Integrative Educational Approach Oriented Towards Software and Systems Development

http://dx.doi.org/10.3991/ijep.v3i1.2345

A.J. Stoica[1] and S. Islam[2]
[1] Uppsala University, Uppsala, Sweden
[2] University of East London, London, United Kingdom

***Abstract*—The paper is based on our academic teaching and research work in software and system engineering to effectively develop modern, complex real-life Web application systems. It bridges the gap between academic education and industry needs and illustrates how such collaboration can be successfully developed in the IT area where technology development is rapid. Its scope covers the processes, models, technologies, people, and knowledge that have the capability to contribute to developing such systems. The paper also relates to contributions of some of Harlan D. Mills award recipients for software engineering achievement, to address the needs to: i) improve the engineering education in an academic setting, and ii) develop real-life software and system projects. Hybrid educational methods are applied for student learning that combine class room approach of teaching fundamental theoretical concepts and practice via real world complex projects embedding intelligence in software and systems products. System thinking demanded by modern design philosophies is applied to interlink products, software, and people. Student groups are developing their projects in an interactive and collaborative manner.**

***Index Terms*—software and system engineering education; software theories and methods; teaching and learning strategies; systems platforms and architectures; Web-based software; teamwork**

I. INTRODUCTION

The increasing pace of change in information technology (IT) and the needs to globally address these changes in the engineering education have guided us to improve the engineering education in an academic setting taking also into account the companies needs to apply these methods for real life software projects. Therefore, teaching software and systems engineering is a complex and difficult task due to great deal of materials in variety of concepts and methods and demonstrate the applicability of the methods in a real project context. Thus learning and teaching through class room teaching is necessary to be further developed for the students to gain knowledge and skills from practical understanding.

The paper is based on our experience related to the theory and practice of computer engineering education, focusing at educational methods for Web Application Systems (WAS) development applied for i) projects designed in an academic environment for educational purpose, as well as for ii) real-life (company-related) projects. The paper presents the concepts, models, and tools integrated in a systemic approach and the lessons learned from teaching by doing, maintaining the balance between conceptual and operational aspects in software engineering education.

In our educational work, we are guided by: i) lifelong learning preparing our students for applications-oriented careers, working in all levels of computer systems engineering in particular software and systems engineering domain; ii) contributions Harlan D. Mills award recipients: Bertrand Meyer for practical and fundamental contributions to object-oriented software engineering, software reuse, and the integration of formal methods into the above; Barry Boehm for developing empirical software engineering models that consider cost, schedule, and quality, as well as software process spiral model, Theory W and the MBASE approach; Lionel Briand for his work on model-based testing and verification; David Parnas for fundamental contributions to large-scale system development by establishing software engineering as an engineering discipline.

This paper is structured as follows. Section II presents software and systems engineering in the academia. Section III provides our curriculum approach and educational strategies. Section IV is dedicated to our activity related to applied methods for software and system development: a) theoretical concepts, models, and tools; b) practical projects having real customers. Section V presents experience gained and lessons learned. Finally, discussions and conclusions are presented in Sections VI and VII.

II. SOFTWARE AND SYSTEMS ENGINEERING IN THE ACADEMIA

Software engineering is the engineering discipline concerned with the application of theory, knowledge, and practice to build effectively and efficiently quality software that satisfy the stakeholders' requirements. The development can be associated to large, medium or small systems intended for use in production environments, over a possibly long period, worked on by possibly many people, and possibly undergoing many changes. Software system development includes management, maintenance, validation, documentation, and so forth. Software professionals certainly play an important role for producing and maintaining the final software product throughout the software life cycle.

Systems engineering here is concerned with the platform (infrastructure) on which the application software is developed, hardware components that are needed, network and communication hardware and software as well as the integration of the above with the developed application software. Security, dependability, cost, schedule, and high performance are also included [17, 24]. Software and

system engineering consider both the business and the technical aspects of all customers with the goal of providing a quality product that meets the user needs [19, 24, 25, 26, 27].

The academic institutions role is: i) to produce computer engineering professionals who will build and maintain these systems to the satisfaction of their beneficiaries; ii) to provide active learning environment in particular encourage teamwork so that students will be more likely to understand the concepts and practice; iii) to train people who will belong to the top tier, taking into account that software engineering literature confirms that ratios of 20 are not uncommon between the quality of the work of the best and worst developers in a project, distinguishing the true software professional from the occasional programmer. Furthermore the academic institutions should also teach the students fundamental modes of thought that will accompany them throughout their careers and help them grow in an ever-changing field. As in hardware design, the technology evolves, but the concepts remain.

III. CURRICULUM APPROACH AND EDUCATIONAL STRATEGIES

A. Goals of Software Engineering Curriculum

Based on [11] and our practical and academic experience, we present the following goals of a Software Engineering curriculum:

Principles - lasting concepts that underlie the whole field, such as: abstraction; distinction between specification and implementation; recursion; information hiding; reuse; complexity; scaling up; designing for change; classification; typing; exception handling

Practices - problem-solving techniques that good professionals apply consciously and regularly such as: configuration management; project management; metrics; ergonomics and user interfaces; documentation; user interaction; high-level system analysis; debugging

Applications - areas of expertise in which the principles and practices find their best expression like traditional specific areas of software techniques: fundamental algorithms and data structures, compiler writing, operating systems, database systems, Web-based systems, AI techniques, numerical computing

Tools - state-of-the-art products that facilitate the application of the principles and practices. The exposure of students to the tools should proceed with a critical spirit, and their study should be understood as the study of a few examples in light of more general principles.

Mathematics - the formal basis that makes it possible to understand everything else: i) applying mathematical reasoning to software development; ii) modeling software issues in mathematical terms; iii) the engineering side of software engineering implies teaching mathematics common to engineering education: calculus, discrete mathematics, applied probability and statistics, logic, and numerical methods.

Besides formal courses, our curriculum contains a software system development project course with real customers. The course consists of developing long-term group projects in year 4 that include aspects of requirements engineering, analysis, system design, testing, and implementation, using models, processes, and tools for IT projects. Architectural models for development of sophisticated Internet applications are also included. In conclusion, our software engineering curriculum maintains balance between the principles and the techniques or between conceptual and operational areas in order to be: i) scientifically founded; ii) technically up to date; iii) firmly rooted in the field's practice.

B. Software and System Development Courses

Our undergraduate program in computer engineering contains courses that cover the previously mentioned goals: principles, practices, applications, tools, and mathematics. Specific software and systems development classes scheduled in the last two years of the undergraduate computer engineering program are:

Software Engineering basic course - discusses a comprehensive spectrum of software engineering topics and techniques.

Analytical Methods in Software Engineering – advanced level course on analytical models and methods used to study how the software factors interplay with economic and human factors in the context of various approaches to the software process.

IT Project Management, Models, Processes, and Tools – provides theoretical knowledge connected to IT project management of modern, complex, n-tier, Internet-based applications and systems.

IT Project Development – practical project development work performed in parallel with other courses during periods 2 and 3 in the last year of undergraduate studies. The course is based on the application of knowledge and skills acquired by students through their undergraduate program in computer engineering in order to develop real-life complex team projects connected either to applications in the academia or to applications generated in collaboration with IT companies.

These projects are completed before the final individual graduation thesis work in period 4.

Our basic course follows the Software Engineering Curriculum Report specified by IEEE Computer Society/ACM Computing Curricula for undergraduate degree program in Computer Engineering "CE 2004 Final Report". The other two courses extend the basic software engineering course with theoretical concepts, analytical methods, models, and tools, in order to give the students the knowledge to work with complex real projects, in particular to develop modern, complex web application systems [17]. The objectives of our specific above mentioned package in software and system engineering are:

- Educate undergraduate engineering students with software and system engineering knowledge, practice, and skills that are useful for them to become engineering professionals
- Cultivate, improve, and deploy best practices to meet their future business goals
- Active learning through real-life projects to achieve as a major output of their education - timely project delivery to satisfied customers as active partners of their education together with their academic institution. Use the "practices" concept from real-life in their education by teaching and actively using processes, methods, tools, and concepts to improve their

projects, skills, work processes, and work products created and/or acquired.

- Projects development with two main product improvement criteria: quality and productivity that require improving: i) processes; ii) people and behaviors; iii) technology and tools.

C. Teahing and Learning Strategies

We follow the *Facilitation theory* [21] for the effective teaching and learning. In particular, lecturer would more able to listen to learners, especially to their feelings and accept feedback from the students. On the other hand, learners should also take the responsibility for learning and provide much input for their learning during the tutorial tasks, open question sessions, and project works as well as reflect their understanding in course work and exam. Therefore learners construct his or her own learning through relevant learning activities and lecturers should provide accurate learning environment to support the learning activities [22]. We also follow *Sensory stimulation theory* [21] where students learn through observing and hearing which is the most effective way to learn. *Reinforcement theory* is also applicable in our context as students are always given feedback at end of the tutorial tasks and positive remark for the correct answer during the lecture session.

Our integrative educational approach is also related to *competence based education* [29] or learning in relation with a professional context. In our undergraduate program in computer engineering, we use *project based education* - students develop their final real-life team projects using the knowledge acquired during their studies. They are further developing *problem solving, critical thinking skills, oral and written communication skills, teamwork*, and follow a variety of laboratory sessions that are essential to the study of computer engineering. Therefore, the *learning process* often consider *experimental learning* using practical experience from the case study. Using *active learning* [23] as a fundamental educational method we develop our cooperation as an educational institution with industrial organizations by including in our curriculum real-life software and system development projects that provide a strong undergraduate program in computer engineering with two final specializations: i) software and ii) system (networking) engineering respectively.

IV. ACTIVITY RELATED TO APPLIED METHODS FOR SOFTWARE AND SYSTEM DEVELOPMENT

A. Theoretical Concepts, Models and Tools

There are several theoretical concepts, models, and tools that we include in our software engineering curriculum such as:

- Object-Oriented Analysis and Design (OOAD) [7], Unified Modeling Language (UML) [8,12], Unified Software Development Process (USDP/RUP) [9, 10] and associated Rational Suite Tools
- Model Based (System) Architecting and Software Engineering (MBASE) integrates models associated to: success, process, product, and properties of software and system development. Identifies and avoids model clashes (incompatibilities among the underlying assumptions of a set of models which produces conflict, confusion, mistrust, frustration, rework, throwaway systems) [3,4, 5]. MBASE can be extended to include relevant models/frameworks [14, 16,19]. Fig. 1 presents examples of classes of models used in software system engineering. Fig. 2 illustrates possible model clashes in software system engineering.

MBASE includes compatible adaptations of:

- the stakeholder Win-Win model [2]
- the DMR Benefits Realization Approach [18]
- elements of Unified Software Development Process [10,13]
- concepts such as Object-Oriented Analysis and Design, [7], Unified Modeling Language [8, 12]
- the COCOMO II suite of software cost estimation models [6]
- Spiral Model with its associated anchor point milestones and risk management models [1].

The basic steps of Model-Based Architecting and Software Engineering (MBASE) are:

- Identify success-critical stakeholders, their shared vision and value propositions
- Establish people, process, and product plans
- Monitor progress & environment with respect to vision elements and plans and apply corrective actions (shared vision, plans, experience-based updates) as necessary.

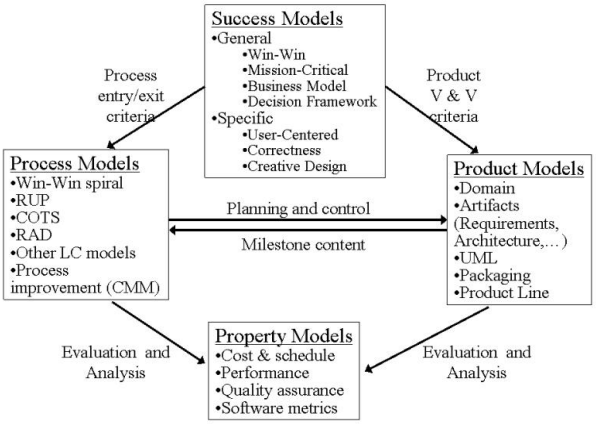

Figure 1. Classes of models used in software system engineering

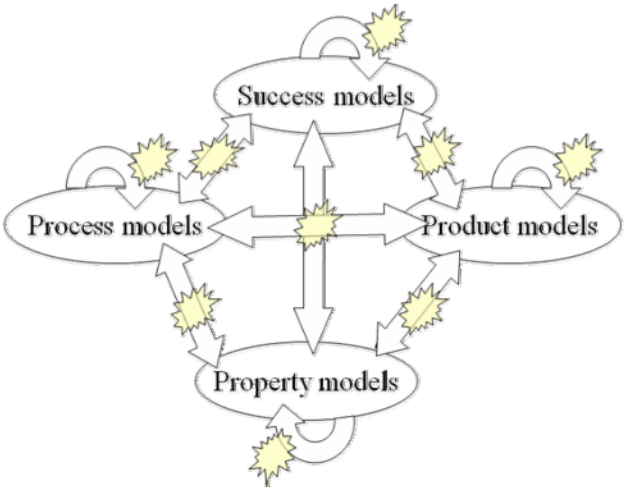

Figure 2. Models and model clashes in software system engineering

Examples of model clashes in software and system development are: i) property model clashes such as minimize cost and schedule and maximize quality; ii) process model and success model clash such as waterfall process model and "I'll know it when I see it" (IKIWISI) prototyping success model. There are several models we used in our projects:

- Success models: in general Win-Win centered. Other success models were used for specific projects.
- Process models: Win-Win Spiral Process or Unified Software Development Process
- Product models: Unified Modeling Language, Object Oriented Analysis and Design, database models, network architecture models and user interface models.
- Property models: schedule, effort, risk models.

There are success models such as Win-Win which depend on Theory W Software Project Management [2]. This theory explains the conditions to make every stakeholder a winner and is the theoretical foundation of the Win-Win approach. Applying the Win-Win model alleviates conflicts such as: i) customers requiring low budget, quick schedule, and ii) maintainers requiring well-documented software systems, no bugs. The solution is using Win-Win collaboration and negotiation model and negotiation media such as Face-To-Face, telephone, Win-Win software, other computer media. Figure 3 shows the Win-Win negotiation model. Applying the Win-Win negotiation model implies that agreements are generated as the result of a process which consists of several basic steps:

- Identify stakeholders
- Identify primary Win Conditions (WinC)
- Identify Issues needing resolution
- Offer Options as potential solutions
- Negotiate ("together") to reach Agreements.

Everything above is referred to Taxonomy and is used to develop and review major software and system development process milestones such as Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA). An example of a Win-Win situation can be letting maintainers act as quality managers during development. Developers think quality management is boring and maintainers think quality management is exciting as it makes their future work easier.

A comparison between Win-Win Spiral and IBM RUP processes is given below:

Win-Win Spiral Process

- Architecture centric
- Cyclic, in particular each cycle is Win-Win and risk driven
- Process major milestones: LCO, LCA and IOC having milestone components defined by the MBASE Guidelines.

Rational Unified Process (RUP)

- Architecture centric
- Use case-driven model
- Iterative, each iteration is like a mini-waterfall

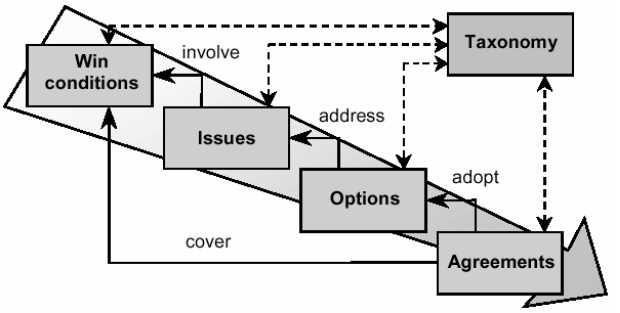

Figure 3. Win-Win negotiation model

- Four stages each consisting of >1 iterations of the software at that stage of development
- Project major milestones: LCO, LCA, and IOC, and numerous artifacts from Rational Process Library.

Both processes are i) based on best practices adopted in software projects, avoiding inventing everything from scratch and reusing processes that have been successful for other organizations; ii) architecture centered, use case driven, iterative, and risk driven. Win-Win Spiral may, but does not need to be based on use cases. Milestones and their very general goals are similar. Resulting artifacts differ significantly between the two processes (not just the names but content also). Win-Win Spiral Process is connected to the success, product, and property models. RUP Process is connected only to product models (e.g. UML models).

MBASE is more abstract - a conceptual framework for model integration that includes Win-Win, model clash analysis, and risk analysis. Associated to MBASE are Guidelines (for LCO, LCA, IOC deliverables), Electronic Process Guide, and tools (Model Guide, Process Guide, Active Templates).

RUP is more concrete - an implementation of the best practices. The IBM Rational Method Composer allows to customize RUP to meet unique needs of a project. Alternative process models such as: Waterfall, component-based development (COTS), rapid application development for hardware-oriented components (DSDM), and Extreme Programming (XP) are possible to be used but their application was not considered to be compliant with the nature of the projects under development that can vary from year to year. As an example, in the Software Engineering course students were divided into project groups. Each group planed and designed a software solution. Two different software and system development approaches were considered – MBASE and IBM RUP. The projects using the MBASE approach were: Fire & Security System, and Newspaper Delivery System and the projects using the RUP approach were: Program Invocator Agent / Workspace Handler, Fire Alarm System, Networking Game, and Newspaper Delivery System.

B. Practical Projects Having Real Customers

The IT Project Management, Methods, and Tools course had two parts: i) Part 1 – teaching theoretical and practical knowledge connected to managing the development of modern, complex software and systems, in particular Web based application systems, and ii) Part 2 – active learning by applying the knowledge from Part 1 as well as the knowledge from the other courses from the computer engineering undergraduate curriculum for

developing the students' final practical real-life complex projects. In the second part of the course, 100 students participated in real projects. The students were divided into 8 groups. After attending Part 1 of the "IT Project Management, Methods, and Tools" course the students responded to a survey regarding the provided theoretical and practical guidance for reasoning about the main aspects of complex software and system development projects in practice: management, methods, models, and tools. At the end of Part 2 of the course, a one-day workshop was scheduled for all project stakeholders with the following goals: final project presentations, discussions, and evaluations. Each project group consisted of two subgroups: i) software subgroup and ii) system (networking) subgroup. The system subgroup built the infrastructure (complete network structure: hardware and software). The software subgroup built an application based on that infrastructure. The technology used in projects, if not stated otherwise was Web-based clients, n-tier, and Java (J2EE) for server-side programming.

In the following, we provide details and results connected to applying our educational methods for software and system development for real-life projects in the following areas: (1) Project Descriptions, (2) Partial Results Obtained in the Inception Phase, (3) Use of Tools, and (4) Project Specific Workflows.

1) Project Descriptions

We illustrate the application of our educational methods for software and systems development in what follows. The following real-life (company related) projects were developed:

- *E-Commerce System* – module-based, B2B, B2C system.
- *Tool for Project Management* – modules for planning resources, time reporting and the like.
- *WAP Service* – for a major service provider. Services already available via the provider's Web-site such as making or cancelling reservations, getting info about events, were to be made accessible via WAP.
- *Estimation Tool* – implementation of an algorithm calculating Use Case Points (UCP) as a means to estimate the development effort needed for software projects.
- *General Web Shop* – implemented for a world-wide non-profit organization, Consisted of WAP-services design and implementation in addition to building a Web interface.
- *Customer Relationship Management Solution (CRM)* - a prototype of a framework, ordered by a consulting company.
- *Knowledge Management (KM) Solution* - a prototype of a framework, ordered by a consulting company.
- *Knowledge Tracking Application* – for planning and following up individual competence plans in a major company.

2) Partial Results Obtained in the Inception Phase

In the first phase (Inception) of the software and development process we present the main (partial) results synthesized from the above-mentioned real-life projects.

- *Win-win* – all stakeholders win conditions were identified and documented.
- *Requirements* – elicitation enough for prototype building.
- *Use case models* – actors and main use cases were identified.
- *Analysis models* – use case diagrams, sequence diagrams and data model were developed.
- *Initial risk analysis* – main project risks lists were identified and presented in tables
- *Process models* – all projects used the RUP process model.
- *Prototyping* – most project groups made Graphical User Interface (GUI) prototypes for checking customer requirements.
- *Architecture* – n-tier Web Systems architecture was found feasible for all projects.
- *Project planning* – project resources were estimated and allocated.
- *Tool selection* – MS Source Safe (used by three project groups) in order to manage the artifacts. Rational Rose, different prototyping tools were also applied.

3) Use of Tools

All software and systems development project groups used tools as state-of-the art products that facilitate the application of the principles and practices. The applied tools were of two categories: i) recommended tools, and ii) additional tools.

a) Recommended Tools

Rational Rose – every group used it for e.g. use-case modeling, sequential and class diagrams.

RUP – every group used it as a reference as well as the accompanying templates.

Requisite Pro – one group evaluated but concluded it was to short time to learn and implement it during the allocated project time.

MBASE – approach was used as framework concept only.

Win-Win – was used as a negotiation technique (tool was not available in Windows architecture).

COCOMO II – was not used because of time constraints and lack of experience.

b) Additional Tools

JBuilder or Visual Age – every group used it. Java was used because it is adequate for developing Web based solutions and students had previous knowledge about it.

Data modeling tools – for example Sybase Power Designer, MS Visio Technical, Direct Modeler. Data modeling course used ER – modeling, not UML.

MS Source Safe – was used for code version handling.

MS Excel –was identified to be adequate for project risk list handling.

Microsoft Project – was used for project scheduling.

4) Project Specific Workflows

We present the following project workflows associated with software and system development of the real-life projects specifications.

Feasibility Study – was based on the stakeholders Win-Win approach.

Business Modeling – was primarily used by the Customer Relationship Management (CRM) project group.

Requirements were:

- Gathered through brainstorming/meetings/use case seminars with the customer, and studying the domain literature.
- Validated through feedback on draft requirements documents and prototypes (close to the Win-Win approach).
- Documented in use case models and supplementary specifications stating the non-functional requirements.
- *Analysis and Design workflow* consisted of:
- *Analysis*
- Modeled in: logical data model, use case model, sequence and collaboration diagrams.
- *Architecture*
- Modeled in: use case models, sequence diagrams, class diagrams and network diagrams. It consisted of: n-tier, Web clients (two project groups had WAP clients); J2EE was used at server side.

Implementation – consisted of 2-3 iterations.

Testing – non-automated testing of use cases was applied.

Configuration and Change Management – three project groups used MS Source Safe, the other project groups used non-automated methods.

Deployment workflow was based on:

- Operating Systems: Windows 2000 server or Unix
- Application Servers: Bea WebLogic, IBM WebSphere or JBoss (freeware)
- Web Servers: Apache
- Data Base Management Systems: Hypersonic (freeware, Apache-connected), MS SQL Server, MS Access or IBM DB2

Project Management workflow – followed the associated course methodology.

Environment workflow consisted of:

- Integrated Development Environment (IDE) for Java programming:
  - Borland JBuilder or Visual Age
- Developing infrastructure:
  - 4 PCs for each group, usually in the roles of: Web, Application, and Database Servers
  - Network configuration
  - Firewall software.

Figure 4 shows as an example of one of the projects' system architecture.

V. EXPERIENCE GAINED AND LESSONS LEARNED

In this section, we present the experience gained and lessons learned designing educational methods for software and system development applied to complex Web based real-life projects.

A. Experience gained

The following main areas have been identified. These are:

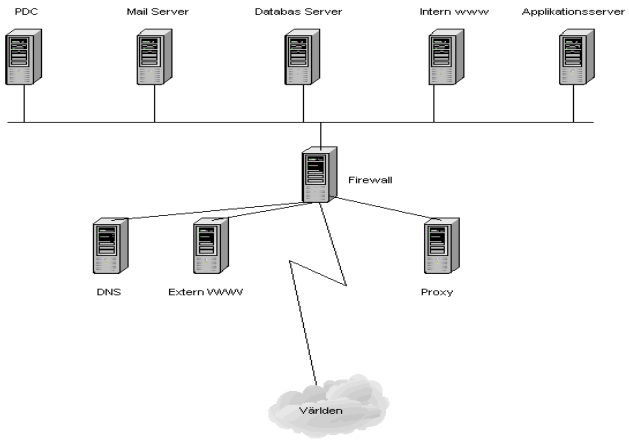

Figure 4. Project system architecture

1) Planning

Overall, all the project teams showed evidence that the time spent on the early phases of software and system development is highly relevant for the project success since it saves considerable resources/ time afterwards.

2) Documentation

The project teams' work confirmed that documentation is not merely formalism for the customer. There is an important feedback engineers/ developers can get from it in later phases. Keeping that in mind, motivation for project documenting/planning should be present during the whole software and system development process.

3) Process

One of the key problems in the development projects was identifying requirements. Solution: using a Win-Win requirement negotiation approach.

4) Technical problems

Concerning the technical problems these were more common, especially due to lack of experience with specific techniques. Solution was often to try out more than one technique in parallel and choose the one that worked best under the project constraints.

B. Lessons Learned

Our evaluation results showed that

- Effort must be dedicated to understanding processes and frameworks.
- Effort is also needed for the model adaptation to specific projects based on theoretical background in various areas (such as software and system project management, database systems, distributed systems, computer security, network components…).
- The students' lack of experience must be compensated. One way is step by step guidance using e.g. checklists, process guidelines, templates, and examples.
- The iterative process demands knowledge in all areas from start - hard for the students. An experienced project mentor was added to each project group.
- The projects could use more tool support, but there is a problem in having time to learn additional tools.
- The applied teaching and learning theory, i.e., facilitation theory, sensory stimulation theory, and reinforcement theory were well applied in our context.

VI. DISCUSSION

A. Planning

To our knowledge, we are the first to introduce these educational methods in computer engineering education (software and systems) in the last year of the undergraduate study program by:

- Designing final degree projects
- Having the theoretical part well assimilated by the 3 - course package (scheduled during periods 1 and 2 in the final academic study year)
- Thoroughly assessed by course labs and exams prior to the complex software and system projects
- Software and system development projects planned and executed during periods 2 and 3
- Close interaction with the customers (companies) and followed up by both academic teachers and company representatives during periods 2 and 3
- Final workshops – pedagogical, scientific, and practical events organized like real- life high level professional events in these areas.

B. Implementation

We address the increasing pace of change in information technology and the need to globally address these changes in improving engineering education by:

- *designing educational methods* for software and system development in an academic setting;
- *helping students become effective* in such skills as: IT project management, process definition, client interaction, requirement negotiation, software and system architecture, project organization and planning, product validation and transition;
- *taking into account the IT companies needs* to apply these methods for real-life projects.

We focus on software and systems development for Web Application Systems [17] and apply a systemic approach of methods, models, and tools integration called Model Systems (MS) [15, 19]. We use a *deep learning approach via active learning* that contributes to increasing the quality of the learning outcomes and provides a learning environment that improves students' performance.

Our educational strategies are:

- *applying teaching-by-doing* in order to achieve a balance between operational and conceptual aspects;
- *involving students in a full software and system development life-cycle*;
- *using risk-driven process models* for all projects (academic and/or real-life company-related).

Our area of application is IT undergraduate education in software engineering and networking engineering. We involve all the main stakeholders: 1) academic teachers; 2) software and system Web developers (student teams); 3) project beneficiaries (customers represented by IT companies). We assess and evaluate our common work and final results by organizing one-day *workshops* for final project presentations scheduled at the end of the undergraduate curriculum in computer engineering. Our projects meet the requirements for Web-based application systems such as: e-commerce, consumer Web, mobile and software as a service areas. These are well illustrated by the project descriptions from previous years and validated by the workshop practical results.

The one day workshop is organized as follows:

- Final project presentations by team leaders followed by practical demos;
- Project work discussion and evaluation by the other project stakeholders;
- Conclusions and suggestions for further improvements.

As a result of the workshop, the students learn from the other project experiences, compare their work with the work done by other fellow students, and develop their communication abilities. The academic teachers advice and evaluate the students as well as use the workshop outcomes for validating their educational methods. The customers evaluate the quality of the project outcomes, make useful suggestions and help the students become familiar with their future work environment. Based on the workshop results, we have a positive feedback regarding our educational methods for software and system development. Recent career news from the IEEE Computer Society [20] show that the most sought positions after 2012 are software engineers and Web developers, and high quality creative design and user-experience personnel. This confirms our role as academic institution to: i) teach fundamental modes of thought to the future software and system professionals in the above mentioned areas that will successfully built and maintain systems to the satisfaction of their beneficiaries, and ii) educate people who will belong to the top tier.

VII. CONCLUSION

In this paper, we have presented educational methods for software and systems development of Web based applications using models, processes, project management, and tools. Class room approach of teaching fundamental theoretical concepts and practice via real world complex projects are applied for student learning in software and systems engineering education. The practical results designing and implementing real-life projects based on the knowledge learned through theory allow us to draw the following main conclusions: our educational methods represent a valid improvement of software and system engineering education demonstrated by their applicability for real-life software and system projects and the high professional level of our graduates acquired during their academic studies. In future we are planning to perform more empirical studies by the students based on the software engineering methods learned during class room and active learning education.

REFERENCES

[1] Boehm B. "A spiral model of software development and enhancement", 1988, Computer 21(5): 61 -72 http://dx.doi.org/10.1109/2.59

[2] Boehm B. "Theory-W Software Project Management: Principles and Examples." 1989, IEEE Transactions on Software Engineering 15(7):902-916 http://dx.doi.org/10.1109/32.29489

[3] Boehm B. and Port D. "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them.", Software Engineering Notes, Association for Computing Machinery, pp. 36-48, January 1999.

[4] Boehm, B. and Port, D. "When Models Collide: Lessons From Software System Analysis", IEEE IT Pro, pp. 49-56, January | February 1999.

[5] Boehm, B., Port, D., Al-Said, M. "Avoiding the Software Model-Clash SpiderWeb," IEEE Computer, pp. 120-122, November 2000

[6] Boehm, B. "Software Cost Estimation with Cocomo II", 2000, Prentice Hall PTR

[7] Booch G. "Object-Oriented Analysis and Design With Applications", 1994, Addison-Wesley Pub Co

[8] Booch G, Jacobson I, Rumbaugh J. "The Unified Modeling Language User Guide", 1998, Addison-Wesley Pub Co

[9] Jacobson J, Booch G, Rumbaugh J. "The Unified Software Development Process " 1999, Addison-Wesley Pub Co

[10] Kruchten P. "The Rational Unified Process, An Introduction", 2nd edition, 2000, Addison-Wesley Pub Co

[11] Meyer B. "Software engineering in the academy", 2001, Computer 34 (5): 28 -35 http://dx.doi.org/10.1109/2.920608

[12] Rumbaugh J, Jacobson I, Booch G. "The Unified Modeling Language Reference Manual", 1998, Addison-Wesley Pub Co

[13] Royce W. "Software Project Management : A Unified Framework", 1998, Addison-Wesley Pub Co

[14] Stoica A.J. "A Decisional Framework in Software Design". Position paper, 21st International Conference on Software Engineering, EDSER1 Workshop, p.1-6, L.A., USA, May 1999.

[15] Stoica A.J. "Facets of Software Development Represented by Model Systems: Analysis and Enhancement". In: Proceedings of the 14th International Forum on Software Cost Modeling , Section 4, p. 1-24, USC-CSE, USA, Oct 1999.

[16] Stoica A.J. " A Theoretical Decisional Framework for Software Process and Applications". Technical report, Uppsala University, Uppsala, Sweden, April 1999.

[17] Stoica A.J. "Aspects of Building Web Application Systems using the MBASE Approach". In: Proceedings of the 15th International Forum on Software Cost Modeling,USC-CSE, USA, Oct 2000.

[18] Thorp J. "The Information Paradox : Realizing the Business Benefits of Information Technology", McGraw-Hill, 1998.

[19] Stoica A.J. , Babu P., Stoica P. "Quantitative Framework for Managing Software Life-Cycle", The Open Software Engineering Journal, Vol.5, No. 1, pp.1-18, Bentham Science Publishers Ltd., 2011.

[20] "Most Sought After in 2012: Software Engineers and Web Developers", BYC Newsfeed, IEEE. Accessed on 3/5/2012. http://www.computer.org/portal/web/buildyourcareer/news/

[21] Dunn, L. . Theories of learning. [Online] Oxford Centre for Staff and Learning Development Available at: http://www.brookes.ac.uk/ ,2000.services/ocsld/resources/theories.html [Accessed 5 January 2012].

[22] Biggs, J. "Aligning teaching for constructive learning", The Higher Education Academy, 1999.

[23] McGill, I. & Beaty, L.," Action Learning, second edition: a guide for professional, management and educational development", London: Kogan Page,1995.

[24] Graduate Reference Curriculum for Systems Engineering (GRCSE), version 0.5, Stevens Institute of Technology, 2011.

[25] Stoica A.J., Margus Nael, "Agile Software Development and ISO/IEC Software Quality Standards : Measuring Economic Benefits and Calculating Quantitative Yields". In: Proceedings of the 25th International Forum on Software and System Cost Modeling, pp. 1- 48, USC CSSE, USA, Nov. 2010.

[26] Islam, S. and Houmb, S. H., "Integrating Risk Management Activities into Requirements Engineering", In Proc. of the 4th IEEE Research International Conference on Research Challengenges in IS (RCIS 2010), Nice, France.

[27] Stoica A.J., Lecture notes in "Tools and Processes for Software", Stanford University, Computer Science Department, Fall 1999.

[28] Briand L., "Embracing the Engineering Side of Software Engineering", IEEE Software, Vol.29, No.4, 2012. http://dx.doi.org/10.1109/MS.2012.86

[29] Kouvenhowen W., "Competence-based curriculum development in higher education: a globalized concept?" in Technology Education and development, A.lazinica and C. Calafate, Eds., Tech 2009. http://dx.doi.org/10.5772/7297

AUTHORS

A. J. Stoica is Visiting Professor at the IT Department, Uppsala University, Box 337, 751 05 Uppsala, Sweden. She worked at KTH/SU Royal Institute of Technology/Stockholm University as Associate Professor in Computer and Systems Sciences. Her interest areas include value-based software and system engineering; frameworks for reasoning and decision-making; managing software life cycle; systems of integrated models for software and system development; associated educational methods. (e-mail: anca.stoica@it.uu.se).

Shareeful Islam is Lecturer at the School of Architecture, Computing, and Engineering, University of East London, London, United Kingdom. His research interest areas are software development risk management model; software quality specifically safety, security, and privacy; model based development and evolution; management and requirements engineering. (e-mail: shareeful@uel.ac.uk).

Received 7 November 2012. Published as resubmitted by the authors 18 December 2012.