



THE ASSISTANT SECRETARY OF THE NAVY  
(Research, Development and Acquisition)  
WASHINGTON DC 20350-1000  
JUL 10 2007

*F - FILE*  
*COPY TO ALL MP's*  
*TO Y CHIEF ARCHITECT*

MEMORANDUM FOR DISTRIBUTION

SUBJECT: Software Process Improvement Initiative (SPII) Software Development Techniques (SWDT) "Software Development Techniques Phase 1 Report"

Under my direction, the SPII is rapidly identifying Navy policy and practices regarding the acquisition and management of software-intensive systems. To this end, I directed a review to survey existing practices, identify issues, and develop recommendations and guidance for improved software development. The attached report provides an overview of existing software development techniques and suggestions for evaluating emerging software development techniques.

A handwritten signature in black ink that reads "Delores M. Etter".

Delores M. Etter

Attachment:  
As Stated

SUBJECT: Software Process Improvement Initiative (SPII) Software  
Development Techniques (SWDT) "Software Development Techniques Phase 1  
Report"

Distribution:

COMNAVSEASYS  
COMNAVAIRSYS  
COMSPAWARSYS  
MARCORSYS  
PEO JSF  
PEO T  
PEO A  
PEO W  
PEO SHIPS  
PEO SUBS  
PEO CARRIERS  
PEO IWS  
PEO EIS  
PEO C41  
PEO LMW  
PEO LS  
PEO SPACE  
DRPM SSP  
ASN (RD&A) CHENG

Copy to:

DASN ACQ MGT  
DASN AIR  
DASN C4I/SPACE  
DASN IWS  
DASN EXW  
DASN LOG  
DASN M&B  
DASN RDT&E  
DASN SHIPS  
NAVY IPO  
DACM  
NAVSEA (05,06)  
NAVAIR (4.0)  
SPAWAR (05)  
COMNAVFACSYS  
COMNAVSUPSYS

**Software Process Improvement Initiative (SPII)**  
**Software Development Techniques (SWDT) Focus Group**  
**Phase 1 Report**

**10 April 2007**



**Prepared by**

**Assistant Secretary of the Navy For  
Research, Development, and Acquisition (ASN/RDA) Chief Systems  
Engineer's (CHSENG) Software Development Techniques Focus Group**

**For ASN/RDA CHSENG**

**Version 2.0**



## Document Revision History

### DOCUMENT REVISION HISTORY

Version	Issue Date	Author/Modifier	Section, Page(s) and Text Revised
1.0	12/13/06	Dr. Clifton Phillips, et-al	Original document
1.1	02/14/07	Dr. Clifton Phillips, et-al	Included adjudicated comments as documented.
1.2	03/14/07	Dr. Clifton Phillips, et-al	Incorporated adjudicated comment from additional table not included in first wave of comments. Included recommended changes from SPII teams, added revision history page, and revised/organized references.
2.0	04/10/07	Dr. Clifton Phillips, Candace Conwell, et-al	Incorporated adjudicated comments from PEOIWS (CAPT. Shannon) and condensed document as suggested by SPAWAR (Ms. Michelle Bailey).

**SPII SWDT Phase 1 Report Table of Contents**

<b>EXECUTIVE SUMMARY</b> .....	<b>V</b>
<b>1 BACKGROUND</b> .....	<b>1</b>
1.1 GOAL AND OBJECTIVES .....	1
1.2 APPROACH AND FINDINGS .....	2
1.2.1 <i>Definition</i> .....	2
1.2.2 <i>Existing Techniques</i> .....	2
1.2.3 <i>Impact of Development Techniques on Software Performance</i> .....	2
1.2.4 <i>Potential for Improvement</i> .....	<i>Error! Bookmark not defined.</i>
1.2.5 <i>Existing Guidance</i> .....	3
1.2.6 <i>Software Reviews</i> .....	<i>Error! Bookmark not defined.</i>
1.2.7 <i>Recommendations</i> .....	3
<b>2 UNDERSTANDING EXISTING SOFTWARE DEVELOPMENT TECHNIQUES</b> .....	<b>5</b>
2.1 PREDICTIVE SOFTWARE TECHNIQUES .....	5
2.2 ADAPTIVE SOFTWARE DEVELOPMENT TECHNIQUES .....	5
<b>3 A FRAMEWORK FOR EVALUATING EMERGING SOFTWARE TECHNIQUES</b> .....	<b>7</b>
3.1 EMERGING TECHNIQUES .....	7
3.2 A PROCESS FOR EVALUATING SOFTWARE DEVELOPMENT TOOLS .....	9
3.2.1 <i>Acquirer Responsibility in New Programs</i> .....	9
3.2.2 <i>Acquirer Responsibility in Existing Programs</i> .....	10
<b>APPENDIX A: PREDICTIVE TECHNIQUES FOR PROGRAMS WITH WELL-UNDERSTOOD REQUIREMENTS</b> .....	<b>11</b>
<b>APPENDIX B: EVALUATING TOOL TYPES</b> .....	<b>18</b>
1 CONFIGURATION MANAGEMENT TOOLS .....	18
2 REQUIREMENTS DEFINITION AND MANAGEMENT TOOLS .....	19
3 SYSTEM ANALYSIS AND MODELING TOOLS .....	20
4 DEVELOPMENT TOOLS .....	22
5 DEFECT TRACKING TOOLS .....	23
6 SOURCE CODE (SECURITY) ANALYSIS TOOLS .....	24
7 TESTING TOOLS .....	25
<b>REFERENCES:</b> .....	<b>28</b>



SPII SWDT Phase 1 Report List of Figures

Figure 3-1-1. The process model output is the weighted sum off all the inputs..... 9  
Figure B-7-1. Test planning is an integral part of the V process..... 26

SPII SWDT Phase 1 Report List of Tables

*Acknowledgements* ..... vi  
*Table 3-1-1. Key features for emergent software development technique assessment process.* ..... 7  
*Table A-1. Is a representation of comparability for some popular existing predictive software development techniques.* ..... 11  
*Table A-2. Is a representation of comparability for some popular existing adaptive software development techniques.* ..... 14



## Executive Summary

The Department of Defense is in the midst of a network centric transformation<sup>1</sup>. The transformation driven by warfighting dominance is dependent in part upon being better equipped in the "information age." One challenge occurring in parallel with the transformation from industrial dominance to information dominance is that software has overtaken hardware in terms of programmatic risks and dollars spent on the acquisition of defense systems. The Navy is reviewing the ways software is acquired within its organization. The goals are to identify improvements that can be realized soon, ensure the guiding policies are not out of date, and incentivize the migration toward successful software practices.

This document serves primarily the acquisition community with the goal to contribute to the body of work that makes government a "smart buyer". The secondary goal is to provide information to the software development community that provides the government with software intensive systems. Several important factors are mentioned herein with the goal of establishing mature software development techniques.

This product provides the findings and recommendations of the Software Development Techniques focus group, which has been lead by PEO C4I and supported by experts in acquisition and software engineering from across all the Program Executive Offices, Systems Commands, and supporting System Center laboratories. The document is divided into three main areas which include a general background section, a section dedicated to understanding existing software development techniques and a section on how to evaluate emerging software development techniques. Additionally there are two appendices which are meant to help guide the acquisition professional on the use and evaluation of the software development techniques and tools utilized for software intensive systems.



## Acknowledgements

Contributions to this work came from across the Naval Enterprise. The following individuals provided deep insight into the problem being addressed here, and contributed to the solutions discussed herein.

Name	Command
Dr. Clifton Phillips	PEO Command Control Communications, Computers and Intelligence (C4I)/Space and Naval Warfare Systems Center San Diego
Lee Rogers	PEO C4I/PMW 150 (Command and Control)
Candace Conwell	PEO C4I/PMW 750 (Carrier Integration)
Melissa Chamberlin	PEO C4I/PMW 750/Indus Technologies (Carrier Integration)
Marsha Ark	Space and Naval Warfare Systems Command /Booz Allen Hamilton
Lynne Spruill	PEO Command Control Communications, Computers and Intelligence (C4I)/MITRE
Judy Froscher	Office of Naval Research
Dr. Bill Bail	Naval Sea Systems Command /MITRE
Brian Groarke	Space and Naval Warfare Systems Center San Diego
Jeff Schwalb	Naval Air Systems Command
Claire Velicer	Naval Air Systems Command
Al Kannis	Naval Air Systems Command
Mike Spencer	PEO C4I/PMW 150 (Command and Control)
Greg Settelmayer	PEO Command Control Communications, Computers and Intelligence
Jose L. Martinez	PEO C4I/PMW 150/Indus Technologies
Michelle Bailey	Space and Naval Warfare Systems Command

Additionally the following personnel contributed significant effort in reviewing this document and providing valuable feedback to the authors.

Name	Command
Peter Lierni	DUSD(A&T), SSE/SSA
Robert D. Buckstad	DUSD(A&T), SSE, SSA
Christine Hines	HPTi
Dr. Carl Clavadetscher	IRM College of Natl Defense University
Jeffrey L. Dutton	Jacobs Technology
Gary L. Hafen	Lockheed Martin
Arch McKinlay	NOSSA
CAPT James Shannon	PEO IWS



## 1 Background

Assistant Secretary of the Navy for Research Development and Acquisition (ASN RDA) Dr. Delores Etter commissioned a Software Process Improvement Initiative (SPII)<sup>2</sup>. Mr. Carl Siel (ASN RDA Chief Systems Engineer) established the SPII team composed of five focus areas to identify potential process improvements. The five focus areas are: Software Acquisition Management, Software Systems Engineering, Software Development Techniques, Human Resources, and Business Implications.

The Software Development Techniques focus group was tasked to survey existing practices, identify issues, and develop recommendations and guidance for improved software development. Software continues to represent one of the highest risk areas for program schedule and life-cycle costs. As the Navy and DoD move to net-centric "publish and subscribe" capability, software must be agile yet robust, and information assurance within software must be improved.

This report provides the findings and recommendations of the Software Development Techniques focus group, which has been led by PEO C4I and supported by experts in acquisition and software engineering from across all the Program Executive Offices, Systems Commands, and supporting System Center laboratories.

### 1.1 Goal and Objectives

The Software Development Techniques focus group objective is to develop recommendations and guidance for improved assessment and/or employment of software development techniques. The specific objectives were established by the ASN RDA Chief Systems Engineer, and include:

1. Examine current techniques and apparent strengths and weaknesses;
2. Provide guidance for assessing emerging techniques; and
3. Report findings that are useful to acquirers and developers and the other SPII focus groups.

### 1.2 Approach and Findings

The Software Development Techniques focus group agreed upon a definition for "development techniques," identified and reviewed existing techniques, discussed software acquisition problems and potential relationship to development techniques, considered potential solutions, examined existing guidance, and developed additional and more specific guidance for assessing techniques. This guidance is consistent with existing policy, and is intended as an interim guide for acquisition professionals, supporting industry, labs, and academia partners. Software development techniques can have considerable impact on program schedule and product quality. Sufficient guidance exists<sup>3</sup> that describes and compares the waterfall and evolutionary approaches or development processes, but little is available for assessing utility of software



development methods and tools. These are addressed in this report and its appendices, with guidance intended to assist the acquisition professional and the developer community. With feedback, the interim material here can be refined and considered for inclusion in formal documentation and training available within the community.

### 1.2.1 Definition

The definition for "development techniques" was derived from Barry Boehm<sup>4</sup>. He describes processes as defining the order and exit criteria of the stages of development while methods and tools are applied within those stages, and often produce specific products. This definition highlights the fact that software development techniques differ from concept refinement to maintenance and a wide array of techniques (methods and tools) are available.

### 1.2.2 Existing Techniques

Techniques include processes, methods, and tools. Waterfall development is employed for well-defined, low-risk programs, and rarely does this apply to software-intensive programs. Evolutionary development includes incremental and spiral development. Incremental development is applied when requirements can be defined for one increment. Spiral development is applied when requirements are not well understood. Each of these approaches to software development includes the stages of requirements definition, design, coding, unit testing, and integration testing. The many methods and tools that support these different phases of development are summarized in Appendices A and B, noting strengths and weaknesses. Methods and techniques evolve, some rapidly, so the Appendices captures just a snapshot of today's market and practices, and are most useful in comparing suggested utility. Some techniques involve considerable cost and learning curve, and thus are most appropriate for larger programs with longer schedules. Others improve speed and accuracy but significantly increase man-hours. The benefits of these techniques must be evaluated against the program risks, constraints, cost, and schedule.

### 1.2.3 Impact of Development Techniques on Software Performance

Next the Software Development Techniques Focus team assessed the relationship of software development techniques to software acquisition and support problems. Clearly, the improper, or unskilled, application of tools and methods can lead to schedule slips, cost overruns, poor quality, and customer dissatisfaction. Some tools produce non-standard outputs and thus limit the ability to distribute products for review by subject matter experts who may be dispersed across organizations and may not have licenses for the tool. Methods and tools may incorporate automation, increasing speed to code, but also increasing the time to conduct fault isolation and implement corrections, affecting both schedule and quality.

### 1.2.4 Acquisition and Contracting Strategies

The acquisition community can be more proactive and effective in their acquisition and contracting strategies, and in their evaluation of proposed software development techniques. The Request for Proposal (RFP) and/or the Statement of Objectives (SOO)



generally include words that emphasize the importance of mature software development capability. In addition, the RFP can be constructed to request a contractor proposed Statement of Work (for the entire or for a designated section of actual or scenario-based work) to include software development methods and tools. The offerors should be told in Sections L and M of the RFP that their responses (including the proposed Statements of Work) will be evaluated for appropriateness of software development techniques given the program characteristics and the offeror's experience and past performance. Criteria can also be developed and presented in Sections L and M for evaluation of the proposed techniques' support for open architecture objectives. Acquirers can also request that offerors describe what metrics will be supported by the proposed techniques and tools and how those metrics can be used by the developer and even the acquirer to monitor risk and progress.

### 1.2.5 Existing Guidance

Very little formal guidance is found to guide acquisition professionals in evaluating proposed selection and application of specific development techniques. Several factors contribute to the lack of formal guidance in the area of development methods and tools. First, program managers often consider developmental activities and tools as solely within the domain of the developer. Additionally, it is hard to develop a core of expertise across a wide variety of methods and tools. Methods and tools evolve rapidly enough that it can be hard to develop expertise with more than a few. Furthermore, interoperability is often a problem across these tools, and suites of interoperable tools can sometimes involve prohibitively expensive investments. Another underlying reason for the dearth of formal guidance in software development techniques is, ironically, that the increasing emphasis on performance-based contracting may be mistakenly interpreted to relieve the acquirer of an initial assessment of risk (that the contractor will be able to apply the techniques successfully).

Formal guidance for Open Architecture (OA) does provide acquisition strategy and contract guidance for achieving standards-based modular components necessary for improved life cycle costs and supportability, and this guidance highlights the importance of applying appropriate software development techniques. The Open Architecture Computing Environment Design Guidance<sup>5</sup> emphasizes the need to evaluate and apply programming techniques and middleware technologies. For software-intensive Services-Oriented Architecture (SOA) based systems, important contract wording and examples are provided within the PEO C4I Net-centric Enterprise Services Initiative (NESI) guidance<sup>6</sup>. NESI provides specific guidance for providing net-centric solutions through carefully-structured contractual language within RFPs, SOO, SOW, and Contract Delivery Line items (CDRLs). NESI and OA guidance provide self-assessment tools designed to assist developers and Program Managers determine where their programs need attention to achieve net-centric and OA objectives.

### 1.2.6 Recommendations

This effort and other SPII activities will result in improved DoD and Navy guidance to ensure that software-intensive systems and development efforts are based on common



understanding of the relative merits of different types of development techniques. In the interim, this document provides acquirers and developers of software-intensive systems a summary of current development techniques and an initial characterization of appropriate use. A process is provided in Section 3 for conducting relative comparisons for suitability of different techniques, and is specifically designed to be useful in assessing emerging or new techniques. Section 3.1 provides a process for comparing methods, new and emerging, and Section 3.2 provides a process for evaluating tools.



## 2 Understanding Existing Software Development Techniques

Software development techniques that are currently popular were assessed at the time of this publication. The range of these techniques is illustrated in Appendix A, Tables A-1 and A-2. Basic information was collected for each technique. Tables A-1 and A-2 column headers cover the name of the technique, a definition, basic features, advantages and disadvantages. Table A-1 contains predictive software development techniques appropriate where requirements are understood and Table A-2 contains adaptive software development techniques appropriate when requirements are not initially well-understood.

For some programs, multiple techniques may be effectively combined. When multiple candidate development techniques are identified, other discriminatory features can be added to the tables such as performance, schedule, cost, supportability, security, and safety. These features can be used as evaluation criterion as described in the best practices section of the Data Analysis Center for Software<sup>7</sup>.

### 2.1 Predictive Software Techniques

Predictive techniques, also known as plan driven or disciplined methodologies, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive software development techniques are the best choice when requirements are mature and stable. Predictive software techniques are not recommended when requirements are poorly understood, as the software estimates that are used to establish the Acquisition Program Baseline schedule and costs will be invalid, and will ultimately cause a baseline breach.

Predictive teams have difficulty changing direction. The plan is typically optimized for the original objectives. Predictive teams will attempt to limit change by modifying the charter of their software configuration change boards to strongly filter out any but "must-have" changes.

Predictive techniques are appropriately applied in waterfall development efforts wherein the phases of requirements, design, development, test, and integration are sequential and the outcome of each phase is deterministic. Predictive techniques are not suitable for programs where requirements must be discovered or refined, including incremental development (where requirements for the next increment are not yet known), and spiral development, or any hybrid of those. Table A-1 is provided in Appendix A and describes a few popular predictive techniques.

### 2.2 Adaptive Software Development Techniques

Adaptive methods are employed where requirements are not well understood and are designed to accommodate change. These are the techniques that are often described in software journals, and are often associated with fads. These techniques are characterized by iteration and are applied in spiral, incremental, or hybrid spiral/incremental development efforts. Adaptation can occur between the iteration cycles, and the customer is often involved in determining exit criteria and objectives for the next iteration.



Incremental and spiral or hybrid approaches are also used when technology is still maturing, when funding is staggered, and/or to manage complexity. Table A-2 is provided in Appendix A and describes a few popular adaptive techniques.



### 3 A Framework for Evaluating Emerging Software Techniques

One of the objectives of this study is to provide a methodology for assessing suitability of emerging techniques, specifically software development methods and supporting tools. Section 3.1 provides a process for relative comparison of different software development methods. Section 3.2 provides a process for evaluating supporting tools.

#### 3.1 Emerging Techniques

New software development methods are frequently promoted in software and management journals. Evaluation of the effectiveness of these methods can be difficult given the lack of substantial data. A process is provided here to rate the individual features of a candidate method for purposes of developing a weighted score reflecting a single rating for use in comparing relative merits of different methods, given program and organization objectives and constraints.

Specifically, the process involves:

1. Using IEEE/EIA 12207<sup>8</sup> framework, list the salient features useful in guiding the development process to where it needs to be, and then provide an accountability mapping to the elements of the framework. The features are selected to define principal components. An example is illustrated in Table 3-1-1.
2. Attach a weighted variable to each selected feature corresponding to relative importance. During evaluation of the feature a number or a color scheme (i.e. green, amber, red) are equally effective. The identification of important features and their relative importance can be accomplished by Integrated Product Team (IPT) consisting of stake holders in the life-cycle processes. In accordance with acquisition policy the relative importance of each category is described in the RFP.
3. In a working group or integrated product team environment, synthesize detailed questions about the emergent software development technique under assessment.
4. Attach a scoring scheme to each question. Then devise a method to total the score under each salient feature.
5. Set the relative values of the weight variables using input from the working group. Some normalization factor may be required.

Table 3-1-1. Key features for emergent software development technique assessment process.

Feature	Mapping to IEEE/EIA 12207	Weighted Variable
Business Practices	Primary	A
Requirements Management	Primary	B
Assessment and Oversight	Organizational	C
Training	Organizational	D
Configuration Management	Supporting	E
Quality Assurance	Supporting	F



Questions developed by the working group or IPT will be tailored for the specific project, and may cover some or all of the areas in Table 3-1-1. Example questions for each of the areas are provided here:

1. Business practices (Business Implications)
  - a. Does the software development method support existing business practices (programmatic performance tracking, deliverables, and reviews)?
  - b. Is the new method supported with existing or mature cost models for estimation?
2. Requirements Management (Software Engineering)
  - a. Does the method support open architecture objectives including modularity and interface standards?
  - b. Is the method appropriate given the level of requirements stability?
  - c. Is the method supported with existing or affordable tools?
  - d. Does the method provide standard, portable products?
3. Assessment and Oversight (Acquisition Management)
  - a. Does the method support objectives for the current acquisition phase of the program?
  - b. Is the learning curve for the new technique and supporting tools acceptable?
  - c. Does the method support program metrics?
4. Training (Human Resources)
  - a. Are developers experienced in similar methods?
  - b. Is training available and affordable?
  - c. Will training be useful in other programs or projects within the organization?
  - d. Will training contribute to employee retention?
5. Configuration Management (Development)
  - a. Is the method compatible with configuration management tools already in place at the vendor and/or acquirer?
  - b. Are associated tools and documents available and adequately configuration managed?
6. Quality Assurance (Development)
  - a. Does the method support program review and verification strategies?
  - b. Does the method support measurable quality goals (i.e. requirements volatility, software trouble report distribution, cost and schedule variances, defect tracking, etc.)?
  - c. Does the method support defect prevention?

The final step is to sum all the output scores from each branch in the process as depicted in Figure 3-1-1. The score(s) can be used for relative comparison of different techniques. Where color scheme only is used at evaluation comparison occurs across similar features vice as a summed score. Otherwise the linear model shown accommodates an output score.

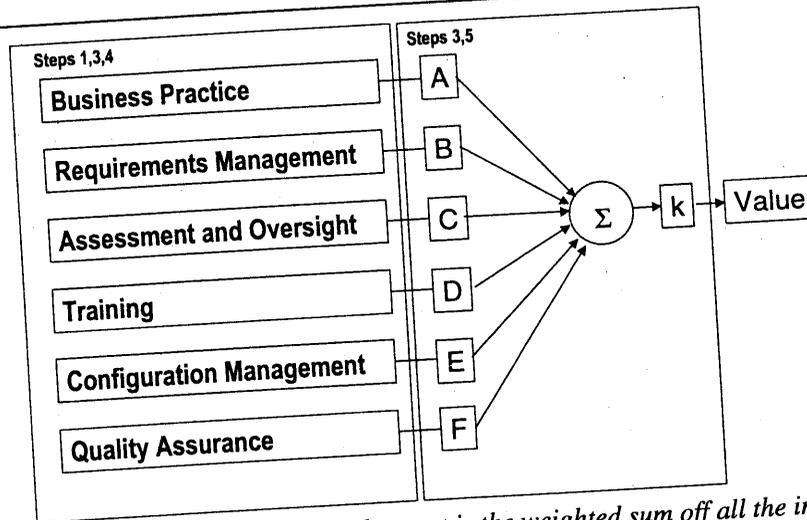


Figure 3-1-1. The process model output is the weighted sum off all the inputs.

### 3.2 A Process for Evaluating Software Development Tools

System / Software Development tools can be effectively used to support each phase of the Software Development Lifecycle (SDLC). Tools can enhance productivity, improve configuration control, reduce errors and rework, and support common "situational awareness" of program products and progress. Tools can be applied throughout the SDLC for requirements refinement, requirements tracking, design development, design walk-through, user assessment, version control, change control, shared development environments, coding, code automation, testing and test automation, training, maintenance, and data collection for progress metric and quality control analyses.

#### 3.2.1 Acquirer Responsibility in New Programs

Acquisition reform has placed the emphasis on performance-based contracting and incentives. Contractors are generally not directed to employ specific tools. Still, the acquirer will be interested in minimizing the risk that inappropriate tools will be applied. The following considerations can minimize risk when evaluating proposals.

1. **Evaluate whether the provider has achieved or demonstrated mature software processes.** CMMI Level 3 (and above) processes indicate a mature development process, and is sufficient to indicate provider capability in tool selection and application.
2. **If specific tools are to be proposed, evaluation criteria could include:**
  - a. Provider experience with the tools, for learning curve and schedule considerations.
  - b. Whether tools produce standard output formats (to minimize dependency on specific tool and to support automated data transfer between tools).
  - c. Is the license cost reasonable and supportable?
  - d. Will maintenance costs be reasonable?
  - e. Evaluate provider skills and impact of learning curve on schedule.



### **3.2.2 Acquirer Responsibility in Existing Programs**

Even when a program is well underway, the acquirer may be involved in decisions to implement new or different tools, particularly if the new tool(s) will need to interoperate or exchange data with other tools in use by the developer or within the program office. For example, if the developer decides to implement new configuration management tools, it would benefit both the Program Office and the developer if the configuration tool could import data from the requirements management tool (which may be operated by another contractor for the Program Office). Appendix B provides a detailed process for evaluating tools. Each tool type is described in terms of what, who, and why, and a short list of questions is provided under each to assist in determining suitability.



## Appendix A: Predictive Techniques for Programs with Well-Understood Requirements

Table A-1. Is a representation of comparability for some popular existing predictive software development techniques.<sup>11</sup>

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Predictive	Software development effort based upon a predictive model that was calibrated, to some extent, with actual results.	Stable requirements and similarity experience basis. Sometimes the design is already developed.	Estimates are usually based upon actual cost encountered in a previous endeavor. Does not necessarily require the expense of creative people.	Not accurate unless external conditions are known perfectly.
Sequential Coding	Sequential lines of code in a high-level procedural language like COBOL, FORTRAN, C, etc.	Program is executed sequentially throughout elements. These programs are well represented by flowcharts.	Coding techniques are mature; libraries are well developed and known. A mature software developer can facilitate reuse by adding functionality to library.	Software developed by sequential techniques are not well suited for event driven functionality applications.
Object Oriented Programming (OOP)	OOP centers on the development of small, reusable program routines (modules) that are linked together and to other objects to form a program.	Event driven objects are developed using a hierarchical class structure. Each class features parent-child relationships that facilitate reuse. Child objects understand codes inherent to their parents, plus their own unique code.	Suitable for event driven functionality. Code reuse facilitated by inheritance and upgrades to short modular code modules in model, view, or controller paradigms. Investment of usable repository promotes reuse and serious reductions in development times. Maintenance of that software will be much simpler and much less error prone as consequence of up front design investment in careful designs.	Lack of standardized coding procedures. Requires skilled developers to exploit the power of OOP. It is possible to write sequential code in an OOP program. Also useful in adaptive category when exploiting reuse.

SWDT Phase 1 Report

SWDT Phase 1 Report

Table A-1 (cont.). Is a representation of comparability for some popular existing predictive software development techniques.

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Rapid Application Development	A short time duration development technique with compressed phases of initiation, development, and implementation.	Requirements identification phase is quick and the capabilities should remain unchanged throughout the development.	Good for design or redevelopment of low-complexity applications when selected as a predictive technique. Given acceptable risk, RAD may be used within development and design phases of another technique.	Bad for complex high risk applications when used for predictive cases. However, the method is fine when selected to precede another technique.
(CASE) Tools	Computer Aided Software Engineering Tools	Provides mechanisms for automated software development. For example a graphical user interface can be developed with CASE tools. When the result is right the code is auto generated.	Can increase productivity, reduce cost, and improve product quality.	Incompatibilities among vendors. High start-up costs. Requires management patience for long-term ROI. Requires similar security coordination and disciplined configuration management as other tools (see section 2.3 for I/A concerns).
Lean SW Development	Lean Software development is a translation of lean manufacturing principles and practices to the software development domain. It can be applied both the predictive and adaptive techniques.	Lean SW development relies on a CMMI-like infrastructure for development and improvement (as so meets CMMI goals). Focus is on the core task of identifying and eliminating waste while driving key decisions outward in the development cycle.	Highly responsive to customer needs, and provides a best-in-class answer to requirements volatility issues	Often requires a facilitator to realize productive gains. Senior management support is a must.



Table A-1 (cont.). Is a representation of comparability for some popular existing predictive software development techniques.

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
MDA	Model Driven Architecture is useful to exploit design patterns of modularity and reuse at the architectural level.	<p>Models expressed in a well-defined notation.</p> <p>Organized around a set of models by imposing a series of transformations between models.</p> <p>Requires industry standards to provide openness to consumers, and fosters competition among vendors.</p> <p>Facilitates meaningful integration and transformation among models, and is the basis for automation.</p> <p>Requires industry standards to provide openness to consumers, and fosters competition among vendors.</p>	MDD elements are easy to synthesize. Inherits reusable functionality from architectural design.	Requires an up front investment strategy. Management must be patient for a ROI beyond two-year scope of government procurement cycles. Requires similar security coordination and disciplined configuration management as other tools (see section 2.3 for I/A concerns).
MDD	Model Driven Development is useful to get the correct scope on total cost of ownership early in a programs life-cycle.	<p>Useful for modeling systems and subsystems at a component level for requirements validation (i.e. to control requirements creep).</p>	<p>Up front programmatic risks mitigation. Applies to codes synthesized from MDA, and non-MDA paradigms. Sometimes supports auto code generation. Provides early feedback to the development team.</p>	<p>Can be used inappropriately if not properly configuration managed. Requires an investment strategy to get the models to the appropriate level of fidelity. Models need to be maintained and be consistent with implemented code.</p>



Table A-2. Is a representation of comparability for some popular existing adaptive software development techniques. <sup>1,12</sup>

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Adaptive	Employ incremental and /or iterative cycles intended to converge upon meeting capability requirements.	Develop Large and complex systems via incremental and iterative development through capability demonstrations.	Can converge upon meeting requirements while managing risk incrementally with assessment points. Process will converge upon what works for the team and the project. Useful in controlling unpredictable processes.	The cost of iterations necessary during the RDT&E program phases requires more front-end investment. Processes in the end phases may be completely different than processes initiated at startup.
Agile	Most agile class of methods attempt to minimize risk by developing software in short time boxes, called iterations, which typically last one to four weeks. Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. <sup>2</sup>	Used to produce sufficiently advanced models. Also inherits adaptive characteristics.	Can converge upon meeting requirements while managing risk incrementally with assessment points. Life cycle costs are actually driven down, delivery time is shorter, and quality is higher.	May require non recurring investment on first implementation.



Table A-2 (cont.). Is a representation of comparability for some popular existing adaptive software development techniques.

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Crystal Family	A technique that puts focus on roles of the development team (i.e. sponsors, designers, users, etc), and communication between the team members. The ideal team is 4 to 6 people.	Use of processes specifically tailored to the skills of the team. Select methods of appropriate weight and tailor them based on project size and criticality.	Exploits unique qualities from within the developer's organization. Usually successful with a small development team.	Does not accommodate for missing skill sets.
Dynamic Systems Development Method	The result of a United Kingdom Industry Consortium extracting best practices and mapping them into a framework.	Fix time and resources then develop the functionality while adjusting functionality with the design build iterations.	On time delivery of products. Empowered software development team build around user.	Not accommodative to requirements that are not flexible.
Extreme Programming	XP is a discipline of software development applicable to small teams who need to produce quick results in dynamic environments.	Assumes requirements are frozen then implements a cycle of planning, design, coding and testing. Good for rapid prototyping.	Customer focused and can produce successfully developed software despite vague or changing requirements. Life cycle costs are actually driven down, delivery time is shorter, and quality is higher.	Difficult to apply in organizations where the acquisition personnel (developers) are not co-located from the users.



Table A-2 (cont.). Is a representation of comparability for some popular existing adaptive software development techniques.

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Lean SW Development	Lean Software development is a translation of lean manufacturing principles and practices to the software development domain. It can be applied both the predictive and adaptive techniques.	Lean SW development relies on a CMMI-like infrastructure for improvement and CMMI goals). Focus is on the core task of identifying and eliminating waste while driving key decisions outward in the development cycle.	Highly responsive to customer needs, and provides a best-in-class answer to requirements volatility issues	Often requires a facilitator to realize productive gains. Senior management support is a must.
Feature Driven Development	(FDD) is an iterative and incremental software development process. It is one of a number of Agile methods for developing software and forms part of the Agile Alliance.	Iterative development of business systems, and an instance of agile technique. FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality perspective.	Frequent and tangible deliverables. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.	The cost of iterations necessary during the RDT&E program phases requires more front-end investment.
Internet Speed Development	An Agile Software Development method using a combined spiral model/waterfall model with daily builds aimed at developing a product with high speed.	Closely aligned to agile development principles. Management oriented framework for fast releases based on time drivers, quality dependencies and process adjustments.	Copes well with fast changing requirements.	Iterations will increase the cost of RT&E program phases.



Table A-2 (cont.). Is a representation of comparability for some popular existing adaptive software development techniques.

Applied Development Technique	Technique Definition	Basic Feature(s)	Basic Advantages	Basic Disadvantages
Object Oriented Programming (OOP)	OOP centers on the development of small, reusable program routines (modules) that are linked together and to other objects to form a program.	Event driven objects are developed using a hierarchal class structure. Each class features parent-child relationships that facilitate reuse. Child objects understand codes inherent to their parents, plus their own unique code.	Suitable for event driven functionality. Code reuse facilitated by inheritance and upgrades to short modular code modules in model, view, or controller paradigms. Investment of usable repository promotes reuse and serious reductions in development times. Maintenance of that software will be much simpler and much less error prone as consequence of up front design investment in careful designs.	Lack of standardized coding procedures. Requires skilled developers to exploit the power of OOP. Also useful in adaptive category when exploiting reuse.
Scrum	Scrum is an agile, lightweight process that can be used to manage and control software and product development using iterative, incremental practices. Wrapping existing engineering practices, including Extreme Programming and RUP	Frequent management activity to identify deficiencies or impediments coupled with developer's selection of techniques. SCRUM is actually a daily mgt. technique applied in agile development efforts.	Prioritized backlog most important items get worked. Supports scalability concepts through OOP approach. It produces a potentially shippable set of functionality at the end of every iteration.	Requires a balance of management but not micro-management. Substantial front-end architectural preparation required. Some coders not comfortable with level of responsibility brought by scrum.



## Appendix B: Evaluating Tool Types

### 1 Configuration Management Tools

- Configuration management tools can be effectively applied throughout the Software Development Life Cycle (SDLC).
- Configuration management tools should be interoperable with requirements management tools, for ease of tracing requirements to baselines.
- An acquiring agency can require that offerors describe their actual or intended processes and tools for configuration management.

#### What It Is -

Software configuration management (SCM) is a methodology designed to:

- Identify software configuration items
- Provide a change control process
- Provide status accounting reports describing baselines, version numbers, open and closed trouble reports, related documentation)
- Support configuration audits.

#### Why It is Needed -

The introduction to the IEEE "Standard for Software Configuration Management Plans" [IEEE 828-1998] says this about SCM: SCM constitutes good engineering practice for all software projects, whether phased development, rapid prototyping, or ongoing maintenance. It enhances the reliability and quality of software by:

- Providing structure for identifying and controlling documentation, code, interfaces, and databases to support all life-cycle phases
- Supporting a chosen development/maintenance methodology
- Producing management and product information concerning the status of baselines, change control, tests, releases, audits, etc.

#### How to Evaluate The Tool -

Determine which of the following are important to the program and can be supported by a candidate tool:

- ➔ Does the SCM Tool address the following areas?
  - Identify functional, allocated, and product baseline configuration items
  - Relate functional, allocated, and product baseline configuration items
  - Associate configuration items with system requirements
  - Provide status reports describing software and documentation configuration items and related trouble reports and change requests.
  - Support online status reports.
  - Support online ad-hoc query.
  - Provide help tools for users.
  - Track all trouble reports by configuration item.



- Track all trouble reports by location (if multiple).
- Allow online submission of trouble reports.
- Allow online submission of change requests.
- Track changes requested and approved for each configuration item.
- Allow users to submit change requests.
- Allow users to prioritize change requests.
- Simplify or automate change request notification to members of the configuration control board.
- Support input from subject matter experts (SME's).
- Support cost /schedule impact input from SME's.
- Track prioritization changes recommended by Configuration Board after SME input.
- Track configuration control board recommended action for each change request (fix, defer, etc.).
- Track configuration control board manager's decision for each change request.
- Track configuration control board recommendation for fixes and enhancements slated for next release.
- Track configuration items by date and location for fielded systems.

## 2 Requirements Definition and Management Tools

- Software requirements management tools are essentially tools that track system requirements and derived requirements and any changes.
- The acquirer can use the RFP to emphasize the importance of requirements traceability, and can evaluate the offeror's proposal for description of capability and process.
- A requirements tool that supports tracking projected costs for associated development can help a project or program manager to make tradeoff decisions (between enhancements and fixes) for future releases.

### What It Is –

Requirements management tools support a continuous process of tracking requirements to the capability provided and tested. Requirements tools can be used to show how a system's requirements map to mission and Joint capabilities, as well as how the requirements are mapped into system design, test procedures, and baselined systems.

### Why It Is Needed –

Rework can account for up to 40% or more of a development organization's total spend - time and money. Correcting requirements defects can be 50 to 200 times more expensive after deployment than during the early design stages. By eliciting, specifying, analyzing, and validating requirements early, costly rework can be reduced later in the development lifecycle. SDLC tools that address collaborative software requirements definition and management are necessary for:

- Solidification of actual requirement.
- Avoiding requirement creep.
- Traceability from requirements to functionality.



### How to Evaluate The Tool –

Evaluation should consider:

- ➔ Does the software requirements definition and management tool address the following areas?
  - **Elicitation:** Method(s) for customizing templates or forms (or surveys) for requirements definition, which can include user scenarios in easily understandable forms.
  - **Specification:** Method(s) for –
    - × documented requirements
    - × traceability from top-level to derived requirements
    - × defining a hierarchy of requirement types, attributes, and relevant data
  - **Analysis:** Method(s) for –
    - × capturing, displaying, and aggregating individual and group evaluation of derived requirements
  - **Validation:** Method(s) for –
    - × capturing requirements review, signoff, and creation of a baseline
  - **Management:** Method(s) for –
    - × tracking changes and associated approval
    - × establishing processes for managing changes (requests, impact analysis, and communicating changes)
    - × focusing resources and project planning
  - **Verification:** Method(s) for tracing requirements to test plans, processes, and test results.

### **3 System Analysis and Modeling Tools**

- System analysis / modeling tools may be proposed or in use by a contractor. The acquirer will need to evaluate advantages described against costs for licenses, training, and required expertise.
- The return on investment for such tools may be difficult to assess, and a significant learning curve is often associated initially.
- The tools should be evaluated for use of interface standards (standard input and output formats to facilitate data sharing and also to make it possible to change to another tool or process if necessary
- The products of the tool should be understandable to any Stakeholders that should be involved in reviewing the design or process captured. If the user/Stakeholder cannot walk through and validate the captured design, then the tool may be impeding common understanding and agreement of design decisions.

#### What It Is –

Modeling establishes a way to visualize the design and compare it against requirements before coding begins. Modeling involves describing desired features and operations in detail, including:

- screen layouts
- business rules



- process diagrams
- pseudocode
- other documentation

The Unified Modeling Language (UML) is common among developers for specifying, visualizing, and documenting software design. The UML diagrams may be difficult for non-developers, so this can be a major consideration if the design needs to be briefed or "walked through" with users or other stakeholders.

#### Why It Is Needed –

Acquirers will want to be familiar with any tool and product that is used to capture design. The more eyes on the design, the more errors will be caught at this early stage. Design errors that are not identified in design walk-throughs too often result in multiple seemingly-unrelated problems, many of which will be caught only in the field, and the fixes to those problems will introduce more bugs. Thus the design tool, and ability to use it with all appropriate stakeholders during design walk-through, is key to reducing rework.

#### How to Evaluate The Tool –

Evaluation should consider:

- Decide who the tool will serve, and obtain input.
- In evaluating a modeling tool, careful consideration should be given to the computing (host) platform(s):
  - Will the software development be for Windows or Unix or both, etc?
  - On which platform will the development be conducted?
  - License cost.
- Does the modeling tool address the following areas?
  - **Repository Support** - Provides data sharing and concurrency control features.
  - **HTML Documentation** - Provides a static view of the object model that can be referred to quickly in a browser, without having to launch the modeling tool itself.
  - **Round-Trip Engineering** - The ability to both forward and reverse engineer source code.
  - **Debugging** - The ability to debug with either the UML (For UML tools) or with code
  - **Data Modeling** – Allows integration of data modeling facilities while also supporting the synchronization of data and object models after each iteration of design.
  - **(For UML tools) Full UML Support** - Diagrams which should be supported are the Use Case, Class, Collaboration, Sequence, Package, and State diagrams.
  - **Versioning** – Allows versions to be saved so that as subsequent iterations are created, previous versions are available for restore.



- **Scripting** – Supports the generation of scripts for the purpose of directly accessing the object model to create other artifacts (i.e. metrics, reports, documentation).
  - **Diagram Views w/Export**- Facilitates customization of the view of a class and its details and the capability to export into common formats such as word or web page format.
  - **Printing Support** - Allows accurate renditions of large diagrams to be produced through multi-page printing.
  - **Metrics** – Provides feedback on the viability of a particular model.
- Does the system analysis tool address the following areas?
- Database schema
  - Data Flow Diagrams
  - Entity Relationship Diagrams
  - Program Specifications
  - User Documentation

#### 4 Development Tools

- Numerous products are available for use in development; the combination of tools will vary by type, size, scale, and complexity of a development effort. There is no single best solution, and many developer organizations will have established practices and tools that evolve over time.
- The developer's plan should include a number of risk-reducing steps within the development phase. Development tools are available to support these important steps of code inspection, code review, and unit testing.
- A Capability Maturity Model of Level III or above is an indication that the developer has the required experience and employs tools appropriately. Higher risk programs may stipulate higher levels of capability maturity.

##### What It Is –

Development tools such as code generators, code inspection tools, debuggers, and tools accommodating test generation compliment implementation processes. Automatic Software Inspection (ASI) tools verify that the software is compliant with coding standards.

##### Why It Is Needed –

Errors are always easier to isolate and cheaper to fix the earlier they are found. Development tools make it easier for the developer to move from design to code, track the progress of each code unit, check each unit for compliance with local and general coding standards, and support communications among the developers.

**How to Evaluate The Tool– This area is intended for the developer community; the acquirer will normally not be involved at this detail.**



- Identify whether the tools full capabilities span the activities of multiple SDLC phases.
  - **Positive:** Provide integrated functionality.
  - **Negative:** Can be expensive.

#### **Code Generation**

- Identify what role a Code Generator will play in code creation / revision.
  - Will manual coding also be needed?
  - Will integration of manual coding and auto generated code be necessary?
  - How will code debugging be conducted?
- Identify whether the tool is a stand-alone code generator.
  - **Positive:** Less expensive.
  - **Negative:** Can be effort intensive because of the need to move back and forth between modeling tool and code generator.

#### **Code review / Software inspection**

- Identify whether code reviews / inspections are being done with an ASI tool.
  - **Positive:** The cost of a complete inspection becomes less prohibitive as a code base grows.
  - **Negative:** The cost and effort required to find true defects using ASI tools is high, because a large number of false positives must be manually evaluated and eliminated.
- Identify whether code reviews / inspections are being done with an ASI service.
  - **Positive:** ASI services provide code review in significantly less time and at a dramatically lower cost than manual inspection or internal use of inspection tools.
  - **Negative:** The cost and effort required to find true defects using ASI tools is high, because a large number of false positives must be manually evaluated and eliminated.
- Does the ASI tool / services address the following areas?
  - Identify the location and describing the circumstances under which the defects will occur.
  - Identify the parts of the code with the greatest risk.
  - Compare code quality with a benchmark.
  - The breakdown of defects by defect class.
  - Identify syntax and interface errors.
  - Identify potential for reducing code volume via redundant or unused code.

## **5 Defect Tracking Tools**

- The acquirer may require in the SOW the use of some type of defect tracking tools by the supplier.



- The size, scale, complexity, program visibility and/or risk may drive acquirer interest in defect tracking. The number and type of defects can be important to identify and correct root cause and thus improve quality control.
- Defect tracking tools can be applied throughout the SDLC.
- Defect tracking tool reports can be effectively used by the acquirer.

#### **How to Evaluate The Tool –**

Evaluation should consider:

- ➔ Does the defect management tool address the following areas?
  - Enable the user to track defects:
    - \* by source unit
    - \* by programmer
    - \* by date
    - \* by type defect
  - Enable the categorization of Defects.
  - Enable customization of Defect content.
  - Support standard and customized reports.

## **6 Source Code (Security) Analysis Tools**

- The size, scale, complexity, and other system / software development components will determine the level and type of source code analysis tools required to be employed by the supplier.
- The acquirer will want to identify security and source code vulnerability as an important area of concern, and may want to specify threat so that offerors can describe security analysis tool selection and application if appropriate.
- Source Code Analysis tools can be applied throughout development and maintenance.

#### **What It Is –**

Source Code Analysis (SCA) tools in general provides risk management through an automated method which is utilized to eliminate coding errors and design flaws. Security Analysis tools manage security risk for coding errors, design flaws and policy violations.

#### **Why It Is Needed –**

Source code security analysis is supported within a number of tools. Source code vulnerabilities are difficult and expensive to identify through manual inspection. Tools continue to improve and evolve along with the threat. Use of the tools will increase cost and schedule, and project estimates must consider these costs. Risk analyses may be conducted to determine the potential impact of realized risk (vulnerabilities being found and exploited) before determining level of investment and application in such tools.

#### **How to Evaluate The Tool –**



Evaluation should consider:

- Does the security analysis tool address the following areas?
  - Inspects calls to identify potential “Insecure” library functions
  - Identifies bounds-checking errors and scalar type confusion
  - Identifies type confusion among references / pointers
  - Detection for memory allocation errors
  - Detection of vulnerabilities which involve sequences of operations (Control-Flow Analysis)
  - Perform data-flow analysis
  - Perform pointer-aliasing analysis
  - Provide customizable detection capabilities

## 7 Testing Tools

The IEEE Standard Glossary of Software Engineering Terminology defines verification and validation (V&V) as:

"the process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements."

The V-model illustrates that test planning as a part of requirements, specification, design and coding efforts should render acceptance, system, integration and unit test plans which are compliant with design.

The V-model illustrates that test planning as a part of requirements, specification, design and coding efforts should render acceptance, system, integration and unit test plans which are compliant with design.

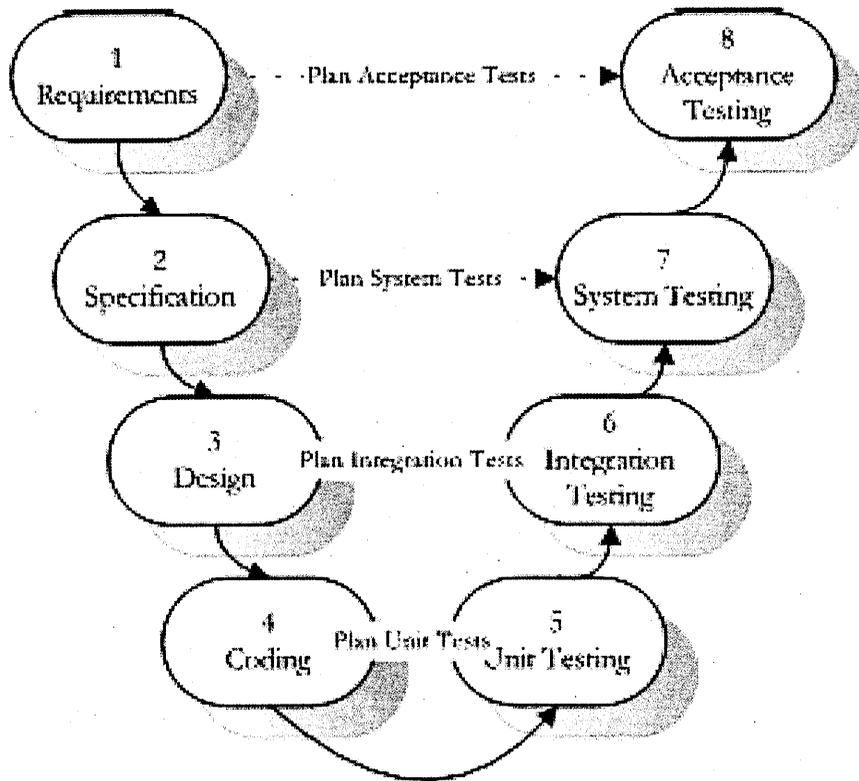


Figure B-7-1. Test planning is an integral part of the V process.

- Testing tools can be used throughout the SDLC, beginning with tracing requirements and derived requirements to test design or plan.
- The acquirer will look for CMM Level III development capability to ensure repeatable processes including test processes.
- The acquirer will likely have independent and/or oversight testing and certification steps that may require acquirer-level investment or understanding of test tools.

**What It Is –**

Test tools provide support in the areas of:

- Test planning and monitoring
- Designing test cases
- Constructing test cases
- Executing test cases
- Capturing and comparing test results
- Reporting test results
- Tracking software problem reports/defects



- Managing the test ware

**Why It Is Needed –**

Test tools can improve the test team ability to conduct repeatable tests, identify defects, track defects to code modules, and produce test reports. While it is impossible to fully test any sizeable computer program, test tools can automate some tests, increasing the number of tests that can be conducted and eliminating some sources of human error.

**How to Evaluate The Tool –**

Evaluation should consider:

- Does the testing tool address the following areas?
  - Test Management - Enable the user to author and maintain requirements:
    - \* Support the authoring of Test Requirements.
    - \* Support the maintenance of Test Requirements.
    - \* Support controlled access to Test Requirements.
    - \* Support discrete grouping or partitioning of Test Requirements.
    - \* Support traceability of requirements to Test Cases and Defects.
    - \* Support canned and user defined queries against Test Requirements.
    - \* Support canned and user defined reports against Test Requirements.
    - \* Support coverage analysis of Test Requirements against Test Cases.
    - \* Support the integration of other toolsets via a published API or equivalent capacity.
  - Test Automation - Enable the user to author, maintain, and execute automated test cases by:
    - \* Support the creation, implementation, and execution of Automated Test Cases.
    - \* Support controlled access to Test Automation.
    - \* Support Data Driven Automated Test Cases.
    - \* Support Keyword enabled Test Automation.
    - \* Integrate with all Tier 1 and 2 Test Management tools that support integration.
    - \* Integrate with all Tier 1 and 2 Defect Management tools that support integration.
    - \* Enable Test Case Design within a non-technical framework.
    - \* Enable Test Automation and verification of Web, GUI, .NET, and Java applications.
    - \* Support the integration of other toolsets via a published API or equivalent capacity.



## References:

---

- <sup>1</sup> Statement of David M. Wenngren, Department of the Navy Chief Information Officer before the House Armed Services Committee Terrorism, Unconventional Capabilities Subcommittee, 3 April 2003.
- <sup>2</sup> "Software Acquisition Process Improvement Program", OSD Memo, March 21, 2003.
- <sup>3</sup> S. L. Pfleeger, Software Engineering: Theory and Practice, 1998.
- <sup>4</sup> B. W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, May 1988.
- <sup>5</sup> Department of the Navy, "Open Architecture (OA) Computing Environment Design Guidance" Version 1.0, 23 August 2004.  
[http://www.nswc.navy.mil/wwwDL/B/OACE/docs/OACE\\_Design\\_Guidance\\_v1dot0\\_final.pdf](http://www.nswc.navy.mil/wwwDL/B/OACE/docs/OACE_Design_Guidance_v1dot0_final.pdf).
- <sup>6</sup> Netcentric Enterprise Solutions for Interoperability (NESI) web site.  
<http://nesipublic.spawar.navy.mil>.
- <sup>7</sup> The Data Analysis Center for Software.  
<http://www.dacs.dtic.mil>.
- <sup>11</sup> N. Davis, J. Mullaney, "The Team Software Process in Practice: A Summary of Recent Results," CMU/SEI-2003-TR-014, September 2003.
- <sup>12</sup> [http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development).