Thirteen Knights and the Seven-headed Dragon: an Interdisciplinary Software Engineering Framework

Nikolay Mehandjiev and Paul Layzell
Department of Computation
UMIST, UK
n.mehandjiev@umist.ac.uk

Pearl Brereton
Dept. of Computer Science
Keele University, UK
o.p.brereton@cs.keele.ac.uk

Grace Lewis
SEI, CMU
USA
glewis@sei.cmu.edu

Mike Mannion
Glasgow Caledonian University
UK
M.A.G.Mannion@gcal.ac.uk

François Coallier
École de Technologie Supérieure (ETS)
Canada
fcoallier@ele.etsmtl.ca

Abstract

This paper summarizes the findings of the STEP2002 workshop on Interdisciplinary Software Engineering, which took place on the 6th and 7th of October 2002 in Montreal, Canada. The workshop considered the future of software engineering as an interdisciplinary activity by identifying ideas, models and techniques which are already used in cognate disciplines, and are deemed applicable to software engineering, with the aim of breaking down rigid barriers between disciplines, representations and processes.

1. The Workshop

The STEP2002 workshop on Interdisciplinary Software Engineering considered the future of software engineering as an interdisciplinary activity by identifying ideas, models and techniques which are already used in cognate disciplines, and are deemed applicable to software engineering.

The need for the workshop arose out of recognition that the role of software has become a critical element in all aspects of modern life supporting wealth creation and deployment in products and processes designed to improve the quality of life. The demands placed on the software engineering community, such as productivity, flexibility, robustness and quality, have increased dramatically. Whilst incremental approaches to improving the software engineering process are delivering significant benefits in the short and medium term, the software engineering community has realized that it should, in the best spirit of scientific enquiry, also take a broader and possibly more radical view about future software engineering techniques, processes and tools [1].

2. The Findings

The workshop identified seven main groups of issues related to contemporary software engineering practice and research:

1) Information explosion;
2) Growing demands in terms of functionality, quality, and time to market;
3) Need for adaptable software systems;
4) Value for money considerations;
5) Procurement decisions;
6) Risk identification and management; and
7) System integration.

Thirteen cognate disciplines were then identified which could contribute ideas, models or techniques to address the seven groups of issues. These disciplines were broadly classified in five groups: disciplines dealing with deterministic systems, those dealing with systems where humans are an important part of the system; those concerned with business systems, society-focused disciplines and finally, a group of sciences including mathematics and computer science.

A framework for classifying the ideas, models and techniques which were believed influential in changing Software Engineering into a holistic discipline was then created as a table. The rows in the table contained the seven main groups of issues, whilst the columns contained the disciplines and their groupings. Each idea, model or a technique was then positioned in cells laying at the intersection of each discipline and the group of issues which they address. To complete the framework,

IEEE COMPUTER SOCIETY

each of them was then classified into one of five levels of diffusion, depending on the current software engineering practices. *Level 1* signified that a model or technique is a common practice which is used in over 60% of industrial projects, such as configuration management. *Level 5*, at the other extreme, contained ideas or techniques which were deemed to require further research on the way they can be applied to software engineering, such as multi-model integration. The middle three levels covered ideas, models or techniques which were perhaps covered by education programs, used in some organizations as best practice tools or waiting for standards to make widespread application possible, such as component reuse. The diffusion levels are summarized in Table 1:

Table 1. Diffusion levels in the framework

Level	*State of practice in software engineering*
1	Common practice, widely used in industry in over 60% of the projects, *e.g.* configuration management;
2	Covered by Software Engineering University programs, but perhaps not applied widely, for example UML and Z;
3	Best practice in some organizations, perhaps covered by frameworks like CMM, SWEBOK, RUP, *e.g.* peer review and extreme programming (XP);
4	Waiting on standards being developed and accepted, *e.g.* component reuse;
5	Needs further research on the way it applies to software engineering, *e.g.* multi-model integration.

The paper presents a summary view of the framework and description of the issues there. Because of space and time constraints, the level of detail in discussing each issue varies. For each issue an estimate of the diffusion level as defined above is specified in round brackets, for example "(3) Specialisms" specifies that, in our opinion, the level of diffusion of the concepts and practice of recognized specialists from civil engineering to software engineering has been estimated at Diffusion Level 3.

Each of the sections below deals with one group of disciplines from the framework. Only the relevant portion of the table is presented and then entries in that portion are discussed. The results from this first workshop do not claim to provide complete coverage of all cells in the framework, for example the lack of entry under the issue of Risk in the Hardware Engineering column does not mean that no useful techniques for dealing with risk can be learned from hardware engineering,.

3. Deterministic Systems

This section covers the disciplines dealing with deterministic systems, such as civil engineering and hardware engineering.

Table 2. Disciplines dealing with deterministic systems

	Deterministic systems	
Issues	*Civil Engineering*	*Hardware Engineering*
Information Explosion	(3) Specialisms	
Growing demands, *i.e.*-functionality,-quality, time to market	(2) Design patterns	(4) Reuse based on standards and catalogues
Adaptable	(3) Realistic expectations	(2) Over-design
Value for money		(4) Markets for components
Procurement Decisions		(5) Component specifications
Risk		
Integration		(4) Testbeds and Standards

3.1. Civil Engineering

Traditional engineering disciplines handle the excess of information which has to be processed with the introduction of *specialisms*: well defined roles taken by experts in a particular activity. The expertise is qualified via certifications within a particular field, for example bridge inspectors, and code analyzers in the Rational Unified Process. The concept of specialisms is currently best practice in some organizations but lack of wider practical use positions it at Diffusion Level 3, together with the lack of coverage in University teaching.

In contrast, design patterns are widely taught at University level as a technique to handle growing demands on software productivity by increasing reuse of previous design knowledge. Civil engineering and architecture are the recognized sources of the idea of design patterns. The use of design patterns is not yet a common practice in the software engineering industry, it has therefore been allocated Diffusion Level 2.

Civil engineering has constrained the problem of adaptability requirements by cultivating realistic expectations of how much a particular building can be adapted. For example, the public does not expect a suburban house to be convertible to an airport terminal. Explicit recognition of the need for realistic expectations exists within the requirements management element of

CMM, for example, but is not widely taught nor used in the software engineering practice. It has therefore been allocated Diffusion Level 3.

3.2. Hardware Engineering

The engineering of computer hardware has responded to growing demands for time to market by introducing extensive standards-based systems of component re-use supported by extensive catalogues. Different levels of granularity are covered, from the level of individual resistors all the way up to rack-mounted server modules. Whilst the application of this approach to software engineering is clear in terms of component-based systems engineering, further practical applications are hindered by the lack of industry-dominating standards for component specification and interoperability. The level of diffusion is therefore 4. The bottom three techniques in the hardware engineering column are also main enablers of component-based engineering:

1) value-for-money achieved by extensive component marketplaces (Diffusion Level 4);
2) effective procurement decisions being dependent on component specifications, that is clear specifications of functionality plus any other relevant attributes which effect suitability of a component (Diffusion Level 5); and
3) integration achieved by mature standards and compliance testbeds (Diffusion Level 4).

Designing for adaptability in the realm of hardware engineering is covered by the concept of over-design, where additional features are embedded in the design to be used by future extensions. Similar ideas are covered in the software design research, best practice and education levels, but not yet widely used in the industrial projects, arguably because of time-to-market and short-term efficiency considerations (Diffusion Level 2).

4. Human-centered Systems

This section focuses on disciplines which are concerned with systems within which humans play an important role.

Table 3. Human-centered disciplines

	Humans "in the loop"		
Issues	*Systems Engineering*	*Production Engineering*	*Project Management*
Information Explosion	(2) Information structuring / views of information / levels of abstraction	(3) Product Information Management Systems	(3) Role of domain expertise
Growing demands	(2) Components and architectures	(3) CADCAM	(3) Time-boxing.
Adaptable	(2) Evolvable architectures	(5) Mass customization and explicit variability	(3) Goal-driven teams.
Value for money			(3) Financial appraisal (ROI)
Procurement Decisions			
Risk	(2) Risk assessment		(3) Proactive monitoring of critical factors
Integration	(4) "Ensemble" view		

4.1. Systems Engineering

Systems engineering is a discipline that advocates an integrated approach to the construction of entire systems designed to perform tasks in what is expected to be most efficient possible manner, with each component of system designed to function as part of a single entity. The two elements that systems engineering combines in order to accomplish this are (1) a component-based system design and architecture that facilitates integrability and evolvability while guaranteeing the overall efficiency of the system and (2) a constant acknowledgment of the system as a single entity - the software, the hardware, the organization, and the humans.

Component-based design [7] in systems engineering is more than specifying components as portions of functionality integrated by a common framework. The systems engineering approach to component-based design and integration uses *an "ensemble" view*. As a component of the system is being defined and specified, it is always considered part of an ensemble - a set of components and the interactions between them. The constant question is "how does this component fit in with the rest of the components in the system?" Fitness in this context therefore means more than component integration; it also considers the effect that the component has on the rest of the system and the effect that the system has on the rest of the component. Issues such as the effect on system qualities, the alignment with business processes, and user interaction are discussed as a part of system design. We estimate that the *"ensemble" view* approach to the *issue of integration* within software engineering is at the Diffusion Level 4.

Some concepts such as enterprise architecture [6], model views [9], and architecture views [8] are starting to have wide usage in the software engineering community as a form of representing and storing information about systems, thus handling the *Information Explosion* issue. They are not a common practice yet (hence Diffusion Level 2), but the deeper problem is the software engineers using these concepts still focus solely on the software system itself, sometimes ignoring the interaction between the software system and the rest of the system. These useful concepts need to be taken to a level higher - from the software system level to the system of systems level (enterprise level). The *Zachman Framework for Enterprise Architecture* is an example of a framework that uses contextual models, business models, system models, technology models, and other detailed representations to produce a complete enterprise architecture [6].

Evolvability is another important aspect in systems engineering that is considered during system design and implementation - the ease with which a system or component can be modified to take advantage of new technologies, to improve system qualities, or to achieve greater efficiencies. Evolvability addresses the issue of creating *adaptable systems* and can only be achieved with *an evolvable architecture* that can itself change and also allow the system to change and advance. An evolvable architecture has the ability to isolate and insulate the system from the effects of change. Elements of an evolvable architecture are specialized layering, modularity, well-defined interfaces, standard interfaces, and certain common mechanisms - all good practices that apply to software engineering as well, but are not a common practice yet (Diffusion Level 2).

Finally, *risk management* is an integral part of any systems engineering approach because the failure of the system can be contributed by any part of the system: the hardware, the software, the organization, or the humans. Again this is at Diffusion Level 2 since it is not in a widespread use in industrial Software Engineering.

Software engineering frameworks such as CMMI [10] and RUP [11] have included disciplines or extensions to address the reality that software practices and systems practices are inseparable.

4.2. Production Engineering

Production Engineering (PE) systems support the management of a portfolio of products, processes and services from initial concept, through design, launch, production and use to final disposal. They co-ordinate products, project and process information throughout new product introduction, production, service and retirement among the various stakeholders. Computer Aided Production Engineering systems (CAPE) model the factory, production line or work cell layout to simulate production processes and generate efficient operations plans and train operators.

Software Engineering can learn from this discipline because many manufactured products are software intensive and the management of the product portfolio necessitates the management of a corresponding software product portfolio. Within Software Engineering, software product line engineering has emerged as a discipline that takes a structured approach to the development of adaptable and reusable software. Despite a greater understanding of software product line engineering methods, deployment is not common because (i) rapidly changing markets and short-term commercial profit objectives often impede an organization's view of appropriate long-term strategy and corresponding structure (ii) return in investment models are not well defined (iii) tool support for such methods are still in their infancy and many organizations are reluctant to embrace methods without technological support (iv) there are few reports about the organizational characteristics required to successfully carry out software product line engineering.

Specific areas of knowledge transfer from production engineering to software engineering include how to manage information explosion, demand growth, and adaptability to changing products. In production engineering, to *manage information explosion* typically a *product information management information system* (MIS) is maintained, which may be physically distributed, but has a single logical index to all the documents containing product, project and process information. PE applications use workflow and authorization rules to give orderly access to this vault's information. The various processes of new product introduction, production, service and retirement use a single source of product information. In software engineering this type of approach is most likely to be carried out only in large manufacturing organizations with a history of production engineering where the software being developed is done in-house and is to be encapsulated in a manufactured product e.g. an aircraft. Universities rarely teach software engineering-in-the-large, hence Diffusion Level 3.

In production engineering, the vast majority of expenditure is on mainstream Computer Aided Design and Computer Aided Manufacturing (*CADCAM*) systems that will automate as many of the specification-to-manufacture processes as possible and permit organizations to respond effectively and efficiently to demand growth. For example electronics are an increasing proportion of many products and many electronic design automation systems allow engineers to model the requirements of a system, and then generate manufacturing instructions to realize the design in a variety of technologies, such as Field Programmable Gate Arrays (FPGA). In software engineering the automatic generation of designs from requirements is not available. The use of design tools with automatic code generators is available but not widespread. Accordingly universities are reluctant to abandon programming teaching to undergraduate computing students. They may do this within the next 15 years. Overall this can be classified at Diffusion Level 3.

In production engineering, the drive towards *mass customization* is a long-term goal for many organisations. Currently integrated production systems (e.g. cellular manufacturing systems, flexible manufacturing systems, computer integrated manufacturing systems) are best applied in medium-variety, medium volume markets. The key to success is adaptability and information integrity. It is not clear that there is any consistency across organizations about logical product model and electronic data exchange formats, or how variability is modelled. However the ISO STEP standard ISO 10303 [2], Standard for the Exchange of Product Model Data, which is a standard for representing and exchanging digital product information, is addressing this issue. In software engineering, at the design level, in practice whilst several variability mechanisms can be identified [3] and certain guidelines for using them provided, most information system methods and accompanying design notations do not permit *variability* to be modelled hampering product line design and development. The application of this technique to Software Engineering requires further research efforts, hence Diffusion Level 5.

4.3. Project Management

According to the Project Management Institute (PMI) Project Management Body of Knowledge, or PMBOK [13], a project is a temporary endeavor undertaken to create a unique product or service. Project Management is thus the application of knowledge, skills, tools, and techniques to project activities in order to meet or exceed stakeholder needs and expectations from a project [13]. Project management and engineering have been intertwined for at least five thousand years, ever since humans started to cooperate for large endeavors.

Project Management integrates many disciplines. The key ones can be summarized as Project Planning, Project Scope, Resources and Time Management, Project Risks Management and Project Execution and Control. Project management is an integral part of the software engineering body of knowledge (SWEBOK) [14]. Problems encountered in software intensive projects have not been because of gaps in the body of knowledge, but rather because of gaps in the practice and implementation of this knowledge. We estimate that the diffusion level from Project Management to Software Engineering is roughly Diffusion Level 3 across the range of techniques and methods.

While the PMBOK does provide generic project management knowledge and practices, it is well recognized that *domain-specific expertise is required by project managers* to be truly effective in handling *the information explosion*. This does not only means knowledge and practices specific to software engineering, which can be found for instance in the SWEBOK[14], but also knowledge specific to the type of product and service that will be developed. Why is it so? This is because each type of product and services has its own challenges. Only a project manager with the proper knowledge and experience of these particular products and services will be able to comprehend all relevant information, and then *assess and manage its associated risks*. This in turn will mean a more efficient project management, and thus greater ability not only to deliver on time and on budget, but also in shorter time and budget.

The traditional application of project management in software engineering was aimed at the delivery of a set functionality within a given schedule and budget. The project life-cycle associated with such an approach is the traditional waterfall model. A more modern approach is to recognize that, for many software projects, *goal-driven approach is more appropriate to handle the adaptability and productivity requirements.* Indeed, the knowledge of which features/requirements must be delivered is itself part of the challenge. These approaches are thus built around an iterative process were the product is built in increments. In such approaches, the decision of when to deliver to the customer becomes thus a management decision. Either of the three project constraints (Resources, Time, and Feature) can thus be truly managed by the project manager. Techniques such a *time boxing* can thus be applied to projects were schedule is the key constraints - features and resources being thus less constrained.

It is common practice for software projects to go through, at project initiation, an elaborate selection process, or *Financial project appraisal* as a means to address the *Value for Money* requirement. Factors such as the *Return on Investment (ROI)* are assessed and taken into consideration. Decision is then taken based on the ROI and other consideration whether to fund the project. When the outcome of the project is a software product that is sold on the market, the predicted ROI can be validated easily. When it is an enterprise application, validating the ROI can be less evident, especially in companies that do not have good business process metrics and fined grained financial data that is readily exploitable.

One of the many challenges in any software intensive projects is the *assessment, mitigation and management of risks*. This can be done in either an ad hoc fashion, or through a more *pro-active monitoring* one were generic risks factors proper to the nature of the projects are assessed at the onset of a particular project.

5. Business Systems

Management, Economics and Finance can clearly contribute valuable techniques and ideas to the software engineering of the future. We have grouped them in the *Business Systems* group shown below in Table 4.

Table 4: Disciplines dealing with business systems

	Business systems		
Issues	*Management*	*Economics*	*Finance*
Information Explosion			(4) Analyse real-time data
Growing demands			
Adaptable	(5) Services rather than products		
Value for money		(3) RoI	(5) Trading crowd appr.
Procurement Decisions	(5) Negotiation	(5) Transactio n cost economics	(5) Option pricing models
Risk	(2) Risk Management		(5) Portfolio management
Integration			

5.1. Management

In the domain of human organizations, providing *services rather than products* is an accepted way to achieve the necessary level of *adaptability*.

In the domain of software, the idea of providing (and therefore acquiring) software functionality as a service over the web is not a new one. Application service providers have offered supply-led services to customers for some years. Such services are negotiated in advance and are essentially software procurement by outsourcing. Two obvious advances in the way in which services can be provided are:

1) through demand-led 'just-in-time' selection and contract negotiation;
2) through the composition (or configuration) of new services from pre-existing (sub) services.

Extending the concept of application service provision in these ways leads to a vision of 'software service engineering' - the dynamic engineering of web-based services as needed - configured, executed and paid for on demand [4].

Such an approach to delivering software functionality would provide a high level of adaptability. New sub-services could be incorporated as and when they become available enabling service users to dynamically upgrade the software (service) they use as their business evolves. This idea is still at early research stages, hence Diffusion Level 5.

Negotiation is an important mechanism for making *procurement decisions.* Indeed, in today's highly dynamic business environments, where the forces of software supply and demand are continuously changing, fixing the price or any other aspect of a software product or service may not be possible or indeed even desirable. In such environments, and particularly when software is procured as a service (rather than as a product) automated negotiation has a significant role to play in the dynamic selection of the software most suitable to the needs of the user.

Clearly software in uniquely placed to benefit from automated negotiation since both the negotiation and delivery of the functionality can be performed electronically and hence automatically [5]. The idea of automatic software negotiation is still at research stage, hence Diffusion Level 5.

Risk management is now widely taught on software engineering courses (and covered in the major text books) and is also practiced to a significant extent in industry, especially in terms of risk assessment. However, there is rarely follow-up in terms of actively managing the risk factors, hence the Diffusion Level 2.

5.2. Economics

There is some evidence that methods for *estimating and monitoring Return on Investment (RoI)* are filtering into software engineering teaching although there is a stronger focus on this topic in the more management oriented courses. Cost-benefit analysis is important in software engineering and RoI calculations can clearly play an important role in this activity. This is at Diffusion Level 3, although the activity of evaluating what is the best approach to the a particular software development project is at Diffusion Level 5.

Transaction cost economics addresses the problem of identifying the true costs of transactions across organisational boundaries. It recognises that there is often a failure to appreciate the complex costs associated with such activities as contract negotiation, tendering, maintaining relationships and opportunism. These, often hidden, costs will become increasingly important as we move towards a service-oriented approach to software deployment, acquisition and composition. Taking these costs into account is unfortunately still at the stage of research and therefore at Diffusion Level 5.

5.3. Finance

The techniques behind the ability of the financial sector to analyze vast volume of real-time data can benefit software engineering project both in terms of project management information and in terms of managing the information about product design and testing. Some of these techniques are well researched and their further application is now waiting on the appropriate standards to be developed (Diffusion Level 4).

Other techniques from the financial sector can also be used to address the following issues of software engineering: value for money can be achieved by "trading crowd" approaches; procurement decisions will benefit from option pricing models, and risk can be managed using portfolio management techniques, traditional within the financial sector. All these are still at research stage and hence at Diffusion Level 5.

6. Society-focused disciplines

Law and the open source movement are both focused on the norms, practices and processes within a society.

Table 5. Disciplines focused on society

	Society	
Issues	*Law*	*Open Source Movement*
Information Explosion		
Growing demands	(1) Outsourcing contracts	(1) Peer review by experts (2) Configuration control
Adaptable	(5) Automated contracting	
Value for money		
Procurement Decisions	(1) Project contracts (5) Automated contracting	(4) Virtual teams
Risk		
Integration	(4) Design by contract	

6.1. Law

It is not common to see courses from the school of law, such as contracts, in a software engineering curriculum. This does not mean that software engineering cannot use some skills in this area.

It is a reality that most organizations can no longer afford to build large software systems (or even small software systems) from scratch. Some pieces of the system may be acquired and some subcontracted. This means that software engineers do require *outsourcing and project contract skills* to deal with subcontractors, vendors, and outsourcing. These are perhaps not ideal but widely used in the industry, hence Diffusion Level 1.

The concept of a *contract* can be taken further as a good software engineering practice. A contract is a legally enforceable promise, normally exchanged as part of a bargain between private parties. This is exactly what a theory called *design by contract* promotes: a software system is viewed as a set of communicating components whose interaction is based on precisely defined specifications of the mutual obligations - contracts [12]. This type of design seeks to specify the relationship between components as precisely as possible - an essential element of seamless *integration*.

Changing software configurations "on the fly" is the ideal of highly *adaptable* software, included in the vision of "autonomic computing". *Automated contracting*, including evaluation of proposed contracts by suppliers, negotiation of terms of the contracts and monitoring of the contract fulfillment is a necessary condition for this, but is still at the stage of research investigation, that is at Diffusion Level 5.

6.2. Open Source Movement

The social and technical practices of the open source movement have proven their viability in addressing some of the issues of contemporary software development, and have consequently been widely accepted by the software engineering practitioners. For example peer review by experts is now widely practiced means of ensuring quality under rapid development practices (Diffusion Level 1), and tool-based configuration control is widely taught at University level but only starting to infiltrate industrial practices (Diffusion Level 2).

The social processes of forming virtual teams where members are distributed throughout the world and have perhaps never met in person seems to work well within the open source community. Research papers have investigated its applicability to software engineering teams in global companies and "round-the-clock" development scenarios, but further progress towards practice is hindered by tool and standards support (Diffusion Level 4).

7. Sciences

The group of sciences includes mathematics, computer science and cognitive psychology.

Table 6. Science disciplines

	Sciences		
Issues	Mathematics and modeling	Computer Science	Cognitive Psychology
Information Explosion	(5) Chaos theory	(1) Semanticall y rich	

		descriptions	
Growing demands	(3) Tools for assessing complexity	(4) High-level Languages (5) Mixed initiative development	(5) Cognit. design of language & representations
Adaptable			
Value for money			
Procurement Decisions			
Risk	(5) Variable selection & uncertainty		
Integration	(5) Multi-model integration		

7.1. Mathematics and modeling

Chaos theory aims to create mathematical models of complex systems. With the *information explosion* the complexity of both software and software development process reaches levels where the models of chaos theory are applicable to software engineering [15], although these instruments are still only applied in the research labs (Diffusion Level 5). For example, we can take a set of component design metrics and test if the set fits into a fractal theory.

Tools for *accessing complexity* which use theory of computation and algorithm complexity are also relevant for managing growing requirements sets. We can for example investigate the complexity metrics of software as a function of growing requirements and their relationships.

Mathematical modeling can be used to estimate *risk* using appropriate selection of *variables* important for assessing risk. For example we can use the mathematical techniques used to select input parameters for weather prediction and modeling. We can also study how weather modeling analyses the *uncertainty* of those parameters. This can then lead to predicting the effect of parameter variability on the system risk, using established simulation techniques.

Techniques used to *integrate multiple mathematical models of complex systems*, for example in weather forecasts, can be used to integrate and manage the links between diverse models of software requirements, structure and behavior.

Despite the long history of mathematical foundations of software engineering, most of the techniques and methods discussed here are still at the level of research investigation, that is Diffusion Level 5.

7.2. Computer Science

From all disciplines described here, computer Science is perhaps the “closest relative” to Software Engineering. We therefore imagine that there are numerous techniques and concepts which originate in Computer Science and can help address one the seven issues plaguing contemporary software engineering. The ones selected here serve merely to illustrate the available variety.

High-level languages, for example, have traditionally been conceived as a mechanism to reduce programming effort and thus allow programmers to better cope with *growing productivity demands*. They are often based on domain abstractions (e.g. 4th generation languages are based on office processing abstractions such as reports and forms). More recently, some high-level languages based on pattern composition have been developed to enable partial automation of design. Both will be very applicable in traditional software engineering, but the lack of dominating standard is holding back their growth, hence they are Diffusion Level 4.

Semantically rich descriptions are closely related to high-level languages. Semantic languages such as DAML-OIL are currently developed by the CS community. The idea is that a high level of semantic in a description language can enable reasoning over the information encoded in such a description, and extracting information and meaning which is not originally specified there in an explicit nor formal manner.

Mixed initiative development can be used together with semantically rich descriptions to facilitate the work of software engineering under the growing demands on productivity. Using AI techniques for interacting with software developers, be they professional programmers or end users, can offload some the development effort to the machine.

7.3. Cognitive Psychology

Many computer scientists and software engineers are increasingly turning to cognitive psychology for help with the design of interfaces, programming languages and conceptual representations. The premise is that languages and representations in use do not seem to be optimized according to the cognitive processing of software developers. The idea is to use theories of cognition to optimize the mapping between mental models and cognitive processes of developer-experts to the languages and the representations they use in their work. This work is at Diffusion Level 5, and has a good potential to improve the ability of software engineering to cope with growing demands in terms of productivity.

8. Conclusions

This paper contains the output of a two-day workshop where participants brainstormed techniques, methods and concepts from different disciplines which can address seven issues of contemporary software engineering. No claim for comprehensiveness of the coverage can possibly be made, neither in terms of the thirteen disciplines nor in terms of the seven main issues. We were, however, purposefully targeting a wide variety of disciplines and issues to increase the coverage of our deliberations.

We were deeply satisfied with creating the framework as a starting point for systematic exploration of interdisciplinary influences in software engineering. We naturally hope to see the framework expanded and used as a guide for more extensive research in the area, but we will also be happy to have merely seeded the idea of such work in someone's mind.

10. References

[1] Bennett K. H.,.Layzell P. J., Budgen D., Brereton O. P., Macaulay L., Munro M., "Service-Based Software: The Future for Flexible Software", *IEEE APSEC2000, The Asia-Pacific Software Engineering Conference*, 5-8 December 2000, Singapore, IEEE CS Press, 2000.

[2] http://www.iso.org

[3] Clements, P., Northrop, L., Software Product Line: Practices and Patterns, Addison-Wesley, 2002.

[4] K. H. Bennett, N. E. Gold, M. Munro, J. Xu, P.J. Layzell, D. Budgen, O.P. Brereton and N. Mehandjiev, Prototype Implementations of an Architectural Model for Service-Based Flexible Software, Thirty-Fifth Hawaii International Conference on System Sciences, Ed. Ralph H. Sprague, Jr., IEEE Computer Society, CA, Jan. 2002

[5] A Elfatatry & P Layzell, Negotiating In Service Oriented Environments, accepted for publication in Comms. of the ACM, January 2003.

[6] Zachman, J. A Framework for Information Systems Architecture." IBM Systems Journal, vol. 26, no. 3, 1987. IBM Publication G321-5298.

[7] Wallnau, K.; Hissam, S.; and Seacord, R.. Building Systems from Commercial Components. New York, NY: Addison-Wesley, 2001..

[8] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; and Stafford J. Documenting Software Architectures: Views and Beyond (SEI Series in Software Engineering). Addison-Wesley, 2002.

[9] Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software 12(6): 42-50 (1995)

[10] CMMI Product Team. CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1) - Continuous Representation (CMU/SEI-2002-TR-011 ESC-TR-2002-011). Software Engineering Institute. March 2002.

[11] Rational Software. "Rational Unified Process for Systems Engineering RUP SE1.1", A Rational Software White Paper TP 165A, 5/02, available for download from: <http://www.rational.com/products/whitepapers/wprupsedeployment.jsp>

[12] Meyer, B. Applying "Design by Contract, in Computer (IEEE), vol. 25, no. 10, October 1992, pages 40-51.

[13] Anonymous 1998, IEEE Guide, Adoption of PMI Standard, A Guide to the Project Management Body of Knowledge, IEEE Std. 1490-1998, IEEE

[14] Abran, A. Et Moore, J. (Ex. Eds), Bourque, P. Et Dupuis, R. (Eds), 2001, Guide to the Software Engineering Body of Knowledge - Trial Version (Version 1.0), IEEE Computer Society Press

[15] Kokol, P.; Podgorec, V.; Cardoso, A. and Dion, F.; Assessing the state of the software process development using the chaos theory, ACM SIGSOFT Software Engineering Notes, 25 (3), May 2000, ACM Press New York, NY, USA , Pages: 41 – 43 ISSN:0163-5948