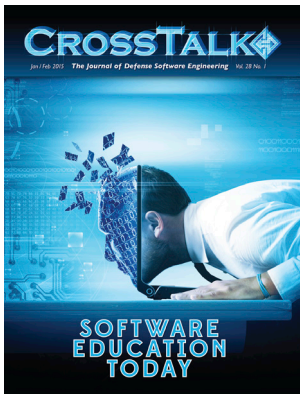
CROSSTALK

Jan / Feb 2015 The Journal of Defense Software Engineering Vol. 28 No. 1

SOFTWARE EDUCATION TODAY

TABLE OF CONTENTS

Departments



Cover Design by Kent Bingham

Software Education Today

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Article Coordinator Heather Giacalone
Managing Director David Erickson
Technical Program Lead Thayne M. Hill
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-777-9828
E-mail Crosstalk.Articles@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: Office of Cybersecurity and Communications in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: CrossTalk is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the CrossTalk webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>.
ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank 309 SMXG for sponsoring this issue.

From the dawn of the modern age, technological advances and innovation have been sustained and further enhanced by our educational systems. With the backdrop of globalization, rising economic pressures and emerging foreign powers, the need for our education system to maintain a position of preeminence in the fields of science and technology have become more pressing than ever before. Educators, industrial leaders and governmental officials often advance differing opinions on the most effective way to ensure continuation of our country's dominance in a plethora of areas or suffer a real and palpable economic and societal loss in mere decades. The current state of our academic institutions is especially pertinent to military departments who must ensure a security and national defense posture, now and into the future. As the pace of technological innovation continues to increase, so has the perceived disconnect between academia and industry in preparing our graduates for immediate real-world integration in the science and technology industry. It is for that reason that we have chosen to highlight the topic of software education to begin the year. Here we will attempt to highlight both the perceived challenges industry faces with recent graduates as well as provide perspective into the difficulty academia faces in adjusting to the rapid pace of technological innovation in curriculum.

In this issue, we will explore the perceived need to further align academic curriculums of our higher educational institutions to face the needs of both research and development organizations as well as illustrate methods to allow those organizations to be more productive through educational concepts. We begin with an collaborative article focused on the software industry entitled "Missed Expectations: Where CS Students Fall Short in the Software Industry" illustrating a perceived lack of essential skills and the need for further specialized training, as the author highlights recommendations to educators and graduates. We continue the discussion with Nary Subramanian's article entitled "Challenges in Academia in Producing Prepared IT Workforce" discussing the difficulty that academia faces with allotments for specific technical coursework associated with traditional computer and information technology degrees. Nary explores a perceived adequacy and potential misalignment of coursework within college curriculums and suggests possible solutions.

Another pressing issue that frequently dominates nightly newscasts is the emerging success of cyber-attacks within both domestic and governmental systems. It is no surprise that securing our systems from intrusions and vulnerabilities could not be pressing. To that end, we have an excellent article from Commander Michael Bilzor entitled "Seeking Balance in Cyber Education" which discusses the need to balance and maximize the potential of the education provided to our future technologists who will be protecting and safeguarding assets against malicious intent.

Those of you that work within the DoD need no introduction to the Defense Acquisition University (DAU) and the educational role this institution provides to government professionals. The article entitled "Training the Department of Defense Software Acquisition Professional" examines the current state of education provided to the software acquisition cadre and the discusses the expansion of software within all DoD systems and career fields and how the University will attempt to address these future needs. Likewise, another software profession of critical importance is that of the Program Manager. Lawrence Peters offers an article entitled "Training Software Project Managers" discussing perceived success criteria and the need to provide this profession with the competencies to overcome potential pitfalls through proper training.

Finally, we turn our attention to the fact that education must be applied by individuals in real-world settings, arranged into groups, working effectively together. Our last two articles address the potential synergy attainable by the cohesion of teams and applying standards to enhance performance. The article entitled "Increase Team Cohesion by Playing Cooperative Video Games" provides us results of a study that explores how collaborative team building activities can contribute to improved performance. While the article entitled "A 'Thinking Framework' to Power Software Development Team Performance" provides us a new comparative software standard applicable to software development teams with insight into why many previous performance improvement efforts may have failed.

As we begin the New Year, we are also beginning our 27th year of CrossTalk publication as well. I would like to express my sincere thanks to all of you who have made the continuation and excellence of CrossTalk possible. To our Co-Sponsors, thank you for providing your generous support and active involvement, which makes our continued efforts possible. To the authors, we appreciate your continued loyalty and for sharing such valuable information to the software community. And finally, to our readers, thank you for your contin¬ued subscriptions and readership to which I sincerely hope we continue to exceed your expectations.

From all of us at CrossTalk, we wish you the best for the New Year!

Justin T. Hill
Publisher, CrossTalk

Missed Expectations
Where CS Students Fall Short in the Software Industry

Alex Radermacher, North Dakota State University
Gursimran Walia, North Dakota State University
Dean Knudson, North Dakota State University

Abstract. Graduating computer science students do not always possess the necessary knowledge to succeed in their careers after graduation. Interviews with twenty-three managers and hiring personnel at different companies in the software development industry highlight the struggles that recent graduates face when first starting at those companies. Recent graduates lack essential skills in different areas to pass an interview. Descriptions are provided about these different areas along with recommendations for educators, industry managers, and recent graduates.

Introduction and Background

One of the main goals for colleges and universities is to prepare their students for their future careers and to ensure that they are equipped with the knowledge and skills necessary to succeed after they graduate. The goals of educators in computing fields are no different in this regard. With the growing body of knowledge and vast variety of different jobs available to students, it is not possible to teach them everything that they will need to know for every potential job. However, there is evidence that educators need to do a better job preparing students for the workforce, especially when multiple sources have identified the same gaps in students' education [1, 2, 3].

Historically, there have been several educators who have evaluated how the recommended curriculum [4, 5] or the education that computer science students were receiving compared to the needs of the software industry. In 1996 Byrne, et al. conducted interviews with 16 project managers at Irish software companies to ask about their perceptions of how graduating students met their expectations and how CS education could be improved [6]. Around the same time, Lethbridge surveyed over 100 software developers at different companies about the skill level currently needed for their jobs and where they perceived their skill level to be when they had just graduated from college [7]. In a more recent study, Begel, et al. conducted a case study at Microsoft where they watched newly hired, recent graduates to determine what parts of their job they struggled with and what other difficulties they experienced [8]. A large number of other researchers also reported similar findings [9, 10, 11, 12, 13].

Our students at North Dakota State University (NDSU) are required to take a capstone project course before they can graduate. In this course, students work with industry companies on real-world projects that will be used by those companies [14]. The purpose of the course is to expose students to what software development is like in industry and to help shape their expectations for their future. Because we work with industry companies, we like to keep in touch and ask for feedback about our capstone course. During one of our discussions with a company, they raised concerns with us about some of the recent graduates from our university who had applied for jobs at their company. Specifically, they had mentioned that the students who had applied had no experience with regression testing and struggled to write unit tests for a small piece of code during the interview.

We viewed this as an opportunity to improve our course and wanted to see if some of the other companies that we had worked with in the past were experiencing the same problem. We also wanted to determine to what extent other researchers were reporting this problem and to see if there were any commonalities in their findings. To do so, we conducted a review of the literature in order to determine what areas, if any, were commonly reported as areas where recent graduates fell short of industry expectations [15]. We found multiple studies that examined this problem and that there were several areas (spanning everything from software tools to problem solving ability to personal skills and communication ability) that were reported more frequently than others. Collectively, we refer to these different areas as knowledge deficiencies.

Although the literature review was helpful in determining which knowledge deficiencies were most prevalent, much of the prior literature contained little or no descriptive information about the knowledge deficiencies. In order to gain a better understanding of knowledge deficiencies, we decided to focus our interviews on better understanding how recent graduates struggle. We spoke with managers and hiring personal at different software companies that we had worked with previously. Twenty-three respondents (20 from the United States and 3 from Europe) provided information about areas where new hires struggled, along with which knowledge deficiencies that specifically prevented recent graduates from being hired by the company. Before describing the results of those interviews, we will first briefly describe the study design.

1. Study Design:

The following sub-sections provide a short description of the purpose of our research as well as the methodology that was used to conduct our study.

Research Goals

As an initial step, we identified a list of goals to accomplish. These were: identify the areas where recent graduates most frequently struggle when starting the first job; identify any common shortcomings that prevent recent graduates from being hired; and determine what issues hiring personnel screen for in interviews, but is still common in newly hired, recent graduates.

Study Subjects

The participants in this study were 23 managers or hiring personnel from software development companies predominantly located in the United States, although 3 participants were from Europe. The companies were involved in many different business areas (e.g., aviation, agriculture etc.) and ranged in size (from tens of employees to thousands). Most of the participants had previously worked with in some capacity, usually as a sponsor for one of our capstone projects.

Study Instrument

We used a semi-structured interview as it provided a good balance between the opportunity for participants to provide

information about knowledge deficiencies and for us to ask follow-up questions to get additional details about the knowledge deficiencies that were identified. Hand-written notes were taken by the primary researcher during the interviews, and all interviews were conducted in English. Interview participants were asked two primary questions during the interview. The first was whether or not there were any knowledge deficiencies that prevented them from hiring a recent graduate. The second was in which areas that newly hired recent graduates struggled, the company felt as though the person's college education should have better prepared them.

2. Results

Following the interviews, responses were grouped into related categories. Table I contains a list of the most frequently identified knowledge deficiencies and whether these were things that companies identified during interviews to weed-out candidates or whether they were only identified after a recent graduate began working for the company. In some cases (e.g. oral communication, testing, and problem solving) the company identified it as something they looked out for in interviews, but still experienced in their new hires. In those cases, it is counted in both categories, but is only counted once in the total.

Knowledge Deficiency	Interview	Job	Total
Software Tools	2	16	16
Project Experience	13	0	13
Oral Communication	9	2	10
Problem Solving	8	3	10
Work Expectations	0	8	8
Testing	2	6	7
Databases	1	6	7
Teamwork	2	5	7
Working with Customers	0	7	7
Written Communication	1	6	6
Coding Practices	4	2	6
Passion for Technology and Work	4	2	6
Ability to See "The Big Picture"	4	2	6

Table 1. Knowledge Deficiencies in Recently Graduated Students

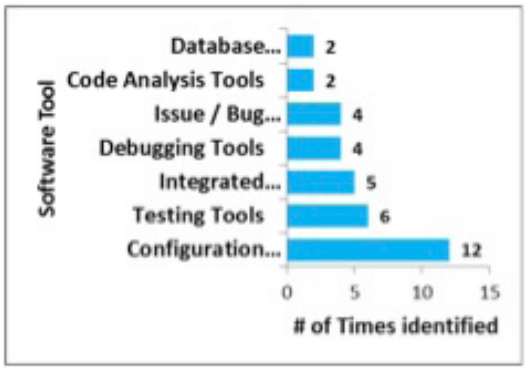


Figure 1. Knowledge Deficiencies for Software Tools

Software tools of some type were reported by 16 of 23 people we interviewed. Fig. 1 breaks down the different types of software tools that were identified. Although interviewees mostly indicated a specific software tool (e.g. Jenkins, Subversion, etc.) the results are categorized based on the type of tool. Although some of the managers we interviewed indicated that they asked about tool knowledge during interviews, the limitations of interviews typically prevent a good assessment of a person's abilities in those regards, and as such knowledge deficiencies related to tool usage were typically identified only after a recent graduate began working for the company.

3. Discussion: Areas Where Recent Graduates Frequently Struggle

Software Tools

A common response was that recent graduates did not have previous exposure to many software development tools. Configuration management tools, in particular, were identified by over half of the respondents. Another common response was that students lacked experience setting up and using tools in a manner similar to an industry production environment. This was particularly the case for Integrated Developer Environments and other applications in the company's toolchain. One respondent reported that it could take up to six months for new employees to reach the same degree of proficiency as other employees. Other participants indicated that recent graduates only had a minimal understanding of tools and one indicated that most had never used version control software beyond committing code to a repository and were incapable of merging or branching. Multiple interviewees indicated that most students did not have any exposure to continuous integration and regressing testing software such as Jenkins or Team Foundation Server. Two respondents reported that recent graduates lacked proficiency with database management tools, one of whom said that students could not even set up a new database.

Job Expectations

A lack of understanding of job expectations was the second most common problem that recent graduates were reported to experience on the job. One aspect of this noted by one interviewee was that newly hired, recent graduates seemed to be afraid of asking questions so as not to appear foolish. Instead, they needed to understand that it was better to ask for help than to waste time being stuck on what might be a simple problem. Another response was that students experienced difficulties adapting to an eight hour work day after college where their work was split up over smaller segments. Another related response was that many recent graduates lacked motivation and initiative and needed closer supervision in order to ensure that they remained productive. Multiple participants also indicated that recently graduated students were lacking in professionalism, including dressing inappropriately, texting on their phones during meetings, or even issues related to personal hygiene.

Communication

Written communication, oral communication, and specifically the ability to communicate with customers were all identified by multiple respondents. One identified aspect of written communication was that recent graduates tend to have problems writing large memos or documents. In terms of communicating effectively with customers, one respondent indicated that

recent graduates tend to use too much technical jargon which impacted their ability to gather requirements. Multiple interviewees indicated that recent graduates had issues communicating with their bosses, such as informing their boss when they had completed an assigned task. Five respondents reported issues related to recent graduates communicating with team members. One specifically reported that some recent graduates struggled with successfully integrating with the rest of the team because they had problems working with other people.

Software Testing

Five of the 6 interview respondents told us that the largest problem related to testing was the inability of recent graduates to construct good unit tests. One interviewee noted that recent graduates had a tendency to write redundant test cases and that while testing, they had a tendency to introduce old or further bugs into the software. Another responded that they had to spend a considerable amount of time training recent graduates on using their test platform and writing good unit tests. There were also four other respondents who had not identified problems related to testing itself, but had indicated that recent graduates lacked exposure to continuous integration and regression testing tools.

4. Areas Where Recent Graduates Fail in Interviews:

Project Experience

A lack of project experience was the most frequently identified problem that respondents indicated that recent graduates expressed in interviews. Most of the interviewed managers indicated that they wanted applicants to have some kind of experience working on a large, team project and for the applicant to be able to describe their role with in the project and how it was important to the project. Interviewees indicated that they realized that recent graduates would not have a lot of experience, but that having an internship, a co-op, or a senior-level capstone project was typically sufficient. One interviewee indicated that it was preferred if the applicant had worked individually on a large project, but the others preferred team-based projects.

Oral Communication

Soft skills are generally cited as important, but one manager indicated that communication ability was even more important than technical abilities. Another stated that while could be overlooked if an applicant had other good qualities, that the ability of such a person to move up through the company would be extremely limited. Several respondents stated that an inability to articulate points and provide clear explanations during interviews were what caused most recent graduates to be passed up during the interview process.

> **"There were five categories in which the knowledge deficiency was identified in newly hired recent graduates, even after the company made an effort to identify them during the interview process: oral communication, written communication, testing, problem solving, and the ability to self-manage."**

Databases

One respondent reported that recent graduates had a poor understanding of the internal mechanics of databases, which was difficult to pick up on the job and lead to unoptimized designs and poor performance. Two other responses indicated that recent graduates had difficulty interacting with databases and tying in the code meant to interface with the database, with one noting that they often had difficulties even creating and designing databases. Two responses pointed to inexperience with database management system (DBMS) tools, and one of those respondents stated that recent graduates lacked even the most basic knowledge such as setting up a new database.

Coding Practices

Two interviewed managers stated that comments from new hires often tend to be of poor quality, if not entirely useless. They noted that comments often offered no actual explanation of what a segment of code was doing or that the comments were unnecessary because they were obvious from the code itself. Another two respondents said that recent graduates often did not comment nearly enough and occasionally not at all. One indicated that recent graduates were not familiar with following a coding standard and frequently produced code that was not to the company's standard.

Problem Solving

Several interviewees indicated that a lack of problem solving ability in a candidate was the most likely reason not to hire that person. One interviewee mentioned that it was common to give applicants small programming problems (e.g. develop an algorithm for searching a tree depth-first) to evaluate their problem solving ability. Another manager indicated that knowing the answer to a problem was not always as important as demonstrating a good approach to attempting to solve the problem. One interviewee indicated that they typically asked increasingly difficult questions in order to evaluate an applicant's problem solving skills. Another stated that they specifically looked for whether or not an applicant would ask follow-up questions or clear up ambiguities in the problems.

Personal Qualities

When faced with several good applicants, some interviewees said that they try to select candidates based on other personal qualities. Several different qualities were given (e.g. leadership ability, being proactive, outgoing and friendly personality), but the most common were an ability to see the big picture and being passionate about technology or the position. One interviewee indicated that

it was important to find candidates who were passionate because even if they were not the most skilled individuals, they would likely make a strong effort to improve. Most respondents indicated that these characteristics were not strictly necessary, but were important factors when determining who to hire and in several cases they made more of a difference than technical ability.

Technical Abilities

Testing ability, database knowledge, and mobile development experience were all reported by interviewees as being necessary skills. One of the interviewed managers indicated that having experience with designing and interfacing with databases was necessary due to the nature of the company's work. Another respondent indicated that as part of the interview process, applicants were asked questions about how they would test various systems and were expected to be able to write unit tests for a small piece of code as that would be the primary focus of their job when starting out. Respondents from two companies indicated that mobile development experience was increasingly important to them as much of their company's new work revolved around developing applications for tablets and smart phones. Another respondent stated that they asked applicants questions about several Linux commands in order to ensure that they would have the technical abilities necessary for working at the company.

Overlap Between Interviews and Jobs

Most of the managers with whom we spoke indicated that they normally did a good job of screening out candidates who were lacking in the areas that they evaluated candidates during interviews. Table 1 shows a rather strict dichotomy between knowledge deficiencies identified during interviews and those identified only after a recent graduate has begun working at the company. However, there were five categories in which the knowledge deficiency was identified in newly hired recent graduates, even after the company made an effort to identify them during the interview process: oral communication, written communication, testing, problem solving, and the ability to self-manage.

5. Conclusion and Recommendations

Several researchers have previously identified various issues that recently graduated computer science students struggle with when starting their new jobs [1, 2, 16]. Our results provide additional support for the existence of several of these knowledge deficiencies including the use of configuration management tools, communication skills, and testing ability. The results from our interviews also provide additional qualitative information for knowledge deficiency categories which were not always well-defined in previous studies.

The results from our interviews also match several of the results that we had uncovered in the literature [15]. This suggests that several of these knowledge deficiencies have been pervasive for some time and that this is an issue that needs to be addressed. Although we do not have solutions for addressing each and every knowledge deficiency, merely being aware of the most pressing issues provides a good starting point for tackling them. However, we do have some general recommendations for academia, industry, and recent graduates.

Recommendations for Academia:

Although it is unreasonable to expect educators to produce students who will be fully capable of starting any job without further issue, there are multiple areas that are in obvious need of some attention. The most glaring issue relates to the use of software tools, particularly configuration management tools. Many of these tools could be incorporated into appropriate existing courses (e.g., using DBMS tools in a database course, test coverage tools in a testing course, etc.) without much hassle and would provide students with earlier exposure to the types of tools that they can expect to use in throughout their careers.

Another area that appears to be a problem is a lack of ability to write unit tests. This is something that can be introduced in early programming courses and reinforced throughout the curriculum. The same also holds true for building communication skills, whether that means students' collaborative skills by including assignments or projects where they work as part of a group, or their writing skills by requiring them to write a technical report in addition to the code that they produce. A required internship, industry co-op, or a projects course where students work with real customers would also help them to acquire a better understanding of the work expectations of industry and provide them with valuable project experience.

Recommendations for Industry:

From our results, it appears as though most managers and hiring personnel can reasonably screen candidates, but there is also room for improvement. Asking applicants to actually write code at a computer or demonstrate their knowledge of software tools may help to alleviate some of the difficulties experienced from hiring recent graduates who lack the necessary skills to contribute to the company. It may also be useful to give applicants a difficult problem that you do not expect them to be able to solve, or one that does not contain sufficient information in order to gauge how they respond to such situations.

Another solution is to identify the areas within your own companies where newly hired, recent graduates struggle and to develop an orientation program that is specifically designed to tackle those problems, whether they are related to maintaining professional conduct or developing training material for the different software tools used at the company so that new employees have a quick reference while first learning to work with a new or different tool.

Recommendations for Recent Graduates:

The field of computing is growing every year as new technologies and ways of accessing information are developed. Even developers who have been in the industry for decades constantly need to acquire new skills and learn about new technologies. It is unlikely that your education will have provided you with everything you will need to know, hopefully it has equip you to continue learning new skills and to refine the ones you already possess.

This research can serve as a good guide for preparing for interviews and for brushing up on in any of the identified areas where you might feel unprepared. It is also good to bear in mind that when you are first starting, you will make mistakes and run into problems that you will struggle to solve. Keep in mind that sometimes it is better to ask for help than to struggle in silence.

ABOUT THE AUTHORS

Alex Radermacher is a software engineering Ph.D. student at North Dakota State University. His research interests include software development processes and student learning.

Phone: 701-367-8965
E-mail: alex.radermacher@ndsu.edu

Gursimran S. Walia is an assistant professor of Computer Science at North Dakota State University. His main research interests include empirical software engineering, human factors in software engineering, and software quality. He is a member of the IEEE Computer Society.

Phone: 701-231-8185
Email: gursimran.walia@ndsu.edu

Dean Knudson is an associate professor of Computer Science at North Dakota State University. He has over 35 years' experience in software development and management.

Phone: 701-231-5612
E-mail: dean.knudson@ndsu.edu

REFERENCES

1. Begel, Andrew and Simon, Beth. Novice Software Developers, All Over Again. Proc. of the 4th International Workshop on Computing Education Research. Sydney, 2008.
2. Lethbridge, Timothy. "Priorities for the Education and Training of Software Engineers." 53.1 Journal of Systems and Software (July 2000): 53-71.
3. Radermacher, Alex, et al. Investigating the Skill Gap between Graduating Students and Industry Expectations. Proc. of the 36th International Conference on Software Engineering. Hyderabad, 20014.
4. The Joint Task Force on Computing Curricula. "Computing Curricula 1991." 4.6 Communications of the ACM. (June 1991): 68-84.
5. The Joint Task Force on Computing Curricula. "Computing Curricula 2001." 1.3 Journal of Educational Resources in Computing (Sept. 2001): 1-236.
6. Byrne, Declan J. and Moore, Jeffrey L. "A Comparison Between the Recommendations of Computing Curriculum 1991 and the Views of Software Development Managers in Ireland." 28.3 Computer Education (April 1997): 145-154.
7. Lethbridge, Timothy. A Survey of the Relevance of Computer Science and Software Engineering Education. Proc. of the 11th Conference on Software Engineering Education and Training. Atlanta, 1998.
8. Begel, Andrew and Simon, Beth. Struggles of New College Graduates in Their First Software Development Job. Proc. of the 39th SIGCSE Technical Symposium on Computer Science Education. Portland, 2008.
9. Carver, Jeffrey and Kraft, Nicholas. Evaluating the Testing Ability of Senior-Level Computer Science Students. Proc. 24th IEEE-CS Conference on Software Engineering Education and Training. Honolulu, 2011.
10. Loftus, Chris, et al. Can Graduating Students Design: Revisited. Proc. of the 42nd ACM Technical Symposium on Computer Science Education. Dallas, 2011.
11. McGill, Monica. Defining the Expectation Gap: A Comparison of Industry Needs and Existing Game Development Curriculum. Proc. of the 4th International Conference on Foundations of Digital Games. Orlando, 2009.
12. Scott, Elsje, et al. The Skills Gap Observed between IS Graduates and the Systems Development Industry - A South African Experience. Proc. of the 3rd Informing Science and IT Education Conference. Cork, 2002.
13. Surakka, Sami. What Subjects and Skills are Important for Software Developers? 50.1 Communications of the ACM (Jan. 2007): 73-78.
14. Knudson, Dean and Radermacher, Alex. Software Engineering and Project Management in CS Projects vs. "Real-World" Projects: A Case Study. Proc. of the 9th International Conference on Software Engineering and Applications. Cambridge, 2009.
15. Radermacher, Alex and Walia, Gursimran. Gaps between Industry Expectations and the Abilities of Graduates. Proc. of the 44th ACM Technical Symposium on Computer Science Education. Denver, 2013.
16. Tesch, Debbie, et al. "An Examination of Employers' Perceptions and Expectations of IS Entry-Level Personal and Interpersonal Skills." 6.1 Information Systems Education Journal (Jan. 2008): 3-16.

Challenges in Academia in Producing Prepared IT Workforce

Nary Subramanian, University of Texas at Tyler

Abstract. A frequent question practitioner asks an academician is "Why don't educational institutions prepare students better for work in the IT area?" Often cited problems with current graduates from Computer Science, Computer Information Systems, and other IT disciplines, include lack of awareness of latest developments in the field, lack of knowledge of applying systematic process to solving problems, and a general unpreparedness to cooperatively work in teams. These issues are pertinent in an increasingly interconnected world where new technologies are emerging at a fast pace: big data analytics, computer security, cloud computing, and mobile computing, are but a few of the latest developments. While it may be appropriate for industry to demand properly prepared students who are work-ready from day one, it may be an eye-opening experience for most practitioners to know the problems faced on the other side of the fence, namely, at educational institutions. In Texas, for example, less than 25% students graduate in four years and less than 50% graduate in six. Typical course-load for a degree in Computer Science area is about 120 hours with about 60 hours dedicated to state mandated, college- and department-specific core courses; this leaves about 60 hours (roughly, 20 courses) to complete the discipline-specific requirements. If we consider the normal course sequence in the degree plan consisting of traditional courses such as programming, software development, and database design at the most only about four courses are left in the plan that can be considered for electives. Assuming students are ready to take an elective in a topic such as SCADA security, the next problem is the lack of appropriate labs and textbooks to teach such courses at the undergraduate level. Another issue that needs to be considered is the availability of faculty members to teach new courses in a manner that not only retains students' interest but also increases enrollment. Therefore, producing well-educated students who are workforce-ready for cutting-edge technology companies within four years of college study is proving to be an increasingly tougher goal for the academia. This article considers this issue and suggests possible solutions to this perceived problem.

Introduction

Undergraduate education builds a nation's workforce. This is especially true in IT related fields such as Computer Science (CS), Computer Information Systems (CIS), Information Technology (IT) or Management Information System (MIS). Students graduating with B.S. are employed by (and increasingly are also employing) leading technology companies such as Google, Apple, Microsoft, public, and government agencies. They form the foundation that helps develop cutting-edge technologies that require and provide jobs to millions of people around the globe. In fact, CEO's of several technology firms are graduates from four-year degree programs such as, for example, Jeff Bezos of Amazon, Mark Zuckerberg of Facebook, and Tim Cook of Apple [1].

However, recently I have been asked by practitioners at several workshops and conferences, "Why do not educational institutions prepare students better for work in the IT area?" In a recent survey by CompTIA, 93% of employers indicated there exists a skills gap in the IT workforce that included both hard and soft skills - lack of knowledge of latest technologies, poor problem solving ability, and inability to work cooperatively in teams [2]. Industry people expect, perhaps rightly so, that products of four-year degree programs from Universities be ready to contribute effectively right from day one. However, considering the dynamic growth of the IT industry, many employers seem to have been disappointed in this expectation as indicated by the survey [2]. These issues are relevant in an increasingly interconnected world where new technologies are emerging at a fast pace: big data analytics, cloud computing, game development, social networking, and mobile computing, are but a few of the latest developments.

In this article, I wish to point out the issues on the "other" side: the academic side, a view that is usually not very clear from the outside world. Major reasons that I consider contributing to this observation from industry practitioners can be categorized in three headings: poor four-year graduation rate, low discipline-specific course load, and lack of academic infrastructure for modern electives. I also suggest some remedial measures that can help improve situation many of which require industry participation. In this article I discuss mainly from the CS perspective though similar arguments may be made for other computing disciplines.

Poor 4-year Graduation Rate

Typical four-year graduation rates are low in US universities. As per the National Center for Education Statistics [3], for the year 2006 cohort, only 39% graduated within four years while only about 59% graduated within six years. However, data varies from state to state: in Texas, for example, the four-year graduation rate is less than 25% while six-year graduation rate is less than 50%. In engineering disciplines, the graduation rates are slightly higher at about 35% while the six-year rate is about 75% [4, 5]. This means out of every 100 incoming freshmen only about 35 are available for industries to hire at the end of four years. In fact, in 2011 about 11,000 undergraduate CS degrees were awarded in the US [6] and given a growth in enrollment of about 10% per year (based on [6]), I estimate the number of students in the incoming freshmen cohort for CS should have been about 26,000 in 2007 – about 42% four-year graduation rate. This means more than half the enrolled students failed to complete their degrees within the expected period of four years. There are several reasons for this: changed majors, funding problems, family issues, job issues, lack of pre-requisites, or simply failed classes.

What does this mean for an employer? When students do not complete the curriculum within the expected duration, they graduate with obsolete skills: they may have forgotten concepts learnt during the sophomore Data Structures class or even the freshman programming class (for example, in the survey [2], employers state that one of the reasons for the skills gap is the fast pace of change in the IT industry). In junior and senior classes, a student graduating late may not remember the basics studied in the pre-requisite class some years earlier – this requires the student to study harder to keep up with the class. Therefore, the employer is likely to find a student taking longer to graduate not as good as a student who graduates on time. In fact one study has concluded that students graduating late tend to take the job they get – not necessarily the job they want! [7]

Low Discipline-specific Course Load

Typical course-load for a four-year CS program in Texas (for example, [8]) includes up to 48 hours of state-mandated core, 42 - 51 hours of required courses, 9 - 12 hours of CS electives, and 21-28 hours of other electives and pre-requisites, for a total of 120 - 130 credit hours of courses. This means that only about 60 hours (or roughly, 20 three-hour courses) are dedicated to CS in which programming fundamentals, algorithms, computer organization, operating systems, database management systems, networking, and software development all need to be fit in a proper sequence over the period of four years. CS electives are usually department-specific and application oriented: they include options that cover latest developments such as computer security, e-commerce, mobile programming, bio-informatics, and the like. However, for some of these electives students usually do not have the necessary pre-requisites ("pre-reqs") and it becomes their responsibility to learn the pre-req along with the course material. For example, for a decent Computer Security class advanced math helps, for e-commerce advanced database concepts help, for iOS programming knowledge of Objective C (or, now, Swift) helps, and for bio-informatics a working knowledge of genetics helps. If such pre-reqs could be somehow built into the main CS curriculum, then students are more likely to graduate with a deeper understanding of the subject and its application areas (of course, students may optionally pursue a Master's degree – however, there are cost and time penalties, and this is discussed later).

In this regard it should be pointed out that there is an upper limit on the number of credit hours a student can take and still pay the lower in-state tuition fees. For example, in Texas, an undergraduate student enrolled after 2006 can take only 30 additional hours beyond what is required for degree completion and still pay tuition at the resident rate; any hours taken in excess of these 30 hours will attract fees at the non-resident rate, which is, in many cases, at least twice that of the resident rate. This rule prompts students to complete their degrees as quickly as possible in terms of credit hours – therefore taking electives becomes an expensive option. These excess hours include any courses repeated due to failing grades or for grade replacement.

Another factor to remember is that not all electives are offered all the time. Based on departmental requirements such as faculty availability, course schedules, and student enrollment, some electives may not be offered on a regular basis. Also, if an elective is offered but there is not sufficient student enrollment, which is usually between 10 and 15 students based on the level of the course, the course may be cancelled at the discretion of the department. So electives may not be the right way to expect students to acquire knowledge of the latest developments in CS.

Lack of Academic Infrastructure for Cutting-edge Electives

I have been asked by industry practitioners as to why we are not teaching distributed control systems and SCADA (supervisory control and data acquisition systems) to our students. The control system industry has become extensively computerized and networked with several overlapping fields of knowledge and finding a student trained in these concepts has become increasingly difficult. SCADA is but one such emerging area; there are others such as cloud computing, big data analytics, and health informatics, where deep interdisciplinary knowledge is required of graduates. In fact, emerging industry paradigm seems to be the convergence of IT and OT (operations technology) – OT is the technology used for the core business processes including manufacturing, customer-service, and product development. Unfortunately, there are several problems that academia faces when trying to incorporate such modern industry requirements into the curriculum.

First of all, we need excellent laboratory facilities for teaching courses in these areas. Simulators may be used for teaching cloud computing concepts and Hadoop has been used to teach big-data analysis: for proper operation of both these systems we need trained IT personnel who can maintain both the hardware and software for optimum use of such systems. However, for SCADA we need expensive hardware/software kits and well trained lab administrators to run such systems (for example, we have such a lab at UT Tyler [9]). Likewise, for health informatics we need to be able to simulate or emulate confidential health records and their use in medical practice.

Second problem relates to useful academic textbooks in these areas (this was a problem while teaching mobile computing in its early days as well [10]). While several industry references exist, not all of these can be easily used in the classroom since most such text books assume minimum knowledge in their readers which is always not the case. In SCADA design, for example, ladder diagrams are used by professionals but most academic curriculum in CS and Electrical Engineering have never taught these concepts for last many years!

Thirdly, we need well-trained faculty members to teach such courses. While industry practitioners may be called for guest lectures, it may not be fair to expect them to also teach such classes with the rigor required. With the availability of free webinars conducted by industry experts as well as MOOC (Massive Open Online Course) classes that are conducted by both academic and industry experts, current faculty could use these resources to prepare themselves to teach modern courses. Another option we considered at UT Tyler was the Professor-in-Residence where a faculty member spends a month or more in the summer working at a local company. Professor-in-Residence could be voluntary or paid; however, this experience will help reduce the barrier between industry and academia and hopefully prepare the faculty to become a better teacher.

Some Suggestions for Improving the Situation

One thought that may arise in the minds of the readers is the need for the state mandated core in CS curriculum – that is, can we not use the time spent on the core on CS subjects? In fact this is the approach followed in many other parts of the world where bulk (90% or more) of the CS curriculum consists of only CS-related courses. However, it is widely acknowledged that exposure to the humanities, arts, and social sciences is essential to improving soft skills such as communication with peers, respect and understanding for other points of view, and being creative and innovative at work [11]. Therefore, the liberal arts core is essential to developing a well-rounded graduate in CS.

There are other possibilities to improve student's learning experience within the four-year curriculum and they are discussed below.

Internship and Cooperative Experience

Many universities actively encourage their students to participate in internship and cooperative programs. Such programs involve students working for a company and learning specific skills that complement their academic experience. Examples include website development, single sign-on using active directory, SAP configuration, system administration, firewall configuration, programming using C#, Java, and C++, and the like. Internship and cooperative programs may be done concurrently with other courses during Fall and Spring semesters or exclusively during summer or even during regular semesters. Several programs are paid and may be located far from the parent University and, very often, also give academic credits to the students. In fact, some Universities have student services departments that provide active guidance to students for such programs. Thus, students participating in internship and cooperative experiences are usually more knowledgeable about the work environment in IT, perform better in class, and are usually more prepared to enter the workforce after graduation.

Meaningful Capstone Projects

Several CS programs require their seniors to participate in a capstone project. Typically this provides an opportunity to experience almost first-hand the typical IT job environment. If these capstone projects are conducted in their true spirit, many valuable job skills such as project management and collaborative skills can be learnt from them. Such projects become more meaningful if they are conducted with industry partners since the latter can not only mentor student groups but also possibly provide them employment or at least references upon graduation. At UT Tyler, we have been conducting these projects with interdisciplinary teams composed of both CS and CIS students where each team does a significant project for a local industry partner. Each team manages its project and works cooperatively to achieve objectives of the industry client. We found that this experience has helped develop well-rounded graduates with skills beyond that taught in the classroom that can be readily applied in practice. However, this requires close collaboration between educational institutions and industry that in turn requires open-minded faculty members and industry partners willing to appreciate each other's capabilities and constraints.

Industry Sponsorship of Education

Another possibility that exists is industry sponsoring their employees for undergraduate education on either full-time or part-time basis. Armed forces provides this option as does National Security Agency [12] though in the latter model the student studies at a University of his or her choice with a commitment to work at NSA subsequent to graduation. Many private employers have similar education sponsorship scheme for their current employees; perhaps they can follow the NSA model and hire incoming freshmen with the commitment that students work at the company during summers and join the company upon graduation. Such a scheme will not only produce well-educated employees but also employees who are already knowledgeable in the skills and technology that the company requires.

Another way in which industry can get latest tools in the hands of students is to sponsor equipment for labs – hardware and software that companies make can be supplied to labs in colleges and universities. Faculty members can integrate such equipment into appropriate coursework. For example, SPEA America donated semiconductor-testing equipment to UT Tyler [13] and this donation helped establish a center that benefitted both faculty and students.

Increase Course Load with MOOC/online Classes

Another option is to increase the number of courses required to obtain a CS major degree – in fact, in many other countries the liberal arts core is not required and the curriculum consists mostly of CS courses. But here we hit another hurdle – sometimes the maximum course loads are mandated by the state. For example, in Texas, 120-hour limit has been imposed by the state and at UT Tyler we had to redesign our curriculum from 127 hours a few years back to 120 hours now. However, there are many programs that are ABET accredited [14] which mandates them to satisfy minimum set of requirements – such CS programs are given exemptions and usually have 6-10 hours more than the state mandate. However, industry has not always insisted on ABET accreditation for CS programs (in fact, there are only 18 ABET accredited CS programs in Texas [15] out of more than 100 programs offering this degree) and the extra expense on the additional courses could be a burden for many students.

Expenses for the additional courses can be offset by taking online courses offered by several junior colleges and universities. There are also MOOC courses from sites such as Coursera [16] and Udacity [17] that students can enroll for free and improve their skillset and knowledge. Such courses may be taken in the summer months when students are not enrolled in normal classes. Universities are exploring ways to provide credit for such online coursework, although much work still remains to be done to make this process smooth and reliable.

Summer Research Experience for Undergraduates

A valuable opportunity exists in the summer months for exposing undergraduate students to the latest developments in the field of CS – the summer research experience program conducted by several universities often funded by major federal agencies. During this summer program, undergraduate students work alongside graduate students on cutting-edge research under the supervision of a faculty member. Such programs can serve as platforms for introducing, analyzing, and evaluating latest developments in the field and getting students interested in them. Sometimes industry partners are also involved in such research programs in which case students are exposed to the latest developments in both academia and practice. Such research experience could lead to co-authored papers, conference attendance, and patent applications, all of which serve to widen the knowledge horizon of students.

Study Abroad

Another option that students at US Universities have to expand their horizon is the study abroad programs. Study abroad programs offer students the opportunity to study a subject of

their choice at an educational institution abroad and get credit for those courses transferred to their primary institution in the US. Students opting to study in non-English speaking countries either already know or learn the language before going abroad. However, students going abroad typically return with a broader understanding of the world and tend to be more understanding employees. Also, some universities abroad offer specialized courses that can be taken by study abroad program students and many of the courses may be offered in English itself. However, study abroad experience may be intensive in summers, can be expensive, and not all credits may be transferred to the parent University.

Encouraging Graduate Education

Currently very few of undergraduates go on to join grad school immediately after graduation – one estimate puts this number between 10% and 30% [18]. By joining grad school for earning an M.S. or Ph.D. degrees, students get to specialize in a field of their choice and become more useful to work in the industry. In fact graduate education is considered key to improve US competitiveness and innovation [19]. However, there are many issues to consider: typically this means students have to pay more to obtain a graduate degree; in many cases, graduate degree does not automatically mean better pay; students lose out on time to obtain valuable experience; sometimes industry does not require PhD or even MS students. For some grad programs field or work experience is preferred – such as for example, cyber forensics. Therefore, while grad programs may not be the first choice for many graduating seniors this may provide an avenue for some of them to acquire advanced knowledge required by the IT industry.

Closer Industry-Academia Relationship

At national and professional levels industry already has a say in academia through ABET accreditation and Professional Engineer (PE) exams [20]. ABET evaluates computing disciplines (CS, CIS, and IT) through its Computing Accreditation Commission (CAC) and if the programs satisfy minimum requirements they receive ABET accreditation. PE exam has been started recently for software engineering that assures the hiring company that the person possessing this certification satisfies minimum industry requirements. Therefore, industry input already has an impact on curriculum. However, industry practitioners can participate even more in the functioning of local colleges and universities by volunteering to become a member Industry Advisory Boards that most computing departments institute, attending senior design/capstone project presentations and providing feedback, delivering guest lectures, inviting faculty and students to visit their offices/factories, participating in career fairs, visiting colleges/universities on open days, and the like. Such involvement will not only provide an insight into the problems, if any, faced by local academia but also provides opportunities to interact with administration officials, faculty members, and students. These sessions can be used to observe, advise, and recommend to colleges/universities ideas for possible improvements.

Another form of industry-academia partnership is service-learning (SL) [21]. SL is an experiential learning wherein students learn by performing some service to a community client as part of course requirements and their learning is assessed by the instructor by means of reflection assignments. Within IT field, several of the initiatives suggested earlier including internships, cooperative education, capstone projects, and summer research experience could all be considered examples of SL. In order for SL to be more widely adopted in computing programs, industry may need to be more closely involved – for example, database classes may do a project for a company and networking classes may actually participate in creating new or troubleshooting existing networks in a company. At UT Tyler we have established the Center for Teaching Excellence and Innovation where SL is an essential component [22].

Conclusion

Industry seems to find several graduating seniors in the computing sciences field lacking in skills that will make them immediately productive in the industry. Typical problem areas seem to be the lack of awareness of latest developments in the field, lack of knowledge of applying a systematic process to solve problems, and a general inability to work cooperatively in teams. This article has explored this issue from an academic standpoint and discusses the constraints faced by the academic community in preparing its undergraduate students – these constraints can be categorized under poor four-year graduation rate, low discipline-specific course load, and lack of academic infrastructure for modern electives. These categories reflect the fact that only about 42% of freshmen graduate from computing programs in four years, that there are only about 20 courses or less in which to teach the core topics in CS, and that there are not enough labs, books, and faculty members to teach modern electives.

There are several avenues for improving the situation and these include internship and cooperative education experiences, meaningful capstone projects, industry sponsorship of undergraduate education, increase course load with MOOC and online classes, summer research experience for undergraduates, study abroad programs, and encouraging graduate education. The article discusses the pros and cons of these options. However, many of these initiatives require a closer cooperation between academia and industry and hopefully this helps reduce the skills-gap that industry seems to face with graduates from four-year programs in the United States.

Another issue that the IT industry faces is sufficient diversity in the workforce, which as per recent media reports, is lacking in terms of women and minority employment [23]. However, this problem exists in academia too, especially in the engineering and computer science disciplines where women and minority participation is low [24]. At UT Tyler, we conduct science camps for high-school girl students in the summer to encourage them to pursue STEM (science, technology, engineering, and math) careers. Another issue to consider is that a significant number (12% in 2012 as per [24]) of undergraduate students in Computer Science are part-time – such students usually take

longer to graduate [25]. Also many part-time students are trying improving their skills in their quest for lifelong learning and it is hoped that industry supports their endeavor [26]. However, it has been this author's experience that part-time students often contribute to a more mature discussion in the classroom.

In this article I have tried to answer the question that has been asked frequently by people in industry: "Why do not educational institutions prepare students better for work in the IT area?" There are several challenges that academia faces in trying to ensure all their "products" satisfy all industry expectations at the time of graduation and I have discussed many of these problems from the point-of-view of an educator and provided some directions in which industry could actively help in remedying this situation. Hopefully, the expectation gap steadily reduces in the future and students in the IT area find the transition to industry more seamless and enjoyable.

Acknowledgements

I thank the reviewer of the original version of this article for providing valuable feedback that helped to significantly improve the quality of this article.

ABOUT THE AUTHOR

Nary (Narayanan) Subramanian is currently an Associate Professor of Computer Science at The University of Texas at Tyler, Tyler, Texas. Dr. Subramanian received his Ph.D. in Computer Science from The University of Texas at Dallas. His specialization is software engineering with particular focus on software architectures and requirements engineering. He co-founded the International Workshop on System/Software Architectures (IWSSA) and served as a co-chair for seven years between 2002 and 2011. He established and directed the Center for Petroleum Security Research at UT Tyler. He is a Fellow of Service Learning at UT Tyler's Center for Teaching Excellence and Innovation. He has over fifteen years' experience in industry in engineering, sales, and management. He is a member of the IEEE. His research interests include software engineering, system engineering, and security engineering.

Phone: 903-566-7309
E-mail: nsubramanian@uttyler.edu

REFERENCES

1. Gene Marks, "Why Do Nerds Become Successful CEOs?", February 10, 2014, available at http://www.forbes.com/sites/quickerbettertech/2014/02/10/why-do-nerds-become-successful-ceos/
2. CompTIA, "State of IT Skills Gap: Full Report", February 2012, available at http://www.wired.com/wp-content/uploads/blogs/wiredenterprise/wp-content/uploads/2012/03/Report_-_CompTIA_IT_Skills_Gap_study_-_Full_Report.sflb_.pdf
3. http://nces.ed.gov/programs/digest/d13/tables/dt13_326.10.asp viewed on August 5th, 2014.
4. http://www.utexas.edu/graduation-rates/documents/GRAD-REPORT.pdf
5. http://dars.tamu.edu/dars/files/ee/ee2a3d0b-38cf-4a0b-addc-847d802aee7b.pdf
6. http://cra.org/uploads/documents/resources/taulbee/CS_Degree_and_Enrollment_Trends_2010-11.pdf
7. Carmen Aina and Giorgia Casalone, "Does time-to-degree matter? The effect of delayed graduation on employment and wages", AlmaLaurea Working Papers, No. 38, September 2011, available at http://www2.almalaurea.it/universita/pubblicazioni/wp/pdf/wp38.pdf
8. https://www.uttyler.edu/catalog/degreeplans/files/COSCBS.pdf
9. https://www.uttyler.edu/cpsr
10. S. K. S. Gupta and P. K. Srimani, "Experience in Teaching a Graduate Course in Mobile Computing", 30th Annual Conference on Frontiers in Education, October, 2000, pp. S1C6-S1C11 (Vol. 2).
11. Charles M. Vest, "Educating Engineers for 2020 and Beyond", National Academy of Engineering, http://engineeringchallenges.org/cms/7126/7639.aspx viewed on August 14th, 2014.
12. http://www.nsa.gov/careers/opportunities_4_u/students/stokes.shtml
13. http://technews.tmcnet.com/business-video/news/2010/10/05/5049852.htm
14. http://www.abet.org
15. http://main.abet.org/aps/Accreditedprogramsearch.aspx
16. http://www.coursera.org
17. http://www.udacity.com
18. http://www.idealist.org/info/GradEducation/Resources/Preparing/ReasonsToWait
19. http://www.cpec.ca.gov/CompleteReports/ExternalDocuments/GR_GradEdAmComp_0407.pdf
20. http://ncees.org/exams/pe-exam/
21. Barbara Jacoby, Service Learning in Higher Education: Concepts and Practices, Jossey-Bass, 1996
22. http://www.uttyler.edu/ctei
23. Elizabeth Weise, "Tech: Where the women and minorities aren't", USA Today, August 15, 2014, available at http://www.usatoday.com/story/tech/2014/05/29/silicon-valley-tech-diversity-hiring-women-minorities/9735713/
24. Brian L. Yoder, "Engineering by the Numbers", available at http://www.asee.org/papers-and-publications/publications/11-47.pdf
25. Jonathan Dame, "Part-time students least likely to graduate on time", USA Today, December 21, 2013, available at http://www.usatoday.com/story/news/nation/2013/12/21/part-time-students-graduation-rate/4156239/
26. David Kember, et al., "How students cope with part-time study", Active Learning in Higher Education, Vol. 6, Issue 3, 2005, pp. 230 – 242.

Seeking Balance in Cyber Education

Commander Michael Bilzor, USN, PhD, U.S. Naval Academy

Abstract. The future cyber warriors of the U.S. and the Department of Defense are being groomed at our nation's universities right now. As they are imbued with the fundamentals of computer network attack and defense, the war rages on in cyberspace. Few questions are more critical to the future of DoD and the nation than how we can most effectively prepare these men and women for their mission. There are pitfalls in gravitating to extremes as we in academia chart their course. In the paragraphs that follow, we advocate for a balanced approach that maximizes educational value, in order to prepare those future cyber warriors for the battles that lay ahead of them.

Theory vs. Application

"A man must know how to choose the mean and avoid the extremes on either side, as far as possible" attributed to Socrates

An important aspect of any educational program is the balance between teaching theory and application. Some refer to this as "training vs. education." If students receive all theory and no application, they will be challenged to apply the theory to contemporary computer systems and networks. If students are only taught specific applications, they will be able to use today's systems, but will have difficulty adapting to new situations, tools, and technologies.

This tension is felt in traditional computer science, as it is in many engineering and scientific disciplines. In the field of cyber security, however, the inherent complexity of systems makes the divergence between theory and application more profound compared to traditional academic disciplines, and it is therefore more challenging to strike a proper educational balance between the two.

To illustrate the contrast, first consider a traditional discipline like material science. Mechanical engineers learn how materials fail by studying and measuring stress and strain, hardness and ductility, etc. They reinforce their theoretical understanding by breaking materials in the lab. When a bridge support or an aircraft bulkhead fails, they can appeal directly to the theory observed in the lab for confirmation.

Now let us consider the security failure of a cyber system. Many of the theoretical underpinnings of secure computer systems were established as early as the 1970s. Take a stroll down memory lane:

- The Bell-LaPadula (BLP) model for multi-level security, first outlined in 1973 [1]. In this model, subjects access objects; both subjects and objects have associated security labels. To preserve confidentiality, access is determined according to three rules: a subject may not read data at a higher security level (no read up, or "simple security"), a subject may not write data to a lower security level (no write down, or the "•-property"), and a subject that can both read and write must do so only at the same security level (the "strong •-property").

Fig 1: Illustration of BLP Formal Security Model

- The Biba model, published in 1975, did for integrity what BLP did for confidentiality [2]. The Biba model associates subjects and objects with integrity labels, with similar rules: a subject cannot write data to an object at a higher integrity level ("no write up"), a subject may not read data from an object at a lower integrity label ("no read down"), and a subject may not request a service from a subject of higher integrity.
- First introduced in 1983, the Kemmerer Shared Resource Matrix Methodology demonstrated the requirements for covert channels to exist in a computer system, and developed code-analysis techniques for identifying them [3].
- The Clark-Wilson model, introduced in 1987, outlined an approach where data integrity is preserved through a well-defined series of transactions [4].

What do these important theoretical security models have in common with the modern, commercially-available operating systems most widely used by DoD? Unfortunately, nothing. Of the vulnerabilities reported on the national vulnerability database for this year, how many refer to incorrect application of one of the formal security models listed above? None [5]. Why not? Because modern, general-purpose, commercially-available operating systems and applications used by DoD, even on classified systems, have not been implemented based on formal security models. The commercial market does not require formal security models, and the modern commercial code base is too complex and too rapidly evolving for this to be practical today.

It is true that some computer systems have been developed with provable security in mind. For example, seL4 is a microkernel whose security properties have been formally verified, but its complexity (8,700 lines of C code and 600 lines of assembler) pales in comparison to that of a full-blown commercial operating system, at tens of millions of lines of code [6]. Even a browser application can run into the millions of lines of code [7], and the overall complexity of a computer system is cumulative in the complexity of its components (software and hardware).

Returning to our example, a security failure of a computer system does not generally reflect a failure in the security theory, or an incorrect application of the security theory. Rather, a security failure of a computer system most often results from human error regarding implementation of a technology specific to that system. Understanding most modern computer security failures requires

not an understanding of traditional theoretical security models, but an understanding of the implementation details particular to a system, whether it is an operating system, a software application, or a piece of hardware. Unlike the mechanical engineer in our earlier example, the computer security engineer is more likely not relying on the basic theory to find the failure in the implementation.

Security theories outlined in BLP, Biba, and other models are covered by most university programs, and in some general certification programs like CISSP, as important background. Many undergraduate and graduate programs in Computer Science (CS), Information Technology (IT), and Information Systems (IS) do treat both the theoretical and applied aspects of security, but they often do so independently, without strongly connecting the two. This occurs in part because, compared to traditional science and engineering fields, theoretical models of computer security diverge further from their actual implementations, in real-world systems. Security is addressed in CS, IT, and IS curricula, but often in a way that leaves theory and application divorced. This leads to the challenge, in academia, of how best to effectively bridge the resulting gap.

Principles and Pillars vs. Applications and Tools

"Fly the Middle Course" – Daedalus to Icarus, in Cretan mythology

If the talented software engineers developing Windows, Linux, and OS X are not deriving their code directly from theoretical security models and applying formal proofs of correctness, at what level of abstraction can cyber security be applied to such systems?

Many experts in the field have enumerated principles -- understandable general statements about properties that can be applied to computer systems, networks, and software. For example, in our Introduction to Cyber Security Course, given to all U.S. Naval Academy midshipmen fourth class (freshmen), we outline three commonly-used principles of cyber defense: the Principle of Least Privilege, Defense in Depth, and Vigilance, and ask the students to implement them in a variety of ways during hands-on labs. [8]

Many organizations frame Cyber Security or Information Assurance in terms of pillars, fundamental properties that must be preserved by a computer network. Three of the most often used are confidentiality, integrity, and availability. Some institutions use only these three pillars. In our course, we add the pillars of authentication and non-repudiation. Others include additional pillars, as well [8,9]. Here again, the educational challenge is the gap between the principles or pillars and the tools and techniques needed to implement them. Without specific instructions explaining how a particular pillar or principal is applied to a specific network, host, or application, students do not find the application intuitively obvious, in our experience.

Alternatively, some organizations define their approach to cyber security at a more granular level, as in the following two examples.

NSA's Information Assurance Division outlines its Top 10 Mitigation Strategies [10]:

- Application Whitelisting
- Control Administrative Privileges
- Limiting Workstation-to-Workstation Communication
- Antivirus File Reputation Services
- Anti-Exploitation
- Host Intrusion Prevention Systems
- Secure Baseline Configuration
- Web Domain Name System Reputation
- Take Advantage of Software Improvements
- Segregate Networks and Functions

The SANS Institute enumerates its 20 Critical Controls as a guide to securing a computer network [11]:

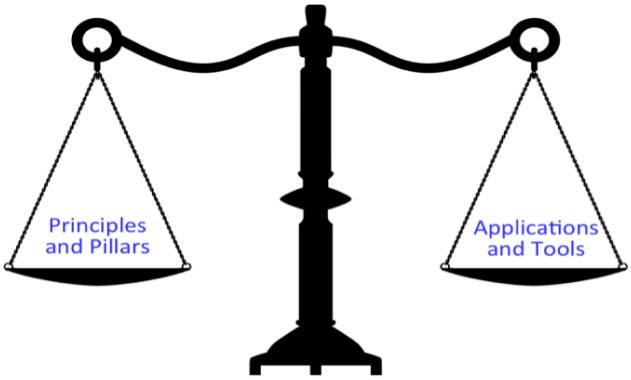
- Inventory of Authorized and Unauthorized Devices
- Inventory of Authorized and Unauthorized Software
- Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers
- Continuous Vulnerability Assessment and Remediation
- Malware Defenses
- Application Software Security
- Wireless Access Control
- Data Recovery Capability
- Security Skills Assessment and Appropriate Training to Fill Gaps
- Secure Configurations for Network Devices such as Firewalls, Routers, and Switches
- Limitation and Control of Network Ports, Protocols, and Services
- Controlled Use of Administrative Privileges
- Boundary Defense
- Maintenance, Monitoring, and Analysis of Audit Logs
- Controlled Access Based on the Need to Know
- Account Monitoring and Control
- Data Protection
- Incident Response and Management
- Secure Network Engineering
- Penetration Tests and Red Team Exercises

Teaching cyber security based only on enumerated lists like these requires teaching technology-specific, application-specific, and even tool-specific content. For example, tools like Microsoft's Active Directory and Group Policy, EMET, and Applocker can be used to support the controls above, but they might not be the principal tools of the trade in 10 or 20 years.

Principles are enduring, but applications and tools tend to change over time. In order to ensure that a cyber security education is not perishable, it should not focus too heavily on the applications and tools. There is a natural aversion in academia

to "teaching the tools", since students may learn the tool but not understand the theory. The solution is not to avoid tools altogether, but to employ them as facilitators of understanding, and connect them to the great principles.

To ensure that an academic program is contemporary and relevant, educators should not omit entirely the cutting-edge tools and technologies of the day. Here again, a balanced approach, containing appropriate quantities of each, may be the most suitable way. In a way, too, enumerated principles and pillars, even if not formally defined themselves (like BLP, Biba, or Clark-Wilson), can act as an informal bridge between enduring security theory and the security tools and practices of the day.



Computer Science vs. Cyber Operations

"I have always sought for the middle ground."
—James Madison

Another important schism in the education of our future cyber warriors is in the relationship between cyber security and the disciplines of computer science and information technology. Some commonality is generally acknowledged, but the degree and nature of the overlap is often debated.

Let's Get Interdisciplinary

A key difference, for many, is the assertion that cyber security is more of an interdisciplinary field of study, compared to traditional computer science or information technology. The Department of Defense, in particular, has acknowledged the impact of cyber-physical systems (CPS) on the future of warfare, which necessarily reaches beyond the traditional computing fields.

There are of course many legal, social, and ethical aspects of cyber security not traditionally covered by computer science degree programs. For example, DoD cyber operators should be familiar with the constructs of Title 10, the section of the U.S. code that clarifies military roles and authorities, and Title 50, which outlines intelligence roles and authorities, since these frequently overlap in the conduct of real-world cyber operations [12]. As another example, it is important to discuss in an educational setting the social, ethical, and legal aspects of insider leaks like the Manning and Snowden incidents, as well as the societal perception of government cyber programs, like those covered in media reports surrounding the Snowden affair [13,14]. In addition, an educational program in cyber operations would be remiss to omit topics like social engineering and activist hacking [15], or "hacktivism."

Let's Get (Cyber) Physical

The classic example of exploiting a cyber vulnerability to conduct a physical attack is of course Stuxnet [16], but there are other examples of how the effects of cyber attacks can be felt in the physical domain:

- In 2000, a disgruntled contractor, after being turned down for a job with the local government, gained control of the Maroochy Water Services system in Queensland, Australia [17]. Using only a laptop and a radio transmitter, the attacker was able to control 142 pumping stations for three months, "releasing over one million liters of untreated sewage into a stormwater drain that flowed into local waterways."
- The "Aurora Test," conducted by Idaho National Labs, despite some artificialities, illustrated how a large electrical generator could be destroyed, via a network connection, simply using cleverly-timed inputs from its SCADA controller [18].
- Although there are no published instances to date of a cyber attack causing a widespread critical infrastructure outage, academic research has illustrated the feasibility of such an attack. In research published in 2004, Albert et al. used a graph-based model of the North American power grid to show how successful attacks against a small number of distribution or generation nodes could have cascading effects on the rest of the grid [19]. A 2009 paper by Jian-Wei Wang and Li-Li Rong, examining the western U.S. power grid, also illustrated ways that critical infrastructure topologies can be vulnerable to attack [20].

Cyber attacks can also target information about military hardware, information that could be used to compromise those systems later on the battlefield:

- According to the U.S. government, " the owner of a Chinese aviation technology company with an office in Canada, conspired with two unidentified individuals in China to break into the computer networks of U.S. companies to get information related to military projects." The man's co-conspirators allegedly "claimed to have stolen 65 gigabytes of data from Boeing related to the C-17 military cargo plane" and "sought data related to other aircraft, including Lockheed Martin Corp.'s F-22 and F-35 fighter jets." [21]
- 2014 Media reports indicated a breach of three different Israeli defense companies, apparently resulting in the exfiltration of proprietary information about Arrow III missiles and Israeli UAVs [22].

However, it is important to note that, in general, as illustrated in the examples above, when a cyber attack involves a CPS, the vulnerability and the compromise take place in the computer system, while the effects are transmitted to the physical system through a PLC or a similar mechanism. An understanding of the interconnect and the physical system is important, but the fundamental security breakdown generally does not occur in the physical system, but in the cyber portion; the physical system's actions are usually just a manifestation of the compromise to the cyber portion.

While important, real-world compromises of CPS have been the exception, rather than the norm. For DoD, the most significant impact of real-world cyber warfare to date has been the compromise of important information, rather than the manifestation of a cyber attack on a physical system.

Foundational Skills

Therefore, a cyber security education, while interdisciplinary in scope, should also include a great deal of the fundamentals traditionally taught in computer science and information technology programs, such as networks, programming and scripting, and operating systems.

When we analyze the skills commonly thought of as supporting cyber security, many rely on a strong foundation in computer science. For example, we can examine NSA's syllabus components for its latest certification criteria as a Center of Academic Excellence in Cyber Operations [23], along with the topics' relationship to fundamental instructional areas in computer science, as well as closely related engineering fields. In Table 1, the first column lists the Cyber Operations content -- first required topics, then optional, of which 60% must be included in the academic program. The next columns indicate, respectively, whether the content referenced is traditionally taught in curricula for computer science and information technology (CS/IT), electrical and computer engineering or systems engineering (CE/SE), or some other department, respectively.

The NSA CAE criteria for Cyber Operations supplies just one example definition of the educational topics supporting cyber security, and others may differ slightly. However, we can conclude that, although the field encompasses topics from multiple disciplines, the preponderance of those derive from traditional areas of computer science and information technology. Overemphasizing the interdisciplinary aspects, therefore, risks giving short shrift to some of the core computing fundamentals.

Mandatory Content	CS / IT	CE / SE	Other
Low-level Programming Languages	X		
Software Reverse Engineering	X		
Operating System Theory	X		
Networking	X	X	
Cellular and Mobile Communications	X	X	
Discrete Math	X		
Overview of Cyber Defense	X		
Security Fundamental Principles	X		
Vulnerabilities	X		
Legal			X

Optional Content (60% minimum)	CS / IT	CE / SE	Other
Programmable Logic Languages		X	
FPGA Design		X	
Wireless Security	X	X	
Virtualization	X		
Large-scale Distributed Systems	X		
Risk Management of Information Systems	X		
Computer Architecture	X	X	
Microcontroller Design		X	
Software Security Analysis	X		
Secure Software Development	X		
Embedded Systems	X	X	
Forensics	X	X	
Systems Programming	X		
Applied Cryptography	X		
SCADA Systems		X	
HCI / Usable Security	X		
Offensive Cyber Operations	X		
Hardware Reverse Engineering		X	

Table 1: Topics required for certification as an NSA Center of Academic Excellence in Cyber Operations, and where those topics are most commonly covered in traditional university curricula.

Summary and Conclusions

In any discussion of an academic curriculum, the theory must be the foundation. However, in the modern field of cyber security, never has there been such a divergence between the traditional theories and the hands-on application. We bridge the gap to some degree with principles and pillars, which express concepts in an understandable way, but still require software tools and application-specific knowledge to implement. As a result, the maximally effective cyber education should be exclusive of neither, but seek a middle ground. Similarly, in the drive to include interdisciplinary studies in the realm of cyber operations, due to their real-world effects and connection to physical systems, we should not do so to the detriment of computer science and information technology, which form the foundation on which cyber attack and defense are built. Our future cyber warriors will be best prepared if we seek balance and find the middle way.

ABOUT THE AUTHOR

Commander Michael Bilzor, USN, PhD, is a Permanent Military Professor at the U.S. Naval Academy. As a Naval Flight Officer, he accrued more than 2,000 flight hours in the F-14 Tomcat and F/A-18F Super Hornet and flew combat missions in Iraq. At the Naval Academy, he served as course coordinator from 2013-2014 for the school's Introduction to Cyber Security class, taken by all midshipmen in their first year. Commander Bilzor has coached the midshipmen cyber competition team the last two years. His research interests are focused in cyber security, and he is currently associate chair of the Computer Science department.

572M Holloway Rd., Michelson 346
Stop 9F, Computer Science Dept.
U.S. Naval Academy
Annapolis, MD 21401-5002
Phone: 410-293-6802
Fax: 410-293-2686
E-mail: bilzor@usna.edu
Web: http://www.usna.edu/Users/cs/bilzor/

REFERENCES

1. Bell, D. Elliott, and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. No. MTR-2547-VOL-1. MITRE CORP BEDFORD MA, 1973.
2. Biba, Kenneth J. Integrity considerations for secure computer systems. No. MTR-3153-REV-1. MITRE CORP BEDFORD MA, 1977.
3. Kemmerer, Richard A. "Shared resource matrix methodology: An approach to identifying storage and timing channels." ACM Transactions on Computer Systems (TOCS) 1.3 (1983): 256-277.
4. Clark, David D., and David R. Wilson. "A comparison of commercial and military computer security policies." 2012 IEEE Symposium on Security and Privacy. IEEE Computer Society, 1987.
5. "NVD - Home." NVD - Home. NIST, n.d. Web. 08 Aug. 2014. <http://nvd.nist.gov/home.cfm>.
6. Klein, Gerwin, et al. "seL4: Formal verification of an OS kernel." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.
7. R., Maxwell. "Lines of Code: How Our Favorite Apps Stack up against the Rest of Tech." Phone Arena. PhoneArena.com, 12 Nov. 2013. Web. 08 Aug. 2014.
8. Various. "SI110: Introduction to Cyber Security, Technical Foundations." SI110: Introduction to Cyber Security, Technical Foundations. U.S. Naval Academy, n.d. Web. 08 Aug. 2014. <http://www.usna.edu/CS/si110/index.html>.
9. Various. "Information Assurance." Wikipedia. Wikimedia Foundation, 22 July 2014. Web. 10 Aug. 2014.
10. Various. "IA Guidance" Information Assurance Guidance. NSA, 15 Jan. 2009. Web. 08 Aug. 2014. <http://www.nsa.gov/ia/mitigation_guidance/>.
11. Various. "Critical Security Controls." SANS Institute -. SANS Institute, n.d. Web. 08 Aug. 2014. <https://www.sans.org/critical-security-controls/>.
12. Wall, Andru E. "Demystifying the title 10 - title 50 debate: Distinguishing military operations, intelligence activities & covert action." (2011).
13. Gellman, Barton. "Edward Snowden, after Months of NSA Revelations, Says His Mission's Accomplished." Washington Post. The Washington Post, 23 Dec. 2013. Web. 10 Aug. 2014.
14. Fishman, Steve. "Bradley Manning's Army of One." NYMag.com. New York Magazine, 3 July 2011. Web. 10 Aug. 2014.
15. Times, High. "Anonymous Unmasked." The Huffington Post. TheHuffingtonPost.com, 01 Apr. 2014. Web. 10 Aug. 2014.
16. Mittal, Pawan. "How Digital Detectives Deciphered Stuxnet, the Most Menacing Malware in History | Threat Level | WIRED." Wired.com. Conde Nast Digital, 09 July 0011. Web. 10 Aug. 2014.
17. Slay, Jill, and Michael Miller. Lessons learned from the maroochy water breach. Springer US, 2007.
18. Zeller, Mark. "Myth or reality–Does the Aurora vulnerability pose a risk to my generator?." Protective Relay Engineers, 2011 64th Annual Conference for. IEEE, 2011.
19. Albert, Réka, István Albert, and Gary L. Nakarado. "Structural vulnerability of the North American power grid." Physical review E 69.2 (2004): 025103.
20. Wei, Dong, et al. "An integrated security system of protecting smart grid against cyber attacks." Innovative Smart Grid Technologies (ISGT), 2010. IEEE, 2010.
21. Pettersson, Edvard. "Chinese Man Charged in Plot to Steal U.S. Military Data." Bloomberg.com. Bloomberg, 12 July 2014. Web. 08 Aug. 2014.
22. Krebs, Brian V. "Krebs on Security." Krebs on Security RSS. Krebs on Security, 14 July 2014. Web. 08 Aug. 2014.
23. Various. "NSA CAE CO" Academic Requirements for Designation as a Center of Academic Excellence in Cyber Operations. NSA, 12 Jan. 2012. Web. 08 Aug. 2014. <http://www.nsa.gov/academia/nat_cae_cyber_ops/nat_cae_co_requirements.shtml>.

Training the DoD Software Acquisition Professional

Robert P. Skertic, Defense Acquisition University

Abstract. Defense Acquisition University (DAU) is the department's premier software acquisition training institution. Software applications are now the primary method of providing warfighter capability in all of our programs. Education about how to do software acquisition management of the requirements, design, development, deployment, operations, maintenance and disposal of software applications is a key factor to providing operationally effective, efficient and timely capabilities for our warfighters. DAU educates our DoD Acquisition professionals with the evidence-based best practices, lessons learned and DoD policy mandates that allow our warfighters to receive highly capable and reliable software-based capabilities. This article describes where DAU is at with software acquisition training, where we are headed in the next couple of years and the long term realization that software now impacts all systems and all career fields. This article will help the reader understand the current DAU training model and how DAU is working with the IT Functional Leader to identify ways to train all of DoD as needed to ensure we deliver software acquisition management training that improves the IT acquisition outcomes for our warfighters.

Introduction

Software applications have become the primary method to provide warfighter capability for most of DoD's systems. DAU is the department's premier software acquisition training institution. This article describes where DAU is at with software acquisition training and where it is headed. Because of the growth of software acquisition, all systems and all career fields need to know some aspect of software acquisition. DAU's goal is to ensure successful software acquisitions across the department for all DoD stakeholders. DAU is working with the IT Functional Leader to identify ways to train all of DoD as needed to ensure we deliver the proper Software Acquisition Management (SAM) training to the DoD workforce. DAU's goal is to improve IT acquisition outcomes for our warfighters.

Background

IT acquisition management training includes hardware and software acquisition. Software acquisition education includes how to manage the requirements, design, development, deployment, operations, maintenance and disposal of software applications. Proper IT (software) acquisition management is a key factor to providing operationally effective, efficient and timely capabilities for our warfighters.

The mission of DAU is to provide a global learning environment to develop qualified acquisition, requirements and contingency professionals who deliver and sustain effective and affordable warfighting capabilities. DAU's vision is to enable the entire Defense Acquisition Workforce to achieve better acquisition outcomes for our warfighters.

The DAU IT career field includes Program Managers, Project Managers, IT Specialists in Policy and Planning, Enterprise Architects, Cybersecurity Specialists, Systems Analysts, Application Software Developers, Operating Systems administrators, Network Services Technicians, Data Managers, Internet/Web Managers, System Administrators and Customer Support personnel.

Currently, DAU's IT portfolio includes mandatory Defense Acquisition Workforce Improvement Act (DAWIA) training, Continuous Learning Modules (CLMs), Mission Assistance and Knowledge Sharing via the Acquisition Community Connection (ACC) Communities of Practice (COP) for IT and Software Acquisition Management (SAM).

Training Courses

Today, IT acquisition management training is primarily focused on four DAWIA courses:

1. Basic Information Systems Acquisition (Information Resource Management (IRM)), IRM101 (Level 1). This is a distance learning course (online course). This course focuses on describing and defining the basic terms of IT. All types of students take this course from the IT career field to other career fields like the Program Management (PM) career field. This course is scheduled to be completed within 60 days.

2. Intermediate Systems Acquisition, IRM202 (Level 2). This is a hands-on classroom experience. This course focuses on a working level experience in the typical acquisition environment, a DoD program office. This is the foundation course for the IT curriculum. This course helps the IT decision-makers identify the evidence-based best practices, lessons learned, rules of thumb and use them via classroom exercises and practicums (role playing in the program office environment). All IT career field supervisors and practitioners must take this course to achieve Level 2 education requirements. This is a two week or 10 day classroom experience.

3. Advanced Systems Acquisition, IRM304 (Level 3 First Course). This is a classroom graduate-level experience. This course focuses on the IRM type experience including CIO, PEO level, and Milestone Decision Authority (MDA) IT level decision-making experience. This course uses cases studies and some subject matter expert guest speakers to help the students understand the decisions needed to be made to ensure the enterprise IT decisions are being made correctly and why they were made for each program. This course is focused on the IT career field supervisors and PEO/MDA level practitioners. All IT career field personnel must take this course to achieve Level 3 education requirements.

4. Advanced Software Acquisition Management, SAM301 (Level 3 Final Course). This is a classroom graduate-level experience. This course focuses on the SAM type experience (PM and PMO software design and development decision-making experience). Using a sound problem-solving model, students practice making management decisions that a typical software program office has to make to be successful. This course includes a balanced number of subject matter expert guest speakers to help keep pace with the ever-changing software acquisition environment. This course is focused on the IT career field supervisors and program level practitioners. All IT career

field personnel must take this course to achieve Level 3 education requirements. SAM101 has been retired since the content from SAM101 was merged with IRM101 a few years ago. The basics of DoD acquisition courses (ACQ101, ACQ201A and ACQ201B) are pre-requisites for all IT courses.

Why IRM and SAM?

Years ago, there were separate SAM and IRM courses at each level. However, with the Services request to decrease training time and increase work time, the level 1 courses of SAM101 and IRM101 were merged into IRM101 and level 2 courses of SAM201 and IRM201 have been merged into one course called IRM202.

The accepted difference between IRM and SAM courses is that IRM is focused on the strategic planning and managing of the acquisition of IT at the system-of-systems level, enterprise IT Policy management issues and Enterprise Architecture levels. These are things that the MDA manages. Whereas, SAM is focused on the detailed planning, managing, designing, developing, deploying, operating and maintaining of the software products being acquired. SAM is tactically focused on the solution architecture, the software architecture being used between and by all five domains of DoD software systems (Weapons, C4ISR, Defense Business Systems, Modeling and Simulation, Infrastructure Systems and Services). These are the software applications and interfaces that DoD program managers manage.

Current IRM curriculum is based on the Clinger-Cohen Act (CCA), Title 40, Subtitle III, 1996 plus the latest NDAA IT management initiatives from 2011 (TITLE VIII on Acquisition Process for IT), DoDI 5000.02, Operation of the Defense Acquisition System, Defense Acquisition Guidebook, Chapter 7 (not updated yet with the latest DoDI 5000.02), DoDI 8500.01, Cybersecurity and DoDI 8510.01, Risk Management Framework for DoD Information Technology (IT), which explains how to certify and accredit IT Systems for authority to operate.

Current SAM curriculum is based on NDAA software initiatives from 2003 (Section 804 SW Improvements), 2007 (Section 853 SW Development emphasis for PMs), 2009 (Section 144 Open Arch, Section 803 SW Reuse), 2011 (Section 241 Software Assurance), DoDI 5000.02, Operation of the Defense Acquisition System, Defense Acquisition Guidebook, Chapter 4 (not updated yet with the latest DoDI 5000.02), and DoDI 8510.01, Risk Management Framework for DoD IT, which describes how to identify the software controls required to be designed in to secure software applications.

DAU also produces online CLM courses that specialize in one functional area. Continuous Learning modules for Engineering (CLE) are where most of the IT CLMs exist. The current IT CLM portfolio includes: CLE010 on Privacy Protection, CLE012 on Open Systems Architecture (OSA), CLE016 on Outcome-based Performance Measures, CLE022 on Anti-Tamper, CLE041 on Software Reuse, CLE060 on Software Measurement, CLE063 on CMMI, CLE068 on Intellectual Property Rights (Data Rights for commercial built software applications), CLE074 on Cybersecurity (March 2015 deployment), and CLL (Logistics)056 on Software Sustainment. Some of these courses are pre-requisites for our DAWIA courses.

Mission Assistance

DAU provides program office mission assistance to help DoD programs at their point of need. If you have current acquisition challenges, DAU can provide assistance. DAU has regional Associate Deans for Outreach and Mission Assistance (ADOMA) that lead the mission assistance efforts across the country.

Knowledge Sharing

DAU has established an ACC website including Communities of Practice (COP). Information Technology has two COPs based on the definitions above. The IRM COP is called the IT COP. The IRM or IT COP focuses on the CIO/PEO/MDA level of knowledge. This is the enterprise IT level of knowledge topics. The SAM COP is called the Software Acquisition Management COP. The SAM COP is the software architecture, software design, software development and management level of knowledge topics.

Organization

DAU has placed software acquisition training within the Engineering and Technology departments across the five DAU campuses (located in five regions: West, South, Midwest, Capital and Northeast and Mid-Atlantic campuses). The Learning Capabilities Integration Center, oversees the curriculum development for all DoD career fields.

Upcoming Changes in IT Curriculum

IT technology is changing at a very fast rate compared to DoD's ability to field programs. Law and policy changes are occurring annually as the federal government learns more about how best to manage IT. Basic IT (Software) acquisition management now touches just about every DoD career field from Program Management to Systems Engineering, to Logistics, to Contracting, to Test and Evaluation, to Science and Technology, etc… Software applications are now the primary method of providing warfighter capability in all of our programs.

Currently, DAU has just updated all of the IT DAWIA training courses with the latest DoDI 5000.02 (Operation of the Defense Acquisition System) and DoDI 8510.01 (Risk Management Framework for DoD IT).

The DoD IT Functional Leader has identified 41 IT competencies which need to be trained to the IT career field workforce. DAU is in the process of fully understanding what needs to be trained within each competency. DAU is establishing the "to be" IT training architecture. DAU will then update the DAWIA training courses with the applicable content from the 41 IT competencies.

Looking to the Future

Because software has taken over the functionality of most of our DoD systems, it is vital that all career fields have an understanding about how best to manage IT within their functional area. In addition, continued software technology advancements are causing the IT content to increase. For example, with the adoption of the DoD Cloud, we are now able to share information across domains anywhere in the world securely (Cybersecurity). The IT content footprint continues to increase but our current DAU courseware and time to cover the topics has stayed constant. DAU, under the direction of the IT Functional Leader, is re-thinking how to get the increased IT acquisition

content out to our IT career field students and how to better insert IT acquisition management across all career fields.

Conclusion

In conclusion, DAU provides the basic evidence-based best practices, lessons learned and DoD Policies/Guidance IT training for all IT career field positions (IT PMs, IT Specialists). Under the direction of the IT Functional Leader, DAU is looking at providing IT training to the other career fields like Program Managers, Systems Engineers, Contracting, Logistics, Business/Cost Estimating/Financial, Test & Evaluation, Production Quality Manufacturing and Joint and Service Program Management Offices. This training will help all DoD stakeholders to understand what it takes to acquire software-based products in the most efficient way providing reliable, quality IT capability for our warfighters! Come to DAU to learn more.

ABOUT THE AUTHOR

Professor Skertic is the Performance Learning Director for Information Technology for the Defense Acquisition University. He has held this role since April 2014. Prior to taking this position, Professor Skertic was the Acting Department Chair for the Capital and Northeast Region's Technology and Engineering Division (NE-ET). From 2011 to 2012, Professor Skertic served as the Deputy, NE-ET. From 2002 to 2011, Professor Skertic was the Software and Information Technology functional lead for the Defense Acquisition University (DAU). In this role, he worked with his peers in attempting to keep up to date with the latest trends and DoD initiatives involving architecture and software acquisition management while teaching DAWIA and Mission Assistance courses on IT and Systems Engineering.

Professor Skertic is a retired Army Lieutenant Colonel. Professor Skertic was commissioned in the Field Artillery. While on active duty, Professor Skertic served as a Product Manager (PM) of the Army's first database on fitness and the establishment of the Army physical fitness standards from 1982 to 2001. He then became the PM for the automation of the separation and bonus systems (finance). From there he became the Technical Director for the Army's Global Command and Control System (GCCS) program and ended his career on the Army Staff as the Army's Software Architect for Task Force XXI and the digitization of the 4th Infantry Division, Ft Hood, TX (i.e., Blue Force Tracking). He was a member of the Army's Acquisition Corps. He was the Army's representative to establishment of DISA's Common Operational Environment (COE) and the creation of the Global Command and Control System (GCCS).

Upon retiring in 1999, Professor Skertic went to work for INRI (eventually bought by Logicon and Northrop Grumman Information Technology) where he served as the contractor Deputy Program Manager for the Marine Corps Systems Command's Systems Engineering and Integration (SEI) Division. During this timeframe he helped the Marine's better understand the Army's tactical and theater software command and control architectures.

Professor Skertic holds a Bachelors of Science Degree in Engineering from the United States Military Academy at West Point and a Masters of Science Degree in Computer Science from the University of Southern California.

Phone: 703-805-5281
E-mail: Robert.Skertic@dau.mil

A "Thinking Framework" to Power Software Development Team Performance

Paul E. McMahon, PEM Systems

Abstract. Essence is a new Object Management Group (OMG) software standard [1, 2, 3] developed specifically for software development practitioners and teams. This article explains specific features of Essence that could help software teams improve performance in ways previous frameworks, including the CMMI®, Lean Six Sigma, and Scrum, have fallen short. The article also provides insight into why many performance improvement efforts fail, and how Essence-- or a framework with characteristics similar to Essence-- could provide the help organizations need to hit performance targets more consistently.

Essence: What it Is

Essence is a "thinking framework." Just saying this should start you thinking about performance differently from the way many view it when using frameworks such as the CMMI, Lean Six Sigma, and Scrum.

This is not meant to imply everything we have been doing in the past to improve performance is wrong. The vast majority of improvement approaches have strengths, and have helped organizations get better. However, there exists significant evidence indicating most improvement projects fail to achieve their goal [4, 5, 6].

Why are we facing this problem?

It is my contention, based on my forty years of working in the software industry, that the problem is not with the frameworks, tools, or practices organizations are using, but rather with the way organizations are going about deploying practice guidance and practice improvements. This is where a framework such as Essence can help.

Essence: What it Is Not

Essence is not a new method, or a new set of practices. It is not in competition with any of the popular frameworks or methods including CMMI, Lean Six Sigma, Scrum, or Kanban. It is a framework that can help organizations achieve performance goals by helping practitioners and teams implement whatever approach they are already using more effectively and efficiently. Today Essence is being tested in University field studies [7, 8] to determine the degree to which it can help teams work more effectively than using other popular approaches alone.

What Do We Mean By A "Thinking Framework" and How Can Essence Power Other Software Development Approaches?

By thinking framework we mean a framework that can help software practitioners and teams think through the tough problems they face by helping them ask the right questions, and find the optimum solutions based on their specific situation.

The reason this framework can work with whatever your team is currently doing is because it contains no practices that might conflict with your current approach, and it brings higher visibility to how well a team is implementing essentials that have been proven to exist on all software development endeavors. More importantly, when projects fail to meet performance goals the root cause can usually be traced to poor implementation of one or more of the essentials.

Why Software Development Endeavors Get in Trouble

Anyone who has been involved in the business of software development for any length of time can tell you the primary reason most software development endeavors get in trouble is because they fail to adequately address things most of us know are essential, and even when they recognize this failure they then fail to take timely corrective action.

Why–When We Understand the Problem–Can't We Effectively Implement the Solution?

We have known for some time how to solve this problem. We know that empowering development teams is a critical part of the solution as the team is best equipped to observe issues early, and take timely action. But exactly how an organization should go about empowering its development teams is hotly debated today--largely due to fear of lost control.

Why Do We Need Another Framework to Help Empower Teams?

The CMMI [9] and Lean Six Sigma [10] are useful improvement frameworks, but they are intended to be applied by trained process professionals, not software practitioners and development teams. Therefore they do little to help empower development teams to solve the common challenges faced each day on the job.

A popular framework commonly used today by practitioners and development teams is Scrum [11]. Scrum provides an effective framework to encourage teams to raise issues, solve issues, and keep their progress visible in a timely fashion, but Scrum provides little help to teams with respect to where they should look for issues, and how to accurately assess where the team really is with respect to addressing issues and achieving its goal.

Furthermore, while Scrum, and Agile methods in general, have been extremely popular, there is considerable literature available that indicates many organizations are continuing to fail when using Agile approaches alone.

Scott Ambler stated, based on a Dr. Dobbs State of the IT Union survey conducted in November, 2009, that only 11% of respondents indicated that their existing governance strategy works well with agile teams. Ambler said that this is an

indication that their organization is likely to apply traditional governance strategies (e.g. command and control) and that this strategy will not work with agile teams [12].

Another reason cited for these failures [13] is:

"trying to force a strict agile approach when it is not appropriate for every environment."

These facts leads us to ask two questions:

"How can we help our software teams determine the right level of agility for their specific project?" and, "How can we help our senior management move to the right governance strategy to best support their software teams and their organizational objectives?"

How a Framework, Such as Essence, Can Help

Essence is not an alternative to Scrum, or any other method, set of practices, or improvement framework. It is not another methodology. It was intentionally developed to be agnostic to specific practices and methods. It is a framework intended to be used with whatever practices, method and lifecycle an organization chooses to use and it can help teams ask the right questions and take the right actions leading to the right balance of agility and control given their specific situation.

This framework was developed by volunteers representing a wide range of software experiences and cultures including people from industry, academia and research in countries around the world [3]. The framework is not just a theory, but is based on what has proven to be essential to effective and efficient software engineering. Most important to the issues being raised in this paper, it has the potential to help senior management and software development teams work together to implement an appropriate agile governance strategy as recommended by Ambler.

To give you a real example demonstrating how the Essence framework can help with the challenge we face, at Carnegie-Mellon West in a recent field study [8] where students were asked to try Essence, the following was reported:

"While most styles of Agile retrospectives tend to focus on known issues, Essence reflections tend to make unknown issues apparent by covering the project holistically and reminding participants of critical areas that might be overlooked. These differences make Essence reflections and Agile retrospectives complementary. This is illustrated by the following student quote:

'Though the team was holding retrospectives every week already, having Essence discussions be a part of it allowed the team to touch on important aspects of the project; aspects which would otherwise be ignored'."

This finding about Essence, based on student feedback from a field study, demonstrates its potential to improve a team's understanding of unknown issues, or risks. Identifying risks is a critical first step necessary for teams to take action early to eliminate risks. This is a similar observation that was made by a senior experienced engineer in a major US DoD organization when first exposed to the Essence framework [6].

According to the Carnegie-Mellon Software Engineering Institute 60-80% of the cost of software development is in rework [14]. Rework is often caused by issues that were unknown to the team when the work was originally done. Rework is preventable by reducing unknowns early. Therefore anything we can do to reduce or eliminate risks can potentially reduce the cost of software development by 60-80%.

Key Elements Inside the Essence Framework

Key elements inside the Essence framework that I want to focus on in this article are Alphas, Alpha States, and Alpha State Checklists. Alphas are the essential things teams work with. There are seven Alphas inside the framework including Opportunity, Stakeholders, Requirements, Software System, Work, Way of Working and Team. Refer to Figure 1.[1]

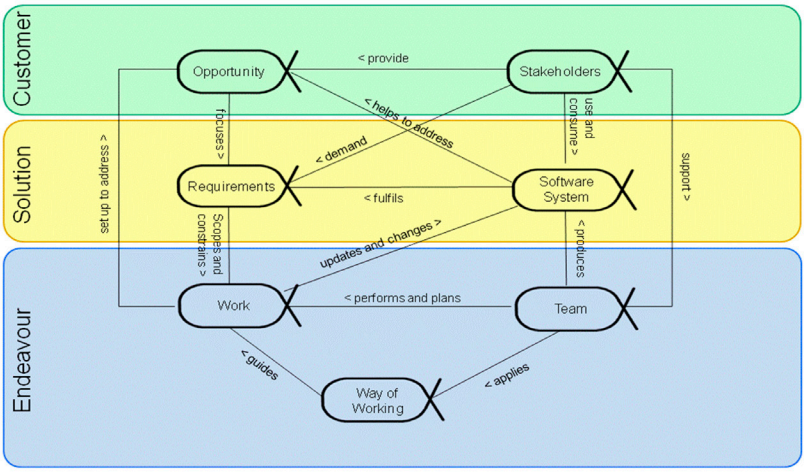


Figure 1 Seven Alphas inside the Essence framework

What Alphas Are, and Are Not

Alphas are ultimately about helping your team more accurately assess where they are and where they need to focus their effort next for project success. Some have struggled with the Alpha notion. Alphas are not abstract work products. Alphas always exist regardless of the degree of concrete work products supporting their existence.

The reason this is important is because when we focus on concrete work products alone we can miss essentials for software endeavor success. Over-focus on work products and missing the real goal is one of the problems commonly observed in the way past improvement models have been applied causing organizations to fall short of their performance goals [6].

How Essence States Can Help Accurate Progress Assessment Regardless of Lifecycle and Practice Choices

The Essence Alphas have states and checklists that can help teams assess where they currently are and where they need to focus their effort next. This includes projects using incremental, agile or waterfall lifecycles. As an example let us look at the Work Alpha.

Two of the states within the Work Alpha are Work Prepared and Work Started. The Alpha States and Checklists can be represented on cards. An example of these two Alpha States with a subset of checklists is shown in Figure 2.

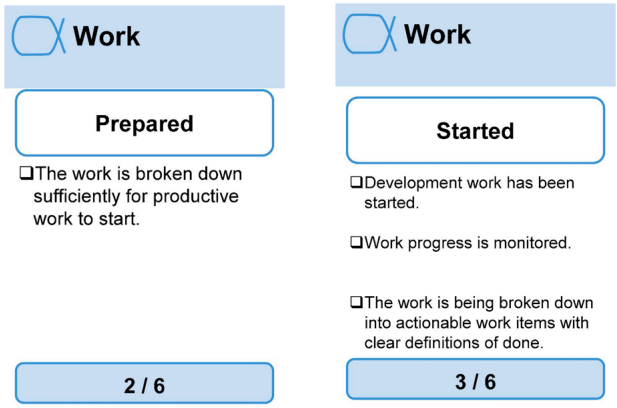


Figure 2 Work Alpha Prepared and Started States

The Checklist shown for the Work Prepared State is:

- The work is broken down sufficiently for productive work to start

The three checklists shown for the Work Started State are:

- Development work has been started
- Work progress is monitored
- The work is being broken down into actionable work items with clear definitions of done

One of the reasons why projects often fail to maintain their schedule commitments regardless of lifecycle or practice choices is due to inadequate work break-down and estimation before the work begins. When the work is not adequately broken down and estimated unrealistic schedule estimates often occur leading to poor schedule performance.

If you look closely at the way these checklists are worded in these two states you can see how a team can effectively use the checklists to assess the work break down status on any project including an agile project, an incremental project or a waterfall project.

For example, if your team is using Scrum and the planning poker practice [15] they could conclude that "the work is broken down sufficiently for productive work to start" as long as backlog items [11] have been selected and the team has reached consensus on the estimates for each backlog item for the first Sprint [11] using their planning poker estimation technique.

As the project proceeds, as long as they are holding their daily standups [11] to monitor work progress, and at the start of subsequent Sprints, backlog items are selected, broken down and estimated by the team, the team could conclude they are meeting all three checklist items under work Started.

On the other hand, a team that is using a waterfall lifecycle might conclude they have not passed the work Prepared state if only part of the work has been broken down because they might deem that insufficient given their lifecycle choice. This example demonstrates how a team can use the Essence framework to more accurately assess if the work is sufficiently broken down and estimated regardless of lifecycle choice.

If the work is not adequately broken down the team can raise this issue and make it a priority to solve the problem before it leads to more serious issues later in the project. Today, on many projects, including those that use other popular frameworks and methods it is too easy to let this type of "small issue" slip by, and lead to a "big issue" downstream.

Experienced practitioners have known for a long time that small issues not handled early are often the cause of big issues downstream, but knowing this has not helped. One possible reason is because we have not given our practitioners and teams a framework that gives them solid evidence that actions need to be taken to resolve issues at an appropriate time.

How Essence Checklists Are Different and Can Help Teams Assess Progress More Accurately and Take Action at an Appropriate Time

Let us now look at an example demonstrating how Essence checklists are different from traditional checklists and how they can help team performance.

Traditional checklists are what I refer to as "existence checks" [6]. Examples include:

- Do you have a plan?
- Do you have a design document?
- Did you conduct a peer review?

Existence checklists are easy to use because the answer is a simple yes or no that requires little discussion. But existence checklists can lead to a checklist mentality, and they do not help with questions related to how well the team is performing or how much of a certain activity the team should be doing. Existence checklists are what many quality organizations use today and they are common among organizations that use the CMMI model. One reason for this is because existence checklists are easy for external appraisers to use, or external quality audit personnel who are not intimately familiar with the project they are auditing.

But existence checks do little to help teams assess where they are in terms of how much effort is still required to get the job done on a specific project. Following is an example demonstrating how Essence checklists go beyond existence checks helping teams improve performance by improving their progress assessment and improving their decisions on where they need to focus their attention next.

Example: Checklist item Requirements Alpha Coherent State

A checklist item for the Requirements Alpha Coherent state is:

- Conflicts are identified and attended to

Just verifying that a requirements document exists would not be sufficient to verify that this checklist item is met. This checklist item is asking us whether or not we have conflicting requirements and if we do are they being addressed? Often the real pain that teams face originates from conflicting requirements that they do not know how to handle, and it is these types of problem areas that often lead to latent defects and extended integration schedules.

Because Essence contains states and checklists as a stable reference it provides a degree of assurance that progress is assessed more objectively and accurately-- and appropriate control is maintained. Furthermore, because the Essence states and checklists are agnostic to a team's lifecycle and practice choices it can help power whatever approach your team chooses to use.

Not Checklists an External Auditor Can Easily Apply

Many of the Essence checklists are not checklists an external auditor can easily apply by looking at a project from the outside. You need to be intimately involved in the project to answer honestly the questions that many of the Essence checklists lead you to ask. This is one reason why this framework is for software teams and why practitioners need to be more actively involved when making key decisions that help to steer a successful high performance project.

Traditional checklists lead us to simple yes/no answers that require little discussion, and fail to provide an accurate status of the project. Many of the Essence checklists lead us to deeper questions, and deeper analysis– the kind that gets to the real issues that ultimately affect project performance.

Example Of a Team Using Essence to Assess and Solve a Difficult Situation

The SEMAT[2] volunteers have been working on an Essence User Guide that will be made available from the SEMAT web site [16] to help guide teams with options they have to apply the framework. One of the scenarios from the User Guide[3] provides an example of how a team could use the Essence framework to assess a difficult situation.

In this scenario the team realizes they have a problem with a resistant stakeholder. The scenario demonstrates how a team can use the alpha states and their checklists to drill down and solve a specific problem.

Since the team knows they are having trouble with a stakeholder they first assess where they are with respect to the Stakeholder Alpha. Their discussion leads to team agreement that they have achieved the first state, stakeholders Recognized, but they have not achieved the second state, stakeholders Represented. They agree their next step is to get a stakeholder representative appointed.

The next step is to get the stakeholder Involved and they do this by interviewing him trying to find out what is behind the resistance. They learn through the interview that he does not see the value of the new system, which leads to the Opportunity Alpha and the Value Established state. This in turn leads to the Software System Demonstrable state once the team recognizes they need to conduct a demonstration to help the stakeholder see the value. Refer to Figure 3.

What you learn through this scenario is how the discussion leads through a sequence of alphas helping the team figure out the next action to solve the problem. This scenario demonstrates how a team can conduct their own root cause analysis and figure out the right actions to solve a problem in a timely way.

When a group of students at Carnegie-Mellon West that used Essence in an early field study [7] were asked if Essence's monitoring and steering approach had value to a project team, 90% of the students said that following the approach was worth their time, and 80% said they would use the approach again on their next project.

Lean Six Sigma provides a powerful tool kit to help organizations conduct root cause analysis, and take action to improve. But to use this tool kit often requires the expertise of a Lean Six Sigma Black Belt which requires hundreds of hours of study to reach the required proficiency level. Like the CMMI, Lean Six Sigma is a tool kit for process professionals, whereas the Essence framework is for development teams to help them solve their problems themselves in a timely way.

What's Different About the Essence Approach?

I stated in the beginning of this article that the reason many organizations are falling short of their performance goals is not because of their choice of practices or tools, but rather because of the way they are going about deploying their practice guidance and practice improvements.

Fundamental to the Essence approach is-- rather than making radical changes to whatever you are doing today-- to make changes in small steps based on where your development teams need the most help right now.

How often do we hear our practitioners say:

"My company processes do not help me with the real problems I face each day on the job."

It is my contention that the cause of this problem is the fact that too many organizations are focusing their guidance and improvement efforts in the wrong area. We spend too much effort trying to tell our teams what to do in situations they already know how to handle, and too little effort helping them with the common but difficult situations where they need help the most. Instead of just telling our development teams what they should be doing, we should be listening more for where they need help, and then focusing our improvement efforts accordingly. This is an area where the Essence pattern notion could help. [6]

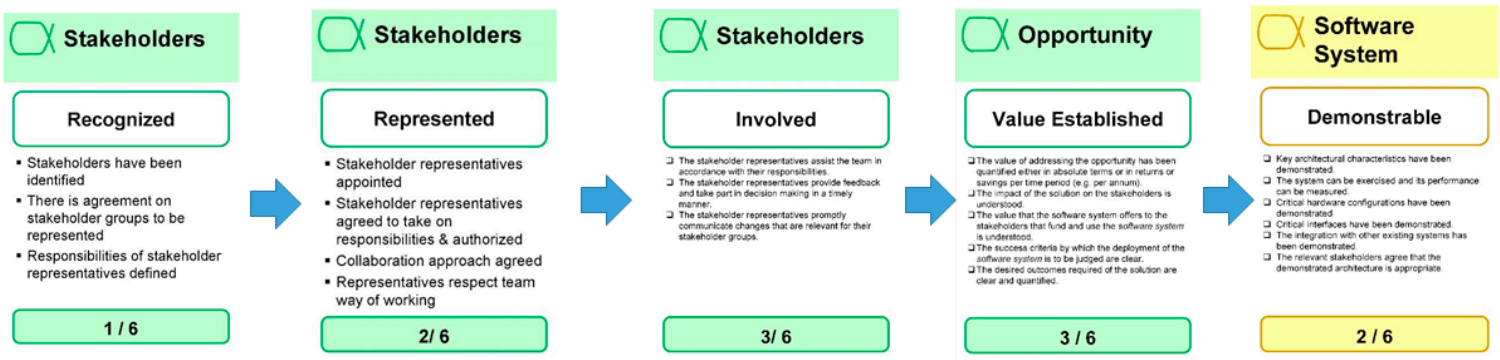


Figure 3 Digging for a Root Cause and Solution Using Essence States and Checklists

What is an Essence pattern?

A Pattern in the Essence system is defined as, "an arrangement of the other language elements (e.g. alphas, alpha states,...) into meaningful structures."

You can think of a pattern as a simple mechanism to deploy a small slice of useful information that can help your teams with a specific challenge. Following is a simple example [6].

The Dictated Schedule Pattern

Project Lead speaking to her team:

"Management has dictated we will not slip schedule so do whatever it takes to get the job done."

A developer responds:

"I will skip my design review, and much of the testing I planned to do because I do not see any options."

A second developer replies:

"We could focus the design review and testing just on the areas where we have seen problems in the past."

What would you do, if faced with the same dilemma? Are there other options?

Using Patterns To Help Empower Your Development Teams To Make Better Decisions

When I help companies that want to move decision-making deeper into the organization I encourage them to write their processes in a way that supports practitioner decision-making. One way to do this is by providing criteria and options in their processes. Now let us look at this scenario from the Essence framework perspective.

The Dictated Schedule Pattern From the Essence Framework Perspective

The dictated schedule scenario relevant alphas are Work and Way of Working. Refer to figure 4.

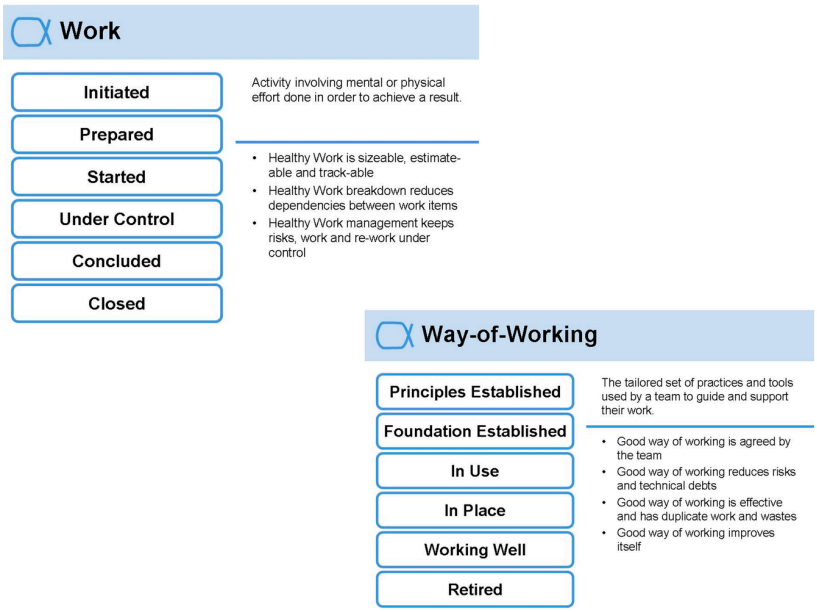


Figure 4 Work and Way of Working Alphas

Work is defined as: Activity involving mental or physical effort done in order to achieve a result. And Way of working is defined as: The tailored set of practices and tools used by a team to guide and support their work.

Now let us assume the team had assessed the project to have achieved the following states (Refer to Figure 5):

- Work Under Control
- Way of Working In Place.

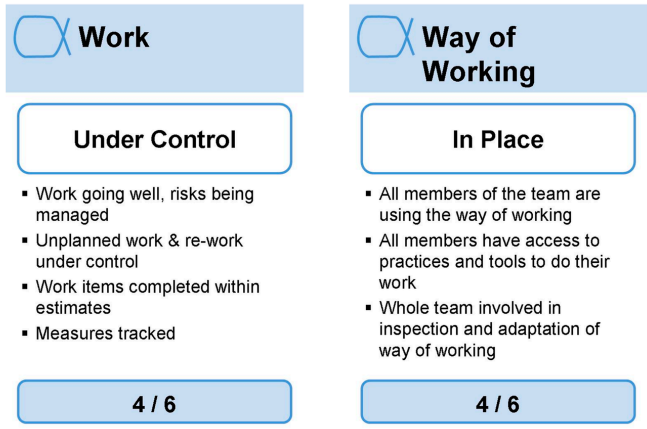


Figure 5 Work Under Control State and Way of Working In Place State

Based on the checklists, these states imply that:

- the work is going well,
- risks are being managed, and
- all members of the team are using the way of working.

Now think about those checklists again, given the current scenario.

Will we introduced new risks if we follow the suggestion of the second developer?

Is the suggestion to focus the testing and review just on areas where the team has observed problems in the past allowed within their agreed way of working?

The answers to these questions depend on each project's specific situation, and your team's agreed way of working. Note, this is an example demonstrating how a team could fall back, and how the Essence framework when applied by a development team as part of their progress assessment tool-set can help a team assess progress more accurately helping them take the timely actions needed to keep their software endeavor on course.

Current Status and Next Steps

The Essence standard is still young having been adopted by the OMG in June, 2014. Some early adopters including MunichRe, the world's leading Reinsurance Company, and Fujitsu Services which used an early version of the framework have reported encouraging results [2].

There is also effort going on in multiple Universities around the world to integrate the Essence framework into existing software engineering curriculum including at the Universidad Nacional de Colombia where Dr. Carlos Maria Zapata has developed and is currently delivering the first full course based on SEMAT and the Essence framework.

The next critical step for Essence is to gather more case study information and hard data to help us better assess the value this new standard can bring to power software development team performance.

Disclaimer:

CMMI® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR

Paul E. McMahon, Principal, PEM Systems has been an independent consultant since 1997. He has published more than 45 articles and multiple books including "15 Fundamentals for Higher Performance in Software Development." Paul is a Certified Scrum Master and a Certified Lean Six Sigma Black Belt. His insights reflect 24 years of industry experience, and 17 years of consulting/coaching experience. Paul has been a leader in the SEMAT initiative since 2010.

Phone: 607-798-7740
E-mail: pemcmahon1@gmail.com

REFERENCES

1. OMG Essence Specification, <http://www.omg.org/spec/Essence/Current>
2. Jacobson, Ivar, Ng Pan-Wei, McMahon, Paul, Spence, Ian, Lidman, Svante, The Essence of Software Engineering: The SEMAT kernel, Oct, 2012, ACMQueue, <http://queue.acm.org/detail.cfm?id=2389616>
3. Jacobson, Ivar, Ng Pan-Wei, McMahon, Paul, Spence, Ian, Lidman, Svante, The Essence of Software Engineering: Applying the SEMAT kernel, Addison-Wesley, Jan,2013
4. Glazer, Hillel, CMMI Failure Modes and Solutions – Paving the Path for Agile & CMMI Interoperability <http://prezi.com/vttomwejpe83/cmmi-failure-modes-and-solutions-paving-the-pathfor-agile-cmmi-interoperability/>
5. Wall Street Journal, Where Process Improvement Projects Often Go Wrong <http://online.wsj.com/news/articles/SB10001424052748703298004574457471313938130>
6. McMahon, Paul, 15 Fundamentals for Higher Performance in Software Development, PEM Systems, July, 2014, <https://leanpub.com/15fundamentals>
7. Peraire, Cecile, Sedano, Todd, State-based Monitoring and Goal-driven Project Steering: Field Study of the SEMAT Essence Framework, International Conference on Software Engineering (ICSE 2014), Hyderabad, India, June 2014, <http://works.bepress.com/cecile-peraire/1/>
8. Peraire, Cecile, Sedano, Todd, Essence Reflection Meetings: Field Study, International Conference on Evaluation and Assessment in Software Engineering (EASE 2014), London, UK, May 2014, <http://works.bepress.com/cecile_peraire/31/>
9. Chrissis, Mary Beth, Konrad, Mike, Shrum, Sandy, CMMI for Development: Guidelines for Process Integration and Product Improvement, V1.3, Third Edition, Addison-Wesley, 2011
10. George, Mike, Rowlands, Dave, Kastle, Bill, What is Lean Six Sigma?, McGraw-Hill, 2004
11. Schwaber, Ken, Sutherland, Jeff, The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game, July, 2011
12. Ambler, Scott, Lines, Mark, Disciplined Agile Delivery, IBM Press, 2012
13. Stafford, Jan, What's Behind the Backlash Against Agile? <http://searchsoftwarequality.techtarget.com/feature/Agile-development-Whats-behind-the-backlash-against-Agile>
14. <http://www.cmu.edu/govrel/PDFs/2011-briefing-book/2.9-software-engineering-2011.pdf>
15. Cohn, Mike, Agile Estimating and Planning, Prentice-Hall, 2006
16. SEMAT Web Site, <www.semat.org>

NOTES

1. The Essence figures in this document are provided courtesy of SEMAT Incorporated
2. SEMAT stands for Software Engineering Method and Theory. SEMAT is the world-wide group of volunteers that developed the Essence framework. For more information about SEMAT refer to <www.semat.org>.
3. I would like to thank the following SEMAT volunteers for their contributions on this scenario: Dr. Cecile Peraire, Dr. Mira Kajko-Mattsson, Winifred Menezes, Barry Myburgh, and Dr. Robert Palank.

Training Software Project Managers

Lawrence Peters, Software Consultants International Limited

Abstract. The presumed goal of training software project managers is to equip them with the knowledge and competencies that will help them to be successful. These will not guarantee success but make success more likely. Over the years, the notion of success has expanded greatly from simply meeting requirements, delivering on time and not exceeding the budget to include a plethora of other success criteria. In fact, what success is often changes many times during the project. This and many other facets of software project management frustrate and perplex untrained software project managers since most enter into this role untrained [1]. This article presents what anecdotal and experimental evidence has shown software project managers need to know that can be conveyed via training programs. Today's software project manager can also benefit from this information to overcome many of the misperceptions about nearly everything regarding software project management.

Targeted Issue

Software Engineering Education Issue - The importance of adequate management of software projects is slowly becoming more apparent. Most current software project managers and those seeking to become software project managers have either been inadequately trained to address today's software engineering issues or have not been trained at all. This article examines what we now know is needed to successfully manage software engineering projects and how education can help transfer this information.

Introduction

Software project management represents a paradox within the software engineering community. It has been described as being more vital to software project success than all other factors combined [2], yet there are still no conferences or journals devoted to this topic. In fact, international conferences on software engineering rarely list software project management as a topic in the call for papers topic list. Finally, we are slowly realizing what other knowledge work related professions have known for a long time – project managers are not born, they are made – through education. The problem is, there is no general agreement on what knowledge and skills a software project manager needs in order to be successful. In fact, we have yet to agree on just what success in software engineering is. This article examines what software engineering project managers need to know, what skills they must possess and how these may be acquired through education. The resources cited are not restricted to software engineering alone as there is a lack of research in this subject.

The Nature of Software Engineers

Some years ago, psychologists sought to answer the following question, "What is it about software development that has attracted so many people from such a broad range of disciplines? They found that most software engineers shared two unique (taken in combination) psychological characteristics [3]:

1. High Growth Needs Strength - A desire to solve challenging problems.

2. Low Social Needs Strength - A strong tendency to work alone.

To summarize, these people are attracted by software development because they want to deal with significant, technically challenging problems and not have to deal with other people. Unfortunately, the scale and complexity of today's software development efforts requires that teams of software engineers build and maintain them. Since software project managers are drawn from the ranks of the software engineering teams, they also display the psychological profile of most software engineers and tend not to have great "people skills." This works against them in a management role [4].

What Software Project Managers Do

Without doing any research, it is difficult to figure out just what software project managers do. The model we will work from proposes that software project managers are responsible for performing 5 basic functions [4] often in parallel, executed in concert with their team:

Scheduling – Laying out a list of milestones and dates consistent with the contract. Contrary to what you may have read [5], scheduling and planning are not the same activity.

Planning – Detailing the tasks and subtasks that must be successfully executed in order to proceed from one milestone to the next. This is an ongoing process throughout the project to adapt to unforeseen problems.

Controlling – Monitoring the project's progress, taking action to recover deviations from the project plan, as necessary.

Staffing – Acquiring the human resources needed to successfully execute the project plan.

Motivating – Creating and maintaining a physical and psychological environment that ensures the development team works at its full potential.

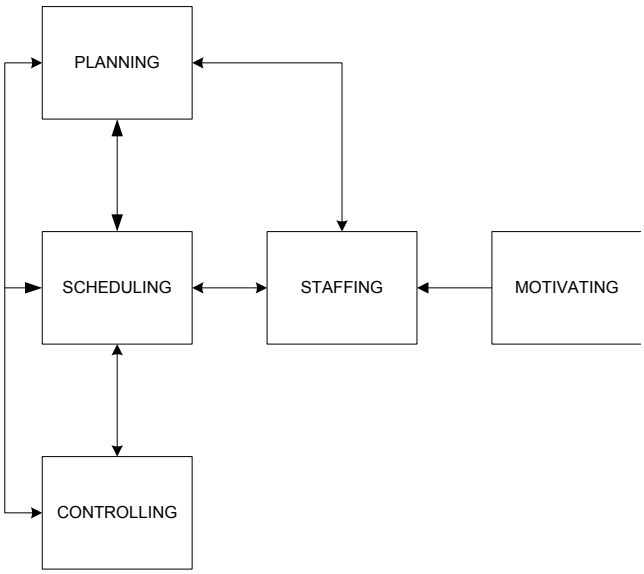


Figure 1: Software Project Manager Activities

Figure 1 shows the relationship among these activities. The relationship among these functions is shown in Figure 1. If those topics appear distant from programming, it is because the software project manager is more like the manager in baseball, not a player. Before heading into software project management, a software engineer needs to consider whether or not they are ready to move into this very different domain.

Where Do We Get Software Project Managers From?

Several decades ago, a very successful CEO (Robert Townsend) wrote a book detailing what needed to be done to improve how companies functioned [6]. One of the points he made seems to have been ignored by the software engineering industry [1]. What Townsend was trying to communicate was the fallacious belief that the most capable person in a team should become its manager. For example, we have a team of 5 people. Someone has to be responsible for meeting with the client, preparing status reports and assigning responsibility for specific development tasks to team members with their concurrence. If software project management duties are assigned to the most skilled software engineer, the productivity of the group is reduced because this highly skilled person has little or no time to do programming. Besides, management tasks and development tasks require very different mindsets. Worse, it is likely that this highly skilled person will not be empathetic to less skilled team members and help them improve their skills, help them solve difficult programming issues, and so forth.

A Few Misconceptions Seen as Self-Evident Truths by Many Software Project Managers

People work for money – Money ranks as 5th in importance [7] (see Table 1). But more importantly those who have studied why we work [8, 9, 10] have concluded we work for self-esteem, fulfillment and other reasons, not just for money. We have a paradox here in that the most expensive reward (salary) is the least appreciated while the most appreciated reward (a simple "Thank You") is the least expensive.

If we get behind, we can catch up – using Earned Value Management, if the project is 15% complete and behind schedule, based on a study of 700 DoD contracts, the chances of getting the project back on plan are nil [11].

Putting Pressure on the Team will Improve Productivity – Presumably the knowledge and experience the team brings to the project are what are needed to do the job. It has been shown that pressure to perform causes the team to break up into individual problem solvers effectively destroying the collective intellectual power of the group [12].

To Avoid Getting Behind, we will start with a bigger team – This is done to avoid adding people to a late project but it is surprisingly ineffective [13].

Offer some big reward, that will get them working – It has been shown that if the reward is big enough, people will cheat to get it [14].

Break up successful teams to other groups to spread the knowledge – The most successful software company(s) in the world do not do this but keep teams intact as much as possible and work to help other teams to learn to be successful [15].

Factor	Manager's Importance Rank	Non-Manager's Importance Rank
Salary	1	5
Job Security	2	4
Promotion/Growth Opportunities	3	7
Working Conditions	4	9
Interesting/Challenging Work	5	6
Personal Loyalty to Workers	6	8
Tactful Discipline	7	10
Appreciation for Work Done	8	1
Help with Personal Problems	9	3
Being in on Things	10	2

Table 1: Relative Importance of Project Factors

Software engineers Do not really need to be reviewed, they know how they are doing – Evaluating the performance of software engineers using the usual "one size fits all" human resource system is inappropriate for most high technology workers [1, 4]. A better approach is to tie evaluations to an individual's contribution to achieving corporate or group strategy [4, 16].

We can accurately predict the future – Nobody has been successful at this but now we know why we fail to accurately estimate software projects and how to correct for our over optimism and failure to fully recognize risk [16, 17].

Treat everyone the same – This century may mark the highest occurrence of multi-generational teams since we were an agrarian society. In addition, both in-house and outsourced projects will involve multiple cultures. Being aware of and responding to the issues of the value systems of these various groups will require knowledge and diplomacy. Due to the worldwide financial crisis, these are issues all software project managers must address [18].

The preceding represent both misinformation and some antipatterns. An antipattern is a solution to a problem that actually makes matters worse. More than 95 of these have been identified, categorized and published [19] and the list continues to grow.

What Do Software Project Managers Need to be Taught?

One way to answer this question is to look at what successful software projects have in common, then glean from that what we need to provide to current and prospective software project managers. A study of nearly 600 software projects worldwide that were reported to have been successful found a number of common "success oriented" factors [20]. These factors did not guarantee success but made it more likely (Table 2).

In examining Table 2, you might conclude that you could have guessed at some of the items without the benefit of a disciplined literature search worldwide. But there are a couple of items you may find need further explanation. One is the term, "Competent project manager." The knowledge and competencies that constitute a competent software project manager is what this article is about. At the present time, we do not know whether or not a software project manager is competent until

it is too late – the project has either been successful or failed. The other item that may seem curious occurs on the output side of Table 2, "Job satisfaction." It turns out that if software engineers do not find the work they are doing to be professionally satisfying, challenging and so forth, their productivity is low. Presumably, successful software projects result in large part from having a productive development team [16].

Input Factor	Brief Description
Clearly stated requirements	Clear and well understood requirements
Involved users	Active and continuous participation of users during the development process
Engaged, competent project manager	A project manager with the required management and leadership skills, able to share the project's vision
Project planned and scheduled	A project plan and schedule developed with stakeholder participation to achieve user goals
Engaged, skilled team members	Competent team members with domain and technical knowledge, as well as positive attitude about the project
Teamwork and communication encouraged	Development team with compatible personalities who enjoy working in a team environment and have a cooperative and mutually responsive relationship
Output Factor	**Brief Description**
Schedule and budget estimate maintained	Finishing the project within estimated budget and timeliness of delivery
Customer and user needs satisfied	Making easy-to-use, user friendly systems that meet requirements
Job satisfaction experienced on development team	The development team has a sense of accomplishment that sufficient quality and functionality were delivered and that they were given enough freedom and independence to be successful
Product quality, functionality and performance meet high standards	The working product reflects the desired scope and overall quality

Table 2: Input and Output Factors of Successful Projects [20]

Who Will Teach Them?

This issue presents us with a dilemma. Most successful software project managers are very busy with little spare time to contribute to the educational system(s) in their area. Besides, the fact that the word "management" would appear in the title of the software project management course can and has set off a turf war between the software engineering department and business administration departments in many universities. Professional development courses or extension courses help but content, quality and instructor qualifications come into play. For example, has the prospective instructor ever been a software project manager? Some very famous people lecturing on this subject have never managed a software project.

What Kind of a Job Are We Doing Now?

Not a very good one. Training in software project management if offered at all at the bachelor, master or doctorate level as part of a degree in software engineering is often optional – should not it be required? Why? Even if the software engineer never goes into management, he/she is going to be asked how long a task will take, what confidence level they have in the estimate, what they will need and so forth. Without training in planning and scheduling the answer to such queries can be problematic. During the execution of the plan, the issue of the status of the work will also come up. Without training in the use of earned value management, the most likely answer will be, "OK." If you have ever sat in on a project status meeting and wondered what metric was used to determine "OK" when you had heard rumors that things were not OK, you know some objective method needed to be applied.

Analysis of the content those university programs that offer software project management, whether it is required or optional, reveals that they are focused on programming issues, Agile, software tools and a general condemnation of the Waterfall Lifecycle.

Is Software Project Management Important?

The software engineering industry has spent more than a half century developing dozens of new methods, and techniques all directed at solving, "the software problem." What effect has all this had – an abysmal one [21]. Programming productivity has improved in a linear way by less than one source line per person month per year from 1960 to 2000. But facets of our profession continue to annoy customers. These facets are uncertainty of delivery date, cost, quality, ease of use, maintainability, communication and so forth. Remember, the software project manager is the biggest single factor in determining project success – bigger than all other factors combined [2]. One would think that with such importance, software project management would be the last subject we would choose to be optional.

What Software Project Managers Need to Know

For those who feel they would like to manage a software project here is a short list of skills you will need and some you won't.

Needed

Interpersonal skills – the ability to connect with each member of your team as a friend and colleague, a servant – leader who team members view as having their best interests at heart.

Communication skills – both in written expression (e.g. written reports) and presentations (e.g. PowerPoint or other format in front of an audience).

Basic Cost Accounting – if you do not know the 3 to 5 most common factors that make a $75,000 per year software engineer cost your project $120,000 or more per year, you do not have these.

Negotiating, motivating, evaluating personnel - the list goes on and on but notice the absence of programming skills.

Mentoring – work to improve the performance of team members and the team as well as creating your own replacement.

Encouraging Prosocial behavior – simply thanking the team and individual members helps to overcome independent tendencies to form an effective team [22].

Sensitivity to Cultural and Generational differences More than at any time in the history of the United States and some other developed countries, multiple generations are having to work together. The differences between the value systems of different generations can cause frictions within the team. Similar comments apply to cultural differences due to the world wide influence of software engineering in nearly every aspect of people's lives.

Self-confidence – consistently hire people smarter and more knowledgeable than you. The productivity of the team will reflect your skill in spotting talent and your job will become easier.

Not Needed

Programming, being quick to anger at failure, tending to fix the blame rather than the problem, embarrassing a team member in front of their colleagues because they made a mistake. This may be a sudden reaction to bad news but it can have negative consequences that go beyond the team member being chastised. Other team members may see this as an indication that trying something new or difficult should be avoided. Regardless, this reduces team productivity.

What Would a Curriculum for Software Project Managers Look Like?

More than 30 years ago I wrote the first M.S. in Software Engineering curriculum published by the Association for Computing Machinery [23]. It was adopted and implemented by Seattle University. Since that time, with the exception of software engineering, the engineering profession in general has incorporated personnel issues into management [24] training as well as the nature of management itself [25]. What is becoming apparent is that the nature of management in general and software project management in particular has changed dramatically since that original curriculum development.

So what would a curriculum look like to train software project managers? A lot different from the ones I found on the internet which emphasized specific lifecycles blaming the waterfall lifecycle for everything from athlete's foot to zits, programming methods (e.g. Agile) and programming languages. But that is to be expected. We tend to teach what we know, what we have experience with and so forth. In the hope that this will help the professors of today and tomorrow, here are some suggestions and subjects that need to be incorporated into software project management curricula:

Suggestions –

Require that all students, regardless of whether they are at the B.S., M.S. or PhD level take and pass at least an introduction to software project management in order to receive their degree. This is already the case in Europe as part of the European Master on Software Engineering (EMSE) program.

Eliminate any discussion of programming languages, methods, software tools and so forth from the software project management class(s).

Seek out software professionals, software project managers and key stakeholders in your area to determine what training software project managers appear to be lacking.

Form a review committee from the people in item 2 to review and critique the content of the software project management course(s).

Prepare a presentation containing an overview of the course and deliver it in person at major employers as well as publicly to ensure sufficient enrollment to fund the class.

Here are Some Things You Can Do

Look at what your local university(s) offer in the way of software project management classes – not just the course titles but their content as well. Ask yourself if someone had the knowledge imparted by these classes, would they be more likely to be successful? If the answer is no, identify what topics should be added, which should be dropped from the course(s) and speak with the college or university focal point for these classes. Be sure to emphasize the increased attendance and revenue that could result from these changes. Also, enlist the help of colleagues from other firms to bring about the required changes. If the university senses there exists an area wide market, they may be more amenable to change.

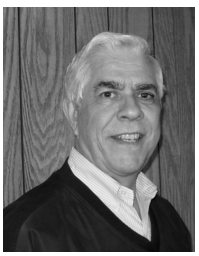
Closing Comments

Decades ago, we began training software engineers in a broad range of methods and techniques directed at both improving the quality and quantity of software systems. Though the progress may seem slow, it is progress. If we do the same for software project management, the benefits will likely accelerate the pace of improvement far beyond what we have seen so far.

Disclaimer :

(Portions of this article contain excerpts from the Kindle eBook, "Managing Software Projects: On the Edge of Chaos, From Antipatterns to Success")

ABOUT THE AUTHOR



Dr. Lawrence (Larry) Peters has 4 decades experience in software engineering as a software engineer, project manager, consultant, and educator (he currently teaches Software Project Management in Spain via Skype). He has worked on many defense systems. His clients have included many Fortune 100 companies, the US DoD, and the Canadian Defense Establishment. He has a B.S. in Physics, an M.S. in Engineering and a PhD in Engineering Management. He created the first Software Engineering laboratory for the Canadian Air Force, written 4 books and published several papers.

E-mail: ljpeters42@gmail.com

REFERENCES

1. Katz, R., "Motivating Technical Professionals Today," IEEE Engineering Management Review, Volume 41, Number 1, March, 2013, pp. 28-37.
2. Weinberg, G., Quality Software Management, Volume 3: Congruent Action, Dorset House Publishing, New York, NY, pp. 15-16
3. Couger, D. J. and Zawacki, R. A., Motivating and Managing Computer Personnel, Wiley-Interscience, New York, N. Y., 1980.
4. Peters, L. J., Getting Results from Software Development Teams, Microsoft Press Best Practices Series, Redmond, Washington, 2008
5. McConnell, S., The Software Manager's Toolkit, IEEE Software, From the Editor column, July/August, 2000
6. Townsend, R., Up the Organization: How to Stop the Corporation from Stifling People and Strangling Profits, Josey-Bass, 2007
7. National Study of the Changing Workforce, Published by the Families and Work Institute, New York, NY, 1993.
8. Herzberg, F., Work and the Nature of Man, The World Publishing Co. , Cleveland, Ohio, 1966
9. Maslow, A. H., The Farther Reaches of Human Nature, Viking Press, New York, N. Y., 1971
10. McClelland, D. C., The Achieving Society, Van Nostrand Reinhold, Princeton, N. J., 1961
11. Fleming, Q. W. and Koppelman, J. M., Earned Value Project Management - 4th Edition, Project Management Institute, Newtown Square, Pennsylvania, 2010.
12. Gardner, H.K., "Performance Pressure as a Double Edged Sword: Enhancing Team Motivation While Undermining the Use of Team Knowledge," Harvard Business School, Working Paper 09-126, 2012.
13. Staats, B. R., Milkman, K. L., and Fox, C. R., The Team Sizing Fallacy: Underestimating The Declining Efficiency of Larger Teams, Forthcoming article in Organizational Behavior and Human Decision Processes.
14. Gino, F. and Ariely, D., The Dark Side of Creativity: Original Thinkers Can be More Dishonest, Harvard Business School Working Paper 11-064, 2011.
15. Huckman, R. and Staats, B., The Hidden Benefits of Keeping Teams Intact, Harvard Business Review, December, 2013.
16. Peters, L. J., Managing Software Projects: On the Edge of Chaos, From Antipatterns to Success, Kindle eBook, Software Consultants International Limited, 2014.
17. Flyvberg, B., From Nobel Prize to Project Management: Getting Risks Right, Project Management Journal, August, 2006, pp. 5 – 15
18. Aiman-Smith,L., Bergey, P., Cantwell, A.R., Doran, M., "The Coming Knowledge and Capability Shortage," Research-Technology Management, Vol. 49, No. 4, July-August, 2006.
19. LaPlante, P. A. and Neill, C. J., Antipatterns: Identification, Refactoring and Management, Boca Raton, Florida, Taylor and Francis, 2005.
20. Ghazi, P., Moreno, A. M. and Peters, L. J., Looking for the Holy Grail of Software Development, IEEE Software, Jan/Feb, 2014, pp. 92-94.
21. Jensen, R., Don't Forget About Good Management, CrossTalk, August, 2000, pp. 30.
22. Grant, A. and Gino, F., A Little Thanks Goes a Long Way: Explaining Why Gratitude Expressions Motivate Prosocial Behavior, Journal of Personality and Social Psychology, Volume 98, Number 6, pp. 946-955, 2010.
23. Peters, L. J. and Stucki, L., A Software Engineering Graduate Curriculum, ACM 1978 Annual Conference Proceedings, ACM New York, NY, pp. 63-67.
24. Thamhain, H. J., Changing Dynamics of Team Leadership in Global Project Environments, American Journal of Industrial and Business Management, 2013, Number 3, pp. 146-156, 2013.
25. Thomas, J. and Mengel, T., Preparing project managers to deal with complexity – Advanced project management education, International Journal of Project Management, Volume 26, pp. 304 – 315, 2008.

Increase Team Cohesion by Playing Cooperative Video Games

Gregory S. Anderson, Brigham Young University
Spencer Hilton, Weber State University

Abstract. Team building activities such as collaborative video gameplay requires a collective effort by players to achieve a common goal. In a business environment, increasing cohesion can improve performance while in a military environment, increasing cohesion can affect morale and combat efficiency. This study measured group and individual cohesion factors with the result revealing that playing a cooperative video game with a minimal time and financial commitment makes it a viable team building activity to increase team cohesion.

Teamwork

Teams have become increasingly important within an organization [1][2] and can only be effective to the extent that team members work cooperatively with each other [3]. Whether it is a software engineering team, a military unit, an acquisition workforce, a sports team, or any organization in which a group of people cooperative to achieve a goal, effective teams are critical in order to achieve success. In order to cooperate, there needs to be a task aligned with a common goal and team members must feel connected to one another having some type of team building activity that promotes interaction [4]. Newman said that team building promotes "an increased sense of unity and cohesiveness and enable the team to function together effectively" [5]. Research has shown that cohesion is linked to team performance [6][7][8] and is considered one of the most important small group variables [9] with cohesion-performance being driven by goal or task commitment [10].

Team building requires group goals to be defined and tasks identified [11]. As team members struggle to define roles and requirements for the project, the group needs open communication to build trust. As the team works towards a common objective, the members develop social relations. When team members demonstrate a level of respect for peers, the foundation is laid to begin having a successful team and the group functions as a unified unit [12][13].

Team cohesion has long been considered by military psychologists to be a significant factor in small-groups [14]. The military contends that cohesive groups are more effective in combat situations, thus providing an advantage over their opponents [15]. Laurel Oliver said "the military maintains that cohesive groups engender effectiveness in combat situations" [16]. Tziner and Vardi said "a non-cohesive unit could lead to fatalities in artillery and tank crews" [17].

Team unity can be accomplished through a variety of team building activities [18] that improve a group member's knowledge about effective communication, group problem solving and teamwork, self-esteem, and organizational commitment. A common team building activity is the use of outdoor management education (OME) [19]. OMEs involve a wilderness experience often using rope and challenge courses. However, OMEs can be costly and time consuming. This article explores the less expensive and time-consuming alternative of team building by playing collaborative video games.

Pros and Cons of Video Games

The U.S. military is recognizing the advantages of using video game simulation in combat training by creating a virtual environment that more closely mimics reality, with realistic threats and having the ability to represent human interaction [20]. Simulations are life-like video games [21] and are helping soldiers from all over the world sharpen their fighting skills and prepare them for the forth-coming battlefield mental stress [22]. They are also bridging the gap between classrooms and real job skills and improving the learning process [23]

In the military, researchers have shown that there is a correlation between cohesion, morale, and combat efficiency [24]. Military cohesion has been defined as "the bonding together of members of an organization/unit in such a way as to sustain their will and commitment to each other, their unit, and the mission" [25]. Frederick Manning defines morale as "a function of cohesion and esprit de corps," and says it is necessary in combat since unit members rely upon each other in order to survive and succeed [26].

Until recently, video game studies might have been considered laughable [27], but an increasing number of studies are being investigated to determine the pros and cons of playing video games. As seen in Table 1, video games are a popular form of entertainment, but they are also a powerful learning tool [28] and are shaping the way we learn. Prensky said "Ever since Pong arrived in 1974, our kids have been adjusting or programming their brains to the speed, interactivity, and other factors in computer and video games, much as their parents the boomers reprogrammed their brains to accommodate TV" [29].

Effect	Description
Cognitive Performance	Video game play can improve short-term working memory, visual attention, mathematical decision making, and auditory perception.
Cooperative Play	Participants must work together to win the game.
Entertainment	Enjoyment of play.
Socially Therapeutic	Playing video games can help players relax, vent frustration, distract pain and help learn.

Table 1. Positive effects of video games

Pediatrician Dr. Benjamin Spock expressed the opinion that "The best that can be said of them is that they may help promote eye-hand coordination in children. The worst that can be said is that they sanction, and even promote aggression and violent responses to conflict. But what can be said with much greater certainty is this: most computer games are a colossal waste of time" [30].

Effect	Description
Addiction	Players may become game dependent.
Aggressive Behavior	There are theories that playing violent video games may be tied to aggressive behavior.
Physical Health Risks	Some video games could cause seizures.
Social Health Risks	Players may become socially dependent upon game play and be socially isolated.

Table 2. Negative effects of video games

Some of the negative effects have been identified in Table 2.

However, in his book "Everything Bad is Good for You", Steven Johnson made the argument, "The most debased forms of mass diversion – video games and violent television dramas and juvenile sitcoms – turn out to be nutritional after all" [30]. In 2003, James Paul Gee, a noted psycholinguistics researcher, said that video games are inherently social and that they have the potential to lead to active and critical learning. He went on to say that the real potential of games is "to get people to think, value, and act in new ways" [31].

The popularity of video gaming not only is perceived as a popular form of entertainment but is being researched as a tool for improving organizational training results. All teams are different and therefore a myriad of instructional strategies should be researched and implemented [32]. As organizations struggle to compete in a global economy the development of intellectual capital has become their most valuable asset [33]. Developing capital such as organization's workers involves the use of training to unleash the potential of human expertise [34] and improving the adult workforce.

In December of 2010, the Defense Acquisition University (DAU), a corporate university for the Defense Acquisition Workforce, launched the Department of Defense casual games site. Dr. Alicia Sanchez, Game Czar for DAU, said that the rationale for the site was a place for employees to play games that were related to the core competencies central to Acquisition jobs [35].

Full Spectrum Warrior is a video game based upon a U.S. Army simulation requiring the player to think like a professional solider in order to survive [36]. Prensky said "the US Military uses more than 50 different video and computer games to teach everything from doctrine, to strategy and tactics" [37]. Games such as America's Army offers a virtual basic training to develop, train, and educate U.S. Army soldiers. In 2011, game designer and author Jane McGonigal said, "Those who deem them [video games] unworthy of their time and attention will not know how to leverage the power of games in their communities, in their businesses, in their own lives" [38]. Video games are here to stay, and one must harness the power of the game play for the benefit of society. This paper demonstrates that one benefit of collaborative video games is increased team cohesion.

Measuring Cohesion

In order to measure team cohesion, one must first understand the correlated cohesion constructs. The Group Environment Questionnaire (GEQ) was used in this study to assess the level of cohesion achieved within a group. Researchers Albert Carron and Lawrence Brawley created the GEQ based upon assumptions that cohesion can be evaluated through perceptions of individual group members. The test identifies four constructs related through different task and social interactions as viewed through the eyes of the individuals about themselves and their team. The authors clarify that the model is a framework that serves as a guideline and should be used in its original content. However, as necessary, revisions are acceptable, including changes to words, the deletion of non-pertinent questions, and the addition of items that are more culturally meaningful to the study.

The GEQ is an 18-item questionnaire based upon Carron's conceptual model of cohesion representing four constructs. The model divides cohesion into two categories: group integration and interpersonal attractions to the group. The model then subdivides the two categories into 4 sub-scales by assessing the Group Integration-Task (GI-T), Group Integration-Social (GI-S), Individual Attractions to the Group-Task (ATG-T), and the Individual Attractions to the Group-Social (ATG-S). The GI-T and GI-S sub-scales represent the "us", "our" and "we" perceptions while the ATG-T and ATG-S sub-scales represent the "I", "my", and "me" perceptions.

Four test questions refer to ATG-T, five questions assess ATG-S, five questions assess GI-T, and four questions assess GI-S. Responses are in the form of a 9-point Likert scale based on strongly disagree (1) and strongly agree (9) with the higher score reflecting stronger perceptions of cohesiveness. Some items on the questionnaire were slightly modified as suggested by the instrument authors to represent the culture of this study. Since team cohesion is a multidimensional construct, all four components of team cohesion do not need to be present in order to show a degree of change in cohesion.

The instrument is based upon three fundamental assumptions: 1) Cohesion can be evaluated through the perception of group members; 2) The group satisfies personal needs and objectives, and 3) A group's concern to the group and members by focusing on task and social factors helping to create unity. As shown in Figure 1, the GEQ model identified four correlated constructs representing the task and social orientations as perceived through the group member about his/herself and about the team.

The "GI" represents group integration and "ATG" represents "attraction to the group." The "S" represents the social relationships within the group and how an individual views the group as a social aspect. The "T" identifies the individual's perception towards achieving a specific goal or objective [39]. The GI-T and GI-S represent the "us", "our", and "we" individual perceptions of the group such as the closeness, similarity and bonding. The ATG-T and ATG-S represent the "I", "my", and "me" individual perceptions of self and the motives to remain in the group.

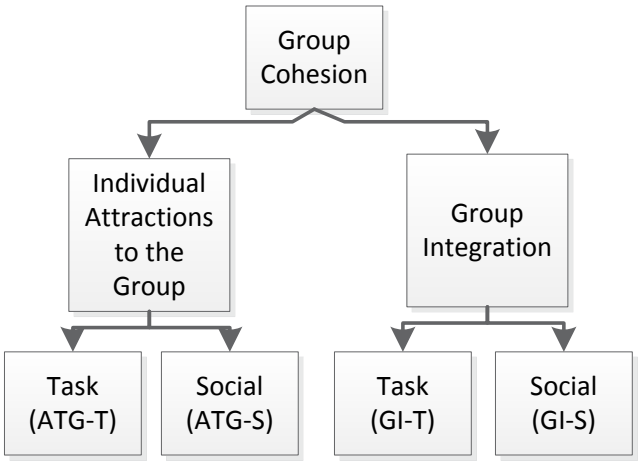


Figure 1. Factors defining cohesion

Although OMEs have been used in the past to increase team cohesion, this research shows that playing collaborative video games can also increase the cohesive sub-scale factors ATG-T, ATG-S, GI-T, and GI-S, thus resulting in a higher level of team cohesion.

Research and Methodology

This study introduces the use of cooperative video games as a tool to mimic the OME environment. The participant is removed from the worries of society by being immersed into video game world. The goal of the video game player is to work with team members to win the video game. The risk involved, when losing, is a state of emotional discouragement for having not succeeded. By using a cooperative video game, a single participant cannot win the game for the team. All members must cooperate, communicate, problem solve, and be committed to the team in order to have a chance at winning. When members do well, their self-esteem is buoyed.

With the environment resembling the OME, this study implemented a quantitative approach using a pretest/posttest design. Teams were randomly formed and assigned a length of intervention of either one or three weeks of game play with the intervention length ranging from one hour to six hours to play a collaborative video game. Like other forms of entertainment, video games were classified into genres. Although not completely standardized, a commonality has been identified from which new video game developers must consider while designing a game to be released [40] the genres (see Table 3) identify the style of game play [41].

The collaborative video game selected for this research was Halo 3. It was designed by Microsoft's Bungie Game Studio and has an ESRB rating of mature for blood and gore, violence and mild language. Halo 3 is an action game genre and is a first person shooter. One to four players participate on one of four teams thus creating a cooperative environment where team members must defend and protect each other against the enemy. If desired, four teams of four players can participate at one time playing against the other teams. Each team uses an Xbox 360 console networked to other consoles and competes against other teams for a specified number of rounds. A round is identified as the number of enemies killed. Players return to Earth to save mankind from the Covenant, an evil alien force. The multiplayer mode should be "slayer" which allows up to four teams of four players each to "rack up" a certain number of kills. The number of kills for each round should be at least twenty-five. The win/loss records were not kept. The teams selected for this study were similar in context and played as many rounds as possible within the one to two hour block time. Game play continued until the team's randomly assigned intervention schedule was completed.

Table 4 summarizes the descriptive statistics of the demographic and covariate variables tested in the analysis.

The 18-item GEQ was administered as a pretest to assess the participant's perceptions of group integration and interpersonal attractions to the group. The model is subdivided into two categories (Table 4) assessing the Group Integration-Task (GI-T), Group Integration-Social (GI-S), Individual Attractions to the Group-Task (ATG-T), and the Individual Attractions to the Group-Social (ATG-S). The participant responses are in the form of a 9-point Likert scale based on strongly disagree (1) and strongly agree (9) with the higher score reflecting stronger perceptions of cohesiveness.

Genre	Description
Action	Involves fast action and good hand-eye coordination.
Adventure	Exploration.
Arcade	Mimics early arcade games.
Combat	Fighting.
Driving	Simulated driving and racing.
First-Person Shooter (FPS)	Action genre from a first person perspective.
Multiplayer	Supports more than one game player simultaneously.
Puzzle	Solving problems, mazes, and puzzles.
Role Playing Game (RPG)	Storyline stressed over action.
Simulation	Mimics reality.
Sports	Traditional sports.
Strategy	Planning and resource management.
Third-Person Shooter (TPS)	Action genre from a perspective above and/or behind the player.
Trivia	Intellectual testing knowledge.

Table 3. Video Game Genres

Variable	Statistic
Age	>= 18 and <= 29 (n=56)
Gender	Female = 13% and Male = 87%
Group 1 Subjects	(n=29) 25 Male; 4 Female
Group 1 Average hours/week playing video games	10 hours
Group 2 Subjects	(n=27) 24 Male; 3 Female
Group 2 Average hours/week playing video games	16.11 hours

Table 4. Descriptive statistics

Sub-Scale	Description
Interpersonal Attraction to the Group ATG-T (task)	Individual's feelings about personal involvement in the group's task, productivity, goals, and objectives.
Interpersonal Attraction to the Group ATG-S (social)	Individual's feelings about personal acceptance and social interactions within the group.
Group Integration GI-T (task)	Individual's feelings about the closeness, similarity, and bonding within the team based upon group's task (playing Halo).
Group Integration GI-S (social)	Individual's feelings about the closeness, similarity, and bonding within the team based upon the group's social unit.

Table 5. Cohesive sub-scales

The one-week study consisted of one hour of cooperative game play for that day. The three-week study consisted of two hours of cooperative game play each week for three weeks, totaling six hours of intervention. At completion of the study a posttest was administered using the modified GEQ survey.

Results

The results of this study confirmed that playing collaborative video games increased team cohesion in every GEQ cohesive factor. Whether the groups played one hour or six hours, there was still an increase in team cohesion. This implies that playing collaborative video games as a group could potentially be a less costly and time consuming team building activity for a positive change in cohesion in an environment where teams are used.

H1: There was a difference in the team cohesion factor ATG-T based upon the intervention.	Supported
H2: There was a difference in the team cohesion factor ATG-S based upon the intervention.	Supported
H3: There was a difference in the team cohesion factor GI-T based upon the intervention.	Supported
H4: There was a difference in the team cohesion factor GI-S based upon the intervention.	Supported

Table 6. Summary of Hypotheses and Results

The results demonstrated support for the hypotheses in that cohesion was positively affected by playing collaborative video games (See Table 6).

The six hours of video game play did produce slightly higher cohesion (see Table 7) but the increase was marginal. The ATG-T and ATG-S, which measured the individual attraction to the task and social aspect, had slight increases in the gain score percentages. But one must consider whether or not the amount of game play to achieve that gain justified the intervention time.

Group 1 – One Hour of Video Game Play Intervention

	ATG-T		ATG-S		GI-T		GI-S	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
Pretest	6.62	1.30	4.86	1.04	5.48	0.90	5.00	0.86
Posttest	7.53	1.42	5.85	1.04	6.57	0.94	5.60	1.32
Positive Gain	**0.91**	**38%**	**0.99**	**24%**	**1.09**	**31%**	**0.60**	**15%**

Group 2 – Six Hours of Video Game Play Intervention

	ATG-T		ATG-S		GI-T		GI-S	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
Pretest	6.19	1.15	4.88	1.29	5.10	0.59	5.19	0.76
Posttest	7.38	1.45	5.98	1.36	6.30	0.91	5.63	1.03
Positive Gain	**1.19**	**42%**	**1.10**	**27%**	**1.20**	**31%**	**0.44**	**12%**

Table 7. Group 1 & 2 Pretest and Posttest Means and Gain Scores.

The one hour of video game actually received the same level of increase for the GI-T cohesive factor and scored a greater increase on the GI-S, meaning that if the ultimate goal was to increase the group member's perception of closeness, similarity, and bonding with the group, then only one hour of game play needs to be implemented to achieve the organization's goal.

Conclusion

Today's global economy requires that organizations constantly seek for ways to improve and to surpass their competition. A variety of strategies could be implemented to improve different aspects of the organization. If team cohesion could be strengthened, the result will likely be improved team performance. Organizations continue to search for mechanisms to improve teamwork by finding and implementing new methods for effectively accomplishing a task and increasing social capacities for individuals to handle problems. Strategies for improvement include making a team more cohesive so that the members are more committed, thus increasing productivity and performance.

In the military, unit cohesion is essential for a strong military force. In fact, it means more than being liked by others; it is a willingness to die for someone else. As there is a correlation between cohesion, morale, and combat efficiency, playing collaborative video games can increase team cohesion. This can result in military units improving morale and combat efficiency, and potentially increasing the rate of soldier survival and operation success.

Whether the organization is striving to improve performance or to improve soldier survival and operational success, this study concludes that it can be beneficial to have teams play collaborative video games even for as little as one hour to increase team cohesion. However, this is just scratching the "tip of the iceberg." Further studies are in the process that measure team cohesion after video gaming and after other endeavors to see whether the team cohesion obtained in video gaming actually transfers to the follow-on endeavor.

ABOUT THE AUTHORS

Gregory S. Anderson is an Associate Professor in Information Systems at Brigham Young University. Pre-academia, he has more than 15 years of industry software development experience. Prior to BYU, he was Chair of Computer Science for 8 years at Weber State University. He has a Ph.D. in Technology Management from Indiana State University, an MBA from University of Colorado – Colorado Springs, and a BA in Computer Science from Weber State University.

Brigham Young University
Marriott School of Business
Provo, UT 84404
Phone: 801-747-9787
E-mail: profganderson@byu.edu

Spencer Hilton is an Assistant Professor in the Computer Science Department at Weber State University. He holds an MBA, as well as a BA in Communication and a BS in Computer Science, from Weber State University. Prior to teaching, Spencer worked as a Software Engineer and Business Intelligence Analyst.

Weber State University
Department of Computer Science
Ogden, UT, 84408
Phone: 801-626-7929
E-mail: spencerhilton@weber.edu

REFERENCES

1. Johnson, Tristan E, et al. "Measuring Sharedness of Team-Related Knowledge: Design and Validation of a Shared Mental Model Instrument." Human Resource Developtment Interational 10.4 (2007): 437-454.
2. Salas, Eduardo, Nancy J Cooke and Michael A Rosen. "On Teams, Teamwork, and Team Performance: Discoveries and Developments." Human Factors: The Journal of the Human Factors and Ergonomics Society 50.3 (2008): 540-547.
3. Stewart, Greg L, et al. Team Work and Group Dyamics. Wiley, 1999.
4. Johnson, D W and F P Johnson. Joining Together: Group Theory and Group Skills. 6th. Prentice-Hall, 1997.
5. Newman, Betsy. "Expediency as benefactor: How team building saves time and gets the job done." Training & Development Journal 38.2 (1984): 26-30.
6. Ahronson, Arni and James E Cameron. "The nature and consequences of group cohesion in a military sample." Military Psychology 19.1 (2007): 9-25.
7. Senecal, Julie, Todd M Loughead and Gordon A Bloom. "A Season-Long Team Building Intervention: Examining the Effect of Team Goal Setting on Cohesion." Journal of Sport & Exercise Psychology 30.2 (2008): 186-199.
8. Carron, Albert V, Lawrence R Brawley and W Neil Widmeyer. Group Environment Questionnaire Test Manual. Morgantown: Fitness Information Technology, Inc., 2002.
9. Lott, Albert J and Bernice E Lott. "Group cohesiveness as interpersonal attraction: A review of relationships with antecedent and consequent variables." Psychological Bulletin 64.4 (1965): 259-309.
10. Mullen, Brian and Carolyn Copper. "The relation between group cohesiveness and performance: An integration." Psychological Bulletin 115.2 (1994): 210-227.
11. Levi, Daniel. Group Dynamics for Teams. Thousand Oaks: Sage Publications, Inc., 2007.
12. Tuckman, Bruce W and Mary Ann C Jensen. "Stages of Small-Group Development Revisited." Group Organization Management 2.4 (1977): 419-427.
13. Levi, Daniel. Group Dynamics for Teams. Thousand Oaks: Sage Publications, Inc., 2007.
14. Dion, Kenneth L. "Group cohesion: From "field of forces" to multidimensional construct." Group Dynamics: Theory, Research, and Practice 4.1 (2000): 7-26.
15. Ahronson, Arni and James E Cameron. "The nature and consequences of group cohesion in a military sample." Military Psychology 19.1 (2007): 9-25.
16. Oliver, Laurel W, et al. "A quantitative integration of the military cohesion literature." Military Psychology 11.1 (1999): 57-83.
17. Tziner, Aharon and Yoav Vardi. "Ability as a moderator between cohesiveness and tank crews performance." Journal of Occupational Behaviour 4.2 (1983): 137-143.
18. Senecal, Julie, Todd M Loughead and Gordon A Bloom. "A Season-Long Team Building Intervention: Examining the Effect of Team Goal Setting on Cohesion." Journal of Sport & Exercise Psychology 30.2 (2008): 186-199.
19. McEvoy, Glenn M. "Organizational change and outdoor management education." Human Resource Management 36.2 (1997): 235-250.
20. Sottilare, Robert A. "Improving Soldier Learning and Performance Through Simulation and Training Technologies." Army AL&T May - June 2005: 31-35.
21. Mitchell, Rebecca and DeBay Dennis. "Get Real: Augmented Reality for the Classroom." Learning & Leading with Technology 40.2 (2012): 16-21.
22. Mitrea, Ioan. "Learning Through Strategic Computer Games in Military Training." eLearning and Software for Education. 2013. 382-385.
23. Aldrich, Clark. Simulations and the Future of Learning: An Innovative (and Perhaps Revolutionary) Approach to e-Learning. Pfieffer, 2003.
24. Stewart, Nora K. "Military Cohesion," in War. Ed. Lawrence Freedman. Oxford: Oxford University Press, 1994.
25. Henderson, William Darryl. Cohesion: The Human Element in Combat. University Press of the Pacific, 2002.
26. Gal, Reuven and Frederick J Manning. "Morale and Its Components: A Cross-National Comparison." Journal of Applied Social Psychology 17 (1987): 369-391.
27. Adams, Ernest. Break Into the Game Industry: How to Get a Job Making Video Games. Emeryville: McGraw Hill Professional, 2003.
28. Prensky, Marc. "Computer Games and Learning: Digital Game-Based Learning." Handbook of Computer Game Studies 2005: 98-122.
29. Prensky, Marc. "Digital Game Based Learning: Computers in Entertainment (CIE)." Theoretical and Practical Computer Applications in Entertainment 1.1 (2003):
30. Johnson, Steven. Everything Bad is Good For You: How Today's Culture is Actually Making Us Smarter. Riverhead Trade, 2006.
31. Gee, James Paul. What Video Games Have to Teach Us about Learning and Literacy. New York: Palgrave Macmillan, 2003.
32. Salas, E., Burke, C., & Cannon-Bowers, J. What We Know About Designing and Delivering Team Training. In Kurt Kraiger (Eds.), Creating, Implementing, and Managing Effective Training and Development (pp. 234-259). San Francisco, CA: Jossey-Bass. (2002).
33. Marquardt, B., Berger, N., & Loan. P. HRD in the Age of Globalization. New York, NY: Basic Books. (2004).
34. Swanson, R. & Holton III, E. Foundations of Human Resource Development. San Francisco, CA: Berrett-Koehler Publishers. (2001)
35. Sanchez, Alicia. DOD Launches Casual Gaming Site. 1 December 2010. <http://science.dodlive.mil/2010/12/01/dod-launches-casual-gaming-site/>.
36. Shaffer, David Williamson, et al. "Video Games and the Future of Learning." WCER Working Paper (2005).
37. Prensky, Marc. "Digital Game Based Learning: Computers in Entertainment (CIE)." Theoretical and Practical Computer Applications in Entertainment 1.1 (2003):
38. McGonigal, J. Reality is Broken. New York, NY: Penguin Books. . (2011).
39. Smith, Joseph W. "The Effect of an Intervention Program on Cohesion with Ninth Grade Female Basketall Teams." Master of Science Thesis. Oregon State University, 1996.
40. Adams, E. Break into the Game Industry. Emeryville, CA: McGraw-Hill. (2003).
41. Bergeron, B. Developing serious games. Hingham, MA: Charles River Media. (2006).

Risk Management for Dummies

An Open Forum Article

Tom DeMarco, Atlantic Systems Guild

In what follows, I'm going to use an elaborate analogy to make a point about risk management of software projects.

Proceed with caution here: if the analogy works according to my nefarious plan, you're probably going to have to think very differently about how you manage risks on your next project.

If that thought worries you, you might want to stop reading here.

Here's the analogy: You are on temporary assignment in San Francisco, working for a boss who spends most of her time in the Los Angeles office. She calls you early in the week and says there is a must-attend meeting scheduled for Friday in LA. She'll be there, as will her boss, the CEO, as well as the CEO of your company's biggest client. And you're the show. You're going to have to put on a whiz-bang presentation of your new software suite, something that will knock their socks off, and secure a whole line of new and highly profitable business for the company. The best of it is that both you and she know you can do it, it's going to be a "piece of cake."

The meeting is scheduled for 11 a.m. Friday. There is an 8 a.m. flight from SFO Friday morning that gets in around 9 a.m., so you figure that will give you plenty of time to get to the office, even though it's twenty miles off in deepest, darkest Glendale. The flight is actually a little late getting in (damn these airlines and inefficient air traffic controllers!). You give the flight attendant a real piece of your mind. And it takes forever for your bag to come onto the carrousel (damn these inefficient baggage handlers). You rush out onto the curb and to your dismay see that there is a long line for taxis. You let the starter know in no uncertain terms that this is completely unacceptable. When you finally get into a cab it is after 10 a.m. The driver pulls out and is immediately caught up in traffic on La Cienega. The on-ramp to the 405 is jammed and he seems inclined to wait it out, so you tell him angrily to find another way. The guy agrees but doesn't seem to feel your urgency. It's nearly 10:20 a.m. by the time he gets onto the freeway. Traffic is moving at a snail's pace. You can't take your eyes off your watch. 10:25 a.m., 10:30 a.m. "This is a damn important meeting," you tell the driver. "I mean really important. You've got to get me there by 11 a.m. I'm absolutely counting on you." He just shrugs. Traffic grinds to a halt. "Well do something," you tell him, "and make it fast. Time is a wasting." He pulls off and gets immediately stuck in local traffic. You scream at him, "My meeting, dammit! It starts in fifteen minutes and you have got us nowhere. This is just totally irresponsible on your part." By the time you arrive it is 11:40 a.m. The client has left and everyone is furious at you. But of course it's not really your fault, you explain: "The airline, the baggage handlers, the cabbie, the traffic …"

What's wrong with this picture? You did everything by the book: applied pressure, expressed your annoyance at substandard performance, told off everyone who was messing up your schedule.

What's wrong is that you started too late. You could have flown in the night before, put up at the pleasant little hotel in walking distance from the Glendale office, had a leisurely breakfast and sauntered into the office a full hour before the meeting was to begin.

"Want to avoid being late? Be early."

I've spent much of the last thirty years looking at and counseling software projects that were in trouble. The trouble they were in varied from project to project, at least the causes varied. What was the same in all the projects was that they were late. My clients wanted to know what to do to put them back on schedule, but they usually knew in their hearts that that was no longer in the cards. Their fallback position was they wanted to know what had made the projects so late. I would do my best to lay out the causes. But now, looking back at all of them as a whole, I can see that the real reason they were late finishing was that they started too late. All of them.

Can I really assert that the reason projects finish late is that they started late? All late projects? Isn't it at least possible that some of them made mistakes along the way, frittered away essential time, or lost effectiveness in mid course, and thus caused lateness in what was otherwise a perfectly doable project? Well, I guess that is possible; it's just that I've never seen it happen.

An elementary school that I pass on a walk into town has a marquis in front that reads, "Want to avoid being late? Be early." But Tom, you might object, We're trying to be early, that's what governs the methods and approaches we use. And my answer would be, Yes, but how early are you trying to be? Are you trying to be months early, or are you only trying to be minutes early? A project that must be done by January two years from now needs to be run on a plan that gives it a highly realistic better than 50-50 chance of being completely done six months before that. Anything else and you're not doing risk management. A project that starts too late to finish really early is one on which no meaningful risk management is possible. Your best tactic is to cross your fingers.

ABOUT THE AUTHOR

Tom DeMarco is a Principal of the Atlantic Systems Guild, a New York and London-based consulting practice. He is the author of thirteen books, most of them about the high tech workplace and its denizens, but also including two mainstream novels and a collection of short stories. His most recent business book is about organizational culture. It's called Adrenaline Junkies and Template Zombies — Patterns of Organizational Behavior, published by Wiley in the US and Hanser Verlag in Europe. A third edition of his classic work, Peopleware: Productive Projects and Teams (with co-author Tim Lister) will come out in July, 2013. His most recent work of fiction is a bit of science fiction: Andronescu's Paradox.

Phone: 207-236-4735
E-mail: tdemarco@systemsguild.com

Upcoming Events

Visit ‹http://www.crosstalkonline.org/events› for an up-to-date list of events.

COMSNETS 2015
7th International Conference on COMmunication Systems & NETworkS
Bangalore, India
6-10 January
http://www.comsnets.org

15th System-of-Systems Engineering Workshop
El Paso, Texas
27-30 January, 2015
http://www.itea.org/share/conferences-and-workshops

2015 International Symposium on Code Generation and Optimization
San Francisco, CA
7-11 February, 2015
http://cgo.org/cgo2015

20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming
San Francisco, CA
7-11 February, 2015
http://ppopp15.soe.ucsc.edu

3rd International Conference on Model-Driven Engineering and Software Development
Angers, Loire Valley, France
9-11 February, 2015
http://www.modelsward.org/

HotMobile 2015: the 16th International Workshop on Mobile Computing Systems and Applications
Santa Fe, New Mexico
12-13 February, 2015
http://www.hotmobile.org/2015

International Conference on Computing, Networking and Communications (ICNC 2015)
Anaheim, Ca
16-19 February, 2015
http://www.conf-icnc.org/2015

International Conference on Software Quality
Long Beach, CA
9-11 Mar, 2015
http://asq-icsq.org/index.html

ETAPS 2015 - 18th European Joint Conferences on Theory and Practice of Software
London, United Kingdom
11-19 April, 2015
http://www.etaps.org/2015/

WICSA 2015 – 12th IEEE Conference on Software Architecture
Montreal, QC, Canada
20-24 April 2015
http://www.computer.org/portal/web/conferences/calendar

Bridging the Gap:

Software Engineering Education and Training

I seem to be a perfect fit for the BackTalk column in this issue. Way back in 1977, I was a (oh so) young Sgt. in the AF, assigned to Keesler AFB. The section I was assigned to teach was the "Intro to Computer Processing" course. We used a Hughes 407L (feel free to Google it – it was old even then). For the next three years, I taught "technical training." I returned to Keesler again in 1983 and taught another three years – this time teaching Ada in the STARS program (Software Technology for Adaptable, Reliable Systems). We were supposed to be conducting "training" – but I am afraid that we ventured into the educational arena.

What's the difference? It's a HUGE difference. Education involves teaching theory and history. It's all about improving your knowledge and making you more intelligent. A crucial difference is that education is not about a job skill. It's often said that an educated person is more employable, but education is not about just getting a job. Training, on the other hand, is about a "skill." Its sole purpose is to transfer practical information and skills to make you more employable. A skill teaches you repeatable tasks that you master to learn your craft. Education is about thinking.

Back in 1983, I found it impossible to "train" software engineers. I suppose there are some skills appropriate to software engineering, but coding in Ada (or any language) is a tiny, tiny part of "software engineering." Coding might be easy. Software Engineering is hard. It's more about education – the theory.

In 1986, I was lucky enough to get an assignment at the USAF academy. I officially moved from "training" to "education" - different type of students, different goals. While I expected my students to be able to code, I was more concerned that they appreciated the differences between a binary tree, a 2-3 tree, a red-black tree, ….. you get the idea. Since 1986, I have been involved in education almost continually (except for a 12-year break as a consultant. Which is where I myself learned a lot. But I still taught college part-time.)

I am currently teaching college again – and don't ever plan to quit. I seem to be a good professor – my students appear not to dislike me too much, and based on tests and projects, my students seem to learn a bit, too.

But what do they learn? That's the dilemma. We follow an ABET-approved curriculum, and we collect enough metrics to ensure that we are meeting our outcomes and objectives. We have about 43 semester hours of computer-science related material plus the required fine arts, natural and physical sciences, math, English, political science, writing and

speaking skills, etc. A typical, well-rounded college education. Not training, education.

Education. That's what a college/university does. I like to think I produce Computer Scientists and Information Technologists that rank right up there with the best of them. So, after four years of nurturing critical thinking skills in such areas as data structures, operating systems, software engineering, information security, discrete math and analysis of algorithms, we've done what we are supposed to do in terms of education. Our students proudly walk across the stage, wave their diplomas to adoring and proud family, and…well, now it's YOUR job to train them. In the Air Force, we called it "On the Job Training" (OJT). They need LOTS of OJT.

Here's why – most students consider 1,000 lines of code a "large" program. They write a program, run it under a single set of test conditions one time, and receive a grade. No realistic configuration management is needed, nor risk management either. They have seldom reused code, nor had to worry much about interfacing with legacy systems. They typically get all of their requirements on a single sheet of paper (sometimes it's actually two-sided!) They probably know Java and C++. Never had serious user interaction, other than an occasional interaction with a professor. I'm not saying this is bad – let's face it, it's about all you can do during a four-year college career. We do the education (and we do it well). You take our educated graduates, add some training, and make productive developers out of them.

So – how are you doing with my bright young crop of educated computer scientists after you hire them? How do you facilitate converting their knowledge and intelligence into usable software development skills? Do you give them a mentor? I mean, not just assign them to a supervisor – but really assign them a effective mentor? One who still feels the excitement and joy of developing software? One who know some of the latest tools and techniques? Do they have time in their schedule to talk with their mentor weekly (daily would be better)? Does the mentor have good people skills? Are there weekly or monthly "brown bags" for them to learn new skills (or sharper the ones they already have)?

How about their working environment? Do they work as part of a team? Experience has shown again and again that working with a peer in developing software is one of the best ways to bring new developers "up to speed." Except for rare group projects – this is not a skill or environment that they have learned during school. In fact, more colleges and universities discourage group work – it's much harder to assign a grade unless each student shows me how well they individually have mastered the theory.

Do you have a way to help them deal with not only the frustration of incomplete requirements and users who don't even seem to know what they want? Trust me – I stay in contact with a lot of my former students –incomplete requirement issues seem to bother many of them a lot.

I hope I speak for the educators out there – we're doing the best we can. We are working to educate our students. Once they graduate – it's your job start their training and expand their job skills. Determine what you want your developers to be capable of – and see what education they are bringing to the job. These new developers want to bridge the gap between their education and the job skills you require – it's up to you to facilitate their training. Different people will have different backgrounds. Design your mentorship/ training programs accordingly – one size does not fit all.

David A. Cook
Stephen F. Austin State University
cookda@sfasu.edu