

Experiences in the Adoption of Requirements Engineering



James Van Buren
David Cook

Based on the paper published in *CROSSTALK*,
The Journal of Defense Software, December 1998



ABSTRACT

It has been known since as early as the 1950s that addressing requirements issues improves the chance of systems development success. In fact, whole software development standards (such as MIL-STD-2167, MIL-STD-2167A, MIL-STD-498, and IEEE/EIA 12207) were designed to enforce this behavior for software-intensive systems. Relatively recently (sidebar), a new field of study, requirements engineering, has begun to systematically and scientifically address barriers to the successful use of requirements in systems development. Since 1992, the Software Technology Support Center (STSC) has been helping organizations adopt new technologies. This article defines requirements engineering (RE) from the viewpoint of technology adoption, discusses which RE technologies are most critical to mission success, why, and which are most difficult to adopt, and outlines successful adoption approaches.

REQUIREMENTS ENGINEERING

Requirements engineering as a field addresses requirements issues in a holistic manner. Understanding the interrelationships between the various requirements activities and how they support each other is as important as understanding the technical details of any one of the individual activities. Contrast this to the 1970s and 1980s, when the software engineering community focused

essentially only on requirements analysis, or the early 1990s, when requirements management was the fad. This holistic approach is the great requirements insight of the 1990s. Understanding requirements activities from this view helps engineers build and follow life-cycle models that account for their project's business goals, the attributes of their requirements, and the strengths and weaknesses of their requirements technologies.

CHALLENGING OLD ASSUMPTIONS

One must challenge the assumption that a requirements specification is equivalent to a development contract. For some projects, blind adherence to this assumption makes project success much more difficult. Once this assumption is challenged on a project basis, this drives what requirements are needed and for what they are needed. Once it is recognized that the project's goals (such as time to market or long-term maintainability) and attributes (such as requirements volatility) should drive its life-cycle development and requirements process, it is fairly straightforward to build or tailor, with appropriate emphasis on the requirements specification, a requirements engineering process. This is overwhelmingly the number one requirements engineering technology adoption lesson learned (and the second great requirements insight of the 1990s).

REQUIREMENTS ENGINEERING BACKGROUND

Papers as early as 1956 have discussed the importance of requirements definition in software development, but it was not until 1976 at the International Conference on Software Engineering that requirements engineering was recognized as a subdiscipline of software engineering. In fact, at the 1968-69 NATO software engineering workshops (where the term "software engineering" was first coined), software engineering was explicitly decomposed into only design, code, and test activities (Ref. [2]).

The first time we noticed the term "requirements engineering" was in conjunction with the 1993 International Symposium on Requirements Engineering (RE '93), the first conference devoted entirely to requirements topics. Before 1993, it seemed that requirements research was stovepiped into areas such as "requirements management" or "requirements analysis." Since 1993, the term "requirements engineering" and its accompanying thesis of holistically addressing all the requirements activities have become widespread. In addition to the RE series of conferences (Ref. [3]), the IEEE's International Conference on Requirements Engineering (Ref. [4]) meets every other year. The IEEE has also published a seminal collection of RE papers (Ref. [5]).

BAD REQUIREMENTS PROCESS LEADS TO DEVELOPMENT FAILURE

In June 1994, the Federal Aviation Administration (FAA) canceled its 10-year effort to modernize the nation's air control system. About \$1.3 billion was written off (Ref. [6]). In 10 years, the requirements elicitation phase had never come to closure. A requirements specification with a height that could be measured in yards was produced, but it was fundamentally incomplete. This is the most expensive development failure due to a requirements failure of which we are aware. One can argue that there were many other problems with the program, but it was during the requirements process that the program failed.

Tom DeMarco produced a brilliant analysis of what went wrong (Ref. [7]). He knew he was on the right track when he could not find a keyboard mentioned in the specification. This led to the observation that the customer, the FAA, was unable to specify if the system was to be centralized (Washington office's desire) or decentralized (controllers' and regional operating centers' desire). DeMarco has since lectured extensively that internal customer conflicts like this must be resolved before a specification can be completed and that conflict resolution is an overlooked arrow in a requirements engineer's quiver.

Requirements elicitation can help identify internal customer conflict. But the customer – not the requirements engineer – must resolve conflicts or the system being built is doomed to fail.

Today's requirements research (Ref. [1]) is focused on issues that come to light when the "requirements are equivalent to development contract" idea is discarded. Research concerns include:

1. How are requirements prioritized? Requirements prioritization becomes critical when fixed development dates or fast development (time-to-market) considerations drive the development rather than the need to meet all requirements.
2. How does a project cope with incomplete requirements?
3. How can requirements engineering support the commercial development paradigm (where feature sets, product sizing, and market window are the focus rather than functional requirements)?
4. What is the interdependence of requirements and design (for example, a strong interdependence is necessary when building commercial-off-the-shelf based systems)?

ELEMENTS OF REQUIREMENTS ENGINEERING

Even if one breaks the link between requirements specification and development contract, this does not alter the need to perform requirements activities. They are merely performed with a different flavoring of objectives. Individual requirements technologies are still best viewed from the perspective of the requirements objectives they address. The caveat is that they must support the chosen overall requirements engineering process.

We divide requirements engineering into the categories of elicitation, analysis, management, validation and verification, and documentation. This taxonomy helps one understand both the requirements problems and the requirements technology adoption issues that face our clients. Like the biological taxonomy, ours is intended to be a living entity, subject to slow change. As we have learned more about RE and as RE matures as a field, our taxonomy has, in fact, changed.

REQUIREMENTS ELICITATION

This field addresses issues that revolve around getting customers to state exactly what their requirements are. Programs large and small still fail to reach closure on this step, in spite of adequate effort. Other programs reach closure, but do not capture all the requirements. This is perhaps the area of requirements engineering with the highest incidence of malpractice. Software engineers all agree that requirements elicitation is important, yet they uniformly spend too little time performing it.

Requirements elicitation is the only requirements engineering field without a definitive technical solution, yet good informal solutions exist. The lack of technical solutions is expected because the elicitation problem is human in nature. The issue is that customers often cannot state what the requirements are because they either do not know what they want, are not ready to fully define what they want, or are unable, due to outside influences, to decide what they want. The FAA sidebar above outlines the classic example of this last behavior.

The biggest elicitation failings (missed requirements and the inability to state requirements) manifest themselves as omissions or inconsistencies, which may not become apparent until requirements analysis or systems acceptance testing or even systems use (see Ariane Flight 501 sidebar). Customers must understand that incomplete, inconsistent, or ambiguous requirements, at best, cost a lot of money. At worst, they guarantee failure of the entire system. Spending additional time "fleshing out" requirements always results in an overall cost saving.

Elicitation Mechanisms

During requirements elicitation, one must derive the system requirements from domain experts – people familiar with their domain but not necessarily with building software systems. The system developers must therefore be conversant in the terms and limitations of the domain, since the domain experts are probably not conversant in the terms and limitations of software engineering. To help

overcome this potential communication barrier, elicitation mechanisms are needed to add formality to what could otherwise be a "seat of the pants" methodology.

Informal elicitation mechanisms (such as prototyping, Joint Application Development, Quality Function Deployment, Planguage (Ref. [8]), or good old structured brainstorming) address motivated and able customers who do not know how to express their needs. We conjecture that the root cause of communication barriers is that the term "requirements" is used differently by different parties. The "requirement" that is the output of the elicitation process has a specific meaning to a software or systems engineer. It is a real need of the customer, it is testable, and it may be prioritized. The requirement can also be validated by the customer.

To an uninformed customer, a requirement is often simply only a statement of need. Elicitation mechanisms help overcome this communications barrier by helping the customer understand and state needs in an objective manner. There are interesting side effects of these mechanisms. Customers develop an ownership in the outcome of the

development effort and better understand the problem that is being solved. The customer's needs and desires (nice-to-haves) are separated explicitly. Developers establish a working relationship with the customer and have an understanding of what the problem is and where trade-offs can be made.

Despite the hype, formal methods are not a complete solution. They are not effective in involving the customer; however, they are effective in gaining greater understanding of constrained parts of the problem domain. They can assist an elicitation approach based on informal techniques, but cannot stand on their own.

The adoption of elicitation technologies first requires the recognition that elicitation can be a problem and recognition that the term "requirement" has different meanings to different people. Once this occurs, the straightforward plan is to obtain training for the organization's elicitors in a variety of elicitation techniques and interpersonal skills. Practicing elicitation on real projects involves the use of a variety of elicitation and validation techniques that together increase the probability that the customer has properly

ARIANE FLIGHT 101 - Bad Requirements Lead to Systems Failure

On June 4, 1996, the maiden flight of Europe's Ariane 5 rocket ended in catastrophic failure with a complete destruction of the rocket and its payload (Ref. [9]). The cause was a software error, perhaps the most expensive software error on record. The root cause of this error was a breakdown in the requirements process – not in the software design or coding processes – that was not caught by the developmental verification and validation process. Within the requirements process, there were problems with elicitation, analysis, and verification and validation.

At liftoff plus 30 s, an operand exception was generated in the Inertial Reference System (SRI) computer during a conversion of a 64-bit floating entity into a 16-bit signed integer. This caused the SRI to crash and output a diagnostic bit pattern. The redundant backup SRI had also crashed 72 ms earlier for the same reason. Ariane 5's on-board computer interpreted the SRI's diagnostic bit pattern as valid commands and ordered full nozzle deflections of both the solid boosters and the main engines. The rocket was then destined to break up.

The official Inquiry Board found that the primary causes of the crash were "... specification and design errors in the software ..." and "... reviews and tests ... did not include adequate analysis and testing." The software requirements were incomplete and neither the requirements analysis activities nor the requirements verification and validation process discovered this omission. They also found fault in exception handling requirements, which basically were to log the error and terminate. This arose from a faulty belief that random hardware failures were the only reason for an exception and that systematic software errors would never occur (systems analysis failure).

The software and software requirements were essentially reused from Ariane 4. An explicit decision had been made to not include the Ariane 5 normal liftoff trajectory as part of the software requirements (requirement elicitation failure). When computing the alignment horizontal bias for the Ariane 5 trajectory, the operand exception will always occur at about liftoff plus 30 s. The operand exception will never occur for the Ariane 4 trajectory in the first 43 s of flight. Had the trajectory been included as a requirement, the official Inquiry Board believed that the developer's analysis and testing process would have observed this exception.

Exception handling had been turned off because of a processor performance requirement (maximum 80 percent processor utilization). The Ariane 4 analysis indicated that horizontal bias would remain within the range of a 16-bit signed integer with the Ariane 4 trajectory. This justification analysis was not easily available to the Ariane 5 development team (requirements process failure).

The reuse of the Ariane 4 software requirements was also flawed. Ariane 4's requirement to continue computing alignment (which includes the horizontal bias) for 50 s after entering flight mode (liftoff is 7 s into flight mode) to support a late hold is not needed for Ariane 5. The original requirement may even be a bit flawed because the alignment calculation is physically meaningless after liftoff (systems analysis failure).

The Inquiry Board also took issue with the verification and validation processes. Its primary finding was that these processes did not identify the defect and were thus a "contributory factor in the failure." The explanation given for not testing or analyzing the Ariane 5 trajectory was that it was not a part of the requirements specification (requirements verification and validation failure).

stated its real requirements and that the development organization understands them. Organizations need to recognize that elicitation is a skill learned through practice. Practitioners generally are proficient after training, but not expert until after several projects.

REQUIREMENTS ANALYSIS

Requirements analysis serves two primary purposes:

1. It is used to make qualitative judgments, i.e., consistency, feasibility, about the systems requirements.
2. It is a technical step in most systems development life cycles in which an extremely high-level design of the system is completed. This high-level design consists of decomposing the system into components and specifying the component interfaces. The critical output for most software development requirements analysis activities is the interface specification for the decomposed components.

There are a number of well-understood technical approaches to analysis, i.e., OMT, Schlaer Mellor, Structured Analysis, and UML. Most have good commercial tool support. Organizations do not have difficulty in finding experts, in developing (through training, mentoring, and experience) experts, or in finding tools to support their analysis efforts. Instead, technology problems arise from both over- and underanalysis. For example, well-funded programs tend to overanalyze. They analyze everything that can be analyzed without first determining what should be analyzed. Often, detailed design occurs during this analysis phase. Programs with cost constraints suffer the opposite fate. They tend to underanalyze, probably as a cost savings measure.

Many methods and tools support analysis, and there are both tool-related and training-related technology adoption issues. Tool-related problems occur when there are inconsistencies between a tool's implied development process and the organization's standard development process. Tool vendors have come a long way this decade in addressing this issue, but it has not gone away. Tools become "shelfware" if they impose their process over the organization's process, even if the organization's process is undefined and ad hoc.

To adopt requirements analysis successfully, pilot the analysis methods manually, identify what steps need to be automated, make a tool selection, tailor the tool's use, then use the tool. Over time, the organization's process can gradually evolve.

We have observed that standard training plans are often inadequate to address new analysis methods. The detailed training of how to apply a method or a tool is necessary, but not sufficient. Education may also be needed if the new method is radically different from established methods, as object-oriented differs from structured. In addition, mentoring on the first pilot project is necessary for all but a few individuals. When adopting new analysis methods, always plan for education, training, and mentoring.

REQUIREMENTS MANAGEMENT

Requirements management addresses aspects of controlling requirements entities. Requirements change during and after development. The accepted requirements volatility metric is 1 percent of requirements per month (Ref. [10]). If it is much less, one should ask if the system will be desirable to its intended audience. If it is much more than 2 percent a month, development chaos is all but ensured.

Requirements management is the requirements issue that most impacts military software projects. In a 1993 report, Capers Jones found that 70 percent of all military software projects are at programmatic risk because of requirements volatility (Ref. [10]). The root causes of this volatility (discussed below) have not disappeared, so we believe his finding is still true.

With one exception, requirements volatility is uncontrollable. It will occur as a byproduct of building a useful product, and one's development processes need to account for it. Software systems do not exist in isolation. As a new system is built, the system will affect its environment, which will in turn change its environmental requirements. This is inevitable and indicates one is building the right system.

Most development efforts, and all development efforts for which requirements management is important, take time, sometimes on the order of years. Over these periods, underlying technologies, user expectations, and even laws change, to name just a few of the many possible external interfaces. If one is not getting requirements change requests on large projects, one needs to ask why. The one source of volatility that can be controlled is the quality of the initial requirements specification. Good elicitation techniques can limit rework as a cause of volatility.

Requirements Management Technologies

Fortunately, the technologies needed to address the requirements volatility issue are relatively simple. The organization needs a defined interface mechanism with its customer by which requirements are changed, a mechanism (usually a configuration management system or a requirements management (RM) tool) capable of defining the current requirements baseline, and a development approach, i.e., incremental life cycle, that supports the anticipated requirements volatility. The primary adoption issues that must be addressed are:

1. Building senior management recognition that this defined mechanism is necessary.
2. Having the discipline to always follow this change mechanism.

The old adage that "the customer is always right" is not an absolute. When customers ask for a requirements change, they must be told the impacts of that change, usually in terms of other prior commitments, and then be allowed to make the final decision. The software development organization must never unilaterally add a requirement.

Tools and Adoption Technologies

Effective management usually implies both exerting control over and knowing the status of requirements. There is a whole class of RM tools (such as Requisite Pro, DOORS, RTM, and Caliber-RM) that automate the tracking of requirements across life-cycle phases. They also support many of the requirements baselining and requirements documentation issues. They are particularly useful for programs that wish to follow a requirements-centric development approach. Technology adoption issues associated with these tools are the standard issues for tools. Understand what the tool will be used for, find a tool that meets those needs, then build a plan for adopting the tool, being careful that the tool is not used for purposes beyond identified needs.

The Software Capability Maturity Model (SW-CMM) (and to some extent the Systems Engineering and the Software Acquisition CMMs) includes requirements management as a Level 2 key process area (KPA). Within this context, the requirements management KPA only applies to managing requirements change. Other requirements management tracking and status activities could apply as part of the Level 3 SW-CMM KPA, Software Product Engineering, depending on the organization's development approach. When implementing a CMM-based technology change program, remember that managing the customer interface comes first. A second step might be tracking the status of requirements across the development life cycle and using that information to manage the development.

REQUIREMENTS VALIDATION AND VERIFICATION

The requirements verification and validation (V&V) portion of RE addresses how quality is built into the RE process. Validation ("Are we building the right system?") addresses the issue of building the system the customer wants. This quality step should identify missing and extra requirements. Validation activities always occur as part of system acceptance testing and also typically, but not always, as part of the elicitation process. There are several orders of magnitude cost difference in requirements misunderstandings that are identified as part of elicitation, before development resources have been expended, vs those that are found during system acceptance testing. This points out the critical need for the elicitation process to include validation.

Customer and domain expert input are necessary for validation. In fact, attempting to validate a system without customer input is equivalent, in the words of one of our customers, to designing a "self-licking ice cream cone." The necessity of user input is another reason formal methods are insufficient for building systems – the customer usually cannot understand and does not want to learn how to validate using formal mechanisms. Mechanisms that a customer can understand easily (and hence validate easily) are almost always based on clear language and easily understandable pictures – input the customer can already comprehend. If the customer has to learn a new notation or method to validate a system, a new quality issue is intro-

duced to the validation process: lack of a clear understanding of the method. The focus needs to be on the solution, not on the method.

Verification ("Did we build the system right?") addresses the issue of meeting all the requirements. Typically, the verification method and sometimes the verification level is included in the requirements specification. Verification methods include demonstration (observable functional requirements), analysis (collected and processed data), simulation (use of a special tool or environment to simulate the real world), and inspection (examination of source code and documentation). Verification levels depend on the intended development environment. They specify the development life-cycle stage at which the verification will be performed, i.e., unit test, integration test, system installation, or flight test.

From a technology adoption perspective, requirements V&V is a question of designing a development life cycle that meets the needs of the product. Emphasis needs to be placed on validation in the elicitation phase. It should be considered software engineering malpractice if requirements V&V is not also included during the design and coding phases. Validation and verification must be performed after the system has been built.

REQUIREMENTS DOCUMENTATION

There are a number of potential standards for structuring requirements specifications. American National Standards Institute/Institute of Electrical and Electronics Engineers-STD 830-1993 specifically addresses requirements specifications. The life-cycle standards Electronic Industries Association (EIA)/IEEE 12207 and the withdrawn standards MIL-STD-2167A and MIL-STD-498 specify another similar format. The basic contents of all these are the same: they all include an overall description, external interfaces, functional requirements, performance requirements, design constraints, and quality attributes.

Another school of thought posits that there should be a bare minimum of requirements documentation. A concept of operation document or a user's manual are all that are needed for a requirements statement. This makes sense for applications where time to market is more important than long-term maintainability.

Adopting requirements documentation technology is fairly straightforward. One should choose a standard that fits the life-cycle requirements of the system, tailor that standard to fit the system's specific requirements, then apply it. If one intends to use an RM or a requirements analysis tool to automate a portion of the document generation, one should pilot the documentation process. Experience has always shown this to be much more difficult than originally envisioned.

TECHNOLOGY ADOPTION

A technology adoption process will increase the probability of a successful technology change. At the STSC, our technology adoption process is based on the IDEAL Model

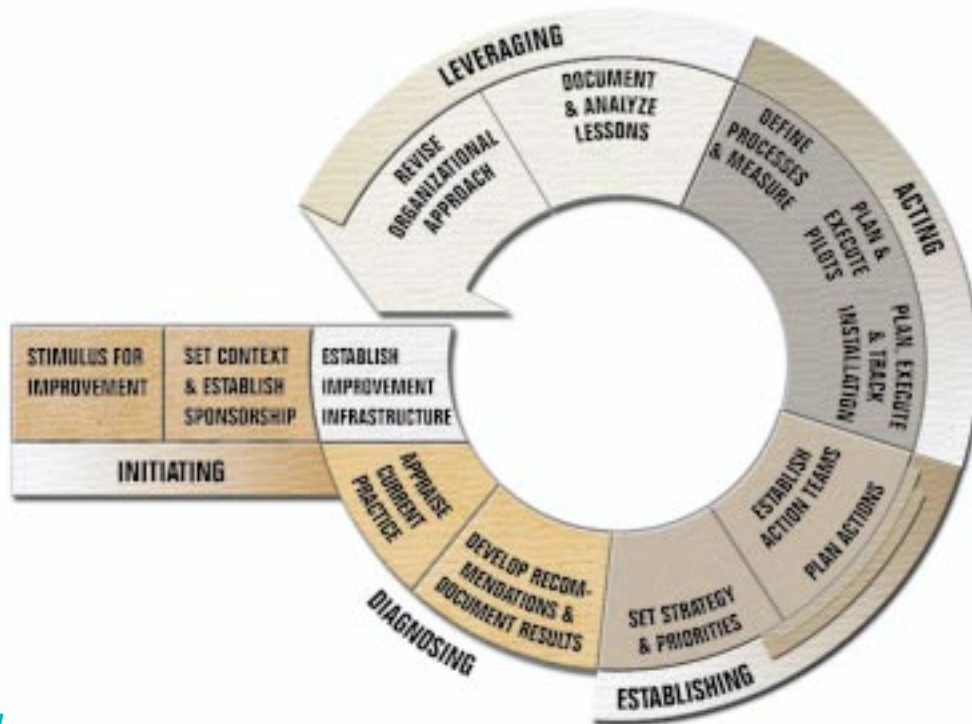


Figure 1
The IDEAL model

(Ref. [11]) (Figure 1). We also use two other important adoption principles:

1. Small improvement steps.
2. Address needs at all levels of the organization.

We discuss the IDEAL Model here, not necessarily because it is the best technology adoption model (although we believe it is), but to demonstrate the importance of picking an adoption model, basing one's adoption process on that model, and improving one's model and process over time. The following is a synopsis of the five steps in the IDEAL Model:

1. **Initiate** – Obtain and maintain sponsorship.
2. **Diagnose** – Assess current practice.
3. **Establish** – Produce a plan to address shortcomings.
4. **Act** – Pilot and use the technology(ies).
5. **Learn** – Collect lessons learned, next steps (such as rollout), cycle back to diagnose.

Our RE field experience indicates that organizations planning RE technology purchases or process changes generally do not follow a process or a model for technology adoption. If they are not planning a purchase, they usually do not even realize that what they are doing is technology adoption. This differs from organizations interested in process improvement, which tends to produce detailed technology adoption plans based on models like IDEAL.

When a tool or a method is purchased, the vendor is consulted regarding its specific adoption recommendations. With one exception, an SW-CMM Level 5 organization, we

have not seen any formal mechanisms that use lessons learned from prior technology adoptions. Our advice is that vendors' recommendations become the functional requirements for the adoption plan and that the plan be driven by the organization's past adoption experiences.

Small Steps

Two adoption principles must be adhered to when building adoption plans. First is the principle of small steps. Many small process improvement steps have a greater chance of success than one giant process improvement leap.

To build a requirements-centric development process, one cannot jump right to the final state. Instead, the first step might be to get all one's requirements changes under control. The second step would be to pilot an RM tool that reports the development status of every requirement and produces requirements documents. The final step would be to use requirements status information to manage one's development efforts.

Address All Organizational Levels

The second adoption principle is that plans must address all levels of the organization: the individual, the project, and the organization in its entirety. For example, an adoption plan to meet the objectives of the RM KPA of the SW-CMM would involve all three levels in different ways.

Senior management, representing the organization, would need background RM training (indoctrination) on why controlling requirements is an important issue. They will have to issue and enforce an organizational policy. More important, they may have to stand up to the organization's

customers and tell them that, unlike the old days, the customers cannot change or add requirements in an uncontrolled manner.

On the project level, a system, most likely tools and processes, will be needed to track requirements baselines. Ultimately, individuals must have the discipline to never allow requirements to creep into the system outside of standard channels.

In the prior example, the adoption emphasis needs to be placed at the organizational level. If that step succeeds, the others will generally follow. But the level of emphasis differs depending on the type of requirements technology. For example, emphasis should be placed on the individual adoption issues when elicitation technologies are being adopted. Elicitation is essentially an individual skill, bordering on art form.

Adoption Effort

Table 1 examines the technology adoption issues for each of the requirements technologies from the perspective of various organizational levels. Although all issues need to be addressed, those that are italicized are the issues critical to adoption success for each of the technology areas.

Finally, there is the question of how hard technologies are to adopt. Some technologies require a lot of effort to master. Analysis technologies are an example of this. The elicitation technologies require a medium amount of effort to become proficient but a lot to master. The other requirements technologies all require a relatively less amount of effort to master.

Another view of the difficulty of technology adoption is how difficult it is to verify that the technology has been adopted. Elicitation is extremely hard, analysis is moderate, and the others are easy. Table 2 summarizes the relative

Table 1
Requirements Engineering Technology Adoption Issues

Technology	ORGANIZATIONAL LEVEL		
	INDIVIDUAL	PROJECT	ORGANIZATION
Elicitation	<ul style="list-style-type: none"> • <i>Training</i> • Mentoring 	<ul style="list-style-type: none"> • Technique selection • Tailoring of V&V strategy 	<ul style="list-style-type: none"> • Persistent training program
Analysis	<ul style="list-style-type: none"> • Education • <i>Training</i> • <i>Mentoring</i> 	<ul style="list-style-type: none"> • Technique selection • Tailoring of V&V strategy 	<ul style="list-style-type: none"> • Persistent training program
Management (control)	<ul style="list-style-type: none"> • Discipline to follow the process 	<ul style="list-style-type: none"> • Tailored process • Tool adoption (if necessary) 	<ul style="list-style-type: none"> • Policy • <i>Policy enforcement by organization's executives</i>
Management (status)	<ul style="list-style-type: none"> • Training 	<ul style="list-style-type: none"> • <i>Tool piloting</i> 	<ul style="list-style-type: none"> • Organizational standards
Validation and Verification	<ul style="list-style-type: none"> • Training (reviews, inspections, test tools) 	<ul style="list-style-type: none"> • <i>Tailored approach</i> 	<ul style="list-style-type: none"> • Enforcement
Documentation	<ul style="list-style-type: none"> • Training 	<ul style="list-style-type: none"> • <i>Tailored approach</i> 	<ul style="list-style-type: none"> • Organizational standards

Table 2
Relative Difficulties of the Various Requirements Technologies

RE TECHNOLOGY	EFFORT REQUIRED TO ADOPT	EFFORT REQUIRED TO VERIFY THAT ADOPTION HAS WORKED
Elicitation	Moderate for proficient application; Hard for expert application	Hard
Analysis	Hard	Moderate
Management (control)	Easy (getting commitment is sometimes hard)	Easy
Management (tracking status)	Moderate	Easy
Documentation	Easy; Moderate, if the documentation is to be automatically generated	Easy
Verification and Validation	Easy	Moderate

difficulties of the various requirements technologies. Note that the table only captures relative differences between the various RE technologies and only addresses adoption issues; it does not address the relative difficulty of practicing each of the requirements activities.

We have observed that organizations usually succeed when adopting "easy" technologies, even without outside assistance. They usually fail when adopting "hard" technologies, unless supported by external consultants.

SUMMARY


Requirements engineering is the systems development activity with the highest return on investment payoff. The cost saving resulting from finding errors during verification and validation of requirements can be as high as 200-to-1 (Ref. [12]).

However, the requirements task is inevitably always harder than it first appears. If one were to presuppose that the customers were motivated and able to specify accurate and complete requirements, that the requirements would never change, and that there were no cost or schedule constraints placed on a development, there would not be any requirements issues. Unfortunately, none of these presuppositions are true. RE is the technical field of study that attempts to address and balance these issues.

The requirements phase is the interface between a customer's needs and the technical development process. The skills needed to perform requirements activities are a marriage of the people skills necessary to interface with the customer and the technical skills needed to understand the development process. At their heart, requirements skills are human based. Tools and technologies can only support requirements activities. When evaluating and adopting new RE technologies, focus on those technologies and adoption issues that support the human requirements engineer.

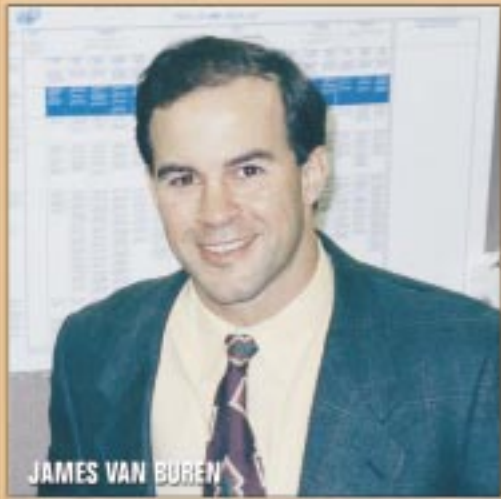
REFERENCES

- [1] Siddiqi, Jawed and M. Chandra Shekaran, "Requirements Engineering: The Emerging Wisdom," *IEEE Software*, March 1996, pp. 15-19.
- [2] Boehm, Barry, *Proceedings of the 2nd International Conference on Requirements Engineering*, 1996, p. 255.
- [3] International Symposium on Requirements Engineering (RE) '92, RE '94, and RE '96.
- [4] International Conference on Requirements Engineering (ICRE), 1994, 1996, and 1998.
- [5] Thayer, R.H. and M. Dorfman, eds., *Software Requirements Engineering*, 2nd ed., IEEE Computer Society Press, Los Alamitos, Calif., 1997.
- [6] Barlas, Stephen, "FAA Shifts Focus to Sealed-Back DSR," *IEEE Software*, March 1996, p. 110.
- [7] DeMarco, Tom, *International Conference on Requirements Engineering*, Tutorial, March 1998.
- [8] Gilb, Tom, "Requirements-Driven Management: A Planning Language," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Ut., June 1997, p. 18, language description is available at <http://www.stsc.hill.af.mil/SWTTesting/gilb.html>.
- [9] "Ariane 5, Flight 501 Failure," Report by the Inquiry Board, July 19, 1996, <http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html>.
- [10] Jones, Capers, *Assessment and Control of Software Risks*, Prentice-Hall, Englewood Cliffs, N.J., 1994.
- [11] Gremba, Jennifer and Chuck Myers, "The IDEAL Model: A Practical Guide for Improvement," *Bridge*, Software Engineering Institute, Issue 3, 1997, also available at <http://www.sei.cmu.edu/ideal/ideal.bridge.html>.
- [12] Davis, A., *Software Requirements, Objects, Functions, and States*, Prentice-Hall, Englewood Cliffs, N.J., 1993.



BIOGRAPHIES

Experiences in the Adoption of Requirements Engineering



Jim Van Buren is on the technical staff of Draper Laboratory, which he joined in 1983. He has supported the STSC and the STSC's customers since 1989 in requirements, design, object-oriented technologies, and other technologies relating to the development of software. He is an SEI-authorized Personal Software ProcessSM (PSP) instructor. He currently serves as Draper's technical program manager at the STSC. Jim has a BS in engineering and computer science from Cornell University.

vanburej@software.hill.af.mil



David Cook is a Principal Member of the Technical Staff at Draper currently working under contract to the STSC. He has over 25 years experience in software development and has lectured and published articles on software engineering, requirements engineering, Ada, and simulation. He has been an associate professor of computer science at the U.S. Air Force Academy, deputy department head of the software engineering department at the Air Force Institute of Technology, and chairman of the Ada Software Engineering Education and Training Team. David has a BS in computer science from University of Central Florida, an MS in teleprocessing from University of Southern Mississippi, and a PhD in computer science from Texas A&M.

cookd@software.hill.af.mil