Critical and Creative Mathematical Thinking with Practical Problem Solving Skills - A *New Old* Challenge

Eleni Berki[1] and Juri Valtanen[2]

[1]Department of Computer Sciences, University of Tampere,
Kanslerinrinne 1, Pinni B, Tampere 33014 Finland
eleni.berki@cs.uta.fi

[2]Department of Education, University of Tampere,
Ratapihankatu 55 (Atalpa), Tampere 33014 Finland
juri.valtanen@uta.fi

[2]VALIO OY
Vihiojantie 3, 33100 Tampere Finland

Abstract. The suitability and applicability of formal (mathematical) methods for problem solving that targets to the construction of software-based systems has long been a controversial discourse in Software Engineering academic education. Additionally, experiences from using formal methods in real software development projects refer to very strong advantages and, at the same time, very strong disadvantages. In this paper, adopting a critical point of view, the authors examine the suitability and applicability of mathematical thinking in the traditional higher education context and in the challenges of modern software development. Discussing and broadly classifying a number of controversial opinions, the authors draw comparisons between classical educational perspectives and current higher education curricula and industrial training programmes. Analysing strengths and weaknesses, the authors provide insights and suggestions for improving mathematics learning. For further understanding of the generic skills and holistic knowledge required when applying formal methods to problem solving, the authors outline a multidisciplinary curricula framework as a thinking tool for educators. The latter offers a broad integrating basis of subjects and could serve for cases in software development where specialisation and generalisation are needed.

Keywords: Software Development Methods (SDM); Formal Methods (FM); Critical Thinking; Creativity; Higher Education (HE) Academic Curricula; Problem-Based Learning (PBL) Skills

1 Introduction

The efficacy and suitability of the development methods used to design and produce software-based systems, e-services and information products do not seem to be comparable in effectiveness to the scientific methods that were used throughout the

history of knowledge, science, and mathematics in particular [1]. Far from the pedagogical frameworks and social reality -supported, though, by Higher Education (HE) curricula and expensive industrial training programmes- the fragmented use of Software Development (SD) methods that are represented by mathematical notations brought about correct -to some extent- solutions; but with limited understanding and justification on doing so. This repeatable phenomenon over the last thirty, at least, years resulted in (i) further limited application of mathematics in SD, and in (ii) the view that Formal Methods (FM) are independent of any problem solving activities [1].

1.1 Background

In the last forty years, SD, mainly being a problem solving process, employed a plethora of problem solving methods (Fig. 1), which are also called systems development methods. These, broadly classified, are: (i) *soft methods* for ill-defined situations - or soft problems- with no definite answers and (ii) *hard methods* for well-defined situations -or hard problems- that require definite answers [2]. Two more categories exist: *hybrid* (e.g. Multiview) and *specialised* (e.g. for expert systems building) methods for contingency approaches and specialised situations, but with specialised and limited use. Figure 1, from [2], depicts a chronological evolution (1965-2000) of SD methods.

Soft methods, such as Soft System Method (SSM) and Effective Technical and Human Implementation for Computer-based Systems (ETHICS), focus on human involvement in most of the steps of the system's development lifecycle. Hard methods such as structured (e.g. SSADM, STRADIS), object-oriented (O-O), formal (e.g. VDM, Z), and semi-formal (e.g. JSD) comprise many techniques and, sometimes, automated techniques in software tools. The latter are applied to consistently and precisely model a problem situation: state the requirements, define the needs and, above all formulate, specify and solve the domain-related problems in a specification language that can be understood and trusted.

The subject of Methodology -the scientific study of all methods- has revealed, through classical and more recent comparative studies, plenty of advantages and disadvantages that make each method suitable for different problems and application domains (see e.g. [3, 4]). Formal methods, for instance, comprise mathematical expressions that are very concise since a great deal of meaning is concentrated in a relatively small number of symbols. Moreover, mathematical forms are said to be unambiguous and clear because of their independence from any cultural context and because they are self-contained. The latter were and continue being considered as strengths and strong advantages when describing a very complex system, i.e. safety-critical, where detailed, trustworthy and testable specifications of the system components are needed.

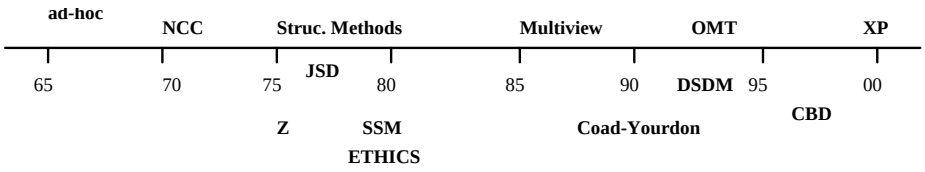

Fig. 1: Thirty five (35) years of Software Development Methods

On the other hand, independence from cultural context and non-ambiguity could be considered as disadvantages when a system, organisation or product contains detailed human activity. Therein, a less structured, O-O or semi-formal method might prove to be more applicable for the analysis, design and specification needs. Furthermore, most formal methods could be regarded as inadequate and unsuitable, considering that co-ordination activities for holistic communication modelling [5] do not exist during FM use in SD lifecycle stages. FM rather jump to specification and design, before any feasibility study or analysis of system requirements and human needs [1, 2] are carried out. Maintenance phase is also limited supported by FM. The degree of people's participation and especially the involvement of the end-user, who still remains the great unknown X in software development, is either limited or non-existent in the FM paradigm.

1.2 Research questions, approaches and paper structure

The need for integration and standardisation in SD methods, techniques, tools and regional procurements on one hand, and the increased global competition and internationalisation of SD on the other, demand formally approved and generally understood high quality specification notations [3]. More than twenty years ago method and tool vendors, software markets, and software knowledge-based economies in Europe welcomed and supported *Euromethod*, a SD method(ology) standardisation project aiming at achieving integration and standardisation [6]. The - yet unfinished- project, firstly and foremost selected and adopted structured methods' notations and techniques (largely diagrammatic) from all European countries' SD standards. The *only* formal specification technique adopted had been Petri-Nets from the French SD standard Merise - also a structured method. One could only question if the lack of tool support and, consequently, tool vendors were the only reasons to exclude FM from the so much promising European SD standard.

Nowadays the new trends of Open Source Software Development (OSSD) and Agile Software Development (ASD) require to utilise formal method(ology) features and formal quality assurance. However, in ASD the question sometimes is “agile quality or depth of reasoning?” [4]. Moreover, having to face the dilemma of applicability versus suitability with respect to all stakeholders needs [4] one might then question how successful will formal method(ology) be to undertake the analysis and design challenges of modern SD. For these reasons, the research questions handled in this paper are: (i) what made formal method(ology) thinking so popular in the past? (ii) what eliminated it? and (iii) how could the use of formal methods be enhanced and increased in educational programmes and in future software development?

For answering the previous questions, this paper examines the historical evolution and use of mathematics, based on a wide literature review, previous evaluations of formal methods, and personal experiences of the authors from many years action research in HE and in industry. The paper attempts a consistent analysis of the strengths and limitations of formal methodology, utilising research approaches from the critical research paradigm [7] as follows: *a) Historical research for* studying past developments in science-mediated learning and their implications on the scientific progress focusing on issues of mathematics learning and application, and their implications on learners empowerment. *b) Discourse analysis* considering studies of discourse practices in order to illuminate individual, social and cultural behaviours of mathematics learners in scientific communities. *c) Action research f*or analysing and evaluating practices in SD, aimed at the improvement of available knowledge and skills in HE.

From the critical point of view -in terms of ontology- this work first explores the reality of Mathematics learning in HE and mathematical knowledge application in SD, as unfolded through different opinions. Emphasis, therefore, is also on the powerful forces that shape the needs of SD and HE - though this is left out of the analysis scope of this paper. Second, the Mathematics learners' reality in HE is constructed through discourse communication that is comprised of (concrete or abstract) representations shaped by particular signifying systems, social contexts and power relations that take place in SD and HE communities. Both first and second issues are reflected in our data collection, which comprises many specialists' opinions analysed in the next sections.

Thus, our main research objective is to deconstruct this reality and provide an improved understanding of signifying systems, socio-technical and cultural contexts, power relations, and the real-world contexts in which the latter are embedded. The authors' role as future researchers will continue being active and motivating towards addressing the above issues by providing transparency and links to informed, committed and progressive action with a new HE curriculum, which is just outlined at section 4 of this paper. In terms of epistemology, it can be stated that the nature of meta-knowledge explored here emphasises that mathematical expertise offered in SD is an extension of knowledge and skills offered in the HE curricula. The curriculum meta-knowledge in HE institutes is bound to the interested groups (learners, educators and SD stakeholders, e.g. local and national industry) involved, and also to socio-cultural and socio-technical constructs and constraints.

2 Past, Present and Future: Praising and Questioning the Use of Formal Methods

Naturally, by using mathematics in software design for requirements formal representation, there is little scope for misunderstanding between different groups of people (SD stakeholders) because a correct formulation of requirements specification can easily be examined for its correctness and errors can quickly, or at least timely, be identified. In [8] Dijkstra in 1981 stated that by its very nature responsible systems design and development in general - in short *programming reliably* - must be an

activity of undeniably mathematical nature. However, he goes on to comment that by and large, *"current mathematical style has been determined by fashion, and current mathematical notation by accident"*. The following sections present a collection of opinions that either reject or support these claims.

2.1 Supportive and Non-supportive Statements on Formal Methods

In Software Engineering (SE) community, it has often been supported that mathematical expressions have the advantage of being precise because they do not need contextual information. All information needed is encapsulated in relevant formulae and what is not needed is omitted [9, 10]. One, though, might question: are context, culture and language insignificant factors in software development? Should they not be taken into consideration at all, while solving problems and providing solutions, and software-based solutions in particular? Classical and recent research results show that organisational and national culture as well as language and background knowledge among systems stakeholders are critical success factors for software quality management, and for reliable and trustworthy software products (see e.g. [11, 12]).

Apart from being -naturally among mathematicians- a shared belief that mathematical forms are independent of spoken (natural) language, it is arguably a fact that mathematical notations can, successfully, be used to express problem specification and achieve unambiguous solutions in software systems development. Software engineers [13, 14], however, hold a somewhat different opinion believing that there is more than specification and implementation to consider in software systems development and construction stages. Shortly the analysis, design, specification, implementation and maintenance of systems that require a high degree of reliability and are user-orientated form a *problem solving* and requirements communication process [15, 16, 17]. Pharmaceutical, defence, medical or safety critical systems, for instance, are exceptionally demanding systems in their analysis, and in the design of a software-based solution. The challenge is in considering both the number of end-users with their needs and activities and their safety-critical requirements that must be realised and supported by suitable and reliable software. Herein, no formal method alone, and furthermore no single SD method, neither hard nor soft (see section 1.1), is suitable to cover the system development and stakeholders' needs. Notwithstanding, there is a need to employ a problem solving approach with subsequent holistic communication modelling (see [5]).

The need -and must- for correct and reliable system specifications that are mapped to correct, and at the same time understandable, implementations and user interfaces has been a problem that software engineers constantly face. Within the context of software development methods, it is believed that the application of formal specification methods is a rigorous way to develop quality information systems [1, 18] that bring about generally -but not holistically- acceptable solutions to human problems [5]. For instance, one of the aims of formal specification methods is to provide the ability to prove the correctness of representations. Correctness (see e.g. [8]) as a software quality property is -claimed to be- achievable only by the use of formal methods [1, 2, 8]. Correctness though, as a semantic definition, has many

facets and is also related to the semantics, pragmatics and semiotics of the software developer's social and professional environment [1, 2, 5]. For instance, the concept of correctness -and not only- in SD can be related to ethical and professional correctness, a soft issue in SD. Formal methods, though, do not support particular values and norms of groups of individuals or people; FM do not support human and especially end-users' involvement in SD [19]. Therefore, only other than formal methods could be utilised to achieve correctness in the previous context. These could be soft methods, especially ETHICS method [19], which emphasises the holistic nature of a system's design by supporting representative and consensus user participation as an ethical and professional meta-requirement of the SD process.

On the other hand, these particular systemic and human needs -correctness, reliability, understandability- and the ways that everyone understands them, led many software developers and engineers to adopt an idiosyncratic approach to problem solving in SD and to the implementation of system specifications (see also [5, 19]). Adequacy of needs, and most importantly reliability, have had many facets and they are certainly related to the correct specification of stakeholders' requirements, in order to have an effective and functional software system. In the recent past of SD, problem specification often obeyed, without detailed analysis or design rationale, to rigid mathematical formulae, led probably by intuition, but not by critical thinking or creativity [1, 2]. Yet, there is no revolutionary or evolutionary change! That being a recent and ongoing thinking and SD tradition with formal methods, supported by and taught in academic curricula and realised in industrial training programmes, it is hard to be optimistic for progressive change and adoption of more participative SD approaches.

2.2 Practical Problems with Known Formal Methods in Software Development

Formal methods such as those mentioned in table 1, next, express systems properties, algorithms, data, events and functions in a rigorous mathematical way, which is not ambiguous. This family of methodological notations may be *model-oriented* that also include diagrams, or *property-oriented* that are mostly based on text reasoning descriptions.

Certain forms of deductive reasoning are often used for quality assurance, although a formal testing procedure is utilised in a limited fashion. Moreover, there are formal methods developed for very specialised situations. For instance, with the exception of X-Machines and possibly FSMs from table 1, all 'formal' approaches to development concentrate on the coverage of the modelling needs of particular application domains. As can be seen from table 1 the limited availability of modelling tools, which by no means are widely used in industry or academy, also makes FM use even more difficult, especially when it comes to issues of formal testing procedures, test cases generation from specifications, computability, reverse engineering and re-engineering [1].

Table 1: Some formal methods with tool support used in software development

Formal Methods	Tool Support
Z-Method	Yes
Vienna Development Method (VDM)	No
B-Method	Yes
Statecharts	Yes
Finite State Machines	Limited
Calculus of Communicating Systems (CCS)	Yes
Timed Calculus of Communicating Systems (TCCS)	Yes
Communicating Sequential Processes (CSP)	Limited
X-Machines	Limited
Object Z / Z++	Limited
Temporal Logic	Limited
Petri Nets	Limited

Some formal methods such as process algebras e.g. CSP and CCS that are also used to model concurrency, and their versions (e.g. TCCS) involve diagrams more than others such as Z and VDM, for instance. The latter, used to model sequential systems, have been less expressive than their extensions. Z++ and Object Z, for instance, do have better expressability and applicability for domain specifics. Notations that describe a system's behaviour expressively, are Petri-Nets (used in many cases to also model concurrent processes), Statecharts and Finite State Machines; these are model-based formalisms, also with adequate textual visualisations of abstraction [1].

Diagrammatic notations are used in SD because they provide a means of abstraction and, in certain cases, allow the behaviour of the system to be clarified through visualisation. This is the main reason why many of the formal methods have been extended in order to conform to the requirements of cognition considerations and further on tool development for their support and coding facilitation. Statecharts, Petri-Nets and Object Z are visual formalisms, which incorporate extensions to the previous concepts. Their mechanisms, though, are sometimes supported by classical Finite State Machines and State Transition Diagrams. These, as extensions were formally defined to augment the method's expressability regarding the capture of dynamic and static aspects.

In considering, though, behavioural properties of 'data and process' of a system, there is no mechanism to straightforwardly derive testable implementations from very often complex and too detailed specifications. The latter is a major weakness of FM, since in reality there should be direct implementations from computable business objectives! In this situation, B-Method, which is based on Abstract Machine Notation (AMN) and concepts from Temporal Logic, is a formalism with a wider application domain on specifying processes and systems inputs/outputs. However, X-Machines

(Finite State Machines generalisations coupled with Z-Method) go even further to express diagrammatically data and process, to support the direct computability of specifications and facilitate their testing. Since this method (X-Machines) demonstrates the strengths of a model-based and property-oriented abstraction to specify systems requirements, makes it possible for the method to have a general applicability in many specialised application domains via the powerful and generic abstraction it possesses as a computational model [see 1, 2, 3, 4, 20].

Thus, apart from problems associated to the mathematical, abstract nature of FM, there are plenty of more practical problems with well-known FM that are significant obstacles to grasp and use mathematical concepts as an embedded software tool and, therefore, as a thinking and learning tool. Notably, the absence of suitable modelling tools that enhance intuition, creativity and critical thinking skills from higher education and software development leads, unfortunately, to further fragmentation of mathematical thinking.

2.3 The future of formal methods in the era of new SD trends

Above all, the constant needs for modelling flexibly and understanding change in a computational [20] and integrated way [21] are, normally, requirements for the design of anything large and complex; here, the authors would like to add: as well as of small and simple functional products. The major challenge for the software developer today is, then, the following: with the maximum of peoples involvement, to utilise computational abstractions with precision and expressability in order to enable the modelling and realisation of the various systems aspects, and provide clear proof for having done so. Such meta-requirements of software development and specific considerations imply a different approach to the use of scientific and mathematical methods, and point to the careful choice of more appropriate and generally acceptable formal notations in software modelling.

Nowadays the two, rather competitive, SD trends are Agile Methodology and Open Source (OS) movement. Both trends require software quality assurance techniques and procedures [3, 4] where it seems rather impossible to avoid the use of formal methods. One of the issues relating to the use of formal methods in industry that is not frequently discussed -while it should be in reference to agile development- is that in practical software development it is rare for a set of requirements (and thus a model) to remain stable for anything other than small applications. Agile Manifesto itself praises most of the certain strengths of formal methodology, encouraging, at the same time, frequent feedback from all stakeholders –end-users in particular- involved in the software tool-facilitated development. Furthermore, agile processes are becoming very popular in the business world, while formal methods tend to ignore the business aspects [22] and they offer minimal user involvement and limited tool support. This is probably due to the complexity of mathematical notations, which require specialist knowledge and, therefore, might not be suitable for flexible and agile development processes.

Open Source Software Development (OSSD) on the other hand, requires proven implementability of software architectures and model-based testing as well as re-engineering and reverse engineering techniques for system and software quality

assessment. In OSS, apart from the implementation artefacts, and regardless of the methods used, all analysis and design products and system components must be analysed and evaluated. In order to enable the latter, accurate methods and semantic definitions should be developed each serving *generic and different* scope and purpose.

Formal methods' major drawback has been that they never supported the active role of the user [see e.g. 2, 3, 4, 5, 19, 22, 23]. Such pitfall is probably responsible for the unpopularity of formal method(ology) and the limited use. Considering human role in current software development, especially in ASD and OSSD, one might say that it is vital. Development and ASD in particular, cannot happen without end-users. In general, someone could observe that as software applications become more powerful, people and end-users in particular, become more knowledgeable and more demanding. Information system engineers are required to use acceptable development methods in order to design more detailed and understandable architectures that encapsulate highly desirable properties and a significant amount of information for further reuse.

Another major drawback in the FM paradigm is that formal notations lack structure and, thus, make it difficult to manage the development of large systems. [22]. Regarding formal methods and structure (see also [22]), many of the formal methods mentioned in table 1 did not offer diagrams in their earlier versions; consequently their use was not supported by tools with graphical representation techniques. As can also be seen (Table 1) associated tools were utilised in a limited fashion since they mostly offered text-based descriptions, and therefore did not consider users' and modellers' cognitive skills, which are mainly taken into consideration with visualisation diagrams. Observing this creative and critical thinking process in modelling system and user requirements in SD, Loucopoulos and Champion in [23] state that capturing and verifying requirements are labour-intensive activities that demand skilful interchange between those that understand the problem domain and those that need to model the problem domain. [23].

3 From Classical Scientific Paradigms to a Re-structured Curriculum

The belief of Dijkstra [8] about the fashionable choices of programming styles and accidental mathematical notations (section 2) had more strongly been expressed by Plato[1] as follows: *"I have hardly ever known a mathematician who was capable of reasoning"*. Indeed, this seems to be a strong view expressed in Ancient Greece, where the study of mathematics had been an absolute 'must' in the then less complex and breathing curriculum, and where mathematical thinking pervaded many areas of public and private life[2].

[1] As quoted in N. Rose: "Mathematical Maxims and Minims" (Raleigh, N. C., 1988).

[2] The term Education in Ancient Greece meant teaching and learning, considering three broad subjects: Mathematics and Music for a healthy mind, and Gymnastics, that is physical exercise for a healthy body. All other sciences, subjects and arts were viewed and integrated through these.

Throughout the centuries the integration of mathematics with other spheres of life and science such as Physics, Astronomy, Religion, Philosophy and Art had as main objective to achieve the highest abstraction and therefore purify the mind to view reality clearly and reasonably. Scientists, artists and philosophers should possess the so much nowadays required skills of hybrid managers in enterprises and hybrid software developers in OSSD and ASD. These multi-scientists or multidisciplinary hybrids had (and have) to filter their scientific, artistic, political and philosophical thoughts through reasonable arguments and proofs in such a procedural and axiomatic way that *anybody* should be able to attend the sequence of the thinking pattern from its initial stimulus to the resulted outcome.

The history of Science and Knowledge reveals that the most useful theoretical discoveries and applications were made by scientists whose main occupation was either in a different field or they were multi-agents of knowledge. It seems that their occupation and contact with different knowledge disciplines gave them the insight to have multiple view points for many specialised situations. Thus they were able to analyse and compare problems requirements, generalise and integrate them and finally synthesise and generalise solutions and the methods for achieving them. Since though science and knowledge became more and more specialised, it seems that only specialisation counted as a useful approach in problem solving and generalisation disappeared together with the ability to think, abstract and apply from the general to specific. This is reasonable, to some extent, since not everything is worth or possible to learn. It remains, however, difficult to think and apply from a very specialised situation to a general one, at least without problem solving guidance.

Attempting to form guidelines and providing an outline for a SE curriculum in HE, suitable to learn and apply formal methods (mathematical thinking), we must recognise the following: Each problem is different and, therefore, requires different solutions [16]. The classification of problems has also led to a classification of general ways for solutions [24]. In the same way, the classification of systems and their environments led to generalised and integrated methods [1, 2, 3, 4, 5] applied to achieve and provide solutions for different classes of problems [25]; and, therefore, to offer different software-based products and services, under different people with different skills, expertise and competencies but with a common development aim.

4 Mathematical Thinking in HE: from Stone to Jelly Curriculum

An academic curriculum is a beforehand made description of what HE plans to do for the students, the industry and the society. It also is a tool to control the evaluation of learning and to guide instructors' actions. It is a negotiated product, whose definition is not an easy matter. Portelli in [26] in 1987 found over 120 definitions, and Longstreet and Shane in [27, p. 7] in year 1993 consider curriculum as a historical accident: *"It has not been developed to accomplish a clear set of purposes. Rather, it has evolved as a response to the increasing complexity of educational decision making"*. The danger is that curriculum is like a stone, hard to change it easily from

stone to water. The real danger is to create and follow a hidden curriculum that brings down the designed and established one.

Today the challenge in SD is to create an easily changeable and transparent curriculum. It is like nailing the jelly into the wall or like guiding water flow into new paths. For this reason it is time to consider some important curriculum questions (see also [28]):

1. Are we developing a more relevant curriculum for software development students?
2. Are we really addressing complex Computer Science (CS) and Software Engineering (SE) curricula dilemmas in a clear and unambiguous fashion?
3. Are we becoming more successful at integrating theoretical issues and practical positions?

Furthermore, we first and foremost have to consider the suitability of particular elements in the SE curriculum. The following sections outline some of the very basic elements and their explanation in relevance to mathematics teaching in HE. We find them important to transfer software development curricula from stone to jelly mode. Apart from being international and multidisciplinary, the features of such a curriculum should be that it is general and generic in terms of knowledge and specialised in acquiring the necessary skills. The main aim should be to educate computer scientists and software engineers with professional and ethical competence, offering them the necessary knowledge and skills and recognising individuals' thinking and learning styles.

4.1 Problem-Based Learning (PBL): A Promising Way to Build SE Curricula

Problem-based learning (PBL) or more precisely *problem-focused education* -because everything is not worth of learning and some problems are more important than others- is, according to Boud and Felletti in 1997 [29] the most promising pedagogical innovation within the last few decades. PBL in computer science and software engineering courses is based on the assumption that problems will motivate students to learn. According to Gallagher in 1997 [30] PBL supports the integration of meaningful problem solving into everyday classroom practice. It is apprenticeship for real-life problem solving. PBL is unique in its integral emphasis on core content along with problem solving. It allows for the integration of problem solving into the curriculum and removes problem solving from the realm of ancillary instruction. However, PBL does not have to be just a method of teaching mathematics. It can be a strategy to build the whole curriculum, and even a philosophy, a way of living, thinking and learning.

According to [30] PBL requires changes in curriculum and in instruction. The curriculum is built around real-life problems, not fragmented, artificial subject matters. Educationally sound, carefully selected and well-constructed ill-structured problems need to become integrated within the curriculum. They must be appropriate for a particular curriculum with specific learning goals. A carefully constructed problem allows students to take over the job of setting a learning agenda and allows the instructor to spend class time focusing on other essential skills. Tutors and instructors have cognitive responsibilities including guiding the development of

students´ sound questioning and reasoning by giving voice to the meta-cognitive questions and metamodelling skills. Tutor (or instructor) is also responsible for matching the challenge of the problem with the abilities of the students and ensuring that the pace of the problem solving allows students to achieve a reasonable resolution in a sensible period of time. By using meta-cognitive questioning and modelling good inquiry, the tutor reveals to students how professionals really approach similar problems. PBL curriculum and classroom engage students in qualitatively different kinds of learning. However, the change of curriculum from the traditional subject-based to the problem-based is not easy for the current SE and CS tradition, students and instructors. The successful change of curriculum should not be based on the lessons of other institution but on the carefully tailored needs of the specific institution.

4.2 Teaching Thinking Skills: Cognitivist versus situative ways

In an effort to promote more meaningful learning, researchers and educators have developed programs for teaching thinking. A large part of their efforts have been dedicated to teaching metacognition, or thinking about thinking. Similarly, SE efforts have been concentrated in modelling about modelling, thinking in another level of abstraction. Metacognitive and metamodelling strategies help learners to become aware of their learning and professional strategies and own skills, and gain more control over the acquisition of substantive knowledge and modelling skills. Developing metacognitive skills is important because errors in reasoning and problem solving often arise from mindless biases [31, pp. 328-330].

We might, hereby, question if mathematical thinking should be taught as a separate subject or should it be integrated into traditional domain-specific SE classes? What is the role of cognitive psychology in teaching thinking? These questions lie at the core of the debate between two opposing camps. The cognitivist camp argues that cognitive psychology can contribute greatly to the understanding of educational practices by examining people´s problem-solving strategies and representations. The situative camp claims that cognitive psychology assumes that learning resides in the mind of the individual, instead of in the interaction between the individual and his or her environment. This camp encourages educators to turn ethnographic and sociological methods of investigating that do not separate cognition from the social context that the individual is a part of. Situative view argues that learning is tied to specific contexts and hence that learners have great difficulty transferring what they have learned from one domain to the next. [31, pp. 330-331].

The domain general view holds that thinking is a subject domain that should be studied on its own right. Students who take a course on thinking are expected to acquire general processes or strategies that can be applied to specific content domains. Teaching thinking from the general to the specific programs embraces the often-cognitivist view that students would benefit from learning general reasoning and problem-solving skills that they can then apply to specific domains. These programs tend to be taught as separate courses and to emphasize both group and individual learning. In contrast, the domain specific view holds that thinking skills are best taught in a specific context or content area. These content rich strategies can be then

applied to other areas, if there is an explicit effort to explicate how these areas are similar. Teaching thinking from the specific to the general programs embraces the often-situative view that students would benefit from learning reasoning and problem-solving skills in a rich content domain that they then can apply to similar domains. These programs tend to be part of the regular curriculum rather than individual programs, and emphasize group over individual learning. [31, pp. 332-353]. Probably the best solution for teaching thinking skills with formal methods in SE curricula lies somewhere in the middle of these two extremes.

4.3 Thinking styles: source of unexplained key factor for success and failure.

How people prefer to think might be just as important as how well they think. The thinking styles approach tries to answer to the question: how we prefer to use the abilities we have? Sternberg in [31] argues that thinking styles are as important as, and arguably more important than abilities, no matter how broadly abilities are defined. Constructs of social, practical, and emotional intelligence expand our notions of what people can do. But the construct of style expands our notion of what people prefer to do, how they capitalize on the abilities they have. A style is a way of thinking. It is not an ability or skill, but rather, a preferred way of using the abilities one has. The distinction between style and ability is a crucial one. An ability or skill refers to how well someone can do something; but a style refers to how someone likes to do something. It is also crucial for adult education curricula designers to understand that we do not have a style, but rather a profile of styles. Understanding styles can help people better understand why some learning activities fit them and others do not.

Crucial is also the match or mismatch between people´s styles and the tasks they are confronting. Different levels of schooling and different subject areas reward different styles, with the result that you can do better or worse as you go through school or job, depending on how your profile of styles matches up with what the environment expects and how the environment evaluates you. So HE institutions and organisations value certain ways of thinking more than others. And people whose ways of thinking do not match those valued by the institutions are usually penalized [31]. Mismatches become particularly serious when they occur in education or work setting. Issues of thinking and learning styles are important when building success for all students into the curriculum. If we do not take styles into account, we risk sacrificing some of the best mathematical minds and programming talents.

4.4 Critical thinking

Critical thinking is now widely seen as a basic competency, akin to reading and writing skills, which need to be taught. Critical thinking is a skilful activity, which meets standards of clarity, relevance, adequacy and, thus, is contrasted to unreflective thinking. According to Fisher in [33] critical thinking skills require the ability to interpret, analyse and evaluate ideas, arguments and observations. It also requires skill in thinking about assumptions, in asking pertinent questions, in drawing out implications, necessary skills for software development students and practitioners.

Interestingly, while many claim that teach their students `how to think`, they would further explain that they achieve this indirectly or implicitly in a course of teaching content. Educators have come to doubt the effectiveness of teaching `thinking skills´ in this way, because most students simply do not pick up the thinking skills in questions. Many of course have become interested in teaching these skills directly. The aim is to teach transferable thinking skills explicitly and directly [33].

4.5 Creativity

Creativity is the ability to produce novel, high-quality, and task-appropriate products. At least eight (8) different approaches exist dealing with the study of creativity in [31, pp. 277-289]. Among them, the type of *confluence* creativity integrates other approaches to creativity. For PBL curriculum relevant to SE in HE we distinguish as directly relevant the following four: (i) *Pragmatic*, focusing on the use of creativity and how to increase creativity; (ii) *cognitive*, dealing with the information processing and mental representations underlying creativity; (iii) *social-personal*, emphasising the roles of other people and of personality traits as well as motivation; (iv) *evolutionary*, viewing creativity as an adaptation that enhances chances of survival in SD groups.

5 Conclusive Remarks and Future Investigation

Nowadays the role of mathematics -or formal specification methods- is, once more, a new old challenge. The discourse has become very controversial and the use of FM questionable as for their suitability and applicability to the area of software development. However, in the context of world-wide civilisations, cultures and religions, the recent history of science and mathematics has had plenty of successful applications and creative achievements through mathematical thinking. The latter, while demonstrating integrated abstract thinking and the interconnections of general problem solving and specialised solutions, do not necessarily come solely from mathematicians in profession, but rather from a multidisciplinary thinking and education and people with multidisciplinary and hybrid knowledge.

In general, the use of formal methods for developing software-based systems did not lead to quality solutions in the past. It soon became clear that analysing and designing a system with formal methods offers some quality assurance regarding the development of unambiguous, consistent, correct and verified mathematically-proven specifications, but there were other issues raised. The most frequently mentioned problem that is associated with the use of most formal methods in SD is the unfriendly and fragmented approach, which prevents students' and SD stakeholders' wide understanding, and results in high costs for later training and prototype construction and testing. Consequently, one might ask: what education on formal methods are we trying to achieve?

Broadly speaking the complementary knowledge offered should be based on cognitive psychology to develop intuition and creativity and on philosophy in order to develop thinking and reasoning skills through a problem-based framework. The

importance of PBL in SE education is already well-known. Armarego' s work [34], for instance, has provided a detailed study on how a student-centred learning and innovative approach such as PBL can be put into practice successfully, taking into consideration the special educational needs and required changes of *one* institution. A wide range of critical thinking skills can also be acquired by providing a representative sample of methodological notations in an informative but comparative way of evaluating the different options. The further skills acquired can, for instance, be diagrammatic, thinking, analytical, design, modelling, reasoning, applicability and abstraction skills, to mention just a few.

In practising and teaching formal methods, we need to take into account end-users' and students' thinking styles and preferences. There is also a need to consider how SE education and practice may deprive able people of opportunities, while giving opportunities to those who might be less able or less suitable for learning and practising software development. Summarising, the authors outline some of the generic steps that, based on this paper's analysis, are suggested to be followed: a) Educating to instil SE knowledge; b) educating to develop the previously listed skills; c) educating for comparing and selecting, and d) educating for thinking ethically and professionally. Our future work will concentrate on further analysing the learning needs and styles of future software developers and fully develop a proposal for a breathing, multidisciplinary curriculum that could flexibly be specialised to accommodate many different institutional needs when in need for innovation and change. This proposal will contain the items and steps that were just outlined in the last sections of this paper.

6 Acknowledgements

The authors would like to thank colleagues and students from all cultures and countries in higher education and in industry from China, Greece, UK and Finland for sharing their experiences and commenting on the issues discussed in this paper.

References

1. Berki, E. (2001). Establishing a scientific discipline for capturing the entropy of systems process models: CDM-FILTERS - A Computational and Dynamic Metamodel as a Flexible and Integrated Language for the Testing, Expression and Re-engineering of Systems. Ph. D. thesis, Nov 2001. Faculty of Science, Computing & Engineering, University of North London, London.
2. Berki, E., Georgiadou, E. & Holcombe, M. (2004). Requirements Engineering and Process Modelling in Software Quality Management – Towards a Generic Process Metamodel. The Software Quality Journal, 12, pp. 265-283, Apr. 2004. Kluwer Academic Publishers.
3. Berki, E. (2006). Examining the Quality of Evaluation Frameworks and Metamodeling Paradigms of Information Systems Development Methodologies. Book Chapter. Duggan, E. & Reichgelt, H. (Eds) Measuring Information Systems Delivery Quality. Pp. 265-289, Idea Group Publishing: Hershey, PA, USA, Mar 2006.

4. Berki, E., Siakas, K. and Georgiadou, E. (2007). Agile Quality or Depth of Reasoning? Applicability versus Suitability with Respect to Stakeholders' Needs. Book Chapter. Stamelos, I. & Sfetsos, P. (Eds) Agile Software Development Quality Assurance. IRM Press and Idea Group Publishing: Hershey, PA, USA. March 2007.
5. Berki, E., Isomäki, H. & Jäkälä, M. (2003). Holistic Communication Modelling: Enhancing Human-Centred Design through Empowerment. Harris, D., Duffy, V., Smith, M., Stephanidis, C. (Eds) Cognitive, Social and Ergonomic Aspects, Vol 3 of HCI International, 22-27 Jun 2003, University of Crete at Heraklion, pp. 1208-1212, Lawrence Erlbaum Associates Inc.
6. Jenkins, T. (1994). Report back on the DMSG sponsored UK Euromethod forum '94, Data Management Bulletin, Summer Issue, 11, 3.
7. Cohen, L. Manion, L. & Morrison, K. (2003): Research Methods in Education. 5th Edition. RoutledgeFalmer, London.
8. Dijkstra, E. W. (1981). The Correctness Problem in Computer Science. Academic Press.
9 . Lightfoot, D. (1991). Formal Specification using Z. Macmillan Press.
10. Diller, A. (1990). Z - An introduction to formal methods. John Wiley.
11. Siakas, K., Berki, E. & Georgiadou, E. (2003). CODE for SQM: A Model for Cultural and Organisational Diversity Evaluation. Messnarz, R. & Jaritz, K. (Eds) EuroSPI 2003: European Software Process Improvement, EuroSPI 2003 Proceedings, 10-12 Dec 2003, Graz, Austria. Pp IX.1-11. Verlag der Technischen Universität: Graz.
12. Georgiadou, E., Siakas, K. & Berki, E. (2003). Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development. Messnarz, R. & Jaritz, K. (Eds) EuroSPI 2003: European Software Process Improvement, EuroSPI 2003 Proceedings, 10-12 Dec 2003, Graz. Pp. IX 31-45. Verlag der Technischen Universität: Graz.
13. Pressman, R. (1994). Software Engineering - A practitioner' s approach. McGraw-Hill, European Edition.
14. Cooling, J. E. (1991). Software Design for Real-Time Systems. Chapman and Hall.
15. Mumford, E. & Weir, M. (1979). Computer Systems in Work Design - The ETHICS Method. Associated Business Press.
16. Jackson, M. (1994). Problems, Methods and Specialisation. Software Engineering Journal, Nov.
17. Manninen, A. & Berki, E. (2004). An Evaluation Framework for the Utilisation of Requirements Management Tools - Maximising the Quality of Organisational Communication and Collaboration. Edgar-Nevill, D., Ross, M. & Staples, G. (Eds) New Approaches to Software Quality. Software Quality Management XII. Proceedings of BCS Software Quality Management 2004 Conference, University of KENT at Canterbury, 5-7 Apr 2004. Pp. 139-160, British Computer Society: Swindon.
18. Fenton, N., Hill G. (1993). Systems construction and analysis. McGraw Hill.
19 Avison, D.E., Fitzgerald, G. (1995). Information Systems Development: Methodologies, Techniques and Tools. McGraw-Hill.
20. Holcombe, M. and Ipate, F. (1998). Correct Systems - Building a Business Process Solution. Springer-Verlag.
21. Stavridou, V. (1999). Integration in software intensive systems. The Journal of Systems and Software 48, pp. 91-104.
22. Bowen J. et al. (1993). A compendium of formal techniques for software maintenance. Software Engineering Journal, Sep.
23. Loucopoulos, P. & Champion, R.E.M. (1990). Concept acquisition and analysis for requirements specification. Software Engineering Journal, Vol. 5, No. 2, pp. 116-124.
24. Rolland, C. (1999). From Conceptual Modelling to Requirements Engineering. Invited Annual ACM talk. University of North London, May 19th.

25. Sutcliffe, A.G. and Maiden, N.A.M. (1993). Bridging the requirements gap: policies, goals and domains. In the Proceedings of The 7th International Workshop on System Specification and Design. IEEE Computer Society Press, pp. 52-55.
26. Portelli, J.P. (1987). On Defining Curriculum. Journal of Curriculum and Supervision. Vol. 2, no. 4, 354-367.
27. Longstreet, W.S. & Shane, H.G. (1993). Curriculum for a New Millennium. Boston: Allyn & Bacon.
28. Marsh, C. (2004). Key Concepts for Understanding Curriculum. 3rd Edition. London: RoutledgeFalmer.
29. Boud, D. & Felletti, G. (eds.) (1997). The Challenge of Problem-Based Learning. 2nd edition. London: Kogan Page.
30. Gallagher, S. (1997). Problem-based learning: Where did it come from, what does it do, and where is it going? Journal for the Education of the Gifted. Vol. 20, no. 4, 332-362.
31. Sternberg, R. & Ben-Zeev, T. (2001). Complex Cognition. The Psychology of Human Thought. Oxford: Oxford University Press.
32. Sternberg, R.J. (1997). Thinking Styles. Cambridge: University Press.
33. Fisher, A. (2001). Critical Thinking: An Introduction. Cambridge: University Press.
34. Armarego, J. (2004). Towards achieving software engineering wisdom. Conference paper available online at: www.herdsa.org.au/conference2004/contributions/RPapers/PO44-jt.pdf (Date Retrieved: 19.10.2007).