PROSE
- Pragmatic Rigorous Open Software Engineering -
A Research Group Proposal

J. Paul Gibson, Telecom SudParis

October 11, 2014

1 Overview

Software engineering is not yet a true engineering discipline, but it has the potential to become one.
Mary Shaw 1990[37].

Among the ongoing problems that the project plans to fix are a lack of code reviews, an inconsistent coding style, poor documentation, no platform strategy and no regular release cycle...
Jeremy Kirk 2014[25].

Software technologies have changed enormously in the last 30 years, facilitating the development of larger, more complex systems without which the world as we know it would cease to function. However, these systems are also known to fail, with consequences ranging from minor inconvenience up to the loss of life. More often than not, these failures can be traced back to incompetence in the software engineering process, including the engineers working within that process. The development of software is still largely an undisciplined activity, with only a few notable exceptions. We propose that further research is required into why this is so, and to find ways to add rigour to the work of all software developers (in the hope that, some time in the future, they merit the title "engineer"). Why are we not surprised by Kirk's analysis of the buggy software in OpenSSL, which is a critical component in many critical networked systems? Perhaps because the potential identified by Shaw 24 years earlier is yet to be realized. We should ask "why".

2 Three Complementary Domains

This document acts as a short proposal to form a research group in the domain of *Pragmatic Rigorous Open Software Engineering*. The problem of making software engineering a "true engineering discipline" can, we believe, be best addressed by working at the intersection of three complementary domains, as illustrated in figure 1.

2.1 Formal methods for automated tool support

Ideally, system developers would all be trained sufficiently well that they would not even think that they are using a formal method or tool. They would routinely use the mathematics underlying the notation of a formal specification language simply as a means of communicating ideas to others on their team or of documenting their own design decisions.
Edmund M. Clarke and Jeannette M. Wing, 1996[8].

If you don't start off with a spec, every piece of code you write is a patch
Leslie Lamport, Turing Prize Winner 2014
"Thinking for programmers"[1]

1 Video Presentation (http://channel9.msdn.com/Events/Build/2014/3-642)

Engineering is about applying science to build a solution to a problem. Mathematics is the language that scientists and engineers use in order to add discipline and rigour to their work. Formal methods are concerned with building mathematical models of systems so that the models can be analysed and transformed in an automated fashion. Formal methods are necessary for disciplined software and systems engineering[24].

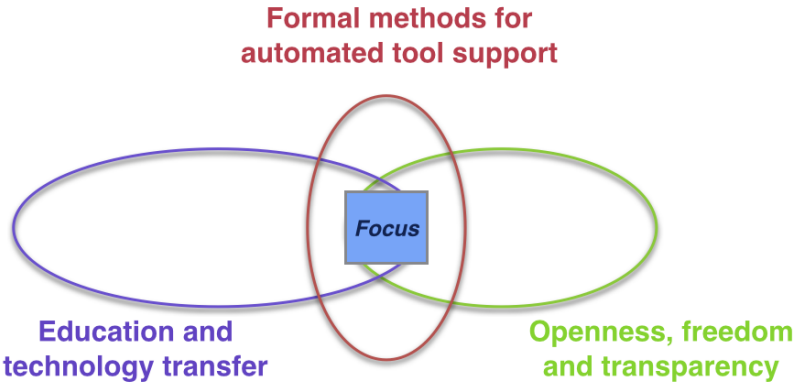

Figure 1: The research focus: at the intersection of 3 complementary domains

Lamport argues — in an amusing and insightful "fairytale"[28] — that modern-day programmers need to become more disciplined thinkers: the mathematical tools exist to help them, but the vision of Clarke and Wing — from 20 years earlier — has yet to be achieved. We should ask "why"?

2.2 Education and technology transfer

Our present approach to education for software professionals is not satisfactory to anyone.
David Lodge Parnas, 1999[35].

We do not know all we need to know about which software engineering technologies work best in which situations. And we cannot wait for years to know, given the blistering speed at which technology changes
Shari Lawrence Pfleeger, 1999[36].

Perhaps the incompetency of software engineers is the responsibility of the people who educated them? Lecturers are required to continually update the material that they teach, and the techniques and tools that they use. Are these updates too frequent, or not frequent enough? More research is needed into why software developers fail to apply well-understood techniques, that they learned during their studies, to solving problems in the real world. A second, related, problem is the transfer of novel research results to industry. Thirdly, we must ask whether the disciple is becoming too large to teach them everything? If so, do we need to take a step back and return to the fundamentals? Or do we need to accept that the world is changing and that we need to employ novel, innovative teaching techniques and technologies in order to better match the profile of our students and the needs of industry?

Dagnino's recent research shows that the lack of partnership between university and industry harms the educational process:

> *Recent graduates that start work in industry are often surprised on how many elements from the Software Engineering field they were not exposed during their academic studies.*
>
> **Aldo Dagnino**, 2014[11].

Despite Parnas and Pfleeger were saying the same thing 15 years before, the software engineering community is not learning from its educational errors. We should ask "why" and what can be done about it?

Higher-level education of engineers is moving towards adoption of innovative educational techniques and tools: MOOCs (Massively Open On-line Courses)[31], PBL (Problem Based Learning)[34], Serious Games[3], ALEs (Adaptive Learning Environments)[39] and Design Thinking[14]. Research is needed,

in the software engineering community, into how to best meet the needs of our students through the transformation of our traditional teaching material into new (open) formats, and through the integration of different technologies and pedagogical methods. The evolution and maintenance of our degree programs can and should be done using the same processes that software engineers use to manage the software life-cycle. Without such a disciplined process, our teaching will continue to fail to meet the needs of our software engineering students.

2.3 Openness, freedom and transparency

. . . , progress in science crucially depends on people being able to work together. Nowadays, though, you often find each little group of scientists acting as if it is a war with each other gang of scientists and engineers. But if they don't share with each other, they're all held back.
Richard Stallman, 2002[38].

98% of the effort [. . .] in fact returns direct learning benefits to those providers
Karim R. Lakhani and Eric Von Hippel, 2003[27].

One of the best ways for engineers to learn, and improve, is for them to examine the work of other engineers. This practice is hindered in the software world because of the way in which many systems are not delivered in an open and transparent manner. Another consequence of this lack of openness is that one has to trust the software engineers when they say that their code was developed following the highest quality standards. Much evidence suggests that the most important systems in our world are infested with software bugs. If such systems were more open, the software engineering community could play a critical role in fixing and improving them. Research is required into developing tools and techniques that promote openness and transparency during all stages of software development. As open/free/libre software reaches its adolescence, Crowston et al.'s recent review demonstrate how critical it is to better understand the engineering methods being used by this rapidly-expanding community of developers.

> *FLOSS has become an integral part of the infrastructure of modern society, making it critical to understand more fully how it is developed.*
>
> **Crowston et al**, 2012[9].

2.4 Research Communities

For each of these areas of focus, there are already significant research communities at a global scale, which is seen by the quality of the main conferences, workshops and journals:

- **Formal Methods**: Formal Aspects of Computing Journal, Automated Software Engineering Journal, Formal Methods in System Design Journal, Symposium on Formal Methods, European Joint Conferences on Theory and Practice of Software, International Conference on Integrated Formal Methods.
- **Computer Science Education**: ACM Transactions on Computing Education, IEEE Transactions on Learning Technologies, ACM SIGCSE, ACM ITiCSE, Journal of Science education and technology, IEEE Conference on Software Engineering Education and Training.
- **Open/Free/Libre Software and Systems**: O'Reilly Open Source Convention, Open Knowledge Conference, International Journal of Open Source Software and Processes, Journal of Open Research Software, Open Source Developers' Conference, Usenix Computing Systems Journal.

It should be noted that there is also much overlap between the communities, with regular specialist workshops reporting on research on the intersection between the domains.

3 A Guiding Principle : Empirical Software Engineering

For 25 years, software researchers have proposed improving software development and maintenance with new practices whose effectiveness is rarely, if ever, backed up with hard evidence.
Fenton, Pfleeger and Glass, 1994 [19].

Software engineering evidence comes from different types of studies and even synthesizing evidence from controlled experiments is a challenge
Claes Wohlin 2013[40].

The three sub-domains must be complemented by an evidence-based approach to software engineering research. Claims that formal methods, better teaching and openness improve software engineering must be accompanied by empirical evidence. In order to collect such evidence, one needs to integrate data collection into software engineering tools. In order to be able to compare evidence, this requires standardisation (using formal, mathematical models) and openness of the data and the collection mechanisms. Then, a new generation of software engineering researchers must be educated in how to use these mechanisms.

In the last 20 years there has been great advances in individual empirical studies concerned with software engineering techniques, tools, methods and processes. However, as Wohlin points out in his recent review, different empirical studies produce surprisingly incoherent evidence. We should ask "why"?

4 Two Research Project Proposals

To illustrate the potential for the group to bring together the different domains, and members' experience, in a coherent and complementary manner, we propose 2 research projects. The research projects leverage ongoing work and experience of the group members, stimulate new collaboration, and share common tasks.

4.1 Platform for Integration of Software Engineering and Education Tools

4.1.1 Summary

As software engineering educators, we are frustrated by the difficulty in motivating students to use standard software engineering tools (for documentation, testing, maintenance, version control, bug tracking, team-work, etc ...). Further analysis is needed concerning how, when and why students do use these tools. There must be a way of synthesising and analysing this data, in a coherent manner. As educators, we are also frustrated with the functionality offered by course management systems, which are not well-integrated with the software engineering tools. A common platform for composing 'software engineering services' with 'educational services' would be an ideal solution to this problem. Then, using the platform, software engineering educators could be more effective in maintaining, refining and extending teaching material; and students would be more effective in reviewing their own learning processes.

4.1.2 Scientific Context

Software engineers are expected to master the use of a wide variety of tools. Many of these are distributed as plug-ins to IDEs such as Eclipse[15], or as collaborative work services, in the cloud and on the web, such as GitHub[10]. Educators are expected to master course management systems such as Moodle[17] and a variety of educational services such as MOOCS[12], computer-assisted assessment[7], adaptive learning enivironments[39], etc ... A major problem exists with respect to integrating such tools and services, into a coherent learning framework, in order to collect meaningful data to provide feedback to students, and their teachers, concerning how students are using the tools[21]. The use of 'big data' in order to improve education processes and services is dependent on the development of standards for inter-tool communication and data sharing, and the formalism of ontologies[16]. This project will provide a development platform, defined as a service middleware, based on the formal integration of ontologies

from software engineering and from linked education. This platform will support the rapid development and deployment of new software engineering education services.

4.1.3 Research Innovation

There are three main research innovations in our proposal:

1. **Formal integration of ontologies from teaching and engineering domains:** It is well-accepted that achieving a coherent semantic integration of domain knowledge is a significant modelling challenge[33]. Fortunately, there are many commonalities between the software engineering process and the education management process[23].

2. **Interoperability of open source development tools:** Work is ongoing on the standardisation of a wide range of open source services available at common software forges[4]. Extending this work to include educational services is a significant challenge.

3. **Development, deployment and analysis of the platform:** As well as being a major challenge in building such a platform, it is important to validate that the services it offers are useful to its users. This will require 2 different types of empirical analysis — the ease by which teachers use the platform, and the academic advantages offered to students who use the services running on the platform. As a first use case study, we intend to use the platform to integrate the open IDE Eclipse with an open formal methods tool Rodin[1] and provide access through the course management system Moodle. Then, we will collect data concerning students' use of the formal methods during a large software development project.

4.1.4 Competencies Required and Group Member Participation

The group members, collectively, bring the following relevant skills which are necessary and sufficient for project success:

1. Formal Methods: Jean-Luc Raffy, Eric Lallet and J Paul Gibson

2. Ontologies and Open standards for data communication: Jean-Luc Raffy, J Paul Gibson and Olivier Berger

3. Academic software/tool support: Christian Bac, Olivier Berger and J Paul Gibson

4. Experience in development of large open-source software applications: Christian Bac, Olivier Berger, Eric Lallet

5. Teaching software engineering : Christian Bac, Jean-Luc Raffy, Olivier Berger, Eric Lallet and J Paul Gibson

4.2 Software Project Management Simulator: A serious game

4.2.1 Summary

Involving the conception and implementation of a simulator for software project management, this simulator will be in the form of a prototype consisting of three main components: a simulation engine, a database back-end and an educational game front-end. The simulation engine will cover the whole software lifecycle and will incorporate all major management decision making parameters: cost, time, risk, quality, maintenance, (human) resource management, etc. The database back-end will log use of the system users and build profiles of different management types/strategies. Analysis of the data will facilitate synthesis and analysis of the cognitive processes involved in the management process; and this will provide feedback in order to make incremental refinements of the underlying simulation engine. The game front-end will use the simulation engine to dynamically construct realistic management scenarios and to challenge the students to experiment with their virtual development teams and processes whilst playing against each other.

4.2.2 Scientific Context

Many software development projects are characterized by problems that lead to late delivery, cost overruns, buggy software and unhappy customers. These problems may be either technical or managerial in nature. In general, the technical problems seen in industry can be adequately prepared for through high level (or further) education. However, good software project management cannot be taught effectively in a traditional academic environment; it comes through experience[30]. Yet, it is undesirable and unreasonable to expect inexperienced software engineers to manage a wide range of projects in order to obtain a deep understanding of the management issues. This is precisely the reason why many projects are unsuccessful, and the global cost has been estimated to be billions of dollars. Simulation has proven itself a useful tool for education, training and development in a diverse range of industrial areas[2, 26]. It is exploited by managers to experiment with various development strategies in order to improve both the development process and the products being developed. The key is that the simulation incorporates an accurate and realistic model (abstraction) of the real world system and, to a greater or lesser extent, the environment in which it is to be used[5, 18]. The simulation can be considered as a type of expert system, and as such contains implicit knowledge of the problem domain under consideration[29].

4.2.3 Research Innovation

There are four main research innovations in our proposal:

1. **Level of abstraction of simulation models:** Currently, industrial strength software development/management simulation models are too complex (and expensive) to be used for training purposes[20]. Less complex simulation tools, often used in undergraduate education, can introduce software engineers to fundamental concepts[32], but they are too simplistic to meet our needs. Our innovative approach is to produce a simulation model whose level of abstraction can be adapted to cover a range of abilities of the model users.

2. **Automated data gathering for dynamic analysis of interaction:** Analysis of the educational effectiveness of simulations has been restricted to analysis of the impact that using these tools has had on students? ability after they have completed the particular course/module in question[13]. There has been very little analysis of how the users interact with the simulations; this is due to the fact that gathering data with respect to this interaction is complex and time- consuming. We propose building an automated data gathering backend to the simulation that will gather the necessary information for analysis without need for human intervention.

3. **Open Software for Remote learning:** To increase the chances of the simulation being used by a wide target audience, it will be released as open software, that can be executed, remotely, across the web. Thus, the potential user base will be greatly enhanced and the amount of data gathered much more likely to give rise to significant results.

4. **Plug-ins for problem-based learning (PBL):** The simulation will provide a global view of the complete software development process and improve understanding of the management issues. It is important that students can relate this high-level view with more specialised software engineering subjects that are studied as part of other modules. The simulation will provide means of plugging in specific problems into the management framework. The integration of PBL[22] into a simulation environment is highly innovative.

4.2.4 Competencies Required and Group Member Participation

The group members, collectively, bring the following relevant skills which are necessary and sufficient for project success:

1. Teaching software engineering process and management: Jean-Luc Raffy and J Paul Gibson

2. Research into simulation and modelling (and dynamic systems): Jean-Luc Raffy, J Paul Gibson and Eric Lallet

3. Development of Games for Educational Purposes: J Paul Gibson
4. Problem based learning and pedagogic theory: J Paul Gibson, Christian Bac and Jean-Luc Raffy
5. Research into technology transfer and the gap between academia and industry: J Paul Gibson, Christian Bac, Olivier Berger
6. Academic software/tool support: Christian Bac, Olivier Berger and J Paul Gibson
7. Educational Experimentation: J Paul Gibson
8. Experience in development of large open-source software applications: Christian Bac, Olivier Berger, Eric Lallet

5 Conclusion

To conclude, consider the recent warning from Bland:

> *Given the extent to which modern society has come to depend on software, the community of software practitioners must hold its members accountable, however informally, for failing to adhere to fundamental best practices designed to reduce the occurrence of preventable defects — and must step forward not to punish mistakes but to help address root causes leading to such defects.*
>
> **Mike Bland**, 2014[6].

This warning needs to be placed within the context of his report on a simple software error (a repetition of the line of code `‘‘goto fail;’’`) in a critical function, concerned with a key security protocol, developed by Apple. This error would have almost certainly been prevented by any one of the following standard software engineering practices: code review, coding standards, code coverage, unit testing, quality metrics, etc ... However, none of these were followed. We should ask “why”?

References

[1] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.

[2] Saad H Al-Jibouri and Michael J Mawdesley. Design and experience with a computer game for teaching construction project planning and control. *Engineering Construction and Architectural Management*, 8(5-6):418–427, 2001.

[3] Francesco Bellotti, Bill Kapralos, Kiju Lee, Pablo Moreno-Ger, and Riccardo Berta. Assessment in and of serious games: an overview. *Advances in Human-Computer Interaction*, 2013:1, 2013.

[4] Olivier Berger, Sabri Labbene, Madhumita Dhar, and Christian Bac. Introducing oslc, an open standard for interoperability of open source development tools. In *ICSSEA*, 2011.

[5] Thomas Birkhoelzer, Emily Oh Navarro, and André van der Hoek. Teaching by modeling instead of by models. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modelling*, pages 185–188, 2005.

[6] Mike Bland. Finding more than one worm in the apple. *Commun. ACM*, 57(7):58–64, July 2014.

[7] Sally Brown, Joanna Bull, and Phil Race. *Computer-assisted assessment of students*. Routledge, 2013.

[8] Edmund M. Clarke and Jeannette M. Wing. Formal methods: state of the art and future directions. *ACM Comput. Surv.*, 28:626–643, December 1996.

[9] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7, 2012.

[10] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[11] Aldo Dagnino. Increasing the effectiveness of teaching software engineering: A university and industry partnership. In *Software Engineering Education and Training (CSEE&T), 2014 IEEE 27th Conference on*, pages 49–54. IEEE, 2014.

[12] John Daniel. Making sense of moocs: Musings in a maze of myth, paradox and possibility. *Journal of Interactive Media in Education*, 3, 2012.

[13] Sara De Freitas and Martin Oliver. How can exploratory learning with games and simulations within the curriculum be most effectively evaluated? *Computers & Education*, 46(3):249–264, 2006.

[14] Peter J. Denning. The science in computer science. *Commun. ACM*, 56(5):35–38, May 2013.

[15] Jim des Rivières and John Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2):371–383, 2004.

[16] Stefan Dietze, Honq Qing Yu, Daniela Giordano, Eleni Kaldoudi, Nikolas Dovrolis, and Davide Taibi. Linked education: interlinking educational resources and the web of data. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 366–371. ACM, 2012.

[17] Martin Dougiamas and Peter Taylor. Moodle: Using learning communities to create an open source course management system. In *World conference on educational multimedia, hypermedia and telecommunications*, pages 171–178, 2003.

[18] A Drappa, M Deininger, J Ludewig, and R Melchisedech. Modeling and simulation of software projects. In *Proceedings of the Twentieth Annual Software*, pages 269–275, 1995.

[19] Norman Fenton, Shari Lawrence Pfleeger, and Robert L. Glass. Science and substance: A challenge to software engineers. *IEEE Softw.*, 11:86–95, July 1994.

[20] Anthony Finkelsteiin, Jeff Kramer, and Bashar Nuseibeh. *Software process modelling and technology.* John Wiley & Sons, Inc., 1994.

[21] Markus Fuchs, Markus Heckner, Felix Raab, and Christian Wolff. Monitoring students' mobile app coding behavior data analysis based on ide and browser interaction logs. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 892–899. IEEE, 2014.

[22] J. Paul Gibson and Jackie O'Kelly. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[23] J. Paul Gibson and Jean-Luc Raffy. A future-proof postgraduate software engineering programme: Maintainability issues. In *The Sixth International Conference on Software Engineering Advances(ICSEA 11)*, pages 471–476, Barcelona, Spain, October 2011.

[24] Mike Hinchey, Michael Jackson, Patrick Cousot, Byron Cook, Jonathan P. Bowen, and Tiziana Margaria. Software engineering and formal methods. *Commun. ACM*, 51(9):54–59, September 2008.

[25] Jeremy Kirk. Openssl project publishes roadmap to counter criticism, July 2014.

[26] Timo Lainema and Sami Nurmi. Applying an authentic, dynamic learning environment in real world business. *Computers & Education*, 47(1):94–115, 2006.

[27] Karim R Lakhani and Eric Von Hippel. How open source software works: "free" user-to-user assistance. *Research policy*, 32(6):923–943, 2003.

[28] Leslie Lamport. Euclid writes an algorithm: A fairytale. *Int. J. Software and Informatics*, 5(1-2):7–20, 2011.

[29] Reuven R Levary and Chi Y Lin. Modelling the software development process using an expert simulation system having fuzzy logic. *Software: Practice and Experience*, 21(2):133–148, 1991.

[30] P Mandl-Striegnitz, A Drappa, and H Lichter. Simulating software projectsan approach for teaching project management. In *Proceedings of the INSPIRE III: Process Improvement Through Training and Education*, pages 87–98, 1998.

[31] Fred G. Martin. Will massive open online courses change how we teach? *Commun. ACM*, 55(8):26–28, August 2012.

[32] Emily Oh Navarro and André Van Der Hoek. Software process modeling for an educational software engineering simulation game. *Software Process: Improvement and Practice*, 10(3):311–325, 2005.

[33] Natalya F Noy. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, 33(4):65–70, 2004.

[34] Michael J OGrady. Practical problem-based learning in computing education. *ACM Transactions on Computing Education (TOCE)*, 12(3):10, 2012.

[35] David Lorge Parnas. Software engineering programs are not computer science programs. *Software, IEEE*, 16(6):19–30, 1999.

[36] Shari Lawrence Pfleeger. Understanding and improving technology transfer in software engineering. *Journal of Systems and Software*, 47(2):111–124, 1999.

[37] Mary Shaw. Prospects for an engineering discipline of software. *IEEE Software*, 7:15–24, November 1990.

[38] Richard Stallman. *Free software, free society: Selected essays of Richard M. Stallman.* Lulu. com, 2002.

[39] Mieke Vandewaetere, Piet Desmet, and Geraldine Clarebout. The contribution of learner characteristics in the development of computer-based adaptive learning environments. *Computers in Human Behavior*, 27(1):118–130, 2011.

[40] Claes Wohlin. An evidence profile for software engineering research and practice. In *Perspectives on the Future of Software Engineering*, pages 145–157. Springer, 2013.