Diffusing Software-Engineering Methods

Sridhar A. Raghavan and Donald R. Chand, Bentley College

Technology transfer has become a major concern in software engineering. But without a framework for diffusing technology among practitioners, the transfer problem cannot be solved.

Technology transfer has become an important area of concern in software engineering. Yet we do not have good frameworks for systematically studying the nature of software-technology transfer, nor for developing sound and prescriptive strategies for facilitating successful diffusion.

Software-engineering technology, as Peter Freeman has defined it,1 consists of any concepts, tools, techniques, methods, and generalized technical structures that software engineers use to create software-intensive systems that meet stated technical, economic, and social objectives. Examples of this technology include modularization techniques, design methodologies, review techniques, programming environments, estimation techniques, and metrics.

Diffusion is the process of transferring technology from those who develop it to those who apply it. Although diffusion is important in all disciplines, it assumes critical importance in applied disciplines such as software engineering where the primary mission is advancing the state of the practice. Therefore, the success of software engineering as a discipline will depend not only our ability as a community to come up with creative and innovative methods to address the problems of software development but also on how successfully we diffuse these methods into practice.

From this perspective, the track record of the software-engineering discipline is far from reassuring. Although software-engineering technology has advanced rapidly, it has not significantly affected the state of the practice of software development. It is thus not surprising that technology transfer is increasingly becoming an area of concern in the software-engineering community.

This increased concern can be seen in IEEE workshops on software-technology transfer, the US Defense Dept.'s 1984 es-
establishment of the Software Engineering Institute, and the establishment of special private consortia like the Microelectronics and Computer Technology Corp. and the Software Productivity Consortium with explicit responsibility for transferring software technology to their member institutions and operating units.

Despite the increasing interest and the growing literature on the topic, software technology transfer remains a poorly understood area of software engineering. There is a tendency on the part of software-engineering innovators to oversimplify the problems of technology transfer. They often attribute these problems to management pressures that interfere with the use of software-engineering practices or to general apathy and resistance to change on the part of potential users.

Although these views may be true in specific instances, they usually represent only a small part of the overall problem. A good understanding of the underlying processes and problems is needed to effectively tackle the technology-transfer problem.

How can we as software engineers understand these problems? First, we need to identify a conceptual framework that can provide a systematic understanding of diffusion processes. Second, we need to identify priority areas through which the software-engineering community can affect the successful diffusion of its innovations.

Existing literature. The existing literature on software-technology transfer fails to address these needs adequately. Much of this literature has a pragmatic orientation in the sense that it outlines strategies to ease the technology-transfer problem, but it provides little help in advancing a systematic framework for studying and understanding the diffusion processes.

For example, Freeman has suggested several strategies for software-engineering technology transfer based on his experience in conducting professional development seminars and establishing software-engineering guidelines in industrial organizations. He advocates that you

- view technology transfer as a system,
- separate the distinct functions of technology transfer,
- differentiate the types of technology and the intended recipients,
- make sure that there are effective feedback paths,
- match the technology to the actual needs,
- obtain broad support for the technology,
- treat each technology transfer as a project,
- build on existing technology,
- deal with organizational change explicitly,
- exploit familiarity to overcome fear of change, and
- continually study the process and develop new strategies.

But while Freeman’s work provides useful guidelines, it does not offer insight into the underlying diffusion processes.

Reports by Louis Tornatzky and colleagues and by Walt Scacchi and James Babcock have dealt better with these underlying processes. They review a body of literature relevant to the study of the diffusion of innovations. However, their focus is too broad for software-technology transfer.

Tornatzky and colleagues focused on the larger issues of managing organizational innovations, so the diffusion process received only limited attention and is buried in a mass of other material. It was also not oriented toward software technology.

The Scacchi and Babcock report, on the other hand, was more focused on software-technology transfer. It developed a conceptual framework, called the Software Innovation Life Cycle, for systematically organizing the available empirical studies on the design, development, adoption, and implementation of information systems in organizations. However, this framework provided only a limited understanding of the diffusion processes because diffusion is only one of the steps in the life cycle and is not covered in detail. Furthermore, the understanding the paper developed is based solely on the few available studies on user acquisition of information-system products rather than on the adoption of software technology by software developers.

Despite these drawbacks, the two surveys do provide a good frame of reference for considering how to diffuse software-engineering technology. This article builds on their work but takes a narrower focus — the diffusion process — and tries to relate it directly to the context of software technology.

Looking at other fields. Although technology transfer and diffusion are relatively new areas of study in software engineering, they are not new to other disciplines. Sociologists and management scientists have studied them extensively for several years. Everett Rogers’s framework for diffusion of innovations dominates this literature.

His framework is widely accepted because of its success in predicting and explaining the diffusion process for a wide variety of innovations. These include educational innovations like programmed instruction and new mathematics that are similar to software-engineering innovations in that they involve transfer of knowledge and changes in how people think.

We strongly believe that studying and adapting Rogers’s framework to the software-engineering context could significantly enhance our overall understanding of the diffusion process and help us develop effective strategies to successfully diffuse software-engineering innovations.

The Rogers framework

Rogers’s framework for diffusion of innovations was originally based on extensive study of agricultural innovations. It was later enriched and validated through many empirical studies in other settings. It is a comprehensive framework that can be used descriptively or prescriptively.
The main elements of the framework are innovation, the communication process, the adoption process, and the social system.

Innovation. In the context of diffusion, innovation is an idea, practice, or object that is perceived to be new by the people who are targets for its adoption. Rogers identified five key characteristics of innovations that directly affect an innovation’s rate of diffusion: relative advantage, compatibility, complexity, ability to try it out (called “trialability” in the management literature), and visibility.

Relative advantage measures how much better an innovation is perceived to be compared to the product or the idea it supersedes. If an innovation is perceived to have a high level of relative advantage by the people who will be instrumental in adopting it, the innovation tends to be diffused very rapidly. The degree of relative advantage may be measured in economic terms, but social factors like prestige, convenience, and satisfaction are often important components.

Compatibility measures how consistent an innovation is perceived to be with the existing values, past experiences, and the needs of the potential adopters. If a new product requires you to significantly change behavior, attitudes, or beliefs, it loses compatibility. Adopting an incompatible innovation may require the adoption of a new value system.

Complexity measures how difficult an innovation is perceived to be to learn and use. New ideas that are easy to understand will be adopted more rapidly than innovations that require new knowledge and skills.

The ability to try out an innovation measures how well you can learn, understand, and apply an innovation incrementally by working from small problems to large one. If potential adopters can try out an innovation, they will be less uncertain about its usefulness and about its potential for quick adoption.

Visibility measures how visible the results of an innovation are to others. The easier it is for people to see the results of an innovation, the more likely they are to adopt the innovation.

Thus an innovation that fulfills these five criteria — one that enjoys relative advantage, that is compatible with the current environment and past experiences, that is easily comprehensible, that may be tried out on small applications, and whose performance is easy to observe and evaluate — has greater potential for rapid and widespread adoption than an innovation that lacks one or more of these properties.

However, the characteristics of an innovation are not determined by innovation alone. They also depend on the nature of the potential adopters and their environment. Tornatzky and K. Klein have provided a more detailed analysis of the subjectivity of innovation characteristics.

Communication process. Diffusion requires a communication process that results in the transmission of information about innovations. Rogers recognized two kinds of transmitted information: hard and soft. Hard information describes the details of the innovation like the concepts it represents, how it works, and how it should be used. Soft information refers to other information about innovation like its costs, ratings of evaluation factors of interest, and analysis of its potential effects and implications. Soft information deals with those aspects that are necessary to make adoption decisions.

The communication process involves innovators, diffusers, potential adopters, and the communication channels that connect them.

There are two types of channels: mass media and interpersonal. Mass-media channels let one person or a few people reach an audience of many. They include media like radio, newspapers, and trade publications. Interpersonal channels are direct communications between two or more people through face-to-face meetings and interactive media like phone calls and electronic conferencing.

Mass-media channels are rapid and efficient in creating awareness of innovations, while interpersonal channels are more effective in persuading people to adopt new ideas. Successful diffusion requires a proper balance between the use of mass-media and interpersonal channels. The proper balance depends on the characteristics of the innovation and the social and cultural aspects of the potential adopters.

Another important factor that affects communication is the nature of fit (homophilous or heterophilous) between the participants. Homophily is how similar pairs of people who interact are in certain attributes like beliefs, education, and social status. Heterophily is the lack of such similarity. Communication is usually more effective when the participants are homophilous.

Adoption process. Diffusion usually proceeds through a series of phases. The first stage is creating an innovation. This is followed by the communication stage, where the innovation is communicated to potential adopters. This triggers the adoption process, which concludes with the adopters rejecting the innovation or accepting and committing to it.

As Figure 1 shows, you can divide the adoption process into five steps: knowledge, persuasion, decision, implementation, and confirmation. In the knowledge step, the potential adopter obtains information about what the innovation is and how it works. During the persuasion stage, the diffuser tries to influence the potential adopter to accept the innovation. In the decision phase, the adopter seeks various kinds of evaluation information to reduce his uncertainty about the consequences of adopting the innovation. The decision phase leads to an adoption or rejection decision. If the adopter decides to accept the innovation, he then implements and tests it in his environment. This leads to the confirmation phase, where the adopter either makes a commitment to the innovation or rejects it.
Studies have shown that the adoption of innovation follows an S curve as shown in Figure 2. This occurs because people differ in their propensity to adopt innovations.

Rogers has grouped adopters into five categories: innovators, early adopters, early majority, late majority, and laggards. These categories also reflect the relative order in which these people will adopt innovations. Therefore, diffusion is necessarily drawn out over time. Furthermore, members of these categories display recognizable social and personality profiles, so you usually can identify early adopters and the early majority in the target audience of adopters. During the early stages of diffusion, it is wise to focus only on these subsets of potential adopters. You can then exploit their adoption experiences to accelerate the rate of adoption by the rest of the target audience.

The number of early adopters is small, so the number of people in adoption start-up is small. In any diffusion process, the start-up is slow, but it then increases until it peaks, and then it diminishes as fewer nonadopters remain.

Marketing research studies have tried to identify the characteristics of people in the innovator and early adopter categories. The results vary, which suggests that the answer depends on the product group being studied, but the five traits have often emerged:

• Venturesomeness: the willingness and desire to be daring and try out new and different things.
• Social integration: how frequently and extensively you deal with other people at work, in your neighborhood, or in your social life.
• Cosmopolitanism: a point of view that extends beyond your immediate surroundings and shows an interest in world affairs, travel, and reading.
• Social mobility: your upward movement on the social scale.
• Privilege: generally defined as being better off financially than others in your group; this trait tends to reflect attitudes about the use of money as much as about the possession of money.

These five traits characterize the innovator, early adopter, and early user groups. But it is not possible to identify who in the

---

Figure 1. Typical innovation-adoption decision process.
innovator group will be the early users. Research has also shown that in the industrial setting early adopters tend to be the largest firms in the industry (although some studies have suggested the opposite), early adopters are those firms that stand to make the greatest profit from the innovation, early adopters spend more on R&D, and early adopters have presidents who are younger and better educated.

**Social System.** The social system is the environment in which diffusion processes take place. Rogers has identified five aspects of the social system that affect the diffusion and adoption process: social structure, social norms, roles of opinion leaders and change agents, how adoption decisions are made, and the consequences of adopting innovations.

The social structure is the arrangement of how members of a social system communicate. Two types of communication structures exist in any social system: formal and informal. A formal structure is designed to promote goal fulfillment, process control, and predictability. This structure gives regularity and stability to the system and tries to reduce the uncertainty of human behavior within the system. Informal structures include mechanisms like friendships, leisure activities, and gossip that are not designed consciously or explicitly. They usually come into existence because of the members' social needs.

The structure of a social system can facilitate or impede the diffusion of innovations in the system. Furthermore, a person's propensity to adopt innovations depends not only on his individual characteristics but also on the nature of his social system. For example, normally eager adopters may dampen their enthusiasm in conservative social systems and otherwise reticent adopters may display increased receptivity in liberal settings.

Social norms are established behavior patterns for the members of a social system. They define a range of tolerable behavior and serve as a guide or standard for the members. Rigid social norms can be a serious barrier to diffusion of innovations. Opinion leaders and change agents play critical roles during diffusion. Their attitudes toward an innovation largely determine the outcome of the diffusion process. If they favor the innovation, they may serve as effective catalysts for rapid diffusion and adoption. However, if they criticize the innovation, they can impede its diffusion.

The decision to adopt an innovation may take place at the individual, group, or authority levels. The social system itself may or may not influence the level at which the adoption decisions are made. Individual-level adoption decisions are called optional-adoption decisions. Here, a person adopts or rejects an innovation independently of the decisions of the social system's other members. However, the person's decision may be influenced by the social norms or by the members of his interpersonal network.

Collective-adoption decisions are choices that are made by consensus among the system's members. These are very efficient from the viewpoint of diffusion because the majority of the social system has agreed to adopt the innovation voluntarily.

Authority-adoption decisions are usually made by a few key people who possess power, status, or technical knowledge. Once they make the decision, it becomes binding on the rest of the system's members.

The level at which adoption decisions are made has many implications for the diffusion process. The collective-adoption processes may take longer than the individual-adoption process to make the initial adoption decision, but once the decision has been made the innovation diffuses rapidly because the decision is based
on a consensus. Authority-level decisions may or may not diffuse rapidly, depending on the attitude of the members toward authority. If the members submit to authority, the rate of adoption is accelerated, but if they resist authority, they may undermine the innovation’s adoption.

Because an innovation is necessarily a new idea, practice, or object, its adoption implies some changes to current methods and practices. Therefore, adopting an innovation amounts to accepting some changes. These changes can interact with the social system in complex ways and produce various kinds of consequences. You can classify these consequences under one or more of the following categories:

- desirable or undesirable,
- direct or indirect, and
- anticipated or unanticipated.

Social systems will generally accept and commit to innovations that produce direct, desired, and anticipated consequences. If an innovation has many undesired consequences, it will be rejected by the social system even if it is accepted initially. However, the consequences of adopting an innovation are not the same for the social system’s various constituencies. What may be a desired innovation to some members may not be desired by the other members, leading to conflicts. In such cases, the final outcome will depend on how the conflicts get resolved—a complex sociological process in itself.

Key issues. Rogers’s framework has nine key points:

- Diffusion is a process by which an innovation is communicated through certain channels, over time, and among the members of a social system.
- The perceived attributes of an innovation have strong implications for the success or failure of its diffusion.
- Diffusion is accompanied by change, so effectively managing change is critical for successful diffusion.
- Diffusion occurs in a social context, so factors like social structure, culture, and norms can facilitate or impede diffusion.
- Diffusion requires effective communication, so the selection of communication channels and the match between the participants are important factors in promoting diffusion.
- Innovation-adoptions decisions are influenced by both rational and irrational factors.
- People differ in their propensity to adopt innovations. Based on this propensity, you can group people into categories like early majority, late majority, and laggards. These categories also reflect the relative order in which these people will adopt innovations.
- Diffusion is necessarily drawn out over time.
- Innovation-adoption decision processes may be carried out individually, collectively, or by authorities. The level at which the adoption decisions are made have significant implications for diffusion.
- Change agents and opinion leaders act as catalysts during diffusion. Their attitudes toward the innovation can largely determine the success or failure of the diffusion.

Software-engineering diffusion

There are two ways that you can use Rogers’s framework in the context of software-engineering diffusion: as a descriptive model or as a prescriptive model.

Descriptive model. Because Rogers’s framework is a model that is well tested on a wide variety of innovations, you can use it as a theoretical foundation for conducting empirical studies to enhance the understanding of software-engineering diffusion. Some studies have already used aspects of Rogers’s framework to study the diffusion of software products in public agencies and of office-automation products in large corporations. Because many empirical studies have been done using Rogers’s framework in other fields, new studies in the software-engineering arena can draw on their methodologies and analysis techniques.

Conducting these empirical studies can provide answers to some important questions: Are software-engineering innovations diffusing as fast as they can? If not, what are the key problems that are slowing their diffusion? How do we evaluate the importance of these factors in the software-engineering context?

Prescriptive model. Rogers’s framework readily provides the following general guidelines for facilitating diffusion of innovations:

- Ensure that innovations rate well on five key characteristics: relative advantage, complexity, compatibility, ability to be tried out, and visibility.
- Generate adequate hard and soft information about the innovation.
- Develop strategies for overcoming the problems of incompatibility and smoothing the transition.
- Select an appropriate communications strategy.
- Develop diffusers.
- Pay attention to communication aspects.
- Recognize the differences in propensity for adopting innovations and develop a plan of attack that takes into account the S-curve phenomenon.
- Understand the social structure and norms of the potential adopters.
- Understand and develop strategies to deal with the potential interactions of the innovation with the social system.
- Develop change agents.
- Manage change.
- Work through opinion leaders.
- Seek collective acceptance of the adoption decision.
- Ensure commitment to the use of innovation.
- Develop methods to establish the consequences of adopting innovations.

By refining these guidelines in the context of software engineering, you can develop more concrete action agendas. Doing this requires understanding and
sensitivity to four issues:

- Translating the constructs provided by Rogers's framework into software-engineering terms.
- The skills and expertise of the software-engineering community.
- Knowledge and level of understanding about the diffusion of software-engineering innovations.
- Pragmatic issues like what can be controlled and what factors can provide leverage in diffusion.

As Rogers's framework points out, successful diffusion requires that you establish successful interaction between the four main elements of diffusion: innovation, communication, adoption, and the social system. The key question is what kinds of skills, knowledge, and expertise are needed to affect the various aspects of the complex interaction underlying diffusion.

Troublesome areas. Proper packaging of an innovation is critical for its adoption. ("Packaging" usually refers to things like training materials, computer-based support tools, and empirical evidence for the performance of an innovation in realistic applications.) Because of their critical importance, the creation of such support materials is an integral part of the innovation process. Therefore, it becomes an added responsibility for innovators. In the software-engineering context, this responsibility must be borne by software engineers.

Because software-engineering innovations — especially methodologies and techniques — are abstract, they are prone to misunderstanding and misapplication. Thus an innovation that is accepted initially may be rejected later because of its apparent failure to produce results. You can avoid these problems only by precisely articulating what these innovations represent, what their intended objectives are, and how to apply them effectively. You must also ensure that potential adopters have correct expectations about the innovations. These issues underscore the importance of communication processes in the software-engineering context and the need for active involvement of innovators and researchers in the communication.

The area that is potentially most troublesome for the software-engineering community is dealing with the social system. Social systems tend to be complex and difficult to control and change. The body of knowledge needed to manage these lies in management studies and must be adapted to the software-engineering context. This knowledge is essentially nonexistent in the software-engineering literature.

To add to the problem, software engineering professionals as a rule receive very little training in management, so the profession must rely on practicing software engineers who have accumulated managerial skills through years of practical experience.

Because the adoption process is greatly influenced by the social system, affecting it becomes difficult. However, you can partially compensate for this by providing adequate hard and soft information about innovation. Adequate hard information is usually available for software innovations, but not soft information like cost-benefit analysis, hidden-costs analysis, effect analysis, risk analysis, scaling-up problems, and available strategies. Getting this information should have a high priority in the software-engineering community.

It is quite possible that managing the social system will turn out to be the major determinant of successful diffusion. However, it can become significant only if the innovations are basically sound and if they are objectively and effectively communicated to the potential adopters.

What must be done. We have developed the following action items that the software-engineering community should undertake immediately to address the needs of software-technology diffusion:

- The diffusion literature, which exists in the management discipline, must be studied and adapted to software engineering.
- The software-engineering community should become more involved in objectively communicating its innovations to practitioners. Software engineers should play active roles as diffusers and opinion leaders and widen the scope of their research and publication activities.
- Software-engineering students can play an effective role as diffusers. They should learn how to manage change, become effective communicators, and learn to deal with incorrect perceptions. They should be taught measurement techniques with which they can evaluate the performance of tools they diffuse into their parent organizations. They should be provided comprehensive information about available empirical results.
- Software-engineering education should address techniques for justifying and diffusing software-engineering technology, not just teach tools, techniques, and methodologies.

Case study

Program-design methodologies have been an active area of research in software engineering for 15 years. Many methodologies have been developed to facilitate the development of quality programs. The same period has seen rapid growth in a satellite industry that offers professional-development seminars, in-house training programs, consulting services, and support-tool development for these methodologies.

Despite the elaborate efforts of the satellite industry and the strong commitment and enthusiasm of the software-engineering community, the adoption of the well-known methodologies — structured programming à la Edsger Dijkstra and Niklaus Wirth, Terry Baker and Harlan Mills's integrated top-down design method, stepwise refinement using abstract data types based on Barbara Liskov's ideas, Jeane Warnier's approach to logical program construction, Michael Jackson's
design method, and structured design methodology à la Larry Constantine and Ed Yourdon — is slow and far below the desired levels. This can be partially explained using Rogers’s framework.

(The question of what constitutes a satisfactory diffusion rate for software-engineering technology is a controversial issue. Sam Redwine and William Riddle have concluded that it takes 15 to 20 years for a technology to mature to where it can be popularized and disseminated to the technical community at large. They based their conclusion on an analysis of several software-technology case studies. But this cannot be used to justify the current software-engineering difficulties because it is a circular argument: It takes 15 to 20 years because it has taken 15 to 20 years. Another study by R.R. Willis concluded that technology transfer takes from four to eight years in most cases.)

We conducted an informal case study based on how program-design methodologies are perceived by the innovators, how they are communicated to the practitioners, and how they are perceived by the practitioners. Figure 3 lists these perceptions. The descriptions are based on our extensive personal involvement with program-design methodologies and on the case studies and user experiences described in the literature. Although the data may be informal, it does represent reality.

Practitioners' problems. The data shows that practitioners perceive several problems with the program-design methodologies. These problems can be systematically grouped by the five key innovation characteristics: relative advantage, complexity, compatibility, ability to be tried out, and visibility.

Practitioners do not seem to perceive

---

**Figure 3. (a) How program-design methodologies are perceived by the innovators and diffusers, (b) how they are communicated to the practitioners, and (c) how they are perceived by the practitioners.**
clear advantages in adopting the methodologies because the methodologies require a lot of peripheral work that may or may not pay off in the long run. This peripheral work creates large overheads and slows down programming activities. Furthermore, the overheads increase rapidly as problem size increases.

These methodologies are too general and do not provide clear guidance for their proper application, making their use complex. Because they are incomplete and need creative adaptation during their application, they can be easily misapplied. It is also very difficult to determine these methodologies’ applicability to a given problem.

Practitioners also perceive problems with compatibility. Because methodologies require a different and systematic way of thinking about design problems, they usually constrain programmers to apply some form of top-down design approach; they are generally incompatible with the informal design processes that are already in use. Furthermore, the methodologies do not provide much guidance about easing the transition problems or about how they may be combined with other methods during transition periods. Last, but not least, there is also a strong perception that these methodologies take away the creative challenge associated with program design.

These methodologies also seem to be difficult to try out because you must apply them in their entirety to derive any benefits. The learning you acquire by applying these methodologies on small problems does not prepare you to attack large problems. They do not scale up well.

Last, the methodologies are also perceived to have poor visibility because it is very difficult to observe or ensure that a chosen methodology is being followed properly. It is usually not clear whether the success of a design is due to the methodology or to the designer’s ingenuity. It is also difficult to establish the benefits accrued by using the methodologies, which is further aggravated by the general lack of empirical results that establish their value.

Communication problems. In addition to the problems with the design methodologies’ characteristics, there are several problems with the communication of these methodologies. The data in Figure 3b shows the wide gap between the perceptions of innovators and those of practitioners. It means that the methodologies are oversold or poorly communicated—or both.

It is also clear that empirical evidence for the strengths of the methodologies are seldom communicated. Information about the risks associated with the methodologies, as well as other evaluation details that are needed for innovation adoption decisions, are either not available or not communicated.

Recommendations. These problems suggest that program-design methodologies will be difficult to diffuse and will diffuse very slowly. This conclusion is consistent with the general belief that program-design methodologies have not diffused well in practice.

Based on our analysis of program-design methodologies, we suggest the following actions for the software-engineering community to take to facilitate these methodologies’ diffusion:

- Improve the methodologies so their advantages clearly outweigh their drawbacks. One approach is to develop computer-based support so developers are free of the mundane overhead activities that slow down the use of these methodologies. CASE tools are now addressing this weakness.
- Extend the methodologies with domain-specific knowledge. This is being explored in knowledge-based CASE tools.
- Develop methods for measuring the effectiveness of these methodologies. This is a central issue in software productivity.
- Develop techniques by which you can incrementally introduce these methodologies and by which you can seamlessly bridge the incompatibilities among the past, current, and future development practices. These ideas are part of CASE reverse engineering.
- Characterize the benefits, drawbacks, and risks of using program-design methodologies. For example, it is being recognized that CASE has both serendipitous advantages and unpredictable side effects and risks. Systematic studies are needed to develop a better understanding of the benefit/risk trade-offs.

Identifying and illustrating Rogers’s framework as an appropriate model of software-technology diffusion is just the first step in studying the diffusion of software-engineering technology. Further work is needed to put Rogers’s framework into practice for the diffusion of software-engineering innovations.

The framework must be empirically validated for software-engineering diffusion. It must be refined so it can predict software-engineering diffusion. Taxonomies must be developed to characterize the state of diffusion and the dominant problems in diffusion. Last, contingency frameworks are needed to prescribe which actions the diffusing agents should take.

Acknowledgments
We are indebted to Roy Schweiker, our graduate research student, who prepared an annotated bibliography on technology transfer, fellow faculty members at the Wang Institute for their insightful comments on an earlier draft of this article, and to the anonymous referees for their suggestions.

References
Software Engineers

Ford Aerospace, Western Development Laboratories Division, located in the beautiful San Francisco Bay Area, is seeking Software Engineers.

Join us in transferring our current Signet software program to a new hardware format. You will also work independently analyzing, synthesizing, and communicating software system design and process solutions. Responsibilities will include coordinating the technical efforts of subcontractors in a large multi-site software development, integration, and test effort.

Requires a BS in Engineering, Math, Physics, or Computer Science and a minimum of 6 years' experience. You should also have experience with distributed PDP, VAX-based computers, hardware and software relational DBMS, and SUN/UNIX™ and C language technology. Must also have some exposure to mini and micro computer development, as well as firmware development. Knowledge of Army/ Air Force tactical communication systems a plus. TCAC experience desired.

We offer an excellent benefits package including a relocation package and a stable work environment. For immediate consideration, please send your resume to: Betsy Smith, Ford Aerospace, Western Development Laboratories Division, Dept. BS-IEEESW, 220 Henry Ford II Drive, P.O. Box 49041, San Jose, CA 95161-9041. An equal opportunity employer. Principals only, please. Candidates must be able to work on all classes of government material.

UNIX is a registered trademark of AT&T.


Donald R. Chand is the chairman of the Computer Information Systems Dept. at Bentley College. Before joining Bentley in 1984, he taught for 15 years at Georgia State University and was chairman of the Information Systems Dept. His research and teaching interests include design methodologies, fourth-generation languages, and applications development technologies.

Chand received a PhD in mathematics from Boston University. He is an ACM lecturer, associate editor of the Journal of Information Resources Management Systems, and member of the Journal of Microcomputer Systems Management editorial board. He is a member of IEEE.

Sridhar A. Raghavan is an assistant professor at the Computer Information Systems Dept. at Bentley College. Before joining Bentley, he was an assistant professor of software engineering at the Wang Institute. His research interests include the use of artificial intelligence and expert-systems technology for software engineering and decision making.

Raghavan received a PhD in computer information systems from Georgia State University. He is on the editorial board of the Journal of Information Resources Management Systems. He is a member of ACM, IEEE, the American Association for Artificial Intelligence and the Institute of Management Science.

Address questions about this article to the authors at CIS Dept., Bentley College, Waltham, MA 02254.