Software Innovation

Eight work-style heuristics for creative system developers

Jeremy Rose

www.dbooks.org

Beta version

First published 2010

By Software Innovation

Aalborg University

Department of Computer Science

Selma Lagerløfs Vej 300

Aalborg 9220

Denmark

Creative Commons License - Attribution-NonCommercial-NoDerivs 2.5

You are free to copy, distribute, display, and perform the work under the following conditions:

- ATTRIBUTION. You must attribute the work in the manner specified by the author or licensor.
- NON-COMMERCIAL. You may not use this work for commercial purposes.
- NO DERIVATIVE WORKS. You may not alter, transform, or build upon this work.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of these conditions can be waived if you get permission from the copyright holder. Your fair use and other rights are in no way affected by the above.

Copyright (c) 2010 Jeremy Rose

Whole or partial use of the book should be attributed (referenced) according to normal academic practice.

www.dbooks.org

Preface

This short book was originally written as course notes for a part of a new Software Innovation course at the Department of Computer Science, Aalborg University in Denmark. The students have primarily engineering backgrounds and a primarily technical education. The book was intended to provide counterpoint – that is to provide a scientific, but non-technical account of an emerging systems development topic, written in a non-normative style, where the intention is to further the students' understanding of their own and other system developers' practice in software innovation, rather than to provide them with a recipe for it. Thus it is intended as a complementary text for many system development courses, which will use as their main text either a method book (in our department currently Mathiassen et al's 'Object-oriented Analysis and Design'), or a software engineering book (such as Pressman or Pfleeger).

The topic – software innovation – is important in several respects. Software innovation re-focuses software development education away from the instrumental, the rational, the automating and the productive towards innovation – the focus that is least discussed, and, arguably most important. The book will be interesting in several contexts. Engineering schools may use it as something of an antidote to their normal traditions, whereas business and humanities schools will find an alternative perspective to their method-oriented system development education. It is designed to be understandable in all these contexts, with one foot in the software engineering tradition and one in the information systems tradition.

The book is intended for teaching rather than as a research contribution, but as a result of its writing process it does also function as an overview or summary of what is currently known about software innovation from many disparate sources. This is because it is researched and written as an extended literature survey of a particular sample of research: those contributions where the writers focus both on innovation and on software or system development. Primary sources for chapters are given at the end of the chapters and a complete list of sources is given at the end of the book. There are, however many gaps in this literature which I have been forced to fill.

Acknowledgement

To my good colleague Ivan Aaen, who contributed many of the better ideas, and to the many talented students we work with.

www.dbooks.org

Contents

Introduction **11**

WHY STUDY SOFTWARE INNOVATION? 11

The global perspective *11*

The competition perspective *12*

The developer perspective *12*

KNOWLEDGE SOURCES FOR SOFTWARE INNOVATION 12

SOFTWARE INNOVATION - THE SHAPE OF THE STUDY 14

EIGHT WORK-STYLE HEURISTICS 15

INNOVATION CONCEPTS AND SOFTWARE DEVELOPMENT 15

Three basic starting places: creativity, invention, innovation *15*

Radical and incremental innovation *16*

Product and process innovation *17*

Installed base (infrastructure) *17*

Innovation and software systems *17*

Sources and further reading: *18*

1. Keep your head up: software trajectories and innovation windows **19**

TECHNOLOGY AND ECONOMIC DEVELOPMENT 19

INSTALLED BASE, INFRASTRUCTURE 20

SOFTWARE TECHNOLOGY TRAJECTORIES 22

SOFTWARE TECHNOLOGY CONVERGENCE 24

THE SOFTWARE INNOVATION WINDOW 25

WORK-STYLE HEURISTIC 1 - KEEP YOUR HEAD UP 26

Sources and further reading: *27*

2. Grow your community: network, knowledge, learning **28**

VIRTUAL INNOVATION COMMUNITY: THE OPEN SOURCE MOVEMENT 30

OPEN INNOVATION 32

WORK-STYLE HEURISTIC 2 - GROW YOUR KNOWLEDGE COMMUNITY 34

Sources and further reading: *35*

3. Target the product's innovation profile: innovative software **37**

Characteristics of innovative software products 37
Utility - hierarchies of technical systems 39
Novelty: levels of innovation 40
Incremental and radical innovation 41
Utility forms 42
Innovation utility form 1: computing infrastructural *42*
Innovation utility form 2: technology enabling *43*
Innovation utility form 3: user service *44*
Innovation utility form 4: business change enabling *45*
Innovation utility form 5: interaction and communication *45*
Innovation utility form 6: entertainment *46*
Work-style heuristic 3 - target your product's innovation profile 47
Sources and further reading: *48*

4. Shape your own process: software process and innovation 49

Software development method – innovation is not a typical goal 49
Linear innovation in industry 51
The software innovation life cycle model 52
Iterative software innovation process models 53
Do agile methods promote innovation? 54
Market-led and technology-led software innovation 54
Improvisation, bricolage 55
Six innovation process strategies 57
Innovation process strategy 1: creative requirements analysis *57*
Innovation process strategy 2: designed process framework *58*
Innovation process strategy 3: low tech prototyping *59*
Innovation process strategy 4: user-driven software innovation *61*
Innovation process strategy 5: community development and the open source model *62*
Innovation process strategy 6: research prototype *63*
Software process innovation 64
The global picture *64*
The local picture *66*

www.dbooks.org

WORK-STYLE HEURISTIC 4 - SHAPE YOUR OWN PROCESS 66

Sources and further reading: *67*

5. Develop your personal creativity: the creative software developer 69

CREATIVITY AS THE DEVELOPER'S MENTAL PROCESS 70

CREATIVITY AS A SET OF PERSONAL COMPETENCES 71

CREATIVITY AS A STYLE OF THINKING 73

CREATIVITY AS META-THINKING: RECOGNISING UNCONSCIOUS PRE-DISPOSITIONS 74

CREATIVITY AS WHOLE-BRAIN THINKING: BEYOND RATIONALITY 75

CREATIVITY AS A STATE OF MIND 76

CREATIVITY AS A RELATIONSHIP BETWEEN THE DEVELOPER AND THE OUTSIDE WORLD 77

CREATIVITY AS A UNIVERSAL MENTAL SKILL TO BE ENHANCED 78

WORK-STYLE HEURISTIC 5 - DEVELOP YOUR PERSONAL CREATIVITY 78

Sources and further reading *79*

6. Be a super-team-worker: the innovative software team 80

CREATIVE/INNOVATIVE WORK ENVIRONMENTS: BARRIERS 81

GROUP DYSFUNCTION 83

INNOVATIVE TEAM ROLES: 84

INNOVATION TEAM INTERACTION 86

TEAM LEARNING AND INNOVATION 88

ACCOMMODATION OF DIVERGENT THINKING 90

EXPERTISE INTEGRATION 90

OVERVIEW: MACRO + MICRO INTEGRATION 91

INNOVATIVE TEAMWORK PATTERNS 92

ENVIRONMENTAL SCANNING 93

WORK-STYLE HEURISTIC 6 - BE A SUPER-TEAM-WORKER 94

Sources and further reading *95*

7. Bring your toolbox: creativity tools and techniques 96

CREATIVITY TOOLS 96

CHARACTERISTICS OF APPLICATIONS SUPPORTING CREATIVITY 96

A SOFTWARE SUPPORT TOOLBOX 99
CREATIVITY TECHNIQUES 100
A STARTING REPERTOIRE OF CREATIVITY TECHNIQUES FOR SOFTWARE DEVELOPMENT 105
Brainstorming *105*
Backward mapping *105*
SCAMPER *106*
Six Serving Men *106*
Six thinking hats *107*
Vision box *108*
Elevator test *108*
WORK-STYLE HEURISTIC 7 – BRING YOUR TOOLBOX 108
Sources and further reading *109*

8. Know when you are (not) innovative: assessment and evaluation 110

PERSONAL CREATIVITY: PSYCHOMETRIC TESTING 110
INNOVATIVE SOFTWARE PRODUCT ASSESSMENT 111
WORK ENVIRONMENT ASSESSMENT 111
ASSESSMENT OVERVIEW 112
HERE-AND-NOW QUICK-AND-DIRTY EVALUATION INSTRUMENT 113
Keep your head up *114*
Grow your knowledge community *114*
Target your product's innovation profile *114*
Shape your own process *114*
Develop your personal creativity *115*
Be a super-team-worker *115*
Bring your toolbox *115*
Know when you are (not) innovative *116*
WORK-STYLE HEURISTIC 8 – UNDERSTAND WHEN YOU ARE (NOT) INNOVATIVE 116
Sources and further reading *116*

9. Software innovation: eight work-style heuristics for innovative system developers 117

SOFTWARE INNOVATION 117

www.dbooks.org

EIGHT WORK-STYLE HEURISTICS FOR INNOVATIVE SYSTEM DEVELOPERS 120
Keep your head up *121*
Grow your knowledge community *121*
Target your product's innovation profile *121*
Shape your own process *121*
Develop your personal creativity *122*
Be a super-team-worker *122*
Bring your toolbox *122*
Know when you are (not) innovative *122*
COMPREHENSIVE LIST OF READING AND SOURCES 124

Introduction

The theme of the book is software innovation - creativity and innovation in the development, design and exploitation of information systems (software). Software innovation is an important topic, since it now underpins most of the significant technological advances in modern societies, but surprisingly little researched. Nor does it figure much in the education of system designers and developers. This education is mainly focused on instrumental and normative techniques - efficient programming underpinned by method and engineering techniques. System developers learn an engineering (or business or design) craft – but sometimes forget the point of their endeavours. This is to provide software and systems which change the practices of their user communities – a core definition of innovation.

Why study software innovation?

Motivations for understanding and studying software innovation are split into three perspectives:

- The global perspective
- The competition perspective
- The developer perspective

The global perspective

The making of software and development of information systems, like all forms of human work, evolves to match the society it has to serve. It would be more accurate to say that software development evolves in a circular and dependent relationship with society, since information systems are also an important part of the society we live in. The world we live in is increasingly *globalised*, which means that the connections between societies and businesses in different parts of the world are becoming stronger. This means that software and systems are increasingly built to serve wide groups of users and organisations in many countries. Programming languages are based on English, but are essentially international which means that, in principle, software can be made anywhere in the world. A further development, consequent upon globalisation, is *standardization.* Microsoft Word is a standardised software package used all over the world; it is primarily the language of the interface which is different. There are many such software packages, including most forms of operating software, personal productivity and business systems (such as enterprise resource planning systems) which are standardized in this way. Of course there are many cultural factors which make it impractical to build software in remote locations, but increasingly software development can be, and often is outsourced. This means it can be produced by well-educated engineers in countries which have much lower wage structures that in the developed countries. Economic factors tend to dictate that software will be outsourced if a similar result can be achieved

www.dbooks.org

more cheaply elsewhere. The last trend which is noticeable in the evolution of software development is i*ndustrialisation*. Whereas software in the 60's was used by a small number of highly educated consumers, and developed by a programming elite with skills that were possessed by only a handful, it is now ubiquitous. Software is found everywhere, in offices, homes, shops and cars. Most of us can't work without a computer, can't communicate without a mobile phone, and can't run a home without a whole range of gadgets run by embedded software. The large scale of software development means that it is becoming mass-produced - made for many by many, rather than by an elite. This trend means that much software development is of a fairly routine nature and can be made relatively quickly by engineers with solid technical educations. This routine kind of development can often be outsourced.

With these three factors (globalisation, standardisation, and industrialisation) dominating the current evolution of software development, software firms in highly developed countries, employing expensive, but highly-educated engineers and consultants) need to think carefully about their market position. Now, and increasingly in the future, they will not be able to compete in the market for everyday routine software, and must focus on development forms with higher value addition. One of these is software innovation.

The competition perspective

The shifting macro-trends mean that software firms in highly developed countries must understand how to be innovative to retain their competitive positions in the market. They need to be able to attract the best engineers, to be flexible in the face of rapid technology development, to understand modern development methods, to incorporate changing software technologies and to position themselves at the leading edges of the markets they serve. Software innovation is one of the key elements in competitive success in mature software markets. Innovative software firms ride the crest of the technology wave.

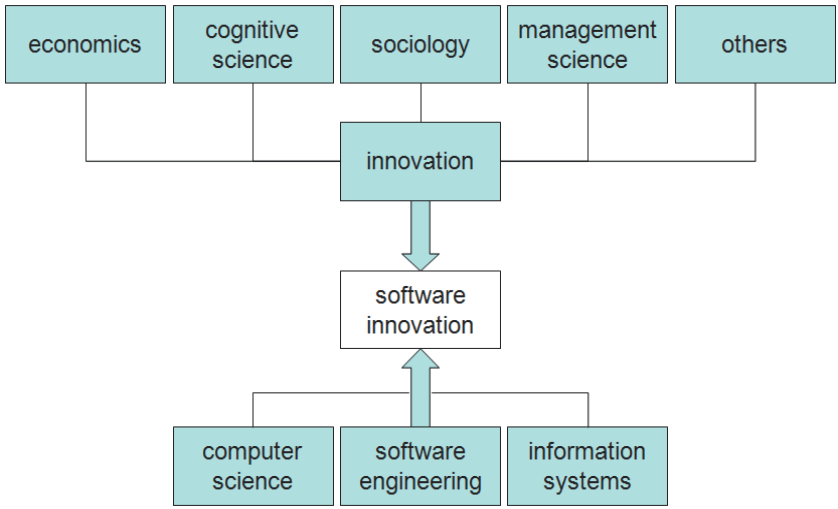
The developer perspective

Highly competent, well-educated and experienced developers and consultants need challenge to flourish. They do not thrive on repetitive and routine work. They need to constantly develop their skills, learn new techniques and technologies, to have both creative freedom of expression, and space and time to express that creativity. They need some control over their own development processes, and to be given enough responsibility to be experiment (and occasionally to fail) without drastic consequences. In other words, they need to be software innovators

Knowledge sources for software innovation

There is, unfortunately, no real science of software innovation, in the form of a definitive textbook or a series of well-defined research programs. However

the study of innovation in general is quite well developed, with contributions from several disciplines, and the various disciplines that focus on software and information system development also have some focus on innovation. The knowledge represented in this book thus comes from many interrelated sources.



- Economics provides us with understandings of how innovation (and particularly technology innovation) drives economic progress in society. The seminal figure here is Joseph Schumpeter.
- Cognitive science and psychology contribute to our understanding of creativity in the individual. For instance, Csíkszentmihályi theorizes the characteristics of the human mind when it is in a creative state.
- The sociology of science gives us a broader understanding of how technology and society develop hand in hand.
- Management science has developed understandings of how to create and manage innovative firms and teams, which are extremely relevant for software firms and development teams.

These and many other contributions belong to the field of studies called innovation. However system designers' core disciplines have also been concerned with innovation and creativity

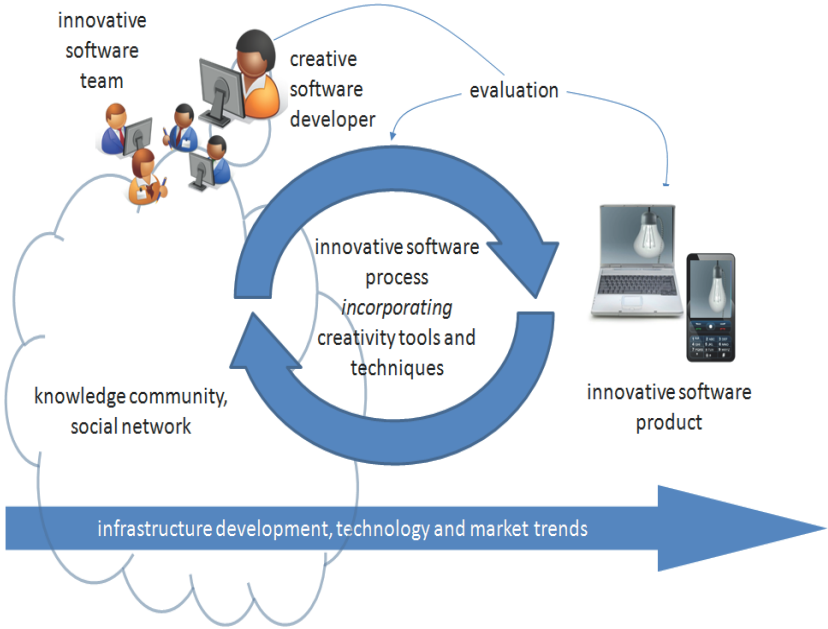
- Computer science contains a long tradition of innovation, and is the base discipline for software engineers; many of the great software pioneers work in this or related fields.
- Software engineering particularly helps with process innovation: here the evolution of agile development styles plays an important role.

www.dbooks.org

- Information systems is important for the understanding of the application of software innovation – thus the relationship between the eventual innovative software product and its implementation context (community, organisation, society), and the impact of software innovation as social change.

When researchers and writers in these traditions address innovation, they often use the concepts and theories developed in innovation studies. Thus the science of software innovation starts in many places, but the combination of these traditions can be used to give a relatively secure basis for exploring the subject.

Software innovation - the shape of the study



Software innovation in this book is understood as a development process which leads to a software artefact – a program, application, algorithm or to code. In principle, the process, the product, or both can be innovative. Software is built by developers working in teams, so it's appropriate to study both how individual developers are creative, and how teams function in an innovative way. We should also be able to understand where development is innovative, and where it is less so - in other words to evaluate it. There are many creativity techniques that might help the development process, and all development activities can be underpinned with software tools, so these should also be studied. Innovation studies also show that innovation is strengthened by community – the network of experts that the developers are

in touch with. Finally, software is not built in isolation, but in a social context, and the study of infrastructure and of technology and market trends can be important to the planning and timing of innovations.

Eight work-style heuristics

The book is not a technique or process guide, or a method or blue-print for software innovation. Instead it suggests that innovative system developers work in particular characteristic ways. These work-styles can be expressed as rather broad heuristics: that is, generalised precepts for attitudes to development work. A heuristic, in this context, is a broad guideline for behaviour or action which 'will provide an acceptable solution to a problem in many practical scenarios but for which there is no formal proof of its correctness' (Wikipedia). The eight heuristics are:

Keep your head up

Grow your knowledge community

Target your product's innovation profile

Shape your own process

Develop your personal creativity

Be a super-team-worker

Bring your toolbox

Know when you are (not) innovative

They are discussed further in the following chapters.

Innovation concepts and software development

Now we will turn to some basic concepts in innovation studies (which will help define a basic understanding of the topic) and their relevance for software development. In each case the terms are explained, not with a formal definition, but as they are consistently used in the book.

Three basic starting places: creativity, invention, innovation

Three terms will appear again and again in this study. They are related but also distinct.

- *Creativity* refers to the personal (or group) characteristics which can lead to invention, often described as internal abilities or states or relationships.

www.dbooks.org

- *Invention* refers to the process or result of creativity - to an idea or artifact which is novel, or the action of developing it.
- *Innovation* describes the creative act and invention carried into wider use, leading to substantial kinds of change; thus the successful exploitation of new ideas.

Thus we should understand that innovation is more than creativity and more than invention. Merely to design something that is new is not innovation; in fact novel ideas are fairly commonplace and often not the difficult part of innovation. The invention must be developed and produced (normally commercially), distributed and brought into use. The end result of innovation is social change, a change of understanding or practice in a community of people.

Thus innovation is sometimes described like a formula:

Innovation = Invention + Exploitation + Diffusion

where innovation is composed of the invention (new idea or artefact) itself, its commercial development and exploitation, and its adoption in a wider community of users. Thus the result of successful innovation is experienced as change in the way people work, the way business is carried out, people's choice of entertainment, their communication habits and interaction, the governance of communities, and in many other aspects of social life. Innovation is itself, as we shall later discover, social – usually the work of many people, rather than a single idea generator.

Radical and incremental innovation

We can distinguish between two types of innovation: *radical* and *incremental*. Radical innovation involves disruptive or discontinuous change, a break with what has gone before and an entirely new way of doing things. Radical innovation is rare (the wheel, the steam engine, the computer, the internet). It is sometimes associated with resistance as peoples ways of thinking and working are changed fundamentally over short periods of times. These changes can be painful, throwing groups of people out of work, or changing the political balance of power. Incremental innovation is much more common, consisting of relatively small improvements to existing practices or ideas. However not every incremental improvement can be described as innovation, in order to be considered innovation, the improvement must have a certain scale of impact. Incremental innovation makes it meaningful to speak of *innovation cycles* – iterating or sequential series of minor and major improvements driving technological advance (for example, from early flying machines to today's airliners).

Product and process innovation

Another useful distinction is between *product* and *process* innovation. An innovative product is an artefact (or a software system) which displays characteristics of *novelty* and *utility*. Novelty means the product has not been developed before, whereas utility refers to its economic value (what consumers are prepared to pay for it). Ford's model T, the first really successful automobile is an example. Process innovation, by contrast, is innovation in the ways that artefacts are made, their development method or engineering process. The model T could not have been successful without the mass production techniques (principally the automated production or assembly line) used to build it. They enabled it to be produced in a certain volume, and at a certain price that could make it widely attractive.

Installed base (infrastructure)

Innovation is time and situation dependent, which means that a change in one situation at one time is not necessarily innovative, whereas the same development in another time or place might be. Take as an example internet provision. In Scandinavian countries almost everyone has access to the internet at the time of writing, whereas in sub-Saharan Africa very few people do. The introduction of the internet constituted a major social change in the 90's, revolutionising both work and leisure practices. However, provision of the internet to the remaining Scandinavian population at this point in time cannot really be described as innovation, even though it might change these peoples' lives to some extent. Nor could the internet be described as an innovation in Scandinavia in the 70's. It existed as an invention, but not one that was sufficiently exploited or diffused to be described as an innovation. However, widespread adoption of the internet in sub-Saharan today might be experienced as innovation by Africans – they have little experience of it and it might produce extensive changes. To introduce another example: agile development methods are not really new, but a development firms introducing them into projects for the first time will definitely experience the change as a process innovation. Therefore the idea of installed base is borrowed from the study of infrastructure; we use it here to describe the starting point for innovation – that is, the current situation in terms of available software, or development process. It follows that we can think of software innovation at different levels – from the global societal level, to the local community level where innovation is experienced as change by relatively small groups of users or developers.

Innovation and software systems

All these basic innovation concepts can be applied without difficulty to software systems. We will be interested in the creativity of software developers and consultants, and the way they work in creative teams. We will be interested in innovative software systems (products) and how they can be commercially developed and consequently widely diffused in society. We

www.dbooks.org

will be interested in the effects that innovative software systems have on their users and in communities at large. It will interest us to study the radical innovations in computing and information systems, but will be largely concerned with what can more often be achieved - incremental innovation. We will be especially interested in process – the ways that system developers work innovatively, and the methods and techniques they use.

Sources and further reading:

COOPER, R. B. (2000) Information Technology Development Creativity: A Case Study of Attempted Radical Change. *MIS Quarterly,* 24**,** 245-276.

DENNING, P. J. (2004) The social life of innovation. Communications of the ACM, 47, 15-19.

FAGERBERG, J. (2005) Innovation: a guide to the literature. IN FAGERBERG, J., MOWERY, C. & NELSON, R. R. (Eds.) The Oxford Handbook of Innovation Oxford, Oxford University Press.

ROBERTS, E. B. (1988) Managing invention and innovation. Research Technology Management, 31, 11-27.

1. Keep your head up: software trajectories and innovation windows

In this chapter we will look at software innovation in its context in society. The theory is mainly derived from social and economic studies of technology innovation, and adapted, as always, to the software innovation context. However the focus will not be a generalized understanding of the role of software innovation in a society or an economy, but instead on the software innovator's application of these kinds of understandings. We have already established that innovation is time-dependent and our subject of interest will primarily be timing – when to innovate. The chapter will develop the idea of a software innovation window – a space of time in which the conditions for software innovation are optimal. The proposition behind the chapter will be that these conditions are at least partly analyzable – and it is thus possible for the smart software innovator to be in the right place at the right time. In order to make this kind of analysis we will have to understand several different phenomena, including

- *infrastructure* (installed base) – the understanding that all software innovation is dependent upon the condition of the infrastructures that will support them, which can be both technical and social
- software *technology trajectory* – the idea that software technologies develop in particular historical directions which can be understood, and to some extent predicted
- software *technology convergence* – a phenomena where software technologies tend to come together and be integrated in applications or devices
- software *innovation windows* – the time box where innovation is possible, and where it is still possible to make an impact in the market before it is dominated by other innovators.

Technology and economic development

Technology innovation is a good indicator for economic growth. Countries that are able to sustain high levels of technology innovation (measured, for instance, in patents) also enjoy economic prosperity. There is even some evidence that the information technology and internet revolutions are widening the gap between innovators and non-innovators -

'affluent states at the cutting edge of technological change have reinforced their lead in the new knowledge economy, but so far the benefits of the internet have not trickled down……productivity gains from information technology may widen the chasm between the most affluent nations and those that lack the skills, resources and infrastructures to invest in the information society.' NORRIS, P. (2001) Digital Divide, Cambridge, Cambridge University Press.

www.dbooks.org

Part of the reason for this relationship is that many societal structures need to be in place to facilitate innovation of the global variety. These can be described as infrastructure, or installed base. In sub-Saharan Africa, where many villages do not have reliable power supplies, there is little point in expecting your users to adopt an internet-based computer game; however, if you innovate in the field of solar panel power generation you have a developing market.

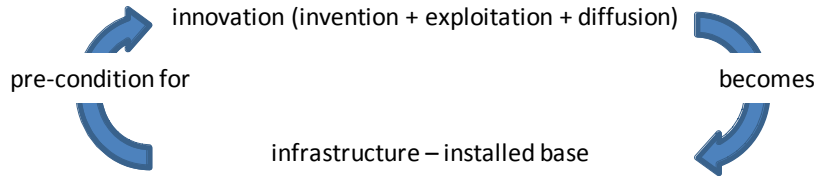
The same picture that operates at the societal level can be found at the level of companies. Those companies that are technologically innovative, for example in their manufacturing processes, or in adopting information technologies, will normally have a competitive edge over non-innovators (though it should be understood that, in both societal and industry arenas, innovation is only one of many factors contributing to economic success). Developed (rich) societies and leading edge companies are more dependent on innovation, better innovators, and earlier users of innovations. In this way they are engaged in a continuing cycle of innovation, where the economic benefits of innovation are invested in more innovation. As innovations are absorbed into general use they become part of the installed base upon which the next generation of innovations can be made. Social structures such as research and development departments and university research teams also build upon past achievements in a cycle of success.

The other side of the innovation cycle coin is the tendency of routine and well-established forms of technical work to move to locations where labour is cheaper. If a technology is relatively well-understood and the local infrastructures are good enough, then non-innovative technology work can often be performed more cheaply in less developed countries. Thus steel manufacture moved away from the countries at the centre of the original industrial revolution, and ship and car building has larger re-located from the advanced western nations to the emerging eastern nations. Software construction is currently undergoing the same transition, where it is sufficiently mature and well-understood (routine) for many parts of it to be outsourced or relocated. India leads the emerging nations drive to capture this market, and the areas around Calcutta and Bangalore have become India's Silicon Valley

Installed base, infrastructure

Central to the understanding of technology development in this rather wide societal perspective is the idea of infrastructure. You can think of infrastructure as yesterday's innovation. Once a railway, the petrol engine, an internet router, or a software compiler was an innovation, but now it is part of your community's daily experience. It's always available, it more or less always works, you don't really think about it unless it isn't working. When you have a power blackout then you will be irritated if you can't charge your phone, but you will never think to be grateful for the vast majority of the

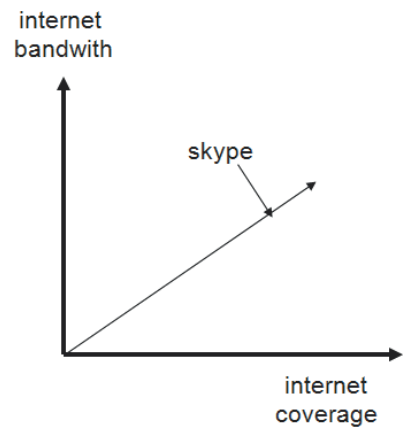
time that you can charge it without effort. Infrastructure is the unnoticed precondition for technology innovation.



We can distinguish two forms of infrastructure, the physical and the social. Our road system infrastructure is partly a physical structure of tarmac, metal and plastic, partly a series of social conventions about which side of the road we agree to drive on, and what we do when the traffic light is red. Infrastructure is not permanent, but under constant development and modification.

Installed base (infrastructure) is critical to software innovation. It's difficult to separate hardware and software development in any rigorous way, but every PC application is dependent on a complex set of hardware requirements to run. An up-to-date commercial application is unlikely to run well (if at all) on a computer that is five years old. The processor will be too slow, there will be too little memory available, and various protocols and operating systems will be missing. Web-based applications depend on bandwidth available to large numbers of users that would have been unthinkable ten years ago. The speed at which software and hardware innovations in the computing and information technology fields are adopted and diffused and become part of the installed base is remarkable in relation to previous types of innovation.

We'll illustrate the relationship between installed base and software innovation with a case study. Skype is an internet based communication programme based on VOIP and peer-to-peer technologies. It integrates a range of communication support services, and exploits its technologies to provide a cost benefit to its users compared to conventional landline and mobile phone services. It became a considerable success, but is entirely dependent on certain infrastructures being in place. Firstly it is dependent on having sufficient bandwidth available to internet uses to support a reasonable degree of sound quality in voice exchanges. In the early days of the internet no one had this, and it is only recently that broadband has been readily available at a price consumers were prepared to pay, and only in developed

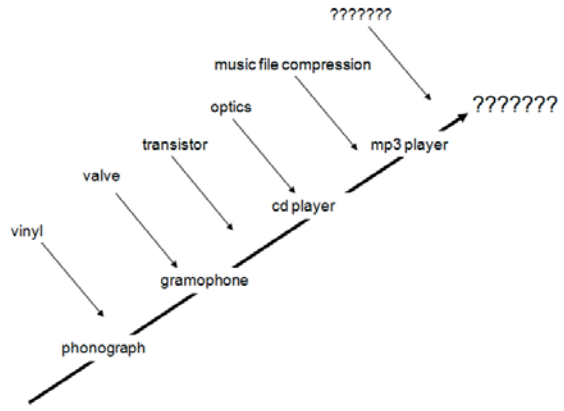


www.dbooks.org

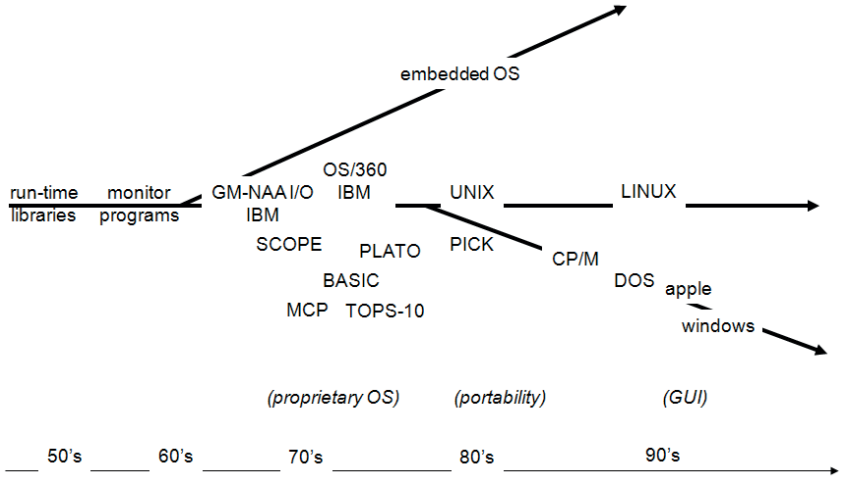
countries. Secondly it is dependent on coverage – the numbers of users are connected. Without many connected users (nodes) there is no-one on-line to talk to. Only when a critical mass is reached is it likely that the people you intend to communicate with will also be online. Many nodes are also necessary to support efficient peer-to-peer architectures. In addition a variety of social infrastructures need to be in place, not least the degree of computer literacy and widespread acceptance of the internet which makes it possible to switch from the conventional known landline phone technology. It's easy to see that Skype can never be a success if these infrastructures are not in place. Few people need a service which doesn't connect you to anyone you want to talk to, or doesn't allow you to hear what is being said.

Software technology trajectories

The second idea we will use to understand timing in software innovation is that of trajectory. A trajectory describes the direction in which something travels, and is used in the literature on 'social shaping' of technology to describe the historical development of technologies, and their relation to their social circumstances. Thus we can examine the development of the modern mp3 player from the early days of sound reproduction technologies – wax cylinders, needles and acoustic horns, through vinyl discs, valves (some music buffs still prefer them), transistors, optical technologies and file compression (the mp3 format). In such an analysis one could trace the way inventions in other fields (plastics, electronics, optics, data transmission) were adopted and integrated into sound reproduction in a chain of historical events. One could also study the evolution of listening habits as a social phenomenon related to the technological developments. Another phenomenon you should be aware of is digitalization: the tendency of mechanical information technologies to become first electronic then digital. In modern music production, the whole of the composition, distribution and listening process can be managed digitally. A song is composed and produced in a sequencer program, using sampled sounds and software instruments; the result is converted to an .mp3 file, distributed via the web, downloaded to, and played on an mp3 player. There need be no mechanical analogue interventions. This digitalization potential lies in any product which is information-oriented.



Software technology trajectories can be analyzed in the same way. In the next diagram, Wikipedia's description of the evolution of the operating system is analyzed as a tree of trajectories, charting the development of operating systems from the run time libraries of the 50's, through the various machine-dependent proprietary operating systems of the 70's to embedded, mainframe-oriented and PC systems of the present day.



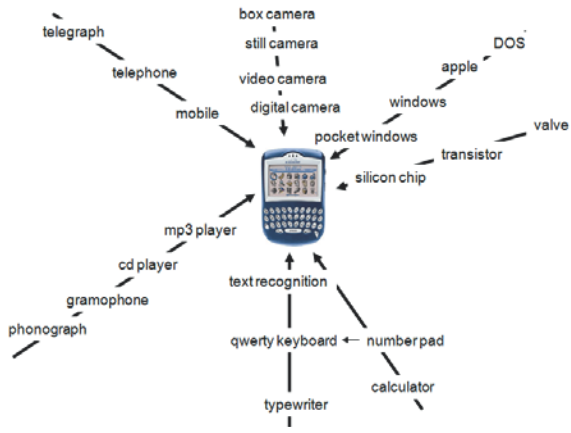
The really interesting question for software innovators has not, however, been asked yet: it is whether these historical descriptions of technology trajectories can be used to predict the future? As in all predictive science, the question cannot be answered with certainty, but it's reasonable to expect that software developers working in leading edge technologies have a deep knowledge of the evolution of the technologies they work with, are very well-oriented in respect to scientific advancement in their field, and work in close contact with other specialists and experts. This puts them in the position to understand what is coming next before the general public can, and to design their products accordingly. They may be fairly certain about those developments they are currently working on, or developments that are just around the corner, but much less certain about what will happen further into the future. Many of the fields move so fast that it's hard to predict more than about five years ahead. The further ahead one tries to look, the greater the degree of prediction uncertainty. Moreover, radical innovations, though rare, can alter the trajectory of a technology quite markedly and quite rapidly. IBM predicted in the 80's that the future of computing (its trajectory) lay in the mainframe, and sold the operating system that later became DOS to Bill Gates, and outsourced its microprocessor technology to Intel. The radical innovation of the microcomputer (or personal computer) took the direction

www.dbooks.org

of the evolution of computing in an entirely different direction, and cost IBM its leading position in the market.

If we return to the analysis of Skype, we can notice two technology trajectories that are particularly important. The first is the development of voice over internet (VoIP). The protocols that enable voice transmission over broadband connections (as an alternative to traditional copper wire telephony) became increasingly sophisticated in the 90's. Traditional telephony companies began to implement internet solutions, major software companies such as Cisco developed switching software, internet service providers saw an opportunity to broaden their service portfolios and customers became interested in a potentially cheap (or free) telephony alternative. The second is the rise of peer-to-peer (P2P) networks. Networks of many nodes arose as an alternative to client server architectures to enable file sharing. UseNet became popular for sharing news articles, whereas Napster became a very widely known music sharing service. BitTorrent is a modern equivalent. All these services combine some P2P elements with a particular perspective on the free sharing of information inherited from the early internet founders. The Estonian developers of Skype (Sky peer-to-peer) were able to understand the future potential of these technologies and combine them in a novel way.

Software technology convergence



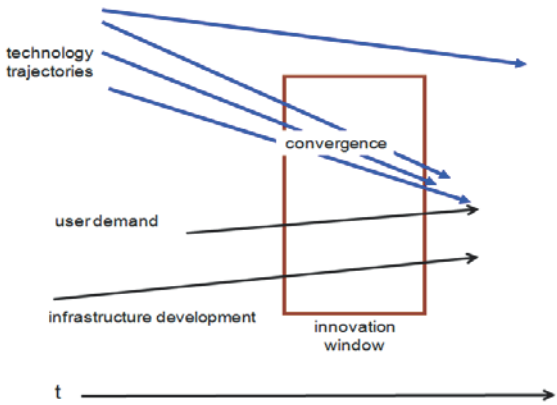
Further tendencies that can be observed with technology trajectories are *digitalization* and *convergence*. These can be illustrated by a pocket handheld communication device such as an iPhone or Blackberry. The device contains many different features and functionalities beyond its basic mobile communication role. It is potentially a contact database, a camera, a music player, a radio, a calculator, a route finder, an internet browser, an alarm clock, a file storage device, a game console. It has an operating system and layers of software handling communications, GPS, SMS, MMS and Bluetooth interfaces, and synchronization with the owner's PC. None of these software technologies are novel in themselves (though they may require considerable programming ingenuity to make) – they are adaptations of well understood concepts. Most of the features have independent lives in other devices. The

designer's job is to put together a package of functionality which has utility for the potential owner. A new technology is regularly added to the mix: camera, touch screen, motion sensors – however these technologies are also found in other devices. The technologies can thus be said to converge in the new device.

Over a longer historical perspective, a further tendency can be observed: the digitalization of mechanical technologies. Neither an address book, a camera, a music player, a radio, a calculator, a route finder, an alarm clock, nor a file storage device are originally digital technologies – in earlier instantiations they were paper address book, box camera, valve radio, map, clockwork and filing cabinet. Over time, the information content of the service is identified, digitalized and the software technologies for manipulating it developed. Eventually the mechanical technology dies out and is replaced by the more convenient digital technology.

We can observe technology convergence and digitalisation in the services that Skype offers: not just a phone substitute but an address book, a contact search facility, an instant messaging service, telephony conferencing, video conferencing, profiling, gaming, synchronisation with outlook, fax, mobile skype, different kinds of interfaces with conventional telephony.

The software innovation window



When we consider the implications of the various analyses in this chapter, it becomes obvious that the timing of a software innovation is important. Too early to market, and the necessary infrastructures will be too poorly developed to support the innovation – at least at a price that the consumers are prepared to pay. Too late to market and the developer risks that the innovation opportunity will be visible to many competitors, and that there will be many competing products available. There is a window of innovation opportunity. During the window, the necessary technical and social infrastructures will become available for the targeted user group. Technologies which will drive the innovation may become mature in their trajectories. A mechanical technology may be ripe for digitalization. A particular mix of technologies may converge in the innovative software product. Finally the potential user community will either be expressing a

www.dbooks.org

demand for new digital services, or there will be a particular stage of social development which will make it possible to create such a demand.

In this final Skype analysis we can look at the social and technical conditions (it´s innovation window) that enabled Skype to be widely adopted and a commercial success. We have already discussed the conditions of bandwidth and user adoption in the infrastructural technology (the internet). We have noted the technology trajectories of VoIP and P2P, which became sufficiently mature to be combined in an innovative way. We have also observed the convergence of a particular suite of functionality which could offer utility to the user. In the social environment we can see the intensification of communication, as people become more used to affordable and mobile telephony – in particular teenagers begin to develop previously unknown telephony habits. Many people are online at work, and sit in front of a computer for long stretches of time and the internet is known as a source of free information – there is little tradition of paying for internet services. At the same time the emergence of online gaming means that many people are also online in their leisure time. The use of SMS services grows explosively, in connection with the rapid expansion of mobile telephony, which also means that the national monopolies of the traditional telephony providers break down, and a more varied telephony pattern is established. Mobile telephony providers are in a price war, increasing focus on the price of telephony, which becomes an important focus in consumers' choice. However, increasing tariffs for broadcasting frequency licenses prevent prices dropping so low as to make them insignificant.

These are some of the social and technical conditions which the developers of Skype were able to recognise and exploit.

Work-style heuristic 1 - keep your head up

In this chapter we have examined the wider context of software innovation - in particular trends in technology development in society. We used Skype as an innovation case study to illustrate the theoretical principles. The observations and analysis point in a two particular directions:

1. software innovation is dependent on a lot more than programming skill and development method – traditional engineering skills
2. many of the situational factors in user communities and societal technology trends can be understood and analyzed.

The situational factors we have considered here are:

- technology trends and trajectories
- convergence and digitalization
- social and technical infrastructure development
- user (market) demand

- timing and innovation windows

It's not the intention to portray software innovation as an exercise in rational socio-technical analysis; nor do the ideas presented here provide the necessary tools to do this. However it should be clear that software innovation is dependent on extremely good situation awareness which makes it necessary for developers to have their heads out of their computer screens from time to time. Another term for this is environmental scanning. This situational awareness makes software innovation teams reliant on a variety of complementary competences which are orthogonal to programming competences. In this book, the shorthand for this style of trend-alert software development is 'heads-up' software innovation.

Sources and further reading:

HACKLIN, F., RAURICH, V. & MARXT, C. (2004) How incremental innovation becomes disruptive: the case of technology convergence. *Engineering Management Conference.* IEEE International

HELO, P. (2003) Technology trajectories in mobile telecommunications. *International Journal of Mobile Communications,* 1**,** 233-246.

KOSKI, H. A. (1999) The Installed Base Effect: Some Empirical Evidence From The Microcomputer Market. *Economics of Innovation and New Technology,* 8**,** 273-310.

WALKER, G. H., STANTON, N. A. & YOUNG, M. S. (2001) Where is computing driving cars? A technology trajectory of vehicle design. *International Journal of Human Computer Interaction*, 13, 203-229.

www.dbooks.org

2. Grow your community: network, knowledge, learning

In innovation theory, innovation is understood as a social process. Though we often picture a stereotypical lone genius scientist in a white coat with flowing hair (Einstein always comes to mind), scientific innovation is usually the work of teams or communities of people working on the same problems. Their professional knowledge and skills are developed, reproduced and enhanced through the various interactions (social practice) of the community. An innovation community can be described as the conjunction of people, ideas and expertise involving both co-operation and competition. Such a community can be physical (face-to-face meeting) or virtual and is commonly a mixture of both. A scientific conference is an example – establishing the principal problems in the field, reviewing and developing the participants' work and establishing new co-operations. Innovation communities are known in innovation theory as networks, but we avoid the term her because of its many other connotations in development work. An *innovation system* (a term used by innovation theorists) can include government policy makers, venture capitalists, non-governmental organisations, scientists, activists and companies - each with a distinct role to play in promoting innovation.

Many theorists have worked with the ideas of knowledge and community. An innovation community can be thought of as an *invisible college* - a term used by Diana Crane to describe the mechanisms for idea cross-fertilization and diffusion of knowledge in scientific communities. Ludvig Fleck used the term *thought community* or *thought collective* to describe communities of scientists producing and reproducing conceptualizations, practices, and technologies. Each thought community has its own thought style, which defines what can be meaningful knowledge for the community in question. This conceptualisation shares certain characteristics with Csiksentmihalyi's use of the *field* concept - the community who understand the domain ideas and practices in which an innovation is received, and who evaluate its value. Etienne Wenger and Jean Lave developed the idea of *community of practice* to describe group of professionals working on shared endeavours, and exhibiting situated learning through practice and participation. Professional knowledge is acquired by becoming a legitimate peripheral participator in a community of practice, through social interaction. Learning is thus dependent upon acceptance in the community, and expertise, identity, and community membership are inseparable. *Network of practice* is a concept used to describe both the tight-coupled community, and the many fellow professionals and colleagues with whom they have much looser (less frequent and involved) connections.

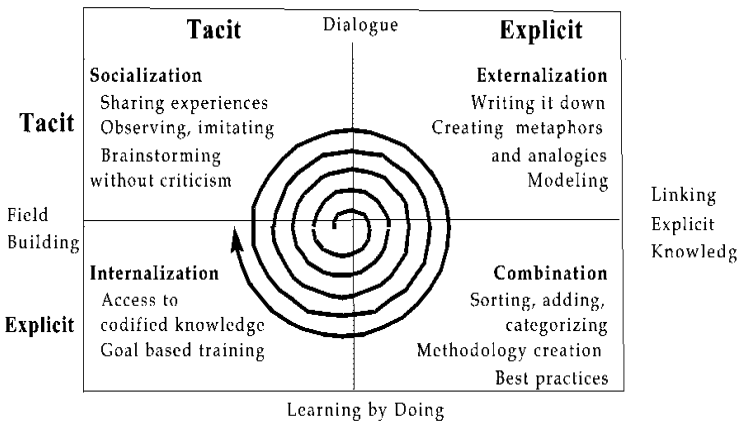
The community and the knowledge it works with are intimately connected:

"During the last decade there has been increasing interest in understanding the social basis of technology and knowledge. It has been argued that knowledge exists only in a social context, and that this social context is created by social practices.

According to this view, knowledge is created and reproduced in communities, and knowledge makes sense only in relation to such communities. Furthermore, this view rejects the idea that knowledge can be de-contextualized, or is something that can in any trivial way be grounded on an "external reality." Instead, this view sees knowledge as a product of a social process. Knowledge organizes social practices by institutionalizing ways of interpreting the world. Knowledge is embedded in social practices, conceptual systems, and material artifacts that are used in social practices. Technology, social practice, and knowledge complement each other and their evolution is part of the same process." (Tuomi, 2001, p 3-4)

Nonaka (1991) also understands knowledge as a social process, in which meaning in constructed, tacit knowledge made explicit and explicit knowledge internalized in a knowledge creation space (ba). Nonaka distinguishes between explicit (codifiable) and tacit (innate) knowledge which interact with each other in the creative activities of human beings. This *knowledge conversion* process is characterised as four activities: socialization, externalization, combination and internalisation.

Socialization transfers tacit knowledge between individuals through



observation, imitation and practice. Externalization is triggered by dialogue or collective reflection and relies on analogy or metaphor to translate tacit knowledge into documents and procedures. Combination reconfigures bodies of explicit knowledge through sorting, adding, combining and categorising processes and spreads it throughout an organisation or professional community. Internalisation translates explicit knowledge into individual tacit knowledge.

If knowledge is a social process, then the idea of a community of knowledge builders also becomes important. A striking physical example of an innovation community is encapsulated in the concept of a science park. Here many scientists, engineers and researchers are concentrated in a relatively

www.dbooks.org

small space. Science parks (such as Zhongguancun in China) attract massive investment and are often placed near to universities. Leading technology companies, as well as entrepreneurial start-ups, find it convenient to be there. There are many expensive facilities (laboratories, equipment, powerful computers), and different expertises which can provide solutions to product development problems. They have economies which are based on research funding, and (often) tax privileges and venture capital. They have incubators for new companies and projects, and a large well-educated employment pool, so that people can move between jobs. Nearby, there are the cultural facilities and housing opportunities which attract the creative classes; talented people move to the area, stimulating both economic and knowledge growth. In the software industry, many companies cluster in California's Silicon Valley to exploit these benefits, and most technology-oriented countries have corresponding areas.

Virtual innovation community: the open source movement

Silicon Valley is a physical example of a software innovation community, but our field has its own striking example of virtual community in the open source model. According to the Open Source Initiative (OSI), "open source promotes software reliability and quality by supporting independent peer review and rapid evolution of source code. To be OSI certified, the software must be distributed under a license that guarantees the right to read, redistribute, modify, and use the software freely." The well-known open source projects (Free Software Foundation, Linux, Freenet, Apache, Fetchmail) display many features common associated with communities. They have their heroes and legends (Richard Stallman, Linus Torvalds, Ian Clarke, Rob McCool, Eric Raymond). They have a strong sense of reputation, and an established co-operation principle. Knowledge is freely distributed in the form of open code. Reputation in the community is important and there are informal joining rituals and a relatively high entry threshold. Interested programmers start with bug-reporting and fixing, before they are encouraged to move on to more important tasks. They have a distinct legal framework for their work: the General Public License, also known as copyleft. They have their own virtual community home - Sourceforge.net, with 50,000+ projects and 500,000+ registered users. The site itself is not simply a code repository, but supports many social networking features, such as profiling, discussion and reputation management.

open-source community principles	
virtual net-enabled community	communities of programmers set the norms for practice and provide a wider sense of contributing and belonging - the community is often enabled by the internet and the products of the community are free and open to all its members.
the software challenge	the focus of the community is the software it builds: the software solution, characterised as a challenge to be collectively overcome.
self-organisation in networks	work is self-organised by independent and equal peers in networks across traditional organisational boundaries, rather than managed in the traditional sense - networks merge, change and dissolve in response to evolving technical challenges.
technical mastery	programmers aspire to technical excellence – mastery of their craft - where the ability to create innovative or elegant programming solutions is the primary measure of success.
self-realisation in the technological meritocracy	the community sets the scene for personal expression, creativity, heroism and championing innovation - membership of, and status in the technological elite is the primary reward, not commercial success.
code sharing, peer feedback, improvisation	improvement of the software solution takes place through code sharing and code revision by other programmers - the process is iterative and improvisatory.
technology leadership	programmer communities aspire to and attain technology leadership through technical mastery applied to the production of software solutions - the techno-elite do not follow markets or technology trends – they lead the markets and set the trends through innovation.
code quality	the engineering quality of the resultant code is the measure of success.
programming competence development	programming competence development is the motivating factor for improvement.

www.dbooks.org

According to von Hippel and von Krogh (2003), the open source movement is an example of an entirely new form of innovation - the private collective model. They identify two existing economic innovation models: the private investment model (typical in industry) and the collective action model (typically the university system).

private investment	collective action
• innovation supported by private investors (typically companies) who expect private returns	• innovation supported by the state
• investors retain knowledge through intellectual property law, copyright, patent	• innovators relinquish control of new knowledge to a common pool for the common good
• knowledge/innovation loss to society	• knowledge disseminated through society

The open source movement represents a new model because it is private, but collective. Developer-users invest their own resources, primarily in the form of programming time – however the end result (the code) is freely accessible. Thus they invest their private resources for the free revealing of knowledge (code), which is benefit to anyone who chooses to use it.

Open innovation

Although we have primarily discussed the classical open source project in the section above, there are many open innovation models, and many of them are hybrid models in which private companies and other closed source innovators are involved. Chesbrough argues that, with open innovation, organisations recognise that:

- not all the smart people work for us; we need to work with smart people inside and outside our company
- external R&D can create significant value; internal R&D is needed to claim some portion of that value
- we don't have to originate the research to profit from it
- building a better business model is better than getting to market first
- if we make the best use of internal and external ideas, we will win, and
- we should profit from others' use of our intellectual properties, and we should buy others' intellectual properties whenever it advances our own business model.

Open innovation projects are not necessarily strategically planned. The Danish company Lego were at first distressed to find that a small community of hackers had built up around their Mindstorms product and tried to stamp it out. Then they observed two things: firstly that this created a great deal of

bad feeling and resentment amongst their most dedicated users, and secondly that some of the code improvements that the hackers contributed were better than the solutions provided by their own developers. The solution: integrate the hackers into the product's beta-community and release parts of the code. The community has since grown into a significant part of the product's development, with its own wiki-like web-site. In this way they were able to activate the four principles of net-based mass collaboration articulated by Tapscott and Williams:

- peering - voluntary collaboration between free agents based on a de-centralised, non-hierarchical model
- sharing – knowledge sharing as the basis of collaboration
- openness – free access to ideas and code
- acting globally – (net-based) access for a wide user base to promote the flow of idea and knowledge exchange.

IBM has a large open innovation commitment with contributions to more than 120 open source projects, including Eclipse, Apache Derby, Apache Geronimo, Apache Tuscany and Apache Harmony, and more than $1 billion in Linux® development. The company runs a partly open access portal as the web support for its community, with downloads, learning resources and community resources including forums and blogs.

"Among the key advantages of open source is that the difficulties and costs of designing, developing, and improving software can be distributed among many contributors. IBM may be spending $100 million a year on development of Linux, but firms such as Nokia, Intel, and Hitachi are making substantial investments as well. Commercial investments in Linux are estimated to exceed $1 billion a year. Sizeable though its contribution is, IBM is sharing with others the effort and expense of developing this core infrastructure..... IBM can take advantage of ongoing open innovation done by others on Linux and other GPL projects because the GPL requires disclosure of source code of derivative programs of GPL software. By studying others' innovations, IBM engineers may perceive opportunities for building new technologies on the open source base. Some believe the open innovation model facilitates a faster pace of innovation" (Samuelson 2006 p.24)

Gassmann and Enkel introduce three core types of open innovation processes:

1. The outside-in process - external knowledge, technology and intellectual property rights are acquired from the outside and brought into the company.
2. The inside-out process - unused technology or IPR are introduced to the market and exploited outside the company.
3. The coupled process - the outside-in and inside-out processes are coupled and the company works in alliance with other companies.

www.dbooks.org

How the company cooperates with others vary, and the locus of innovation is often outside the company.

Open innovation has also been seen to advantage in the emergence of standards, for instance with the Open Mobile Alliance, which has been influential in the development of standards for 3G mobile services such as device management, messaging services, location and presence services, broadcast services and digital rights management.

A further dimension of the open innovation movement is the emergence of intermediaries (brokers) whose primary function is to match innovation problem owners with solution providers. InnoCentive, for example, allows potential innovators to search catalogues of unsolved problems ('challenges' organised by discipline and problem area ('pavilion') to locate innovations where their skills and knowledge, patents and property rights can be invaluable. The major ERP vendor SAP has their own pavilion where they issue many challenges representing their current innovation problems in the hope of attracting solutions that they themselves cannot necessarily envisage or implement.

Work-style heuristic 2 - grow your knowledge community

This chapter is intended to dispel the myth of the lone software innovator, working from her garage. Software innovation is understood as a community venture, because innovation is coupled with learning and knowledge generation and these are also community-based. Communities further the collective advancement of knowledge through collaboration and co-operation, but also through competition. Innovation communities are sometimes internal to large companies such as Google and Apple, who need to protect their knowledge advances and product ideas for commercial reasons, but often span developer companies, user companies, lead users, beta user communities, research institutions and universities. The open source movement is a specialised case in the software development field, where it is particularly easy to see the community elements at work; however there are many other models of open innovation. It seems that most software innovations happen in a social context; innovation has, in Denning's words, a 'social life'.

If we should extract some lessons for young software innovators and companies, then they should understand how to manage and improve their social networks – the community links which are important for their work and personal development. These may be people they work with on projects, experienced software managers and developers, teachers and mentors, researchers they come in contact with at conferences, and many types of users, groups and organisations who commission and work with the systems they build. These groups help to define what a promising software innovation is, to generate and test ideas, to prototype and user test, and to exploit, market and diffuse both the knowledge and the products that are the

end result of the creative process. They need to target those practice communities who are important in their work, and earn admission to them. Young developers need to understand the state of the art, to team with colleagues with complementary skills and to test their ideas with other experts in the field. In the same way, a software innovation project is not always best served by commercial secrecy. In many cases there are many advantages to openness and collaboration, partnering and expertise sharing. The software innovator is not a lone wolf – but an expert networker.

Sources and further reading:

CHESBROUGH, H. (2003) Open Innovation: The New Imperative for Creating and Profiting from Technology, Boston, MA, Harvard Business School Publishing.

CSIKSZENTMIHALYI, M. (1997) Creativity: flow and the psychology of discovery and invention, Harper Perennial.

DENNING, P. J. (2004) The social life of innovation. *Communications of the ACM,* 47**,** 15-19.

FAGERBERG, J. (2005) Innovation: a guide to the literature. IN FAGERBERG, J., MOWERY, C. & NELSON, R. R. (Eds.) The Oxford Handbook of Innovation Oxford, Oxford University Press.

GASSMANN, O. & ENKEL, E. (2004) Towards a theory of open innovation, three core process archetypes. *R&D Management Conference.* Sesimbra.

NONAKA, I. (1991) The Knowledge-Creating Company. Harvard Business Review, 69.

POWELL, W. & GRODAL, S. (2005) Networks of Innovators. IN FAGERBERG, J. (Ed.) The Oxford Handbook of Innovation. New York, Oxford.

SAMUELSON, P. (2006) IBM's Pragmatic Embrace of Open Source. *Communications of the ACM,* 49**,** 21-5.

TAPSCOTT, D. & A.D, W. (2006) *Wikinomics: How Mass Collaboration Changes Everything,* New York, Portfolio Hardcover.

TUOMI, I. (2001) Internet, Innovation, and Open Source: Actors in the Network. First Monday, 6.

TUOMI, I. (2003) Networks of Innovation. Oxford Press.

VON HIPPEL, E. & VON KROGH, G. (2003) Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. Organization Science, 14**,** 209-223.

VON KROGH, G., SPAETH, S. & LAKHANI, K. R. (2003) Community, joining, and specialization in open source software innovation: a case study. Research Policy, 32**,** 1217-1241.

www.dbooks.org

3. Target the product's innovation profile: innovative software

In this section we inspect the innovative software product. This, like many of the terms used in this study is hard to define precisely, but refers to the output of the systems developer – primarily code. This may take the form of an application, an operating system, a suite of communication protocols (such as the OSI model), an algorithm, embedded software, mobile software, or a variety of other expressions. In principle, all computing artefacts that are not hardware are software products, but in the case of embedded software, even this distinction is hard to maintain. In many cases, hardware and software are developed side by side (think of a mobile telephone, for example), so that they are interdependent. We have adapted the term 'software product' from the innovation literature, though it sounds a little foreign to most developers, in order to cover these many different instantiations. We look at the characteristics of innovative software, and develop the idea of a product's 'innovation profile'.

Characteristics of innovative software products

Innovative software products, according to innovation theory, display the qualities of *novelty* and *utility*.

Global novelty is a characteristic of a software product which is a significant advance on other products in its domain across the board – i.e. never previously developed. However, software product innovation is both time and user community dependent. An innovation is an innovation at a point in time – when taken widely into use it becomes part of the installed base upon which other software innovations are built. A software product is also novel in relation to the experience of a particular user community – if they have never seen or used it before, then it is novel to them (*locally* novel) even if it is not globally novel.

Software products can represent an *incremental* or a *radical* innovation. Thus the move from a command line operating system interface to a graphical windows-based one based on a desktop metaphor could be understood as a major (radical) innovation, whereas subsequent improvements, for example improvements to support the utilisation of various kinds of media might be understood as incremental innovations. Radical innovations are usually discontinuous, divergent and often contentious; the emergence of the Mac OS and Microsoft Windows in the personal computer market signalled the almost universal adoption of the new style of interaction amongst end users. However, systems administrators and programmers, with a much better grasp of the conceptual organisation of an operating system tended to stick to a command line interface to Unix and Linux, and the available graphical user interfaces to these operating systems (X Window, for example) have not required the same degree of sophistication and polish. However radical

www.dbooks.org

innovations of this type are fairly rare and most innovation in software products is incremental and proceeds by versioning. Each new version of an operating system provides an incremental improvement to the previous one and incorporates small advances. More major changes are signalled by new product releases (Windows 3.0, Windows 95, Windows XP, Vista, Windows 7). Over time these many small improvements constitute a significant degree of innovation in the product – accumulated or *evolutionary* innovation.

IBM's "Window Manager" Patent - January 30, 2001

The preferred embodiments of the present invention provide a method and apparatus for managing and controlling the size and location of windows in a GUI-based computer system. Specifically, a window control mechanism is provided to enhance the basic functional features of a window in any windowing environment. By interacting with the window control mechanism, a user can quickly and easily relocate and resize a window without unnecessary mouse movement. In one preferred embodiment of the present invention, the user invokes the window control mechanism by positioning the cursor over the title bar of a window and using both buttons of a two button mouse. In another preferred embodiment of the present invention, the user invokes the window control mechanism by positioning the cursor over a window decoration and using both buttons of a two button mouse. Yet another preferred embodiment of the present invention allows the user to specify a keyboard keystroke combination to invoke the window control mechanism.

In addition to novelty, software product innovations display utility: they have some form of application which users value and are prepared to pay for. The role of software in developed societies is extremely broad and some different forms of utility are considered later in this section. The utility of a software product is therefore inseparable from the market that the product enters – which defines what can be paid for a given product at a given point in time. Market considerations, such as the availability of rival or substitute products, the cost of the product in relation to its perceived utility, and the availability of loan capital to finance the purchase of the product dictate how widely and rapidly a new software product can be adopted (diffused). Some exceptions to this principle in relation to open source innovation and freeware are discussed elsewhere. Market conditions are in turn affected by wider societal trends, such as changes in working patterns, or increases in leisure time.

New software is often patented, particularly in the United States, where the patent laws allow access for a wide variety of software products. European

laws are somewhat more restrictive, but still allow for many product patents that incorporate software. Patenting is, for reasons already discussed, a better indicator for invention than of innovation. Patenting is a contentious issue, particularly for the free software movement, and it is unclear whether patenting promotes innovation (by safeguarding the investment of the original developer) or hinders it (by preventing the invention becoming part of the (accessible) installed base so that other inventions can incorporate or build on it). The result of the diffusion of an innovative software product should more properly be evaluated by looking at the change that software facilitates in its user community. Thus successful software innovation can promote quite widespread changes in the behaviour of its user community – think for example of the spread of social networking software such as Facebook. We use the shorthand of social change to refer to these changes:

innovation (invention + exploitation + diffusion) *leads to* social change.

Social change does not usually indicate a change in a whole society, but a change in social practice – that is, the way a group of people habitually behave or interact. Please note that this formulation does not specify that the change is always positive for all groups of people at all times. Software innovation is not a universal good, and can easily have effects which many would not find desirable (in weaponry, for example). Innovations can throw large groups of people out of work, or disturb the political balance of power. Social change is, moreover, relative to a particular user community. This means that a software application does not have to make a society-wide impact to be innovative. Examples (like Facebook) which do this are useful because many can relate to them, but robot laser surgery is a very significant innovation amongst a tiny group of users: eye surgeons. Digital sequencers (Garage Band) and music notation programmes (Sibelius) are a significant advance for song-writers and composers which also incorporate another social change – extending the range of people to whom the activity is available. You can more or less write a song in Garage Band without any form of musical expertise.

Utility - hierarchies of technical systems

We can examine the extent of social change that a software innovation can be instrumental in with the help of Altshuller's hierarchies of technical

broad ↑

system	subsystem
transportation	cars, roads, maps, drivers, service stations
cars	power train, brakes, heating, steering, electrical
brakes	brake pedal, hydraulic cylinders, fluid, brake pad assembly
brake pad assembly	pad, mounting plate, rivets
pad	particles a, b, chemical bond
chemical bond	molecules a, b

↓ focused

www.dbooks.org

systems. This hierarchy illustrates the way that technology innovations are interrelated and highly dependent upon each other. We could understand the transportation system as a very broad society-wide system composed of many sub-systems. In the table on the right, each technical system is broken into subsystems which are then broken into further sub-systems. Chemical bonds are part of the structure of a brake pad, which is part of a brake pad assembly, which makes up the braking system of a car which is part of our transportation system. Changes in the transportation (broad) system (teleporting, personal flying machine), can have very far reaching social impacts, where as changes in the focused (chemical bond) level have relatively little impact (at least in this hierarchy). However the most common way of innovating is incremental, through multiple innovations in sub-systems. Thus the design of a car remains relatively stable (four wheels, metal frame, doors, windows), but many of its subsystems are continually improved, leading to evolutionary incremental improvement in the overall design.

It's also possible to think about software in this many layered hierarchical way (the OSI model is also a form of hierarchical layering). Thus massive multi-player online games (something of a revolution in gaming) are facilitated by many small advances: in net transport and routing protocols and hardware infrastructure which make it possible to transmit enough data and host with enough users, in personalization techniques and in 3D and graphic programming, in algorithms for representing natural laws (for example gravity) and in techniques in database farming and distributed applications. World of Warcraft rests at the top of a pyramid of supporting hierarchies of technical innovations – where only the combined weight of technical progress makes the summit possible.

Novelty: levels of innovation

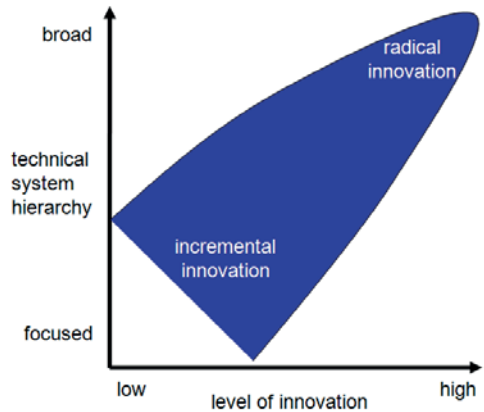
Whereas Altshuller's theorization of hierarchies of technical systems can help us understand impact, utility and social change, his specification of five levels of innovation (together with research indicating how common the different levels are) helps us to understand novelty in innovation. The five levels are:

- level 1 - routine design problems solved by methods well known within the specialty - usually no invention needed.
- level 2 - minor improvements to an existing system using methods known within the industry.
- level 3 - fundamental improvement to an existing system using methods known outside the industry.
- level 4 - a new generation of a system that entails a new principle for performing the system's primary functions - solutions are found more often in science than technology.
- level 5 - a rare scientific discovery or pioneering invention of an essentially new system.

Level 1 innovation is really closer to problem solving in everyday software design work, which most software developers encounter daily without really thinking of it as especially innovative. According to Altshuller's research, the majority (nearly seventy percent) of innovations can be categorised as level 1, and about 95% as level 1 or 2. Radical and pioneering work (level 5) is much rarer (less than one per cent) and often combined with advances in computer science. Here we could think of pioneering work with ARPANET, the precursor of the internet, between the American Defense Department and computer scientists at American universities including Stanford and UCLA.

Incremental and radical innovation

Putting theories of technical system hierarchies and levels of innovation together (the graph on the right) will also help us to understand radical and incremental innovation better. Whereas level 5 (pioneering) innovations which contribute to broad technical systems (the internet, for example) will be experienced as quite radical change by many user communities, most innovation will take place in more focused technical systems at lower innovation levels and will be experienced as incremental innovation. Some work is so focused, and the innovation level so low, that we would normally think of it as ordinary design work or problem solving, rather than label it innovation. However these judgements are always made with historical hindsight – this is because the impact of an innovation can take many years to develop.



innovation (invention + exploitation + diffusion) *leads to* social change

impact delay: delay between invention and social change caused by the time required for commercial exploitation and diffusion to the user community

This picture is further complicated by the many incremental accumulative innovations of the internet and its many sub-systems as exploitation and diffusion progress over time. Thus the impact of ARPANET in terms of social change is rather small, and it is only years later, after many additional

www.dbooks.org

developments, that it can be recognised as part of the radical innovation known as the internet, which also lies at the summit of the many evolving technical hierarchies (accumulating innovation) necessary for its continued development and diffusion.

Utility forms

In his section of the chapter we look at some of the purposes of innovative software. Software is used in many different contexts, for many different purposes, which means that it can lead to many forms of change. In order to correspond with terminology used earlier, we will look at utility forms – the different ways in which innovative software brings benefit and leads to change. It's impossible to provide a series of categories which cover all possible forms of utility, but we here specify six utility forms which give a wide coverage of the range of utility that can be achieved. The purpose of doing this is to unlock the mental silos in which many of us operate, where computer science students may be primarily interested in technical infrastructural issues, whereas business school students tend to understand computing as large business applications. Part of innovation is the ability to relate disparate influences, skills and concepts, and innovators tend to have both deep expertise, and heads-up overview. The six utility forms are:

- computing infrastructural
- technology enabling
- user service
- business change enabling
- interaction and communication
- entertainment

This analysis will also give us the opportunity to examine some concrete examples of innovative software products, and ask why they should be considered innovative. What do they contribute, how are they novel and useful, and what kinds of change do they lead to? These will be basic questions that developers ask themselves when working on new concepts and software solutions.

Innovation utility form 1: computing infrastructural

Software innovations can provide underlying improvements for the delivery of other computing services. In this way they help to move the infrastructure of computing forward, and to build platforms which enable new types of applications to run. Some examples could be:

- PC operating systems – providing the platform for software applications to run on
- network protocols – providing the transmission framework for the exchange of digital data

- mobile routing – enabling transparent switching between net cells and service providers
- grid computing – providing the computing environment for extreme processing heavy applications

Such innovations enable infrastructure change and alter the practice of software developers and system administrators, but it could be argued that social change is a secondary effect of infrastructural innovation. Thus that the internet infrastructure does not directly influence society at large, but that the applications that run on it (email, VOIP, messaging, distributed applications) do.

Innovation example: TCP/IP (1973-8)

TCP Header

Bit offset	Bits 0–3	4–7	8–15	16–31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Data offset	Reserved	CWR ECE URG ACK PSH RST SYN FIN	Window
128	Checksum			Urgent pointer
160	Options (optional)			
160/192+	Data			

It's hard to remember that computers used to be standalone devices – a kind of calculator occupying a whole room. The early implementations of packet switching which enabled the military ARPANET were developed into the Transmission Control Protocol and Internet Protocol by Vinton Cerf and his team based at Stanford University. They ensure that data packages arrive in the correct order, that they have minimal error, that duplicate packages are discarded, that lost packets are resent, and manage traffic congestion. The protocols were first adopted by the American military, then by American computer manufacturers in the 1980's. They proved to be extremely robust (there's an implementation for carrier pigeons that is proved to work!) and still form the basis of the extended layered suite of protocols that enable data transmission on the modern internet. They are not the cause of the rise of the internet (here there are other developmental and commercial factors at work) - but they are a precondition. The internet is associated with a very significant societal level of social change, described in the sociologist Manuel Castells' book 'The Rise of the Network Society.'

Innovation utility form 2: technology enabling

Embedded software can enable innovation in other technology products, such as cars and washing machines. Here the software is not necessarily the innovation, but the technology product which it enables is innovative. Many new technology products such as robot vacuum cleaners, automatic braking systems, washing machine control systems and programmable toys (such as Lego Mindstorm) are dependent on embedded software to provide the part of the functionality of the machine which is experienced as novel and useful.

www.dbooks.org

The software may require great skill and ingenuity to write, or may alternatively be a fairly routine programming job. The point is, that it is not directly the utility of the software that is in question, but the utility it enables in the new machine.

Innovation example: Copenhagen metro

It can be an unnerving experience to step on a train which resembles a conventional train and look forward through the front windscreen directly into a tunnel. For a second you don't realise why you are disturbed, but then you realise that you expect to be looking at the back of the driver's head – and the driver isn't there. The experience can provoke an instant of resistance – what will happen in the event of an accident or something unexpected? In this innovation the embedded software enables the driverless train. Software drives the very complex control systems that preserve functionality and safety in the train in the absence of the person who normally assumes responsibility for control. However the train is experienced as the innovation – and the complex software systems used to run it remain out of sight. However the innovation is dependent on many design and engineering decisions – in which the software plays its part (and the software may also be innovative in its own right). Though the software and engineering systems are expensive to build, the train is eventually cheaper to run, since there are no driver salaries to pay. The innovation can therefore be expected to be unpopular with at least one group of people – train drivers.

Innovation utility form 3: user service

Software innovations can provide new, improved, more efficient or cheaper services for communities of users. These types of innovations typically take an existing service and provide some combination of extended functionality, improved usability, cost saving and/or quality improvement - representing various aspects of utility for the user.

Innovation example: Skype

Our innovation example here is Skype - an extension to conventional telephony service. Skype combines internet and peer-to-peer technologies to provide extended convergent functionality (phone, chat, address book, video, conferencing, file exchange). In addition it has good interfaces with the more conventional land line and mobile telephony services. None of this functionality is unique to Skype or innovative in itself (though the decentralised implementation solution is). However the service provides a convenient package of functionality which is distinct from its main competitors. The decentralised internet platform supports one further utility to its customers – it's cheap. Cost savings to customers are off-set by a lower level of service and increased security risks, but many people are able to accept these.

Innovation utility form 4: business change enabling

Innovative software can be an enabler or driver for business change. Here it supports new ways of:

- doing business (for example eCommerce)
- internal administration (for example automation of insurance claims)
- reaching, holding and communicating with customers (as with Customer Relationship Management systems)
- developing and manufacturing products (as with Computer Aided Design and robotic production lines)

Innovation example: SAP (ERP system)

Enterprise Resource Planning Systems provide integrated support for most conventional business administration. In the last ten years they have been almost universally adopted by major globalised companies, despite very large costs and many implementation and adaptation problems. They are now rapidly spreading to smaller firms and to the public sector. In comparison with the previous generation of function-oriented stand-alone systems (payroll, human resource management, stock management), they offer many advantages:

- common data model and database
- customisable interfaces
- variable implementations
- best practice business models
- replaces many function-oriented stand-alone systems
- integrated management information and data mining
- web + eBusiness interfaces
- supply chain connectivity and management

Innovation utility form 5: interaction and communication

Innovative software applications can change the way people interact and communicate. It's especially the development of widespread access to the internet, and Web 2.0 concepts and technologies, together with good interfaces to mobile devices, which facilitate these types of innovations. We should distinguish between email and Skype (where traditional communication and interaction forms are simply better facilitated) and applications which encourage rather new types of interactions. Current innovations in communicative interaction centre around:

- greater reach and range - access to many social contacts from many geographical ant time zones at a loose coupled level, where the level of interaction is fairly superficial

www.dbooks.org

- time independence – storing the context for interaction and facilitating both synchronous and asynchronous communications
- supported interactions – offering different opportunities for interaction such as video (file) exchange and gaming, or remote interaction through avatars
- varying communication media - offering support for mixed media interaction (voice, text, chat)
- social network building
- on-line persona – control of the way the user is presented to other users
- platform connectivity – the interweaving of different mobile and net interaction platforms

Innovation example: Facebook.

Facebook is far from being the first social networking software to become popular, but has achieved (at the time of writing) levels of use which far outstrip its rivals, at least in certain parts of Europe and the US. It supports social networking activities such as:

- displaying user profile
- finding and making friends
- organising groups
- staying in contact
- dating support
- entertainment apps
- event monitoring and feeds
- email + messaging
- support for file exchange in various media
- notice board (wall)

It has an open API, and anyone is allowed to develop applications for it, within its editorial guidelines. In common with many web 2.0 applications it has massive utility for many users, but not the kind of utility that they will necessarily pay for – so basic use of the service is free and the revenue model is primarily based on advertising – which is attractive because of the many users and access to segmented socio-economic groups.

Innovation utility form 6: entertainment

A relatively large part of modern software innovation is design to underpin novel entertainment forms. Whereas business systems underpin our work activities, entertainment systems support our leisure activities. Some of the most significant developments in recent years are related to gaming (which

has recently become on-line), media clip (music and video) distribution, and the evolution of user-generated content. Such developments are dependent on infrastructure improvements (primarily band width and data storage) but also upon the reach and range of the modern internet – more users both to contribute and to participate. The generation of content has also become an important part of users' self-expression (you can visit my garage band compositions at http://www.reverbnation.com/theelectricmusicbox) and the border between what is a recreational and what is professional eroded (as in many bands' pages at MySpace). The exchange of this content is an important new form of interaction (see above). Many people find it more interesting to spend their leisure time with an interactive computer, than sitting passively in front of a television.

Innovation example: World of Warcraft

As our example for this type of innovation we can take the extremely popular massive multi-player on-line game WoW. Here the technical innovations are concerned with 3D programming of the virtual world, graphics, and handling multiple players over the net. Other aspects of the game (role playing, questing, guilds, levels of skill acquisition, rewards and the fantasy world background) are familiar from the stand-alone game world.

Work-style heuristic 3 - target your product's innovation profile

In this section we investigated the innovative software product. This is important for those who make their living from software innovation, since they need to be able to distinguish an innovative product, which has a possibility of finding a niche in the marketplace, from other types of software product. Many software projects are commissioned, for example by government ministries, and are judged by how well the expectations of the commissioners are met. However some companies, such as Google and Apple have high innovation expectations, and expect to lead the software market, not follow it. Many smaller companies operate in niche markets where they need to stay ahead of their competitors to survive. The innovative software product was understood to display both novelty and utility, and can eventually be measured by its ability to change its user community's patterns of behaviour. However this change is subject to impact delay, such that it is rarely evident in the early stages of product launch. We can understand utility



www.dbooks.org

better by referring to Altshuller's hierarchies of technical systems – here we are guide to understand why some innovations have wide impact, and some others relatively little. We can understand novelty better with the help of his levels of innovation theory. Here we understand that the scale of some innovations is more profound than that of others – from the solution of routine design problems to pioneering scientific breakthroughs. Finally we looked at six different utility forms for software innovations. The scope of software innovation is extremely broad, but focusing on utility concentrates developers, attention of the eventual use of their product. Products which are novel, but not widely used, are inventions; innovations have to be adopted by their user communities. Think of a highly successful product such as the Apple iPhone. Now imagine how many other innovations you have never heard of – the ones that came to the market, but never really made it into widespread use. Much of the difference can be expressed as utility.

If we use these ideas, and some from earlier chapters, we can understand that a software product has a particular *innovation profile,* which developers need to understand. Here are the major components:

- the software has a particular *user community*, and the characteristics of that community are understood
- the software is *novel* – it does something that other software cannot for its user community
- the software has a particular *utility* for the community, the form of which can be understood
- when the software is in use in the user community their behaviour will be different in certain ways (*social change*) and it is understood how
- the user community can be understood as a *market* in an economic sense, and the software has an economic value, price and cost which is understood
- the software is technically innovative, perhaps displaying *digitalization* or *convergence*, in the context of a particular *technology trajectory*
- the necessary *infrastructure* for the user community to use the product is in place, or will be when the product is released, and is understood.

Sources and further reading:

ALTSHULLER, G. S. (1988) *Creativity as an Exact Science,* New York, USA, Gordon & Breach.

Fagerberg, J., C. Mowery, et al., Eds. (2005). The Oxford Handbook of Innovation. Oxford, Oxford University Press.

4. Shape your own process: software process and innovation

Software processes describe the tasks and actions, the forms and norms and the formal and informal procedures that lie behind software development. These are expressed in the methods, tools and techniques that organize the work of a developer. A well-known, though oversimplified, distinction that will also be used in this chapter is between traditional development methods and agile methods. Traditional methods are structured linear analysis and design method development methods often expressed in a series of stages. They focus on rational paper-based analysis, modelling, linear stage models, documentation and accountability. There are many hundreds of such methods, most based to some extent on the simplest expression of a staged development method: the systems development life-cycle (SDLC or waterfall model). Well-known examples are Yourdon, Jackson, Information Engineering, SSADM, Merise, Euromethod, the more recent generation of object-oriented methodologies (Booch, Buhr, Coad and Yourdon, Colbert, Mathiassen et al, Rumbaugh, Shlaer-Mellor, Wirfs-Brock) and (to some extent) the rapid development methods. Agile methods (Adaptive Software Development, eXtreme Programming, SCRUM) represent a reaction to this tradition, and focus on practical development tasks, programming, prototyping and customer contact - usually in an iterative or incremental process which is better at handling change.

In this chapter we will investigate two related, but separate phenomena. In the first part the focus will be on the processes that lie behind the innovative software process. A relevant question to pose here will be 'how do you develop an innovative software product?' As the response to this we will look at six known innovation process strategies. In the second part we will instead concentrate on innovation in development processes. The relevant question is 'how do we improve (innovate in) development processes in software companies and teams?'

Software development method – innovation is not a typical goal

A brief examination of traditional and agile methods will reveal that producing an innovative product is not really their focus, purpose or aim.

Some typical goals of software development methods are expressed in the following table.

www.dbooks.org

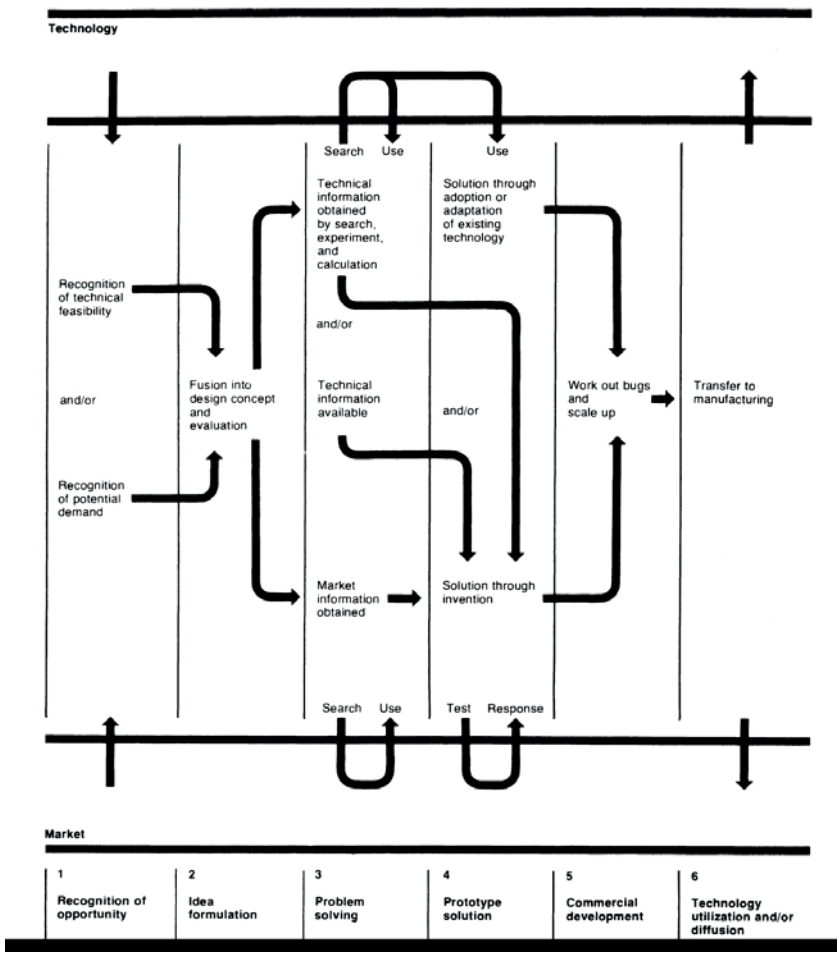
goal	method type	explanation
complexity management	traditional	organisation of large development efforts, with many developers, requirements, lines of code, complex architectures
uncertainty management	agile, prototyping	management of development where requirements, costs, technology, people, time scales are unknown or cannot be reliably predicted
project management	traditional, agile	planning; disposition and monitoring of tasks, people, time and resources
rational analysis and modelling	traditional, contextual design	understanding a work situation or user environment through models
communication through documentation	traditional	providing explanations for colleague developers, future developers and users
design through modelling	traditional	structuring design and programming work
automation of manual work processes	traditional, agile	providing computerised support for manual work processes in the work situation
working code	agile	focusing on programming work
speed	rapid development, agile	producing a working system in a reasonable time period
close relationships with customers and users	agile, participatory development	improving interactions between people

Innovation has seldom been the focus of the professionals and researchers that write normatively about building software. The most striking exception to this analysis is the business reengineering movement of the 1990's where IT was understood as a potent enabler for business change. Tom Davenport wrote about process innovation, process visions and organizational change, Michael Hammer and James Champy about radical redesign of companies and disruptive technologies and Henry Johansson about 'breaking the china' (a metaphor for radical change). However this literature was directed at

business managers and consultants, not at software developers, and business processes were the target for re-invention rather than application software. Software was understood here not as an invention in itself, but as the pre-existing facilitator for business change. It's therefore hard to find guidance or inspiration for innovative system builders in this literature.

Linear innovation in industry

Since the processes behind the development of innovative software are rather poorly researched, this discussion is anchored in rather better researched models of innovation processes in production industries. Here is a typical model as described by a leading researcher in the field.



Roberts, E.B., *Managing invention and innovation.* Research Technology Management, 1988. **31**(1): p. 11-27.

www.dbooks.org

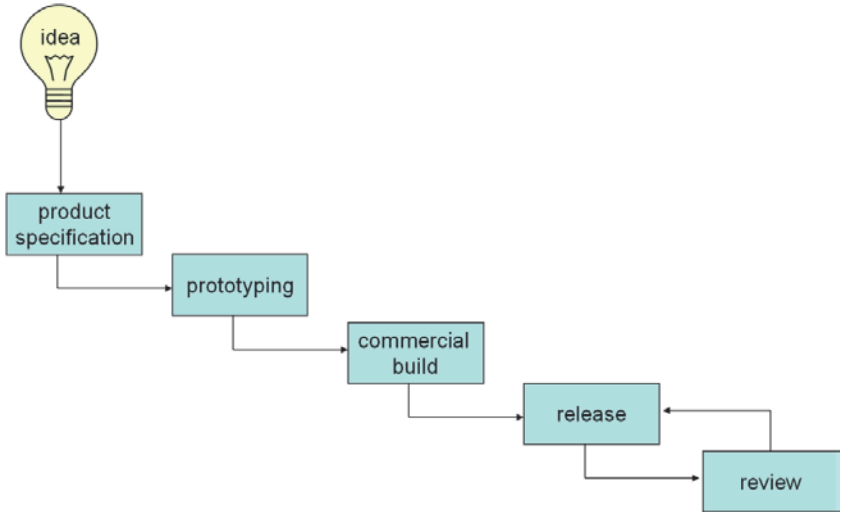
The model is characterized by two features

- its six linear stages or phases (opportunity recognition, idea formulation, problem solving, prototyping commercial development, technology diffusion)
- it's two motivations: technology push and market pull (discussed further below)

The model describes a linear process which is informed both by the technological demands of product development, and by the potential demand from the market. The stage/phase model could remind us of similar linear software development models.

The software innovation life cycle model

A linear account of an innovation process, rooted in a very traditional account of system development looks like this.

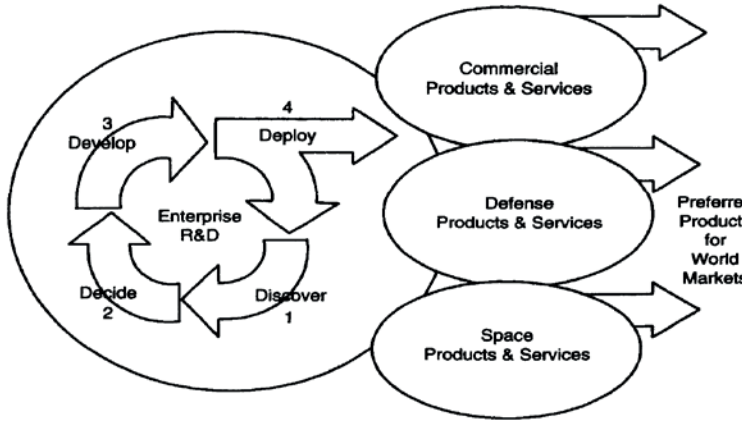


The model represents a rather simplified version of new software application development in large software firms. It is nicknamed 'the light bulb model' to indicate that it is dependent on a novel software product idea as the inspiration that sets the process in motion. However the idea that innovation is principally dependent on an inspirational idea (discovery point), though intuitive, is shown by the psychology of creativity to be oversimplified. The waterfall shape indicates a sequence of stages or phases each of which is dependent upon the successful completion of the first. The concept for the software product needs to be fully evolved at an early stage, before its realization in code, though there is some room for improvement though prototyping. The original product idea represents the primary innovation,

though there is also room for incremental innovations in subsequent releases of the software.

Iterative software innovation process models

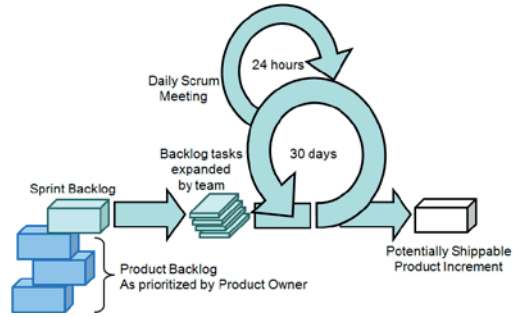
Roberts' model of industrial product innovation can be related to traditional software development models, but there are also iterative models. Boeing's innovation model (Lind, 2007) with its discover-decide-develop-deploy cycle, is appropriate for a large technology company with many innovation projects running concurrently, and a portfolio of research and development spanning minor improvements through blue sky research with no obvious applications in the near future.



Agile software development techniques are also primarily iterative in character (see the illustration of the SCRUM process), though rather more informal in character than Boeing's. Agile methods are not primarily focused on innovation, but Aaen's ESSENCE (discussed more thoroughly later in the chapter) offers a glimpse into an iterative, informal software innovation process. Such a process does not necessarily start with an 'idea' - a fully-formed software concept at the beginning of development. Innovation instead takes place through highly focused and creative bursts of development activity – 'storming.' Creativity is in focus throughout the life of the project, and is not confined to an idea generation phase at the beginning (thus Aaen proposes three development modes for developers to work with - idea generation, planning, growth - rather than a linear phase model). The creativity and energy of the process offers the conditions for innovative programming and development. Creativity techniques and games take the place of formal rational analysis.

www.dbooks.org

Do agile methods promote innovation?



Many developers may suspect that agile methods offer a software process that is more conducive to innovation. It has been pointed out that some aspects of agile methods resemble innovation techniques elsewhere in industry (like Boeing's process). It's clear that the introduction of agile methods to a traditionally-oriented software firm can be a process innovation. There are also some theoretical reasons to believe that they should be helpful for innovation. Flexibility helps deal with the uncertainties of working with leading edge software technologies, bureaucracy (avoided in agile methods) is a known creativity barrier, and interaction with customers develops domain knowledge. However agile methods were developed in response to the perceived need for more effective, programmer-friendly development methods - not in order to further innovation. There are few studies or evidence which supports the idea that agile methods lead to more innovative software products than traditional methods. Furious development in response to rather un-reflected use cases and feature backlogs may actually hinder innovation by removing the incentive and opportunity for idea generation. It's possible to argue that some elements of agility are necessary for an innovative development process, but an agile process will hardly be sufficient.

Market-led and technology-led software innovation

Roberts points out that technology innovation is influenced both by technology developments and potential markets. We could thus understand software innovation as either primarily market driven, or primarily technology driven



market-led	*technology-led*
user communities have sets of needs which develop over time	software technologies develop in particular directions at various speeds
some software firms have a very good understanding of their users' needs, and the markets they compete in	some software firms are at the leading edges of those developments
user and market needs can be analyzed and, to some extent, predicted	leading edge software technologies enable new products which will create their own demand in the market
the innovative software development process is targeted at responding to perceptions of future user needs (the market)	innovative software development process is targeted at providing novel technology products which have not previously been possible

Thus some software development firms will define themselves by their relationships with their customers, and their focus on customer needs, whereas the identity of other firms will be more closely connected to their ability to be at the breaking edge of technology development. However the two perspectives are not entirely mutually exclusive; many firms keep an eye on both technology and market. Nevertheless current understandings of software development processes are very heavily focused on responding to user needs, rather than understanding the needs of a market, or working with leading edge technologies.

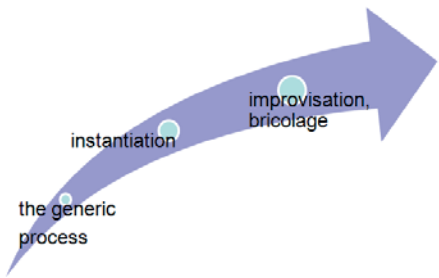
Improvisation, bricolage

An underlying process model, whether traditional or agile, only provides a skeleton underpinning the work of software innovation. Improvisation and bricolage flesh out the skeleton, whatever the underlying process. Developing technically exploratory software involves manoeuvring in uncharted waters, where development platforms are uncertain and untried, so it is unlikely that generic process models or formal development methods can provide enough support for the developer. Progress in response to complex and rapidly developing problem solving tasks will always depend on the creativity of the individual developer and the team. Improvisation implies the pre-existence of a set of resources (plans, tools, knowledge, and social structure) as the basis for variation. The wider the cumulative experience of the team, the greater the opportunity for variation, but not all variations are appropriate, or likely to lead to a successful problem solution. Improvisation

www.dbooks.org

represents a sequence of deliberate choices in a situation, not a series of accidents. Nevertheless it is extemporized during action – without prior plan or method. In a development situation, improvisation often takes the form 'let's try this……:' a customer meeting, a programming technique, a diagramming technique, a different hardware component. Improvisation is the natural response to complex problem situations which cannot be planned or fully anticipated in advance, but whereas planning is usually based on rational analysis (estimation, resource management, task distribution, budgeting), improvisation is inspirational in character. This does not mean that it is unreasoned, but rather that the time, resource or knowledge necessary for rational planning is not available.

Bricolage, the use of whatever resources and repertoire one has to perform whatever task one faces, is a related idea. Imagine a craftsman repairing a machine. His workshop is full of tools and spare parts – the results of many years of building and repairing similar machines. The damaged machine needs a new part but it will take three weeks to arrive, and a special tool to install. He takes a similar part from his shelf, puts it on the lathe and machines it to size, and installs it by improvising a tool from other tools in the workshop. The repair takes half an hour instead of three weeks. In the software situation, developers bring all kinds of techniques and programming knowledge from earlier projects and have previously developed applications stored on their hard disks. There are code components designed for re-use, and many open source applications and routines. There are known programming algorithms in text books. Sometimes problems are solved by throwing things from programmers' repertoires together rather than by solving all problems from the ground up by logic. Sometimes this is also a source of innovation and inspiration – creating new ideas through unplanned juxtapositions.



Most software projects can be understood as an instantiation (tailored adaption) of a generic process or method. No normative process or method can ever anticipate the situation of a real project and fully meet its needs. In addition, whatever the generic process and its tailoring, unforeseen events will always require a certain amount of improvisation and bricolage. In non-innovative (routine) development projects we have come to rely rather heavily on following an established generic process. In fact, some formal software process improvement techniques, such as the Capability Maturity Model are designed to enforce compliance with a company's generic process. In an innovation situation however, there are likely to be many complicating factors (such as new technologies and very uncertain requirements) and a

great deal of change. Projects often lose focus, go down blind alleys or get stuck. Here it will not usually be enough to adopt a pre-defined process and stick to it, even if such a process existed. The role of improvisation and bricolage will be crucial in overcoming these situations, and a critical survival instinct will be to recognise when the development process is no longer productive, and to adapt it in a direction which will move the project forward.

Six innovation process strategies

Though there are few methods and process frameworks which specifically address innovation in software development and, as yet, little discussion in the normative systems development literature, there are a number of innovation process strategies which can be understood and adopted. The six that are discussed here are:

- creative requirements analysis
- the designed process framework
- low tech prototyping
- user-driven software innovation
- community development
- the research prototype.

Innovation process strategy 1: creative requirements analysis

Where a software innovation lifecycle (light bulb) model is envisaged (with a consequent need to determine the shape of the software product early in the process) the logical focus will be upon the requirements phase – where the specification of the product is elicited from its future users. There is quite a lot of research into creative requirements analysis. It tends to focus on replacing conventional requirements engineering with more imaginative interaction and creativity techniques. Many of the techniques employed (e.g. Soft Systems Methodology, RESCUE) involve facilitated workshop activity with users. RESCUE (Requirements Engineering with Scenarios for User-Centred Engineering) uses conventional requirements analysis techniques (activity modelling, system goal modelling, use cases, context diagrams, storyboarding, requirements management) and combines them with techniques from creativity theory. Scenario based walkthroughs are used to hold different ideas in play. Workshops are organised to support the phases of preparation, incubation, illumination and verification (see chapter 5). The process is organized around creativity modes: exploratory, combinatorial, and transformational. It also employs the use of reasoning by analogy and metaphor. These techniques seek to establish a more creative relationship with users and customers, whereby more imaginative product ideas can be elicited and formulated into something resembling a conventional requirements specification. The role of the facilitator is not, as in

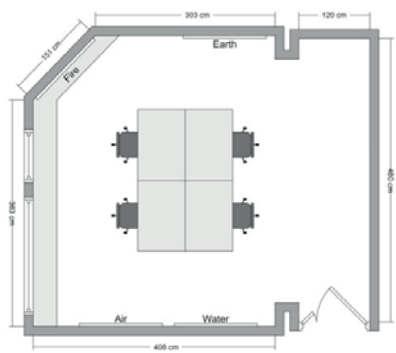
www.dbooks.org

conventional requirements engineering, to determine a set of requirements expressed by users, but to help the users to think more imaginatively about their work situations, about new and different ways of working, and about the software features that can support these. The facilitator will also teach the techniques which underpin creative user thinking.

Though particular types of users (lead users) drive innovation in other contexts (see below), experience shows that there are some limitations to the creativity of user groups. User groups are often bound to their habitual ways of doing things, threatened by change, and intimidated by expert developers. They are seldom good at understanding the relevance or work implications of emerging software technologies with which they are not familiar. It should also be noted that much software innovation by software firms is not driven by a relationship with a specific user group, but by more general market considerations. Here the innovative product thinking must come from the developers themselves.

Innovation process strategy 2: designed process framework

Researchers and practitioners interested in software innovation have the opportunity to design normative frameworks in the long tradition of the evolution of method in software development. Such a designed framework will normally combine some tools, techniques and practices with an underlying process model to give guidance to teams working with innovation and creativity in mind. Those frameworks are slow to appear, but the most mature to date is Ivan Aaen's Essence. Essence belongs to the agile tradition, borrows several ideas (for instance the explicit use of roles) from that tradition and can also be used in conjunction with well-known agile methods such as SCRUM and XP. It is a process framework, rather than a formal development method. ESSENCE can stand alone, but is designed to be used in conjunction with a particular infrastructure - an arrangement of space which utilises haptic thinking to establish particular work views on each of the four walls of a room. In the Software Innovation Research Laboratory (SIRL) at Aalborg University, four large interactive screens represent ESSENCE's four views: product, people, project, process. The product view is used for the various depictions of the system that will be built (metaphors, function lists, architectures, object models, code). The people view represents the user domain – the work systems, requirement suggestions, use patterns and examples and the various communications with users. The process view is used to explicitly depict the development process, to adapt it to changing circumstances and to make sure that appropriate tools,



techniques and practices are deployed in order to respond to changing circumstance and keep the project moving forward. Finally, the project view is used to manage the project – to ensure that schedules and sprints are established, that work is distributed sensibly and that deadlines are adhered to. Four distinct roles are employed: challenger, responder, anchor and child. The challenger represents the customer, user or product owner - the person with the domain knowledge and vision for the product. The responsibility here is to articulate the software challenge. Responders are developers whose job it is to prepare ambitious responses to the challenge. The anchor has the responsibility of facilitating the project: making sure that the process and roles are working and intervening and proposing adaptations if there are problems. Anyone can temporarily adopt the child role at any time – in this role one is empowered to ask the naïve question or make a divergent or counter-intuitive suggestion. In addition to views and roles, Aaen proposes three activity modes: idea generation, planning and growth. The modes are neither sequential, nor strictly iterative but are intended to be alternated in response to project circumstances. *Idea generation* usually relates to developing ideas for the project, but can also be used where there are process problems. *Planning* relates to the organization of the next part of the project, whereas *growth* is primarily dedicated to programming - building the product. In the work done at Aalborg University, the idea generation mode of ESSENCE is often integrated with creativity techniques such as the ones described in chapter 7.

Whereas users are primarily responsible for creativity and innovative ideas in creative requirements analysis, this responsibility is placed firmly in the hands of developers in ESSENCE. The rationale for such a designed process environment is to help developers to work creatively.

Innovation process strategy 3: low tech prototyping

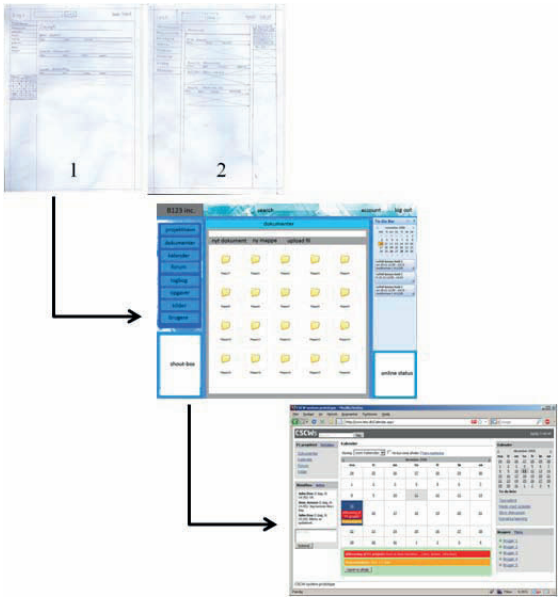
Prototyping, according to Tom Kelly, is the shorthand of innovation. Prototyping is used in system development in many contexts: horizontal prototyping represents a high level view of a complete system, vertical prototyping an in-depth representation of a particular set of functionality. Throwaway prototyping develops systems which are never intended to be the basis of commercial production software, whereas evolutionary and incremental prototypes represent early stages or parts of an eventual finished system. The conventional rationale for using prototyping is an improved dialogue with users and customers, and prototyping as a development approach nearly always calls for iterative review by users. In the innovation context this can certainly be useful, but the primary form of prototyping will be low tech (or low fidelity), and the principle rationale will be low cost experimentation and rapid learning. High tech prototyping has a tendency to lock developers into system ideas which have been heavily invested in, whereas low tech prototyping allows experimentation with little investment as part of the generation of many ideas or scenarios for a product. A low

www.dbooks.org

tech prototyping process will allow much room for experimentation with different inexpensive prototyping forms, before moving to more conventional code prototypes which require more investment of coding time, and are thus harder to discard.

low tech		
	lists and pictures	feature list, simple architecture or component diagram, rich picture
	paper prototypes	sketches, wireframe, storyboard, card, wizard-of oz
	low-fi mockups	foam and cardboard models, drawing tool, video, PowerPoint, html
	existing code re-combination	mash-up, patchwork
	code simulation	screen generators, application definition or simulation software, component reuse, open source components
	code prototypes	rapid application generators, visual basic
high tech		

A project using a low tech prototyping strategy might start with a list of possible features and a rich picture of some users with their system in a work situation, move to making paper prototypes – sketches of screens, a storyboard - and then make a very simple PowerPoint representation of the system's main screens. None of these representation forms consume much time, so many strategies and alternatives can be kept alive, discussions with users can continue, and prototypes can be improved, radically adapted, combined and discarded many times. When a direction is established, more effort-intensive prototypes can be developed: html mock-ups, mash-ups and patchwork prototypes. Here many things may be borrowed, copied and slung together to give higher fidelity, but still without the commitment of extensive coding. Even as ideas become reasonably firm, many tools exist which simulate or automate coding work to allow experimentation with systems which have partial functionality and some degree of realism and can be further explored with users in real situations. Only when the product idea is well-established will coding begin in earnest.



In the example on the left, a feature list was developed by interview and questionnaire with users before developing some wireframe paper prototypes which investigate different structures of the front page of the system. The next prototype was made in PowerPoint and automates progression between the various screens and some approximate content. The final prototype (before coding) looks fairly realistic but is an html mash-up – hardly any of the functionality works and many screen elements are cut and pasted - for example the calendar is nothing more than a screen dump from Microsoft Outlook. Each of the prototypes also served as a discussion piece with users.

Innovation process strategy 4: user-driven software innovation

Users are a very potent source of innovation in some situations and markets. Many users are driven to innovate because of their perceived work needs. An eye surgeon anticipates the need for a robot that can carry out surgery with a precision that cannot be achieved by the human hand. A stock market analyst needs a tool to gather and process large quantities of stock movement data and analyze it against a particular algorithm. A development manager responsible for enterprise resource planning systems with a very large number of diverse users needs to collect and analyze requirements for the next release. In each of these cases the user has a specific need, and a complex domain knowledge which is extremely hard for an analyst or programmer to acquire. The perceived need may be emerging and is therefore not (yet) recognized by software development firms. The need may be rather specific, making it too costly to go out and learn the requisite domain knowledge; it may be a niche market which is not yet attractive for developers. Domain knowledge that is difficult to transfer from users to manufacturers is known as *sticky* information. In these cases a particular kind of user can drive software innovation. Such a *lead user* is at the cutting edge of their profession (thus has demands that are not yet met) and normally has unusually well-developed software or computer competences. These competences are unlikely to be engineering or programming competences, but may involve a familiarity with software packages acquired in their practice,

www.dbooks.org

and the ability to conceptualize and describe a new software application, or to articulate their requirements in an unusually precise and logical way. Lead users also have much to gain from the innovations they participate in.

For developers, working with lead users poses a number of problems which are not easily solved through their engineering training. Much of conventional development practice is predicated upon the idea that user domain knowledge is (easily) transferred into the head of the developer and from there to a working program. Developers conduct analysis studies, or work through conversations with onsite customers. In user-driven innovation the attempt to understand the user domain may be abandoned, and the primary role of the developer may become facilitation – helping the lead user(s) to express their innovation ideas as code. Sticky solution-side knowledge belongs to the developers, but sticky domain, problem and need knowledge remains with the users, who must therefore be incorporated into the design process. Here a 'toolkit' will be valuable. This, according to von Hippel and Katz, will have the following five characteristics:

- it will enable users to carry out complete cycles of trial and error learning
- it will offer a solution space encompassing the software designs the user wishes to create
- the toolkit can be operated by the user using their own design language and skills, without advanced programming skills
- it will contain libraries of previously developed modules or components which the user can incorporate in their own designs
- it will ensure that the resulting design can easily be transferred to the production environment of the developers without requiring massive re-programming.

High end toolkits will be used to include users in the innovation process, whereas low-end toolkits facilitate tailoring and personalisation of software by users. Many Web 2.0 software providers provide toolkits exhibiting some of these features – for example the Google maps API which makes geographic-oriented application development available to programmers with general skills. Even simpler to use are Google gadgets and the many widgets available to users of social networking services. Computer game developers provide level builders, character-building kits, and scripting languages which allow tailored gameplay within the game environment (mods).

Innovation process strategy 5: community development and the open source model

Sometimes lead users form communities – sub-sets of professional bodies with a particular interest in developing new computerized work tools or gaming platforms. An example is the Linux community. Here the developers are also the lead users for the operating system they develop and refine. The next innovation process strategy was discussed in detail in chapter 2 and

is reiterated briefly here. It embraces the advantages of web-based mass collaboration:

- peering - voluntary collaboration between free agents based on a de-centralised, non-hierarchical model
- sharing – knowledge sharing as the basis of collaboration
- openness – free access to ideas and code
- acting globally – (net-based) access for a wide user base to promote the flow of idea and knowledge exchange.

Web-based mass collaboration is combined with a strongly incremental and iterative development strategy which should be understood as bottom-up continual improvement. The initial development effort may be carried by a small number of talented individuals, but once the community reaches its critical mass many individuals may contribute to both extending the content and functionality of the software, and to improving the quality of the code base. Open-source development is associated with a new innovation model - the private-collective model - but we can distinguish several types of open innovation. Two particular kinds of community development are highly innovative:

1. the lead user community - where the developers are also expert users with a strong need for new product features to manage their own work-lives. An example here is the Apache Software Foundation, with their suite of products for software developers and administrators
2. the platform/content model, where the developers are responsible for the software platform, but the user community is largely responsible for content. The technical platform for Second Life is provided by Linden Labs, but the game is nothing without the extensive in-world experience which is entirely provided by the user community, using the toolkits made available to them.

Innovation process strategy 6: research prototype

The last innovation process strategy is again collaboration-oriented, but this time with the objective of matching different kinds of research and development expertise. The model is extensively run by the European IST (Information Society Technologies) framework programs for research into ICT (Information and Communication Technologies). The frameworks provide for several different research instruments, but most demand:

- a mixture of private and public funding

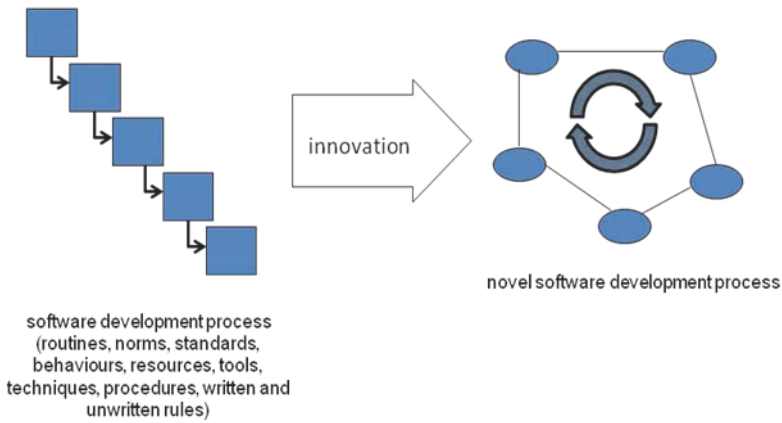
www.dbooks.org

- groupings of organizations, called consortiums, spanning both research organisations (such as university departments), software and hardware development firms, and (often) user organisations
- the development of working prototypes, called demonstrators.

The projects also provide access to a variety of other innovation services encouraged by the EU such as partnering, networking and innovation-friendly procurement. The rationale for this innovation strategy combines collaboration with researchers and developers at knowledge boundaries with a non-commercial funding mechanism which allows for experimentation without serious penalties for failure. However the strategy demands a relatively well-articulated product idea with an expected societal benefit and a track record in research and innovation as the price of entry to the European system. There is no proscribed development approach or method, but the review system strongly encourages formal project management. The National Science Foundation organizes similar programmes in America

Software process innovation

A further aspect of this process chapter concerns new software processes and how they come into existence and become employed.

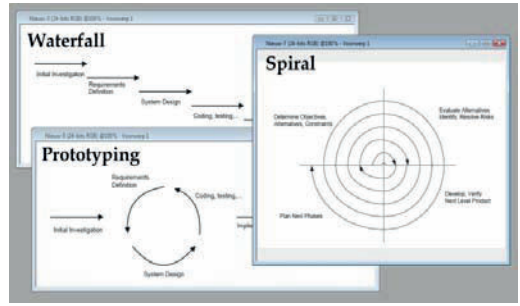


It will help us to distinguish between the global picture – how novel software processes are designed in the abstract, and the local level – concerning actual practices in software companies and development teams.

The global picture

Methods and software development techniques have been in constant evolution since the early days of software development. There is a strong normative tradition, where experienced practitioners and academics write books and articles describing idealized and generalized processes, which

practitioners are supposed to implement in their work. Traditional methods are often elaborations of variants of the waterfall model, or some variant of prototyping. Classical project management and software engineering techniques such as estimation, risk management, configuration management and various test strategies are also well established. There are many thousands of incremental innovations to these established techniques, and more appear every year. Relatively few of them become well established in practice, either because practitioners cannot see value in them or find them hard to learn and adopt, or simply because they never become widely known.



At the same time there are several more radical software process innovations currently in focus, most of which respond to a perceived flaw in the traditional norms.

innovation	examples	perceived flaw in traditional software process
participatory development	ETHICS, Scandinavian school	responds to lack of serious user involvement
context-aware methods	Contextual Design	responds to heavy focus on computer system design
rapid development	RAD	responds to poor speed of delivery
test driven development	TDD	responds to lack of rigor in delivering bug free code
agile methods	XP, SCRUM	responds to analysis-heavy and programmer unfriendly
open source	LINUX, REDHAT projects	responds to hierarchical and commercially oriented development style
business-focused	Business Process Re-engineering	responds to inability to focus on business process innovation
systems theory-focused	Soft-Systems Methodology, User Centred Design	responds to heavy focus on rational analysis and hard systems tradition

www.dbooks.org

This represents a complex picture of both incremental and radical innovation in software methods, techniques tools and processes.

The local picture

Though the global picture of software process innovation is one of constant evolution, the relationship between these rather abstract systems of normative ideas and what really happens in practice is a complicated one. Many software firms and development teams have years of history with traditional methods, and these are firmly embedded in their work practices. It's natural to want to improve the way you work, and software firms often have method departments and process improvement initiatives which are dedicated to achieving these ends. However the weight of tradition makes it easier to introduce incremental improvements which create less disruption and require lower learning curves. The benefits of more radical innovation at the local level, such as a move to contextual inquiry or agility, are often speculative or unknown, requiring a large leap of faith. Software process improvement (SPI), developed at Carnegie Mellon University is a relatively well-used-way of focusing on software process, and produces process innovation in many companies that take it seriously. However the level of innovation is local; SPI enforces compliance to rather traditional normative development models which may be new to the company concerned, but are well-understood at the global level.

Work-style heuristic 4 - Shape your own process

In this chapter we investigated which software processes are used to build innovative software, and how software processes innovate. Innovation models from industry can be related to the development models we are familiar with, but there is little research into software innovation methods, or innovation-focused normative tradition to lean upon. The chapter distinguished six innovation process strategies:

- creative requirements analysis
- the designed process framework
- low tech prototyping
- user-driven software innovation
- community development
- the research prototype

Whereas the first three strategies are conventional process strategies, which articulate in some respect what the developer should do and when they should do it, the last three are principally collaboration strategies. These reflect the strong presence of the network/community model in innovation theory. However, process, method and collaboration considerations are only one of a number of factors which enable software innovation. Moreover the special circumstances, risks and degree of change involved in being at the

leading edge of technology development make it unlikely that the semi-formal generic methods in the system development literatures will be successful. We are left with more general guidance frameworks – and the improvisation and bricolage skills of experienced expert developers.

The absence of a golden process route has a particular consequence: the innovative software developer must take control of the process. If different process and collaboration strategies show potential for helping innovation, then developers must mix and match them appropriately. If the development style in a particular software house hinders innovation in a particular project then developers must modify it. If there are many complex problems and no pre-determined methods for solving them, the developer must adjust the development process and solve them as they arise. When the project is stuck, and little progress is made, then the developer needs to introduce something into the process which allows it to turn the corner. An innovative software development does not need to be a process inventor - there are many appropriate strategies tools, techniques and practices – but they need to be a process shaper. The process by itself will never guarantee success.

Sources and further reading:

BANSLER, J. & HAVN, E. (2004) Improvisation in information systems development. IN KAPLAN, B. (Ed.) *Information Systems Research.* Boston, Springer.

DEARDEN, A. & HOWARD, S. (1998) Capturing user requirements and priorities for innovative interactive systems. *Proceedings of the Australasian Computer Human Interaction Conference*, 160–167.

DUGGAN, E. W. & THACHENKARY, C. S. (2004) Integrating nominal group technique and joint application development for improved systems requirements determination. *Information & Management,* 41, 399-411.

FLOYD, I. R., JONES, M. C., RATHI, D. & TWIDALE, M. B. (2007) Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software. *HICSS 2007: 40th Annual Hawaii International Conference on System Sciences.* Hawaii.

HOLMQUIST, L. E. (2004) User-driven innovation in the future applications lab. *CHI '04 extended abstracts on Human factors in computing systems.* Vienna, Austria, ACM.

KELLY, T. (2001) Prototyping is the Shorthand of Innovation. *Design Management Journal,* 12, 35-42.

LIND, J. (2007) Boeing's Global Enterprise Technology Process. *IEEE Engineering Management Review,* 35, 38-52.

MAIDEN, N., MANNING, S., ROBERTSON, S. & GREENWOOD, J. (2004) Integrating creativity workshops into structured requirements processes.

www.dbooks.org

Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques. Cambridge, MA, USA, ACM.

MAIDEN, N. & ROBERTSON, S. (2005) Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 105-116.

ROBERTS, E. B. (1988) Managing invention and innovation. *Research Technology Management,* 31, 11-27.

VON HIPPEL, E. & KATZ, R. (2002) Shifting Innovation to Users via Toolkits. *Management Science,* 48, 821-33.

AAEN, I. (2008) Essence: facilitating software innovation. *European Journal of Information Systems,* 17, 543-553.

5. Develop your personal creativity: the creative software developer

Creativity is often understood as a mental quality, ability, orientation, state of mind or set of skills. This means that the most advanced understandings of creativity are delivered by psychologists. Creativity implies novel and unconventional thinking, motivation and persistence, the ability to work with vague and poorly defined problems, and heuristic rather than algorithmic thinking. Creativity studies have two different foci which are not always well differentiated. Creativity is studied in the creative arts: music painting, literature, etc. Creativity is also studied in the context of scientific invention and innovation, and it is this form of investigation we are primarily interested in. Many researchers have also been interested in creativity in the systems development process. They usually consider creativity to be an important asset in a software developer (otherwise they would not be interested in the topic) but are less good at specifying exactly why this should be the case. Among the questions that these researchers investigate are:

- What does creativity bring to systems development?
- Can creativity be learned?
- How do you measure and evaluate creativity in software products and processes?
- Are systems developers more creative than other professionals?
- How do you create work conditions which stimulate creativity?
- Can creativity in software development be learned?

Rather than try to answer these questions one by one, this chapter will describe eight different approaches to the study of personal creativity in software development. The intention will not be to try and prescribe how a developer or programmer should be creative, but to understand and characterise the different ways creativity studies are used to shed light on software development work and personal creativity. If you have been though a conventional systems development or software engineering education, or studied the mainstream literature it is quite possibly the first time that you this topic has been raised for you. The literature largely makes the assumption that system development concerns the precise application of a set of engineering techniques and skills in order to automate a manual business process. Yet programming is an intensely creative activity – you start with nothing and slowly build a working application in which a myriad of design decisions are incorporated. Design literatures are rather different, emphasising the value of creativity and the fusion of function and aesthetics – so it may well be that the conventional system development assumptions are misplaced.

www.dbooks.org

Creativity as the developer's mental process

The psychologist G. Wallas in his book The Art of Thought (1926) identified five stages in the creative process, seen as a personal intellectual act.

1. *preparation* (preparatory work on a problem that focuses the individual's mind on the problem and explores the problem's dimensions)
2. *incubation* (where the problem is internalized into the unconscious mind and nothing appears externally to be happening)
3. *intimation* (the creative person gets a 'feeling' that a solution is on its way)
4. *illumination* or insight (where the creative idea bursts forth from its preconscious processing into conscious awareness); and
5. *verification* (where the idea is consciously verified, elaborated, and then applied).

Most system developers and programmers are familiar with this process – even if they do not articulate it in this way. Fixing an annoying programming bug may take some minutes (or hours) of staring at the code, trying to understand the way it is constructed, its purpose and intention and what is wrong with it. Hypotheses for identifying the exact nature of the bug are formulated and some experiments for fixing it are carried out. Eventually the problem comes into focus and the programming mistake is identified, rectified and tested. There may be a particular moment of insight, which could variously be described as an 'aha' moment, or a flash of illumination. The light bulb is used as a universal pictorial symbol for this moment, and we sometimes refer to it as a 'discovery point'. At other times the discovery point may be elusive – the bug may remained unfixed for hours or days, and the discovery point may happen in the midst of some un-related activity – showering or preparing food. Suddenly the problem solution is clear and it only remains to run to the computer and execute it. Most systems developers are extensively trained in rational problem solving and learn to articulate problems and their solutions through engineering formalisms, but Wallas reminds us that much of the creative act lies outside our conscious mind. The unconscious part of the mind continues to generate ideas and weigh solutions even when the rational thought process is focused elsewhere, and there is increasing evidence that sleep is an important time for unconscious mental processes. This is why stress is not conducive to creativity. Sometimes the most intelligent way to solve a difficult problem is to take a break and think about something else.

Creativity as a set of personal competences

In a software developer, creativity could be understood as a set of personal competences, in the same way that professional skills like object modelling, algorithm design and experience with a particular programming language are understood as competences. Thus a developer or project manager could understand their existing creativity competences and set out to improve them. Creativity competencies are concerned both with solving problems and with recognising opportunities. This is an important distinction.

- *Problem solving* requires an internal focus on a worrying aspect of a project: a program that doesn't run, a model that is incomplete or a customer who is unhappy. Engineering techniques and rational thinking are the best tools we have for problem solving. However everyone who has been in a difficult development project knows that you sometimes also need inspiration, gut-feeling and the courage to proceed in an incompletely understood direction. Herbert Simon explained this with the idea of 'bounded' rationality: the limits of our human capacity to analyse complexity and the consequent need to take decisions which lie on the borders of, or outside our rational analysis.
- *Recognising opportunity*, on the other hand, requires an external focus, an overview of what is happening around you, a sense of the boundaries and limits of a given technology or business process (and how these limits can be extended) an ability to relate apparently unconnected phenomena. A shorthand for this way of thinking is 'heads-up' developing – that is a style of developing where the participants make conscious efforts to be aware of many aspects of what is going on around them, rather than solely focusing on the code editor on their screen.

If you work in an organisation and are asked to build a mobile commerce platform, then you have a problem that needs to be solved. If you work in a company that develops mobile applications and you talk to a friend in a company that sells though a conventional ecommerce platform then you can recognise an opportunity.

Here are some competences which creative software professionals can be expected to exhibit:

- They should be able to cope with poorly-defined problems. There are many of these in system development: incomplete requirement specifications, new technologies which are relatively unknown, development methods which only partly serve to organise a sensible development process, customers who are not happy, deliverables

www.dbooks.org

which are behind schedule. In each case the art is to be able to make some progress in the project which also serves to reduce uncertainty. Thus a solution to an incomplete requirements specification could be to prototype some known requirements and then ask the customer to consider what is missing. Now there are two steps forward, there is a small prototype and there is more information about the customer's needs.

- They should be capable of novel and unconventional thinking, and not be locked in to a pre-defined position (for example that a database time stamping problem is always solved with a particular technique or algorithm).
- They should be self-motivated and take the initiative for problem solving or opportunity recognising, and they should not be reliant on a project manager (or other superior) to define their work and monitor whether it is done well.
- They should show persistence – the ability to follow a work-situation through even when it becomes difficult or uncomfortable. However staring at a bug for five hours, when your more experienced colleague could have spotted it in 30 seconds is not persistence, it's bad teamwork.
- They should display heuristic, rather than algorithmic thinking – that is they should be able to respond to developing situations rather than to stick slavishly to a pre-programmed plan of action. Thus experienced professionals seldom follow a given design methodology rigorously, they apply and adapt it to a given development situation.

Most of these competences are not innate – that is they are not determined by the psychological make-up of the individual which cannot be changed. Therefore there are not creative and un-creative personality types, and creativity competences can be evaluated, learned and improved. Moreover competence is always experienced-based as well as expertise-based. This means that developers who know a programming environment and application area extremely well, and have worked on comparable types of problems before are likely to have better creativity competencies than inexperienced developers. They have many relevant experiences which can be drawn on to expand the range of possible solutions that they can consider.

A further creativity competence is the ability to keep many aspects of a problem in play simultaneously: overview. Providing an innovative solution to a development problem may be dependent on the ability to relate facets of the customer's business model, the projected user experience, some usability principles, the underlying data-model, a mathematical theory behind double entry book-keeping and a little-known design pattern. The point is that the solution may not be available to someone who cannot see the bigger picture. Most engineers and analysts have very well developed conceptual modelling

skills - they learn to use many diagrammatic forms for expressing program structures, architectures, information flows and data structures. A program can itself be understood as a model of a part of an external reality. Conceptual modelling skills are very useful in fostering overview, and understanding the relationship between many complex facets of a development task. A simple mind map, drawn on a whiteboard, can be a breakthrough in a system design if it re-organises significant components in the design, and represents them in a simple way which everyone can understand and work towards.

Creativity as a style of thinking

The recognition that personality types do not determine creativity (or lack of it), led American business psychologist William Miller to develop different styles of thinking which could lead to innovation. He developed a questionnaire which would help people understand which kind of innovative personality they displayed. The styles are often used in business consultancy and you can find and take the test on the web.

The four innovation styles are:

- Visioning – using one's instincts, insights, and intuition to focus on an idea for how something could be in the future, and carrying it through with persistence and determination.
- Exploring – taking a new and unknown direction, looking around for solutions in unexpected places, having many competing ideas and following them to see if they lead somewhere.
- Experimenting – meticulously following a series of alternatives in pursuit of a given idea until the optimal solution is found, applying established processes and trial and error.
- Modifying – working with and adapting existing ideas, products and processes to produce something new and useful.

How do you approach a development project? Can you see a new device or application which allows a group of users to work in an entirely new way – you have vision. Do you generate many ideas and solutions to problems and run them past your team to see what reaction you get – you explore. Do you like to play with some rapid development tools and make a few screens or simple prototypes to stimulate the imagination – you experiment. Do you look for an open source solution or refer to a code reuse library as a starting point - you modify. Most people tend to one of these styles – but they can in some respects be seen as complementary. Thus innovative teams can be put together by combining developers with different innovation styles.

www.dbooks.org

Creativity as meta-thinking: recognising unconscious pre-dispositions

Though we like to imagine ourselves as independent thinkers, this is not really the case. In reality our thinking is the product of the way we interacted as children in our families, of our many thinking habits and experiences, and of the things people around us believe in teams, organisations and societies. This produces a strong tendency to think in conventional and well-tried patterns. These patterns can be expressed as mindset. Mindset is 'a set of assumptions, methods or notations held by one or more people or groups of people which is so established that it creates a powerful incentive within these people or groups to continue to adopt or accept prior behaviours, choices, or tools' (Wikipedia). When a particular way of thinking is unquestioningly adopted by a group of people it is known as groupthink. Checkout out your own mindset: what is the core discipline of system development? Is it programming, development methodology, or business logic? If the answer is obvious, then you have a mindset. There are various other ways of describing mindset: the German sociologist Dilthey used the term 'Weltanschaung' to describe an underlying basic view of the world, whereas his British colleague Giddens used the idea of structure to describe common rules and procedures adopted by societies. What makes mindset difficult to deal with and potentially inhibiting for creativity is that it is always, at least to start with, unconscious. If you know you have a mindset, then you can also change it, which means that it is not really a mindset. However, if you have a way of thinking which you normally adopt, but is unconscious, how are you supposed to recognize and change it? Maybe most of the programs you build have an architecture that resembles model-view-controller, without you even noticing it. You would need someone to point this out in order to be aware of it, and come up with an alternative strategy. Even so, you might be quite resistant to adopting a different architecture – arguing, quite reasonably, that it had always produced good results in the past. You might even be quite nervous at launching out in an unknown direction. There are many techniques for exposing and working with mindsets (for example Mitroff's 'assumption surfacing, and dialectical techniques), however one of the most sustained insights into how to recognise mindset and develop creative alternatives (also known as thinking out-of the-box) is the lateral thinking of Edward de Bono. Lateral thinking is contrasted with vertical thinking – orthodox, logical, unimaginative, but effective in many situations. A serious of examples, games and techniques are used to break up the normal pattern of vertical thinking (which is dependent on mindset), and introduce another style which is inventive, playful and discontinuous. The intention is to go beyond conventional linear logical thinking and generally-held assumptions.

In a development team situation it is important to be able to challenge the mindset of others, provoking uncharacteristic reactions. It's neither necessary nor advisable to continue with waterfall development or adding

piecemeal code to business systems written in COBOL simply because that's the way it has always been done in the place you work in. However one of the marks of a really creative developer is the ability to recognise and change their own mindset, if it is challenged.

Creativity as whole-brain thinking: beyond rationality

Since Freud we have understood that our conscious thinking patterns, the rational voice that we hear in our head, is only part of consciousness, and possibly a rather insignificant part. Our thinking patterns can be understood as the conscious (what we are aware of and have access to), the pre-conscious (what we are unaware of but can get access to), and the unconscious (what we cannot be aware of). Our education as programmers and system developers is heavily analytic and concentrates rather heavily on developing our conscious rationality. Brain researchers have demonstrated that the two hemispheres of the brain have rather different functions. In very general terms, the left side deals with the rational, whereas the right side deals with the intuitive.

LEFT BRAIN	RIGHT BRAIN
uses logic	uses feeling
detail oriented	"big picture" oriented
facts rule	imagination rules
words and language	symbols and images
present and past	present and future
math and science	philosophy & religion
can comprehend	can "get it" (i.e. meaning)
knowing	believes
acknowledges	appreciates
order/pattern perception	spatial perception
knows object name	knows object function
reality based	fantasy based
forms strategies	presents possibilities
practical	impetuous
safe	risk taking

Again our training is very heavily oriented towards developing the left brain function; however it's easy to conclude that many of the abilities and thinking styles we have been considering are really right brain functions. Maybe to develop our creative abilities as system developers, we also have to focus on right brain functions.

www.dbooks.org

Creativity as a state of mind

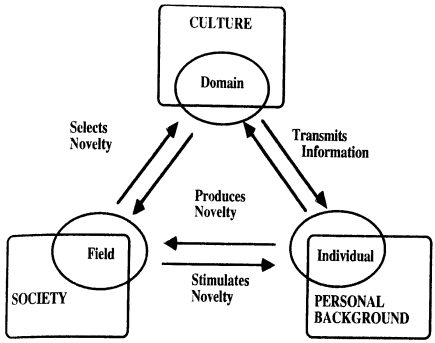
Most people are familiar with the kind of intense concentration experienced when involved in a computer game. Others experience it when playing chess, or on the football field. Many also experience it when absorbed in their work. Csiksentmihalyi became interested in this state of mind and gave it a name: flow. Someone in this mental state experiences most of the following, (according to Csiksentmihalyi)

- clear goals at every stage
- immediate feedback
- challenge/skill balance
- action and awareness merged
- distractions excluded from consciousness
- no worry of failure
- self-consciousness absent
- time distortion
- activity becomes autotelic (an end in itself)

Thus a teenager absorbed in World of Warcraft is perhaps following a particular quest (goal), controlling their avatar on the screen with rapid finger movements in battles with others (feedback) working towards achieving the next skill level which is within their reach (challenge, success certainty), completely lost in the game to the complete exclusion of everything around them (awareness, distractions). The distinction between the player and the avatar is blurred (absent self consciousness), the game may be played for hours (or days), but the player is unaware of time, and the only purpose in playing is to reach a higher level in the game. Playing a computer game is normally understood as a leisure activity (and can also be addictive and harmful), but Csiksentmihalyi also associates the state of flow with creativity. Most programmers will also recognise that periods of their work are spent in flow, and this usually indicates that the work is going unusually well, and they are achieving a great deal. Those periods outside the state of flow, where nothing will work and one stares at a blank screen without being able to get ideas expressed in code, are correspondingly frustrating and unproductive. Most other development work can also induce the flow state; one can be just as engaged in developing an entity relationship model, or sitting with a user developing a use case. It's easy to see that these periods are unusually productive, but the reason that Csiksentmihalyi associates flow with creativity, is that one can observe creative people (for example leading artists and scientists) working in the state of flow for exceptionally long periods. Flow is an easily understood measure of creativity – when you simply have to get the screen you are working on finished and you look at the time and you missed lunch, then it's probable that you were in flow, and also likely that you were unusually productive, and unusually creative.

One potentially negative consequence: flow is essentially personal, whereas software development is team work. A team member who is solely focused on his own project, and not communicating with other members of the team risks producing a lot of work which is misdirected in terms of the overall goals of the project. The result may be code design which cannot be integrated with other parts of the system, divergent interpretations of requirements and use-cases, presentation screens which do not match the underlying database and a host of other problematic consequences.

Creativity as a relationship between the developer and the outside world



In this section, we will introduce the idea that even though we are here focused on the individual developer, we cannot really divorce this individual from the systems of people and ideas that they are engulfed by. Csiksentmihalyi demonstrates this with his systems model, where he relates the individual, with their thinking process and creative attributes to two other important considerations. The first is domain: a set of symbolic rules and procedures which make up the creative field of enquiry. In system development the domain is the world of relational algebra, agile methods, process re-engineering, nondeterministic polynomials and Bayesian networks - the system of thinking that constitutes the professional and research worlds of system development. It is against this extensive system of thinking, that we determine whether a contribution is novel - whether it constitutes an advance in science or practice. The second consideration is field: the people who act as gatekeepers to that domain. Here we must understand that the system of ideas is operated by a community – these are people who work with the domain. Without the community who work with the system of ideas, the ideas themselves are relatively meaningless. The field will determine whether a thinking act is creative, by evaluating it against the current domain. Meaningful advances will then be incorporated in the domain.

Csiksentmihalyi provides us with a useful reminder that it is rather pointless to speak of personal creativity – it is only set in relation to field and domain that creativity acquires validity and meaning.

www.dbooks.org

Creativity as a universal mental skill to be enhanced

Having understood creativity as a positive and productive force, which is not determined in our character or personality (though it may be suppressed through our experiential development) we should regard it as a universal mental skill to be encouraged and enhanced. We can recognise and focus on development project situations where we have unusual productive energy and allow our curiosity space to unfold. We can value and cultivate flow, foster divergent thinking and have the courage to communicate our ideas even when they seem unconventional. We can listen to the ideas of others without dismissing them if they seem strange at first hearing. We can challenge those practices which seem determined by custom rather than effectiveness. We can experiment, and learn not to think of ourselves as failures when something goes wrong or we make mistakes. We can strengthen our knowledge in the domains that we choose and become experts, and we can learn to use our reflective skills to include many facets of our experience.

Work-style heuristic 5 - develop your personal creativity

This chapter investigated eight different perspectives on personal creativity in software development. We studied creativity in the software development process as:

- the developer's mental process: recognising and exploiting discovery points
- a set of personal development competences concerned with both solving problems and recognising opportunities
- a style of thinking associated with different strengths in individual's development personalities
- meta-thinking: recognising predispositions and tendencies in one's own (and others') thinking and coming beyond them
- whole-brain thinking: beyond rationality
- a state of mind: the way the developer's mind is disposed when being creative (flow)
- a relationship between the individual developer and communities of people and ideas (domain, field)
- a universal mental skill to be enhanced

Some developers who want to improve their innovation potential will be able to make progress through changing their mental frame of reference and observing their own practice in relationship to these ideas. Others will want some more concrete help and they are referred to the chapter on tools and techniques.

Sources and further reading

COUGER, J. D. (1990) Ensuring Creative Approaches in Information System Design. Managerial and Decision Economics, 11, 281-25.

COUGER, J. D. (1997) Creativity/Innovation in Information Systems Organizations. System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on, 3.

COUGER, J. D. (1997) Results of a trans-discipline research structure for study of creativity/innovation in IS. System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on, 3.

CSIKSZENTMIHALYI, M. (1997) Creativity: flow and the psychology of discovery and invention, Harper Perennial.

DE BONO, E. (1971) The Use of Lateral Thinking: A Textbook of Creativity, Penguin.

MILLER, W. C., COUGER, J. D. & HIGGINS, L. F. (1993) Comparing innovation styles profile of IS personnel to other occupations.

SIMON, H. A. (1982) Models of Bounded Rationality: Behavioural Economics and Business Organisation, Cambridge, Mass., The MIT Press.

WALLAS, G. (1926) The art of thought, J. Cape.

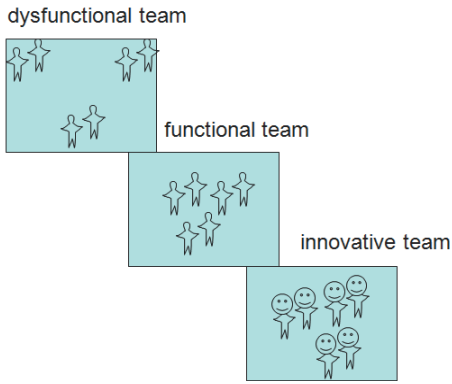
WALZ, D. B. & WYNECOOP, J. (1994) Creativity and Software Design - is formal training helping or hurting? Systems, Man, and Cybernetics, 1994. 'Humans, Information and Technology'., 1994 IEEE International Conference

www.dbooks.org

6. Be a super-team-worker: the innovative software team

Software is not normally built by individuals (though individuals sometimes get credited with breakthroughs) but in teams. When we want to understand how innovative teams function, then we need to turn primarily to the literatures of management science. An innovative software team is, by definition, not dysfunctional – this is helpful to us because we know quite a lot about the different ways in which teams go wrong. However a functional, well-performing team is not necessarily innovative – it can function well in performing relatively routine tasks such as maintaining a banking system.

In trying to understand how an innovative team works, we will need to go beyond understanding how a dysfunctional team becomes functional. We can examine the psychological, managerial and physical environment of the team, the composition of the team, the team processes (the way the team members communicate and work together), how the team learns, how it integrates the different expertise of its members, how it develops a common understanding or overview of the project it is working on, how it accommodates divergent thinking and the social patterns and interactions of the team members.



If the image of a software team conjures up an image of 7-10 developers working intensively at their desks in an open plan office, then it's worth remembering that software teams can be quite diverse in nature. This is a co-located small project team, but teams can be larger (or smaller) and can be geographically distributed. Teams can be composed only of core members, or may have many associated experts playing tangential roles who come in to resolve specialised problems. They may interact with their external environment (customers for example) or be locked away from it. Distributed teams work together with a number of communication tools: conference calls, shared programming screens, net whiteboards, file sharing systems and elaborate collaborative work tools (CSCW) or shared programming environments and version control tools such as Subversion. Work can be synchronous with frequent IM communication, or asynchronous through file sharing. Thus there can be frequent face-to-face interaction, or none at all - as in some open source projects. SourceForge provides an internet working environment for open source projects which supports many forms of interaction and tools for communities working on code bases.

Teams can have an all-powerful project manager or be self-organising. Some projects document every comma in their code, and some let their application speak for itself. Most software development teams are commercial, and eventually have to earn a profit for their companies, but others work in universities where they are funded by the state but have to publish their research. Still others work in open source project where the motivation has more to do with belonging to a community. This is a varied picture of teams and teamwork, but there is little research to suggest which of these various factors make a team especially innovative. We will simply assume that most types of team and most types of teamwork can promote innovation.

We will first take a look at what is known about negative factors in team operation, in the form of creativity barriers and group dysfunction. Then we will move on to more positive approaches to supporting the innovation potential of software teams:

- software team roles
- communicative interactions
- the accommodation of divergent thinking
- team learning
- overview (common purpose)
- expertise integration
- social practice patterns
- environmental scanning.

Creative/innovative work environments: barriers

Every development team operates in a work environment which is mainly beyond its own control. This environment is made up of standardised work practices, resource constraints, physical surroundings, management practice, and psychological and cultural factors to do with the way people normally interact with each other and the cultural norms operating in the work situation. Research shows that certain environmental conditions function as barriers to creativity – that is, they inhibit innovation in the workplace. It's also possibly, through instruments such as the Work Environment Inventory, to investigate work conditions and isolate factors for improvement. The main creativity barriers are:

- Workload/time pressure – excessive pressure on teams and individuals provokes stress and reduces scope for reflection and experimentation, effectively ruling out exploring, modifying and experimenting (see Miller's innovation styles). Thus developers running between several projects, behind their delivery dates and running out of development hours to finish their project are in much need of creative thinking, but badly placed to deliver it.

www.dbooks.org

- Stress – has many psychological and physiological consequences (particularly when experienced over longer periods) which reduce developers' productivity. It is sometimes accompanied by a misleading 'high' of excitement and drive, which leads the developer to mistakenly believe that they are performing well. Team interactions normally become poorer when the group is stressed. Creativity is associated with heightened performance, not lowered performance.
- Resource shortage – forces development teams into well-known practices and constrains experimentation and exploration.
- Rigid work practices – standardised and enforced work practices, such as strict protocols for requirements specifications, documentation requirements, required adherence to a particular development method, are thought to inhibit creativity. This is because they encourage algorithmic (follow a set of instructions) thinking rather than heuristic thinking. These techniques can, of course be productive in more routine situations.
- Bureaucracy – is often concerned with enforcing adherence to rigid work practices. Developers over-document their work processes primarily so that their managers can be certain that the correct processes have been followed.
- Inappropriate evaluation systems – many software firms have evaluation procedures which do not really reward creativity – developers are promoted or rewarded because they get on well with the managers, because they are reliable, or because they have been with the company for many years. Innovation skills can be perceived as dangerous or provocative, especially where they are associated with divergent thinking or challenging prevailing mindsets.
- Reward systems that penalise mistakes – innovative projects will inevitably fail from time to time, because innovation requires a certain acceptance of risk (new applications with untested markets, new programming techniques which may prove troublesome as developers learn them, a venture into a new technology). If the failure means, in a particular software firm, that the project manager will never be put in charge of another project, then this will act as a strong disincentive to innovation.
- Routine work – too much is dispiriting for creative individuals and teams.
- Poor project management - authoritarian project management styles may work in routine situations of moderate complexity, but they

discourage many of the characteristics of innovative team work, such as dialogue and evolving shared purpose.

The natural conclusion is that an innovative software team needs some basic external conditions to be in place in order to be more than simply functional. If we reverse the 'barrier' way of thinking we can begin to envision creative work environments enhancers: freedom, empowerment, challenging work, sufficient resources, supervisory encouragement, workgroup support, organisational encouragement, professional recognition, good teamwork, harmony, cohesiveness, shared vision, team learning, creativity training, mentors, role models, networking, multi-disciplinary teamwork, creative tension.

Group dysfunction

Management researchers also know quite a lot about the signs for, and causes of dysfunctional group interaction and poor teamwork. Some major dysfunctions are associated with

- Destructive dominance – a group member or members exhibiting unusually high influence over other team members, thus preventing other team members from expressing their own ideas or getting them adopted, and steering the project in sub-optimal directions.
- Freeloading – the reverse: team members contributing little or nothing. Innovative teams are performing exceptionally well, which means that they need to maximise the contributions of all the members.
- Conformance – the emergence of group norms ('we need a database solution,' 'it's impossible to avoid a significant degree of error in calculating geo-coordinates') and a consequent unwillingness to speak up with a better solution.
- Conflict avoidance – unwillingness to challenge ideas due to fear of unpleasant personal confrontations – sometimes resulting in group decisions which are contrary to the desires of a majority of members.
- Destructive conflict – its reverse: prolonged unresolved personal antagonism leading to an unproductive psychological climate, and posturing behaviour more rooted in the underlying conflict than the progress of the work of the team.
- Anchoring - digression from the main goals of the project in response to the tangential interests of powerful group members.
- Search behaviour - premature commitment to under-researched solutions primarily due to anxiety about progress in the project.

www.dbooks.org

- Groupthink – the group equivalent of mindset, where a set of ideas become so powerfully entrenched that no one is able to think of alternative solutions. This kills divergent thinking, an important component of innovation.

Some or all these phenomena are seen in most dysfunctional teams. An innovative team is a team with super-function, where there is not normally room for these kinds of problem. As we will see, the functioning of the super-team will need to enable some relatively difficult forms of interaction, including divergent thinking, expertise integration, overview development and revision, and relatively unusual social patterns.

Having understood something of creativity barriers and group dysfunction we turn to some positive features of teamwork that can promote innovation.

Innovative team roles:

Every software project manager wants to get certain developers on their team. The reasons are sometimes a mixture of practicality and expedience. This developer is good with network configuration, this one has a wide experience with .asp, a third can be relied on to work steadily and not complain, a fourth is good friends with a senior manager that should be impressed. Innovation is expertise-based, so innovative teams are usually composed of people are very good at what they do. However the right blend of technical expertise dependents on the particular project, so here we will focus more on innovation roles. Team roles are a way of thinking about what team members contribute to their team and how these contributions can produce a synergy that can make the team much more effective and productive than its members could be if working alone. Classical team roles include:

- idea generators – those who come up with many varied ideas and problem solutions
- entrepreneur/product champion – the visionary who can hold the eventual goal in focus
- project manager/leader – the person who takes charge and organizes the work of the team
- gatekeepers/boundary communicators – those who communicate with people outside the project
- sponsor/coach – senior figures providing organizational legitimacy and encouragement.

Meridith Belbin, working in the field of management psychology contributed the best scientifically underpinned taxonomy of team roles:

- plant - creative, unorthodox and a generator of ideas, often the divergent thinker.
- resource investigator – the vigorous pursuer of contacts and opportunities, focused outside the team, maker of possibilities, networker.
- coordinator - confident, stable and mature, task delegator, decision maker.
- shaper - task-focused leader with drive and energy – the winner type, committed to achieving goals and channelling the team members towards these goals.
- monitor evaluator - fair and logical observer and judge, analytical thinker and rational evaluator of options.
- team worker - good listeners and diplomats, talented at smoothing over conflicts and helping parties understand each other without becoming confrontational.
- implementer – converter of ideas into positive action, efficient and self-disciplined, can be relied on to deliver on time.
- completer finisher - the perfectionist who goes the extra mile to make sure everything is delivered on time and works perfectly.
- specialist – expertise, concentration, ability, and skill in a particular field.

These roles are presented in positive terms, but Belbin recognizes that all roles also have a negative side when carried to excess, or simply placed in the wrong situation. The plant can continue to supply unorthodox ideas when it is time to implement what has been decided, the networker may introduce all kinds of externalities and lack of focus, the coordinator can be perceived as manipulative and work shy, and so on.

James Coplien adopted the idea of roles in his study of social process in software projects, describing repeating types of figures with a direct influence on the project – for example: patron, solo virtuoso, gatekeeper, matron (social supporter), mercenary analyst (someone brought in to relieve the team of drudge work like user documentation), surrogate customer, legend, wise fool, peace maker, sacrificial lamb, guru, producer, supporter, deadbeat.

It would have been helpful if one of these authorities had provided a recipe for how to compose a software team that was guaranteed to be innovative. Unfortunately this has not been the focus of their studies, and in any case it is almost certain that no such formula exists. Nevertheless it is easy to see that a team composed of, producers, implementers, team workers and completer/finishers will get through a lot of productive work, but will only be innovative in one particular situation, that is when the specification for the innovative software product is delivered to them from outside the team. A

www.dbooks.org

team composed of plants and shapers will have originality, drive and ambition – but no-one to deliver the solid work necessary for making progress. Innovative teams will have different compositions in different circumstances, but it's an interesting exercise to take your current project, analyze the role distribution, and imagine an optimally creative composition of roles.

The idea of role is incorporated in modern agile methods. Here, instead of analyzing roles of team members based on their personality types, the authors ask developers to consciously adopt roles, with the understanding of what the role entails. In Beck's eXtreme Programming, the roles of coach, programmer, tester, tracker, consultant and big boss are specified, with some advice on how they should be filled in an effective team. A special role is allotted to the on-site customer – that of simplifying the complexity of relationships with outside stakeholders and users. SCRUM allots the roles of product owner (prioritizing the product backlog), scrum master (a developer with special responsibilities for conducting sprint meetings and protecting the team from outside interference) and chicken (those who may observe but not interfere). These role allotments are concerned with making the development team more effective, but in Aaen's Essence the idea is further developed to focus on creativity. Here roles are designed to ensure that the team develops the necessary creative tensions for innovation:

"Team members have roles defining their characters. Each role has a set of ideals or values providing a clear raison d'être to the role. The Challenger is the customer and has all the responsibilities of an on-site customer, yet should pose project requirements in the more open form of challenges. The Responder is the developer employing technical competence to deliver ambitious responses. These two roles engage in a dialogue where solutions are developed by contrasting application area needs and desires with technical opportunity. The Anchor serves to keep the team absorbed and focused on delivering exciting solutions. The last role is the Child; this role is temporary as anyone on the team can take this role temporarily at any given time. The Child may raise any idea or issue – even when contrary to decisions made earlier by the team. This role is named after the child in Hans Christian Andersen's The Emperor's New Clothes who said 'but he hasn't got anything on' - and thereby revealed the emperor's folly." AAEN, I. (2008) Essence: facilitating software innovation. European Journal of Information Systems, 17, 543-553.

Innovation team interaction

Dysfunctional groups can be said to have dysfunctional group interaction, and much of improving team function is concerned with improving interactions. Interaction comes in three main forms – it can be free interaction, facilitated interaction, or be determined by technique. Most information system development teams use free interaction (without really considering the choices). This means that the group interactions are left to evolve by themselves. Most experienced developers are quite conscious of how their team is functioning, and have various responses (derived through experience) to the periodic interaction threats which are a natural part of the ups and

downs of every project. Often the responses are intuitive, and sometimes unconscious – thus an old hand will smoothly interrupt someone who is being much too pedantic about a small system feature, and make sure the discussion moves on, without anyone really noticing what is going on. Sometimes group interactions move beyond commonplace ups and downs and become dysfunctional. One response to this situation is to employ a facilitator – someone who is experienced at group interaction, and just as important, is external to the team, so that they are not bound up in the internal dynamics of the team's interactions. Other reasons for employing a facilitator can be to improve the performance of the team, or to manage temporary teams with well-defined tasks, such as a user focus group. Here the facilitator comes with a preordained understanding of the task (e.g. usability testing an interface) and the process for solving the task, and steers the group though the session. The third way to influence group interaction is to use a technique designed for the purpose. An example is nominal group technique. NGT consists of five steps

1. participants independently and silently generate a list of ideas
2. the facilitator records one idea at a time going round the group
3. group members discuss each idea for clarification only, without considering its merit
4. participants independently rate and rank the ideas
5. the group prioritizes the suggestions by voting

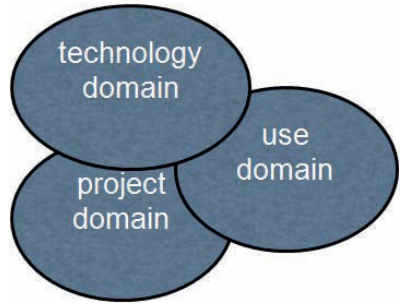
This process effectively removes much of the typical dysfunction described earlier (destructive dominance, freeloading, etc.). However it can easily be experienced as artificial. Duggan experimented with combining this technique with Joint Application Development, with good results.

Many of the techniques used in agile methods are focused on achieving good group interactions – often those that are based on some forms of mutual respect, combined with genuine commitment to the work of the team. Thus a stand-up meeting in SCRUM is designed to ensure daily communication of progress and objectives, and to generate commitment to achieving the day's work. There is eye contact between the team members, and the scrum master is the meeting facilitator, not the project manager delegating tasks. The meeting form encourages a particular type of group interaction. Once more, however, there is no recipe or blueprint for group interactions that result in innovation. Nevertheless, focus on interaction is vital for innovative software teams, which must combine effectiveness with a wider variety of divergent thinking, challenge, creative tension and uncertainty than other development groups. Group processes must enable both a wide spectrum of idea generation, and testing of mental models, with the evolution of common purpose and a high degree of efficiency in implementation.

www.dbooks.org

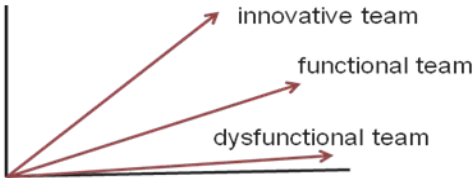
Team learning and innovation

A team with good communicative interaction is in a position to learn effectively. This is important because there is typically much to learn in a development project. Projects differ, and it is impossible to characterise all the things that the team must learn, but a starting point could be to consider three learning domains.



1. Project domain: all those things that are concerned with the organisation and management of the development project, its process, structuring, management, tool support, financing, project members and customers.
2. Technology domain: the hardware, software environment, programming languages, design techniques, architectures and algorithms.
3. Use domain: the understanding of the application area that the software is intended to be used in. The habits and work (or entertainment) patterns of the users, their way of interacting, the purpose and function of the software product in its use context.

It could be noted that when innovation is the focus, the learning curve may be relatively heads-up – with an eye on external competitors, the trends and trajectories of technology evolution, and the use patterns of the future. Technologies may be leading edge – normally involving a very steep learning curve for project members. Thus an innovative development team is normally learning very fast, and very effectively. This allows the project to move quickly, to cover much ground, and to stay in front of other competitor teams working on similar products.



The primary mode of learning in a team is through internal communication, but discussion is not learning in itself. The management theorist Peter Senge points out the difference between discussion and dialogue. Discussion may involve much re-stating of individual positions, various forms of power struggle and negotiation, defensive routines where participants defend their own ideas where they feel them to be under attack, lack of focus, tangential episodes and many other important interactions, which however do not constitute team learning. If the participants' underlying mental models, their ways of thinking, their view of their own private expertise areas and their

underlying attitudes to the project do not change, then team learning is not taking place. If this is difficult to understand, consider the case of three politicians from different political parties publically discussing an issue. Chances are that there is interaction, but no move towards a common position. The participants simply continue to explain their own positions in new and different ways. The basis of team learning, Senge points out, is not discussion but dialogue. In dialogue, group members agree to suspend their individual positions and assumptions and to listen openly to the ideas of others - accepting those ideas as equally valid, but different. The collective mind of the team moves forward and learns through the generation of common understandings and purpose. He specifies three conditions which must apply for dialogue to evolve.

- Participants must suspend their assumptions – particularly the instant unreasoned judgments that they make because of their underlying assumptions.
- Participants must regard each other as colleagues – a special relationship where the other has an intrinsic right to their opinions and arguments.
- There must be a facilitator who stays outside the content of the dialogue, but instead holds the context in play – particularly the first two conditions.

Many dialogues can be observed in system development practice. Two developers argue out their testing strategy with JUnit. They have different, but valid strategies. An experienced developer is coding a few feet away at her desk. She doesn't participate much, but makes it known by her body language that she is following the conversation – she facilitates. The tone of the conversation is constructive, the arguments well-reasoned and technical in nature. Eventually the developers agree, and more importantly, both commit to the testing strategy and the experienced colleague turns back to her screen. The team has progressed; the testing strategy decided, communicated and bought in to.

The ability of a team to learn effectively, and fast, gives it a particular characteristic which is helpful in most innovation situations – agility. Agility is the ability to respond productively to changing situations and circumstances, by changing the direction of (parts of) the project. Innovation projects are likely to throw up many difficulties – leading edge technologies which do not work as expected, unstable, programming environments, user scenarios which are unexplored and therefore subject to sudden alteration. Teams which cannot respond to these problems (my geo-mapping program worked fine on our PC mobile simulators, but crashes on my Nokia phone) cannot make much progress. Agile teams are typically able to respond constructively to many different types of changes in the three learning domains.

www.dbooks.org

Accommodation of divergent thinking

Just as Darwinian evolution is dependent on genetic divergence (small mistakes in the transferral of genetic information in the reproduction of the species), creativity is dependent on divergent thinking. If we all think alike, then innovation is not possible. Creative teams therefore have a special responsibility in the way they handle divergent thinking. A team which is not open to divergent thinking, at least in certain phases of its work, is not likely to learn effectively, integrate its expertise or respond to challenges well. However ideas are not good ideas simply because they are new – in fact most new ideas are not very productive. Some are dangerous. The suggestion that the social networking web system that the team is building should really be a mobile application might be the stroke of genius which will earn the company a lot of money, or a red herring which will derail the project and lead to a catastrophic failure. If the team is closed to the idea then it cannot be adopted anyway – so we will never know. If the team is open to creative ideas, then assumptions must be suspended and a proper dialogue set up to evaluate the merit of the idea – a process which could take hours, days or weeks. If the team produces many divergent ideas, and they all evaluated in this way, then actual progress in designing and building anything will rapidly come to a halt. Many divergent ideas are usually more helpful in the earlier idea generation phases, than two weeks before the deadline, with an expectant customer waiting in the wings. Creative teams develop their own ways of handling these dilemmas. A team member with many divergent ideas (a plant) may be taken more seriously at the beginning of the project. A deadline may suppress all new ideas for a short period of time. At other times there may be long discussion of apparently wild ideas. There may be a sub-group who are prepared to listen and vet ideas before they reach everyone.

Expertise integration

Teams are made up of developers with many different kinds of expertise. Some come with particular competences in databases, java, or web engineering. Others are experienced communicators or project managers. The company's senior architect may pass by to discuss basic design issues, and the company's lawyers may drop in to discuss patent issues. Developers often undervalue a totally different kind of expertise which is displayed by their customers and users – they are normally the experts in the use domain, not the developers. The creativity of a team is associated with the integration of expertise. This is not simply knowledge transfer between individuals. A programmer working on the engine of a computer game may have many conversations with the graphic designers. The graphic design influences how the engine should be configured, and what the engine can do influences the way the game will eventually look and react for its users. Expertise integration means that the different skills of designer and programmer are integrated at the team level – so that the team learning of

the project is re-focused. Integration of expertise, such that team members can meaningfully interact with colleagues with different specialities, increases the range of design possibilities and solutions that are available for discussion. It increases social relational capital – the trust and interaction quality of the team. A team which is able to draw on diverse forms of expertise, and is also able to integrate and absorb this diversity is likely to be highly creative.

Overview: macro + micro integration

Creativity in a development team is also associated with overview. A management theorist like Senge might prefer to call this common purpose. Software development is one of the most complex activities known to humanity, and there are enormously many details, starting with hundreds, thousands, or millions of lines of code. A visionary innovator can define a software product at a high level – a three dimensional operating system interface, for instance – but many details must be in place remain before it can work. A team needs to work in a common direction –to be aligned – to be really productive. This means that the whole team has understood and communicated the more general directions in which everyone is moving – overview. Without this overview (itself a form of team learning), individuals may be productive in their own right, but may work in their own directions, with different understandings of use cases (for example), a different internalisation of an underlying database model, or functionality which duplicates that of a team colleague. Overview can be difficult enough to create in a normal development situation – and a conventional project leader might spend a lot of their time working with this, but the problem is exacerbated in innovation work. There are more unknowns, less routine to fall back on, vaguer ideas about outcomes, and difficulties with unresponsive technologies. There are more challenges to the direction of the project, which the team must respond to in an agile manner by altering objectives and practices. This means that yesterday's overview is likely already to be out of date, and in need of repair. Thus innovative software teamwork combines both a greater need to create viable overview, with a greater difficulty in achieving it. Sometimes the visionary in the team is capable of maintained direction. More often there are too many details and subsidiary expertise involved, and the group must have tools and techniques and the will to constantly improve their overview. Simple tools such as mind mapping may be helpful. However purpose-built innovation facilities, like the Software Innovation Research Laboratory at Aalborg University, often have built-in tools for helping with overview. SIRL has four large interactive screens on its walls, which are normally used to keep four major aspects of the project in play at the same time. This allows the team to focus together on a more inclusive version of their project.

www.dbooks.org

Innovative teamwork patterns

Coplien extended the idea of software design patterns to also cover the social (work) practices of the software teams. He started with the idea that productive teams have habits of working (practices) which are successful over a range of projects. He then researched these patterns in development companies using an appropriate research method – social network analysis.

There are really too many of these patterns to discuss them at length, but Coplien's own identification of the most important (top ten) practices is in the box on the next page. In common with much good research into system development, the object of the research is to identify good practice rather than innovation potential. However we can extend his basic idea to suggest that innovative software teams have particular work practices which encourage creativity. We can also note some similarities with other features of teamwork discussed in this chapter. *Unity of purpose* and *architect controls product* correlate with overview, except for the idea that the project leader bears the sole responsibility for developing it. The customer focus reflects the need to learner about the use domain (amongst other things). Domain expertise is discussed under the heading of expertise integration. In the absence of secure knowledge about which social patterns or work practices promote effective innovation, we must be meticulous in observing our own experience, in order to learn.

Coplien's top ten software practices

unity of purpose - 'the leader of the team must instill a common vision and purpose in all members of the team.'

engage customers – 'it's important that the development organization ensures and maintains customer satisfaction by encouraging communication between customers and key development organization roles – closely couple the customer role with the developer and architect roles'

domain expertise in roles – 'hire domain experts with proven track records, and staff the project with the expertise embodied in their roles'

architect controls product – 'create an architect role as an embodiment of the principles that define an architectural style for the project and of the broad domain experience that legitimizes such a style'

distribute work evenly – avoid concentration of work on a few dependable producer types

function owner and component owner – 'ensure that every function and every component has an owner'

mercenary analyst – 'hire a technical writer who is proficient in the necessary domains but who does not have a stake in the design itself' avoid drudge work and documentation

architect also implements – 'beyond advising and communicating with developers, an architect should also participate in implementation

firewalls – 'create a manager role that shields other development personnel from interaction with external roles'

developer controls process – 'make the developer the focal point of process information'

Environmental scanning

Many software development teams are quite internally focused. The many intrusions and distractions that interfere with the flow of development work, and the (experienced as) noise which can come from managers and customers, quite apart from resource shortages and pressing deadlines, cause developers to tend to put their heads down, and to focus on producing code that works and a solution that can be delivered more or less on time. This is a very natural tendency and in many situations desirable practice, but is less likely to promote innovation. Innovative teams are usually intensely aware of rival products, technology departments, new market opportunities, leading edge scientific developments, practice developments in their specialist fields,

www.dbooks.org

and a variety of other environmental factors. Environment here refers to everything which is not internal to the development project; thus environmental scanning is the process of collecting relevant information from outside. As discussed in chapter 2, technology innovation is seldom conducted in isolation. The popular image of the inventor with the brilliant idea which immediately revolutionises a field is somewhat misleading. Technology innovation takes place in waves, on the backbone of developing infrastructures, amongst communities of experts in relation to technology trajectories, in specific innovation time windows. These factors mean that innovative software teams take a 'heads-up' position – extremely alert to what is happening in their environment.

Work-style heuristic 6 - be a super-team-worker

We have discussed a number of factors which contribute to innovative software teamwork. We can focus an understanding team dysfunction and barriers to creativity in order to repair and remove. We can understand that innovative teams are likely to display

- good understanding and exploitation of roles, especially those which promote creativity
- highly functional communicative interactions, including accommodation of divergent thinking
- high levels of team learning leading to flexible response to challenges (agility)
- good shared understanding of common purpose (overview) even in the situation of rapid change
- constructive software practice patterns – that is productive work practices
- diverse and deep expertise, well integrated
- intense awareness of their environment

Here are some defining questions to ask of your software team:

- which *people* do you want in your innovative team?
- which *roles* should be filled in an innovative team process?
- how structured should the *team process* be (tools and techniques versus free interaction)?
- what is the creativity *environment* for a your team and how can you improve it?
- what are the innovative *work habits* (patterns) of your teams?
- how does the team promote *team learning* and *dialogue*?
- how does the team develop a *shared purpose* and *overview*?
- what kind of automated *tool support* does an innovative team need?

- what learning do you need from outside the project and when do you need it?
- how do you know when the team is working innovatively?

When you have satisfactory answers to most of these questions, then it is possible that you have an innovative team, and that you are a super-team-worker.

Sources and further reading

BECK, K. (2000) Extreme Programming Explained: Embracing Change, Boston, Addison Wesley.

BELBIN, M. (1981) Management Teams, London, Heinemann.

COPLIEN, J. O. & HARRISON, N. B. (2005) *Organizational Patterns of Agile Software Development*, Pearson Prentice Hall.

COUGER, J. D. (1996) Press: Measurement of the Climate for Creativity in IS Organizations. *Creativity and Innovation Management,* 5**,** 273-279.

DUGGAN, E. W. (2003) Generating Systems Requirements With Facilitated Group Techniques. *Human-Computer Interaction,* 18**,** 373-394.

DUGGAN, E. W. & THACHENKARY, C. S. (2004) Integrating nominal group technique and joint application development for improved systems requirements determination. *Information & Management,* 41**,** 399-411.

JORDAN, P. W., KELLER, K. S., TUCKER, R. W. & VOGEL, D. (1989) Software storming: combining rapid prototyping and knowledge engineering. *Computer,* 22**,** 39-48.

LOBERT, B. M. & DOLOGITE, D. G. (1994) Measuring creativity of information system ideas: an exploratory investigation. *System Sciences, 1994. Vol.IV: Information Systems: Collaboration Technology Organizational Systems and Technology, Proceedings of the Twenty-Seventh Hawaii International Conference* Hawaii.

LYYTINEN, K. & ROSE, G. M. (2006) Information system development agility as organizational learning. *European Journal of Information Systems,* 15**,** 183-199.

SAMPLER, J. L. & GALLETTA, D. F. (1991) Individual and organizational changes necessary for the application of creativity techniques in the development of information systems. *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System* Sciences. Hawaii.

SENGE, P. M. (1990) The Fifth Discipline, London, Century.

TIWANA, A. & MCLEAN, E. R. (2003) Expertise Integration and Creativity in Information Systems Development. Journal of Management Information Systems, 22, 13-43.

AAEN, I. (2008) Essence: facilitating software innovation. European Journal of Information Systems, 17, 543-553.

www.dbooks.org

7. Bring your toolbox: creativity tools and techniques

In this chapter we investigate the developer's repertoire – that is the toolset that a developer has available to help with creativity and innovation. The thesis will be that an innovative software developer or team has two basic competences in relation to toolsets. The first is *repertoire* – that is knowledge of tools and techniques which may be used to support various aspects of the development process, particularly in regard to creativity. The second is *selection* – that is the ability to choose the tool or technique which is productive in the particular development situation at a particular point in time.

The terms tool and technique are often used interchangeably, but here we will primarily use 'tool' to mean a piece of software, and 'technique' to mean an abstract procedure for doing something. Every developer has an existing repertoire of tools and techniques derived from their education and experience which from time to time can be used in a creative way, or to improvise a solution to a problem, so here we focus on tools and techniques for innovation.

Creativity tools

Information system development is heavily automated. It comes with many software tools which make the task easier – typically automating drudge work. Software developers don't function well without their compilers, translators, programming interfaces, diagramming and case tools, version control tools, and many others. There's a software tool that supports every development task, from requirements management to test and implementation, if the developer cares to use them. Access to the internet has largely replaced volumes of documentation of the various programming technologies involved. Tools automate routine work, and structure large quantities of information, but can also create unnecessary bureaucracy and divert attention from more important tasks. It follows that the developer working with innovation will also require tool support, and this section asks the question: what kind of software tool support can underpin innovative software projects? The research literature is again too thin to be really helpful, so the section will end with a proposal for a toolbox which is a form of summary of the various research contributions.

Characteristics of applications supporting creativity

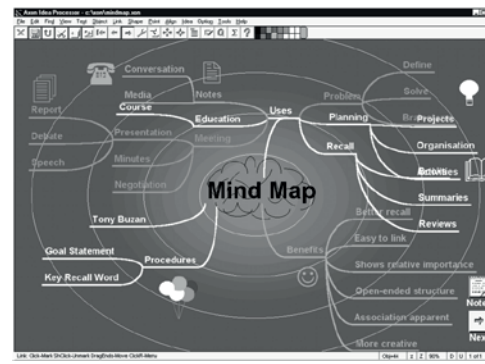
There is a fair amount of research into applications that support creativity, where creativity is understood as a generalized scientific or artistic phenomenon - that is, not particularly directed at information system development. The tools described in this literature include automated mind mapping, argument support tools (for structuring discussion), tools for

enhancing arts (painting and music) experiences, a dialysis simulation tool (which helps to explore various dialysis scenarios) a tool for visualizing blood protein chemistry in a three dimensional graph, and a gearbox design tool. According to Greene, creativity software tools support:

- (pain free) exploration and experimentation (sandbox mode)
- engagement with content to promote active learning and discovery
- search, retrieval and classification
- collaboration
- iteration
- instructive mistakes
- domain-specific actions

He is talking about a wide variety of tools supporting creativity in many fields (including the arts), but his characterization supports our emerging picture of a development process which is engaging, allows space for experimentation and errors and has a high learning curve. Shneiderman, also speaking generally, specifies the tasks for such tools:

- searching and browsing digital libraries, the web, and other resources
- visualizing data and processes to understand and discover relationships
- consulting with peers and mentors for intellectual and emotional support
- thinking by free associations to make new combinations of ideas



- exploring solutions—what-if tools and simulation models
- composing artifacts and performances step-by-step
- reviewing and replaying session histories to support reflection
- disseminating results to gain recognition and add to the searchable resources

As the list below shows, there are many software tools available for purchase or download. Many of these tools can support aspects of a software project (for instance initial idea generation) but there is little or nothing that is directly intended for the software developer. The examples given in the literature are eclectic and the frameworks rather generalized. In addition

www.dbooks.org

there are potentially many aspects of software innovation to be supported, and many different use situations.



If we try to translate these principles into a software context we could understand that experimentation and exploration often take the form of trying out different system ideas, where very low tech prototypes (sketches and paper mock-ups) can be replaced by digital and code prototypes as ideas shape up. Often these will serve as the starting point for exchanges of ideas with users. Developers will be engaged in active discovery and learning in both the technology and use domains – generating constructive technical solutions, perhaps with cutting edge or unfamiliar technologies, and understating their products innovation potential – in particular its utility for a particular user community. Developers can spend much research time on the net: searching code libraries, looking at related open source projects, understanding the use context, assessing competing development platforms and environments, and honing their programming skills. Sometimes this knowledge is recorded or codified for later use or for dissemination amongst other developers in a knowledge base, wiki or intranet. Software development in an innovative context makes heavy demands on communication and collaboration. Some of this is internal to the project:

expertise exchange, developing common purpose, design decisions and standards, alterations to the code base, refactoring. Other aspects are external, involving collaboration with clients and users: contextual studies, use cases, requirements, feature list discussions, prototype demonstrations and so on. Many aspects of this collaboration can be supported in relatively simple ways; for instance, if your user does not have time to play the role of on-site customer and be physically present all the time, then they can at least be available for a video conference on Skype. Much innovative software development is relatively iterative – applications are seldom built perfectly first time in increments; rather they are experimented with, prototyped, discussed, improved, thrown out, added to, tested, demonstrated. In addition there is much routine drone work - 1% inspiration and 99% perspiration, as the cliché has it. Software tools have the potential to contribute here. A further role for software tools is for visualisation. Many system analysis and design techniques are essentially concerned with visualisation - diagrammatic representations of complex sets of relations - and are supported by tools like Rational Rose. Architectural design communicates a unifying structure to a design and is also often visualised in a diagram.

A software support toolbox

As a way of summarizing this debate, and focusing it on software innovation the following software support toolbox is proposed:

purpose	examples
support for escaping routine work	programming editors, visual editors, case tools and diagrammers, project management tools, code management and versioning tools
sandbox tools	prototyping tools, screen painters, demo makers, animation and slide-show software, visual RAD tools
knowledge tools	search tools, technical problem solving and coding documentation sites, knowledge base, wiki, experience exchange, idea repository
collaboration tools: internal (project), external (customers and other stakeholders)	co-operation and communication tools, collaborative writing, social network support, dialogue support
visualization and overview support	simple diagrammatic support for visualizing and agreeing common purpose in the face of complexity: mind maps, Microsoft Visio, case tools
creativity technique support	support for particular creativity techniques used in the project

www.dbooks.org

The toolbox constitutes a *repertoire* – an available set of tools which can be drawn upon. More is not normally better, in the sense that most tools involve a learning curve, input demands and some bureaucracy. Most also threaten with goal displacement – that using the tool will become the goal, rather than improving innovation performance. Adding Microsoft Project and Rational Rose to a three month, three man project is more likely to drown it in documentation than add to its creativity. *Selection* – adding the right support for the particular project - is the key.

Creativity techniques

Creativity can be enhanced and there are many techniques which can be used to stimulate it. It's not really the purpose of this chapter to describe these in detail, but some of the best known are:

- Assumption Surfacing
- Brainstorming
- Card Story Boards
- Causal Mapping
- Crawford Slip Writing
- Dialectical Approaches
- Five Ws and H
- Lateral Thinking
- Mind Mapping
- Nominal Group Technique
- SWOT Analysis
- TRIZ

There are many descriptions of these kinds of techniques at Wikipedia and other places on the web. The following list is adapted from the wiki Mycoted provide.

7 Step Model
AIDA
ARIZ
Advantages, Limitations and Unique Qualities
Algorithm of Inventive Problem Solving
Alternative Scenarios
Analogies
Anonymous Voting
Assumption Busting
Assumption Surfacing
Attribute Listing
Backwards Forwards Planning
Boundary Examination
Boundary Relaxation
BrainSketching
Brainstorming
Brainwriting
Browsing
Brutethink
Bug Listing
BulletProofing
Bunches of Bananas
CATWOE
Card Story Boards
Cartoon Story Board
Causal Mapping
Charette
Cherry Split
Chunking
Circle of Opportunity
Clarification
Classic Brainstorming
Collective Notebook
Comparison tables
Component Detailing
Concept Fan
Consensus Mapping
Constrained BrainWriting
Contradiction Analysis
Controlling Imagery
Crawford Slip Writing
Creative Problem Solving - CPS
Criteria for idea-finding potential
Critical Path Diagrams
DO IT
Decision seminar
Delphi
Dialectical Approaches
Dimensional Analysis
Disney Creativity Strategy
Do Nothing
Drawing
Escape Thinking
Essay Writing
Estimate-Discuss-Estimate
Exaggeration
Excursions
F-R-E-E-Writing
Factors in selling ideas
False Faces
Fishbone Diagram
Five Ws and H
Flow charts
Focus Groups
Focusing
Force-Field Analysis
Force-Fit Game
Free Association
Fresh eye
Gallery method
Gap Analysis
Goal Orientation
Greetings Cards
Help-Hinder
Heuristic Ideation Technique
Hexagon Modelling
Highlighting
Idea Advocate
Idea Box
Ideal Final Result
Imagery Manipulation
Imagery for Answering Questions
Imaginary Brainstorming
Implementation Checklists
Improved Nominal Group Technique
Interpretive structural modeling
Ishikawa Diagram
KJ-Method
Keeping a Dream Diary
Kepner and Tregoe method
Laddering
Lateral Thinking
Listing
Listing Pros and Cons
Metaplan Information Market
Mind Mapping

www.dbooks.org

Morphological Analysis
Morphological Forced Connections
Multiple Redefinition
NAF
NLP
Negative Brainstorming
Nominal Group Technique
Nominal-Interacting Technique
Notebook
Observer and Merged Viewpoints
Osborn's Checklist
Other Peoples Definitions
Other Peoples Viewpoints
PDCA
PIPS
PMI
Paired Comparison
Panel Consensus
Paraphrasing Key Words
Personal Balance Sheet
Pictures as Idea Triggers
Pin Cards
Plusses Potentials and Concerns
Potential Problem Analysis
Preliminary Questions
Problem Centred Leadership
Problem Inventory Analysis - PIA
Problem Reversal
Productive Thinking Model
Progressive Hurdles
Progressive Revelation
Provocation
Q-Sort
Quality Circles
Random Stimuli
Rawlinson Brainstorming
Receptivity to Ideas
Reframing Values
Relational Words
Relaxation
Reversals
RoleStorming
SCAMMPERR
SCAMPER
SDI
SODA
SWOT Analysis
Sculptures
Search Conference
Sequential-Attributes Matrix
Similarities and Differences
Simple Rating Methods
Simplex
Six Thinking Hats
Slice and Dice
Snowball Technique
Soft Systems Method
Stakeholder Analysis
Sticking Dots
Stimulus Analysis
Story Writing
Strategic Assumption Testing
Strategic Choice Approach
Strategic Management Process
Successive Element Integration
SuperGroup
SuperHeroes
Synectics
Systematic Inventive Thinking
TILMAG
TRIZ
Talking Pictures
Technology Monitoring
Think Tank
Thinkx
Thril
Transactional Planning
Trigger Method
Trigger Sessions
Tug of War
Using Crazy Ideas
Using Experts
Value Brainstorming
Value Engineering
Visual Brainstorming
Visualising a Goal
Who Are You
Why Why Why
Wishing
Working with Dreams and Images

Different techniques can suit different situations, different development tasks, different people, and different group dynamics. With so many techniques available, selection becomes a real problem, and there are some taxonomic schemes which try to group techniques by function. Mycoted use the scheme:

- Process
- Problem Definition
- Idea Generation
- Idea Selection
- Idea Implementation

A more complex scheme for idea generation techniques (provided by Martin Leith) revolves around the worldviews that the techniques exemplify

- Worldview 1 - The World is a Machine (based on rational cause and effect thinking, and first order change - emphasis on producing many ideas and selecting the brilliant one)
- Worldview 1 Plus - The World is a Network of Relationships
- Worldview 2 - The World is a System (complex issues are addressed through context manipulation, pattern analysis and constraint removal)
- Worldview 3 - The World is a Field of Energy and Consciousness (involves heightening the perception of the idea generator)

Each of the worldviews has several categories:

www.dbooks.org

Worldview 1 - The World is a Machine	Inventory Making Combining Deconstructing Building Springboards Ideas across Frontiers Constraint Removal Laddering (moving across abstraction levels) Anchoring and Spatial Marking (from neuro-linguistic programming) Working Backwards
Worldview 1 Plus - The World is a Network of Relationships	Conversational Collaborative
Worldview 2 - The World is a System	Break The Rules Do More Of What Works Minimalist Intervention
Worldview 3 - The World is a Field of Energy and Consciousness	Experiential Shamanic

Some well-known techniques (for example mind-mapping) have easily obtainable freeware tools. With so many different techniques available there are problems both of repertoire (which techniques should one have available) and selection (which technique is appropriate to a given situation). The techniques considered here vary a great deal in scope, purpose and the level of background expertise necessary (from read once and run to full blown academic problem solving methods requiring several weeks of study to master). However most creative people have their favourite techniques, and an instinctive idea of when they should be used.

Schneiderman spent some time categorizing the primary types of creativity tools and techniques: They tend to support (individually or in some combination).

- establishing purpose and intention
- building basic skills
- encouraging acquisition of domain-specific knowledge
- stimulating and rewarding curiosity and exploration
- building motivation, especially internal motivation
- encouraging confidence and a willingness to take risks
- focusing on mastery and self-competition
- promoting supportable beliefs about creativity
- providing opportunities for choice and discovery
- developing self-management (meta-cognitive skills)

- teaching techniques and strategies for facilitating creative performance

Creativity techniques that are aimed directly at software and system development are much rarer, however. Some general techniques can be fairly easily adapted to software development; for example Luke Hohman's innovation games. Re-fashioning a work process as a game is a deliberate strategy for opening different creative thinking possibilities – starting with the idea that what is happening is entertaining and open, rather than the routine execution of a generic work process. Hohman's games support: deciding product features (requirements analysis), understanding projected customer (user) experience with the new product, understanding the market relationship of the product (software application) with other products in the market, and a variety of other useful innovation processes. The focus is upon 'ideation' – relatively early development of product ideas in relation to customer needs. They are also focused on teamwork – the subject of an earlier chapter.

A starting repertoire of creativity techniques for software development

This section provides some suggestions for useful techniques based on work conducted in the Software Innovation Research Laboratory and the education programmes at Aalborg University. In each case there is a short description of the technique together with its role in the software development process.

Brainstorming

Brainstorming (introduced by Alex Osborn) is a technique for generating new ideas which is so well known that it needs no description. However, there are some simple rules and techniques for optimizing a brainstorming session, which are often ignored, and it's wise to look these up first and use them.

We typically use brainstorming for establishing the very first ideas about a product and its features. Both users and developers can easily contribute. It also establishes a good creative dynamic in the group, where many ideas come into play, divergent thinking is welcomed and an evaluative rather than critical tone of communication is established. If users (or other stakeholders) and developers work together, it also establishes the principle that good ideas may come from anywhere (not just a user-driven requirements list), and that they also need to be well communicated to people with different expertise backgrounds.

Backward mapping

Backward mapping is a visioning technique that starts from the premise that one is in the future in the desired state of success, happiness, safety, delight

www.dbooks.org

or excitement. It then requires the team to work backwards and specify the steps that they took to achieve this state.

This technique is often used to focus the attention of developers on the principle experiences of the users. The team is asked to envision being the user of the finished software, and to specify its novelty and utility, that is, how it positively contributes to their practice and experiences. We ask them not to be constrained by practicality, technical feasibility or economics, but to set themselves in the ideal user experience where their life is transformed. If they can describe this, then they can map backwards to describe the various features of the software and how they might be developed.

SCAMPER

SCAMPER (from the book Thinkertoys by Michael Michalko) assumes that everything new is based on something already existing. New ideas, products, services etc. can be developed by taking something already existing and developing it into something new. It provides seven ways to do this: Substitute, Combine, Adapt, Magnify, Put to Other Uses, Eliminate (or minimise) and Rearrange (or reverse). It's often used with freeware software which generates random questions provoking the seven alteration modes.

This technique is used when an initial idea for a software product is in place, or later in the development process where there is a perceived need for a substantial review. The objective is to radically improve the current software concept, which is often expressed as a feature backlog, an interface sketch or some other kind of prototype, design models, or as part of an incrementally developed working system. A feature or screen can be substituted with something else that works better, combined with another feature, changed so that it performs a different function, made the central focus of the entire concept, or more or less eliminated and so on.

Six Serving Men

This technique is based on Rudyard Kipling's poem:

I keep six honest serving-men
(They taught me all I knew);
Their names are What and Why and When
And How and Where and Who...

It requires the participants to consider the questions raised by the six major interrogatory words in the English language.
This technique is used in our programmes at the point of initial specification, when there is some kind of concept, but before there is a real design or implementation effort. We use questions like these:

- *Who* will use the main features? e.g. manager, user, web-site visitor or customer?

- *Where* are the main features used?
- *What* major components and architectures will be involved?
- *When* should components be available?
- *Why* are these features needed?
- *How* will a feature be designed or programmed?

Six thinking hats

In this discussion technique developed by Edward de Bono, six different styles of thinking (factual, emotional, cautious, positive, creative, and controlling) are identified, and then associated with hats of different colours. Participants adopt different hats to accommodate kinds of contribution to the discussion.

Six thinking hats is a good project review technique and can be used periodically to give a quick status impression, perhaps at a stand up meeting. All the thinking styles should be covered, with someone who feels in tune with a thinking style leading a short discussion. It should provoke questions like

- *The White Hat (facts):* are we keeping our schedule and budget? Is our feature list complete, do we have the people and resources we need?
- *The Red Hat (emotions):* Do you feel good about the software product? Do you feel the team working well together? Is there the right balance of challenge and security, do you feel happy with your tasks
- *The Black Hat (caution):* What are the main risks, drawbacks, and points of critique?
- *The Yellow Hat (logical positive):* What are the main opportunities and the exciting challenges?
- *The Green Hat (ideas):* What else could we do that we're not currently doing? Can we work smarter? Should we change direction?
- *The Blue Hat (control):* How should the rest of the project be organized? Which steps comes first? How do we make sure we meet our goals?

The next two techniques are targeted at providing focus and overview in relation to the principle direction of the project, and the principle characteristics of the software product. They are typically used either

1. in the idea generation phase, when there are many good ideas (for example for features), but a need for a central governing concept or metaphor, or

www.dbooks.org

2. later in the project when design and implementation are more advanced, but the project is developing in several directions, or
3. in situations where the project or product needs to be effectively communicated to other stakeholders.

Vision box

The Vision Box is a technique for developing a marketing message that can drive the product development effort. If the product were to be marketed in supermarket, how would its box look? What product features, benefits, and attributes would be highlighted on the box to attract shoppers and encourage them to buy? The technique was first adopted in the software field by Jim Highsmith and leads to a low tech (cardboard and crayon) mock up.

Thus technique works well in some development contexts – for instance where a computer game is being developed where it will eventually compete for shoppers' attention with other games on the shelves of a specialised video game shop. A variant is to develop the product's home page.

Elevator test

The elevator test develops the team's ability to explain the innovative software product to someone within two minutes – the time it takes to ride an elevator. It comes from Geoffrey Moore's book *Crossing the Chasm.* It follows the form of a logically connected statement:

for *(target customer)*
who (*statement of the need or opportunity*)
the (*product name*) is a (*product category*)
that (*key benefit, compelling reason to buy*)
unlike (*primary competitive alternative*)
our product (*statement of primary differentiation*)

Work-style heuristic 7 – bring your toolbox

Software is seldom built without a variety of software tools and analysis and design techniques. The innovative software developer will clearly also need to work with suitable tools and techniques, but there is little research which helps to specify which tools and techniques will help. The choice will clearly be contingent not generic – that is it will be project and situation specific. There are very many developer tools of different kinds – but these are not normally targeted at creativity or innovation. There are even more creativity techniques to choose from – but these are not targeted at software development.

Creative information system developers will therefore need a *repertoire* - a range of tools and techniques that they are familiar with. It's impossible to learn everything, and this would also make the job of choosing impossible. A

repertoire of handful of each is probably specific. Software tools might cover (as suggested):

- support for escaping routine work
- sandbox tools
- knowledge tools
- collaboration tools: internal (project), external (customers and other stakeholders)
- visualization and overview support
- creativity technique support

Creativity techniques might cover some basic process functions such as idea generation, focus and overview, review and direction change. The next step will be *selection* – to match the tools and techniques in the repertoire to the development situation at hand. This may be formal and built into the project's structural conditions, but it's just as likely to be improvised as a response to the current demands, challenges and threats in the project. Here intuition based on experience will guide selection.

Sources and further reading

ADAMIDES, E. D. & KARACAPILIDIS, N. (2006) Information Technology Support for the Knowledge and Social Processes of Innovation Management. *Technovation,* 26**,** 50-59.

GREENE, S. L. (2002) Characteristics of applications that support creativity. *Communications of the ACM,* 45**,** 100-104.

HOHMAN, L. (2007) Innovation games: creating breakthrough products and services, Boston, Pearson.

LEITH, M. Compendium of idea generation methods. http://www.idea-sandbox.com/wiki/index.php/Compendium_of_Idea_Generation_Methods

MYCOTED, Creativity Techniques wiki. http://www.mycoted.com/

SHNEIDERMAN, B. (2000) Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI),* 7**,** 114-138.

SHNEIDERMAN, B. (2002) Creativity support tools. *Communications of the ACM,* 45**,** 116-120.

SHNEIDERMAN, B. (2007) Creativity Support Tools. *Communications of the ACM,* 50**,** 20-32.

www.dbooks.org

8. Know when you are (not) innovative: assessment and evaluation

Where developers focus on software innovation, it will be important for them to know whether they are achieving it or not. In this section we look at some formal and informal methods of assessment. The question under scrutiny is: how do you know if you are being innovative in development work?

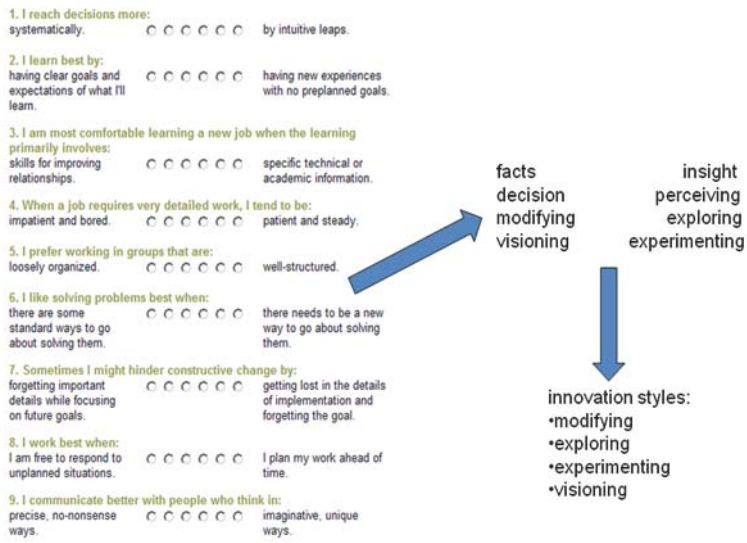
The literature contains examples of several types of evaluation; in particular of

- personal and group creativity
- software product assessment
- work environment inventory

The chapter will describe these assessment principles, but will also acknowledge that they do not go far enough: in particular they are of little help to development teams trying to assess their own performance in a concrete project situation. Therefore a short here-and-now, quick-and-dirty project evaluation instrument is presented.

Personal creativity: psychometric testing

Psychometric testing is quite widely used in both research and in practice; for example for assessing the suitability of job applicants. Though this is a rather popularized use, in many cases the tests are build on solid theory and large datasets. Researchers have used innovation style profiling (see the chapter on personal creativity) to evaluate the innovation styles of system developers.



A set of questions lead to numerical scores on asset of four scales, which is then used to plot the interviewee's innovation style. Though interesting research, it's hard to see how these techniques could be of widespread benefit to practising software developers, apart from helping them to understand and develop their personal creative potential.

Innovative software product assessment

A more relevant technique was developed by Lobert and Dologite. They tried to evaluate the innovation potential of proposals for software products, and developed a theoretically based framework for the purpose. The

EVALUATION▸ PERSPECTIVE CREATIVITY DIMENSION▾	PROJECT IDEA	ORGANIZATIONAL	TECHNICAL
NOVELTY	unique unusual astonishing	revolutionary astounding	astounding pioneering
RESOLUTION	appropriate	operable inexpensive useful	workable sensible
SYNTHESIS & ELABORATION	well-crafted clear elegant attractive organized meticulous	complex	complex

framework is base on three creativity dimensions (novelty; resolution; synthesis and elaboration) which are used to evaluate the project idea, its likely effect on the organization it is designed for, and its technical elements. A series of questions are developed to cover these nine parameters. There are no objective measures; instead the answers are evaluated by experts. Two obvious limitations with the research are that innovations are evaluated at the proposal (idea) stage and that the experts in the study were the professors of the students making the suggestions! However, in principle the idea is sound and could be applied.

Work environment assessment

An assessment technique which is suitable for software managers is the Work Environment Inventory (WEI). The inventory offers six parameters for assessment, with questions and assessment scores.

- freedom
- challenging work
- sufficient resources
- supervisory encouragement

www.dbooks.org

- work group support
- organizational encouragement

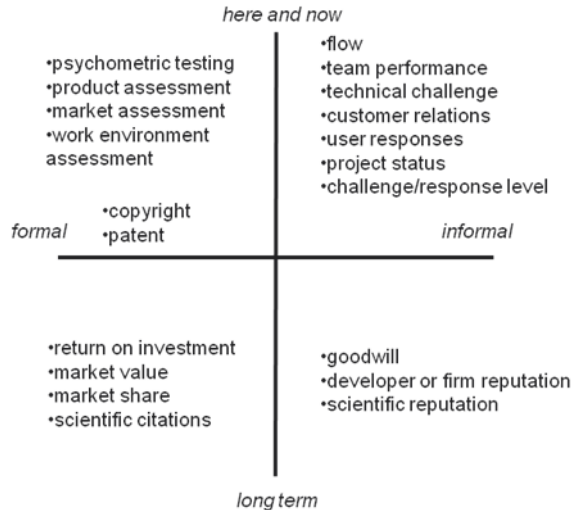
The end result of the study gives a picture of how supportive the work environment is for creativity and innovation.

Assessment overview

Though the techniques presented above offer some potential they fall far short of any comprehensive or usable way of measuring or describing software innovation. Among the problems that should be recognized are:

- they are eclectic examples
- there is no systematic framework
- there are many sides of software innovation to be evaluated
- many conventional economic assessment techniques are long term measures, whereas innovation often involves long delays before returns can be evaluated
- software innovation is highly situation dependent, and generic techniques are not necessarily applicable.

In the framework on the right some assessment measures are proposed and organized around two axes. The axes distinguish measures which can be applied *here and now* from those which will be appropriate in the *future*, and *formal* measures (based on methodical data collection and interpretation) from *informal* types of impression gathering. The formal measures that can be applied her and now include the research-oriented techniques described above. A very conventional way of measuring innovation is by counting patents, but this often relies on having a formally described invention in a relatively late stage of development. In addition, many, perhaps most inventions will never be exploited of diffused, and thus cannot really ne



described as innovations. Software patents are easier to obtain in the US than in Europe, and there are disagreements over what constitutes a patentable software innovation. The conventional economic measures, such as return on investment (RoI) and market share, are dependent upon having the innovation in production (or distribution in the case of software licenses) and impact delays in innovation mean that these measures can be rather long term. It may be some years after the launch of a product before its economic effects can be properly evaluated. Longer term informal measures are related to goodwill from customers and reputation. Of more interest at this stage of the evolution of software innovation theory are informal here and now measures. These can allow developers to develop a feel for their project. They include:

- flow – developers can often assess their own contribution by observing their psychological state during their work. Flow indicates productivity and concentration, whereas thrashing is less desirable
- team performance – innovative teams have good role distribution, dialogue, some degree of non-destructive conflict, the ability to work with divergent thinking
- technical challenge – innovative projects are challenged by the technical and programming demands – but not to the point where the project's survival is threatened
- user responses – innovative development projects have constructive and challenging relationships with their customers and users
- project status – an innovative project is often behind schedule, because there are many difficulties, but not to the degree that it will be fatally compromised
- challenge/response level – developers are challenged – but in ways to which they are able to find suitable reposes. They are stimulated and excited – but not under unmanageable stress for long periods of time.

Here-and-now quick-and-dirty evaluation instrument

The following instrument is not based on empirical research; however it does have a theoretical basis since it relies on the material in this book, which is derived from the available research in the area. The questionnaire takes ten minutes to complete and is divided into eight areas which are given equal weight. In each case you should consider the questions (each of which reflects a specific theoretical sub-area), then give your project a score out of seven, where seven means you are doing well. Other members of the team can also use ten minutes, and you can compare results.

www.dbooks.org

Keep your head up

Do you understand the latest technical trends and developments in the field you are working on?

Do you know the rival products that other software companies are working on?

Do you understand the emerging technology potential?

Have you assessed what infrastructure your product requires, and will it in place when the product is released?

Have you investigated the potential market for your product?

Is your timing right?

score [ ]

Grow your knowledge community

Are you in contact with leaders in the field: other development groups, researchers, universities, lead users?

Do you partner to improve your expertise base?

Can you import necessary expertise for the project when you need it?

Do you get valuable external feedback from outside the project? From outside the company

Are you a member of relevant online and offline knowledge communities?

score [ ]

Target your product's innovation profile

Can you articulate the added value (utility) for the user?

Have you determined how your product is new and original?

Do you understand your user community – their work and leisure habits?

Do you understand how your users' lives will change when they use your product?

Do you work with the product's innovation profile?

score [ ]

Shape your own process

Do you have an innovation process strategy and is it suitable for the task?

Do you have the correct balance of market-led and technology-led strategies?

Are there techniques and practices which stimulate the creativity of the team, and does it allow space for creativity and innovation?

Can you improvise your way out of the difficulties?

Do you continually and explicitly adapt your process to the current needs of the project?

score [ ]

Develop your personal creativity

Are you, personally, learning fast?

Does your role in the project suit and stimulate you?

Can you bring your expertise and experience to bear on the software challenge?

Are you challenged and stimulated by the tasks you have without feeling chronic stress?

Are you often in flow?

score [ ]

Be a super-team-worker

Are you aware of the factors that hinder the team's innovation and do you work to improve them?

Does the team recognise sub-optimal teamwork and work to improve it?

Does the team work towards an evolving shared vision and know where it is going?

Does the team work at effective communication (dialogue)?

Does the team understand how to accommodate divergent thinking?

Do the team members communicate their experience and expertise and learn from each other?

score [ ]

Bring your toolbox

Does the project have a repertoire of formal or informal creativity techniques and use them where appropriate to help you to move forward?

Do you have the right tool support to maximise creative progress and minimise drudge work?

score [ ]

www.dbooks.org

Know when you are (not) innovative

Does the team recognise when it is not moving forward, discuss it openly and do something differently as a result?

score [ ]

Now identify the areas where you scored poorly and consider appropriate responses. Don't simply dismiss them as inappropriate for your circumstances. There is a theoretical background behind the questions which means that it is more likely that you are letting your unconscious pre-dispositions dominate too much, than that the question is irrelevant.

Work-style heuristic 8 – understand when you are (not) innovative

Understanding the innovation status of a software project or the innovation potential of a team of developers is a complex problem. Some researchers have developed rather limited techniques for assessing aspects of software innovation, but these are eclectic and unsystematic. Nevertheless the alert developer and software manager do have informal here and now ways for monitoring their work. Innovative software projects have many challenges: technical challenges, process challenges, knowledge challenges, relationship challenges communication challenges and economic challenges. This means that they are inevitably difficult, at least in some periods. However, long unproductive periods, sustained conflict or work stress, insurmountable technical challenges and poor communication with customers and users (and many other things) threaten innovation. Developers need to understand when they no longer innovative, and react by changing things. The here-and-now quick-and-dirty evaluation instrument can help in this process.

Sources and further reading

COUGER, J. D. (1996) Press: Measurement of the Climate for Creativity in IS Organizations. *Creativity and Innovation Management,* **5,** 273-279.

DAVILA, T., EPSTEIN, M. J. & SHELTON, R. (2006) *Making Innovation Work: How to Manage It, Measure It, and Profit from It,* Upper Saddle River, Wharton School Publishing.

HIGGINS, L. (1996) A Comparison of Scales for Assessing Personal Creativity in IS. *International Conference on System Sciences.* Hawaii, IEEE.

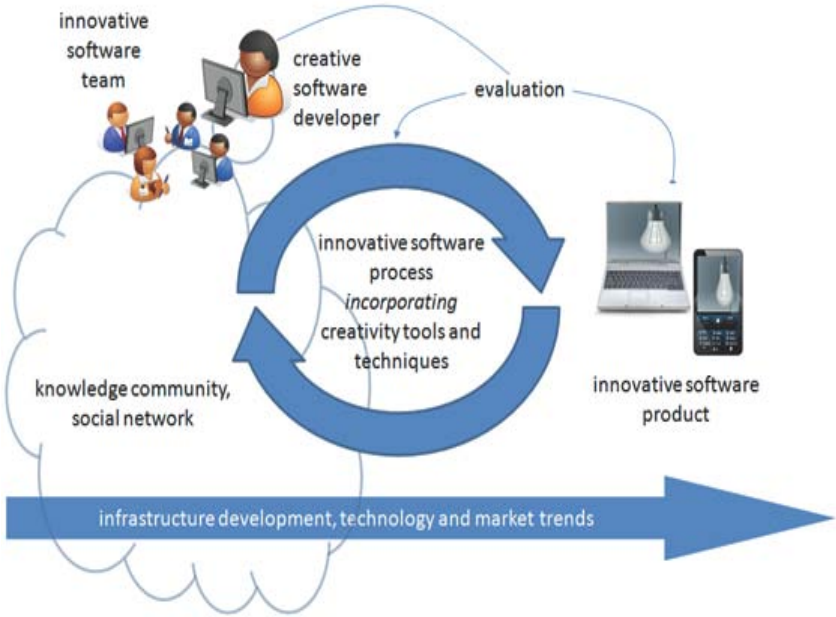
LOBERT, B. M. & DOLOGITE, D. G. (1994) Measuring creativity of information system ideas: an exploratory investigation. *System Sciences, 1994. Vol.IV: Information Systems: Collaboration Technology Organizational Systems and Technology, Proceedings of the Twenty-Seventh Hawaii International Conference* Hawaii.

MILLER, W. C., COUGER, J. D. & HIGGINS, L. F. (1993) Comparing innovation styles profile of IS personnel to other occupations.

9. Software innovation: eight work-style heuristics for innovative system developers

Software innovation

In the various chapters of this book we have examined the innovative software process and product, the creative software developer, the innovative software team, creativity techniques and software tools, innovation knowledge community and network, technology trajectories and innovation windows, and innovation assessment.



Software innovation is conducted in relation to social developments, in particular *infrastructure development and market and technology trends*. Technologies trends can be understood as trajectories – moving in directions which can be analyzed. Mechanical technologies become digital and converge. Markets for software and user demand also develop and display trends. Software innovation is always dependent on social and technical infrastructures – if these are not sufficiently developed the innovation is likely to fail. If we put these ideas together we can develop the concept of an innovation window – the right time to build a particular application.

Scientific, business and engineering innovation is not conducted by isolated groups or individuals but in *innovation networks and knowledge communities.* The innovative software developer's connections to equally innovative colleagues in related fields are very important for building and sharing knowledge and ideas. Software development has its own success story here

www.dbooks.org

– the open source community. Open source development has been hailed as a new 'private collective' innovation model.

The *innovative software product*, we concluded, displays both novelty and utility. Invention (the novelty part) is not really enough; innovation also contains the idea that the invention should be exploited and diffused - that it should reach its target community of users and be adopted. The consequence of adoption is social change, not usually in the sense of profound alterations to society (though can also happen), but in the sense of communities of users who alter their work or leisure practice through using the software. Software innovations can be incremental (small practice changes in small niche user communities), or radical and discontinuous. Altshuller's theories of technical system hierarchies and levels of innovation can help us to understand the difference. The range of innovation is very wide – software technologies are woven into the very fabric of our lives in developed societies. The six utility forms describe the width of types of innovative software products

- computing infrastructural
- technology enabling
- user service
- business change enabling
- interaction communication
- entertainment

The book develops the idea of an innovation profile for a software product, involving the explicit consideration of the product's novelty, utility, user community, social change, market, technical innovation and infrastructure dependence.

The *innovative software process* can be either linear – starting with a flash of inspiration leading to a more or less complete new product specification (the light bulb model), or iterative. This reflects a contemporary evolution in software development methods in which agile methods have come to complement traditional methods. In the linear process form, focus is thrown on creative requirements analysis; creative interactions with users can be a way of developing an innovative information system which does more than reflect an existing work process. However examples of actual systems development methods focused on innovation are rare – one such is Aaen's ESSENCE, in the iterative, agile tradition. Agile methods are a global process innovation (they have only recently emerged) but this does not automatically mean that the result of an agile method will be an innovative software product. Agile methods are targeted at efficiency, not innovation. However there is some reason to expect that reduction in bureaucracy and the rational analysis load could be conducive to innovation, and that rapid response to change is essential. Most software development methods are focused on user needs, but another way of thinking of software innovation is as market-

led (what consumers will buy) or technology-led (what developers can come up with). Whatever the approach, it will be wrong to assume that an external imposed formalized method could (by itself) lead to an innovative product; these situations will always require human skills of improvisation and bricolage. The book develops six innovation process strategies:

- creative requirements analysis
- the designed process framework
- low tech prototyping
- user-driven software innovation
- community development
- the research prototype.

Software process innovation can take place at a global or local level. Agile methods are an example of global innovation, whereas the many software process improvement initiatives can lead to local innovation.

The book develops eight perspectives on *the creative software developer* – understanding creativity as:

- the developer's mental process: recognising and exploiting discovery points
- a set of personal development competences concerned with both solving problems and recognising opportunities
- a style of thinking associated with different strengths in individual's development personalities
- meta-thinking: recognising predispositions and tendencies in one's own (and others') thinking and coming beyond them
- whole-brain thinking: beyond rationality
- a relationship between the individual developer and communities of people and ideas (domain, field)
- a state of mind: the way the developer's mind is disposed when being creative (flow)
- a universal mental skill to be enhanced

Developers work in teams and we characterized the *innovative software team* as a super-functional team. We recognised various creativity barriers and styles of group dysfunction as hindering the creative team-process. Many factors can play a positive role. An understanding of team roles enables the building of innovative teams. The quality of the team's interaction is important, the way it accommodates divergent thinking and the way it recognises productive social patterns in its work. Team learning and expertise integration help with the development and maintenance of overview, vision and common purpose. Finally, an innovative team has a

www.dbooks.org

heads-up attitude; it understands the commercial and scientific developments, as well as the social trends, around it.

Software tool support is an important aspect innovative development and the book develops a software support toolbox, designed around:

- support for escaping routine work
- sandbox tools
- knowledge tools
- collaboration tools
- visualization and overview support
- creativity technique support.

Creativity techniques can be an element of the creative software development process. There are many of these, and they normally require a degree of adaptation to be meaningful in system development work. The book develops a starting repertoire including:

- brainstorming
- backward mapping
- SCAMPER
- six Serving Men
- six thinking hats
- vision box
- elevator test.

Finally, software innovators need to develop an instinct for when their work is going well. There are many *assessment and evaluation* techniques, but few which are directly targeted at software innovation. A sensible strategy is to focus on intuitive here and now assessments:

- flow
- team performance
- technical challenge
- user response
- project status
- challenge/response level.

Eight work-style heuristics for innovative system developers

The book is not a method or process guide. Instead it works with the idea of work-style heuristics – a broad characteristic which typifies a developer's way of working. Here are the eight heuristics developed in the book.

Keep your head up

Systems and developers are engineers and business professionals with many skills at their fingertips: analysis techniques, methods, programming languages, project management techniques and many others. Much of a development project involves the application of those skills in a very close communication with colleagues and (sometimes) users. The focus of this work is inevitable inward. Programmers have a very intense relationship with their code editors and spend many hours with their heads buried in their screen. Software innovation, however, involves the placement of a specialised product in a community of users at a particular time. If the technology is immature, the infrastructure insufficiently developed or the market undeveloped, then the product will fail. If the technology is well-established, the infrastructure more than adequate, and/or the market screaming for the product, then the chances are your competitors are already well ahead of you and you will never catch up. No-one is better placed to understand these things than you are, if you study them daily.

Grow your knowledge community

Innovation is dependent on knowledge, and knowledge is a social process. Software innovators are usually not lone geniuses, despite many of the myths that circulate. Most technical advances involve many contributors in larger or smaller roles, even if one person is later awarded the credit by history. Companies, university researchers, research institutes, lead users, specialised bedroom programmers, venture capitalists and policy makers all have their role to play. In the development world there are important roles for open innovators, in the various open source communities and in collaboration with companies. Software innovators recognise the importance of the communities they are members of, know their place in those communities, and work actively to foster them.

Target your product's innovation profile

An innovative software product has certain characteristics. It is novel in relation to its projected user community and it has a particular utility for them. It may be an incremental innovation or a more radical one, and have more or less significance for a small, or a very large social group. However the innovator needs to understand and target the innovation potential of the product. This can be described as its innovation profile. Of course some innovations have a life of their own and find uses that were never intended, but, in general a development team needs to understand how the product will fit into the work or entertainment practice of their users, and how it will develop that practice.

Shape your own process

Developing an innovative software product is likely to be turbulent and challenging, and, although systems developers have a long history of

www.dbooks.org

developing and implementing (and sometimes even using) design processes, it is far from clear what an optimal software innovation process is. Most likely there are as many successful development processes as there are innovative teams. Common to all process will be the ability to improvise the way out of difficulties, dead ends and seemingly insurmountable challenges. Alertness is the key, and the developers needs to take charge of their own process, adapt to the needs of the particular project, and adapt it again when it's no longer working. And again, and again.....

Develop your personal creativity

Everyone is creative in their own way, and every developer has their own creativity style and personality. These can be quite different, and highly situation-dependent. A ground-breaking contribution in one situation can simply be a handicap in another. However software innovators are aware of their own creative profiles: what they can and cannot contribute. They actively develop them and understand how to incorporate them into the work of the team.

Be a super-team-worker

An innovative software team is a super-functional team. It works particularly well in communication and problem-solving, can manage its own process and achieve overview and coherence, and exploits the competences of each of its members to achieve a synergistic result. Innovative developers understand their role in such a team and contribute not only to their personal performance, but to the joint performance of the team – which is, in the end result, more important.

Bring your toolbox

Software development is dependent on software tools and development techniques, and innovative software development is no exception. An innovative software developer has a repertoire of tools and techniques, and applies them selectively – the right tool or technique for the correct development situation.

Know when you are (not) innovative

Not everything goes smoothly all the time. Projects function well in periods, then develop problems and recover. Sometimes they become disaster areas. Innovative projects are particularly challenging and therefore liable to encounter non-creative periods. Sometimes these will be temporary and unimportant, but often developers will need to be capable of recognizing that there is a problem and reacting appropriately. Often the appropriate reaction will be some form of process adjustment to get things working again.

Keep your head up

Grow your knowledge community

Target your product's innovation profile

Shape your own process

Develop your personal creativity

Be a super-team-worker

Bring your toolbox

Know when you are (not) innovative

Now it's your turn…….

Jeremy Rose

Aalborg, December 2010

www.dbooks.org

Comprehensive list of reading and sources

ADAMIDES, E. D. & KARACAPILIDIS, N. (2006) Information Technology Support for the Knowledge and Social Processes of Innovation Management. *Technovation,* 26**,** 50-59.

ALTSHULLER, G. S. (1988) *Creativity as an Exact Science,* New York, USA, Gordon & Breach.

AMABILE, T. M., CONTI, R., COON, H., LAZENBY, J. & HERRON, M. (1996) Assessing the Work Environment for Creativity. *The Academy of Management Journal,* 39**,** 1154-1184.

AMOROSO, D. L. & COUGER, J. D. (1995) Developing Information Systems with Creativity Techniques: An Exploratory Study. *Proceedings of the 28th Annual Hawaii International Conference on System Sciences.*

BANSLER, J. & HAVN, E. (2004) Improvisation in information systems development. IN KAPLAN, B. (Ed.) *Information Systems Research.* Boston, Springer.

BEARDON, C., EHN, P. & MALMBORG, L. (2002) Design of Technology-Augmented Creative Environments. *Proceedings of Computer Support for Collaborative Learning 2002***,** 7-11.

BELBIN, M. (1981) *Management Teams,* London, Heinemann.

BROOKS, F. P. (1975) The Mythical Man Month.

CANDY, L. & EDMONDS, E. (2000) Creativity enhancement with emerging technologies. *Communications of the ACM,* 43**,** 63-65.

CHESBROUGH, H. (2003) *Open Innovation: The New Imperative for Creating and Profiting from Technology,* Boston, MA, Harvard Business School Publishing.

COOPER, R. B. (2000) Information Technology Development Creativity: A Case Study of Attempted Radical Change. *MIS Quarterly,* 24**,** 245-276.

COPLIEN, J. O. & HARRISON, N. B. (2005) *Organizational Patterns of Agile Software Development*, Pearson Prentice Hall.

COUGER, J. D. (1990) Ensuring Creative Approaches in Information System Design. *Managerial and Decision Economics,* 11**,** 281-295.

COUGER, J. D. (1996) Press: Measurement of the Climate for Creativity in IS Organizations. *Creativity and Innovation Management,* 5**,** 273-279.

COUGER, J. D. (1997) Creativity/Innovation in Information Systems Organizations. *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on,* 3.

COUGER, J. D. (1997) Results of a trans-discipline research structure for study of creativity/innovation in IS. *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on,* 3.

CSIKSZENTMIHALYI, M. (1997) *Creativity: flow and the psychology of discovery and invention*, Harper Perennial.

DAVILA, T., EPSTEIN, M. J. & SHELTON, R. (2006) *Making Innovation Work: How to Manage It, Measure It, and Profit from It,* Upper Saddle River, Wharton School Publishing.

DE BONO, E. (1971) *The Use of Lateral Thinking: A Textbook of Creativity*, Penguin.

DEARDEN, A. & HOWARD, S. (1998) Capturing user requirements and priorities for innovative interactive systems. *Proceedings of the Australasian Computer Human Interaction Conference*, 160–167.

DENNING, P. J. (2004) The social life of innovation. *Communications of the ACM,* 47, 15-19.

DUGGAN, E. W. (2003) Generating Systems Requirements With Facilitated Group Techniques. *Human-Computer Interaction,* 18, 373-394.

DUGGAN, E. W. & THACHENKARY, C. S. (2004) Integrating nominal group technique and joint application development for improved systems requirements determination. *Information & Management,* 41, 399-411.

ELAM, J. J. & MEAD, M. (1987) Designing for creativity: considerations for DSS development. *Information and Management,* 13, 215-222.

ELAM, J. J. & MEAD, M. (1990) Can software influence creativity. *Information Systems Research,* 1, 1-22.

EVANS, M., WALLACE, D., CHESHIRE, D. & SENER, B. (2005) An evaluation of haptic feedback modelling during industrial design practice. *Design Studies,* 26, 487-508.

FAGERBERG, J. (2005) Innovation: a guide to the literature. IN FAGERBERG, J., MOWERY, C. & NELSON, R. R. (Eds.) *The Oxford Handbook of Innovation* Oxford, Oxford University Press.

FAGERBERG, J., MOWERY, C. & NELSON, R. R. (Eds.) (2005) *The Oxford Handbook of Innovation,* Oxford, Oxford University Press.

FELLERS, J. W. & BOSTROM, R. P. (1993) Application of group support systems to promote creativity ininformation systems organizations. *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on,* 4.

FLOYD, I. R., JONES, M. C., RATHI, D. & TWIDALE, M. B. (2007) Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software. *HICSS 2007: 40th Annual Hawaii International Conference on System Sciences.* Hawaii.

www.dbooks.org

FRANKE, N. & HIPPEL, E. (2003) Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. *Research Policy,* 32**,** 1199-1215.

FRASER, J. (2005) Inspired innovation: how Corel is drawing upon employees' ideas for user focused innovation. *Proceedings of the 2005 conference on Designing for User eXperience*.

GARFIELD, M. J., TAYLOR, N. J., DENNIS, A. R. & SATZINGER, J. W. (2001) Research Report: Modifying Paradigms--Individual Differences, Creativity Techniques, and Exposure to Ideas in Group Idea Generation. *Information Systems Research,* 12**,** 322-333.

GASSMANN, O. & ENKEL, E. (2004) Towards a theory of open innovation, three core process archetypes. *R&D Management Conference.* Sesimbra.

GLASS, R. L. & DEMARCO, T. (2006) Software Creativity 2.0.

GREENE, S. L. (2002) Characteristics of applications that support creativity. *Communications of the ACM,* 45**,** 100-104.

HACKLIN, F., RAURICH, V. & MARXT, C. (2004) How incremental innovation becomes disruptive: the case of technology convergence. *Engineering Management Conference.* IEEE International

HELO, P. (2003) Technology trajectories in mobile telecommunications. *International Journal of Mobile Communications,* 1**,** 233-246.

HIGGINS, L. (1996) A Comparison of Scales for Assessing Personal Creativity in IS. *International Conference on System Sciences.* Hawaii, IEEE.

HOHMAN, L. (2007) *Innovation games: creating breakthrough products and services,* Boston, Pearson.

HOLMQUIST, L. E. (2004) User-driven innovation in the future applications lab. *CHI '04 extended abstracts on Human factors in computing systems.* Vienna, Austria, ACM.

JORDAN, P. W., KELLER, K. S., TUCKER, R. W. & VOGEL, D. (1989) Software storming: combining rapid prototyping and knowledgeengineering. *Computer,* 22**,** 39-48.

KANTER, R. M. (2006) Innovation: the classic traps. *Harvard Business Review,* 84**,** 72-83.

KELLY, T. (2001) Prototyping is the Shorthand of Innovation. *Design Management Journal,* 12**,** 35-42.

KOSKI, H. A. (1999) The Installed Base Effect: Some Empirical Evidence From The Microcomputer Market. *Economics of Innovation and New Technology,* 8**,** 273-310.

LEINBACH, T. R. & BRUNN, S. D. (2002) National innovation systems, firm strategy, and enabling mobile communications: the case of Nokia. *Tijdschrift voor Economische en Sociale Geografie,* 93**,** 489-508.

LIND, J. (2007) Boeing's Global Enterprise Technology Process. *IEEE Engineering Management Review,* 35**,** 38-52.

LOBERT, B. M. & DOLOGITE, D. G. (1994) Measuring creativity of information system ideas: an exploratory investigation. *System Sciences, 1994. Vol.IV: Information Systems: Collaboration Technology Organizational Systems and Technology, Proceedings of the Twenty-Seventh Hawaii International Conference* Hawaii.

LYYTINEN, K. & ROSE, G. M. (2003) Disruptive information system innovation: the case of internet computing. *Information Systems Journal,* 13**,** 301-330.

LYYTINEN, K. & ROSE, G. M. (2006) Information system development agility as organizational learning. *European Journal of Information Systems,* 15**,** 183-199.

MAIDEN, N. & MANNING, S.

MAIDEN, N., MANNING, S., ROBERTSON, S. & GREENWOOD, J. (2004) Integrating creativity workshops into structured requirements processes. *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques.* Cambridge, MA, USA, ACM.

MAIDEN, N. & ROBERTSON, S. (2005) Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on***,** 105-116.

MARAKAS, G. M. & ELAM, J. J. (1997) Creativity Enhancement in Problem Solving: Through Software or Process? *Management Science,* 43**,** 1136-1146.

MARTINICH, L. (2002) Managing innovations, standards and organizational capabilities. *Engineering Management Conference, 2002. IEMC'02. 2002 IEEE International,* 1.

MCCONNELL, S. (1998) The Power of Process. *Computer,* 31**,** 100-102.

MCLEAN, E. R. & SMITS, S. J. (1993) The I/S leader as `innovator'. *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference.*

MICH, L., ANESI, C. & BERRY, D. M. (2005) Applying a pragmatics-based creativity-fostering technique to requirements elicitation. *Requirements Engineering,* 10**,** 262-275.

www.dbooks.org

MILLER, W. C., COUGER, J. D. & HIGGINS, L. F. (1993) Comparing innovation styles profile of IS personnel to other occupations.

NONAKA, I. (1991) The Knowledge-Creating Company. *Harvard Business Review,* 69.

PINK, D. H. (2005) A Whole New Mind. *Riverhead, New York, NY.*

POWELL, W. & GRODAL, S. (2005) Networks of Innovators. IN FAGERBERG, J. (Ed.) *The Oxford Handbook of Innovation.* New York, Oxford.

ROBERTS, E. B. (1988) Managing invention and innovation. *Research Technology Management,* 31**,** 11-27.

ROSE, J. (2010) *Software Innovation: eight work-style heuristics for creative software developers,* Aalborg, Software Innovation, Dept. of Computer Science, Aalborg University.

SAMPLER, J. L. & GALLETTA, D. F. (1991) Individual and organizational changes necessary for the application of creativity techniques in the development of information systems. *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences.* Hawaii.

SAMUELSON, P. (2006) IBM's Pragmatic Embrace of Open Source. *Communications of the ACM,* 49**,** 21-5.

SHNEIDERMAN, B. (2000) Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI),* 7**,** 114-138.

SHNEIDERMAN, B. (2002) Creativity support tools. *Communications of the ACM,* 45**,** 116-120.

SHNEIDERMAN, B. (2007) Creativity Support Tools. *Communications of the ACM,* 50**,** 20-32.

SNOW, T. A. & COUGER, J. D. (1991) Creativity improvement intervention in a system development workunit. *System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference*

STEPHENSON, N. (1999) *In the Beginning...was the Command Line*, Harper Perennial.

STREITZ, N. A., GEIßLER, J., HOLMER, T., MÜLLER-TOMFELDE, C., REISCHL, W., REXROTH, P., SEITZ, P. & STEINMETZ, R. (1999) i-LAND: an interactive landscape for creativity and innovation. *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit***,** 120-127.

TAPSCOTT, D. & A.D, W. (2006) *Wikinomics: How Mass Collaboration Changes Everything,* New York, Portfolio Hardcover.

THOMAS, J. C., LEE, A. & DANIS, C. (2002) Enhancing creative design via software tools. *Communications of the ACM,* 45**,** 112-115.

TIWANA, A. & MCLEAN, E. R. (2003) Expertise Integration and Creativity in Information Systems Development. *Journal of Management Information Systems,* 22**,** 13-43.

TUOMI, I. (2001) Internet, Innovation, and Open Source: Actors in the Network. *First Monday,* 6.

TUOMI, I. (2003) Networks of Innovation. *Oxford Press*.

VON HIPPEL, E. & KATZ, R. (2002) Shifting Innovation to Users via Toolkits. *Management Science,* 48**,** 821-33.

VON HIPPEL, E. & VON KROGH, G. (2003) Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science,* 14**,** 209-223.

VON KROGH, G., SPAETH, S. & LAKHANI, K. R. (2003) Community, joining, and specialization in open source software innovation: a case study. *Research Policy,* 32**,** 1217-1241.

WALKER, G. H., STANTON, N. A. & YOUNG, M. S. (2001) Where is computing driving cars? A technology trajectory of vehicle design. *International Journal of Human Computer Interaction,* 13**,** 203-229.

WALLAS, G. (1926) *The art of thought*, J. Cape.

WALZ, D. B. & WYNECOOP, J. (1994) Creativity and Software Design - is formal training helping or hurting? *Systems, Man, and Cybernetics, 1994. 'Humans, Information and Technology'., 1994 IEEE International Conference*

WARR, A. & O'NEILL, E. (2005) Understanding design as a social creative process. *Proceedings of the 5th conference on Creativity & cognition***,** 118-127.

AAEN, I. (2008) Essence: Facilitating Agile Innovation. *XP2008.* Limerick, Ireland.

AAEN, I. (2008) Essence: facilitating software innovation. *European Journal of Information Systems,* 17**,** 543-553.

www.dbooks.org