Software Engineering Education in the Era of Outsourcing, Distributed Development, and Open Source Software: Challenges and Opportunities

Matthew J. Hawthorne
Empirical Software Engineering Lab (ESEL)
ECE, The University of Texas at Austin
hawthorn@ece.utexas.edu

Dewayne E. Perry
Empirical Software Engineering Lab (ESEL)
ECE, The University of Texas at Austin
perry@ece.utexas.edu

ABSTRACT
As software development becomes increasingly globally distributed, and more software functions are delegated to common open source software (OSS) and commercial off-the-shelf (COTS) components, practicing software engineers face significant challenges for which current software engineering curricula may leave them inadequately prepared. A new multi-faceted distributed development model is emerging that effectively commoditizes many development activities once considered integral to software engineering, while simultaneously requiring practitioners to apply engineering principles in new and often unfamiliar contexts. We discuss the challenges that software engineers face as a direct result of outsourcing and other distributed development approaches that are increasingly being utilized by industry, and some of the key ways we need to evolve software engineering curricula to address these challenges.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *curriculum*; D.2 [**Software Engineering**]: D.2.9: Management – *programming teams*, *software process models*, *cost estimation*, *time estimation*; D.2.11 Software Architectures; D.2.1: Requirements/Specifications; D.2.13: Reusable Software – *domain engineering*; K.7 [**The Computing Profession**]: K.7.1: Occupations; K.7.2: Organizations.

General Terms
Management, Design, Economics.

1. INTRODUCTION
Software engineering is changing in fundamental ways, so software engineering education must also change. These changes follow three basic trends: 1) the movement of software development offshore to the lowest-cost location *de jour*, 2) the growth and maturation of languages and platforms such as Java and .NET, and 3) the reliance on third-party components for core system functionality. These trends mean that the mix of competencies required to successfully practice software engineering includes much more than just software development skills. In today's competitive environment, short-term market opportunities and financial results concerns increasingly impact software development projects, changing requirements and priorities on short notice, and increasing the pressure to produce software at the lowest possible cost. Further, the increasing commoditization and distribution of hands-on software development via outsourcing and the use of third-party components is causing the demand for traditional software development skills in many areas to be eclipsed by new opportunities available to software engineers who possess certain kinds of enhanced software *engineering* expertise.

To enable engineers to take advantage of these opportunities, we need to augment traditional software engineering strengths in software architecture [3], design, and development processes with new techniques that will enable software engineers to *apply engineering principles* in larger organizational and project management contexts. We need to produce software engineers who are just as comfortable practicing engineering in environments where the "tools" are distributed project teams and third-party components as they are designing and implementing complete systems themselves. We also need to equip engineers with architecture, design, and development approaches that facilitate third-party component integration, and the simultaneous evolution of product family architectures and external components. Given the right set of knowledge, skills and perspectives, we see this trend toward distributed and third-party development as an excellent opportunity to build on the unique strengths of software *engineering* as a distinct discipline, by equipping software *engineers* to play a central role in ensuring the success of development organizations as they explore these new development models.

Specific areas of competency that we believe will be among the most valuable for software engineers, and hence require additional curricula to support, include *organization and process engineering*, *system and product family architectures*, *integration techniques and technologies*, *product and product line management*, and *distributed project management*.

2. PROCESSES & ORGANIZATIONS
As software development moves offshore, we envision software engineers increasingly utilizing their expertise in architecture, design, and process engineering to take ownership of *engineering-in-the-large*, moving beyond implementation engineering to reengineer the very organizational processes and teams that are involved in conceiving, designing, developing and deploying software solutions. In the traditional implementation domain, this includes determining the optimal organization of distributed projects that may include a mixture of internal resources and outsourcing, as well as third-party (COTS and OSS) components. *Project allocation, estimation and control* present special challenges in this kind of mixed-mode distributed project, and depend on stringent requirements specification. Besides new approaches to software development itself, software engineers should understand the entire range of roles in the extended software engineering organization, including business management (e.g., *cost vs. risk analysis*), sales and marketing (e.g., *requirements gathering and prioritization*), customer support (e.g., *usability and supportability*), and others. An understanding of the *intellectual property ownership* and *security* issues involved with incorporating external project teams and components is also critical.

3. SYSTEM ARCHITECTURES
The increasing use of distributed and outsourced development, and the incorporation of third-party components makes it even more critical than ever that software engineers are equipped to understand, develop, and extend *product family architectures* that are robust and comprehensive enough to encompass all these disparate elements, and organize them into a consistent architectural view. To

ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.
ACM 1-58113-963-2/05/0005.

accomplish this, engineers will need to be very familiar with *integration techniques* such as *connectors*, *adapters* and *wrappers* so they can use them as building blocks to create architectures that effectively manage the complexity induced by components designed and developed in parallel by diverse teams. Approaches that enable implementation architectures to be derived more directly from abstractions of the problem domain, such as *goal- or prescription-based architectures* [1] and *intent-based implementations* [2], will also be important techniques to reduce complexity and enhance architectural consistency. Knowledge of industry application frameworks such as J2EE and .NET, and architectural options for integrating components from different frameworks, are useful technical skills.

4. PRODUCT & PROJECT MANAGEMENT

Product management, or managing the direction and functionality of software products and product families, including many aspects of *requirements engineering (RE)*, is increasingly performed by marketing professionals. Software engineers often only become involved in software development projects at implementation time, after the domain and marketing experts have specified the requirements (and often, the resource budget and project schedule), without considering engineering concerns like reusability and product family architectures. We need to prepare software engineers to be fully engaged in projects from the outset, when important decisions about project feasibility are made. As approaches like goal-driven prescriptive architectures and intent-driven implementations become more prevalent, RE will become the most critical part of the software engineer's job. Engineers will need to be *problem domain experts* in order to engineer the implementation domain. This will require engineers to work more closely with customers, business managers and other stakeholders, and to be flexible enough to reason about software design and usage from diverse points of view.

Project management will also be crucial as projects become more widely distributed. Software engineers need to become much better at *planning and estimation*, using advanced techniques for accurately estimating the time it will take the various internal and external teams to complete their parts of the project, including integration and quality assurance (QA). This will also require much better *interpersonal communication skills* than many software engineers currently possess to effectively communicate project requirements to distributed and outsourced team members, and verify that the requirements are satisfied.

5. DISCUSSION

To prepare software engineers to thrive in an increasingly distributed and outsourced environment, we recommend starting software engineering programs with an enhanced introduction to software engineering principles, including material that explicitly addresses outsourcing and distributed development, before any core technical coursework is attempted. The goal is to give students both the tools and the perspective needed to become software *engineers* who design *solutions*, rather than programmers who write programs. Starting with engineering principles gives students a firm foundation in software engineering, avoiding the need to "retrofit" them with this knowledge later on, and also enables the technical course materials, assignments, and projects to reinforce and expand upon these principles. Industry practitioners and managers should also be brought in regularly to provide perspective on how software engineering principles are used in industry.

Software development process curricula should emphasize *requirements engineering (RE)*, including *requirements management*, i.e., methods and techniques for managing the expectations of customers, managers and other stakeholders. While much of requirements gathering and prioritization is currently done by marketing professionals, software engineers should be prepared to take a leadership role in RE, taking market concerns into account, and working closely with market researchers and other stakeholders. Curricula should also teach students when and how to adopt aspects of "agile" methodologies and other *iterative approaches* to software development projects. And process education should include *organization engineering*, i.e., methods and techniques engineers at all levels can use to reengineer their organizations to optimize the success of development projects.

Architecture and system design curricula should emphasize *distributed system architectures*, and *component-connector architectures*, both of which make it easier to incorporate components developed by third parties, whether outsourced, OSS, or COTS. Architecture curricula should also include *product family architectures*, to give engineers tools to develop frameworks that bring order and consistency to systems developed by diverse distributed teams. And finally, *prescription-based architectural approaches* [1,2], in which the system architecture is directly derived from the requirements, will give software engineers important techniques to help ensure that systems they design fulfill the specified functional goals and constraints.

Project management education should be expanded to include *program management*, including such business-related concerns as cost/benefit analysis and market analysis, among others. Project management education should emphasize *cost and time planning and estimation for distributed projects*, including projects that utilize *internal and external outsourcing*. Software engineers need to think of themselves as manager-integrators who are comfortable engineering systems using outsourced teams and third-party components, while avoiding inefficient practices like unnecessary hands-on development. Outsourcing often includes international teams, so the effects of *cultural and linguistic issues* on team communication, expectations, etc., should also be addressed. And since the success of software projects largely depends on resource availability (time, engineers, equipment, tools, etc.), project management education would not be complete without addressing *advocacy and resource management* for development projects. Software engineers must be able to determine as early as possible whether the resources allocated to their project are sufficient to enable the project to be successful, so they can reengineer teams and processes, manage requirements and expectations, and do whatever else is necessary to ensure success.

6. CONCLUSIONS

For our profession to remain relevant, we need to prepare software engineers to assume leadership roles in engineering system requirements, building solutions using internal and third-party components and distributed development resources, and integrating software elements from these diverse sources into coherent product families using optimal component and connector architectures. To successfully develop systems using distributed resources, engineers will need to learn better product and project management, organization and process engineering, and interpersonal and cross-cultural communication skills. We see these challenges as an excellent opportunity for properly prepared software engineers to play a central role in ensuring the success of software projects and product families, by applying enhanced engineering principles to manage the architectural and process complexity induced by the current drive toward distributed and third-party development models.

7. REFERENCES

[1] Brandozzi, M. and Perry, D. *Architectural Prescriptions for Dependable Systems*, WADS'2002, May 2002.

[2] Hawthorne, M. Perry, D. *Exploiting Architectural Prescriptions for Self-Managing, Self-Adaptive Systems: A Position Paper*, WOSS '04, Oct. 2004.

[3] Perry, D., Wolf, A. *Foundations for the Study of Software Architecture*, ACM SIGSOFT Software Engineering Notes, 17(4):40, Oct. 1992.