The Emerging Role of Software Testing in Curricula

Tara Astigarraga[1], Eli M. Dow[2] , Christina Lara[1], Richard Prewitt[2] , Maria R. Ward[3]

IBM Systems and Technology Group

{asti, emdow, chrslara, prewitt, mrward}@us.ibm.com
[1]Tucson, Arizona 85744-0002
[2] Poughkeepsie, NY 12601-5400
[3] Austin, Texas 78758-2493

Abstract— **In order to produce quality products, companies require new engineering students that have good problem solving, debugging, and analysis skills. Many graduates enter the work force with exceptional development skills, but lack proficiency in test, debugging, and analysis skills. This is in part because academic curricula emphasize development at the expense of teaching software testing as a formal engineering discipline. The majority of curricula today emphasize the initial phases of a development life cycle, namely: requirements gathering, architecture design, and implementation. The skills which are retained in this area of test are often learned ad-hoc while working on solutions for an implementation-oriented course. The lack of formal test education among graduates forces industry to spend substantial resources to properly educate graduates in the art and science of software testing. The contribution of this paper to the literature includes an evaluation of software testing as an industry profession, a survey of current curricula guidelines, a survey of software testing education in practice today, and a discussion of ongoing efforts to advance the status of software testing in academic curricula through a novel, crowd-sourced, industry-expert, approach to software test education.**

Keywords- Software Testing, Education, Software Engineering, Courseware, Computer Science Education

I. INTRODUCTION AND MOTIVATION

In order to produce quality products, companies require new-hire candidates that have good problem solving, debugging, and analysis skills. In software engineering and computer science many graduates enter the work force with exceptional development skills, but lack proficiency in test, debugging, and analytical skills. This is in part because of the current academic curricula emphasis on software development at the expense of teaching software testing as a formal engineering discipline.

The majority of curricula today emphasize the initial phases of a development life cycle, namely: requirements gathering, architecture design, and implementation. In some instances there are courses devoted to each individual aforementioned phase, though the bulk of academic curricula today place a heavy emphasis on design and implementation. A search of the 2009 computer science degree plans for two of the top CS universities found only one test course included in one program [CMU] and none in the other [STAN]. Test skills are often learned ad-hoc while working on solutions for an implementation-oriented course. The lack of formal test education among graduates forces industry to spend substantial resources to properly educate graduates in the art and science of software testing.

As computer systems grow in size and complexity, testing is becoming increasingly difficult. Advances in software engineering principles, methods, and tools have made programmers more prolific than ever before, however tools for assisting quality assurance have not kept pace. The test engineering role is becoming increasingly important and is growing as a unique engineering discipline of its own. As society depends on technology for day-to-day survival, software dependability has become critical in all aspects of life. The need for better software testers becomes even more obvious as the need for continuous availability in enterprise systems grows.

A study conducted by the National Institute of Standards and Technology (NIST) in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed [NIST]. For example, the 1997 crash of Korean Airlines 747 in Guam resulted in 200 deaths was attributed to a faulty software configuration [NTSB]. In addition, there were nearly 30,000 deaths and 600,000 injuries attributed to medical device software failures reported between 1985 and 2005 [CDRH]. Faulty software in anti-lock brakes forced the recall of 39,000 trucks and tractors and 6,000 school buses in 2000 [CBS]. The $165 million dollar Mars Polar Lander Space Probe was destroyed in its final descent in 1999, likely because its software shut the engines off 100 feet above the surface [CBS]. Additional examples are not difficult to find in existing literature.

An age of increased attention to responsibility, both corporate and personal, is upon us. If software engineering is to be held to the same rigor as other engineering fields, such as civil or mechanical engineering, even more effort needs to be placed on rigorous testing. The remaining sections of this paper introduce contributions to the literature which include an evaluation of software testing as an industry profession, a survey of current curricula guidelines, a survey of software testing education in practice today, and a discussion of

ongoing efforts to advance the status of software testing in academic curricula through a novel approach to the creation of software testing course content.

A. Software Testing as a Profession

A former editor of ST&P, a software testing trade magazine, who served as chairman of the Software Test & Performance Conference recently reported that from his analysis of the 2007 U.S. Bureau of Labor Statistics, there are 349,140 people within the US who are identified as being software testers (specifically that population was identified as "computer software engineers, system software" which was the only job category to include "software testing" in the description). Interestingly there were an additional 394,710 persons identified as "computer programmers", as well as 495,810 self-identified "computer software engineers, applications". It is reasonable to assume that members of the latter group spend at least some portion of their time performing software testing roles [DANI].

To examine these figures and place a low estimate on the percentage of software workers principally doing a job role citing software test explicitly in the description, it is concluded that 28.16% of software developers are testers nationally (349140/(890520+349140), the number of testers divided by the total number of developers and testers). We believe, as does the author of the citation, that as agile methodologies such as test-driven development become more prevalent in industry the number of software graduates performing test roles in a professional capacity will continue to increase.

From the evidence presented, a significantly large portion of the software development population is performing test roles as their professions. Consider for a moment that many of software testers were at one point interviewed or selected for their job by managers or other organizational leaders who are themselves uneducated in the software testing discipline. It seems quite possible that organizational leaders will often recruit software engineers into test positions when they do not themselves possess a basic understanding of the test disciplines and recommended test practices.

1) Professional Software Testing Societies

An interesting fact regarding the software testing profession is that there are relatively few professional software testing societies. The number of societies, though scant today, remains an improvement over the not so distant past when none existed. The only example of a professional software testing society of reasonable size which the authors have been able to locate is the Software Test & Performance Collaborative. This organization contains more than 50,000 members, and offers training, education, and conferences [STP].

2) Software Testing Publications

Similarly, there are relatively few publications oriented specifically at the software testing community. Consider that there is no dedicated ACM or IEEE Journal focused specifically on software testing. In so far as the authors of this paper can determine, the closest thing to a dedicated testing journal is the ACM special interest group in software engineering series proceedings entitled "International Symposium on Software Testing and Analysis" (ISSTA) [PORTAL].

In general, there seems to be a lack of professional organizations, and publications in the software testing discipline as compared to other fields of software engineering such as design or development. Professional journals, as well as other professional publication vehicles, are an excellent means for conveying best practices and innovation in those fields. It can be surmised that it is difficult for academia to shape their curricula without such venerable means describing the state of the art.

B. Effects of Industry on Software Testing Education

One is hard-pressed to think of a software-producing organization, or software customer, that would not benefit from improving how well software engineers are educated in the science of software testing as part of their university training. Industry recognizes, that all too often, software engineers hired directly from universities have had no formal training in software testing. The impact to software development organizations is lower productivity, additional training costs, and a lower quality of testing, which ultimately results in higher development and support costs. To date, in industry, many organizations are forced to take explicit action to train new hires in the basics of test engineering. The next sections examine how industry is shaping software testing in academia in order to alleviate some of this burden.

1) Industry Demand for Better Software Testing Education

One of the simplest ways an industrial entity can provide support for increasing software testing education is to voice their support directly. Consider the remarks from the following companies.

Jamie M. Thomas, Vice President, IBM Rational Product Development, Delivery, and Customer Support has said: "In order to meet the needs of the business, software delivery organizations must continue to mature their focus on the life cycle of software. This means that academic programs will need to expand their reach to disciplines like requirements management and quality management. In the context of quality management, focus on the testing discipline is absolutely critical both for software in embedded systems and traditional IT systems."

John Thompson, president of Mobile Collaborative Education Consulting, has expressed that companies have realized a crucial need for hardware and software vendors to conduct extensive system level testing before product release. He has further commented on the growing requirement for universities to develop good testing and verification curricula at the undergraduate level. In addition he is outspoken regarding his views that software testing is a viable professional career choice for aspiring college graduates.

Dr. James Whittaker, Director of Test Engineering at Google, former Microsoft architect, and former professor of Computer Science at the Florida Institute of Technology has expressed his belief that the typical few lectures on software testing during a semester-long course is not adequately teaching software testing. He has said "If universities paid

more attention to testing they'd probably find a lot more partnerships with industry and certainly more jobs for their graduates".

These remarks from industry experts underscore the need for better education and standardization of software testing. Let us now look at a common industry method of defining educational standards, certification.

2) The Impact of Industry Certification on Academic Curricula

Several certification programs exist to support the professional aspirations of software testers and quality assurance specialists. Unfortunately, no certification currently known to the authors of this paper actually requires the applicants to demonstrate their software testing ability. In fact no two certifications are likely to be comprised of the same accepted body of knowledge.

Certainly there are no universally accepted certification programs for software testing. Furthermore, the certifications that do exist are likely to be unknown to many academics and industry professionals alike. Programs such as the "Certified Software Quality Engineer" from the American Society for Quality, the Association for Computing Machinery "Certified Software Development Associate" and "Certified Software Development Professional" curriculum, as well as the International Software Testing Qualification Board's certificate in software testing tend to focus on terms and definitions rather than practice as shown in our empirical investigation into the examination of their outline materials. Academia would do well to standardize, at minimum, on teaching those very same terms, obviating the need for such certifications. Consider as further evidence, the following identified certifications shown below:

a) The American Society for Quality Certified Software Quality Engineer Certification

The American Society for Quality's Certified Software Quality Engineer certification has a 27 hour review course for the exam, which includes only1-2 hours to cover software testing. The remainder of the time is spent discussing standards, process models, vocabulary, and metrics. The certification examination itself covers few skills, and is regarded by some as having little depth. As a justification for these claims, to the best of our knowledge, the examination consists of 160 questions to be answered in a duration of 4 hours. The exam is said to contain approximately 10 questions on software test (and are of a general nature) [ASQ]. Based on discussions with experienced software development project planners, and testers, these values proportionally do not represent the actual amount of effort spent on testing and development in industry.

b) The ACM Certified Software Development Associate (CSDA) Certification

The ACM CSDA is intended for graduating software engineers and entry-level software professionals and serves to bridge the gap between your educational experience and real-world work requirements. The CSDA is the first step towards becoming a "Certified Software Development Professional (CSDP)", which is discussed in the next section [CSDA].

Though the CSDA does not call out a specific ratio of test-oriented question given on the examination, the number of test-specific sections (namely "Testing" and "Quality") spread across the four high level modules that comprise the course, making up less than 14% of the total sections covered (2 sections, from a total of 15 that comprise the course outline)[CSDA2]. From the 15 sections covered in the practice CSDA exam, it would appear only two, "Software Testing" and "Software Quality", are relevant to this discussion [CSDA3].

c) The ACM Certified Software Development Professional (CSDP) Certification

From an investigation into the CSDP certification materials online [CSDP1] it is determined that software test constitutes 5-17% of the exam, while issues relating to software quality fall somewhere between 6-8%. This yields a combined range of 21–25%, see Figure 1 for a complete breakdown by component. By comparison, software design is specified at minimum to be 22% [CSDP2]. In the syllabus for that CSDP, Module 5 entitled "Software Testing" covers the following three items: "Software Testing Overview", "Test Types", and "Test Design" [CSDP3]. This seems to omit entirely the science of test execution, test artifacts and numerous other things, which are core to software testing in practice. Module 11, entitled "Software Quality", covers: Software Verification and Validation: Software Quality Assurance, and Data Collection.

Given the increased content covered for this examination as compared to the previously mentioned CSDA, it is likely that relatively less emphasis will be placed on software testing than the CSDP shown previously. The sections relevant to this discussion further break down as follows:

Software Testing

A. Software Testing Fundamentals (I)
B. Test Levels (C)
C. Test Techniques (C)
D. Human Computer User Interface Testing and Evaluation (C)
E. Test-Related Measures (C)
F. Test Process (C)

Software Quality

A. Software Quality Fundamentals (I)
B. Software Quality Management Processes (C)
C. Software Quality Practical Considerations (C)

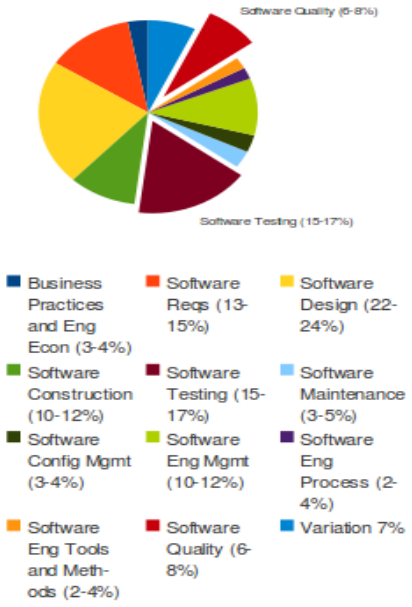

Figure 1. CSDP Exam Questions By Component.

According to the information contained on the sample examination page cited previously, items are classified with a notation of (I) or (C). The former is intended to "represent someone who has basic capability to perform in a knowledge area, but needs help, supervision, and review to do so like an apprentice" while the latter demonstrates skills which are "expected of a typical, experienced, professional they work largely on their own." It is alarming that software testing fundamentals are covered at a level expected of persons needing supervision or review.

d) Certified Tester Foundation Level Syllabus (2007)

The Certified Tester Foundation Level Syllabus is published by the International Software Testing Qualification Board and provides an in-depth curricula outline, which ultimately leads to the ISTQB Certificate in Software Testing Exam. The introductory outline is divided into six major chapters [CTFLS-1]. Each chapter contains sub-sections and provides outlined information aimed at helping students learn, understand, and apply the materials. Original analysis on the six major chapters of the ISTQB certifications is outlined in Table 1.

TABLE I. ISTQB CHAPTERS

Title	Minutes	Total Curricula %
Test design techniques	285	33
Test management	170	20
Fundamentals of testing	155	18
Testing throughout the software life cycle	115	13
Tool support for testing	80	9
Static techniques	60	7

Overall, this is an exceptional introductory outline, however despite the existence of high-level bullets on many key topics of test engineering, the details, industry experience, examples and lab exercises appear nonexistent from a cursory search. To exist as true class materials, a professor with in-depth industry and hands-on experience to help bridge the gaps and make the materials tangible and relevant for students should be required. It should also be noted that the syllabus outline itself is not a complete work upon its own and is meant to be used in conjunction with other materials, standards and books as mentioned in its appendix sections. The syllabus is intended to culminate in the certification process (ISTQB Certification) associated with the authors other works.

Further, the ISTQB exam entry criteria suggest that the materials are suitable for candidates with at least a minimal background in either software development or software testing, for example "6 months experience as a system or user acceptance tester" [CTFLS-2]. It is also strongly recommended that certification candidates complete a course that is accredited by the ISTQB. Such a course contains material that is precisely what is recommended for inclusion as part of the core curricula in undergraduate software engineering degrees. This certification provides value and standardization in the international test engineering industry in complement with other education methods and on-the-job training or experience, but is not a complete entity within itself.

3) Results of Industry Certification Survey

The net result of a survey of these industry certifications shows them to be relatively inconsistent across certification, and fairly limited in practical demonstration of software testing capability. There is little surprise why academia has not widely embraced the guidelines proposed by industry certifications.

II. Analysis of Current Curriculum Guidelines

The term "software engineering" first appeared in the 1968 NATO Software Engineering Conference. Since then, it has continued as a profession and field of study dedicated to designing, implementing, and improving software that is of higher quality, more affordable, maintainable, and quicker to build. Since the field is still relatively young compared to its sister fields of engineering, there is still discussion around what software engineering actually is, and if it conforms to the classical definition of engineering. It has grown organically out of the limitations of viewing software as just computer programming [SEDEF].

Perhaps some of the reason software testing is not rigorously taught has to do with conceptions of engineering in general. Consider the following definition of engineering: "Engineering is the discipline, art and profession of acquiring and applying technical, scientific and mathematical knowledge to design and implement materials, structures, machines, devices, systems, and processes that safely realize a desired

objective or inventions" [ENGDEF]. Note that this definition specifically calls out the word implement, yet only implies the need for test in the latter part of the definition.

As a second example, let us look at The American Engineers' Council for Professional Development (ECPD, the predecessor of ABET) definition of engineering: "The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property" [SCIENCE]. This definition contains an implicit emphasis on development by its foremost positioning in the sentence. The definition here includes words about "intended function", when testing is essentially the art of uncovering unintended function or malfunction. Only in the last clause of the definition is there mention of safety to life and property. We will concede that from the previous definition, when one is engineering a physical structure it may be easier to understand the implicit need for test.

What can be observed from common definitions of software engineering specifically? Consider this definition: "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software" [SEDEF]. This definition contains an explicit reference to development with no mention of test. It may be argued that in strict software engineering terms, "development" subsumes design, writing software, and testing, but that view belies the truth. The connotation present when asking a professional developer or student of the software engineering discipline to develop software typically does not immediately engender thoughts of test variations, but rather the act of authoring software. In fact in industry job postings almost universally explicitly differentiate the terms "develop" and "test" such that any implicit intended meaning is diluted by the colloquial use. Software authorship and test are both parts of the development process in the realm of academia. In practice development is strictly reserved for the former.

Perhaps if the definitions of software engineering were altered to make testing an explicit component therein, it may lead to adjustments in thinking about software engineering curricula.

Semantics aside, no evidence of a Bachelor's-level degree program in the United States specifically for software testing has been found during the course of these investigations. Only a few universities grant Masters or Doctoral degrees in Computer Science or Software Engineering for software-testing related work. However, within the top ranked schools, continuing education courses and test certification programs do exist. An example of one of these universities is the University of Washington, which has two certification programs in Software Testing and Software Test Automation. Further analysis on these findings can be found in section 3.4 "Institutions Emphasizing Software Testing Education".

III. Analysis of Current Curricula in Practice

In order to get a sense of how much emphasis on testing exists in curricula today, a survey of the course catalogs and requirements of 27 of the top ranked computer science schools as listed in the US News and World Report on-line magazine [EDRANK] was performed. This section presents an analysis of that survey.

A. Survey Methodology

Our survey of the current curricula was based on 27 of the Computer Science programs ranked by the US News and World Report magazine as being the among the "Top 20" ranked programs in the country. In order to complete this analysis some assumptions and judgment calls were made based on the information available on-line. In some cases, only partial course catalogs and descriptions were available for analysis. During the review of each program, courses that were hardware-related were excluded from the results. The authors acknowledge that this is not a purely scientific survey, but every effort was made to ensure that the data collected was accurate and consistent with the study criteria. In cases where the results were not clear, a consensus was made with the other authors before classifying the program data. In addition, emails were sent to the department chairs or designated course advisors for confirmation of the findings presented here. Although responses were not received from every school, every attempt was made to validate these survey results.

With these basic understandings in mind, the primary task was to identify and access the school web site through the School Rankings article. The initial search was for specific degree programs (either undergraduate or graduate) with a software testing focus. From here, granularity was increased down to the course level by identifying courses dedicated to software testing and then any courses which included the word "test" in the course description (excluding any course with a focus on hardware). Next a search for any other software testing or software quality-related academic activities was performed. In some cases, this led to findings of research work being done, continuing education courses, and test certification programs. This information is documented in Appendix A at the end of this document.

B. Survey Data

Appendix A presents the raw results without critique or analysis. The following section will present an interpretation of these data points.

C. Analysis of Survey

Based on the ST&P survey findings that approximately 28.16% of all software developers are software testers, one would expect that the top-ranked computer science programs would reflect a similar percentage of software test-related courses being offered to prepare these computer science students for their professional careers. In fact, the percentage of courses dedicated to software testing from these top-ranked

programs was meager (less then 5% in the majority of cases). Several schools offer only a single dedicated software test course. The total number of software-related courses with the word "test" in the description was also under 5% of the total course offerings. Based on these findings, it is clear that the education being offered to computer science students is not on par with the demand for quality software testers in the industry.

The trends discovered in the survey results show that there was little or no test content in undergraduate coursework. Courses with a test component tended to be in the software engineering or basic programming courses with slightly more emphasis in the graduate and research programs. Because of this level of testing education, the majority of computer science graduates are unlikely to know anything about system level testing, testing at scale or combinatorial testing, which is clearly a hard science to teach or pick-up on in a classroom. Students will have only a basic understanding of the techniques they used to test their own course projects. The net result is that universities in many ways are paying only lip service to testing as a core competency.

It is our assertion that professors at major leading universities (which arguably set the academic trends in education) are likely to have never done any software testing professionally for a substantial period of time. Although no data is available on the cause, this could be because software testing work is often not seen as glamorous or intellectually stimulating to the average Ph.D.-level researcher who might rather pursue novel development-oriented challenges instead. Encouragement in support of more testing education at a local university has been attempted, but remains largely unsuccessful due to the more prevalent focus on research at that particular institution. The curriculum planners in this case failed to realize the potential for challenging research in such areas as coverage modeling and combinatorics.

There may very well be some expected form of rampant cognitive dissonance among faculty who have read this paper up to here. They can appreciate all of the statements in support of more test education, yet often revert to the position that software engineering is predominantly intended to teach software creation (vis a vis programming) and the associated practices needed to practice the art of software creation (requirements gathering, UML, object-oriented design, agile or waterfall methodologies, and ultimately a demonstration of the ability to write software). When in fact, through the evidence marshaled thus far, it is clear that many who graduate with software engineering degrees become career testers. It remains difficult to endear the gravity of this concept to academics.

D. Institutions Emphasizing Software Testing Education

This section would be remiss if no attempt was made to present examples of university settings from the selection of survey schools that are providing some emphasis on software testing education. Consider for example the University of Washington, which has two certification programs in Software Testing and Software Test Automation. These programs are offered as continuing education courses. Though these courses serve a purpose in the test profession, from a reading of the curriculum and program, neither requires students to demonstrate software testing skill and knowledge. In addition the programs are offered to students of all skill levels. Although these continuing education programs are developed and offered as a partnership with the University of Washington, their purpose is not to complement the computer science education of the university students, but rather to help people in industry improve their skills. Communications directly with the university indicate that the continuing education program is intended for people in industry looking to improve their career skills and is not targeted at, nor generally taken by, undergraduates.

Also worthy of interest is the fact that these continuing education certification programs are developed through advisory boards that help define, review, and update the program based on the current industry needs. These boards are made up of some university faculty, but mainly industry experts with experience in the area of testing. For example, the Advisory Board for the Software Testing Certification program is made up of a combination of 17 software test managers, quality assurance managers, test architects and other experts from companies such as Microsoft Corp, Boeing, Safeco Insurance, Construx Software, and many others but only two university faculty and extension program members. This is evidence of the growing need in industry to improve the overall software testing education [WSU1][WSU2].

Based on our findings, there were four institutions from the top ranked programs that provide some emphasis on software testing education. At the Georgia Institute of Technology, a specialization in software engineering can be obtained simply by taking two graduate level software engineering courses. The University of Pennsylvania offers a Ph.D. program with a software engineering focus. And finally, the University of California at San Diego and the University of Maryland offer graduate software engineering research projects with a focus on testing and analysis.

E. Summary of Empirical Data Analysis

Based on our empirical study, software testing education for undergraduates has been shown to be extremely limited. In what seems to be the common theme, a course advisor at a top school in California confirms that they do not offer any classes or programs specifically focused on software testing, but to do well in class, thorough testing of projects by the students is required. Students will often set up their own test files and small frameworks with teaching faculty providing some guidance and examples. Students are encouraged to improvise their own testing strategies in order to thoroughly test their code and to excel in the software and systems courses. This implies that the students who are able to devise their own testing strategies are more likely to be successful in their coursework. This may lead to a satisfactory grade in class, but does not provide a good foundation for the level of software testing skills required in industry to develop and deliver well tested, quality products.

As indicated earlier in this paper, the motivation for improving test education is being driven by the increasing

number of accidents, deaths, equipment failures, and financial losses that are being attributed to software failures today. It is clear that many industries are being impacted by these problems. More focus on software testing in curricula is needed to reverse this trend. The case study of the University of Washington certification program shows that industry is indeed interested in reversing this trend and is willing to participate by sitting on advisory boards, developing course material, and even teaching the courses. This is a step in the right direction, but it is surmised that a more preemptive approach would be to improve the current curricula in practice to increase the focus on software testing in credit-based courses and increase the student's proficiency in test, debugging, and analytical skills. It is expected that improvement in these trends will occur when academic curricula starts recognizing software testing as a formal engineering discipline.

IV. PRIOR WORK

It should go without saying that others have worked on transforming test education in the past. This section serves to highlight some of the better work in the field, which may contribute substantially to new curricula requirement consensus.

A. IEEE & ACM Joint Task Force on Computing Curricula

In December 2001, The IEEE-CS Curriculum Development Committee in conjunction with the Association for Computing Machinery (ACM) published the Computing Curricula 2001 (CC2001), which outlines curricular guidelines for undergraduate programs in computer science. The goal of the CC2001 was "to review the Joint ACM and IEEE/CS Computing Curricula 1991 and develop a revised and enhanced version for the year 2001 that will match the latest developments of computing technologies in the past decade and endure through the next decade" [CC2001-1].

The CC2001 provides a list of 47 detailed course descriptions for various curriculum models. After reviewing all class titles, outlines, and materials it became obvious that while test was occasionally listed and mentioned as sub-bullets in some of the coursework itself, there were not any courses dedicated specifically to test. Only 4 of the 47 courses (8%) mentioned test explicitly in the course description, and those courses which do list test in their course description teach software testing principles at high-level and limited scope. Further, of the four courses that mention test in the course descriptor, three of them refer to hardware test and only one of those classes appears to cover software test.

The CC2001 also includes a list of 80 additional advanced courses proposed for undergraduates [CC2001-F1]. Of the 80 courses, none of them are devoted to test. Further, the areas of study for both entry level and advance courses fail to include test as a subject area.

In December 2008, Computer Science Curriculum 2008: An Interim Revision of CS 2001 was released. In the Release report from the Interim Review Task Force they noted the input they sought and received from industry members. Quality issues including testing, debugging, and bug tracking were among key topics that were noted as a recurring themes based on feedback and attention provided by industry participants [CC2008].

Unfortunately, after reviewing the 2008 Interim review document, there are still no courses dedicated specifically to test and the amount and depth of test materials in other computer science courses does not appear to have changed much since the original CS2001 [CC2008].

B. Software Engineering Education Project 2004

In August 2004, the ACM and IEEE-CS Joint Task Force on Computing Curricula published Software Engineering 2004 (SE2004-3) – Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. The material published in SE2004 was based upon work supported by NSF Grant No. 003263. The publication of SE2004 was developed by a broad group of volunteers including representatives from computer societies world-wide, specifically the Australian Computer Society, the British Computer Society, and the Information Processing Society of Japan [SE2004-1].

The SE2004 document contains two main contribution pieces. First, a section on Software Engineering Education Knowledge (SEEK) – what every software engineering graduate must know and second, a section on Curriculum [SE2004-2]. The SEEK materials are broken down into 10 knowledge areas [SE2004-T1].

The knowledge area labeled VAV (Software Verification and Validation) proposes 21 hours related to testing and 6 hours related to human computer user interface testing and evaluation. This provides a total of 27 hours of test related materials suggested in the SEEK model. A review of the suggested coursework reveals an outline for obtaining the suggested 27 hours of test related knowledge. The coursework includes two courses focused directly on test (SE221 Software Testing and SE321 Software Quality Assurance and Testing) and five additional courses that mention test in the course description (CS101I, MA271, SE101, SE102, and SE201).

The two courses focused on test contain an impressive overview of test including the breakdown of core test principles and phases. However, step back and realize that 27 hours of total test education compared to 459 hours of total software engineering course work means that test education and principles still only account for 5.8% of the total undergraduate curricula. If including software quality as a superset of test an additional 16 hours of coursework are suggested in the SEEK model, bringing the total to 9.4% of the total SE undergraduate curricula [SE2004].

C. National Science Foundation Interest 2001-2004

In January 2001, Professor Cem Kaner J.D., Ph.D. applied for and received NSF grant (EIA-0113539 ITR/SY+PE) entitled "Improving the Education of Software Testers." In his grant proposal, Kaner cited the shortage of software test skills among computer science graduates and also pointed out the lack of alternative collegiate programs in the area of software test. He proposed his project would "lay foundation for significant improvements in the quality of academic and commercial courses in software testing" [KANER].

While the research and materials developed by Kaner and the graduate students employed by the NSF grant are clearly a step in the right direction for the test profession, we recommend a more industry based approach using industry test professionals to collaboratively create test materials based on demonstrated working expertise in software test.

Another key difference between materials generated as post-NSF grant materials and the courseware recommended in this paper is the target audience. For example, instead of Kaner's selected approach which "lends itself to self-paced study and remote learning" [KANER], we believe curricula changes should instead be implemented at the in-class main-stream IT/CS/SE collegiate level through work with the IEEE Curricula Task Forces and universities. An industry and academic partnership to package and distribute the materials to enable classroom-oriented learning and hands-on applicable lab exercises seems reasonable.

While the efforts of Kaner and his team should be commended, it is believed that there is still a need for new efforts to gather materials from a broad industry perspective and target curricula standards and university course offerings. The new proposed materials would therefore impact a much broader collegiate audience with industry-driven expert-elicited test materials.

V. A Candidate Solution for Test Engineering Education

In order to ensure a supply of sufficiently skilled test-oriented engineers during the next five to ten years, actions in higher education curricula must be taken. It is our belief that industry must partner with academia to elevate the status of testing as an equally crucial component of software engineering as the development phase is held today. Technology companies are in a unique position to externalize the knowledge of practical test engineering to academia in order to yield the requisite number of highly skilled test engineers. This section discusses one company's efforts to alleviate this problem by creating coursework centered on the multifaceted disciplines that comprise software testing. A discussion of ongoing work-in-progress of the creation of an introductory level software testing class is presented, which addresses a broad variety of software test disciplines (with their abbreviations used throughout the paper) including Unit, (Unit), Function (Func), System (Sys), Integration (Integ), Service (Serv), Performance (Perf), Beta (Beta), Architecture Verification (Arch), Usability (Usab), Security (Sec), Algorithm Verification (Algo), Global System Integration (GSI), Regression (Regr), Combinatorial (Comb), and Scalability (Scale) as well as an overview of widely used testing techniques for product validation.

A. Expert Crowd-Sourced Content Generation

How do you accumulate the knowledge collateral necessary to prepare students for potential employment as productive and effective testers? Some propose that a guiding expert, collaborating with a group of students, could effectively build educational curriculum that would provide the necessary knowledge to prepare those entering the work force. As mentioned previously, funding by the NSF (National Science Foundation) was granted to attempt such an approach [KANER].

A crowd-sourcing approach was used to generate the content and provide thorough review to assemble testing educational materials. This technique is certainly similar to those previously employed for other content creation projects such as Wikipedia. The most dramatic differentiator of the work presented here is that the volunteer contributors and reviewers are professional software testers by trade. The experience levels of the contributors and reviewers varied from greater than 5 years of professional industry-based testing experience through career testers with 25+ years.

The courseware was developed to introduce a target student body to the wide variety of testing phases used for developing mission-critical systems and applications. The comprehensive effort was led by a core team of six testing professionals with the goal to ensure that the content associated with all of the various test phases was progressing toward target completion dates and that the material was consistent with a modeling template. Each core team member was designated as a core team coach for up to four test phase teams. This approach provided real-time feedback of content creation status without getting bogged down with the challenges of working in large teams.

In addition to the core team members, the content material for each phase of testing was developed and reviewed by a team of individuals that were specialists in those respective areas of test. This approach is in contrast to Kaner's NSF work discussed earlier, whose team was made up of undergraduate and graduate students, with no mention of their external testing experience. The authors of this paper worked to develop the content with well-known experts for thorough review of the materials for each corresponding test discipline.

The development of the content and the assurance of its accuracy were time-intensive activities. Having the effort be spread across a larger team of people helped to reduce the personal impact to individuals, but the hours of work performed was still impressive to say the least. If you look at the key test phases to which the majority of mission-critical software is exposed (i.e., Unit Test, Function Test, System Test, Integration Test (also known in the literature as Acceptance Test), and Beta Test), it is clear that the efforts at building testing educational collateral for these phases was roughly proportional to their phases of typical planning and execution. For example, Function Test is a critical phase of testing for removing the majority of defects during the life cycle of a software project, and as such it is often the most costly. Tests such as Unit, System and Integration show comparable percentages to their normal life cycle efforts. Results indicate that constructing the educational material for specialized test phases was proportionate across those phases (i.e., Usab, Sec, Algo, Arch, etc). One exception to these findings was the Performance Test phase, as this phase was built by a team of individuals whose working relationship came together quickly and were able to agree on techniques based on standardized processes used in the performance

industry. Figure 2 focuses on the preparation time spent on each phase and its corresponding review time. The hours of effort were tracked by each individual team.

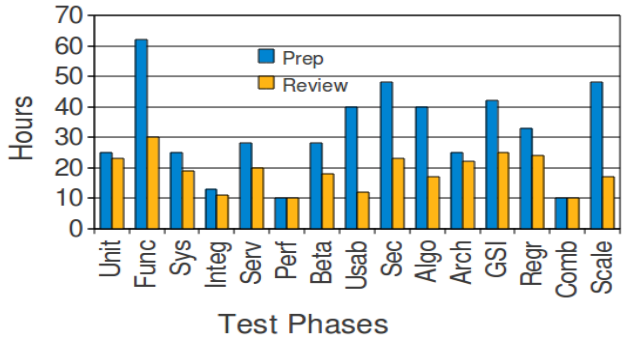

Figure 2. Preparation Time vs. Review Time Across Phases.

The amount of effort was extensive, roughly equivalent to 19 weeks (assuming 40 hours per week) of time spent developing and assuring the content for the testing class. That implication is that an individual trying to construct the equivalent content would spend approximately 4¾ months of time in class content creation. The total number of hours to compose the educational materials was 758 hours. The ratio of preparation (developing materials) to review time was approximately 2/3 to 1/3.

The ratio of the preparation time to the review time was observed to be within intuitive expectations of 2/3 to 1/3. The average preparation and review time, across individual teams, of 32 and 19 hours respectively aligned well with the overall results. The work done was spread across approximately 3.5 people per test phase team on average. Figure 3 outlines the distribution of hours of the 15 different phases of testing targeted during the construction of the class materials.

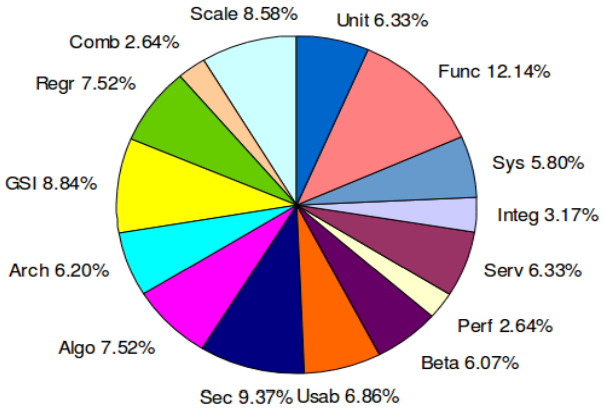

Figure 3. Distribution of time spent across the test phases

Overall, this crowd-source technique provided a cost-effective and expert-evaluated approach. It is believed that this approach will yield much-needed, high-quality, test education for incoming students. But one common issue kept coming up during content creation, how can one educate students based on industry practice when those students may not have access to the industry tools which experts are so familiar with?

5.2 Role of Reduced Cost Tools and Licensing

Within academia, teaching the science of software testing cannot be dependent on proprietary software products and testing tools for examples, laboratory projects, or exams. The only viable and practical means for students to use "real tools" and perform validation on any kind of production software is to utilize cost-reduced tools or licensing. These examples include packages available in the open source community or, via collaborative arrangements with corporations. This alternative seems the most reasonable, short of a revolution in academic licensing for professional testing tools. Furthermore, the selection of open source software (as a product to test or as a testing tool) and cost-reduced licensing with corporations is available today, with no need to change the professional licensing status quo. More often than not, the software can be obtained readily.

For those who wish to teach the use of commercially available, supported, enterprise testing software, each of the Top 5 Quality Assurance Testing Software Solutions [QASS] have academic licensing, and courseware options available as shown in Table 2.

TABLE II. TOP 5 QA TESTING SOFTTWARE ACADEMIC LICENSING

Company	Solution	Academic Licensing	Courseware
TechExcel	DevTest Studio	*	[TECH]
Seapine Software	QA Wizard Pro	*	[SEAP]
Pragmatic Software	Software Planner	*	[PRAG]
IBM	Rational Test Manager	[RAT1]	[RAT2]
HP	HP Quality Center	*	[HPQC]

* Information obtained by authors via phone and email correspondence with company representatives.

These commercial test solutions provide a good platform from which to provide students with real-life experience on state of the art testing tools and courses. There is another nuance to be addressed when teaching someone to be a good software tester. Instructing students on clear and concise bug reporting practices is crucial and considered necessary to build effective software testers.

B. Teaching the importance of Good Bug Reports

Whether you have chosen to take advantage of the corporate reduced licenses or the open source approach to have tools and products on which to test, another key skill that should be taught and introduced at the undergraduate level is bug (or more appropriately, defect) tracking and management skills. The effectiveness of a tester is based not only on the number of defects they open, but most importantly, the impact

and severity of the defects they open and the diagnostic symptoms and details they provided in the actual bug report.

Student's exposure to well-written defect or bug reports would only likely be understood if they are fluent in open source software and have used "production" bug tracking systems and software for open projects. The better a tester becomes at documenting and pinpointing root cause and/or the steps to re-produce the problem, the more likely the defect will be fixed by development in a timely manner. With hundreds if not thousands of bugs for developers to wade through, a clearly written and well-documented defect has a much higher likelihood of catching the attention of the developer and justifying the time it will take to provide a fix.

As part of that documentation, it is critical to outline the impact that the defect will have on its customer and their day-to-day business. This documentation should also include specific details on the symptoms observed and the steps required to reproduce the bug. Conversely, a poorly documented defect cannot only diminish the credibility of the tester, but also cause immense delays and frustration for all involved. The delays, brought on by a poorly written bug, can have significant impact on test progress while the products are within the development cycle and it can do damage to customer relations if the client is awaiting a critical fix for their business. Also, if no clear component or element owner is specified within the defect, then it is extremely likely that a "hopping" effect will occur with the problem moving from one group to another for problem diagnosis. Therefore, it is critical to follow a good set of characteristics in your defects. Examples of exemplary bug reports do exist [BUG1] and instruction on their creation should be a critical component of any software testing course.

As discussed previously, identification and extraction of the highest impacting defects is simplified by a well-documented defect report. Teaching students that practice early in their career will allow them to be effective testers on the job on day one. While teaching software testing skills and curriculum, it needs to be requisite behavior to enforce teaching the use of "Bugzilla" or comparable tools since they are readily available. Linux live-CDs, which require no installation and run entirely within a computer's RAM, can be used to find defects in a distribution of Linux and bugs can be opened. This provides great practice for building the skill of effective bug reporting. Participating in the reporting of defects on a project as large as Linux provides a sharp contrast to the fact that students work principally on small scale (semester length) projects in which obvious bugs are fixed in quick iterations. Often those students may not have any experience with, nor grasp the complexity of, a traditional bug life cycle. Bugzilla's version of that life cycle can be referenced online within Chapter 5 of the Bugzilla Guide [BUG2].

The strong combination of a set of test classes developed and assured by professional testers, a corporation-supported set of tools and products free of charge to abiding faculty, and instruction on the proper way to build defect reports, will greatly enhance the skills of incoming graduates into the workforce of software testers.

VI. Conclusion and Further Development

As complexity in the IT industry and customer solutions continues to grow, the need for highly-skilled software test engineers will also continue to increase. In an industry where quality is expected and defects can cause costly or even life-threatening errors it is imperative that the shortage of software test skills among recent university graduates is addressed. If students are to be prepared for jobs in software test, curricula guidelines and available course materials must be reformed. The courseware developed by IBM experts across the multiple phases of test is part of an ongoing approach to bridge these skill gaps and target test education to undergraduate students.

The courseware materials developed by IBM software test experts are IBM Intellectual Property, which is intended to be shared across university partnerships and partners in the business services industry. The authors of this paper remain eager to work with the IEEE and ACM to help propose curricula advancements in the area of software test. Furthermore, there is great interest in forming partnerships with industry in the area of curricula and courseware.

As completion of introductory level software test courseware described in this paper nears, plans to continue this effort with a second phase targeting higher level courseware focused on diving further into the details of different test phases, approaches, and methodologies are ongoing. The second phase of advanced courseware development is expected to start in earnest in the fall of 2010.

It is our hope and vision that the future of software test education be addressed and formal changes are implemented starting with the curricula guidelines and flowing all the way through the existence of more test courseware, journals, societies, and education opportunities for software test engineers. This is a growing field with opportunities for skilled software test engineers to have thriving careers in this industry.

VII. Acknowledgements

The authors would like to thank their employer, International Business Machines Corporation for supporting their efforts to produce educational content. We would also like to thank those parties who provided quotations for use in this paper.

References

[CMU] Carnegie Mellon University, "B.S. in Computer Science Curriculum Requirements Page," June 2009. [Online]. Available: http://www.csd.cs.cmu.edu/education/bscs/currreq.html [Accessed: November 14, 2009].

[STAN] Stanford University Computer Science Education, "CSE Hosted Class Page," [Online]. Available: http://cse.stanford.edu/class/ [Accessed: November 14, 2009].

[NIST] National Institute of Standards and Technology, "Software Errors Cost U.S Economy $59.5Billion Annually," June 28, 2002. [Online]. Available: http://www.nist.gov/public_affairs/releases/n02-10.htm [Accessed: November 13, 2009].

[NTSB] National Transportation Safety Board, “Aircraft Accident Report: Controlled Flight Into Terrain Korean Air Flight 801,” August 6, 1997. [Online]. Available:http://www.ntsb.gov/Publictn/2000/AAR0001.htm . [Accessed: November 13, 2009].

[CDRH] Center for Devices and Radiological Health, "Ensuring the Safety of Marketed Medical Devices - CDRH’s Medical Device Postmarket Safety Program" January, 18 2006. [Online]. Available: http://www.premierinc.org/quality-safety/toolsservices/safety/topics/bar_coding/downloads/03-mdpi-report-0106.pdf. [Accessed: November 14, 2009]

[CBS] B. Cosgrove-Mather, “Software Bugs Can Be Lethal,” CBSNews.com, Apr. 29, 2003. [Online]. Available: http://www.cbsnews.com/stories/2003/04/29/tech/main551492.shtml. [Accessed: January 20, 2010].

[DANI] E. J. Correia, “There Are 349,140 Software Testers In The U.S.,” Nov. 12, 2008. [Online]. Available: http://www.daniweb.com/news/post975934.html [Accessed: January 20, 2010].

[STP] Software Test & Performance Collaborative. [Online]. Available: http://www.stpcollaborative.com/membership [Accessed: January 20, 2010].

[PORTAL] The ACM Digital Library, “ISSTA: International Symposium on Software Testing and Analysis,” 2009. [Online]. Available: http://portal.acm.org/browse_dl.cfm?coll=ACM&dl=ACM&idx=SERIES392&linked=1&part=series [Accessed: January 20, 2010].

[ASQ] American Society for Quality, “Software Quality Engineer Certification Preperation,” [Online]. Available: http://www.asq.org/certification/software-quality-engineer/prepare.html, Paper and Pencil Sample Exam PDF [Accessed: Feb 9, 2010]

[CSDA] IEEE Computer Society, “CSDA Certified Software Development Associate,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/csda/home. [Accessed: January 20, 2010].

[CSDA2] IEEE Computer Society, “CSDA Online Learning System,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/csda/prep [Accessed: January 20, 2010].

[CSDA3] IEEE Computer Society, “CSDA: Sample Test Questions,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/csda/sampletest [Accessed: January 20, 2010].

[CSDP1] IEEE Computer Society, “CSDP Certified Software Development Professional,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/certification [Accessed: January 20, 2010].

[CSDP2] IEEE Computer Society, “CSDP: About the CSDP,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/certification/about;jsessionid=0E3A7506E9178858DBFD0A1704D25DA2 [Accessed: January 20, 2010].

[CSDP3] IEEE Computer Society, “CSDP: Prepare for the Exam,” IEEE Computer Society. [Online]. Available: http://www.computer.org/portal/web/certification/prep [Accessed: January 20, 2010].

[SEDEF] Wikipedia: The Free Encyclopedia, “Software Engineering,” Feb. 7, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Software_engineering [Accessed: January 22, 2010].

[ENGDEF] Wikipedia: The Free Encyclopedia, “Engineering,” Feb. 9, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Engineering [Accessed: January 22, 2010].

[SCIENCE] “Engineering,” ScienceDaily.com. [Online]. Available: http://www.sciencedaily.com/articles/e/engineering.htm [Accessed: January 22, 2010].

[EDRANK] “Rankings Computer Science, Ranked in 2008,” USNews.com, 2008. [Online]. Available: http://grad-schools.usnews.rankingsandreviews.com/best-graduate-schools/top-computer-science-schools/rankings [Accessed: January 20, 2010].

[WSU1] University of Washington, “Certificate in Software Test Automation,” [Online]. Available: http://www.outreach.washington.edu/ext/certificates/sta/sta_gen.asp [Accessed: February 1, 2010].

[WSU2] University of Washington, “Certificates in Software Testing,” [Online]. Available: http://www.outreach.washington.edu/ext/certificates/sot/sot_gen.asp [Accessed: February 1, 2010]

[CC2001-1] “Computing Curricula 2001 Computer Science,” December 15, 2001, [Online]. Available:http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE22806.pdf.[Accessed: January 23, 2010].

[CC2001-F1] “Computing Curricula 2001 Computer Science,” p. 235, Figure B-4, December 15, 2001, [Online]. Available:http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE22806.pdf.[Accessed: January 23, 2010].

[CC2008] “Computer Science Curriculum 2008: An Interim Revision of CS 2001,” p. 11, December 2008. [Online]. Available:http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE22805.pdf.[Accessed: January 23, 2010]

[SE2004-1] “Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,” p. 2, September 25, 2004. [Online]. Available: http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE-22803.pdf. [Accessed: Jan 25, 2010].

[SE2004-2] “Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,” p. 7, September 25, 2004. [Online]. Available: http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE-22803.pdf. [Accessed: Jan 25, 2010].

[SE2004-T1] “Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,” Table 1, p. 27, September 25, 2004. [Online]. Available:http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE-22803.pdf. [Accessed: Jan 25, 2010].

[SE2004-3] “Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,” September 25, 2004. [Online]. Available: http://www.computer.org/portal/c/document_library/get_file?p_l_id=1330505&folderId=1337155&name=DLFE-22803.pdf. [Accessed: Jan 25, 2010].

[KANER] Cem Kaner, “Improving the Education of Software Testers,” NSF Grant EIA0113539 ITR/SY+PE, p. 5, January 24, 2001. [Online]. Available: http://www.testingeducation.org/general/nsf_grant.pdf

[CTFLS-1] “Certified Tester Foundation Level Syllabus,” p.9, April 12, 2007. [Online]. Available: http://www.istqb.org/downloads/syllabi/SyllabusFoundation.pdf. [Accessed: February 5, 2010].

[CTFLS-2] “Certified Tester Foundation Level Syllabus,” p.68, April 12, 2007. [Online]. Available: http://www.istqb.org/downloads/syllabi/SyllabusFoundation.pdf. [Accessed: February 5, 2010].

[QASS] “Top 5 QA Testing Software Solutions – 2009,” 2009. [Online]. Available: http://www.business-software.com/downloader.php?k=L2RhdGEvd3d3L3Zob3N0cy9wcm9kL2J1c2luZXNzLXNvZnR3YXJlLmNvbS9wZGYvdG9wXzVfcWFfdGVzdGluZy5wZGY%3D [Accessed: February 5, 2010].

[IBMAI] IBM, “IBM Academic Initiative.” IBM. https://www.ibm.com/developerworks/university/academicinitiative [Accessed: February 6,2010].

[RAT1] IBM, “Rational Software Academic Initiative Program”. [Online]. Available:https://www.ibm.com/developerworks/university/rational/index.html. [Accessed: February 6, 2010].

[RAT2] IBM, “Rational Courses”. [Online]. Available: https://www14.software.ibm.com/webapp/devtool/scholar/web/coursewarePickPage.do?source=aicoursersdd. [Accessed: February 6, 2010].

[BUG1] Software Testing Help, “How to write a good bug report? Tips and Tricks,” [Online]. Available: http://www.softwaretestinghelp.com/how-to-write-good-bug-report/#more-102 [Accessed: February 2, 2010].

[BUG2] The Bugzilla Guide, “Life Cycle of a Bug,” [Online]. Available: http://www.bugzilla.org/docs/2.22/html/lifecycle.html [Accessed: February 2, 2010].

[TECH] TechExcel, "Download pre-recorded demos". [Online]. Available: http://www.techexcel.com/Formwise/cSurvey.asp?k=DT50Overview&apptype=1. [Accessed: February 9, 2010].

[SEAP] Seapine Software, "Training", [Online]. Available: http://www.seapine.com/training.html. [Accessed: February 10, 2010].

[PRAG] Pragmatic Software, "Welcome to Software Planner". [Online]. Available: http://www.pragmaticsw.com/SoftwarePlanner.asp. [Accessed: February 10, 2010].

[HPQC] HP, "HP Software Training". [Online]. Available: http://h10076.www1.hp.com/education/hpsw/hpsw-training.htm#qc. [Accessed: February 10, 2001]

Appendix A. Certified Software Development Professional Exam Questions By Component

Rank	College	Masters or Doctoral Degrees In C.S. or S.E for Software-Testing Related Work	Required or Dedicated Test Courses	Courses with Test Component	Other
1	Massachusetts Institute of Technology, Cambridge, MA	No	0	0 (*)	
1	Stanford University, Stanford, CA	No	0	0 (*)	
1	University of California, Berkeley, CA	No	2	1	
4	Carnegie Mellon University, Pittsburgh, PA	No	1	1	
5	University of Illinois--Urbana-Champaign	No	0	0	S.E. certificate available on top of B.S Requirements
6	Cornell University, Ithaca, NY	No	0	1	
6	Princeton University, Princeton, NJ	No	0	1	
6	University of Washington, Seattle, WA	No	0	1	Continuing Education / Certificate programs dedicated to testing
9	Georgia Institute of Technology, Atlanta, GA	Yes	0	2	S.E. is a specialization; it is achieved by completing at least 2 graduate level courses
9	University of Texas, Austin, TX	No	0	2	
11	California Institute of Technology, Pasadena, CA	No	1	0	
11	University of Wisconsin, Madison, WI	No	0	1	2 research projects are offered that have a testing focus; computer security & systems research
13	University of California, Los Angeles, CA	No	1	6	
13	University of Maryland, College Park, MD	Yes	1	1	Testing focus under Graduate S.E. Research Programs
13	University of Michigan, Ann Arbor, MI	No	1	7	
16	Columbia University, New York, NY	No	0	2	Some under graduate research projects with test focus
16	Harvard University, Cambridge, MA	No	0	0	
16	University of California, San Diego, CA	Yes	1	6	Research project in S.E. has "Testing and Analysis" focus
19	Purdue University, West Lafayette, IN	No	0	0	Software testing is a course that can be taken if S.E. is chosen as specialization, but not a required course for graduation. S.E. research area
20	Brown University, Providence, RI	No	0	3	
20	Duke University, Durham, NC	No	0	5 (*)	
20	Rice University, Houston, TX	No	0	5	
20	University of Massachusetts, Amherst, MA	No	0	4 (*)	
20	University of North Carolina, Chapel Hill, NC	No	0	2	Software Systems Research: S.E. Group conducts research into systematic testing
20	University of Pennsylvania, Philadelphia, PA	Yes	0	2	PhD program offers opportunities in S.E.
20	University of Southern California, Los Angeles, CA	No	0	2	
20	Yale University, New Haven, CT	No	0	1	

* Items with an asterisk indicate course listings which did not appear to be complete listings.