

Matching Software Development Lifecycle (SDLC) Management Tools to Your Environment

A whitepaper published by

QAVantage

<http://www.qavantage.com>



QAVantage

All trademarks referenced are the marks of their respective companies.

Table of Contents

Introduction.....3

The Middle Ground.....6

1. Active Support for Multiple Methodologies**Error! Bookmark not defined.**

2. Integrated Requirements Lifecycle Support**Error! Bookmark not defined.**

3. Data Model and Orientation ... **Error! Bookmark not defined.**

4. Traceability.....

5. Estimation, Prioritization, Reprioritization and Reporting.....

6. Simplicity in Structure and Implementation**Error! Bookmark not defined.**

7. Services and Best Practices

8. A Special Case for Ease of Use

Conclusion.....10

Abstract

This paper is for Executives, Business Analysts and Managers (Project, Program, Product, Development and Test Managers) who would like to understand more about the orientation of Software Development Lifecycle (SDLC) management tools and the business environments they are optimized for. It characterizes a large area of the marketplace (mid-level project complexity) for these tools and discusses why this area is generally underserved by today’s SDLC management offerings. It then identifies important features that SDLC management tools should possess in order to be of value to companies that have projects that fall into this market category.

About the Author

Daniele Chenal is Co-Founder, Senior Consultant and COO of QAvantage.

Daniele has over 18 years of experience in the high tech and commercial software industry working for leading technology companies including Compaq Computers, Tandem Computers, HP, HandySoft and OneSoft. Prior to QAvantage, she was Vice President of Product Management at HandySoft Global Corporation. Daniele has helped companies develop, launch, sell and maintain software solutions spanning such diverse areas as Business Process Management, GRC/Compliance, Payment Processing and E-Commerce.

QAvantage provides software products and consulting services that assist companies with software lifecycle management.

Introduction

Looking at practices in many of today's IT shops and commercial software companies, it appears that two sharply different approaches to streamline the software development lifecycle have developed. In larger companies, these two approaches may be present in different organization silos or sometimes even within the same project team.

On one hand, companies seeking to accelerate delivery and improve software quality are moving from rigid methodologies like Waterfall to Agile or Agile Hybrids. Agile's rapid iteration technique can reduce the amount of cycles analysts and developers spend producing elaborate requirements documentation. It also avoids much of the heavy process imposed for recording and collecting project management information.

On the other hand, management is also under pressure to increase oversight and control. That pressure can become acute when schedules have slipped severely or when coordination between projects is critical, or when there is need for regulatory compliance documentation or when requirements and service levels are contractually committed.

It seems a contradiction: sometimes the very same organizations, looking to streamline by implementing Agile are also contemplating the use of some heavy weight tools in hopes they will produce better documented and consistent requirements and instill some best practices. Are the two approaches really in opposition? If both approaches are used, are the benefits self-cancelling?

Changes to SDLC systems and process can be a difficult and costly transition that needs to be well thought out. Design guru Alan Cooper said it well: "It's harder than you might think to squander millions of dollars, but a flawed software development process is a tool well suited to the job."

When it comes to gauging overall software management effectiveness, it's obvious there are plenty of essential tools that help with the raw science (compilers, spreadsheets, bug tracking tools, version tracking, etc.) but what helps us with the *art* of software project or product *management*, and why do so many tools that address this area end up as shelfware?

To answer this, the discussion really needs to shift from philosophy to the brass tacks of an individual company's environment and the kind of software that they produce. That's because tools that can truly assist in some project environments are often misapplied to inappropriate areas, and that many of these tools are too dense and heavy to be worthwhile in typical software endeavors. For a discussion about Requirements Management Tools in this regard, see Forrester's excellent paper [*Selecting The Right Requirements Management Tool — Or Maybe None Whatsoever*](#)

To explore this a bit more and look at it from a wider SDLC management perspective, let's examine what is needed against a spectrum of the primary determinant of suitability: the complexity of a software development project and its environment. Let's assume for arguments sake that perhaps 60% to 80% of the software development projects in existence today fall in between the complexity extremes depicted in Table 1 below. On

the extreme ends, most all of the attributes listed are found. In the middle ground we expect a mix of attributes found in the extremes.

Attribute	Least Complex Software Project Attributes 10%-20%	Mid-Level Complexity 60%-80%	Most Complex Project Attributes 10%-20%
Structure	Single, Non-Interacting System or Component or Module	Interacting Components and Modules	Highly Integrated Systems/Multi Component
Stakeholders	Few	Stakeholders with Differing Viewpoints	Many Stakeholders with Divergent Goals and Viewpoints
Methodology	Single Development Methodology	Single or Multiple Development Methodologies	Likely Multiple Development Methodologies and Languages
Team Size	Small Team or Individual Developer	Single Large Team or Multiple Linked Teams, Some Outsourcing	Multiple Teams/Contractors Geographically Dispersed
Environments	1 or Few Hardware Support Environments	Multiple Hardware Support Environments	Multiple Hardware Support environments
Deployment	Iterative Deployment Possible	Phased Deployments Possible	Highly Synchronized Deployment/Cutovers
Requirement Scale	Dozens to less Than One Hundred Requirements	Hundreds to Tens of Thousands of Requirements	Tens of thousands of Requirements and Above
Dependencies	Few Dependencies	Many Dependencies	Many Dependencies
Schedules	Independent/Less Rigid Schedules	Milestones Dates Set, Missed Deployment Penalties Possible	Very Tight Deployment Schedules with Very High Failure Penalties
Development Tools	High Level Development Tools	Mix of High and Low Level Development Tools	Mix of High and Low Level Development Tools

Table 1

Software Development Environment Complexity Characteristics

To get a visual picture of the relative value and effectiveness we find in the tools applied to the extremes and the middle ground, we've constructed Chart 1 below.

For more complex environments (right side of Chart 1), there are a variety of excellent traditional point tools (red line) that have evolved from the overriding need for assistance in managing highly complex projects. This is especially true for eliciting, validating and managing requirements. There are also collections of point tools that manage the other facets of Application Lifecycle Management that deal well with the larger number of teams and their more complex interactions. But since these tools must have the flexibility and depth to deal with heavyweight Requirements Management and ALM, they can be very complex and costly. And in order to be successfully adopted, they require considerable ramp-up time, organizational buy-in and operational discipline. These tools have enthusiastic fans, especially in companies producing embedded systems or highly integrated, multi-component systems. That's

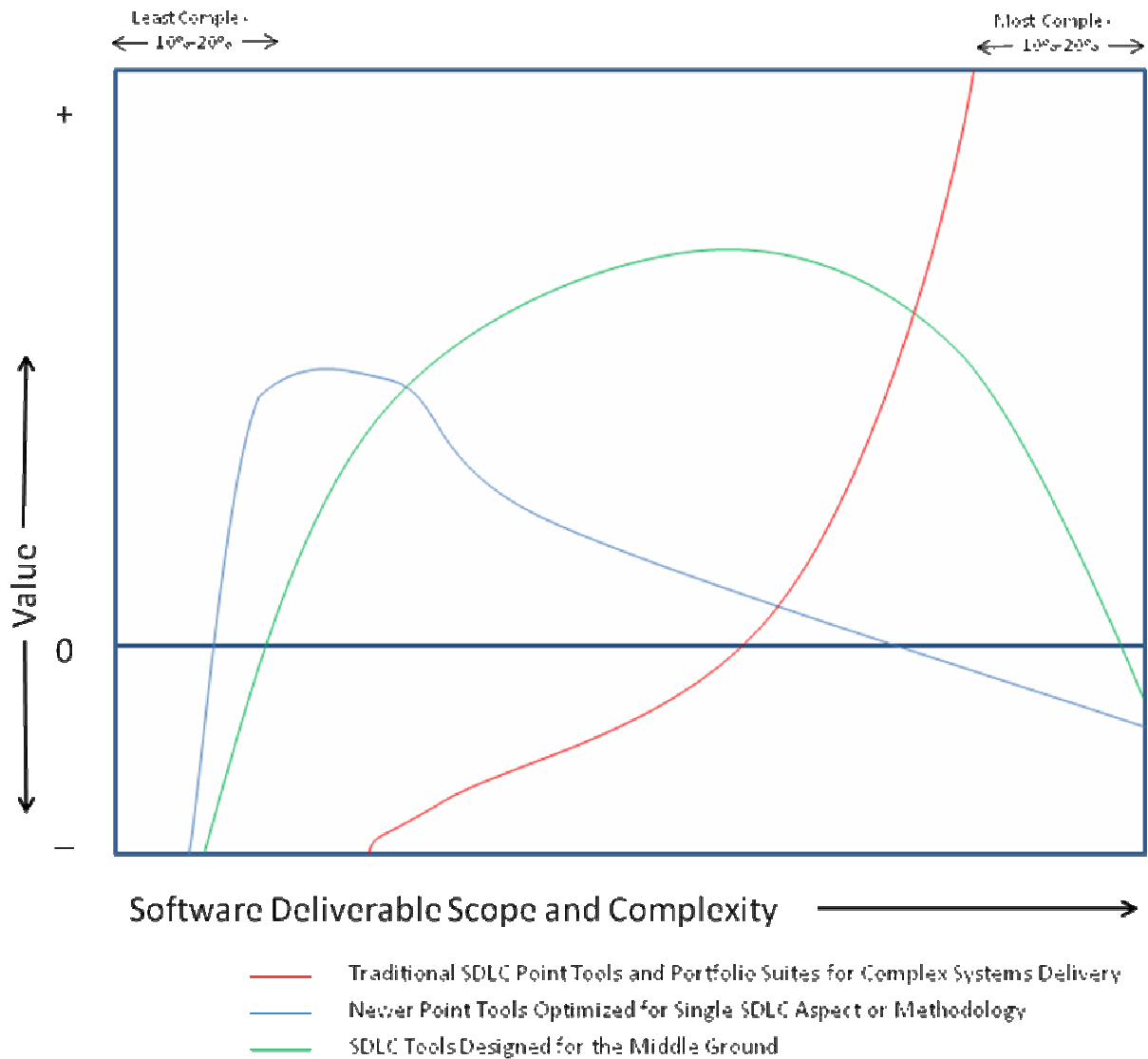


Chart 1

SDLC Management Tool Value

because the work products of many of these companies are so complex that they would simply be impossible to produce without them.

But while complex tools may look attractive when viewing their (typically very long) feature list, using them for small development projects can be like buying a Learjet to enable you to cross the street. The value of these tools goes negative quickly as they are applied to smaller projects, and they become shelfware fast when everyone realizes the expense in resources just to get (what were previously manual) tasks completed. And if these projects are suffering from lack of collaboration within and between teams and users, imposing a complex tool to get that result might encourage co-operation, but don't be surprised if the cooperation is limited to distributing the pitch forks and lighting the torches.

Improving productivity in well functioning small teams in less complex projects and their environs (left side of chart) can be a big challenge. Small teams, by virtue of their small size, usually have a better chance of developing close cooperation between developers, testers and users. If that level of interaction is in place, there is probably no software project tool that will improve delivery outcome significantly to be worth the trouble implementing.

If we examine the smaller footprint SDLC management point tools (blue line), we have to generalize a bit more. These point tools have been constructed to be useful in certain lifecycle aspects (requirements management, or test coverage etc), but their value tends to roll off as project complexity increases. These management tools have been developed more recently and typically don't provide tight integration to a wide set of other point tools that more complex environments demand. In addition, many do not actively support multiple methodologies and lack flexibility in needed areas. So while the value of some of the traditional, feature rich packages can go off the chart in value because they are absolutely needed to produce any product at all, these more recent tools provide their maximum value only on smaller projects and can be even be less than useless in complex or diverse environments.

The Middle Ground

So, if large expensive and elaborate SDLC management tools find diminished value below a threshold value, and the very small projects can use manual methods (spreadsheets, Microsoft Word, sticky notes etc.) and if point tools suffer from the lack of necessary breadth of function, what is really needed (green line) for that large mid-complexity area that is found in so many IT and software product companies?

First we need to look at just what is in the marketplace to understand what types of tools are available in the marketplace and look at their *functional* approach and basic feature set. Looking at that alone will narrow the choices. A summary of the basic feature sets found for each type of is summarized Table 2.

Capability Comparison	Mid-Tier RM and SDLC	Pure RM Tools	Development Centric Tools	Product Planning Tools	Document Based Tools	Quality Tools
	Examples: RTIME	Examples: Doors, Caliber RM	Examples: Rally, Software Planner, MS Team	Examples: Feature Plan, Accept 360	Examples: Requisite Pro	Examples: HP Quality Center
Requirement elicitation, elaboration, Collaboration	●	●	◐	●	●	◐
Multi-level Requirement To Requirement Traceability	◐	●	◐	◐	◐	◐
Complete Traceability (All artifacts)	●	○	●	○	○	◐
Supports Multiple Development Methodologies (Full SDLC)	●	○	○ Rally ● Others	○	○	○
Manual Test Case Management	◐	○	◐	○	○	●
Automated Test Case Management	○	○	○	○	○	●
Planning and Analysis Tools for Decision Support	◐	○	○	●	○	○
Defect Management	◐	○	●	○	○	●
Comprehensive Reporting	●	◐	●	◐	○	○
Customizable	●	◐	●	●	◐	●
Easy to Implement/Roll-Out	●	○	●	◐	○	○
Affordable	●	○	◐	○	○	○

● = Excellent ◐ = Good ○ = Lacking




Chart 2

Capability Comparison

Beyond those fundamentals depicted in the table above, it's worth looking closely at some of the most important features and special needs for this less well served area of companies in the middle-ground.

1. Active Support for Multiple Methodologies

Development methodologies tend to vary with complexity of the environment, but complexity by no means dictates the appropriate method to use. It's quite efficient to use Agile techniques to develop simple web sites for instance. And while Waterfall techniques might be found more frequently in complex environments, many of the project components, if well isolated and defined, can be rapidly and efficiently iterated using Agile. In practice however, a majority of organizations have a combination of methodologies or hybrids in use, not only because they are in process transitioning from Waterfall to Agile, but that they have specific needs that call for a hybrid or both. The point is that Management tools *should not dictate any one particular methodology* over another. Moreover, these tools should not 'support' multiple methodologies simply by being a blank slate that requires a big configuration effort to get them set up appropriately. Rather, they should actively let you select from a library of templated methodologies that can be readily customized.

2. Integrated Requirements Lifecycle Support

SDLC management tools of any breadth for the middle ground should obviously cover the critical functions in the lifecycle. That means they should offer substantial assistance in capturing and manipulating requirements, collecting raw and refined estimates, planning functions based on estimates and available resources, task management fundamentals, and test coverage. Cutting across functions, the solution should support IM and Outlook integration for collaboration, offer extensive reporting facilities, notifications and approval workflow functions. In addition, it should be able to integrate to some of the standard ALM elements that might already be in place like popular version control and bug tracking systems.

A complete feature list matched to your needs is important, but how effective the tool is has a much to do with how well it handles tricky *transitions* that information makes as it evolves through the lifecycle. Look at how the tool handles:

- Textual Requirements to/from Graphic Requirement Depictions
- Graphic or Text based Requirements to/from Structured Database Requirements
- ROM Requirement Estimates to Refined Task Estimates
- Requirement Decomposition/Association to Tasks

3. Data Model and Orientation

SDLC management tools can have very different orientations that directly influence the traceability offered and the way data transitions are handled. The orientation depends primarily on the data model behind the tool.

Task oriented models assume that task activity is the key data element. These tools deliver value to development teams, but they can shortchange business analysts and other stakeholders. They can force users to treat requirements as tasks without much assistance with requirement vetting, approval or prioritization and can be difficult in requirement-to-task linkage.

Some tools centered on testing allow you to associate requirements to tasks, but these tools isolate the requirement and testing information from the project planning and task information. That can easily lead to missed dates or disconnects when changes occur to requirements.

Requirement centric tools do the best job holding scope creep in check because estimation, traceability, change impacts and other downstream functions are always related back to requirements rather than a subset of lifecycle activity.

4. Traceability

From both a pure engineering standpoint as well as a project management standpoint, traceability capabilities are essential to being able to examine requirement, task and other attribute interdependencies and to determine the impact of changes. Traceability can be crucial in complex project environments and invaluable in software compliance issues. Many requirement oriented point tools have traceability as a key feature. But the value of the traceability functions in these tools can vary significantly and they often are not integrated into

other mainstream lifecycle areas such as task and test management. Traceability also plays a crucial role in our next area as well.

5. Estimation, Prioritization, Reprioritization and Reporting

In all but the least complex of environments, one of the most common areas of stress, especially in commercial software product companies, is the process of correctly estimating, setting priorities, allocating resources and then, inevitably, changing and re-allocating resources against changed business priorities. Too often the information needed to make these decisions is difficult to collect and synthesize.

While *Agile development* methodologies are helping companies react faster in their development organizations, it doesn't do much to allow companies to *manage* their projects in a more agile way. The evidence is everywhere- there are too many overworked Product and Project Managers spending countless hours collecting status information, crunching numbers and then preparing PowerPoint presentations for fateful meetings. Common traceability capabilities are often held out as the helpful aid and they certainly have value, but more is needed.

The problem is that requirement management tools and the traceability they provide often don't do much more than point out what modules are impacted when things change. Managers need more assistance. They need useful tools that help them:

- Collect status and completion levels of tasks and tests
- Load ROM and refined estimates and costs into trade-off analysis tools
- See Project Progress in other views (Gantt Charts etc)
- Gather and track priorities based on business objectives
- Generate meaningful reports on the above

Indeed, the fate of many products and projects, (to say nothing of the career paths of their managers!), have been determined by the clarity and relevance of reports used in management decision meetings. Any traceability function worth its salt is going to provide impact information and trade-off analysis tools that will enable managers to defend their projects priorities and secure needed resources with credible numbers that are easily developed.

6. Simplicity in Structure and Implementation

When it comes to this area, it doesn't really matter which type of solution we are talking about. The number of databases that underpin and are required to support an SDLC management solution is one of the key determinants of the solution's complexity and ease of implementation. There is never really a good reason why any solution that claims to cut a wide path of capability should have any more than one database. If it's going to have to interface to existing ALM components or SDLC management point tools already in place, the openness and level of integration to those external databases becomes key.

7. Services and Best Practices

Be aware that some vendors will not readily assist in making recommendations about how to best apply their solution. That's because some SDLC management or requirements management solutions are so flexible and configurable that there is a risk of configuring a requirements database, or a workflow or a screen design or some other element in a sub-optimal way. The poor choice may not reveal itself right away, and in cases where a customer has already committed lots of data and training to a system that has to change in a significant way, late discovery of these problems can lead to buyer remorse. Consequently, many vendors will train you on the product's use but not offer much in the way of best practices. Effective suppliers of tools that suit companies in the middle ground need to either be prescriptive within the tool by providing templates and seed data, or supply best practices consulting for their customers.

Look carefully at the amount of training, course requirements and costs. It will reveal much about the relative complexity of their offering. Ask vendors if they offer services to completely configure and optimize their tools to your specific needs.

8. A Special Case for Ease of Use

Think about how high the bar is set for software management tools with regard to ease-of-use. These tools ask overworked teams of managers, project leaders, business analysts, developers, testers, and even user representatives to contribute to a database that is being built that contains information about the status of their work. The software professionals involved are the very people who can quickly spot a poorly designed user interface, will find anemic response time not just depressing, but offensive, and will rise up in protest when a tool bogs them down with irrelevant data collection. The adoption success and value of these solutions hinges, in an overriding way, on the *willingness* of these contributors to enter information.

Developing that willingness is not just a matter of clean design, fast response time and ease of navigation. Support for interaction with familiar tools like Microsoft Word, Outlook, Excel and Project lead to adoption. Fast, successful adoption requires the ability for contributors to get value from the system for their contributions. They need to feel and know that tighter collaboration is allowing them to work less. A valuable tool will allow them to be less stressed because they can see the work of teammates and plan their work accordingly. It will allow them to take corrective actions when things are slipping and to feel more confident that managers have made good decisions based on objective data instead of stilted perceptions.

Ease-of-Use issues in for companies in the middle-ground area can be more demanding than what is needed in complex environments where contributors may spend a fair amount of time working with a tool to examine and enter data on a complex project's intricacies. In the middle-ground, many contributors may only be occasional users that may have to waste time to rediscover hidden features that are buried in a complex tool.

Conclusion

Tools that assist IT and Software product companies manage the software development lifecycle have evolved from the extension of fundamental software development tools like compilers and source and version control systems, to the very comprehensive, well evolved ALM and Requirement Management systems offered today. These tools are elaborate and densely complex products that require investment, commitment and discipline to

use. Unfortunately, they can have net negative ROI when misapplied to some of the more typical IT and product software undertakings we see today.

In today's intense business environment, resource scarcity and the move to Agile has spawned some innovation and reinvention from tool vendors. While these new and updated tools can provide value in environments where newer methodologies are introduced, they have a narrower band of functionality that prevents them from being useful for complex environments where mixed methodologies are likely in use and where more complex work products are produced by a multitude of teams.

The majority of software development activity today falls into an area of project complexity that is not especially well served by the more traditional elaborate tools or by the newer point tools. This market will be better accommodated as the newer tools expand, and as the traditional tools work towards the middle, but this will take time. Those seeking SDLC management solutions for projects with the characteristics of the typical mid-level complexity should examine the few tools available today that are