Position Paper:
Critical Problems in Software Engineering

Draft 1
December 22, 2009

Capers Jones, President
Capers Jones& Associates LLC
Email: CJonesiii@cs.com

INTRODUCTION

Software has become the driving force of industry and government. Without software few modern industries would operate at all, and none would operate efficiently. Yet in spite of the importance of software to business and government, software remains one of the most troubling occupations of the century; and indeed of all time.

Software applications and software engineering have a bad reputation among top executives and government officials and this bad reputation is justified. Software projects are characterized by:

1. The highest failure rates of any engineering field.
2. The largest number of cost overruns of any constructed artifacts.
3. The longest schedule slips of any constructed artifacts.
4. Quality levels of delivered applications ranging from marginal to catastrophic.
5. Significant numbers of lawsuits filed by disgruntled clients.
6. Software lacks reliable quantitative data on quality and productivity.
7. Software uses metrics bad enough to be professional malpractice.
8. Custom construction of products that should be composed of standard parts.
9. Construction of applications riddled with security flaws.
10. Refusal by software vendors to provide meaningful warranties.

The Software Engineering Methods and Theory (SEMAT) organization is proposing to examine and hopefully eliminate the root causes of software engineering problems and place software engineering on a solid empirical basis. In order to accomplish these goals, the method of studying software engineering should itself be placed on a solid platform and supported by workable metrics. Following is my proposal for the sequence of study of software engineering problems:

1. Analyze the metrics and measurement practices of software and eliminate or modify those with serious economic flaws such as “lines of code” and “cost per defect.”

2. Assemble accurate tables of the cost drivers of software applications ranging from small one-person projects through massive applications developed by teams in excess of 1000 personnel.

3. Acquire or collect quantitative data on a significant number of software applications in terms of sizes, schedules, costs, and quality levels.

4. Publish accurate data on the sizes, schedules, and costs of a significant number of software applications.

5. Acquire or collect data on the tools, methodologies, languages, and CMMI levels of a statistically significant number of software applications.

6. Perform multiple regression studies to identify the effectiveness of methodologies, CMMI levels and other factors that are asserted to improve software performance.

7. Establish a workable definition of the term "software engineering malpractice."

8. Establish a workable framework for software warranties that would provide clients with protection against poor quality levels or excessive security vulnerabilities.

9. Establish an effective certification program for reusable components to encourage assembly from standard parts rather than custom development.

10. Establish a framework for licensing of software engineers and additional frameworks for major specialties within software engineering.

Following in rank order are the 15 major cost drivers of software applications derived from studies within about 150 large corporations:

U.S. Software Costs in Rank Order Circa 2010

1. The cost of finding and fixing bugs.
2. The cost of cancelled projects.
3. The cost of producing English words.
4. The cost of dealing with security flaws and attacks.
5. The cost of unplanned requirement changes.
6. The cost of programming or coding.
7. The cost of customer support after deployment.
8. The cost of meetings and communications.
9. The cost or project management.
10. The cost of renovation and repair of legacy applications.
11. The cost of innovation and new features.
12. The cost of litigation for failures and disasters.

13. The cost of training and learning new skills.
14. The cost of preventing security flaws.
15. The cost of creating reusable materials.

The pattern of current software cost drivers indicates an unhealthy tendency for failures, bug repairs, and security problems to outweigh the costs of innovation and actual code development. This is not the pattern of a true profession.

My recommendation to SEMAT is to try and change the sequence of cost drivers so that defect repairs, failures, and security flaws move to the bottom of the list. A healthier pattern might have the following sequence:

Proposed U.S. Software Costs in Rank Order Circa 2025

1. The cost of innovation and new features.
2. The cost of renovating legacy applications.
3. The cost of customer support after deployment.
4. The cost of creating and utilizing reusable components.
5. The cost of meetings and communications.
6. The cost of avoiding security flaws.
7. The cost of learning and training.
8. The cost of project management.
9. The cost of requirements changes.
10. The cost of producing English words.
11. The cost of programming or coding.
12. The cost of finding and fixing bugs.
13. The cost of security flaws and attacks.
14. The cost of cancelled projects.
15. The cost of litigation for failures and disasters.

About 150 years ago the medical profession was also troubled by malpractice, lack of standards, poor measurements, and problems that are remarkably similar to those of software engineering in 2010. The path followed by the American Medical Association to improve medical practice and introduce licensing and certification has many lessons for SEMAT and software engineering.

A useful book that describes the AMA path is The Social Transformation of American Medicine by Paul Starr. This book won a Pulitzer Prize in 1982 and remains perhaps the most complete description of the steps needed to transform a craft into a true profession.