Proceedings of the e-Skills for Knowledge Production and Innovation Conference 2014

Cite as: Zykov, S. V. (2014). Architecturing software engineering ecosystem. *Proceedings of the e-Skills for Knowledge Production and Innovation Conference 2014, Cape Town, South Africa,* 543-550. Retrieved from http://proceedings.e-skillsconference.org/2014/e-Skills543-550Zykov844.pdf

Architecturing Software Engineering Ecosystem [Invited Presentation]

Sergey V. Zykov,
National Research University Higher School of Economics,
Moscow, Russia

szykov@hse.ru

Abstract

Building an ecosystem in software engineering is a dramatic challenge due to the differences in the objectives and goals of the participating sides: governmental, industry, academic, and research organizations. The paper describes the plans and first outcomes of the recent Russian startup, the ambitious Innopolis project. The new city is under construction next to Kazan', the capital of Tatarstan Republic. The idea is to use the synergy of IT academicians, researchers, and practitioners in a single location. The city will incorporate a kindergarten, a STEM training school, a university, and an IT park. According to the construction plan, by 2030 Innopolis' population will total 155,000 people including 10,000 students of the first ever 100% Russian IT university. The Innopolis University acquires the best experience from world-known software engineering schools. The new Kazan' satellite city will provide high-end residential conditions and recreation facilities in an ecology-friendly environment. The first project results are constructing the basic infrastructure, professors and instructors training, building strategic alliances with leading universities, governmental and industry organizations, and IT companies.

Keywords: software engineering, software development skills, software engineering education, software engineering ecosystem, innovative skills.

Introduction

The objective of the paper is a systematic outline of the lessons learned by the author while being trained in software engineering at Carnegie Mellon University (CMU). The training purpose was curricula developing for Innopolis, a new innovation IT ecosystem, which is currently underway in Russia. The overall Innopolis project idea is to bring together education, research, and practice, and to create a unique IT city for versatile software development with well equipped labor recreation facilities. Currently, the project is in its initial stage.

The project location is Kazan', the capital of Tatarstan Republic, which is some 1,000 miles to the South-East from Moscow, the Russian capital. However, the project is more of a major Russian-wide startup than a local Tatar venture. It is supported by both Russian and local Tatarstan governments, and it is ambitious enough. The name Innopolis comes from Greek and means the Innovative City. The new city is going to

This paper accepted by executive review

host around 150,000 inhabitants by 2030. The Innovative City will become Kazan's satellite - the capital of Tatarstan is just 20 miles away. One big challenge is that the entire Innopolis infrastructure is being built literally from scratch: currently the nearby surroundings are uninhabited. The other obstacle is to manage the differences in the objectives and goals of the multiple participating sides: governmental, industry, academic, and research organizations. However, the initial idea to integrate the powers of the two governments, IT academicians, researchers, and practitioners in a single location will likely assist in coping with these difficulties.

Innopolis will support the entire lifecycle of IT people; the city will incorporate a kindergarten, a STEM training school, a university, and an IT park. Of these, the university is chosen as the primary focus of the paper, since it is to become the brainpower of the future city.

Concerning the Innopolis University, quite a number of aspects are essential. However, the paper focus is specifically master curricula in software engineering. The choice is made because of the author's recent CMU visiting faculty experience. The purpose of the training was to identify the ways to adjust the CMU curricula in software engineering and IT to meet the requirements of the new Russian innovative initiative.

CMU has been chosen for Innopolis faculty training since this is the birthplace of software engineering, and the top ranked university in the SE and IT field ("Colleges Rankings and Reviews," 2014). The CMU SE master's program is a well-balanced alloy of research and project practice, core and elective courses. However, direct curricula copying from CMU to Innopolis is not an entirely correct solution because these two universities obviously have certain differences in their training goals. So, the paper discusses possible ways of context-specific adjustment of the CMU courses to meet the new Russian university objectives.

The paper is organized in several sections as follows. The *Designing the Software Engineering Ecosystem* section discusses the Innopolis project and the University development outlines. The *Software Engineering: The Start-up* section covers the CMU approach to master curricula in SE and IT in more detail. The *Lessons Learned at the Carnegie Mellon* section contains the major takeaways from CMU. The *Application Outline for the Lessons Learned* section presents suggestions on SE and IT master curricula tailoring to be applied at Innopolis next academic year. The summary of the lessons learned and suggestions generated is given in the *Conclusion* section.

Designing the Software Engineering Ecosystem

Innopolis is a fairly recent project; it started from scratch in 2011. However, the ambitious idea is to co-locate a number of institutions in a tax-free and ecologically clean environment.

For Russia with its over 250 year university tradition, the new startup is still a pioneering venture. This is so because the university to be built is the first ever 100% IT higher school in the country. Prior to Innopolis, the majority of universities have been either multi-domain classical or technical ones. The school to appear will get a precise focus on applied IT and SE (the research projects to be carried on at Innopolis will also be industry-focused rather than fundamental ones).

The new project will embrace the entire lifecycle of applied IT/SE engineer. That is, the continuous education process will start at kindergarten. Further educational steps will take place at a STEM-based secondary school and the University.

Later on, internship and/or full-scale employment will follow at an IT leading company residing nearby at the technology park. All of the above-named facilities will be "tightly coupled", i.e., located within a convenient 15-minute walking distance.

The living and working conditions seem to be a lot better than at any other Russian city to date, and the salaries, from university staff to IT cluster employees, will be likely be world competitive.

The location for the new enterprise is unique. The Kazan' city, capital of Tatarstan Republic, is located just between Europe and Asia. The average distance from the city to major European and Asian destinations is around 1.5 hours shorter than that from Moscow. Thus, the Tatar capital is located closer to the geographical center of Russian Federation than Moscow. Historically, Kazan' is an Asian location, however, at present the location is fairly international. 115 nationalities, Muslims Christians and a number of other confessions peacefully coexist at this location for nearly 1000 years. The city is a megapolis with a population exceeding 1.2 million, a history of over 1200 years, and it is sometimes referred to as Earth civilization birthplace ("IT University Innopolis, 2014).

Kazan' is a world known academic and research location. Its major university was founded over 100 years ago, almost the same time as Carnegie Mellon, and it is among the top level Russian higher schools. The city has been recently renovated; its new generation infrastructure now is capable to support the top-level international cultural and sport events. A number of large IT companies are located there, represented by either headquarters (such as ICL) or subsidiaries (such as Fujitsu-Siemens). Tatarstan is #5 contributor of nearly $40 billion to Russian GRP, its production annually grows by around 7%.

The dominating sectors are oil-and-gas, automobile industry, construction, transportation, and communication. So, the location potential allows creation of world-class facilities to support the new innovation city.

According to the project plans, Innopolis City is to host over 100,000 inhabitants by 2030. Currently, the university received tuition license of Tatar Education Ministry. The faculty training is underway at Carnegie Mellon University, Pittsburgh, PA, U.S.A., which is top school of SE and IT in the world.

The project has been granted a prominent status of Federal Innovation Platform, which means it is among the strategic priorities of the Russian government. The secondary school STEM Program in robotics has already yielded nearly 1,000 graduates. Seminars of leading world IT scientists and industry representatives are regularly held at the campus, which is currently located in Kazan' and will be moved to Innopolis in 2015.

Construction of the main facilities – the major university building, residential halls and the IT park offices – is underway; it will be complete by the end of 2014. However, a lot of work is still ahead. The roadmap of the project is still in progress; it is to be finished in 2016. The Innovation City development strategy will be completed be 2020.

Though the project outline is somewhat vague at this point, it is clear enough that the university will be the brainpower of the Innopolis. Thus, the school curricula and staff training become critical issues at the moment. They will be covered in more detail in the following sections.

Software Engineering: The Start-up

The foundation of software engineering, as a discipline, has been influenced by a number of factors. One of the key events which influenced this was the historical NATO conference, where the relation between software engineering and material production was clarified (Naur & Randell, 1969). The major conference decision was that, regardless of the fact that SE has much in common with material production, the new area has a number of clearly distinct features, which require a different lifecycle and development approach.

The birthplace of the industry is Carnegie Mellon University, specifically, the Software Engineering Institute (SEI), which is interwoven into the CMU structure and activities. The SEI has been founded as a result of the historical NATO conference decisions. The SEI focus historically has been research and development of large-scale software systems with heavy duty and high reliability. The primary application of such software was military systems requested by the US Department of Defense. Since SEI is the “executive producer” of state-of-the-art standards in SE, such as SWEBOK, the CMU educational standards are probably most close to the industry ones. However, CMU had a long way prior to SEI came into being, and the initial CMU way influenced the training in software engineering a lot.

The CMU was founded in 1900 (Carnegie Mellon University, n.d.) as a synthesis of Carnegie School and Mellon Institute, which were among the leaders in US research in technology and economics respectively. The synergy of the new alloy was so powerful that the new venture quickly became a leading university in the area.

In the 1800s, the Carnegie Technology School was founded by Andrew Carnegie, who came from Scotland to the steel producing area. Among his general ideas was the learning-by-doing approach. This hands-on starting point helped him a lot to train the steel workers' children in a college-like environment. The idea was to deliver the just right amount of knowledge to master the innovative engineering technologies. The approach was practically oriented and the deliverables were real-world engineering systems and projects. The main idea of A. Carnegie's way holds true till the present day; it leads to realistic, well justified solutions for heavy duty software systems with a solid engineering-based reasoning. Such justification is based on rigorous software engineering metrics to guarantee development and maintenance of “good enough” software systems in terms of availability, performance, modifiability, security, usability, and a number of other quality attributes.

One more ingredient of the CMU success was perhaps an early adoption of cognitive approach to software engineering, which has been chosen due to tight integration with a number of psychologists who contributed to foundation of the famous CMU School of Computer Science or SCS (http://www.cs.cmu.edu).

The general framework of CMU curricula is based on the above mentioned factors of learning-by-doing, Carnegie and Mellon schools alliance, interaction with SEI and SCS. Further on the discussion will focus on masters' SE and IT courses, which currently are among the primary concerns of the Innopolis University project.

The following core MS SE/IT courses were examined during visiting faculty training at CMU:

- Analysis of Software Artifacts;
- Architectures for Software Systems (an alternative was an executive course for LG software engineers training on architectures);
- Personal Software Process (PSP).

The takeaways from the above mentioned courses are summarized in the following section, and the ideas of their application to Innopolis University curricula are given in section located directly after the following one.

Lessons Learned at Carnegie Mellon

During the training, the primary attention was focused on the Architectures for Software Systems. The course is casestudy-based (especially the LG executive version). It gives clear reasoning to change the chaotic development approaches to a process-driven one, where the general idea is to have a "just right" amount of the process to be efficient. The chief architect is free to adjust the

process to fit the project aims and scope. The course is practically oriented; it is supported by recitations and driven by a real world project (with actual customers involved and actual software product outcome) incorporated into it. Teamwork is well aligned with individual assignments. Perhaps, sometimes the individual contribution of each student to the team project is not quite clear, at least in terms of the final grade, which is heavily dependent upon team results, so a little bit of "free-riding" is quite possible. However, it is not easy for a "free rider" to get an overall pass, since individual contribution amounts to around 30% of the final grade. So, the grading policy is fair enough.

The course approach can be easily scaled up to large projects. The course Studio projects seem to be typically small/medium-scale ones. The ACDM/ATAM framework fits the Studio projects; however, it is often a challenge for the students to tailor the framework in order to fit the project size and scope. This challenge is a strong point of the course rather than a drawback, since it teaches the students to think architecturally and systemically.

Another strong point of the course is the "just enough mathematics" (J.E.M.) approach. On the one hand, the course assignments never violate mathematics. On the other hand, such an approach keeps the course focused on its primary goal, which is to form the architectural thinking.

It is a challenge to master the course. One reason for this is that it requires intuition and influences the way of thinking; these are hard things to do not only for students, but also for some software engineers.

Another MSIT/SE core course examined was the Analysis of Software Artifacts. The course is comprehensive enough. It embraces quite a number of areas, including quality planning, model checking, all kinds of testing, etc. Similarly to the previous one, the Analysis course approaches software quality in a systematic way and focuses on different techniques for quality assurance. Again, I have known a number of courses for software testing, which focus on certain techniques, e.g., "white box", "black box", and state-based testing. The Analysis course is dealing with all the above techniques to a certain extent; however, its overall goal is much broader and more complex. It introduces the idea of software quality improvement throughout the entire lifecycle. It is not just coding and testing which adds to product quality. The development processes also matter a lot. That is why the course includes - along with other standards - a CMMI overview.

Another issue, which contributes to software quality, is security. Again, this is a vast area; however, security relationship to software quality often remains somewhat subtle.

Systemic approach to quality through product security testing adds value to the course. Combinatorial testing, queuing theory, defect taxonomies, and a number of other aspects are presented together in a systematic way, which truly makes this course unique. Actually, the course presents a brief history of quality assurance, and the students are supposed to know the key dates and names. The course is supported by a "launchpad" for hands-on software quality assurance application. There is a large code repository (naturally, with a certain amount of bugs injected), and the students are applying out the quality assurance methods they have just studied in a realistic environment, i.e., to the "third-party" software artifacts.

As for reference and reading materials, these are very well supported. The course is based on a database of publications which totals to gigabytes of information and is easily full-text searchable. It takes a couple of minutes to extract the reference required and several more minutes to incorporate a quotation or a table/graph into a lecture slide. This is an impressive and hi-tech illustration to the quality of this courseware. Also, the course provides a lot of really helpful references for further reading and self-improvement concerning the above mentioned directions (and many more) and potentially assists in postgraduate studies, research, and publications.

The course fits well into the overall framework of the Studio project and requires software quality plan production at certain point. Each student team gets at least a couple of chances to present the intermediate results, and certain methods are used to engage every team not only into project presentation, but also into product quality discussion. Thus, students get the understanding of how and why does the software product quality matter. Teamwork is well aligned with individual assignments. The course is a well-justified combination of individual contributions and team presentations. It is professionally monitored and mentored.

Grading policy is quite clear to the students. The rubrics are clear, concise, and easy to use, feedback on rubrics is instant, so grading is fast and easy, and the result is fairly accurate. The grading policy is explicit and thus fair enough. Also, due to well organized processes and clearly stated "rules of the game", grading is fast and smooth, the students get a clear, well justified and prompt feedback. Teaching assistants are involved not only in lectures, recitations and grading, but also in developing assignments and rubrics.

The "just enough mathematics" approach seems be the right choice for this course as well as for the previous one. Yes, the course potentially requires a lot of more sophisticated mathematics (e.g., specific forms of Petri nets, advanced statistics, etc.). However, these would be seldom (if at all) used by the software engineering practitioners. Still, since the engineering approach has to be rigorous and metrics-based, such metrics (such as throughput) and general laws (such as Little's law) are introduced and applied instantly. Such an approach keeps the students practically focused on thinking of what adds quality to the product.

The course is introducing a way toward more efficient software development. It is not just about coding (design and testing). It is about establishing a process and a standard in development (including planning and tracking) and persistently following them. Planning and tracking takes time and effort. However, it pays for itself. One gets it really straight from the course. The course is a predecessor of CMM(I) standards. Statistics are heavily used to assess current process, detect bottlenecks and/or improvement areas, adjust the planning and tracking process and follow them. Thus, the course is focused at continuous individual improvement, which is quite a value by itself. The basic course book by W. Humpfrey (1995) is influential, since it illustrated both how disciplined should the software development process be and how this discipline pays for itself right away and eventually. Also, projecting for the future seems to be a major takeaway of the course. The course, however, is much more than just a number of book chapters. It features a toolkit for personal improvement, which also includes code reviews, Pomodoro technique for time management, and a number of other powerful things. Taken together, they form a Swiss army knife-like collection, which adds a lot to CMU students professional skills, and which is extremely useful and helpful before they actually start their intensive master level studies.

The PSP course is interwoven with the Analysis and Architecture courses since it is focused on quality and planning activities improvement to, probably, the best extent possible. Further process improvement is possible at the team level of the TSP course; however, this is a different story.

The course is supported by the Dashboard tool, which assists in planning, tracking, and estimating. Data analysis is also possible. The course features tools and techniques for data extraction and analysis, which can be performed fast enough. Grading templates are available to speed up the assessment process in case of large-scale remote student groups. The students early enough in the course focus on their individual development skills improvement and this becomes a personal motivator and driver. The more accurately the data is collected and applied, the better future work can be estimated and predicted. The course provides a number of links to current standards and policies of software development, which helps to align it with the future requirements, which the students will probably have to meet.

The course will assist in the Studio projects and their outcomes (it helps to provide document templates, planning strategies, to estimate the progress, to justify the results obtained, and to adjust).

The rubrics are easy to follow, they work both face-to-face and remotely, so grading is efficient and accurate enough. Due to well organized processes and templates, and clearly stated “rules of the game”, the students get a clear, well justified and prompt feedback.

Application Outline for the Lessons Learned

The above takeaways analysis resulted in the following considerations concerning their implementation in the MSE/IT program of the Russian Innopolis University.

The courseware set follows general principles of learning-by- doing and just-enough-mathematics. The course set is a great start for the masters' program in SE at Innopolis, since it is well thought through and practically approved at the world topmost software engineering school. Course-specific benefits, issues and challenges are discussed below.

Architectures for Software Systems

The course is practically oriented and supported by recitations and a real-world project. Teamwork is well aligned with individual assignments. The grading policy is fair enough. The course can be easily scaled up to larger projects. It is challenging for to tailor the ACDM/ATAM framework to fit the project size and scope. This is the strong point of the course, since it teaches to think systemically. The course requires intuition and influences the way of thinking, these are hard things to master. The course requires a lot of "soft skills", such as communication, time management, deep self-reflection, etc.

Analysis of Software Artifacts

The original course is designed in such a way that it is relatively easy to get a minimum pass grade. However, it is a challenge to master the course. First, this is so because the course embraces quite a number of key concepts, laws, and techniques, and it is not easy to be good enough at all of them. Second, the course is focused at practical application of nearly all the techniques introduced (either trough a Studio project or by practical assessments).

Thus, those who get best grades also acquire a passion for quality, which is vital for a software engineer. Thus, the material mastered in the course would help the students a lot in their future career. However, in order to master the above mentioned versatile concepts better, it makes sense to present a more detailed and systematic introduction of the above mentioned areas.

Personal Software Process

Many of the assignments depend upon initial level of student fluency in programming and individual performance, planning, and time management skills. The course guides, but it is not intended to instantly boost, individual development performance in each and every aspect. More achievements will happen in case the students keep improving their development processes and standards and following them. Every successful student is to get a feeling of high process quality, which is vital for a software engineer. Thus, the course material would bring discipline to the students' software development activities.

Conclusion

The paper outlines suggestions for tailoring the masters' program in software engineering taught at Carnegie Mellon University to meet the requirements of the new and ambitious IT University project, Innopolis, which is underway in Russia.

The CMU MSIT/SE curricula courses (Analysis, Architecture and PSP) are tightly interwoven, and integrated with the practically -oriented Studio project. They are focused on quality and planning activities improvement to, probably, the best extent possible.

The curricula requires special training of primary instructors and/or teaching assistants in teaching excellence/proficiency. Thus, it is highly recommended to take an adjacent iCarnegie course named Teaching Excellence Professional (TEP).

References

Colleges Rankings and Reviews. (2014). US News and World Report. Retrieved July 25, 2014 from http://colleges.usnews.rankingsandreviews.com/best-colleges/carnegiemellon-university-3242

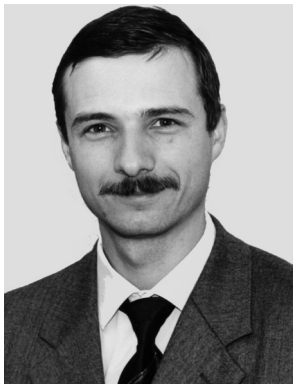
Naur, P., & Randell, B. (Eds.). (1969). *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO. Retrieved July 25, 2014 from http://dl.acm.org/citation.cfm?id=1102020

Carnegie Mellon University. (n.d.). *History & traditions*. Retrieved July 25, 2014 from http://www.cmu.edu/about/history

Humpfrey, W. S. (1995). *A discipline for software engineering*. Addison Wesley.

IT University Innopolis. (2014). Presentation. Retrieved July 25, 2014 from http://innopolis.ru/files/docs/uni/innopolis_university.pdf

Biography



Dr. Sergey V. Zykov was born in Moscow in 1971, M.Sc. in Computer Science from Moscow Engineering Physics University (MEPhI, Russia, 1994). He holds a PhD in Computer Science from MEPhI (2000, formal models and methods of ERP integration). He has a 20-year experience in IT, including 2 years as Vice-CIO of ITERA Group. He has a Certified Web Professional certificate in web design and e-commerce. He has a 20-year experience in teaching computer science and software engineering, and a mentor certificate from Carnegie Mellon University. He holds a PGCert in Higher Education from the London School of Economics. Currently, he is an Associate Professor of Higher School of Economics, Moscow Aviation Institute, Moscow Engineering Physics University, Moscow Institute of Physics Technology, and Innopolis University. He authored 10 books and over 100 papers on computer science and software engineering. Primary research fields: enterprise software development, enterprise application integration, data modeling, web portals.