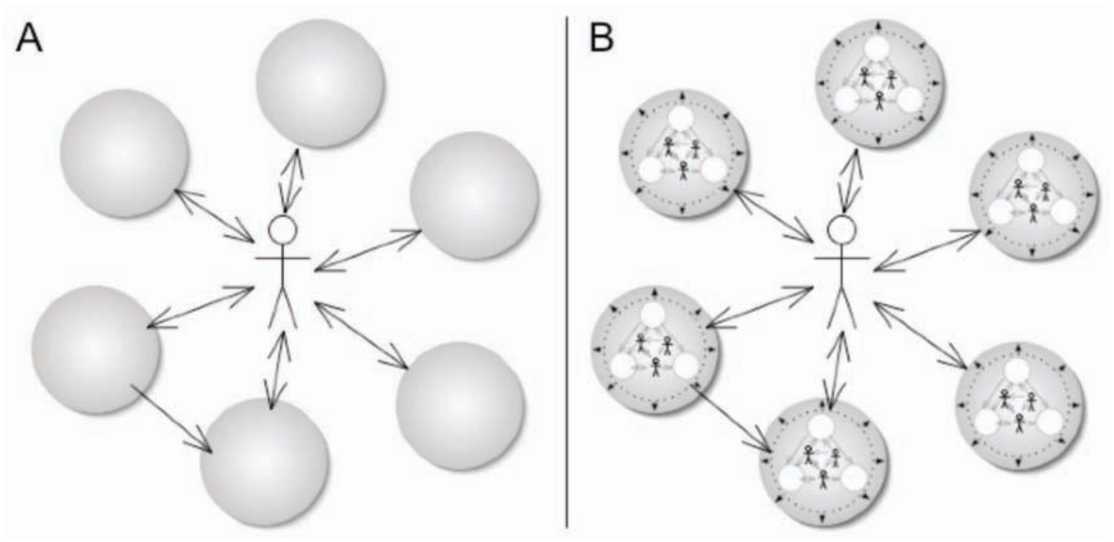

Figure 10. Based on the view that the product represents the organization that produces it, the "system of systems" of everyday life (A) exposes itself as a system of loosely coupled organizations (B).

After adopting the view of the inherent complexity of the "systems of systems" of everyday life the other points of convergence between CSE and co-design are more than obvious:

Co-design and CSE both are based on a view of systems of digital products as having the characteristics of an *ecosystem*.

Co-design and CSE are both emerged from the notion that traditional approaches to system engineering are having trouble dealing with the systems having the characteristics of an ecosystem.

Co-design and CSE both seriously take the environment surrounding the systems. CSE sees complex systems as being alive and constantly changing; responding to and interacting with their environments – each causing impact on and inspiring change in the other [NoK04]. Co-design concerns the social system (and its design) as integral part of the development process and its outcomes.

Given these points of convergence it seems reasonable to argue that co-design and CSE are compatible. Further, there seems to be synergy in combining these two approaches: whereas complex systems engineering provides a conceptual framework

to manage the evolutionary development of system of systems, through co-design the outcome spaces are possible to characterize and need to explore paths that lead to dead-end is reduced.
In addition, applying complex systems engineering in combination with co-design could benefit the development of the digital environment of everyday life toward interoperable systems of systems. Together these could form a comprehensive view to the development of digital environment of everyday life as it is argued in following.

As it was pointed out, based on our understanding the challenging usability problem of digital environment of everyday life is that people are forced to act as "systems integrators". However, from our point of view the problem is not that people have to integrate products but that they cannot or they have to "integrate" the products most of the time they are using them because there are not enough possibilities (e.g. open interfaces) for people or third parties easily to compose independent products to systems of systems. Fortunately, there are already some developments, which support systems of systems of everyday life and suggest that system of systems thinking is rising, although some of them advocates systems of systems only from technical perspective.

The first of the systems of systems supporting developments is AppleScript by Apple. AppleScript was long treated by Apple itself as unwanted and it has even (according to a legend) at times come being abandoned but has been embraced and acknowledged lately [Neu03]. AppleScript enables integration of applications by letting the applications use each other's functionality but unfortunately AppleScript is available only for the Mac platform and taking advance of it requires programming skills, so anyone cannot use it. Second of these developments is the *semantic web*[1]. It also facilitates development of applications of applications (see e.g. HBM03, Con04) by facilitating integration. However, these developments are more technological facilitators than "worldviews" for guiding development of products. The third development is the Free/Open Source Software movement[2]. By advocating free access and redistribution of software it makes possible to adapt systems to everyday life practices and integrate them to a system of systems. Unfortunately this is possible

[1] http://www.w3.org/2001/sw/ [25.4.2005]
[2] http://www.opensource.org [28.4.2005]

only for technically skilled people. On the other hand a community of people is able to overcome this obstacle: skilled people may share their achievements with others. Together people produce something greater than the sum of the individuals. Finally, there is research activity on approaches to the *whole system design* (e.g. NeS03). One of them is our proposal for co-design philosophy.

By viewing the everyday life as whole – or using the terms of complex systems engineering as a system of systems which include technological systems and social systems, as well as political systems, and their relations, co-design aims to counter the interoperability and integration problems of everyday life digital environment. In a way the co-design can be considered to be about "ecosystem usability" as the complex system engineering can be considered to be about "ecosystem engineering". Together these approaches may form a comprehensive approach that could address the problem of "usability engineering of digital ecosystem". It would view usability as a function of the digital environment of the everyday life, the complex of systems and their relations as having the characteristics of an ecosystem, and would improve usability of everyday digital products by facilitating (co-)design and removing obstacles that hinder collaboration (and competition) of organizations to provide answers to the usability challenges of present digital environment. In order to get into the situation where separate applications could be composed to systems of system, we should consider products as the representations of the organizations that produce them (see Figure 11). The interactions needed to provide capabilities for integrating applications should take place mainly during design time not in use time by the users but the same time the users should be enabled to do the finalization of the design, the integration of preferred applications to desired system of systems.

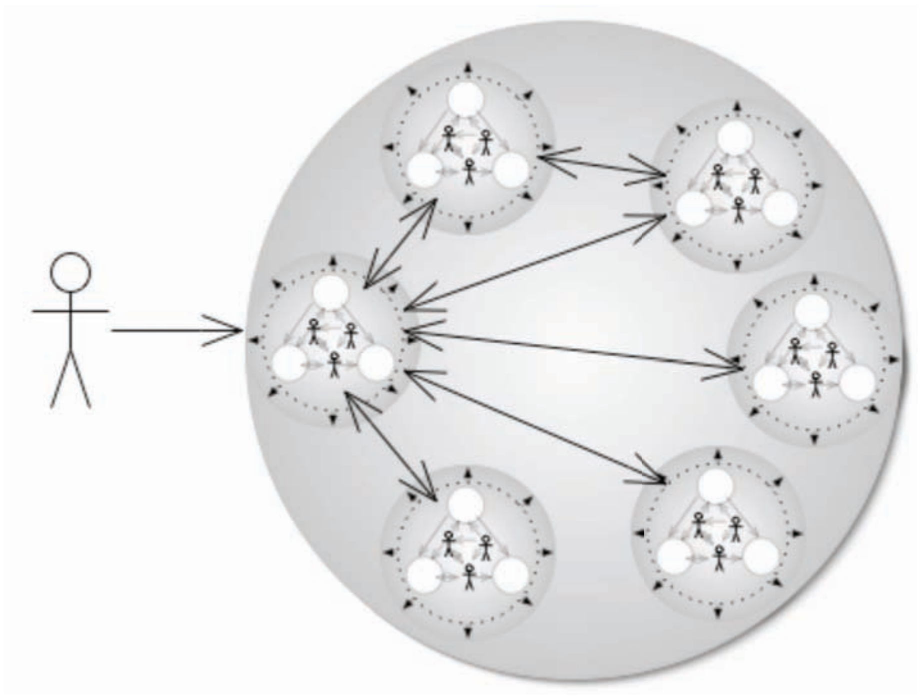

Figure 11. Interactions that provide capabilities for an integrated "product" should take place on organization level during the development time and should not be provided by the user e.g. copying and pasting and sending files back and forth.

6 Conclusions

At the beginning of this research I set my objective to study applicability of system of systems engineering approaches to engineering the ARKI research group's application development environment (ADEA) and their compatibility with our proposal for co-design. In this thesis I have shown that the system of systems engineering approach, complex systems engineering (CSE), can be considered compatible with our proposal for co-design and applicable to the engineering of ADEA. The applicability of CSE to engineering the ADEA I have justified by showing that it is reasonable to view the ADEA as a system of systems having the characteristics of complex systems. Based on those characteristics I have argued that the CSE approach is preferable to other systems engineering approaches to the development of the ADEA. In addition, I have suggested concrete applications of the CSE tactics presented in the literature to engineering the ADEA as well as presented implications of system of systems view to single systems developers.

Further, in this research I have argued that the development of everyday life applications might benefit from the system of systems perspective and the combination of CSE and co-design approaches in general. In order to improve further usability of the applications of everyday life it seems that there is need for a more comprehensive approach to the application development than the approach of traditional application development. In this respect the concept of system of systems seems to provide valuable perspective. However, I have pointed out that obtaining the system of systems perspective does not require that single system developers should provide systems of systems but rather that they should provide capabilities, open interfaces for developing systems of systems of everyday life. In addition I have argued that complex systems engineering offers only a methodology to develop these system of systems. Obviously complex systems engineering cannot reveal the practices, *patterns*, of everyday life and emerging needs, pressures, for developing specific systems of systems. For directing the complex systems engineering efforts of everyday life application development a practice-centered approach is needed in order to find new applications. The ARKI research group proposal for co-design is one proposal for such an approach.

The first topic for future research is studying the modeling methodologies to develop a quantifiable model of the ADEA. This will take place as a natural part of the future *research-in-action* development and continuous characterization of the ADEA.

The second topic for future research is the methodology and practices of complex systems engineering and their utility in practice to engineering the ADEA and the everyday life systems of systems. In some extent CSE is already practiced in consumer markets (as suggested CSE tactics indicate) but more conscious efforts to take advantage of system of systems thinking and collaboration of companies might benefit all parties. This research could take place e.g. in European Union's joint research projects.

The third topic for future research relates to the observable phenomenon of *digitalization* that is, transforming electronic (analogical) devices into digital devices, computers (and information used in and used by them into digital bits, and communication channels into digital networks). The computer is special as the first "meta-medium" [KaG77] that can be programmed to function like any other electronic device, but the essence of digitalization is not the computer or the network; it is software. Software, on the other hand, is almost infinitely flexible. It can be programmed and adapted over and over again to do almost anything to the extent that resources permit. The complex systems engineering approach presented in this thesis has left practical software design decisions for independent development tracks. However development could be facilitated if some common principles could be defined. In order to design open systems that are adapted to people's everyday life practices we need more qualified software than ever to guarantee interoperability and ease of integration and continuous adaptation. Hence, one topic for the future research is the characteristic of *building blocks*, their conceptual basis and architectural and software design principles which would guarantee them to be "co-design compatible".

References

AAP88	Anderson, P. W., Arrow, K. J., Pines, D. (eds.), *The Economy as an Evolving Complex System*. A Proceedings Volume (V) in the Santa Fe Institute Studies in the Science of Complexity, Perseus Books, Reading Massachusetts, 1988.
Abr02	Abrahamsson, P., et al., *Agile Software Development Methods*. Review and Analysis. VTT Publications 478, Espoo, 2002.
ACM92	ACM SIGCHI, *Curricula for Human-Computer Interaction*, 1992. http://www.acm.org/sgchi/cdg. [15.08.2004]
ADL97	Arthur, W. B, Durlauf, S. N, Lane, D. A. (eds.), *The Economy as an Evolving Complex System II*. A Proceedings Volume (XXVII) in the Santa Fe Institute Studies in the Science of Complexity, Addison-Wesley, Reading Massachusetts, 1997.
Ale79	Alexander, C., *The Timeless Way of Building*. Oxford University Press, New York, 1979.
Ash56	Ashby, W. R., *An Introduction to Cybernetics*. Chapman and Hall, London, 1956.
Bar97	Bar-Yam, Y. *Dynamics of Complex Systems*. Perseus Books, Boulder, Colorado, 1997.
Bar02	Bar-Yam, Y., Large Scale Engineering and Evolutionary Change: Useful Concepts for Implementation of FORCEnet. Report to Chief of Naval Operations Strategic Studies Group, 2002, http://necsi.org:16080/projects/yaneer/SSG_NECSI_2_E3_2.pdf. [26.4.2005]
Bar03	Bar-Yam, Y., When Systems Engineering Fails – Toward Complex Systems Engineering. *International Conference on Systems, Man & Cybernetics*, 2, 2003, IEEE Press, Piscataway, NJ, pp. 2021–2028.
Bar05	Bar-Yam, Y., About Engineering Complex Systems: Multiscale Analysis and Evolutionary Engineering. In Brueckner, S., et al. (Eds.): *ESOA 2004*, LNCS 3464, pp. 16–31, Springer-Verlag Berlin Heidelberg, 2005.
Bec99	Beck, K., *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts, Addison-Wesley, 1999.
Boe88	Boehm, B. W., A Spiral Model of Software Development and Enhancement. *IEEE Computer* 21, 5, (1988), pp. 61–72.
Boe02	Boehm, B., Get Ready for Agile Methods, with Care. *Computer* 35, 1, 2002, pp. 64–69.

Bot03	Botero Cabrera, A., et al., Codesigning Visions, Uses, and Applications. Paper presented at "TechnE Design Wisdom" 5th European Academy of Design Conference 5-EAD University of Barcelona, Barcelona, Spain, April 28–30, 2003, http://arki.uiah.fi/arkipapers/codesigning_ead.pdf [19.5.2005]
Bot04a	Botero, A., et al., Notes on Co-design Approach. Internal document, July 2004. (Not available)
Bot04b	Botero Cabrera, A., et al., *ARKI Thematic Publication 1, ADIK/ Remembering and Reminding*, 2004, (in press).
BrN05	Brandenburger, A., Nalebuff, B., *Co-opetition Interactive*, 1996/2005, http://mayet.som.yale.edu/coopetition/index2.html. [18.2.2005]
Bus96	Buschmann, F., et al., *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, West Sussex, England, 1996.
Coc02	Cockburn, A., *Agile Software Development*. Addison-Wesley, Reading, Massachusetts, 2002.
CoK03	Correa, Y. Keating, C., An Approach to Model Formulation for Systems of Systems. *IEEE International Conference on Systems, Man and Cybernetics*, 4, 2003, pp. 3553–3558.
Con04	Connolly, D., et al., *Semantic Web Application Integration: Travel Tools*, 2000/2004, http://www.w3.org/2000/10/swap/pim/travel.html. [24.3.2005]
Dea04	Dean, A., Modeling Complexity: Holism vs. Reductionism and the Potential for Distortion and Errors That May Occur Due to Either Diminution or Abstraction of a Complex System. *Proceedings of the 25th Annual Conference of American Society for Engineering Management*, October 20-23, 2004, Hilton Alexandria Center Alexandria, Virginia, pp. (not known). http://www.asem.org/conferences/2004conferenceproceedings/Dean017.pdf [20.5.2005])
FiG04	Fischer, G., Giaccardi, E., Meta-Design: A Framework for the Future of End User Development. In *End User Development – Empowering people to flexibly employ advanced information and communication technology*, Lieberman, H., Paternò, F., Wulf, V., (Eds.), Kluwer Academic Publishers, Dordrecht, Netherlands, 2004, (in press).
FiO02	Fischer, G., Ostwald, J., Seeding, Evolutionary Growth, and Reseeding: Enriching Participatory Design with Informed Participation. *Proceedings of the Participatory Design Conference* (PDC'02), Malmö University, Sweden, June 2002, pp. 135–143.

Fis01 Fischer, G., Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems. *Proceedings of the 24th IRIS Conference*, August 2001, Ulvik, Department of Information Science, Bergen, Norway, pp. 1–14.

Fis05 Fischer, G., From Reflective Practitioners to Reflective Communities. *Proceedings of the HCI International Conference* (HCII), Las Vegas, July 2005, pp. (in press). http://l3d.cs.colorado.edu/~gerhard/papers/reflective-communities-hcii-2005.pdf. [19.4.2005]

Fow03 Fowler, M., *The New Methodology*, http://www.martinfowler.com/articles/newMethodology.html. [12.08.2004]

GDM05 Gideon, J. M., Dagli, C. H., Miller, A., Taxonomy of Systems-of-Systems. *Proceedings CSER* (Conference on Systems Engineering Research) 2005, March 23–25, Hoboken, NJ, USA, Stevens Institute of Technology, (not published), http://www.stevens-tech.edu/cser/authors/36.pdf [20.4.2005]

HBM02 Hendler, J., Berners-Lee, T., Miller, E., Integrating Applications on the Semantic Web. *Journal of the Institute of Electrical Engineers of Japan*, 122, 10 (2002), pp. 676–680.

HeK91 Henderson, A., Kyng, M., There's No Place Like Home: Continuing Design in Use. In *Design at Work: Cooperative Design of Computer Systems*, Greenbaum, J., Kyng, M., (Eds.), Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1991, pp. 219–240.

Hey95 Heylighen, F., Joslyn C., Turchin V. (eds.), *The Quantum of Evolution. Toward a Theory of Metasystem Transitions*. Gordon and Breach Science Publishers, New York, 1995.

Hig02 Highsmith, J., *Agile Software Development Ecosystems*. Addison-Wesley, 2002.

Hob98 Hobday, M., Product Complexity, Innovation and Industrial Organization. *Research policy* 26 (1998), pp. 689–710.

Hol95 Holland, J. H., *Hidden Order: How Adaptation Builds Complexity*. Reading; Addison-Wesley, 1995.

IEE99 IEEE Standard for Application and Management of the Systems Engineering Process, Institute of Electrical and Electronics Engineers, Inc., New York, USA, 1999.

INC04 INCOSE Systems Engineering Handbook, Version 2a, International Council on Systems Engineering, 2004.

ISO98	ISO 9241-11, *Ergonomic requirements for office work with visual display terminals (VDTs), Part 11: Guidance on usability*, 1998, http://www.iso.ch/iso/en/ISOOnline.frontpage. [13.12.2004]
KaG77	Alan Kay, A. Goldberg, A., Personal Dynamic Media. In Wardrip-Fruin, N., Montfort, N. (Eds.) *The New Media Reader*, the MIT Press, 2003, pp. 391–397; originally published in *Computer* 10 (3), 1977, pp. 31–41.
Kau93	Kauffman, S.A., *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, New York, 1993.
Kea01	Keating, C. B., et al. *A Methodology for Analysis of Complex Sociotechnical Processes*, Business Process Management Journal, 7, 1, (2001), pp. 33–50.
Kea03	Keating, C., et al., System of Systems Engineering. *Engineering Management Journal*, 15, 3 (2003), pp. 36–45.
Kom01	Kommonen, K.-H., Digital Dimension. Concept paper, 16.4.2001. http://arkinen.uiah.fi/concepts/digitaldimension [2.9.2005]
Kom03a	Kommonen, K.-H., Arjen yhteisöjen digitalisoituvat käytännöt (Emerging Digital Practices of Communities). Project plan, 8.5.2003. (Not available)
Kom03b	Kommonen, K.-H., Codesign – the Emerging Design Strategy for a Digital Society. Presentation in the 6th Asian Design International Conference, October 10–17, 2003, Tsukuba, Japan. (Available on request)
Kom03c	Kommonen, K.-H., Notes About Building People and Life Oriented Visions of Future Technology. Paper presented at the Wireless World Research Forum 10th Meeting, October 27–28, 2003, New York, USA. http://arki.uiah.fi/mdr/mdr-files/ddis/Kommonen-NotesAbVisB.pdf [2.9.2005]
KSM03	Keating, C., Sousa-Poza, A., Mun, N., Toward a Methodology for Systems of Systems Engineering. *Proceedings of the 24th Annual Conference of American Society for Engineering Management*, October 15–18, 2003, St. Louis Missouri - Sheraton Westport, pp. 1–7.
Kuw00	Kuwabara, K., Linux: A Bazaar at the Edge of Chaos. *First Monday*, 5, 3, (2000), http://www.firstmonday.org/issues/issue5_3/kuwabara/index.html [3.9.2005]
Lar04	Larman, G., *Agile & Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.
Leh05a	Lehtimäki, K., et al., *ARKI Thematic Publication 2, ADIK/Voice Notes*, 2005. (To be published).

Leh05b	Lehtimäki, K., *Applying Tacit Craft Knowledge Onto the Design of Modern Everyday Products: Enabling Craft Students and Artisans to Experiment with the Design and Construction of Digital or Electronic Interfaces*. Accepted for Cumulus Conference, Lisbon, Portugal, May 26–29, 2005. (Available on request)
Mai96	Maier, M. Architecting Principles for Systems of Systems, *Proceedings of the Sixth Annual International Symposium, International Council on Systems Engineering*, Boston, MA, 1996, pp. 567–574.
Nar93	Nardi, B. A., *A Small Matter of Programming*. The MIT Press, Cambridge, MA, 1993.
NeS03	Nelson, H. G., Stolterman, E., *The Design Way: Intentional Change in an Unpredictable World: Foundations and Fundamentals of design Competence*. Educational Technology Publications, Inc., Englewood Cliffs, New Jersey 07632, 2003.
Neu03	Neuberg, M., *AppleScript: The Definitive Guide*. O'Reilly & Associates, Inc., 2003.
NoD86	Norman, D. A., Draper, S. W. (Eds.) *User-Centered System Design, New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1986.
NoK04	Norman, D. O., Kuras, M. L., Engineering Complex Systems, Mitre Technical Papers, January 2004, http://www.mitre.org/work/tech_papers/tech_papers_04/ norman_engineering/norman_engineering.pdf [23.6.2004]
Nor04	Norman, D. O., Engineering a Complex System: A Study of AOC, Mitre Technical Papers, July 2004, http://www.mitre.org/work/tech_papers/tech_papers_04/norman_aoc/norma n_aoc.pdf [25.4.2005]
PaF02	Palmer, S. R., Felsing, J. M., *A Practical Guide to Feature-Driven Development*. Upper Saddle River, NJ, Prentice-Hall, 2002.
RiJ00	Rising, L., Janoff, N. S., The Scrum Software Development Process for Small Teams, *IEEE Software* 17, 4, (2000), pp. 26–32.
Roy70	Royce, W. W., Managing the Development of Large Software Systems. *Proceedings IEEE WESCON*, 1970, 328–338.
ScN93	Schuler, D., Namioka, A. (Eds.) *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.
Sha48	Shannon, C. E., A Mathematical Theory of Communication. *Bell System Technical Journal*, 27 (July, October 1948), pp. 379–423, 623–656.

Sim69 Simon, H., *The Sciences of the Artificial*. The MIT Press, 1969.

TaF93 Taylor, J.C., Felten, D. F., *Performance by Design: Sociotechnical Systems in North America*. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

Tuo02 Tuomi, I., *Networks of Innovation*, Change and Meaning in the Age of the Internet. Oxford University Press, 2002.

Tur77 Turchin, V., *The Phenomenon of Science*. Columbia University Press, New York, 1977.

Tur95 Turchin, V., A Dialogue on Metasystem Transition. In Heylighen, F., Joslyn C., Turchin V. (Eds.), *The Quantum of Evolution. Toward a Theory of Metasystem Transitions*, Gordon and Breach Science Publishers, New York, 1995.

Uni05 United States Department of Defense, *Defense Acquisition Guidebook*, http://akss.dau.mil/dag/welcome.asp [5.4.2005]

APPENDIX 1

ZOPE – WEB APPLICATION SERVER

Zope (Z object publishing environment)[1] is an open source web application server primarily written in the Python programming language[2] for building content management systems, intranets, portals, and custom applications. It was originally authored by Zope Corporation[3] but parts of it have been open-source as far back as 1996 and the whole application server since 1998. Zope Corporation still drives the primary vision and development of Zope, with each release being maintained and developed by more and more community members, either by submitting patches to the bug collector or by developers with access to check in new features and fixes.

Zope is interesting because of its open source development model, extensionability and through-the-web development possibilities. There are numerous products (plug-in Zope components)[4] available for download to extend the basic set of application building tools: new content objects, relational database and other external data source connectors, advanced content management tools, and even full applications for content and document management or bug and issue tracking. Zope itself includes its own HTTP, FTP, WebDAV, and XML-RPC serving capabilities, but it can also be used with the Apache or other web servers. It also features a transactional object database (which can store not only content and custom data, but also dynamic HTML templates and scripts), a search engine, and relational database (RDBMS) connections and code. In addition Zope's through-the-web development model allows users and developers to update web sites from anywhere in the world.

In order to have a more comprehensive understanding of Zope and its features as well as the Zope community, which consists of hundreds of companies and thousands of developers all over the world, it is worthwhile to have a look at the Zope Development Site (http://dev.zope.org/), which contains ongoing projects and proposals for Zope's past and future.

---

[1] http://www.zope.com/Products/Zope.html [19.1.2005], http://zope.org [17.05.2005], http://zope.org/WhatIsZope [17.05.2005]
[2] http://www.python.org [17.5.2005]
[3] http://www.zope.com [19.01.2005]
[4] http://www.zope.org/Products/ [19.01.2005]