Determining the Impact of Software Engineering Research on Practice

Leon J. Osterweil, University of Massachusetts, Amherst
Carlo Ghezzi, Politecnico di Milano
Jeff Kramer and Alexander L. Wolf, Imperial College London

The Impact Project provides a solid and scholarly assessment of the impact software engineering research has had on software engineering practice. The assessment takes the form of a series of studies and briefings, each involving literature searches and, where possible, personal interviews.

Software engineering has been an active research field since 1968, when the seminal NATO Conference, held in Garmisch, Germany,[1] raised awareness of the field's scientific and practical significance. Following that landmark event, the International Conference on Software Engineering has been held regularly since 1975—in recent years attracting more than 400 submissions and more than 1,000 attendees—and the ACM SIGSOFT International Symposium on the Foundations of Software Engineering has been held annually since 1982. Researchers view these events as the two best conferences in the field. Other technical meetings devoted to software engineering are held regularly in Europe, Asia, India, South America, the Pacific Rim, and most other areas of the world.

There are also innumerable smaller events, often focusing on specific areas of software engineering. In addition to the proceedings of these various technical meetings, journal publications serve as important vehicles for disseminating research results. The leading archival software engineering journals include the *ACM Transactions on Software Engineering and Methodology* and the *IEEE Transactions on Software Engineering*, the latter having been in publication since 1975. Numerous other journals and magazines are also devoted to software engineering.

The enormous change in software engineering practice that has occurred over the past decades is incontrovertible. Actual practice has evolved from one centered on batch runs of punched-card programs of perhaps several thousand lines of code in languages such as Fortran and Cobol, to one centered on rapid composition of multimillion-line concurrent, distributed, real-time systems composed of ever larger and better validated components, written in multiple modern languages. The worldwide software industry now generates hundreds of billions of US dollars in revenue annually and continues to expand in scope and revenue volume. Indeed, the software industry is increasingly acknowledged worldwide as a key driver of both social and economic growth.

In view of all this, it seems prudent to consider the interplay between these two major activities—software engineering research and practice. Toward that end, we provide an overall view of the motivations behind the Impact Project, the research methodology followed, and the project's development plan. We also address these issues by exploring more specific questions that separate study groups within the project have investigated.

A growing number of software engineering research and practice areas are under study. Here we present some projects whose studies have progressed the farthest so far.

0018-9162/08/$25.00 © 2008 IEEE

Impact of Software Engineering Research on Modern Programming Languages

Barbara Ryder, Rutgers University
Mary Lou Soffa, University of Virginia, Charlottesville

The production of software requires software engineering techniques such as specification, design, implementation, testing, and maintenance. The selection of programming languages as an implementation vehicle is essential to performing the last three phases of this process. The chosen programming languages must offer the application programmer the power to naturally express the task at hand in a disciplined manner. The requirements of expressiveness and software engineering methodology, evolving over time, clearly impact the design of programming language features.

To address the unique demands of application areas, thousands of programming languages have been designed. In practice today, Java, Ada, C, C++, Perl, Visual Basic, and Cobol are widely used across many application domains. Software engineering and the design of programming languages enjoy a synergistic relationship, each influencing the other.

Our study focuses on the impact of software engineering research on a select set of languages, namely Java, C++, Ada, and Visual Basic, and some of their significant features, including exceptions, concurrency, abstractions, inheritance, types, and control and visual programming concepts.

We found that software engineering research has had an impact on the development of most of these features. It has influenced the design of abstractions, including control, procedural and data abstractions; types, including polymorphism and generics; and exception handling and visual programming properties.

Interestingly, we found little evidence that software engineering research influenced the development of concurrency as found in the languages we considered. We also found that programming language design influenced software engineering research in, for example, the relation of software reliability to strong typing and software reuse to inheritance. Thus, the interaction between software engineering research and programming language design historically has been one of synergy.

As an example of our findings, consider the history of exception handling. PL/I, a programming language developed in the mid-1960s, used a statement, called an On condition, whose execution defined an exception handler at runtime, making compile-time checking of the completeness of exception handling impossible to accomplish. In the mid-1970s, after reviewing some existing exception mechanisms and reliability concerns, John Goodenough discussed requirements for good exception handling to help prevent and detect programmer errors.[1-3] He argued for the effectiveness of lexical association of handlers with operations that might throw an exception. Goodenough also advocated compile-time checking of the completeness of exception handling. Subsequent programming languages, including the research language CLU,[4] Ada,[5] C++,[6] and Java[7] all adopted his recommendation of lexically binding the handler to the statement whose execution directly or indirectly caused the exception. This is just one example of how software engineering researchers, concerned about the software reliability issue, directly contributed to programming language design.

References

1. J.B. Goodenough, "Language Design for Program Reliability," *IEEE Trans. Software Eng.*, June 1975, pp. 179-191.
2. J.B. Goodenough, "Structured Exception Handling," *Proc. ACM SIGPLAN/SIGACT Symp. Principles of Programming Languages*, ACM Press, Jan. 1975, pp. 204-224.
3. J.B. Goodenough, "Exception Handling: Issues and a Proposed Notation," *Comm. ACM*, Dec. 1975, pp. 683-696.
4. B. Liskov and A. Snyder, "Exception Handling in CLU," *IEEE Trans. Software Eng.*, Nov. 1979, pp. 546-558.
5. J.D. Ichbiah et al., "Rationale for the Design of the ADA Programming Language," *ACM SIGPLAN Notices*, June 1979, pp. 1-261.
6. A. Koenig and B. Stroustrup, "Exception Handling for C+," *The Evolution of C++: Language Design in the Marketplace of Ideas*, J. Waldo, ed., MIT Press, 1993.
7. G. Gosling, B. Joy, and G.L. Steele Jr., *The Java Language Specification*, Addison-Wesley, 1996.

Barbara Ryder *is a professor of computer science at Rutgers University. Contact her at ryder@cs.rutgers.edu.*

Mary Lou Soffa *is a professor and head of the Department of Computer Science at the University of Virginia, Charlottesville. Contact her at soffa@cs.virginia.edu.*

Configuration management is a multibillion-dollar industry that provides important support for software engineering practice. This area has benefited significantly from researchers' development of this area's techniques and tools.

Middleware is increasingly the glue used to assemble the large applications now pervasive in virtually all areas of human endeavor. Software engineering research has contributed to the maturation of middleware tools and technologies.

Programming languages are also a primary tool of software development. The evolution of language features has, in turn, influenced and been influenced by software engineering research.

Basic project management methods, such as the collection and analysis of various productivity metrics, are regularly employed in industrial practice, integral to modern software engineering. Research in this area has been published in software engineering texts for years.

Software engineering, following other engineering disciplines, has long used various design methods, models, and notations. These development aids have been influenced by the many years of research into software engineering methods and tools.

In summary, these efforts typify the extent and nature of software engineering research's impact on software engineering practice.

While we see some evidence of substantial impact, the contrary perception seems widespread. We believe that a clear understanding of this impact's effects has importance for researchers, practitioners, industry, and government.

Thus, it seems timely to replace conjecture and prejudice about impact or the lack of it with solidly researched hard evidence. This research project therefore aims to determine what impact, if any, software engineering research has had on practice and to substantiate any such impact with scholarly research appropriate to a scientific community.

PROJECT OBJECTIVES

The general goal of evaluating software engineering research's impact on software engineering practice can be decomposed into the more precise and specific objectives that drive our project activities.

The project provides a solid and scholarly assessment of the impact software engineering *research* has had on software engineering *practice*. The assessment takes the form of a series of studies and briefings, each involving literature searches and, where possible, personal interviews. The assessments undergo careful review by the project members and by anonymous outside reviewers to assure it measures up to the highest standards of scholarship.

The project informs the software engineering research community, the software engineering practitioner community, other scientific communities, and both private and public funding sources about the return on investment from past software engineering research. Using the assessment, project members engage in publication, lecturing, and briefing activities designed to maximize the dissemination of its findings.

Some early reports have already sparked lively debate, and this is seen as a desirable project outcome. The studies have confirmed our initial expectation that impact can be achieved in a variety of ways and will likely be measured differently from differing perspectives. Our early experiences have also confirmed initial expectations that dissemination of project findings can lead to a sharpening and convergence of opinions about how to measure and assess impact. We anticipate that this will ultimately lead to improvements in, and increased satisfaction with, technology transition activities.

Technology transition is most effective and best expedited when research and commercialization maintain a close synergy over an extended period.

The project clarifies both the scientific issues and the engineering activities that provide the complementary foci of software engineering research, both to focus the software engineering research community's activities and to improve perceptions of the discipline's nature. Much of the community's work is expressed through development of prototype methods, tools, and systems. Experience, both in software engineering and diverse other disciplinary areas, has indicated that the impact of these prototypes might take 20 years to manifest.[2]

Early results confirm this considerable time lag, but they also suggest a far more complicated picture of technology transition. These findings indicate that technology transition is most effective and best expedited when research and commercialization maintain a close synergy over an extended period. Early observations such as these encourage our belief that the project will help establish a clearer and more detailed understanding of which transition approaches have been most successful and identify the ways in which research impact has been most effective.

The project uses these assessments and clarifications to project key future opportunities and directions for software engineering research and practice. We expect that the historical perspectives of our work will increasingly provide the basis for a clear, objective assessment of the state of the art and current practice in software engineering. This, in turn, should provide a solid basis for making projections. In particular, we expect that a clearer view of software engineering research's history should provide a clearer view of the key challenges and opportunities to be faced in the immediate, intermediate, and long-range future. Historical analysis should also help identify which research modalities were relatively more successful than others. Likewise, this view should help indi-

Software Configuration Management

Jacky Estublier, Grenoble University, France

The impact of the software configuration management discipline on the software industry at large is easy to discern: A successful, billion-dollar marketplace of SCM tools and services has arisen, and SCM tools have become so pervasive that virtually all major projects use and rely on them. SCM concepts and techniques also have had major influences in other fields like interactive development environments, Internet protocols, and now product-line architectures.

To distill the role and impact of SCM research on practice, a team was assembled consisting of the primary architects from two major SCM system vendors, two academic researchers, and three contributors with both industrial and academic experience in the field. With such a balanced and knowledgeable team, whose members knew each other fairly well, it should have been easy to identify the impact of research on vendor tools. It was not.

Early in our discussions, researchers claimed ownership of most ideas, dismissing tool realization as "engineering common sense." By the same token, vendors claimed they invented everything they needed, dismissing research concepts, ideas, and architectures as "engineering common sense." From the intensive discussions that followed emerged a much more balanced perspective.

This study demonstrated that accurately pinpointing the origins of most ideas is difficult. Therefore, our report relies on publications and actual systems to trace research contributions and their uses in SCM systems. In doing so, it became clear that both academic and industrial research have made significant research contributions, and that many original ideas take a long time and require deep reworking and reengineering before they can be successful in industrial practice.

After discussing the details of the research contributions and failures at length, our report concludes by making the observation that the international academic community serves an important role by providing a forum in which ideas can be published, discussed, and scrutinized. Continual attendance at the SCM workshop series by chief architects of some of the most influential SCM systems—as well as the flow of researchers between industry and academia—have been the major means by which research impact occurred in our field, and to a large extent explains the software configuration management success story.

Jacky Estublier *is a senior scientist, leading a software engineering research team at Grenoble University. Contact him at jacky.estublier@imag.fr.*

cate how to more effectively transition these research directions into industrial practice.

The Impact Project thus has several beneficial effects. It

- helps software engineers better understand their field of activity, explaining the origins of the practice's current state in academic and industrial research and in industrial innovation;
- recognizes the contributions that various research ideas, techniques, and technologies have made to various tools and methods;
- improves education by clarifying the continuous research and development threads that have led to the state of the art; and
- provides investment organizations with insights into models of past research modalities that have led to success, with understandings and appreciations of research and its impacts upon practice, and with suggestions for appropriate measures of research success.

We anticipate that our project will also encourage others to embark on similar exercises for other computing fields. The lack of historical perspectives is a common problem we share with most current high-technology disciplines. It is especially important to recognize if and how research has affected progress, both within academia and industry, for technical fields where evolution advances with extreme rapidity and generates continuous market transformations.

PROJECT ORGANIZATION AND RESEARCH METHOD

We intend the Impact Project to be highly inclusive in nature, involving the efforts of all constituencies within the software engineering community, as well as others whose interests and expertise overlap with software engineering. Thus, the project includes not only academic researchers, but also industrial researchers and a broad spectrum of software engineering practitioners. To represent that the project should belong to the entire software engineering community, it was initiated under the auspices of ACM SIGSOFT, which represents the community at large. Funding for the project has been secured from the US National Science Foundation, the UK's Institution of Engineering and Technology, the IEEE Computer Society, and the Japanese government.

The Impact Project's scope encompasses some of the key areas of software engineering practice. Because limited project resources make it impossible to address all software engineering areas, it has been necessary to select some that seem representative.

In each area, the project is producing one or more main reports and several ancillary artifacts. Each report is authored by a team consisting of as many as six to eight academic and industrial experts, led by one or two authors responsible for the work's direction, integrity, and completeness.

Each lead author also belongs to the project Steering Committee, which serves as an internal review board for the reports issued by the entire project. This group sets directions, policy, and structure for the various reports and briefings. It also secures preliminary reviews of the reports and mediates discussions and conversations between authors and reviewers.

Once the internal review process has been completed and a report has been released for external publication, it is subjected to the normal anonymous peer-review process. The Steering Committee is headed by an Executive Committee that has oversight responsibility for the entire project.

The Impact Project's scope encompasses some of the key areas of software engineering practice.

In general, the approach taken to researching impact in a particular area is to begin from the practice and work backward in time to find its roots, if any, in research. The project explicitly eschews the reverse, less sound approach of starting from the research and working forward toward a practice that could be argued to employ the research; such an approach could be seen as an attempt to justify individual research ideas or results rather than to find genuine impact. Notice that by working back from practice, the effect is to identify only those research activities that have traceable impact on it.

Before producing finished reports and other materials, the Impact Project organizes open discussion of its work and findings at the major technical meetings attended by the software engineering community, through panels and ad hoc project presentations, and by disseminating early versions of the reports in newsletters, such as ACM SIGSOFT's *Software Engineering Notes* and the SIGSOFT website (www.acm.org/sigsoft/impact/).

The major output of each study is submitted as a regular article to ACM *Transactions on Software Engineering and Methodology* (TOSEM), and indeed the first two of these reports were published in the October 2005 issue,[3] with the third report scheduled for publication in October 2008. Different summaries and abstracts of these studies are also being made available to reach diverse audiences through various existing publication and presentation venues.

A further important aspect of this research is its contribution to the history of science and technology in the specific area of software engineering. From the outset, it has been clear that the project's success depends on adoption of sound approaches to historical research methods, as this will ultimately affect the project findings' validity. For this reason, the project has as an active participant Michael Mahoney, of the Princeton University Department of History, a leading expert on the history of science and technology who has written extensively on the history of computer science.

PROJECT STATUS AND PRELIMINARY FINDINGS

The two study reports published in ACM TOSEM address configuration management[4] and modern programming languages.[5] The October 2008 report will address middleware.[6] A preliminary report on runtime assertion checking was published in ACM *SIGSOFT's Software Engineering Notes*[7] and will shortly enter the review cycle for ACM TOSEM. Other reports address the following areas: reviews and walkthroughs, lead authors Dieter Rombach and Ross Jeffery; cost and economic models, lead authors Barry Boehm and Ricardo Valerdi; and software architecture, lead author Richard Taylor.

The five sidebars offer brief summaries of some study reports. While each describes findings specific to the individual study, they also suggest some findings that could apply quite generally to these and many other studies still under way. These preliminary studies include the following findings.

Finding 1

Software engineering research has significantly affected software engineering practice. The evidence accumulated in the course of the studies completed and those well under way supports a finding that research has positively impacted practice. Successful deployment of technologies that support practice clearly requires many types of contributions from many types of participants. But the studies also make it clear that researchers' participation has been of considerable importance.

Finding 2

Lasting impact seems to come most readily from ongoing interactions between research and practice. While technology transition might be thought of as moving a key idea or prototype out of the research environment and into the commercial sector, our studies indicate that success in practice seems to require continued interactions between research and practice, resulting in continued upgrading and improvement of a capability.

Inspections, Reviews, and Walkthroughs

Dieter Rombach and Marcus Ciolkowski, Fraunhofer Institute for Experimental Software Engineering

Inspections, reviews, and walkthroughs are integral parts of most industrial software development lifecycle models and standards today. However, reported effects upon quality improvement or cost reduction vary widely. Companies reporting repeatable success tend to employ well-defined techniques for the analysis of code and other documents (often referred to as reading techniques) and embed inspections, reviews, or walkthroughs into a measurement-based feedback system to facilitate motivation and optimization. Successful companies such as Allianz, NASA's Software Engineering Laboratory, Motorola, and IBM report up to 95 percent defect detection rates before testing, overall cost reduction for newly developed lines of code by 50 percent, and even shortening of delivery times by up to 50 percent.

Various software engineering research results have enabled these effects. They include advances in: (1) process research resulting in entry/exit criteria-based processes such as Fagan inspections[1] or reading techniques such as perspective-based reading for informal documents[2]; (2) formal methods research resulting in closed-form mathematics-based semantics suitable for stepwise inspections such as the functional semantics approach[3]; (3) in-project measurement such as the goal-question-metric paradigm,[4,5] allowing for the TQM-style optimization of reading techniques and inspection effectiveness[6]; and (4) empirical methods research in software engineering, enabling the careful study of cause-effects of applications inspections, reviews, and walkthroughs,[7] starting with early studies showing the variance of effects[8] as well as the effects of specific techniques in a concrete context.[9]

The success of human-based software processes such as inspections depends on a combination of software engineering results in methods and empiricism. Without the development of closed-form mathematics-based notations and algorithmic reading techniques on the one hand, and the establishment of an infrastructure for software engineering measurement and empirical studies to fine-tune and optimize the effects toward a specific environment on the other hand, success would still be varying and unsustainable. Companies aiming at sustainable effective use of inspections, reviews, or walkthroughs should mimic the transfer guidelines of successful companies.

References

1. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182-211.
2. V.R. Basili et al., "The Empirical Investigation of Perspective-Based Reading," *J. Empirical Software Eng.*, vol. 2, no. 1, 1996, pp. 133-164.
3. H.D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems," *Computer*, June 1988, pp. 23-36.
4. V.R. Basili, G. Caldiera, and H.D. Rombach, "Goal Question Metric Paradigm," J. Marciniac, ed., *Encyclopedia of Software Eng.*, John Wiley & Sons, 1994, pp. 528-532.
5. R. van Solingen and E. Berghout, *Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*, McGraw-Hill, 1999.
6. F. McGarry, "What Is a Level 5?" *Proc. 26th Ann. NASA Goddard Software Eng. Workshop*, IEEE CS Press, 2001, pp. 83-90.
7. H.D. Rombach, V.R. Basili, and R.W. Selby, eds., *Experimental Software Engineering Issues: A Critical Assessment and Future Directions*, Springer-Verlag, 1993.
8. V.R. Basili and R.W. Selby, "Comparing the Effectiveness of Software Testing Techniques," *IEEE Trans. Software Eng.*, vol. 13, no. 12, 1987, pp. 1278-1296.
9. R.W. Selby, V.R. Basili, and F. Baker, "Cleanroom Software Development Approach: An Empirical Evaluation," *IEEE Trans. Software Eng.*, vol. 13, no. 9, 1987, pp. 1027-1037.

Dieter Rombach *is a professor of computer science at the University of Kaiserslautern and executive director of the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany. Contact him at dieter.rombach@iese.fraunhofer.de.*

Marcus Ciolkowski *is a researcher at Fraunhofer IESE and a PhD student at the University of Kaiserslautern. Contact him at ciolkows@iese.fraunhofer.de.*

While a single interesting and successful research prototype can make a noticeable contribution to practice, practice will likely make continuing demands for improvements. In cases where the research community has remained in continued contact with these demands, industrial response has often been expedited and facilitated, leading to a more sustained and significant impact upon practice. Thus, for example, the software configuration management study noted that the earliest SCM tools drew upon research prototypes and ideas. But the early prototypes quickly stimulated demand for substantial improvements. Some key ideas for these improvements, which have kept SCM active as a marketplace line of business, have come from continued contact with SCM researchers.

Similarly, the appearance of exception management features in early programming languages led to a clearer

understanding of the need for improvements, which have been appearing in later languages. The Ryder and Soffa report documents how observation of the shortcomings found in early exception management stimulated research findings that researchers then picked up and incorporated into subsequent, more successful, language features. This synergy seems to be continuing, leading to a more sustained impact of research on practice.

Finding 3

Research impact might not be fully felt for at least 10 years. These studies also provide both illustrations and explanations of why it typically takes at least 10 to 20 years for a good idea to move from initial research to final widespread practice. This observation has been made before, most notably by William Riddle and Sam Redwine,[2] who suggested that successful transition into practice takes approximately 17 years.

It typically takes at least 10 to 20 years for a good idea to move from initial research to final widespread practice.

Our studies indicate that at least a few cycles of prototype development and marketplace evaluation are needed for widespread adoption of a technology. Given that each cycle will likely take a few years, the entire process typically requires decades.

Finding 4

Continued support for sustaining a vigorous research community is required. Each study group has struggled to determine the exact transition paths for research ideas and the exact ways in which developers have inserted these ideas during the steady evolution of practice. But, while the details might vary from one technology area to another, research ideas invariably have been shown to play identifiably important roles in this evolution.

In several studies, we found that researchers commonly communicate key ideas or the precise identification of key problems—which sometimes play a critical role in developing a successful practical tool—during hallway conversations at technical meetings, even though they might not be contained in the meetings' technical presentations. Such ideas were often described as being "in the air." The impact of such ideas is hard to deny, even though no single conversation can be identified as the vehicle for such transitions.

Clearly, these situations complicate our efforts to document the precise flow of ideas and technology. We note, however, that these findings underscore the key role technical meetings have played in creating the "air" in which such ideas float. By implication, this, in turn, underscores the importance of the communities that maintain such meetings and meeting series. This observation of the fundamental importance of maintaining a community was reported most clearly in the SCM paper and seems to be particularly important and relevant for the Impact Project, as it is researchers who have traditionally assumed responsibility for maintaining such communities.

The support needed takes several forms. Clearly, financial support is essential as it enables the training of graduate students, the funding of community meetings, and the travel that expedites collaborations between researchers and their groups. But the need goes beyond money and includes the commitment of funding sources to maintain research support over extended periods. The observation that successful transition of research ideas into practice can take decades suggests that disruptions and discontinuations of funding during that time can be expected to slow the entire process of technology maturation and dispersion.

A corollary to this finding is that targeted funding for technology transition might be less effective than broad-based community funding support. Although the Impact Project has not focused on interactions that have proven ineffective, the project has started collecting anecdotal evidence that overtly explicit attempts to fund the transition of specific research ideas have enjoyed mixed success. Instead, the emerging picture suggests that diverse interactions of varying degrees of formality taking place over extended periods might have been quite effective in assuring a positive impact of research on practice. Perhaps broad efforts to maintain the strength of communities that foster this broad spectrum of interactions could ultimately be most effective. This suggests the importance of support aimed specifically at a research community's overall health.

Finding 5

A research community must provide nurturing in a variety of ways. While external support is vital, the community itself also has an obligation to provide nurturing and support for its own members. Here, too, this support can take a variety of forms. We have observed that maintaining a series of meetings and workshops seems to be quite important because it provides vehicles for putting important ideas "in the air" and maintaining them there. Maintaining publication venues is similarly important, providing a forum for disseminating ideas and offering mechanisms for the sort of rigorous review that identifies ideas that seem to be particularly worthy of note and attention.

Our studies also show the importance of studies that track the movement of people in technology development. The middleware study provided good examples of cases where people moved between research and development situations and brought with them insights, ideas, and approaches that materially advanced the maturation of ideas. The community can and should

Middleware

Wolfgang Emmerich, University College London
Mikio Aoyama, Nanzan University, Japan

Middleware, a term first introduced at the NATO Software Engineering Conference in 1968, now commonly subsumes application servers, application integration solutions, message-oriented middleware, and transaction processing monitors. According to a recent Gartner market analysis report,[1] the middleware market comprised annual license sales of US $8.5 billion in 2005. While that is a significant market in its own right, it only represents the tip of the iceberg for overall economic activity driven by middleware technology.

The Gartner study does not reflect the considerable amount of open source middleware in use. For example, the JBoss application server had 10 million installations in 2005, with the install base set to increase considerably following its acquisition by RedHat. The study also does not include the revenue of middleware vendors and their partners as a result of providing professional services.

An assessment of the true economic impact of middleware must also include the wealth created by e-commerce, finance, telecommunications, and governmental applications whose construction middleware enables.

Our study considers the impact of research on the most important classes of middleware, namely Web services and enterprise service buses, application servers, message-oriented middleware, distributed object systems, and remote procedure call systems. We do not attempt to provide a complete account of all impact that research might have on middleware technology. Instead, we provide detailed evidence for the existence of the impact of research results on a few aspects of middleware standards and products.

We identify research results that had a significant and lasting influence on the middleware technology omnipresent today. These were produced in computer science departments—including those at Brown, Cambridge, Carnegie Mellon, Newcastle, MIT, Vrije, and Washington, and in industrial research labs such as APM, AT&T Bell Labs, DEC Systems Research, HP Labs, IBM Research, and Xerox PARC.

Research's impact on middleware technology was neither direct nor immediate. The impact traces that we found frequently cross the boundaries between operating systems, distributed systems, databases, programming languages, and software engineering. For example, one root of message-oriented middleware is the broadcast message server originally devised for the Field software development environment; the notion of a transaction was first devised as operating system concepts, then nurtured by the database community, and subsequently adopted in transaction processing middleware and application servers.

The time it takes for research results to have an impact is substantial. Our findings confirm earlier studies[2] that it takes between 15 and 20 years from initial publication of an idea until that idea becomes widely used in products.

References

1. J.M. Correira and F. Biscotti, *Market Share: AIM and Portal Software, Worldwide, 2005*, Gartner market research report, June 2006; www.gartner.com/DisplayDocument?ref=g_search&id=492790.
2. S.T. Redwine and W.E. Riddle, "Software Technology Maturation," *Proc. 8th Int'l Conf. Software Eng.*, IEEE CS Press, 1984, pp. 189-200.

Wolfgang Emmerich *is a professor in the Department of Computer Science, University College London. Contact him at w.emmerich@cs.ucl.ac.uk.*

Mikio Aoyama *is a professor in the Department of Information and Telecommunication Engineering at Nanzan University, Japan. Contact him at mikio.aoyama@nifty.com.*

play an important role in facilitating such movement. Some movement of people from one location to another might result from traditional approaches to job seeking. An effective community of researchers and practitioners, however, can expedite the identification of promising matches and the movement of people that are relatively more likely to be successful.

Finding 6

The interplay between research and practice can be difficult to determine precisely and communicate clearly.

Our early studies have demonstrated that impacts occur in many ways. The transmission of a research idea to a specific commercialization might be possible to identify with reasonable certainty. And the identification of a difficult industrial problem as a research topic likewise might be reasonably easy to document. But these straightforward forms of impact are far from the norm. Often ideas are disseminated as research students migrate from research labs to industry. Sometimes problems and approaches move from industry to research labs when senior developers accept research positions. At other

times, ideas are exchanged through consulting arrangements, sometimes through conversations—either during visits or technical meetings. Sometimes the intellectual debt has been acknowledged in product literature, but that is unusual.

In some cases, the debt has been demonstrated in our reports, even when it has not been acknowledged. As noted, the full scope of these interplays spans years and decades, with the transitions between research and practice taking different forms at different times during the development of a practical technology.

In some cases, we have found that there can be honest differences of opinion about how much credit the research community should be given for its impact. The Impact Project currently takes no position about how much credit should be given, instead engaging simply in documenting the various forms of the flow of ideas, tools, technologies, and people.

The Impact Project hopes to introduce some clarity into the ongoing discussion of the impact of software engineering research.

Designing a clear and precise form for documenting these flows is presenting some interesting challenges in itself. Each report that is either completed or in the late stages of completion contains at least one diagrammatic attempt to summarize impact. The variety of these diagrams speaks to the complex and elusive nature of research impact on practice. Indeed the elusiveness of a clear picture might be a significant contributor to the widespread differences in perception regarding the nature and importance of research impact. We emphasize, however, that, while these pictures of impact might be complex and elusive, they nevertheless support an increasingly clear conclusion that research impacts upon practice have been real and substantial.

Finding 7

It seems more beneficial to identify the many and varied ways in which research and practice have influenced each other than to seek precise criteria for determining what should be considered "research impact." Most fundamentally, we have found it difficult to identify uniform standards of success for the different technologies we have studied. In some cases, such as SCM and middleware, we have noted that researchers have spent billions of US dollars annually on tools and technologies.

In other cases, however, such as modern programming languages, monetary expenditure seems to be a less appropriate measure of success. As more technology becomes available in the form of free downloads, the use of expenditures as a measure of success will become increasingly questionable.

Thus, the impact of technology areas can be difficult to assess. Each study area has been charged with identifying its own measure of success, and these measures are identified and justified separately in each report.

As a corollary, it is correspondingly difficult to evaluate the various individual contributions to an entire area's success. The SCM report, for example, notes that publication of an idea that finds later use in a popular product should qualify that idea, and its author, with having impact. The report also notes that commercial enterprises will likely claim that the engineering of a successful product has impact, and that the fundamental ideas around which the product has been engineered are less deserving of credit for impact. Clearly, both positions are extreme, and we have sought a more balanced view of how impact should be evaluated.

It is a triviality to observe that practice moves forward only when innovative ideas are effectively integrated with strong engineering to produce robust and effective products, and that this requires contributions from a spectrum of participants ranging from researchers through product engineers, and potentially involving marketers, early adopters, and support personnel. What these reports seem to make manifestly clear is that it is presumptuous and unfair for any party to claim overriding importance. Rather, each must acknowledge the role that others play in effecting impact upon the universe of practice. These reports serve to document just what that impact has been and how it has been made.

Throughout the Impact Project, we hope to introduce some clarity into the ongoing discussion of the impact of software engineering research. After 40 years, it seems timely to assess its benefits and efficacy. We believe that the project will have important positive benefits for the software engineering community, as well as the companies that rely on software technology, the agencies charged with assuring the health of software research and practice, and the public at large. That being the case, it seems important to address the question of how the Impact Project's success might itself be determined.

This difficult issue could be simply but trivially measured by the number of reports and the quality of the venues in which they are published. An important but difficult evaluation of the project will, however, be based upon the clarity, efficiency, and professionalism of future discussions regarding software engineering research's effectiveness. The Impact Project aims to address the need for clear and objective data so that discussions of matters such as research impact can proceed based on carefully researched and validated evidence. We anticipate that the project will produce data and factual material that will indeed be useful in expediting the community's arrival at consensus about how well research has

Runtime Assertion Checking

Lori A. Clarke, University of Massachusetts, Amherst
David S. Rosenblum, University College London

Assertions are known to be one of the most useful automated techniques available for detecting faults and providing information about their locations, including finding intermediate states that deviate from their expected value. As our report describes, assertions have a long and distinguished history in the annals of software engineering and programming language design.

Indeed, the origins of formal, logical assertions about program behavior predate computers. Konrad Zuse's Plankalkül provided the earliest reference: a visionary proposal for a programming notation developed in the mid-1940s that included support for assertions among its many novel ideas.[1] Herman H. Goldstine and John von Neumann's work on reasoning about programs used the term "assertion" for documenting invariants in algorithms.[2] Turing also advocated using assertions to document the states that can be associated with various points in a routine, indicating that this information is subsequently useful in determining the correctness of the whole program.[3]

From these primitive beginnings emerged now-classic foundational work on program verification, in which assertions were used as a means of stating expected or desired program properties as a necessary step in constructing formal, deductive proofs of program correctness.[4-6] Assertions have found many other applications in software engineering over the years.

One application area having a significant impact on development practice, and the one on which we focus in our report, is their use for automated runtime fault detection, in which formal assertion checks are instrumented into a program for execution along with the program's application logic.[7-10] The state of the technology is at the point now where assertion constructs have become first-class constructs of widely used programming languages such as Java and C#, where assertions are supported by popular commercial and open source tools such as Jtest and JMS, and where the systematic use of assertions in development is systematized in the notion of design-by-contract.[11]

Readers can find a fairly large amount of anecdotal evidence and testimonials about the use and effectiveness of assertions in development practice, but there has been relatively little quantitative information published. C.A.R. Hoare reported that about 250,000 lines of the source code of Microsoft Office are assertions, representing roughly 1 percent of the source code.[12] And Patrice Chalin recently surveyed several software projects to determine the density of assertion statements in source lines, reporting an average assertion density of 3.27 percent in the surveyed proprietary projects, 5.10 percent in the surveyed open source projects, and 6.42 percent in the surveyed Eiffel projects.[13]

As tools that process assertions, such as Jtest, become more widespread, we expect developers to enjoy a corresponding increase in the benefits of using assertions during development.

References

1. K. Zuse, *Der Plankalkül,* Gesellschaft für Mathematik und Datenverarbeitung, no. 63, BMBW-GMD-63, 1972.
2. H.H. Goldstine and J. Von Neumann, "Planning and Coding Problems for an Electronic Computing Instrument," *John von Neumann, Collected Works*, A.H. Taub, ed., Pergamon Press, 1963, pp. 80-235.
3. A. Turing, "Checking a Large Routine;" www.turingarchive.org/browse.php/B/8.
4. J. McCarthy, "Towards a Mathematical Science of Computation," 1963; www-formal.stanford.edu/jmc/towards.pdf.
5. R.W. Floyd, "Assigning Meaning to Programs," *Proc. Symp. Applied Mathematics*, Am. Mathematical Soc., 1967, pp. 19-32.
6. C.A.R. Hoare, "An Axiomatic Basis of Computer Programming," *Comm. ACM*, Dec. 1969, pp. 576-580.
7. S.S. Yau and R.C. Cheung, "Design of Self-Checking Software," *Proc. Int'l Conf. Reliable Software*, ACM and IEEE CS Press, 1975, pp. 450-457.
8. D.C. Luckham, *Programming with Specifications: An Introduction to Anna, a Language for Specifying Ada Programs*, Springer-Verlag, 1990.
9. B. Meyer, *Eiffel: The Language*, Prentice Hall, 1991.
10. D.S. Rosenblum, "A Practical Approach to Programming with Assertions," *IEEE Trans. Software Eng.*, vol. 21, no. 1, 1995, pp. 19-31.
11. B. Meyer, "Applying 'Design by Contract,'" *Computer*, Dec. 1992, pp. 40-51.
12. C.A.R. Hoare, "Assertions: A Personal Perspective," *IEEE Annals of the History of Computing*, vol. 25, no. 2, 2003, pp. 14-25.
13. P. Chalin, *Ensuring Continued Mainstream Use of Formal Methods: An Assessment, Roadmap and Issues*, D.S.R. Group, ed., Concordia University, 2005.

Lori A. Clarke *is a professor of computer science at the University of Massachusetts, Amherst. Contact her at Clarke@cs.umass.edu.*
David S. Rosenblum *is a professor of software systems in the Department of Computer Science at University College London. Contact him at d.rosenblum@cs.ucl.ac.uk.*

served practice and how ongoing and proposed research can be ever more effective.

The project has also indicated the need for further investigation beyond its current scope. Preliminary observations suggest that the development of research ideas into practice requires resource levels that grow over time. It seems important to verify this observation and, if it is found to be true, to try to determine the rate at which support has grown over time in cases of the successful transition of research ideas into practice.

The Impact Project has not identified the sources of support for the key contributions and contributors in each of the studies. However, the importance of an active and fertile community cannot be overemphasized. Finding 3 suggests that continuity of community support over a period of 10 to 20 years seems crucial for expediting the successful transition from research into practice. It would be interesting and useful to determine whether the successful transitions that have been studied in this project did indeed benefit from this sort of constancy and continuity of support.

Acknowledgments

The work of this project is largely unfunded, relying on pro bono contributions of time and effort by the authors, especially the lead authors, of the various study areas. That said, funding is still required to cover such costs as communications, travel to meetings, and publication expenses. The project participants gratefully acknowledge the financial support for these meetings obtained from the US National Science Foundation, ACM SIGSOFT, the IEEE Technical Committee on Software Engineering, the Institution of Engineering and Technology (IET) in the UK, and the government of Japan. Additional support for field trips to interview historically important individuals and examine primary source materials in the form of corporate documents and records was also necessary. We gratefully acknowledge the support of these individuals and corporations in granting access to this primary source material.

References

1. P. Naur and B. Randell, eds., *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*, NATO, Scientific Affairs Division, 1969.
2. S. Redwine and W. Riddle, "Software Technology Maturation," *Proc. 8th Int'l Conf. Software Eng.*, IEEE CS Press, 1985, pp. 189-200.
3. L. Osterweil et al., "Editorial," *ACM Trans. Software Eng. and Methodology*, vol. 14, no. 4, 2005, pp. 381-382.
4. J. Estublier et al., "Impact of Software Engineering Research on the Practice of Software Configuration Management," *ACM Trans. Software Eng. and Methodology*, vol. 14, no. 4, 2005, pp. 383-430.
5. B.G. Ryder, M.L. Soffa, and M. Burnett, "The Impact of Software Engineering Research on Modern Programming Languages," *ACM Trans. Software Eng. and Methodology*, vol. 14, no. 4, 2005, pp. 431-477.
6. W. Emmerich, M. Aoyama, and J. Sventek, "The Impact of Research on Middleware Technology," *ACM Trans. Software Eng. and Methodology*, to appear Oct. 2008.
7. L. Clarke and D. Rosenblum, "Historical Perspective on Runtime Assertion Checking in Software Development," *SIGSOFT Software Eng. Notes*, vol. 31, no. 3, 2006, pp. 25-37.

Leon J. Osterweil *is a professor of computer science and former dean of the College of Natural Sciences and Mathematics at the University of Massachusetts, Amherst. His research interests are in precise definition and analysis of processes, healthcare process improvement, and online dispute resolution. Osterweil received a PhD in mathematics from the University of Maryland, College Park. He is a member of the IEEE Computer Society and a Fellow of the ACM. Contact him at ljo@cs.umass.edu.*

Carlo Ghezzi *is a professor of software engineering and the Rector's Delegate for Research at Politecnico di Milano. His research interests include specification, design, analysis, and verification of dependable and evolvable software. Ghezzi is a Fellow of the IEEE and the ACM and a member of the Italian Academy of Science (Istituto Lombardo). Contact him at carlo.ghezzi@polimi.it.*

Jeff Kramer *is a professor of computing and dean of the Faculty of Engineering at Imperial College London. His research interests include rigorous techniques for requirements engineering, software analysis and software architectures, particularly as applied to distributed and adaptive software systems. Kramer is a Fellow of the ACM. Contact him at j.kramer@imperial.ac.uk.*

Alexander L. Wolf *is a professor in the Department of Computing at Imperial College London, where he is head of the Distributed Software Engineering Research section. His research interests are in the principles and development of technologies to support the engineering of large, complex software systems, emphasizing software engineering, distribution, and networking. Wolf received a PhD in computer science from the University of Massachusetts. He is a Fellow of the ACM and holds a UK Royal Society-Wolfson Research Merit Award. Contact him at a.wolf@imperial. ac.uk.*