

# Agile Requirements Definition and Management Will Benefit Application Development

Gartner RAS Core Research Note G00126310, Matt Light, 18 April 2005, R1741 03232007

**The flexibility with which requirements are gathered and managed shows how disciplined an AD process is. AD organizations with automated requirements definition and management environments will better support change control, gain testing efficiencies and reduce future maintenance burdens.**

## WHAT YOU NEED TO KNOW

The manner in which requirements are gathered, stored and managed shows the level of discipline in an application development process. Agile approaches to defining requirements, a few at a time, have a place in “just enough” software process architectures. An AD organization with an automated requirements environment will better support change control efforts, gain testing efficiencies and potentially reduce its future maintenance burdens.

## STRATEGIC PLANNING ASSUMPTION(S)

By 2009, automated approaches to supporting RDM will enable a reduction in the cost of AD quality by 30 percent (0.8 probability).

By 2009, user satisfaction levels with medium-size to large systems that are developed with well-automated RDM processes will usually improve from “fair” to “good,” or the equivalent (0.8 probability).

By 2009, maintenance- and enhancement-phase costs for medium-size to large systems that are developed with well-automated RDM processes will decline by 10 percent (0.8 probability).

By 2008, a growing market encompassing automated RDM tools will exceed \$400 million in annual revenue for licenses, related services and maintenance (0.6 probability).

## ANALYSIS

At a time when many large, internal application development (AD) organizations have turned to rapid AD

### Key Issues

What are the suitable levels of requirements, project management and other processes for application development to deliver Web services and other projects?

What are the dynamics affecting this market?

What is the size of the market, and how are vendors positioned in it?

Where are the future growth prospects for this market?

What are the best practices to maintain quality while improving delivery cycles?

(RAD) and NeoRAD (or “agile”) approaches for 25 percent to 35 percent of their AD project portfolios, meticulous procedures for thorough application requirements definition and management (RDM) have become suspect. Many AD managers see such procedures as costly overhead – that is, “analysis paralysis,” which introduces unacceptable delays and undermines agile AD.

On the AD side, in their eagerness to proceed directly to programming and avoid being perceived as slow, less-proficient developers often practically skip the documentation of requirements. On the business side, business sponsors have often been unwilling to provide sufficient numbers of users for multiple joint AD (JAD) sessions to gather, define and validate requirements. As a result, expertise in requirements gathering declined as JAD fell out of favor during the 1990s.

Lately, however, the general increase in the external sourcing of AD can pull the same AD organization in the opposite direction – toward more-extended RDM, to structure contracts with external service providers (ESPs). When sourcing AD services from third parties, clear requirements specifications are necessary. However, these apparently opposing drivers can create tension in many AD organizations.

## Getting Good Requirements

When contracting with an ESP, an internal AD organization's business analysts and designers are best positioned to define and document application requirements. Their access to users isn't seen as billable time, but rather as a natural part of providing the requested system. Their understanding of business processes is often based on years of experience with the same company, often working on related applications. In addition, their familiarity with the software "ecosystem" provides design expertise exceeding that of contracted AD service providers.

More IT organizations in industries that are increasingly enabled by software (for example, financial, retail, communications and others) are realizing that relying on ESPs to capture, define and redefine requirements is often more costly than doing it internally and then coordinating with the ESP (if the programming is to be contracted – offshore, for example). Regaining expertise in the "lost art" of requirements will reduce the risk and cost of building accurate systems with ESP assistance, and users will be more satisfied with applications that more-exactly meet their primary needs. In addition, well-defined and managed requirements can help restrain application total cost of ownership, because staffs performing future maintenance and enhancement work will need less time and effort to identify system elements on which to work.

The use of requirements tools – which increasingly provide collaboration, simulation and other features and interfaces – will amplify these savings and satisfaction benefits. By 2009, automated approaches to supporting RDM will enable a reduction in the cost of AD quality by 30 percent (0.8 probability). By 2009, user satisfaction levels with medium-size to large systems that are developed with well-automated RDM processes will usually improve from "fair" to "good," or the equivalent (0.8 probability). By 2009, maintenance- and enhancement-phase costs for medium-size to large systems that are developed with well-automated RDM processes will decline by 10 percent (0.8 probability).

Gathering the requirements of a new system or major enhancement presents a project team with its first (and, in many ways, most important) choice: between a disciplined engineering approach and an ad hoc "code and go" approach. We advise AD organizations to define a software process architecture that enables developers to appropriately employ "just-enough processes" to different project types. Too often, analysts have described systems to address a "business problem" that an amorphous set of users will be addressing. Having pre-defined the problem, the analysts then formulate questionnaires and conduct interviews based on unexamined assumptions formed separately from users' ongoing experiences. By contrast, NeoRAD or "agile" development methodologies explicitly call for including users on the AD team. Users best understand the functional, performance and reliability needs of the applications they use in their jobs. Thus, analysts familiar with the business and with users' needs are necessary, but insufficient.

Users, however, tend to be little aware that requirements involve attributes beyond their functionality needs – attributes as simple as a requirement's version number, which becomes important as understanding of the requirement is revised and refined. Other examples of nonfunctional attributes include designation as a parent or child requirement, or indications of any dependencies – perhaps including links to external projects. A structured approach to considering requirements (see Note 1) can more-fully describe the requirements in context. The lack of a thorough description of requirements can yield disconnects and defects that are invisible to developers until future iterations, which may require extensive rework. For larger, more-critical systems, traceability among requirements is often necessary to understand the impact of changes. With traceability, project managers can analyze the effects of changes on the rest of the system and the project overall. Each documented user requirement should be traceable to a software function.

## Note 1

### Elements of RDM

*Content: Analysts should examine these items to ensure accurate and comprehensive requirements definition:*

- Does the specification document the choice of users? Is this the best list? How do you know? Are users classified by favored status (target, extraneous, undesired)? Is there at least one undesired user on the list (showing consideration of security)?
- Does the specification document the user inclusion strategy? Does it describe how users will be represented (by a surrogate, limited participation, complete involvement)? Is the representation to be classified by behavior (controlled experiment) or judgment (you ask them)? Will the representation be continuous or periodic? Does the specification document the list of attributes?
- Are constraining attributes documented?
- Are optimizing attributes documented?
- Are “nice to have” attributes documented?
- Are attributes correctly classified (for example, this is the No. 1 problem haunting software development)?
- Does the specification document the list of constraints? Are they specified objectively? Measurably? Testably?
- Does the specification document the list of use cases? Are they classified as “normal,” “abnormal but possible” or “beyond reasonable use”? Are they specified in stimulus-response form?
- Does the specification document the list of functions? Are they specified as problem statements, not solutions? Are they classified as evident or hidden?
- Does the specification document the list of assumptions?
- Does the specification document the list of unresolved issues?

*Process: Several processes can be performed to ensure that things aren't missed, and to reduce ambiguity in requirements. Some questions are:*

- Did naming the project or product inadvertently prejudice the requirements process toward a particular design?
- Were context-free questions used to validate the contents of the requirements specification?
- Has ambiguity been measured throughout the requirements process? Do the measures show that the requirements are sufficiently unambiguous? If not, then you aren't ready to build the product.
- Has user review of models, prototypes or application simulations validated the requirements?
- Has a user satisfaction survey instrument been designed, based on the distinguishing attributes? Have plans been made and documented as to how the survey will be applied?
- Have user expectations been defined and tested? Is there some input needed for the marketing process to help set expectations?
- Have the product's limitations been defined?

Source: Gartner, Donald Gause and Brian Lawrence

Furthermore, the reverse should hold: Each software function should be traceable to a user requirement, because if a software function is developed without being traceable to a requirement, it will result in unspecified (and often undesirable) system behavior. Traceability links requirement attributes to show that user needs are met and that the system won't work in unexpected ways. In a just-enough-process environment, NeoRAD projects can appropriately dispense with considering all requirements attributes and supporting traceability – particularly when the application is “opportunistic” for a smaller constituency and isn't expected to integrate with or provide data to more-critical systems. NeoRAD approaches call for continual requirements definition, sometimes documented in the form of test cases. As under Extreme Programming, defining test cases before building (or

assembling) the release's feature set encourages accurate requirements, because building the test case can reveal misunderstandings, and because unspecified features encounter the test-case hurdle. Additionally, design problems may be exposed early, because of the need to implement features in short time frames (for example, a week or two vs. six months or more).

However, one danger of NeoRAD approaches arises from stress on the quick delivery of priority requirements – after five or 10 iterations, requirements gaps or redundancies may not be apparent, and the impact of changes to requirements may be difficult or impossible to discern. Also, users and developers often have difficulty judging when priorities diminish; so, for example, the same sense of urgency may be forced on lower-priority features after

four to eight months of development. With many applications in the portfolio, constant change to the production environment can be chaotic, and the interruptions engendered by servicing many iterative AD projects can increase the sense of chaos. RDM processes and tools can help extend the initial understanding of user priorities to maintain the initial agreement between the business and the AD organization throughout the development, along with realistic budgets and schedules. Hence, after achieving an agreed-on number of priority features, AD organizations should establish a regular, monthly or quarterly rhythm of application releases (small and large). Releases force a “bigger picture” design view of the overall set of changes. They provide for more-efficient testing and thereby realize economies of scale in the design and test phases. Most AD organizations will find that meeting 70 percent to 80 percent of requests for minor fixes, application changes and enhancements via monthly (or quarterly) releases will minimize project and application portfolio disruption, speeding the delivery of maintenance/enhancement work and, potentially, new development. Releases reduce the volatility that can lead to regression errors when following a random release process.

## Application Support

Ongoing requirements management can be simplified if initial requirements definitions are captured in a database-based tool to enable collaborative review for completeness, use-case creation, test-case creation, traceability, and to facilitate versioning/change control. A few principal vendors represent most of the market for high-end, traditional requirements-management tools (see Note 2). These and other lesser-known players offer features that are more preferable to the awkward approach that attempts to define, plan and control requirements in spreadsheets or text only. Some innovators focus on novel approaches to prototyping “mock-ups” of applications as a way of clarifying and capturing requirements, sometimes extending their solutions to generate models or support traditional requirements traceability (even via integrations). By 2008, a growing market encompassing automated requirements management and requirements definition tools will exceed \$400 million in annual revenue for licenses, related services and maintenance (0.6 probability).

## Note 2 RDM Tools

### *Principal Software Requirements Management Tools*

IBM Rational RequisitePro

Borland CaliberRM

Serena Requirements & Traceability Management

Telelogic Doors

### *Lesser-Known Requirements Tool Innovators and Niche Players*

Apptero – Apptero 2004

Axure Software Solutions Rapid Prototyper

Compuware Reconcile (with QACenter, DevPartner)

Goda Software Analyst Pro

iRise Application Simulator

MKS Requirements 2005 (with Integrity Manager)

Sofea Profesy

SpeeDev – SpeeDev RM

SteelTrace Catalyze

TCP Integral Requisite Analyzer

### *Additional Systems Engineering Requirements Management Tools*

3SL (previously Structured Software Systems) Cradle

UGS Teamcenter (*Disclaimer: UGS is a portfolio company of Silver Lake Partners, an investment firm that also owns a substantial, publicly disclosed interest in Gartner Inc., and has two seats on Gartner’s 10-member Board of Directors. Gartner research is produced independently by the Company’s analysts, without the influence, review or approval of our investors, shareholders or directors.*)

ViewSet Pace

Vitech Core

### Acronym Key

**AD** application development

**ESP** external service provider

**JAD** joint AD

**RAD** rapid AD

**RDM** requirements definition and management