

SUN AIM FOR SOFTWARE METHODOLOGY BRINGING REPEATABILITY TO ENTERPRISE SOFTWARE DEVELOPMENT AND DELIVERY

White Paper
August 2009

Abstract

This white paper describes an enterprise software development and delivery methodology developed by Sun Professional Services (SunPSSM) personnel for use during customer engagements. Readers will learn how this proven methodology can be used to help increase software quality, reduce cost, and improve time to market for software projects. It is intended to be used by Sun Microsystems personnel, SunPS customers, and other enterprise developers working on software projects independently of Sun.

Table of Contents

Chapter 1—Executive Summary	3
Business Benefits of the AIM for Software Methodology	3
Chapter 2—History of the AIM for Software Methodology	4
Chapter 3—Common Challenges and Pitfalls	6
Failure to Adequately Define Requirements.....	6
Considering Application Architecture as an Afterthought	6
Lack of Comprehensive Early Testing	7
Old School Project Management.....	8
Chapter 4—Principles and Strategy	9
Adequately Define Requirements	9
Develop a Unified Architecture Early	10
Test Early and Validate the Architecture Framework	10
Employ an Iterative Approach	11
The Whole is Greater than the Sum of the Parts	11
Guiding Principles of the AIM for Software Methodology	11
Chapter 5—Iterative Development and Project Organization	13
Organization of an AIM for Software Project	13
Managing an AIM for Software Project	15
Chapter 6—Conclusion	17
For More Information	18
Appendix A—Sample Quality-of-service Metrics	19

Chapter 1

Executive Summary

The AIM for Software methodology is a software project development and delivery system that can help ensure that enterprise application development initiatives reliably capture the business differentiators, while ensuring that projects are delivered on time, within budget, and with the highest quality.

AIM is an acronym for Architect, Implement, Manage, and is the name of Sun's general project engagement methodology, which provides best practices for the major stages of the project lifecycle. By bringing a similar project organization to enterprise software development projects, the AIM for Software methodology not only incorporates the general principles, practices and products of the AIM methodology, but also extends and customizes them for enterprise software development.

Developed over the past decade within the Sun Professional Services software delivery organization, the AIM for Software methodology brings together best practices from two popular industry methodologies—Unified Process and Agile methodologies. AIM for Software provides all of the elements needed for a successful project, including core foundational principles, actionable descriptions of these practices, and a delineation of their most efficient order of execution. It includes the necessary artifact templates for each step.

AIM for Software is also designed to conform to the individual needs of each development project. It offers sufficient flexibility to enable exactly the right amount of ceremony to be brought to bear, while accommodating the requirements of modern distributed computing and object oriented architectures.

Business Benefits of the AIM for Software Methodology

The customizable development process described above can dramatically elevate the success rate of software projects in the following ways:

- *Reduced time to market*—The methodology imposes a discipline that minimizes redundancy while simultaneously optimizing the order in which project steps are executed. This helps avoid rework and unnecessary steps that can prolong project delivery.
- *Lower cost*—The project team is more efficient, thus reducing costs, sometimes dramatically.
- *Increased quality*—There is a keen focus on defining all of the proper requirements up front, including both functional and quality-of-service requirements. There is also an emphasis on system organization, requiring that the architecture be designed and tested early in the project, at a time when systemic changes can still be made without losing control.

Chapter 2

History of the AIM for Software Methodology

With the advent of Web-based and distributed computing that began taking shape about a decade ago, enterprise customers began asking Sun Microsystems to assist them with designing and implementing their custom business solutions on these new platforms. This was a natural request since Sun has long been a leader in network computing, and continues to lead the development and standardization of Java™ and other network-based enterprise technologies.

When Sun Professional Services (SunPS) answered this call, it surveyed the landscape for a repeatable delivery methodology in order to drive these customer engagements. At that time, the state of the art for enterprise software development was centered on single-address-space computing. Sun adopted the Unified Process as its starting point, since that methodology had developed significant mindshare, and had distinguished itself by embracing a comprehensive, project-based approach, while at the same time, being developed from a sound, object-oriented foundation.

However, SunPS noticed significant challenges in dealing with the multi-dimensional aspects of the new domain. In the distributed applications paradigm, individual applications find their components distributed across multiple computers, and distinct applications often interact with each other in novel ways.

This situation forced typical developers to reluctantly, but necessarily, become very involved with middleware, operating systems, hardware platforms and network issues when designing application software. The entire compute stack had to be considered during software development. Also, network interactions between application modules had to be accounted for.

These areas were generally outside the comfort zones of traditional developers. They now had to educate themselves about program partitioning, network communications, partial failures, advanced levels of security and other complex concepts. They were forced to spend more time on these strange and challenging new issues than on application programming.

This all meant that the profession of application programming had changed. Practitioners were forced to become system engineers first and programmers second—while still being as good or better at programming than before.

These issues demanded a new level of software development methodology. The Unified Process had by then gained significant industry acceptance and mindshare, and was quite mature. Sun found much to like with the Unified Process, but realized that it failed to address many of these new issues.

Consequently, SunPS adopted the Unified Process, but extended its principles and practices into a software delivery methodology that could accommodate these new demands. After developing and evolving the new approach over several years, and incorporating various emerging industry developments, SunPS named the result the AIM for Software methodology.

Various principles and practices were adopted into the AIM for Software methodology from emerging industry influences. From eXtreme Programming (XP), the notions of test first and just-in-time development were modified to accommodate the general needs of enterprise and distributed computing, and then judiciously integrated. From the Agile Software movement, the idea of reducing the amount of ceremony involved in a delivery methodology were modified to be responsive to the needs of customers, resulting in the notion of appropriate ceremony—since some engagements call for more and some for less. The Open Group’s Architecture Framework (TOGAF) provided a consistent and powerful framework for project architecture development.

This now-mature process has driven the successful execution of hundreds of SunPS customer application development engagements over the past decade. All the while, it has evolved and been refined by its creators based on constant learning from real customer engagements.

Chapter 3

Common Challenges and Pitfalls

Today's evolving Internet, Web and distributed applications require significant investment in architecture design as well as an iterative execution process that enables large systems to be built from a number of smaller projects. Since today's systems are also frequently deployed in a highly heterogeneous environment using an integrated network infrastructure, the inherent complexity makes it vital to be aware of the potential pitfalls described below.

Failure to Adequately Define Requirements

Running faster than everyone else in the wrong direction is a prescription for failure. Yet that is precisely what many software development teams do if they don't thoroughly understand a project's requirements.

Defining requirements in today's landscape is challenging because project requirements are continuously developing and changing. Not only do requirements change in response to changes in the business environment, but they can also change due to increased understanding of capabilities and limitations of the infrastructure. Requirements can and will change during the course of the development life cycle.

The truth about requirements is that nobody has a very good idea of what they are in the beginning and everyone is learning about them as the project progresses. Another truth is that developers and customers generally speak different languages and frequently don't understand each other very well.

These facts combine to make the requirements specification process arguably the riskiest stage in the development of enterprise application software.

Considering Application Architecture as an Afterthought

There was a time in the past when it made sense to design and implement all of the code first, and then subsequently integrate it into a whole system. In the era of distributed, networked enterprise applications, those days are gone.

Historically, the typical approach has been to implement the required features, functions and capabilities first, in a favorite programming language, and then attempt to integrate all of the pieces afterwards. However, this approach treats the architectural aspects of the system as an afterthought, with the foundations never being properly engineered, thus compromising quality.

In today's computing environments, the project is bound to fail if the application code is developed first and then integrated with the infrastructure and other external systems afterwards. Problems become nearly impossible to isolate. There are simply too many pieces and they are all from distinct computing paradigms.

Developers must always be aware that their application software must operate on top of a complete system stack that consists of:

- Middleware including application server, database management, enterprise service bus, distributed communications, and other application infrastructure technologies
- Multiple operating systems
- Different hardware platforms
- A network comprised of heterogeneous network components

It is not reasonable to assume that one can design and implement all the code first and then expect it to actually work when subsequently integrated with all of the middleware and external systems that will inevitably be involved with the new system. Rather than simply integrating the application code with existing back-end systems, developers must heed the presence of networking gear, server and storage hardware, operating systems and application infrastructure platforms (middleware).

Lack of Comprehensive Early Testing

The need to test software early and often is widely accepted, but frequently not executed. Why is this?

SunPS has observed that midway through typical enterprise application development schedules, release milestones are often missed. When this happens, the project can begin to be perceived as difficult to manage. A certain amount of panic sets in, with concurrent desperation. There is a sense that the complexity has begun to dominate and the project has become intractable.

The inevitable response to this falling behind is to begin to cut corners. One of the first practices to slip is that of testing. It is easy to succumb to the misconception that there is not enough time to test properly, and that reducing testing will speed up development and get the project back on track. In actuality, however, it only leads to more bugs, and slows down the project considerably.

It may seem counterintuitive, but the failure to define the architecture first and then develop and qualify an *architecture prototype* is often the underlying cause for the project becoming unmanageable. The probability of success can be dramatically improved by building out the basic components and their systemic relationships prior to starting to develop the applications features. At this early stage, the basic architecture of the system is exposed in a simpler configuration compared to later in the project, making it more manageable.

While it may seem that initial architecture development and prototyping are extra steps, if the architecture is not validated until all of the code is written, the cost and time involved to make modifications later can increase almost exponentially because of ripple effects throughout the code.

Old School Project Management

Since the inception of the art of software development, it has been considered good practice to design a software application completely before developing the code for it. The same was held for completing the code before beginning testing. (This is referred to as *waterfall development*.)

However, as software began to get very complex, this approach began to fail. It simply became too unwieldy to attempt to complete all project requirements for each stage prior to starting the next. The next approach that was attempted was *cyclic development*. In this approach, a cycle of design-code-test was repeated—each cycle attempting a small portion of the required functionality—until the entire project was completed. This approach worked well until the advent of distributed computing, when it too began to fail.

The cyclic approach failed to realize that earlier cycles should emphasize certain activities, whereas later cycles should emphasize others. The Unified Process remedied this by popularizing a third approach called *iterations*. Iterations represent a sequence of mini-projects, each of which successively develops a manageable portion of the required functionality.

Maintaining a discipline of iterative development is not simple. It requires an unlikely combination of the discipline of pre-planning and the ability to respond flexibly as the project progresses.

Chapter 4

Principles and Strategy

The AIM for Software methodology is designed to directly address the challenges of today's modern enterprise software development. The sections below articulate the process while giving an overview of the guiding principles and overarching strategy.

Adequately Define Requirements

SunPS has found that more time is wasted, more money lost, more opportunities missed, and more quality sacrificed through misunderstanding about requirements than from any other software development issue. Having a tightly defined requirements process is a powerful advantage.

Software development methodology must anticipate the eventualities inherent in today's development environments right from the early stages. Otherwise, the project team will eventually have to wrestle with them anyway, causing the project to lose valuable time and incur unnecessary cost.

Capturing and having a mutual understanding of the requirements is the guiding principle of the Requirements Process in the AIM for Software methodology. It offers a four-pronged approach:

- Define the problem first.
- Enable the customer to articulate their needs and issues.
- Translate the customer's wishes into a form that is buildable.
- Enable the requirements to evolve in a controlled way throughout the project lifecycle.

Manage both functional and quality-of-service requirements

One key differentiator of the AIM for Software Requirements Process is that it specifically defines quality-of-service requirements in addition to functional requirements. There are many components to quality, such as application availability, acceptable response times, security, maintainability, and serviceability (see Appendix A for a sample list).

By defining metrics for these quality-of-service items up front, project stakeholders are forced to make conscious decisions about the tradeoffs between cost and quality. Rather than asking the project team to build the highest level of quality possible and risk a budget overrun, project stakeholders can decide which areas of quality are most important and define metrics for the desired level of quality. For example, a user response time metric might be defined to require that 90% of all transactions

be completed in less than two seconds. This tells the project team that the system must be built for two-second response times under normal loads, and that there is room for slower response times under heavy loads as long as they occur somewhat infrequently.

Develop a Unified Architecture Early

In the age of networked applications, the system architectural aspects are generally far more complicated than are the functions and features. Thus, these aspects should be considered first, and their risks mitigated early. The functions and features can be added later on top of this architecture.

The AIM for Software methodology recommends an approach of architecting first and building out the functions and features later. Developing an architecture for the system and committing it to documentation before starting to implement is a significant step toward risk reduction and success.

In addition to mitigating the highest technical risks, this approach also tames the implementation process by reducing the amount of architecture-on-the-fly and implementation rework. All of this improves project manageability, time-to-delivery and quality of the deliverables.

The AIM for Software Architecture Process actually consists of multiple steps along with a set of supporting artifacts:

- Identifying the layers of the stack.
- Selecting the best technology for each layer.
- Identifying the components needed at each layer.
- Identifying the dynamic patterns of component relationships at each layer.

The Architecture Process then documents the results of this exercise in the *Architecture Specification*, which is used as a blueprint for the system builders.

Test Early and Validate the Architecture Framework

The methodology includes development of an architecture prototype to test the most complex and risky technology issues in the early stages of the project. The prototype can illustrate how all of the coarse-grained sub-systems fit together before designing and implementing the application code. It presents the opportunity to thoroughly test out the foundations of the system from end-to-end before complicating it with feature sets.

The architecture prototype consists of the major component types from all levels of the stack that have been identified in the *architecture specification* document. The prototype connects all of these components, within and between stack layers as well as across distributed nodes, in order to test out the configurations, patterns and micro-architectures that are identified in the Architecture Specification.

Generally, the architecture prototype models the system architecture from end-to-end — from front-to-back and all the way up and down the stack. The prototype thus forms a foundation upon which the functionality of the system can be erected during the Construction Phase of the project.

When it has been built, the prototype should be subjected to intensive quality tests to help ensure that the prototype is robust. This helps verify the nonfunctional qualities such as reliability, availability, scalability, flexibility, throughput and responsiveness.

The combined result of these practices is a reduction in time-to-delivery, cost of development, and total cost of ownership as well as an improved level of quality. This is counter-intuitive to those who are in a rush to begin implementation quickly, but the process is far more effective.

Employ an Iterative Approach

Since the advent of the Unified Process, the need to employ an iterative approach to enterprise application development has been widely accepted. Enterprise projects are too complex to develop without it. However, the successful execution of the iterative approach has been infrequent.

What the iterative approach added to previous methods was the notion that different activities needed to be given varying emphasis depending on the stage of the project lifecycle in which they appeared. These sets of activities with common emphasis profiles were called phases. The following chapter provides further clarification on how phases are managed in the AIM for Software methodology.

The Whole is Greater than the Sum of the Parts

All of these processes executing together — requirements, architecture, quality and implementation — combine to result in a disciplined execution in a way that cannot be explained by the success of any one of them. The net result is an on time, within budget, high quality implementation and deployment of enterprise software.

Guiding Principles of the AIM for Software Methodology

The methodology provides principles that drive prescriptions (activity specifications), which in turn result in specific work products that can be delivered according to a schedule.

A description of the foundational elements and an example of each are provided in Table 1.

Table 1. Foundational elements of the AIM for Software methodology.

AIM for Software Foundational Element	Description
Principles	<p>Principles are the concepts and understanding that explain the foundations of project operation, and guide successful software engineering and delivery. They provide the framework for why best practices minimize time-to-delivery and cost, and maximize quality.</p> <p>Example: The primary goal of the AIM for Software Architecture Process is to decompose the system requirements into a collection of high-level service descriptions and their organizing relationships. These services are then re-composed into an integrated whole by using various techniques such as architecture patterns and micro-architectures to synthesize the whole system under development.</p>
Prescriptions	<p>Prescriptions are the best practices and step-by-step procedures that guide the delivery of successful customer engagements.</p> <p>Example: The AIM for Software Requirements Process derives a Domain Model from the Business and Technical Requirements artifacts and delivers a format that emphasizes entities and their relationships. The result therefore has the look of an architecture even though it restricts itself to business issues.</p>
Products	<p>Products are the artifact templates and workflows that enable and promote timely and successful delivery of work products to customers.</p> <p>Example: The AIM for Software Project Management Process contains the following products: Phase Plan, Iteration Plan, Project Plan.</p> <p>Example: The AIM for Software Architecture Process contains the Architecture Specification and the Architecture Prototype.</p>

Chapter 5

Iterative Development and Project Organization

Organization of an AIM for Software Project

In order to manage the complexity of networked enterprise applications, a software delivery methodology must enable project phases to be partitioned into small, tractable and manageable sub-projects.

AIM for Software methodology organizes a project via two dimensions: phases and processes. The *phases* define the project lifecycle, while the *processes* define the categories of activities. These two dimensions provide the organizing principle for *iterations*, the basic unit of management in an AIM for Software project.

An iteration can be viewed as a mini-project that establishes a limited but manageable set of functionality to be accomplished. To help ensure this, each iteration must consist of some activities from each of several activity categories, or processes, including requirements, architecture, implementation, testing, etc.

At the same time, however, the entire project has an overall flow of activity types that tend to be heavier in one activity type at one phase of the project, and to be heavier in other activity types during other phases of the project.

For example, in the middle phase of most projects, iterations will be most heavily involved in coding and testing of new features. Thus, these iterations will emphasize the implementation-type activities. However, they will also involve reviews of the requirements, revalidation of the architecture and a lot of testing activity.

Thus, the organizational scheme of the AIM for Software methodology must produce a way of organizing a project's timeline so that it characterizes how each iteration is composed of activities of various types, and how the various iterations of a project flow sequentially across the project time line.

The AIM for Software methodology does this by defining a small number of activity types (called *processes*), and partitioning the time line of a project into five *phases*, each of which is characterized by a particular emphasis on certain processes over others. These five phases constitute the AIM for Software project *lifecycle*.

In other words, one could say that each of the five AIM for Software phases exhibits a *process profile* that shows which processes are emphasized by all of the iterations that are executed within that phase. Of the five phases, four are inherited from the Unified Process, which is the methodology that has been standardized by the Object Management Group. The first phase (Conception) however, was added to account for activities that occur prior to project initiation.

As an example, Figure 1 depicts how seven *processes* (activity categories) are distributed across a lifecycle of five *phases*. The five phases are further partitioned into some number of iterations in order to optimize project manageability and predictability.

AIM for Software Project Organization

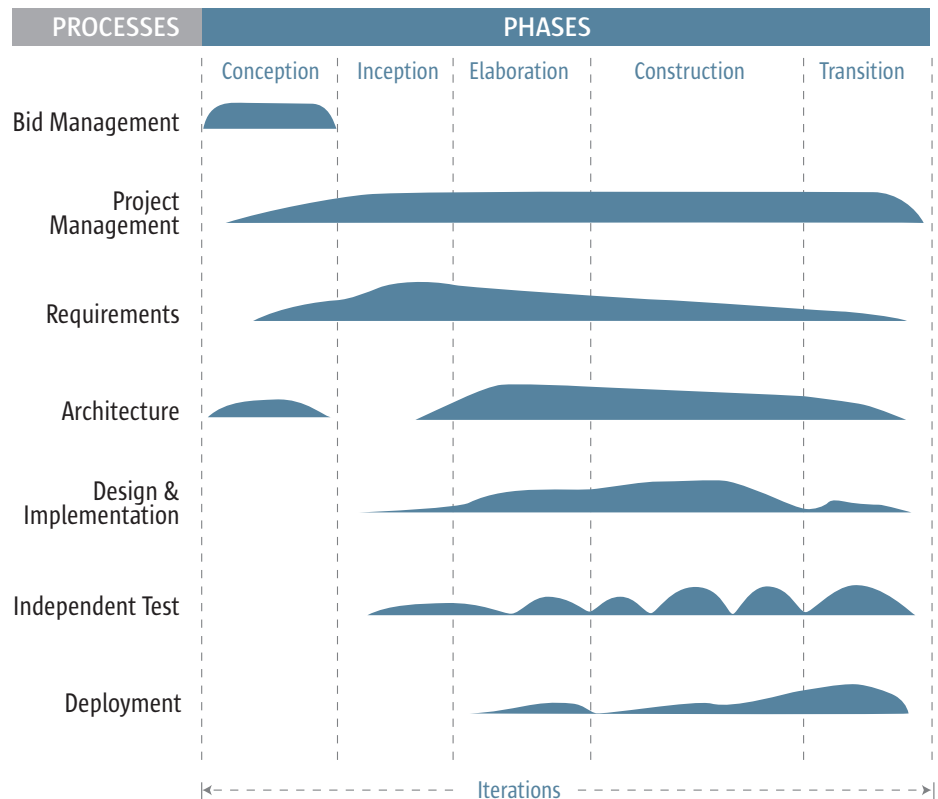


Figure 1. AIM for software processes are distributed across five phases.

One can deduce several conclusions about all AIM for Software projects from this diagram. For example, the Conception phase emphasizes activities that are executed prior to the beginning of a delivery project, since it includes bid management, project planning, requirements gathering and a bit of architecture. The Inception phase is mostly about requirements definition, since that is the only process that dominates the iterations of that phase. Of course, there is also project management activity, but that is true for all the phases. Elaboration seems dominated by architecture activities, with a generous portion of requirements as well. Other generalizations can be abstracted from the diagram regarding the Construction and the Transition phases.

Managing an AIM for Software Project

The AIM for Software project organization falls within the overall AIM engagement methodology used at Sun Microsystems. While the purview of AIM for Software concentrates on the delivery cycles of an enterprise software development project, the AIM methodology defines a customer engagement all the way from the sales pipeline through to the maintenance of production cycles and customer management responsibilities. As such, AIM is more inclusive than AIM for Software, and can be applied to project types other than enterprise software development and delivery.

Figure 2 illustrates such an example.

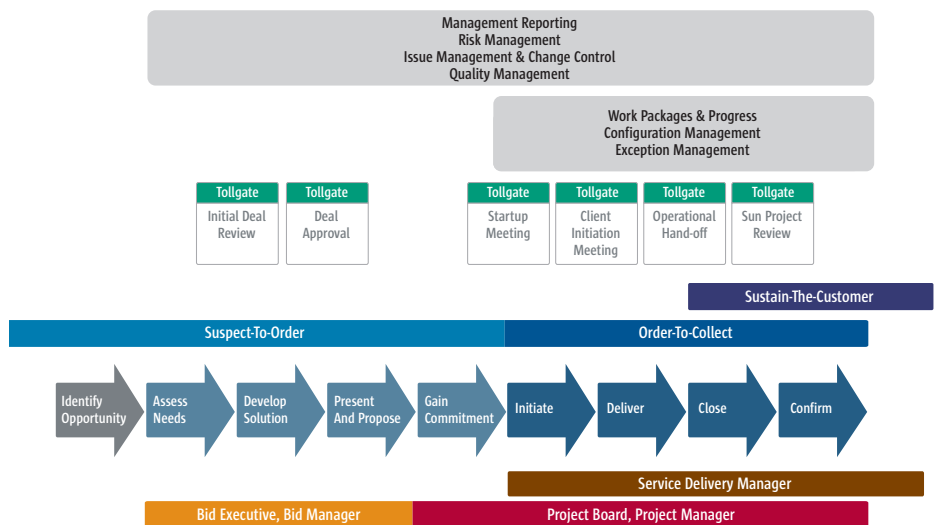


Figure 2. AIM for software fits within the broader Sun AIM Methodology

AIM for Software concentrates on AIM’s project-oriented phases and tailors them for enterprise software development. The AIM phases are Assess Needs, Initiate and Deliver phases. AIM for Software overrides these three AIM phases and replaces them with the AIM for Software phases as described in Table 2 below.

Table 2. AIM methodology versus AIM for Software methodology phase mapping.

AIM Lifecycle Phases	AIM for Software Lifecycle Phases
Identify Opportunity	<Inherited from AIM>
Assess Needs	Conception
Develop Solution	<Inherited from AIM>
Gain Commitment	<Inherited from AIM>
Initiate	Conception
Deliver	Inception
	Elaboration
	Construction
	Transition
Close	<Inherited from AIM>
Confirm	<Inherited from AIM>

Thus, the relationship between AIM and AIM for Software can be summed up by the following three statements:

- AIM defines the complete lifecycle methodology for managing a customer engagement from the earliest pre-sales activities to the long-term management of relationship with the customer.
- AIM is defined broadly, so that it can be applied to any type of engagement: hardware, software, data center, managed services, software development, etc.
- AIM for Software extends AIM in order to provide a specialization of AIM for enterprise software development projects. Rather than having the purview of an engagement lifecycle, AIM for Software is project-oriented.

AIM for Software, therefore, is implemented by working within the AIM lifecycle definition. It inherits most of the AIM phases, and overrides those AIM phases that pertain to project lifecycles.

Chapter 6

Conclusion

The AIM for Software methodology provides a repeatable development and delivery methodology that has been proven to deliver enterprise software projects on time, within budget and with the required quality. It has evolved over the past decade while being used in hundreds of customer implementations. The methodology tackles the shortcomings of undisciplined enterprise development habits by providing the operating principles, best practices, work products, and development and delivery protocols that help maximize the success of enterprise software development projects.

The success of AIM for Software stems from the way that it re-engineers the three aspects of enterprise software development described below.

Requirements

AIM for Software recognizes the risks and complications inherent in developing requirements specifications for enterprise software. The articulation of requirements from the business perspective is necessarily different from how it must be conveyed to system builders. In addition, the requirements process must be flexible enough to allow the requirements to be adapted to changes in the business environment as well as to lessons learned during the progress of the project. The AIM for Software methodology provides the principles and practices to navigate this challenging but critical process.

Architecture

Developing a desktop or mainframe application was mostly a matter of application software design and programming. However, designing and developing a networked enterprise application involves selecting and assembling a number of technology components and implementing a complete system stack that consists of network, hardware and software infrastructure components. This flexible stack enables application code to be easily distributed across the network. Because this environment is more complex than traditional architectures, it requires a more advanced skill set than application programming alone. AIM for Software provides the concepts, practices and work products to master the challenges of distributed computing architectures. While the optimal approaches often appear counterintuitive (e.g. architecture first, application code later), these practices have been proven over hundreds of successful projects to avoid wasteful dead-ends, reduce time-to-delivery, and optimize the quality of the delivered software.

Quality

Quality is not provided by testing, it can only be proven by testing. Therefore it must be considered from the very beginning in order to be realized in the final product. Good quality begins with good quality-of-service requirements. Consequently, AIM for Software provides a mechanism for capturing and defining such requirements in addition to functional specifications. Quality is designed into a system through architecture.

AIM for Software provides an architectural process whose criteria help ensure that the quality-of-service requirements are realizable. It also provides support for implementation practices that build them into the system. A testing strategy and testing plan process help in continuous and final verification. Quality is ensured by a coordinated discipline of requirements, architecture, implementation and testing.

These three aspects of AIM for Software result in a number of other advantages.

For example, it:

- Leverages industry-trusted methodology frameworks, including the Unified Process, Agile Methodologies and TOGAF.
- Enables right-timing and right-effort on project components
- Provides the correct emphasis on requirements management as well as application and system architecture
- Enables an integrate-early approach
- Incorporates proven best practices based on hundreds of successful enterprise Java application development engagements and more than a decade of Sun experience building enterprise-class applications.
- Provides disciplined structure to help avoid common pitfalls in Requirements Process, Architecture Process, and Independent Test Process, thus helping to mitigate risks and ensure project success.
- Enables an iterative and incremental approach to project organization and management by organizing projects along two dimensions: Lifecycle (Phases) and Activity type (Processes).

For More Information

For additional information on Sun Professional Services offerings and the AIM methodology, visit the Web sites below or contact a local Sun representative.

Table 3. Web links for additional information.

Web Site URL	Description
sun.com/service	Sun Services home page
sun.com/service/professionalservices/	Sun Professional Services
sun.com/service/findanswers/virtualization/aimmethodology.pdf	Sun Professional Services AIM methodology overview

Appendix A

Sample Quality-of-service Metrics

The Sun AIM for Software methodology addresses quality requirements by defining metrics for specific quality of service elements. Since the types of quality requirements can vary greatly from one project to another, the list of quality-of-service metrics must be flexible. Table 3 shows a list of metrics that Sun uses as a starting point for collecting quality requirements during the Requirements Process. This list is customized for each individual project and a measurable objective is defined for each quality-of-service aspect that is included as a requirement.

Table 4. Sample quality-of-service metrics.

Quality-of-service Element	Description
Availability	The percentage of time the system is able to provide service to its users.
Responsiveness	The amount of time users waits for the system to respond to their inputs.
Throughput	The number of users the system can support at specific thresholds.
Scalability	The ability of the system to sustain anticipated future system demands beyond the current release with minimal impact.
Security	The system's ability to protect the system and its data from unwanted access.
Correctness	How much the output produced by the system can be trusted to be accurate.
Flexibility	How easy it is to add or change capabilities without breaking existing capabilities, architecture, or quality.
Serviceability	How easily the system can be upgraded or fixed by measuring the downtime required to do so.
Extensibility	What the ability is to make significant enhancements or changes easily.
Maintainability	The degree to which faults can be detected, diagnosed, and serviced.
Manageability	The ability to easily start, stop, restart, and monitor the system. Also refers to the ability to control or organize the system.
Usability	How easily the user can learn and operate the user interface.
Reusability	The ability to allow components of the current system to be incorporated into other systems.
Buildability (And Customizability)	The amount of confidence that the system can be built within a given timeframe.
Frugality	A measure of whether there are unnecessary expenditures to build, maintain or extend the system.
Timeliness-to-Market	Requires the least time to design, implement, test and deploy and still meet specified requirements.



Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN (9786) Web sun.com

©2009 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, SunPS, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the United States and other countries. Information subject to change without notice. Printed in USA 09/09