

◆ Successful Introduction of Domain Engineering into Software Development

Mark A. Ardis and Janel A. Green

During the last five years, the Switching and Access Solutions Group of Lucent Technologies has made a considerable investment in domain engineering. Much of this work has been motivated by a desire to reduce the software development interval. The use of domain engineering has helped to decrease considerably the time required to design and implement software. In this paper, we examine four domain engineering projects. We find that each project was guided by two sets of criteria for success—one addressing the concerns of good engineering practice and the other addressing the special needs of technology transfer. The five critical attributes of new technology identified by E. M. Rogers's Diffusion of Innovations—relative advantage, compatibility, complexity, trialability, and observability—play an important role in domain engineering. In the projects we examine, attention to these attributes helped shape the final products and the processes by which they were created.

Introduction

We report some experiences in the use of domain engineering to construct switching systems at Lucent Technologies. The software development teams for the projects we investigated introduced new technologies to their organizations, so they were obliged to solve two sets of problems—those related to the construction of software and those related to ensuring eventual adoption of the systems they built. The interaction of technical decisions with adoption strategies is the main subject of our study.

Others have observed similar problems arise during the introduction of new technology into software development.^{1,2} While we believe our lessons apply to other types of software systems and to other types of technology, they are especially relevant to organizations considering the adoption of domain engineering.

In the next section, we describe the domain engineering method practiced by the project teams. In the third section, we review the technology adoption issues identified by E. M. Rogers.³ In the fourth section, we summarize experiences of project teams, par-

ticularly those that relate to technology adoption. We make some final observations in the last section.

Domain Engineering

The Family-oriented Abstraction, Specification, and Translation (FAST) process⁴ is a software development process that leverages the knowledge of domain experts. FAST recognizes that many software systems, such as versions of a product, share significant functionality and behavior. Rather than maintain duplicate copies of these commonalities, it is better to have only one instance. Therefore, FAST proposes that software families should be constructed by a two-stage process:

1. *Domain engineering.* Analyze the family of products to understand what is common and what is variable among family members, then build an application engineering environment that can generate software from a specification of a family member.
2. *Application engineering.* Use the environment to generate family members—that is, customized products.

Panel 1. Abbreviations, Acronyms, and Terms

AIM—Asset Implementation Manager
CAL—Call Accounting Language
DECC—Domain-Engineered Configuration Control
FAST—Family-oriented Abstraction, Specification, and Translation
GUI—graphical user interface
HSI Designer—Hardware-Software Interface Designer
SETT—Software Engineering Transition Team

Figure 1 shows this two-step process and the feedback between the two. The domain engineering part of FAST identifies, specifies, and maintains the conceptual integrity of the system. As new family members are requested, they are checked for compliance with the common characteristics of the family. As long as the original design accommodates the request, application engineering can be used to quickly generate the new system. If not, further domain engineering is performed to extend or modify the design.

Domain engineering is an activity performed by those who have rich domain knowledge and expertise. They are the guardians of a system's integrity. Application engineering is intended to be performed efficiently and may be carried out by application engineers who are unaware of the details of the underlying machinery. This separation of concerns helps to maintain the integrity of a design as a system evolves and increases the efficiency of those who produce individual products.

The FAST process evolved from work originally performed by researchers at the Naval Research Laboratory in the 1970s.^{5,6,7,8} D. M. Weiss⁴ developed and introduced the FAST method in collaboration with software developers at Bell Labs. More than 30 domain engineering projects have been initiated within Lucent, many of them in the switching systems area.

Technology Transfer

Rogers's work³ on the diffusion of innovations remains the oracular source of wisdom on technology transfer, its problems, and the factors that influence its success or failure. Rogers presents a model of innova-

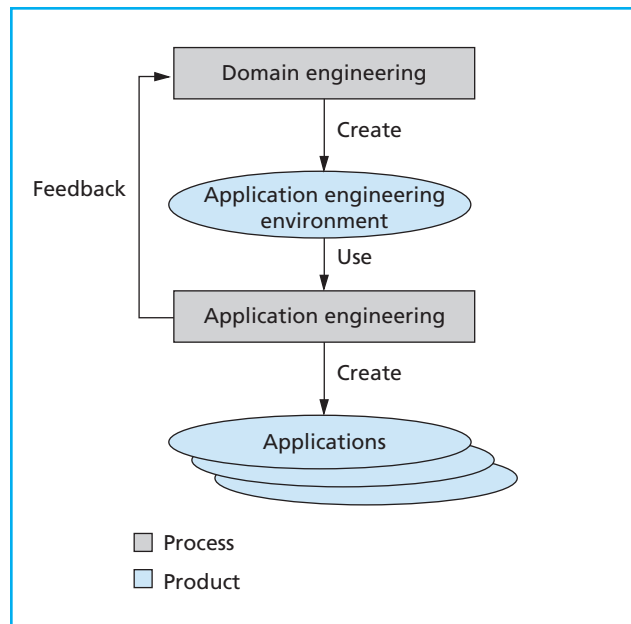


Figure 1.
The Family-oriented Abstraction, Specification, and Translation (FAST) process.

tion adoption that describes the interplay of several actors and processes. In particular, it distinguishes between different categories of people:

- *Innovators*, who create new technologies;
- *Early adopters*, who are the first to try innovations;
- *Early majority*, who establish an innovation's success by adopting it for regular use;
- *Late majority*, who adopt an innovation after its success has been demonstrated; and
- *Laggards*, who never adopt or who do so reluctantly after it becomes necessary.

In this study, we are interested in the interaction between the innovators (those who create new technologies via domain engineering) and the early adopters (those who first try these technologies). We hope to revisit these projects when they progress to acceptance by later adopters.

Attributes of Innovation

In Rogers's model,³ the rate of adoption of any new innovation is affected by the following attributes of the innovation:

- *Relative advantage*, the degree to which the innovation is perceived to improve upon existing solutions;

- *Compatibility*, the degree to which the innovation agrees with the values of the culture that will adopt it;
- *Complexity*, the difficulty associated with mastering the new innovation;
- *Trialability*, the ability to experiment with the innovation before adopting it in normal operations; and
- *Observability*, the ease with which improvement is noticed after adoption of the innovation.

These attributes have been observed to be significant predictors of innovation adoption across several industries.³ In particular, they have been found to be significant in software engineering.⁹

Domain engineering presents some interesting innovation adoption challenges. There are two types of adoption that must be addressed. The first type is the innovation adoption of domain engineering itself. Domain engineering scores well on the attributes described above, which explains its success within Lucent. The second type is the introduction of the environment that is to be used for application engineering. We are interested in both types of innovation adoption—adoption of domain engineering by software development groups and adoption of the resulting innovations by application engineers. Although most of this paper concentrates on the latter type of adoption, we start by discussing efforts made to assist the former.

Facilitating the Adoption of Domain Engineering

Weiss,⁴ a member of the Software Production Research Department, introduced the FAST method of domain engineering at Bell Labs. Early adopters from the 5ESS[®] switch organization collaborated with Weiss and his research colleagues to refine the FAST method. The role these pioneers played in furthering the adoption of domain engineering cannot be overestimated. Most of the success of their projects can be traced to their passion for improving the software development process and their vision for its new form.

Other members of the Software Production Research Department have also supported these domain engineering projects. Some have participated as moderators and reviewers of analyses while others have contributed technologies or helped team mem-

bers learn new tools and methods. One technology that has been particularly helpful is InfoWiz[™], developed and supported by L. H. Nakatani.¹⁰ InfoWiz is a system for making special-purpose languages quickly and easily. Models of artifacts in an application domain are specified in the specialized languages, and then the models are processed to automatically generate the artifacts. InfoWiz is easy to learn and use, scoring well on the complexity attribute and providing significant relative advantage over other language development tools.

Concurrently, the 5ESS organization created a team of technology change agents—the Software Engineering Transition Team (SETT)—to ensure the success of new innovations. This group investigates and supports several innovations in software development, but we limit our discussion to their efforts in support of domain engineering. SETT provides training, support, and a small amount of funding to encourage project participants to try domain engineering. Data about each project is kept and disseminated via a Web site. SETT provides a focal point for potential adopters of domain engineering and serves as a repository of knowledge and experience for management.

The following are some of the activities of SETT that optimize Rogers's attributes for domain engineering:

- *Relative advantage*. Data collection efforts help to identify the advantages of domain engineering. Presentations at internal conferences and at small group meetings include discussion of the advantages gained by successful projects.
- *Compatibility*. Because the members of SETT originated from within the software development community, they are more like potential adopters than research staff. That is, they know the language, organization, and other details of life within the software community. Sensitivity to potential conflicts helps keep domain engineering in synchrony with the existing culture.
- *Complexity*. Course-training and consulting is provided to help new adopters get started. Ongoing support is provided by frequent informal gatherings of all domain engineering

adopters. SETT also investigates and develops tools to ease the construction of application engineering environments.

- *Trialability*. The modest funding provided to start-ups reduces the cost of trial use. SETT often participates in presentations to management to ensure their support and commitment.
- *Observability*. A simple set of metrics is collected by all domain engineering teams. These data are then reported via the Web site to all interested parties.

Domain engineering is supported by other organizations within Lucent, but SETT plays a significant role in the adoption of domain engineering within the 5ESS organization. It serves as a model for adoption support groups in other business units.

Strategies to Encourage Adoption

When a project team decides to use domain engineering, its members face the challenges of developing and transferring an innovation within their own organization. That is, they must become innovators and change agents. Knowing this, they are often influenced to make decisions based on the likelihood of future adoption of the innovation. We decided to investigate this influence by examining the effect of each attribute on decisions.

In the discussion that follows, the term *adopters* refers to the intended users of the application engineering environments. The term *innovators* refers to the domain engineering project members, reflecting their role. Each of the five attributes that influence innovation adoption suggests strategies that might be used to favor adoption. The following are some strategies for optimizing attributes:

- Relative advantage
 - *Technology improvement*. Introduce a new technology that is more powerful than the existing technology. An example of this is the use of a design language that may be compiled into code rather than translated by hand. Another example is the use of a new process that eliminates some steps.
- Compatibility
 - *Relevance*. Make sure that the problem solved by the innovation is important to

adopters. That is, select a problem to solve that is already a high-level concern.

- *Realism*. Do not try to change too much at once or to please too many different types of users.
- *Customer focus*. Seek input from current and future adopters and design a solution that they want. For example, interview potential adopters or observe their use of a prototype.
- Complexity
 - *Developer-friendliness*. Reduce the learning curve for developers of the innovation. If innovators must learn a new language or tool in order to create their environment, they may produce a system that is flawed in performance or functionality. Using well-known, established techniques leaves more time to focus on the problem at hand.
 - *User-friendliness*. Reduce the learning curve for adopters. Make the innovation easy to learn and use. Note that this depends on the knowledge and experience of adopters.
 - *Reuse*. Reuse as much of the old process and technology as possible. For example, make no changes to the old process or simply eliminate some steps. Alternatively, continue to use old debugging tools rather than introduce new ones.
 - *Education*. Provide tutorials and demonstrations to potential users and managers. Publish useful information on Web pages and offer pointers to early adopters.
- Trialability
 - *Cost*. Reduce the cost of trial use. The cost of learning the new technology and process is one item to minimize.
 - *Likelihood*. Increase the likelihood that trial use will succeed. For example, consulting or help-desk support can help a new user avoid common problems.
- Observability
 - *Measurement*. Collect data about the old and new technologies for comparison. Effort data is particularly helpful but performance data is also important.

- *Testimony*. Provide forums for adopters to describe their experiences. These may be designed to increase awareness within the technical community or to report results to management.
- *Shadowing*. Provide a side-by-side comparison by running two projects with the same goals, but with one using the old technology and the other using the new. This may be difficult to arrange, but it can be very convincing.

We do not claim that these are the only possible strategies for optimizing adoption attributes. They are merely representative of the behavior of the project teams we observed. Furthermore, we do not claim that these strategies are independent or complete. Strategies often overlap in application or are complementary at least. Finally, we do not claim that these strategies are novel or unusual. In fact, it is their obvious fit to the problems that makes them useful.

Experience

As part of an effort to document the early lessons of domain engineering, we conducted informal interviews with several project teams. (**Panel 2** contains descriptions of the projects.) During these interviews, project members were encouraged to describe those experiences that were significant challenges or that suggested valuable lessons to pass on to future project teams. We also examined the reports they prepared as part of an internal symposium. Finally, we conducted a sharing session in which many issues common to all projects were discussed and compared.

In discussions with project participants, we identified several common issues. Not all the project teams experienced the same level of difficulty with these issues, but almost all had to face them. There were, of course, other issues that were unique to one project.

In the following sections, we highlight the issues faced in each area and the solutions chosen or recommended by the project teams. It is important to point out that many of the issues and solutions are related to human behavior rather than to computer science. The technical computer science issues are easily handled by software developers; the “people” issues are more

challenging. Software development is inherently a social activity and people are extremely complex. This constitutes a major risk. We are actively pursuing strategies to help domain engineers succeed, especially in the realm of human behavior.

After collecting information on project team experiences, we examined it for influence by adoption strategies. In this analysis, we tried to match decisions made by project teams with the strategies proposed above. Of course, many decisions had nothing to do with future adoption but were simply the application of good software engineering techniques. Some decisions were clearly influenced by future adoption, however, and these were matched with appropriate strategies. For each issue, we identified the strategies employed by the project teams.

Planning and Management Issues

Planning a domain engineering project includes setting goals, selecting participants, and selecting technologies—that is, methods and tools—to use. These are issues that arise early in a project, but many need to be revisited due to the iterative lifecycle used. Some of the management issues faced by the project teams include resources, funding, and reporting progress. These are described in more detail in **Table I**.

Construction Issues

Creating the application engineering environment involves analysis, design, and implementation as for any other software development project. The most interesting challenges posed by domain engineering projects are those in which the team tries something new or integrates its new environment into an existing system. In these activities, the team must contend with adoption issues as well as the technical issues involved in constructing the software. Construction issues are detailed in **Table II**.

Deployment Issues

After the innovation has been built, it must be deployed for use. This is always more complicated than it first appears. Project teams must have a strategy for introduction to ensure that this happens successfully. Issues related to deployment are described in **Table III**.

Panel 2. Projects Within the 5ESS Organization

The 5ESS® switch is a complex system of hardware and software that has evolved almost continuously since its introduction.¹¹ One driving force for change has been the improvement in hardware and software technologies. Another force has been the diversification of customer needs as the market for switching products has evolved. When the 5ESS switch was first introduced, it was sold only within AT&T. It is now sold to service providers around the globe.

Within the 5ESS organization, there are several groups that maintain different components of the software. Some have adopted domain engineering to improve their efficiency and to maintain the integrity of their component's design. In this paper, we examine four of these projects: Asset Implementation Manager (AIM), Call Accounting Language (CAL), Domain-Engineered Configuration Control (DECC), and Hardware-Software Interface (HSI) Designer.

AIM

International customers of the 5ESS switch frequently request customization of the forms that are used for entering and changing data. The group that provides these customizations saw an opportunity to improve their efficiency by creating AIM, a domain-specific environment that allows software developers to specify forms using a GUI. AIM then generates the code and supporting documentation for those forms. Because many customers' requests are similar, standard templates are available to use as building blocks. This simplifies the construction task for software developers and also simplifies the specification task for the customer.

CAL

For our service provider customers, billing is one of the most important features of the 5ESS switch. As might be expected, service providers frequently request customizations to the standard set of billing records that are generated for each call. CAL was created to allow specification and generation of customized billing records. A GUI is used to specify records, then translators

generate the code and documentation. The CAL project team also constructed a suite of testing tools to simplify the validation process.

DECC

Improvements in technology have enabled Lucent to introduce new hardware components in the 5ESS switch to reduce costs. New applications of the telephone network, such as Internet access, have also created a need for new interfaces. As a result, the individual hardware components of the 5ESS switch have evolved continuously. Configuration control is that part of the software in the switch that maintains the overall status of hardware units. Each request to change the status of a unit is first tested to determine whether the performance or integrity of the switch might be compromised. If the request is allowed, a sequence of steps is followed to ensure that all units function smoothly during and after the transition. DECC is a suite of languages and software tools designed to simplify the construction of configuration control software for new hardware units. It is based on a set of attributes common to all hardware units despite their unique functionalities.

HSI Designer

Another domain affected by changes in hardware is the interface between hardware and software. Hardware designers must specify the structure and operation of new hardware units so that software developers can write code for those units. For the 5ESS switch, this specification is provided in a document called HSI. In order to reduce the effort required to create these documents, a team of developers created a special tool—HSI Designer. This GUI tool simplifies the job of specifying details about hardware elements, such as registers and memory devices. HSI Designer audits the specification for hardware-software consistency and generates several artifacts for downstream development processes. For example, it creates documentation, software header files, intermediate data used to construct hardware logic, and executable code from abstract scenarios.

Table I. Planning and management issues.

Issue	Description	Strategies
<p>Goals: How will domain engineering solve a perceived problem?</p>	<p>Documenting a vision and a set of goals at the start of a project was found to be important. A vision is not a specific solution, but a collection of ideas about how development in a domain will be different—and improved—within the application engineering environment. Project teams found goal setting to be particularly important. A good set of goals keeps a project on target and prevents it from wandering too far from the main objectives. One way to ensure that the right goals are chosen is to do an economic analysis to find the best return on investment. Since the project teams were working under the constraint of a legacy system, existing organizational boundaries were a factor in establishing goals. These boundaries limited the scope of work that the teams could accomplish.</p>	<p>Technology Improvement: Establish relative advantage over the status quo. Relevance: Ensure that the problem solved by the innovation is a concern to users. User-Friendliness: Provide a solution that is user friendly. Measurement: Collect data on the old process to ensure that the right problem is being addressed. Reuse: Reuse existing technology when no change is needed. Realism: Solve the most important problems and avoid doing too much.</p>
<p>Participants: Who are the right people to work on the project?</p>	<p>Collecting and keeping the right participants was difficult for many of the project teams. An effective domain engineering team needs a variety of skills to succeed. First, it is critical to get domain experts to participate—some as active members of the team and others as consultants and reviewers. These experts are often the most influential in getting the new innovations adopted later on, so their commitment to the success of the domain engineering effort is essential. Second, since experts are often closely tied to the current design, the participation of people newer to the domain is also useful as a source of new ideas. Third, it is helpful to have people knowledgeable in the technologies that are being considered for implementation. Support by management is needed to ensure that the project staff will not be diverted to work on other tasks.</p>	<p>Testimony: Take advantage of a favorable impression presented by peer adopters to convince others to join a team or participate as consultants and reviewers. Education: Provide tutorials and demonstrations to attract the interest of those with needed skills.</p>
<p>Technologies: Which technologies (methods and tools) should be used?</p>	<p>As the team prepares its strategy for design, it often encounters an overwhelming number of technology and process choices. One reason is that those who practice domain engineering are interested in using new processes and technologies and many possibilities exist. Another reason is that FAST is a flexible process that does not dictate the technologies to be used for design and implementation. To minimize risk, prototyping is often chosen to explore and test the feasibility of these choices.</p>	<p>Technology Improvement: Introduce new technologies or improve old ones. Customer Focus: Spend time with potential adopters to ensure that changes meet user expectations. Developer-Friendliness: Avoid trying every new technology and limit the number of new things that team members must learn. Reuse: Reuse existing technology when no change is needed.</p>

(continued on next page)

Summary

We have examined four domain engineering projects from the switching systems area of Lucent. All of the project teams produced and introduced new technologies to their organizations. In anticipation of problems related to the adoption of these innovations, the teams practiced a collection of strategies. Some of these strategies were potentially contradictory, which increased the difficulty of their application.

In our study, we encountered two basic results that

deserve emphasis. First, issues related to the potential adoption of innovations were pervasive. Second, domain engineering has worked well at Lucent. That is, many groups have successfully employed domain engineering to improve their efficiency.

How strongly did adoption issues affect these projects? We were surprised at the scope and degree of influence. First, adoption issues affected decisions made throughout the software development lifecycle for all the projects we studied. Second, project teams

Table I—*continued*.

Issue	Description	Strategies
<p>Resources: How can the project team obtain and retain the necessary resources for its work?</p>	<p>All the projects we studied were partially funded by a technology transfer organization. To obtain funding, each team was required to prepare a business case. Participants found this task onerous but rewarding. A good business case can aid in project planning as it emphasizes a project's expected benefits. Another difficulty, noted earlier, was attracting and retaining the necessary staff. If an expert is in a different organization from the rest of the team, his/her manager may be reluctant to give up the expert's time. As a result, the expert may be diverted from domain engineering work to address critical issues in other projects. Earning and retaining management support is an important prerequisite to obtaining and maintaining resources.</p>	<p>Measurement: Collect data and build a business case. Testimony: Use demonstrations and presentations to broaden support, especially with upper levels of management. Realism: Minimize the resources needed by setting realistic objectives.</p>
<p>Increments: How should the work be divided?</p>	<p>Since all the domain engineering projects used an iterative lifecycle model, all needed to identify and report incremental progress. This was a challenge for many of the project teams, since most other software development projects use a "waterfall" lifecycle model.</p>	<p>Technology Improvement: Use an iterative development approach rather than a traditional "waterfall" process. Cost: Produce early system prototypes that are easy to use on trial cases. Reuse: Reuse existing technologies and processes to lower the effort required for each iteration.</p>
<p>Estimation: How can the effort for a domain engineering project be estimated?</p>	<p>Domain engineering was a new method for all the project teams we studied. This made it difficult for them to predict expected effort. Fortunately, the iterative strategy they used helped to minimize this risk. As each team gained more experience with domain engineering, its members were better able to estimate future tasks.</p>	<p>Measurement: Collect data to demonstrate that the project is on track.</p>

FAST—Family-oriented Abstraction, Specification, and Translation

often chose to increase technical risks in order to decrease adoption risks, rather than the reverse. For example, graphical user interfaces (GUIs) were chosen for all projects, even though none of the teams had much experience creating them when they started. Finally, adoption issues often interacted, making divide-and-conquer strategies insufficient. Project teams were obliged to employ multiple strategies and then monitor the interaction of these strategies as they evolved. For example, in order to increase the relative advantage of their innovations, project members were encouraged to change the current process for constructing software. In order to increase compatibility, however, they were required to minimize the amount of change in existing processes. Striking the right balance required constant attention and customer focus.

Why does domain engineering work at Lucent? We think there are several reasons, including:

- There are successful products that have evolved

over several years. Such legacy systems offer many opportunities for domain engineering.

- There are many domain experts. Many of the developers have several years of experience in their domains. This acquired wisdom is an essential ingredient of domain engineering.
- The software community embraces appropriate use of automation. Several years of experience with software development tools have helped to convince our developers that some domain-specific tasks may also be ripe for automation.

These three reasons provide the opportunity, the resources, and the motivation to adopt domain engineering.

Acknowledgments

Domain engineering is a community activity requiring the cooperation of many talented individuals. We are especially grateful to the many innovators and early adopters of domain engineering within

Table II. Construction issues.

Issue	Description	Strategies
Legacy: How can new software be integrated into a large legacy system?	In order to introduce a new component into a legacy system, one must identify a clear set of interfaces with components that are not going to be changed. This requires careful specification and negotiation with the owners of those components. In addition, changes made to existing processes must be examined for their impact on downstream users. For example, those who integrate components into larger systems must be informed of changes to structures and software development processes.	Customer Focus: Work with downstream users—their needs may be different from those of others. For example, users building executable load images have different needs from those creating new applications. Reuse: Minimize risk by reusing old processes and technology whenever possible. Shadowing: Demonstrate innovation applicability through a shadow project.
New Languages: How should the project team create new languages and tools?	Not all project teams that use FAST invent new languages; however, those we studied did. That is, they created domain-specific languages to specify unique characteristics of family members. The specifications were then translated into other forms that could be compiled into code. Although the languages and tools were simple, all the teams were relatively inexperienced in language design and translator writing when they began their projects. Each team invested in additional training to master these new skills and worked with consultants to improve skills as the work progressed.	Technology Improvement: Introduce new languages to capture domain knowledge and improve efficiency of domain experts. Customer Focus: Work with users to minimize the risk that the innovation will be too difficult to learn and use. Developer-Friendliness: Use tools that are simple and familiar. Invest in training and consulting for unfamiliar tools.
User Interfaces: How should a user interface be designed?	All the project teams we studied created GUIs. The user interface is an important aspect of an innovation, since a poor interface can easily lead to rejection by potential users. Project teams reported that a clear set of goals was needed before attempting interface design and that customer focus was essential. In some cases, it was necessary to delay interface design until base functionality was better understood. This is an area where consultation with human factors experts can be especially helpful. Incremental development was useful, particularly if it included feedback from potential users.	Technology Improvement: Create a GUI to make users more efficient. Customer Focus: Work with potential users to ensure that the design is correct. User-Friendliness: GUIs are often simpler to learn and use than non-graphical user interfaces. Cost: GUIs are easier to use experimentally than non-graphical user interfaces.
Reviews: How can the project team get feedback from reviewers?	Obtaining adequate reviewer feedback for analysis, design, and code can be a problem for domain engineering project teams. First, experts are in high demand, so it is difficult to get much of their time. Second, reviewers may have difficulty providing useful feedback if they are unfamiliar with the innovation or with domain engineering. Some education or training may be needed to prepare reviewers. Furthermore, not all reviewers may be receptive to domain engineering. Some experts may resent the need to learn yet another method. Others may feel that domain engineering weakens their positions as knowledge resources in their domains. Still others may be skeptical of anything new.	Customer Focus: Talk to users about their problems before asking them to review proposed solutions. Reuse: Reuse old technology and processes to facilitate review. User-Friendliness: Emphasize the similarity with traditional analysis and design methods in order to persuade potential reviewers to contribute. Education: Provide tutorials to explain the process of domain engineering and the importance of review.

FAST—Family-oriented Abstraction, Specification, and Translation

GUI—Graphical user interface

Lucent. In particular, we thank Dick Braatz, Mark Bradac, Dave Cuka, Diane Nordin, Bob Olsen, Paul Pontrelli, and Doug Stoneman of the Switching and Access Solutions Group for sharing their project experiences. We thank Carl Amport of the same group and Lynn Pautler, formerly of that group, for their support of the SETT organization. We thank our colleagues in research for their help in creating new technologies,

especially Lloyd Nakatani of the Software Production Research Department for creating InfoWiz. We thank David Weiss of the same department for his pioneering work in developing FAST and for his support of its continuing diffusion throughout Lucent. Finally, we thank the reviewers for their help in improving the presentation of this work.

Table III. Deployment issues.

Issue	Description	Strategies
<p>Environment Introduction: What is the right approach to introducing a new system?</p>	<p>It is tempting to flash-cut deployment, but this is rarely successful. Incremental approaches are often better even though they may entail more work. Maintaining two systems during the transition incurs extra cost but may be the best strategy. That is, allow users to choose either old or new methods for the first few applications. This means that the new system must interoperate with the old.</p>	<p>Reuse: Reuse as much old technology as possible to reduce the learning curve for users—perhaps even allow alternative methods of use. Cost: Reduce the cost of experimental use to encourage new users to try the innovation. Likelihood: Increase the likelihood of success with first use of the innovation—for example, provide extensive consulting during trial use. Shadowing: Demonstrate innovation benefits through a shadow project.</p>
<p>Acceptance: How can the project team avoid rejection of its work by users?</p>	<p>When the system is available, it may encounter resistance from some users. It is important to get acceptance from the most influential users, so project teams should pay attention to their concerns. These users sometimes just need to be reassured that their expertise will continue to be valued. It is worth making the effort to help early adopters succeed. Their success can pave the way for those who follow. It is important to pay attention to user problems early and often. Each user has a different background and skill set and each may yield new insight into any remaining flaws in the innovation.</p>	<p>Relevance: Review problems and potential solutions with users to make sure that the right problem is being addressed. Customer Focus: Work closely with early adopters to make sure that the solution is compatible with the user community. Education: Offer tutorials and demonstrations to show simplicity of use.</p>
<p>Application: How can the project move from experiment to production status?</p>	<p>Even if you build it, they may not come. That is, the existence of an innovation is no guarantee that it will be adopted. Persuading application project teams to use a new innovation requires marketing. In addition to convincing technical staff, management must also be convinced of the benefits of adoption. One way to do this is to give talks and demonstrations. Another is to collect data to show expected improvements and reduced effort. Innovators need to be aware that even if an application team is interested in using the new environment, its members often want some other project team to do so first.</p>	<p>Relevance: Make sure that the right problem is being solved before moving to production use. Customer Focus: Work with users to ensure that they succeed. Measurement: Collect data to demonstrate innovation benefits. Testimony: When data is unavailable, seek testimonial support from early adopters. Shadowing: Demonstrate that the innovation performs as promised through a shadow project. Education: Provide tutorials and demonstrations to establish credibility with potential users.</p>

References

1. D. Fafchamps, "Organizational Factors and Reuse," *IEEE Software*, Vol. 11, No. 5, Sept. 1994, pp. 31–41.
2. R. G. Fichman and C. F. Kemerer, "Object Technology and Reuse: Lessons from Early Adopters," *IEEE Computer*, Vol. 30, No. 10, Oct. 1997, pp. 47–59.
3. E. M. Rogers, *Diffusion of Innovations*, 4th ed., Free Press, New York, 1995.
4. D. A. Cuka and D. M. Weiss, "Engineering Domains: Executable Commands as an Example," *Proc. 5th Intl. Conf. on Software Reuse*, Victoria, Canada, June 2–5, 1998, pp. 26–34.
5. D. L. Parnas, "On the Design and Development of Program Families," *IEEE Trans. on Software Eng.*, Vol. SE-2, No. 1, Mar. 1976, pp. 1–9.
6. D. L. Parnas, "Designing Software for Ease of Extension and Contraction," *Proc. 3rd Intl. Conf. Software Eng.*, Atlanta, Ga., May 1978, pp. 264–277.
7. D. L. Parnas, P. C. Clements, and D. M. Weiss, "The Modular Structure of Complex Systems," *IEEE Trans. Software Eng.*, Vol. SE-11, No. 3, Mar. 1985, pp. 259–266.
8. D. L. Parnas and P. C. Clements, "A Rational Design Process: How and Why to Fake It," *IEEE Trans. Software Eng.*, Vol. SE-12, No. 2, Feb. 1986, pp. 251–257.
9. S. A. Raghavan and D. R. Chand, "Diffusing Software Engineering Methods," *IEEE Software*, Vol. 6, No. 4, July 1989, pp. 81–90.

10. L. H. Nakatani and M. A. Jones, "Jargons and Infocentrism," *Proc. DSL '97, Univ. of Ill. Comp. Sci. Report, ACM SIGPLAN Workshop on Domain-Specific Languages*, Paris, Jan. 18, 1997, pp. 59–74, <http://www-sal.cs.uiuc.edu/~kamin/dsl>
11. K. E. Martersteck and A. E. Spencer, "Introduction to the 5ESS® Switching System," *AT&T Tech. J.*, Vol. 64, No. 6, July–Aug. 1985, pp. 1305–1314.

(Manuscript approved August 1998)

MARK A. ARDIS is a member of technical staff in the Software Production Research Department at Bell Labs in Naperville, Illinois. He holds a B.A. in mathematics from Cornell University in Ithaca, New York, and an M.S. and Ph.D. in computer science from the University of Maryland in College Park. Dr. Ardis conducts research related to domain engineering and the application of formal methods.



JANEL A. GREEN is a member of technical staff in the 5ESS® Call Processing and Japan Wireless Application Offers and Development Department at Lucent Technologies in Naperville, Illinois. As a member of the Software Engineering Transition Team, she is responsible for supporting organizations engaged in the adoption of domain engineering. She holds a B.S. in computer science from Benedictine University in Lisle, Illinois, and an M.S. in computer science from Northwestern University in Evanston, Illinois. ♦

