



AT&T Operations & Service Management

Software Development Technology Evaluation: Proving Fitness-for-Use with Architectural Styles

*21st NASA SEL Software Engineering Workshop
December 4-5, 1996
Greenbelt, MD*

James Cusick
AT&T
Bridgewater, NJ
James.Cusick@att.com

William M. Tepfenhart
AT&T
Middletown, NJ
William.Tepfenhart@att.com

Software Development Technology Evaluation: Proving Fitness-for-Use with Architectural Styles

1. OVERVIEW

A cursory glance at a few trade journals will indicate that hundreds if not thousands of development tools are available on the market. Today, with the boom in Internet technologies, dozens of new tools enter the market place each month. Faced with this situation we were asked to define how to choose the best tools for use in the development of hundreds of AT&T's business applications. Starting in early 1995 we began a revitalization of the software tool assessment practices of AT&T and especially AT&T's Network Services Division (NSD). These efforts are discussed in this paper.

An evaluation methodology was developed based on the concept of fitness-for-use as measured by the construction of architecturally representative applications within a laboratory environment. This method was used to evaluate dozens of commercial software development tools in order to select specific tools as corporate-wide standards.

This work presents the specifics of our software technology evaluation methodology, including our research efforts, tool taxonomy, and evaluation procedures (especially our use of software architecture-style-derived certifying test suites). This paper does not present the specific tools selected through the application of this methodology.

2. SOFTWARE TECHNOLOGY EVALUATION

Many evaluation techniques are known and meet with varying levels of success. Weighted averaging, benchmarking, figures of merit, etc., each have certain advantages and disadvantages (Kontio, 1995). Our approach is instead centered on the concept of demonstrated fitness for use in the environment of choice as measured by the applicability of any given tool to the dominant software architectures found within the target business environment. This approach reflects the "habitat models" suggested by Brown (1996).

This approach stems from viewing evaluation of software from the question: How well does the provided functionality of a product span the needs associated with tasks to be performed using it? Evaluation is highly dependent on the use for which the product is intended and the results are subject to greater ambiguity than evaluations of other classes of products. Many manufacturers of software products will be more than happy to provide metrics for common performance criteria. Other questions are more subtle - does the tool provide the right abstractions, is it easy to use, does it take one hour to do something or ten days. It is these subtle metrics that we intended our evaluation environment to measure and for this we turned to Architecture Styles.

3. SOFTWARE ARCHITECTURE STYLES

A year long study of our software systems identified (at least) four basic architectural styles present in our business applications (Belanger, et. al., 1996). These styles are: transaction, data streaming, real time, and decision support. These styles consistently appeared, in part and in full, in a wide variety of systems including those for Financial, Maintenance, Provisioning, and Asset Management domains. We say, in part, because a majority of our systems are actually hybrids of these different architectural styles.

We eventually derived several certification applications from these styles in order to drive our evaluation process. Our core reasoning being that the development of small scale applications modeled after our target development tasks would prove the suitability of the product under evaluation. This turned out to be true for virtually all the products we evaluated. The entire process of which the architecture styles play a key role is now presented in detail.

4. THE EVALUATION PROCESS

Our approach to evaluating software technology is to appraise technology as “fit-for-use” if we can succeed in developing a sample application which has a reasonable similarity to our production applications. In other words, we use the product under evaluation in an environment modeled after the target development environment. The process can be summarized in the following manner:

1. Survey the available products
2. Classify according to a technical framework
3. Filter the list using screening criteria
4. Construct evaluation criteria templates
5. Use the target tools to build an **Architecturally Representative** Application
6. Record findings against the templates
7. Judge the best scores and select the recommended product

4.1 Survey the available products

The overall evaluation process begins with surveying the tool market for candidate products and classifying them according to a technical framework sometimes called a taxonomy. Consider the survey effort first.

Initial research into software tool availability, capabilities, and trends, can be both rewarding and daunting. The goal of tool research is to identify all or most of the tools currently available for the support of a particular stage of the software development process. This research is technical in that one must understand the technological capabilities of each tool. At the same time, this research is market oriented in that one must also understand trends and supplier positioning. Some of the techniques used in this activity include:

- **Literature Reviews:** Books, journals, trade press publications. Key information on technical capabilities, product announcements, corporate changes, tool assessments and recommendations are readily available.
- **Trade Shows and Technical Conferences:** We have found trade shows to be *decreasingly* helpful in identifying technologies of interest. This is due to the generally poor level of technical information available at such venues. Technical conferences on the other hand *remain* helpful in putting the available products into a theoretical or practical context.
- **Direct Mail:** Believe it or not this is an effective means for collecting information once you are on enough mailing lists. (This may not be ecological but it is economical in terms of time; it only takes a few seconds to sort incoming product information.)
- **Automated Topic Searches:** We receive weekly or monthly summaries extracted from current publications on software technologies and trends via email.
- **Web Browsing:** This has become a significant source of information and freeware tools. We maintain a list of vendor web sites and this has often provided up to the minute information on particular products.
- **Vendor Demonstrations:** Slicing through the sales pitch to the technical meat is often difficult but this remains an effective means of collecting detailed product knowledge for selected tools.

- **Evaluation Copies:** A time or event determined interval of hands-on experience, execution, and utilization of the tools is invaluable in understanding actual tool capabilities (this is discussed in detail below).
- **Professional Information Services:** Several organizations are under contract to us providing strategic information on the software industry. This information is often helpful but can also be factually incorrect or misleading. These sources are useful more as sounding boards than anything else.
- **Private Contact Network:** Having a wide network of software professionals to draw upon for knowledge of the industry and technology cannot be overlooked in research efforts. For example, teaching a continuing education course at a local university has brought several new tools to our attention through conversations with students.
- **Experience:** Having been around the development community for a number of years directly impacts your ability to scan and decipher information on tools. Oftentimes “new” tools end up being familiar tools refaced.
- **Project Reference:** Having access to the real life trials of hundreds of development projects we know early on what is needed, what works, and what provides less than advertised.

The output of this research includes summary information on current product availability, industry trends, software standards and standards activities, computing techniques and methods, and development resources both internal and external. The specific products or technologies identified during our research efforts are given an initial classification in the tool and technology taxonomy discussed next.

4.2 Classify according to a technical framework

A Software Development Environment (SDE) can be viewed as an integrated set of tools and processes enabling analysts, designers, programmers, and testers to collaborate on the production of high quality software solutions. Traditional Software Engineering Environment (SEE) frameworks support the concept of creating an SDE by creating a view of the computing infrastructure as a unified and sensible environment with specified functional interrelationships instead of just a random assortment of tools (Brown, 1992).

Unfortunately, SEEs are not well suited to the task of tool classification since they are operational in nature. We required a classification scheme to build our SDE recommendations that could be used to organize toolsets of an eclectic nature resulting from our market research. Existing tool taxonomies (Kara, 1995; Fugetta, 1993; Sharon, 1993) typically focused on particular application domains, limited platforms, or were designed to cover only CASE tools. Since these taxonomies did not meet the needs of our scope (multi-platform, process driven tool standards), we derived our own classification for software tools.

To begin with our classification scheme inherited some structure from our corporate context. Domains typical of most software engineering environments sometimes fall outside of our mission charter. For example, operating systems, databases, and communications protocols are defined by other AT&T teams. Our mission was limited to a constrained view of Application Development technologies.

We decided to base our tool classification on an existing software engineering framework (Utz, 1992) and then modify it as needed (see **Figure 1**). The major categories provided by Utz are re-defined by us below. Each of these major categories are further detailed into sub-categories. Representative sub-categories are shown in **Table 1**. As our market research efforts turn up tools, we categorize them in the taxonomy. Currently we have approximately 1,000 tools in a database organized by these categories. This database allows us to perform ad hoc queries on tool use within AT&T and to quickly produce candidate lists when evaluation efforts are begun.

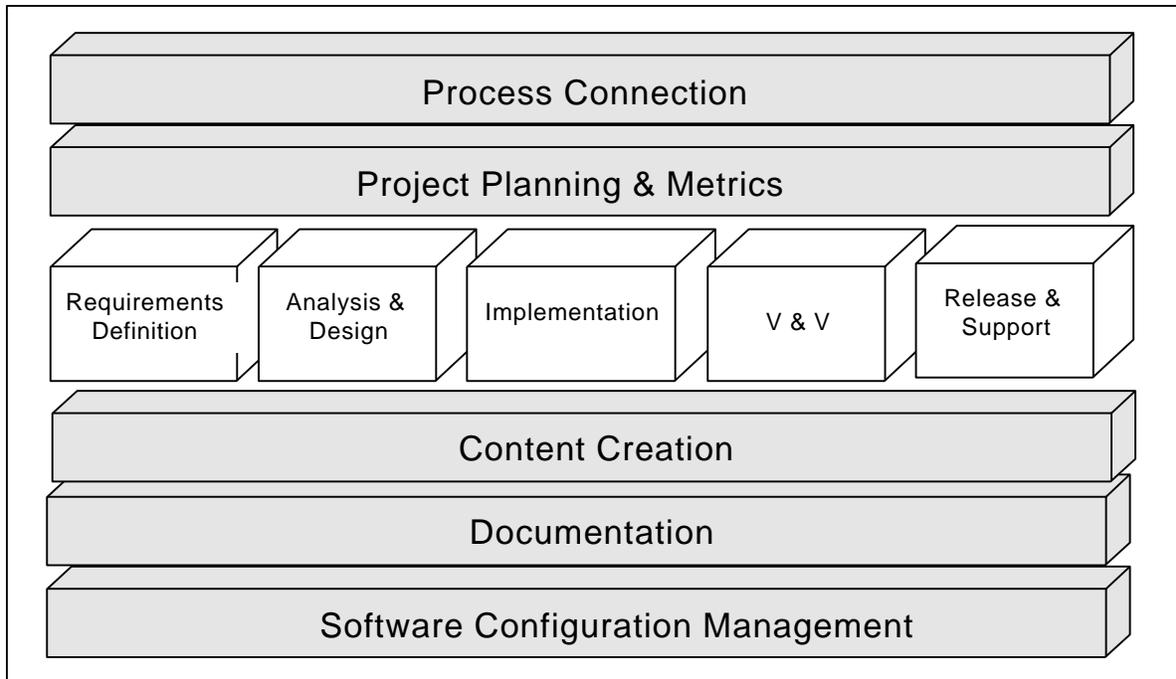


Figure 1: Software Engineering Environment Framework as Tool Taxonomy

4.2.1 The framework categories defined

- **Process Management**: Tools supporting the specification, implementation, and compliance management of development processes.
- **Management & Metrics**: Tools supporting the planning, tracking, and measuring of software development projects.
- **Requirements Definition**: Tools supporting the specification and enumeration of requirements.
- **Analysis & Design**: Tools supporting high level design and modeling of software system solutions following specific formal methodologies and often including code generation and reverse engineering capabilities.
- **Implementation (Code/Debug)**: These tools allow both low level code implementation to support the edit-compile-debug cycle of development in 3GLs and visual based programming targeted at rapid application development by use of screen painters/generators with graphical pallets of reusable GUI components with 4GLs.
- **V&V**: Tools providing software verification and validation, quality assurance, and quantification of reliability. These include test case management, test selection, and automated test support.
- **Release & Support**: Tools targeted at supporting enhancements and corrections to existing code as well as browsers, source code analyzers and software distribution.
- **Content Creation**: Tools used for developing Internet materials such as electronically published documents, graphics and multimedia components of Internet sites.
- **Documentation**: Tools supporting creation and distribution of system documentation, specifications, and user information. These tools include documentation storage, retrieval, and distribution.
- **Software Configuration & Manufacturing**: A broad class of tools related to the control of software components and development artifacts including documentation for the purpose of team based programming, versioning, defect tracking, and software manufacturing and distribution.

Process Process Definition & Compliance	Verification & Validation Test Management & Design Record & Playback Stress, Load & Performance Coverage
Project Planning & Metrics Project Planning Function Points General Metrics	Release & Support Distribution Reverse Engineering Emulation Utilities
Requirements & Definition Requirements Trace	Content Creation Web Document Authoring Graphics Authoring Multimedia Authoring
Analysis & Design Object Oriented Analysis & Design Structured or Other Design Methods RDBMS Modeling	Documentation & Workflow System Documentation Help Authoring Workflow
Implementation Languages Editors Compilers & Debuggers IDEs GUI/Visual Development Cross Platform Development Database Development Components	Software Configuration Management Source Code Control Defect Tracking Configuration or Manufacturing Integrated SCM

Table 1: Selected Tool Taxonomy Sub-Categories

4.3 Filter the list using screening criteria

With a thousand tools in the taxonomy we have to start trimming the list whenever a particular technology sub-category must be evaluated. Using basic technical requirements many candidate tools can be eliminated. Platform support, negative reviews in the trade press, vendor instability or financial losses by a vendor can all be used to quickly eliminate certain products from the evaluation list. If negative criteria do not work we use positive criteria: is the tool “Editor’s Choice” or does our development community already use it as a de facto standard? These types of tools need to be on the evaluation list while others should be dropped.

4.4 Construct evaluation criteria templates

Each of the tool categories in the taxonomy needs specific evaluation criteria to measure the relevant attributes of each tool type in our taxonomy. Towards that end a set of templates must be developed for each type of technology evaluated. These templates resemble the ones found in many trade journals and bench-marking reports. The following must be created or reused:

1. First, one overall template for generic tool and vendor measurement is provided. This generic template covers such items as documentation, support, pricing, and platform availability. A standard set of issues regarding tools such as iconic design, menu features, ergonomics, printing, and so on, is included.
2. Each analyst must then define a specific template which covers the technical aspects of the particular class of tool under investigation, if it does not already exist in our repository of templates. This must be created for each category.

4.5 Use target tools to build Architecturally Representative Applications

Recall that we are interested in demonstrating “fitness-for-use”. To do this we now build a representative application with the product(s) selected for evaluation from the taxonomy. Before evaluating any software technology we must first consider what capabilities it has and how to construct a suitable test suite or if our current set of application specifications will need expansion.

4.5.1 Technologies and Their Tasks

Each type of software product dictates certain kinds of tasks that will be the subject of evaluation. For example, word processors might be evaluated in terms of developing on-line (in program) documentation, help files, man pages, hard copy user manuals, and HTML documents. On the other hand, one would not evaluate a compiler in terms of its support of those same tasks. In some cases, products span more than one functional category. For example a C++ IDE might provide a visual programming environment, a class system, and a general purpose compiler. Since each of these is a separate endeavor, an evaluation of a C++ IDE will concentrate, independently, on the visual programming environment, class system completeness, and compiler performance. These are individual and discrete evaluations. Each will need specific resources to carry out the evaluation.

4.5.2 Software Resources for Evaluation

The software resources required to complete the data collection demanded by the evaluation template fall into three categories: 1) the software under evaluation; 2) supporting software (i.e., the operating system); and 3) software in the form of test cases (e.g., a sample design to implement). As we have shown, common architectures run through most AT&T applications. Our concept was to derive the required test cases from these architecture types or patterns.

Software patterns (Gamma, 1995; Coplien, 1995) formalize some of the concepts on recurring underlying software construction themes. We devised evaluation test cases to demonstrate that any tool recommended supported AT&T’s specific computing problem domains. Thus we developed and specified a set of representative applications modeled after architectural styles or patterns observed in the field, to serve as certifying test suites for any tool slated for review (see **Table 2**).

Arch/UI Style vs. Sample Apps	ARCH STYLE				GUI STYLE				
	OLTP	Data Stream	Decision Support	WWW	Forms	Active Graphic	Alert Panel	Map Based	Hypertext Browser
Contact	X		X		X				
COD		X					X		
GEM		X				X			
NetAnalyst			X			X		X	
ToolBase	X		X	X	X				X

Table 2: Representative Applications and their Architecture Styles

The representative applications and their relationship to the generic architecture styles of **Table 2** are briefly described below:

- **Contact Data Base:** The Contact Data Base is a very simple system for managing contacts on a project-by-project basis. Contacts are managed at the level of tracking individuals associated with a project, individual meetings, and tracking tools employed on the project. This application demonstrates a forms based interface for data entry and reporting.

- **Co-Operative Document System (CODS):** The Co-Operative Document System allows multiple people to work on the same document. The basic capability of checking a document into and out of a document control system is augmented with a message broadcasting feature alerting users of a subscribed document's state. This represents a client server system with data streaming and on-line transaction architectural components.
- **Graphic Enterprise Modeler (GEM):** The graphic organization display provides the ability to model graphically the structure of a corporate organization. It visually illustrates relationships among people, projects, and teams. To find answers to specific questions regarding an organization, the user follows semantically meaningful links and uses active graphics controls. This application demonstrates the user interaction style of the active graphics variety.
- **NetAnalyst:** This application is a map based data visualization tool. It takes a set of real telecommunications data (the 1994 L.A. earthquake phone traffic) and plots it geographically. This is a common type of application profiling decision support and mapping.
- **ToolBase:** This is an Intranet based front end to a product tracking database. This application provides for the evaluation of many types of Internet technologies and the extent to which they can support the architectural styles of OLTP and decision support on the Intranet.

Returning to our evaluation process, an appropriate application is selected to test the tool class and development against a set of specifications describing the sample application is begun. Often, the specifications need modification or additional software design efforts need to be conducted to fully stress the products under evaluation (e.g., our Internet application did not test multimedia features as initially designed). Inferior products fail during implementation of the specifications and quickly drop out.

4.6 Record findings against the templates

Throughout the work of building the sample application, feature performance data must be captured on the custom template constructed for this technical category. This includes objective and subjective measures. Subjective data includes how intuitive the product was or how friendly the help desk was when called. Objective data includes if the promised features worked and if you could accomplish the task of building the sample application.

Weighted Scoring Method (WSM) is normally used to provide a simple rating mechanism for each product under evaluation. In this method each item in the criteria matrix is assigned a score or weight score. Usually a score of 1 to 5 is given to the product for each criterion. Then an overall score can be derived using the formula below (Konito, 1996):

$$\text{Score}_a = \sum_{j=1}^n (\text{weight}_j * \text{score}_{aj})$$

4.7 Judge the best scores and select the recommended product

The final step is recommending a product. Out of the short list all products are evaluated. Using the sample application as a test suite the superior product normally emerges. With a WSM technique there is very small opportunity for any ties. The analyst must, however, still exercise their best judgment in selecting a product for recommendation.

5. EVALUATION PROCESS RESULTS

Within a laboratory environment we developed these representative applications repeatedly using different software technologies. We also carried out other tasks in support of this simulated development work, such as configuration management, using still more products under

evaluation. This approach provided clear evidence of the suitability of one product over another and was much easier to derive than by only looking at a feature capability matrix. We had a high degree of confidence that the product would work on a real development project using this method.

Dozens of tools have been evaluated using this method and still others are currently under examination. From this work many standard products have been chosen that are now part of AT&T's overall body of internal technical standards. Through controlled introduction using pilot projects and consultative jump-starts many of these products have also proven to be successful on large-scale software projects. Recently this technique was also used successfully to evaluate over 30 software products used in Internet based development projects.

6. PORTING THE PROCESS

Deployment of this technique to a different environment requires minimal modifications. We have reused this process from the evaluation of Windows based tools to the evaluation of Internet based tools seamlessly. To transfer this process to a different development base or user community we recommend making the following changes:

1. The tool taxonomy must be recalibrated to fit your environment and goals. Our taxonomy does not address databases, office automation, or operating systems. You need to add the appropriate technologies to fit your computing framework.
2. Your architectural styles may vary from ours. We develop very few "hard" realtime systems or embedded systems of any kind since our spin-off of Lucent Technologies. There may be other significant architectural styles you will need to identify.
3. After adjusting the framework and architectural styles you now need to document your screening criteria and create your detailed evaluation criteria templates. A good template typically requires a couple of days for an analyst to create. They are reusable and typically only one is necessary per technical category.
4. *Execute*. This is the crucial step where the watch-word is "emulation". That is, emulation of your actual development process and tasks.

We are confident that by following these simple steps the process we have been using for the last two years can be re-deployed in any software development technology evaluation laboratory.

7. CONCLUSIONS

Using applications derived from clearly relevant architectures keeps the evaluation process honest. Analysts with development backgrounds typically feel more comfortable building an application than acting as a software critic. Simulating the development tasks in this way does not solve all the problems with technology evaluation. Politics and compromise are inescapable factors when making decisions that will commit a corporation to spending or not spending large sums with any given vendor. Also, some variability remains in the scoring technique. Each analyst tends to have peculiar habits in working through a 200 item feature matrix. One may score "high" or "low" while another may include "medium". Nevertheless, we feel confident that architecture styles add a healthy modicum of extra validity to the otherwise typical process we have described.

8. ACKNOWLEDGEMENTS

Naturally work of this type cannot be accomplished without the cooperation and support of dozens of people. We are indebted to several senior managers for their enthusiastic top-down support of this work including Judy Page and Dick Machol of NSD, and Rod Mack and Illene Hochman of BMD. Our immediate support managers, Barbara Beech and Moses Ling, were especially helpful in guiding this effort. Finally, we would like to thank the dozens of people who provided information, feedback, and carried out the evaluations across AT&T and NSD. In particular we appreciate the work of the NSD Software Development Environment Team and the AT&T Foundation Architecture Software Development Tools Team. This paper reports on the combined work of each of the individual team members.

9. REFERENCES

1. Belanger, D., et. al., "Architecture Styles and Services: An Experiment on SOP-P", AT&T Technical Journal, Jan/Feb, 1996, pp54-63.
2. Brown, et. al., Software Engineering Environments: Automated Support for Software Engineering, McGraw-Hill, 1992.
3. Brown, et. al., "A Framework for Evaluating Software Technology", IEEE Software, September 1996, pp39-49.
4. Coplien, J., & Schmidt, D., eds., Pattern Languages of Program Design, Addison-Wesley, 1995.
5. Fuggetta, A., "A Classification of CASE Technology", Computer, Dec. 1993.
6. Gamma, et al, Design Patterns: Elements of Reusable Design, Addison-Wesley, 1995.
7. Kara, D., "Client/Server Development Toolsets: A Framework for Evaluation and Understanding", Application Development Expo, New York, NY, April 4, 1995.
8. Konti, J. and Tesoriero, R., "A COTS Selection Method and Experiences in its Use", Proceedings of 20th NASA Software Engineering Workshop, Greenbelt, MD, November, 1995.
9. Konti, J., "A Case Study in Applying a Systematic Method for COTS Selection", Proceedings of 18th International Conference on Software Engineering, Berlin, Germany, March 25-26, 1996.
10. Sharon, D., "A Reverse and Re-Engineering Tool Classification Scheme", IEEE Software Eng. Tech. Committee Newsletter, Jan. 1993.
11. Utz, W., Software Technology Transitions: Making the Transition to Software Engineering, Prentice Hall, Englewood Cliffs, NJ, 1992.