Packaging and Prioritizing Software Engineering Research to Maximize Industry Impact

Will Tracz
Lockheed Martin IS&GS
Endicott, NY
USA
Will.Tracz@LMCO.com

ABSTRACT
The success of Software Engineering research can be measured by its impact on software system implementation quality, productivity, and ability to address complexity initially and in the future. While it is recognized that it may take years for new techniques, tools, theories, or processes to mature and transition from the state of the art to the state of the practice, it is the thesis of this paper that with proper preparation, packaging and prioritization, advances in software engineering research can more rapidly transition to industry. To this end we propose a software engineering research classification framework to better set expectations of potential impact on industry applications. The paper closes with a list of "Software Engineering Challenges" collected from industry software engineers.

Categories and Subject Descriptors
D.2. [**Software Engineering],** D.2.10 [**Design**] Subjects: Methodologies

General Terms: Management, Measurement, Documentation, Performance, Design, Reliability, Experimentation, Security, Verification.

Keywords
Software Engineering Research, Software Tools and Techniques

1. INTRODUCTION
Software Engineering research has resulted in significant advances over the decades since the term "Software Engineering" was coined. Some research has never progressed (or will never progress) out of university or industry laboratories into commercial practice for a number of reasons. One possible reason for this transition failure is the lack of real world application, which should not be confused with lack of real world applicability. It is the premise of this paper that the application of the proposed software engineering research classification framework would lead to improvements in software engineering research packaging and prioritization that would consequently result in better expectation setting for industry, which in turn would expedite transitioning tools, techniques, and theories from the state of the art to state of the practice.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.
Copyright 2010 ACM 978-1-4503-0427-6/10/11...$10.00.

1.1 Position Paper Organization
This paper is organized into 2 sections. The first section describes a proposed software engineering research classification framework. The second section provides a list of representative "Software Engineering Challenges" facing industry today. While this list is not a list of "Software Engineering Research Topics," it could serve as the basis for grounding research in industry applications and has the potential for creating industry / academic partnerships.

2. SOFTWARE ENGINEERING RESEARCH CLASSIFICATION FRAMEWORK
Software development in industry is based on a series of decisions that lead to the language, tools, methodology and team structure that will be used for developing commercial off-the-shelf (COTS) software or a software-intensive system. Some of these decisions are based on engineering tradeoffs; others are based on personal experience, and some mandated by corporate standards or contract requirements. The thesis of this paper is that Software Engineering research would benefit if the research focus could be classified / characterized as it relates the following (inter-dependent) factors facing software developers:

- system size (i.e. implementation effort / complexity),
- team size (i.e., how many people will be involved in developing the product),
- market size (i.e., who will be using the product),
- component size(i.e., integration/configuration effort), and
- schedule size.

The proposed framework has as its genesis the 1975 paper by Frank DeRemer and Hans Kron "*Programming in the Large vs Programming in the Small*."[1] Our framework extends their analysis to include the following comparisons:

- Software Engineering **in the Small** versus Software Engineering **in the Large** (System Size)
- Software Engineering **by the Small** versus Software Engineering **by the Large** (Team Size)
- Software Engineering **with the Small** versus Software Engineering **with the Large** (Component Size/Configuration and Integration effort)
- Software Engineering **for the Small** versus Software Engineering **for the Large** (Market Size)

- Software Engineering **during the Small** versus Software Engineering **during the Large** (Schedule Size) (Note "during" is perhaps not the best choice of descriptive words, but until a better one is found, it will suffice.)

An obvious challenge in doing such comparisons is determining / defining: *What is "Large?"* and *What is "Small?"* Table 1 provides an initial set of intuitive yet unsubstantiated thresholds for smallness and largeness. In addition, it speculates the need for an additional category "Ultra-large" (which is not meant to imply the need for a "medium" category). Another goal of this research is to determine size / complexity thresholds that drive software development decisions to certain outcomes.

Other related factors not currently addressed by this framework include the number of requirements, the complexity of requirements, and the available resources (funding being the most important) and complexity of the environment.

Table 1: Software Engineering Development Factors

Size	By the (Team)	With the (Component)	In the (Code)	For the (Market)	During the (Schedule)
Small	1-N	Function/calls	K	~1-10s	Weeks
Large	Dozens	SoS/SOAP	M	~100s-1000s	Months
Ultra-large	100s	SoS	10M+	-	Years

2.1 Assumptions

For reasons of simplification and to facilitate discussion, we make the following assumptions:

1. All software engineers are same experience / expertise level
2. There is a cost associated with all tools
3. There is a cost associated with Commercial Off the Shelf (COTS) Software
4. There is a smaller cost associated with Free and Open Source Software (FOSS) (e.g., service / support contract) than with COTS Software
5. The complexity of the system correlates to the size of the system (a weak assumption, as there are complex algorithms and platform, network or physical environment constraints)

Note: A paradigm shift, such as advances in model-based system development, could skew this analysis and re-prioritize certain tradeoffs or redefine size boundaries.

2.2 Examples of Software Engineering Decisions

The list below provides an initial, high-level set of decisions faced by software practitioners at the start of a software development project. It is a partial list, presented here for discussion purposes, and would vary according to project-specific aspects such as domain, available expertise, existing resources, and funding. A software engineering decision tree, such as this, is insightful in not only providing guidance to software developers, but also in allowing software engineering researchers to set expectations about the "*domain of applicability*" of their software engineering research and to help place the research in a practitioner's context so as to better facilitate collaboration and transition.

1) Language

a. Scripting
b. Procedural
c. Object-Oriented
d. Dynamic

2) Process

a. Waterfall
b. Incremental
c. Agile
d. Ad hoc

3) Tools

a. Requirements Management
b. Architecture Specification and Analysis
c. Design Specification and Analysis
d. Configuration Management
e. Release Management
f. Defect Management
g. Integration and Verification

Note: Again, the author acknowledges that this set of decisions, as presented here, is speculative and incomplete. Feedback from this paper and the availability of empirical data will be used to guide future efforts to extend and validate this framework.

3. Industry Software Engineering Challenges

The following is a loosely edited list of challenges facing industry today:

1. How can we migrate existing software to new processors when facing such challenges as:
   - Big/Little-endian incompatibility?
   - Single core to multi-core scheduling?
   - Homogeneous vs. heterogeneous multi-core architectures?
2. What tools/techniques are available to design software to better utilize multi-core hardware configurations?
3. What tools/techniques are available to address distributed embedded real-time systems development challenges such as performing non-intrusive real-time debug of embedded systems?
4. How can we apply the theoretical foundation for probabilistic real-time analysis (i.e., schedulability for probabilistic arrival intervals)?
5. What techniques may be applied to partitioning a software architecture in a manner that improves its scalability, maintainability and reliability?
6. What system/software architectures reduce complexity?
7. How can we determine system reliability of a software intensive system?
8. How will system integration strategy impact the partitioning of the system/software architecture?
9. How does architecture selection impact the lifecycle cost of the software?
10. What is the implication of the "convergence" of technologies: computer architecture, network architecture, database architecture and application software architecture?
11. How reliable are architecture-based cost estimation techniques?
12. How do we close the gap between system design and software implementation?

13. What is the most reliable metric to determine the cost estimate for software development?
14. What is the "missing links" between systems architecture, SysML, UML, data modeling, code generation and integration and test?
15. Are there precise operational semantics for modeling languages? If so, how do we leverage them?
16. How do we fully integrate and test, Service-Oriented Architecture (SOA) services with Model-Based Systems Engineering constructs?
17. Are there firm metrics for Model Based methods for system engineering and software (requirements traceability, systems analysis, code generation, software design, system life cycle)?
18. What issues will need to be resolved in moving from Model Based Systems Engineering to automatic code generation for software intensive systems?
19. What are the "new" questions that technical leadership needs to ask the development teams based on the changes in system/software methodologies and design techniques?
20. How will software productivity be justified when building a system by invoking services from existing frameworks, such as .Net?
21. What concept of operations (CONOPS) needs to be put in place to insure that system / software reuse is an effective way to minimize development and support costs?
22. What techniques/tools will be required to allow the integration of pre-defined software services used to construct system capabilities?
23. Is it time to revisit formal methods to improve the development of software systems?
24. What role does "system software" have in mitigating the cyber-security type threats?
25. Are there better ways of addressing dependency analysis for complex systems?
    - At start up
    - At restart (partial system and full system)
26. How do development environments need to evolve to support new methodologies associated with systems and software development?
27. What tools and techniques can help software integration – sometimes called integration testing? (Note: Testing is a misnomer. What we are really doing is "discovering" all of the interfaces that were not properly designed.)

In summary, these questions reflect how software practitioners are looking for advances in software engineering theory, tools and techniques to help them deal with ever increasingly more complex requirements. While some software engineering decisions will always be made for non-technical reasons, many software intensive systems engineering organizations recognize the need to collaborate with software engineering researchers. Once fully defined and validated, the proposed framework presented in this paper will support the establishment of those relationships by facilitating the setting of expectations and defining of software engineering research domains of applicability that software practitioners can better understand.

4. ACKNOWLEDGMENTS

I would like to thank the members of the Advanced Software Technology Focus Group at Lockheed Martin for providing inputs on Software Engineering challenges, especially Bob Epps and Ben Watson.

5. REFERENCES

[1] DeRemer, Frank, Kron, Hans 1975. Programming-in-the large versus Programming-in-the-Small *Proceedings of the International Conference on Reliable Software* (Los Angeles, California) 114 – 121.