New Orleans JAZZED about Engineering Education

Paper ID #17202

Enhancing Software Engineering Curricula By Incorporating Open, Data-Driven Planning Methods

Mr. John (Lalit) Jagtiani, University of Bridgeport

Mr. Lalit (John) Jagtiani is currently a Ph.D. candidate focused on Technology Management at the University of Bridgeport, School of Engineering. His research interests include software technology management, software metrics, technology change management, and technology risk management. Mr.Jagtiani has 25+ years of industry experience with technology management and strategic business solutions. He currently serves as a consultant to several organizations and teaches graduate level courses at the University of Bridgeport.

Dr. Neal Lewis, University of Bridgeport

Neal Lewis received his Ph.D. in engineering management in 2004 and B.S. in chemical engineering in 1974 from the University of Missouri – Rolla (now the Missouri University of Science and Technology), and his MBA in 2000 from the University of New Haven. He is an associate professor in the School of Engineering at the University of Bridgeport. He has over 25 years of industrial experience, having worked at Procter & Gamble and Bayer. Prior to UB, he has taught at UMR, UNH, and Marshall University. Neal is a member of ASEE, ASEM, and IIE.

Enhancing Software Engineering Curricula By Incorporating Open, Data-Driven Planning Methods

Abstract

For several decades, Software Engineering (SE) course work has been an integral learning subject for computer science, information systems, and technology related matriculated students at undergraduate and graduate levels. During the same period, industry has continued to experience high failure rates, missed business outcomes, and poor user adoption for large software development initiatives. While there are many reasons for these issues, educators must focus on ensuring that SE course curricula and learning outcomes strive to create long-term positive impact. This paper will address how to improve readiness levels of students in learning environments by integrating best practices that can address these issues in the context of SE course work.

We create a clear case for change within the SE education pedagogy, and a research method to achieve the objective is outlined. The SE course curricula, current standards, and intended learning outcomes are examined which enables us to outline the changes required. Recommendations and suggested methods to help close the gaps are outlined.

Introduction

Many industrial and academic oriented studies have examined software development performance over the last 30 years. Theoretic and practice oriented experts recognize that several reasons exist for the sub-optimal performance. While much of the failure can be attributed to industry and business climate forces, SE educators want to be sure that the foundation which they provide to students is sound, realistic and aligned with desired learning outcomes.

In this paper, we explore how experience-based software project data can serve as predictors and play an important role in the management of SE projects. Predominantly, today's SE curricula do not adequately focus on these aspects. The goal is to present the case for change and to suggest practical and flexible methods of improvement. To achieve the goal, this paper examines why SE efforts experience high failure rates, how current curricula are structured, the intended learning outcomes of SE courses, and where there is a gap in the achievement of those outcomes. Finally, practical suggestions are made for enhancing the standard SE course syllabi. These suggestions are offered in the form of implementable methods which can enable educators to infuse an experiential, fact-based, and data-driven planning and measurement approach to current SE course work.

Literature Review

Numerous studies, surveys, and assessments are periodically done by organizations and independent third parties to better understand software project failure rates. The CHAOS Report published by The Standish Group is an example of a trusted source has been depicted in Figure 1[1]. The organization has been publishing the report for more than thirty years and has used the same criteria for project classification since inception. The report delineates project outcomes into three different categories:

1. Successful: Projects that are completed on-time and on-budget, with all features and functions as initially specified.
2. Failed: Projects that are cancelled at some point during the development cycle.
3. Challenged: Projects that are completed and operational but over-budget, over the time estimate, and/or offer fewer features and functions than originally specified.

Analyses show high failure or challenged project rates with the root cause centered squarely on planning, readiness, and assessment. Mandal and Pal clearly declare that their "research indicates that more than 50% of all Information Technology (IT) projects become runaways – overshooting their budgets and timetables while failing to deliver the expected outcomes"[2-4]. Further to this, if we examine project results based on the triple constraint model of time, cost, and scope we find that the outcomes are even more concerning. In Table 1 below, we see that the majority of the projects slip on two of the most important constraints: time and cost.

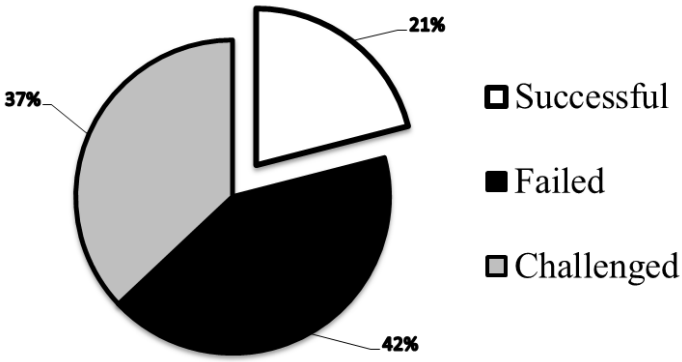

Figure 1. Software Project Success, Failure, and Challenged Rates[1]

Year	Successful	Challenged	Failed	Unsuccessful	Time Overrun	Cost Overrun	Undelivered Scope
1994	16%	53%	31%	84%	N/A	N/A	N/A
1996	27%	33%	40%	73%	N/A	N/A	N/A
1998	26%	46%	28%	74%	N/A	N/A	N/A
2000	28%	49%	23%	72%	N/A	N/A	N/A
2002	34%	51%	15%	66%	N/A	N/A	N/A
2004	29%	53%	18%	71%	84%	56%	36%
2006	35%	46%	19%	65%	72%	47%	32%
2008	32%	44%	24%	68%	79%	54%	33%
2010	37%	42%	21%	63%	71%	46%	26%
2012	39%	43%	18%	61%	74%	59%	31%

Table 1. Project Performance Statistics[1]

There seems to be no meaningful improvement year-to-year for software projects for these two constraints even while scope is not fully addressed as planned. Project success is often attempted

by reducing scope rather than improving performance using other levers[5]. This raises a significant issue for SE organizations to contend with, as it plagues aggregate productivity resulting in missed expectations and allocation of greater resources than planned. Software quality predictors have been studied by several researchers in the past. In one such study, SourceForge, an open source repository, was analyzed and it showed that software classifiers such as the number of downloads, rank, OS, language, and days to build can in fact be examined to predict outcomes[6].

Intuitively, most SE students and educators understand that project planning and measurements tied to success criteria is paramount to achieving SE project success. Leveraging past experiences that are fact-based is important in the planning process for software development. The concept of Analogy-based Software Development Effort Estimation (ASEE) has been explored by many researchers and is based on the principle that actual performance values achieved earlier, and within a similar context, are better indicators and can predict future project performance better than scratch estimates. These researchers also suggest that ASEE techniques outperform the other prediction models[7].

Equally important to using proven predictors is the notion of establishing metrics that can be tied to processes being addressed programmatically as part of the software engineering process. Catal suggests that metrics based models are so important that they must be frequently revised as software engineering is underway – perhaps even rebuilt from scratch each time the process or organization experiences a change[8]. Other researchers have also suggested that evaluation and prediction are two separate learning schemes using historical defect data to predict defects for new data[9].

Researchers and practitioners have acknowledged that there is an abundance of free and open source tools available and many are employing these to meet their software requirements. The use of these tools has resulted in large Open Source Software (OSS) communities of active and engaged contributors documenting the features of the product and enabling the release of frequent enhancements. Proper due diligence can be easily facilitated to determine track record, performance, and maintenance aspects of software[10]. Therefore, OSS communities can be a robust source for use, benchmarking, and serve as a knowledge base to students planning an SE project.

After examining SE failure reasons and potential areas that can be alleviated, the literature review was extended to review current SE course curricula, standards, and learning outcomes. A key report published by the IEEE Computer Society was inspected. The IEEE Computer Society is a leading organization dedicated to computer science and technology. The group serves over 60,000 members and is a trusted information source for software engineering professionals. The group is a leading authority on SE and publishes a credible body of knowledge to promote the advancement of both theory and practice in the field[11]. In 2004, with the support of The National Science Foundation and The Association for Computing Machinery, a joint task force was established to develop an exhaustive report which included a complete SE course curricula with learning outcomes which can be adopted by learning institutions[12].

Most SE related courses offered in universities today remain consistent with the authoritative IEEE report even today. Niazi generally acknowledged the SE project failure reasons reviewed earlier and after studying SE projects in the U.K., he claimed that “most of these reasons are due

to the lack of effective training programs which can provide sufficient knowledge and skills relating to global software engineering"[13].

Research Method

A gap between the theory and practice exists. SE educators and practitioners are well intentioned but may have differing objectives. This difference could stem from personal agendas, experiences, and individual perspectives. This may result in a lack of alignment between sound SE practices and SE study curricula inhibiting successful outcomes already underscored in the literature review. Furthermore, our hypothesis suggests that there may be a difference between SE curricula and educators' desired learning outcomes. Our research analysis focused on 3 key steps:

1. Examine past survey data and identify the relevant problem experienced by the SE industry. This step highlighted the extent of the problem for projects and identifies the root-cause. Since the scope of this research is focused on SE education, the remainder of the study was focused on SE training and curricula.
2. Understand the current standards, course curricula, and learning outcomes intended for use by academia. This step highlighted the gaps between intended learning outcomes for SE course work and actual SE course structures – the difference between theory and practice.
3. Recommend enhancements to address gaps.

Results

Software projects are either successful or unsuccessful considered in the aggregate. Figure 2 shows that although the trend shows more optimism, the case for change is even greater if we group year-to-year challenged projects and failed projects statistics from above Table 1:

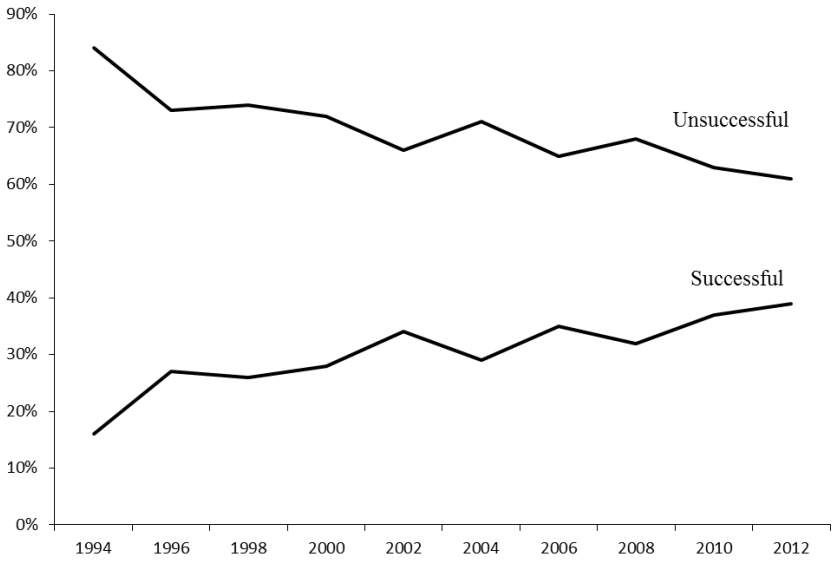

Figure 2. Aggregate Project Performance

Furthermore, in Figure 3, we see that overruns on software project time and cost and under delivery of project scope is more evident if we consider product features that remain undelivered.

Time and cost delays are a much greater cause of project failure than is undelivered project scope:

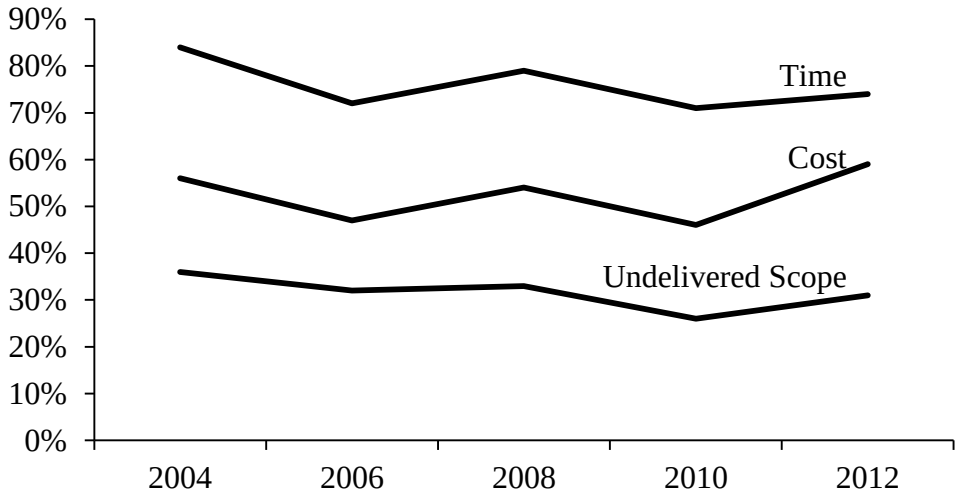

Figure 3. Failed Projects – Rate of Project Overruns and Under Delivery

Intuitively, since time and cost are discrete and much easier to compare and report than scope deviations, improved planning and measurement methods can have a positive influence on project outcomes.

Our fear is that while the published IEEE Software Engineering 2004 Curriculum Guidelines[12] recommended strong learning outcomes, the suggested course content which it outlines may not realize the desired outcomes. Insufficient focus on planning and metrics and measurements was evident. Table 3 summarizes the learning outcomes outlined by the report and our assessment for where these two focus areas are strongly required.

	Summary of Intended Learning Outcomes For SE	Gap Assessment: Course Topics Coverage
1.	Gaining knowledge of SE and issues.	Planning, Metrics Required
2.	Learning to work individually and in teams.	Topics Adequately Cover
3.	Resolving conflicts between cost, time, knowledge, existing systems, and organization.	Planning Required
4.	Designing appropriate solutions that satisfy and integrate ethical, social, legal, and economic concerns.	Planning, Metrics Required
5.	Learning to apply theories, models, and techniques to identify problems, implement solutions and verify results.	Planning, Metrics Required
6.	Understanding the importance of negotiation, effective work habits, leadership, and good communication.	Topics Adequately Cover
7.	Learning emerging models, techniques, and technologies as they emerge and the importance of doing so for ongoing SE efforts.	Unable To Determine

Table 3. Comparison of Learning Outcomes and Course Topics with Gaps Identified

The IEEE guidelines also highlight a comprehensive set of 28 suggested SE related courses. It is implied that universities may offer any or all of these courses to teach students all there is to know about SE. However, though conceivable, it is unlikely that a given university will offer every suggested course at any level – undergraduate or graduate. Even so, each suggested course syllabus was analyzed as part of this research project to determine the level of coverage it places with respect to the following key variables:

1. Front-end planning related topics (planning focus) and
2. Metrics and measurements (metrics focus) related topics that can be used throughout the software development life cycle (SDLC)

A simple methodology was used to assess each of the suggested courses in the report. A rating from 1 to 10 was given to each variable after careful inspection of individual syllabus suggested in the report. The values were summed up and are shown in Figure 4.

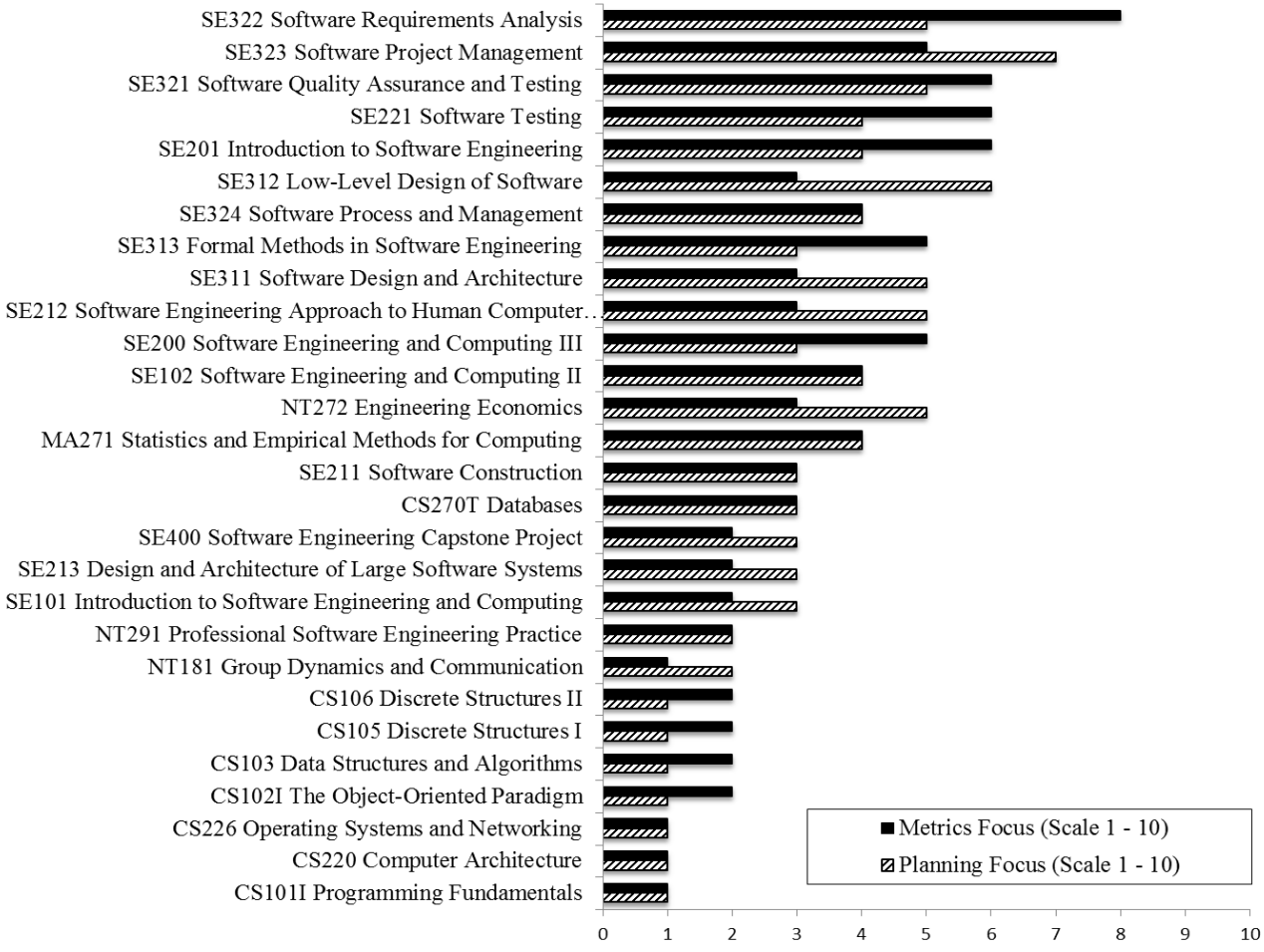

Figure 4. SE Related Courses - How Much Focus is there on Planning and Metrics?

It is evident that the case for change and our hypothesis demonstrates some merit. Of particular interest to concerned educators will be the following 3 results of the analysis:

1. There is inadequate course coverage of critical topics required for long-term successful SE efforts – both planning and metrics related topics are inadequately covered across the entire population of suggested courses given current standards (i.e. low population mean defined as μ):

   $\mu_{planning\ focus} = 3.2$ and $\mu_{metrics\ focus} = 3.3$

2. The sample of common SE courses from the list of all courses also has inadequate coverage of the critical topics required for long-term successful SE efforts. A conservative tabulation of the courses suggests that only 6 out of the list of 28 suggested courses are standard SE course offerings in most universities – namely, Introduction to Software Engineering and Computing, Software Engineering Capstone Project, Software Engineering and Computing II, Software Engineering and Computing III, Formal Methods in Software Engineering, and Introduction to Software Engineering. Planning & metrics related topics are inadequately covered across these common SE courses (i.e. low sample mean defined as $\overline{x}$):

   $\overline{x}_{planning\ focus} = 3.3$ and $\overline{x}_{metrics\ focus} = 4.0$

3. The best coverage of the critical topics required for long-term successful SE efforts is evident in non-traditional SE courses which are very often not offered or required by universities. The 3 top most rated courses where planning and metrics topics are not among the common SE course offerings in most universities include Software Requirements Analysis, Project Management, and Software Quality Assurance & Testing.

Evaluation, Assessment, and Recommendations

The premise for this research is that having improved focus on planning SE efforts will help students during the early stage of learning, before they join industry. Such learnings can instill best practices in the minds of students before adverse experiences of the commercial world. Most universities adopt curricula which have strong learning outcomes; however, we can see that course structure and topics may not be aligned with the intended outcomes. As a result, educators risk their students missing out on key learnings necessary for good software development practices in the long run.

To address the above, the following 2 recommendations are being made:

Recommendation #1: Enhance SE Course Curricula

The level of industry adoption of sound SE practices is dependent on two factors: 1) the business environment and the resulting pressure placed on software engineers and 2) the quality of training and readiness that has been delivered to future software engineers and those who lead them. Since the triple constraint model, which includes time, cost, and scope is always in play for all SE projects, these constraints must be carefully planned for and measured throughout the SDLC. Performance against these three constraints has been sub-optimal in industry and the trends have not improved upon further analysis. This is particularly the case for large SE projects. Therefore, the learning outcomes of SE courses which have been outlined earlier in this paper must be achieved by including key topics and techniques generally missing from SE courses.

The following enhancements are being proposed to enhance all critical SE related syllabi. These techniques offer ample flexibility with respect to implementation:

1. Conduct external benchmarking – increase the focus on fact-based and data-driven planning techniques using data from open source software repositories.
2. Conduct internal benchmarking – build, nurture, and institutionalize an open source repository of SE projects within the university (and perhaps across universities).
3. Identify and implement success metrics – explicitly include the derivation and ongoing use of metrics and supporting measurement techniques during the SDLC for all SE projects expected from students as part of course work. Make sure that metrics are Specific, Measurable, Attainable, Realistic, Timely (S.M.A.R.T.)

By adopting the above enhancements to required SE course curriculums, we can improve upon the learning objectives:

Recommendation #2: S.M.A.R.T. Leverage of Open Source Software (OSS) Communities

Effective external benchmarking activities must be encouraged and taught in SE courses in order for students to understand modern and relevant software benchmarking techniques. An easily adaptable method is to use one of several large and reliable OSS repositories. These repositories are easily accessible on the internet without cost. In this paper, we discuss an example of such a repository (SourceForge.net) and how to potentially use it as an external benchmarking tool for SE planning purposes. This repository can be used by students to publish and distribute their completed or work-in-progress SE projects. SourceForge.net serves as an open source community resource and thrives on open community collaboration to facilitate open source software development and distribution. This single resource boasts over 430,000 projects and with a registered base in excess of 3.7 million users. Software professionals use SourceForge.net to develop, download, review, and publish open source software making it one of the largest such repositories in the world.

Below are suggested steps on how an SE student can leverage an OSS such as SourceForge.net to effectively plan their SE course project for success:

1. Identify key software project attributes – e.g. application and purpose, programming language, operating system and integration requirements.
2. Search SourceForge.net by project attributes. Leverage attributes that are S.M.A.R.T. where possible.
3. Select 5 open source projects that are similar in size, scope, and complexity. Depending on risk tolerance and criticality of project, students may select more or less projects to benchmark.
4. Review developer forums and discussion boards related to each of the 5 short-listed projects and determine experiences related to the triple constraints – time, cost, and scope.
5. Directly validate with individual developers of benchmarked projects for more precise information on their experiences with respect to the triple constraints. Open source developers are generally very collaborative and happy to help fellow developers. To demonstrate gratitude, SE students may consider acknowledging collaborators who gave valuable inputs to help make their SE project a success.

This process is an example of what can be directly included in a SE class to enhance planning efforts. These steps, with representative visuals, are show in Figure 5 below:

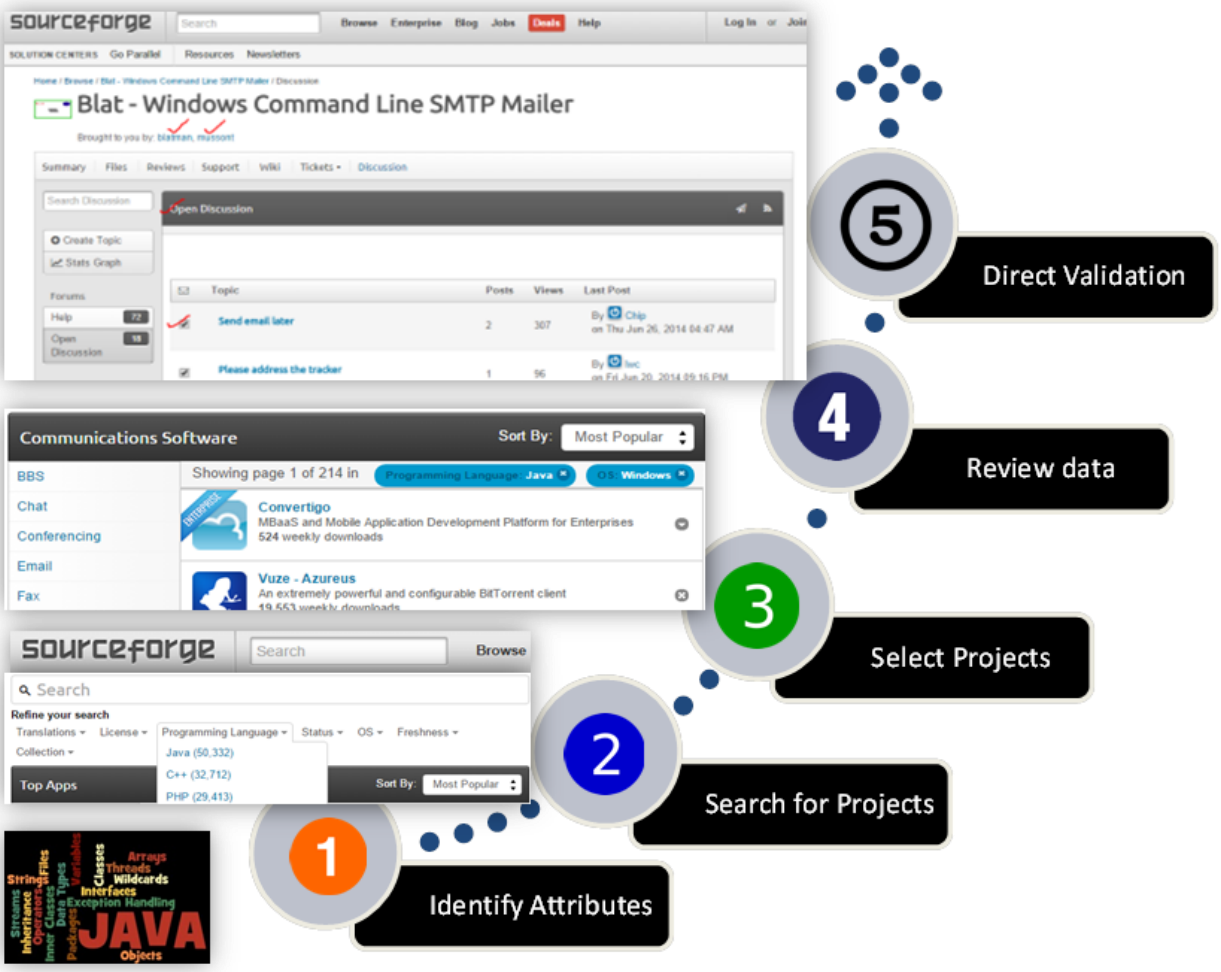

Figure 5. An Open, Fact-Based, Data-Driven Approach to Planning SE Projects

Concluding Remarks

Recognizing real world experiences in SE enables educators and administrators in academia to support critical changes to SE curricula, when required. There is insufficient emphasis being placed on upfront planning, benchmarking, and metrics. Important changes must be made now to SE training methods to teach future SE matriculated students how to utilize experienced-based knowledge to improve software project outcomes. The changes required focus on including more planning and ongoing metrics validation for SDLCs. This is critical for successful SE projects inside the classroom which should eventually have a positive influence on industry projects in the long run.

Aligning course work with the recommendations made in this paper will better prepare students who may assume key planning roles within industry or academia in SE. In this paper, it has been demonstrated how SE course learning outcomes, that are otherwise on point, remain at risk of

not realizing intended outcomes if status quo is maintained. Strong convictions have been discussed regarding the current issues in SE and the role of benchmarking and a metrics based approach which can start to address these issues from the very foundation – the SE education curricula. The good news for educators is that there remains ample flexibility on how suggested techniques can be implemented, the selection of tools and platforms which can be used, and how much is to be adopted in a classroom setting versus the use of a disciplined, supported, and self-paced learning approach. Finally, in this paper, we shared an enhanced student use case to show how OSS repositories can offer an easy mechanism to enhance the software planning process in SE courses.

As a final note, recommendations made in this paper can offer valuable insights to all those who are developing curricula for course work in all SE related disciplines such as software management, project management, new product innovation, product commercialization, management of technology, information systems analysis, and technology change management. The hope is that the research presented in this paper may serve as a launch pad for further research on this subject and the expanded use of innovative tools to achieve better SE results. Future researchers must continue to develop methods that can be easily adopted by educators who chose to heed the call for action. Should curriculum owners within universities decide to adopt any of the recommendations made in this paper, the software industry will benefit for years to come.

References

1. *The CHAOS Manifesto, 2013: Think Big, Act Small*, 2013, The Standish Report International.
2. Al-Ahmad, W., et al., *A taxonomy of an IT project failure: Root Causes.* International Management Review, 2009. **5**(1): p. 93.
3. Fabriek, M., et al. *Reasons for Success and Failure in Offshore Software Development Projects*. in *ECIS*. 2008.
4. Mandal, A. and S. Pal, *Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models.*
5. Jagtiani, L.J. and N. Lewis, *The Impact of Big Data on the Management of Business Software Technology Projects.* 2015.
6. Tsatsaronis, G., M. Halkidi, and E.A. Giakoumakis, *Quality Classifiers for Open Source Software Repositories.* 2009.
7. Idri, A., F.a. Amazal, and A. Abran, *Analogy-based software development effort estimation: A systematic mapping and review.* Information and Software Technology, 2015. **58**: p. 206-230.
8. Catal, C., *Software fault prediction: A literature review and current trends.* Expert systems with applications, 2011. **38**(4): p. 4626-4636.
9. Song, Q., et al., *A General Software Defect-Proneness Prediction Framework.* IEEE Transactions on Software Engineering, 2011. **37**(3): p. 356-370.
10. Dowling, P. and K. McGrath, *Using Free and Open Source Tools to Manage Software Quality.* Queue, 2015. **13**(4): p. 20-27.
11. Bourque, P. and R.E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*2014: IEEE Computer Society Press.
12. The Joint Task Force on Computing Curricula -- IEEE Computer Society, A.f.C.M., *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering -- A Volume of the Computing Curricula Series*, 2004.
13. Niazi, M. *Teaching Global Software Engineering: Planning and Preparation Using a Bloom's Taxonomy*. in *Proceedings of the World Congress on Engineering*. 2013.