Amir Tomer

Amir Tomer is the Director of Systems and Software Engineering Processes at RAFAEL Ltd., Israel ,with whom he has been since 1982 ,holding a variety of systems and software engineering positions ,both technical and managerial. His B.Sc. and M.Sc. degrees in computer science are from the Technion, Israel, and his Ph.D. in computing from Imperial College, London, UK. Amir also teaches software engineering at the Technion and other colleges and is active in research and publication in software and systems engineering.

Iterative Software Development – from Theory to Practice

Amir Tomer, Boaz Shani, Ely Bonne – RAFAEL, Israel
P.O.Box 2250/1P, Haifa, Israel, tomera@rafael.co.il

Abstract

Iterative or incremental software development appears to be a promising software process approach, and it looks great in theory. However, it's anything but simple when it comes time to apply it to real projects. Iterative development is even more difficult to implement when your staff is familiar — and comfortable — with the waterfall method. This paper describes how iterative software development process has been adopted in RAFAEL, based on the Unified Software Development Process (USDP) introduced by Rational Inc. The paper identifies the core differences between iterative and waterfall software development, addresses the possible difficulties in applying the iterative process and details the elements of an iterative software development process tailored to RAFAEL, in view of its activities, work products, reviews and other terminology adaptations.

1 Introduction

RAFAEL is a research and development institute producing sophisticated military systems, including missiles, EW systems, communication systems, simulators, intelligence systems and other systems. Most of RAFAEL's products are software-intensive and therefore software development has always been in focus. During the last two decades the standard software life cycle was a "waterfall" style, based on the military standards DOD-STD-2167A and MIL-STD-498. Recently, together with the adoption of the Object-Oriented paradigm, the advantages of iterative development has been recognized as a means of early mitigation of development risks and minimization of rework effort. The Unified Software Development Process (USDP), introduced by Rational Inc. has been examined and found appropriate to be adopted as a standard software development process in RAFAEL, subject to necessary adaptations and tailoring.

In this paper we describe how the USDP has been interpreted, tailored and adapted in RAFAEL. A major principle in this adaptation was the mapping of the new process into the existing one, to the widest extent possible.

Since software is now taking larger parts in sophisticated military systems, many elements of software engineering are largely applicable to other engineering disciplines, mainly to systems engineering. The software development process described here starts, in fact, at the early stages of the system and covers the entire system lifecycle. Therefore, the context of "software development" should be interpreted as "development of a software intensive system".

2 Software Development "Waterfall" Style

The waterfall model for software development was initially proposed by Royce [1] and has been adopted and implemented widely in the software industry. The model is based on a set of activities (requirements, analysis, design, code etc.) applied sequentially along the development life cycle. Each development stage is accomplished by verification of its work products, before proceeding to the next phase. When defects are revealed through verification, they are fed back to previous stages in order to be corrected.

The waterfall model underlies the military standard DOD-STD-2167A [2] which was widely used since 1988. In 1994 it was replaced by another military standard, MIL-STD-498 [3], which is still used widely for software development, both in the defense and in the civilian industry.

The military standards contain detailed templates of the work products (named DIDs – Data Item Descriptions) to accompany software development. Many of these DIDs, together with their names and acronyms, have penetrated as standard vocabulary to software engineering.

The main flaw of the waterfall model is its strict policy of accomplishing a certain phase before proceeding to the next one. For example, programmers should not start coding before full completion and approval of the design documents, based on previously approved requirements documents, system design documents, system specification documents and others. The direct impact is that no tangible visibility into the system is provided until late stages of the life cycle. In other words, clients, developers and other stakeholders are expected to understand and approve the entire system on the grounds of written documentation. The consequences are, in many cases, that major defects are revealed only after pieces of code has been programmed, integrated and tested – long after the analysis and design has been approved. Besides causing lengthy modification cycles, risk mitigation is postponed to very late stages.

3 The Unified Software Development Process

The Unified Software Development Process (USDP) [4] is a detailed scheme for iterative and incremental software development. The USDP has been developed at Rational Inc. on the basis of Ivar Jacobson's Objectory [5] and many other sources. Therefore it is commonly referred to as RUP (Rational Unified Process) or simply as UP. The major difference between the USDP approach and the waterfall approach is that the phases of the waterfall process (such as requirements, analysis, design, code etc.) may be applied iteratively throughout development, and therefore are defined in USDP as "workflows" rather than "phases". In turn, the entire product life cycle is divided, along the time axis, into four phases, as follows:

- The **inception** phase, through which the concept, the scope and the vision of the product is being formed, up to a stage which enables it to be engineered;

- The **elaboration** phase, through which the resources, activities and work products are defined and planned, together with the product life-cycle architecture;
- The **construction** phase, through which the product is implemented incrementally;
- And the **transition** phase, through which versions of the product are delivered to the client and deployed for operational use.

Each of these phases is sub-divided into iterations. In each iteration all the workflows may be applied, to the extent according to the current level of development. Each iteration is expected to produce an operational version of the system (also referred to as "a build"). The USDP scheme is depicted in Figure 3.1, taken from [6].

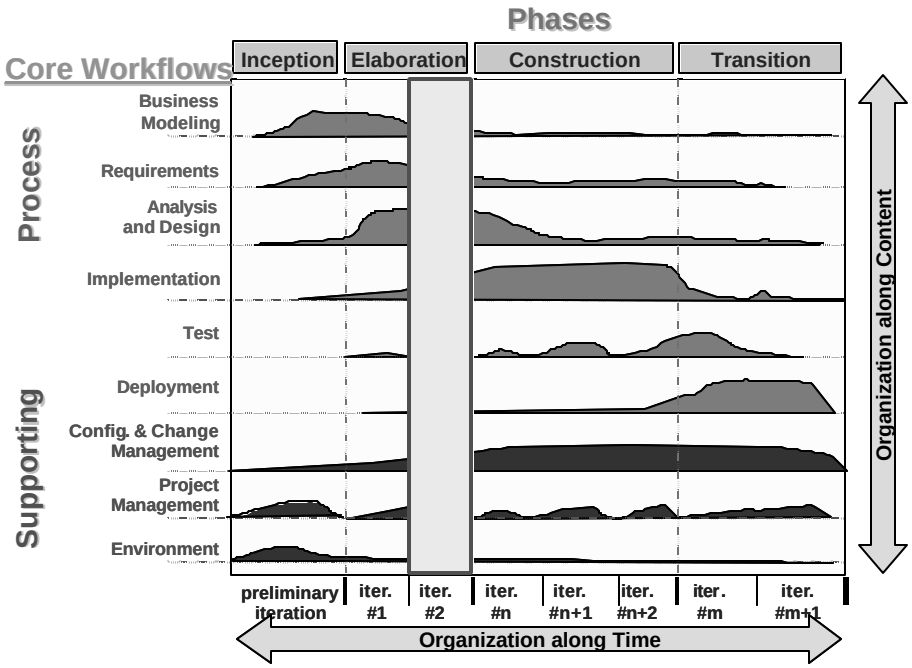

Figure 3.1: The Unified Software Development Process

The USDP has recently been proposed, by Rational Inc. and other companies, to the OMG, to be adopted as an industrial ad hoc standard, for software and systems development.

Although being a general process, the USDP fits best the Object-Oriented paradigm for software analysis, design and programming. Moreover, it is best applied when the underlying engineering activities utilize the UML (Unified Modeling Language) [7], which was also developed at Rational but has been adopted as an industrial standard by OMG [8].

4 From MIL-STD-498 to USDP

The MIL-STD-498, and its predecessor DOD-STD-2167A, has been used in RAFAEL as the standard life cycle models for the last two decades. RAFAEL software development procedures are written "in the spirit" of those standards, directing software developers to perform their activities and produce their products according to these standards. The DID templates have been recommended as standard documentation forms, and some of them have been translated into Hebrew. In recent years the Object-Oriented paradigm has been widely adopted by software developers, many of whom have started to use UML as their analysis and design language, utilizing appropriate CASE tools. Along with this trend, the benefits of iterative development became clearer to many engineers, managers and clients, and the USDP was examined for its applicability.

In this section we describe how the USDP has been interpreted and adapted to the software development process already in place. Most of the adaptation work has been done by teams of skilled and experienced software engineers, whose task was not only to interpret and explain the terms in a vocabulary most people are familiar with, but also to tailor commonly used templates for hosting the new process scheme.

4.1 Life-Cycle Phases

The USDP phases, as described above, form the basis of the development life cycle for software intensive systems. They are interpreted as follows:

- During the **inception** phase the scope and concept of the system is well defined, and solution alternatives are proposed and examined through feasibility tests;
- During the **elaboration** phase a stable operational system architecture is built and tested and the major risks are mitigated;
- During the **construction** phase the product is implemented in iterations, each of which produces an operational version of the system, which may, or may not, be delivered to the client;
- During the **transition** phase the system is installed at the client's site, together with appropriate instruction, support and maintenance.

According to the USDP principles, multiple activities are performed at each phase, as described in the following section.

4.2 Stages, Activities and Work Products

Any development process comprises a set of staged activities. The differences between various development processes and life cycles lies in the order and frequency in which each one of these activities is applied throughout development.

Stages are defined on two levels: system level and CSCI level. System level stages are applied first, in order to define the system general requirements, design and architecture and system breakdown to Computer Software Configuration Items (CSCIs) and HardWare Configuration Items (HWCIs). Then, stages are applied at CSCI level to

each CSCI. Upon completion of CSCI development, additional system level stages are applied, in order to accomplish the system towards delivery.

Table 4.1 below summarizes the development stages at both levels. Detailed descriptions are provided in the following two subsections.

Table 4.1: System development stages and work products

Level	No.	Stage	Work Products	Phase
System level	1	System boundary definition	• System Charter / Vision Document	Inception
	2	System Requirements	• SSS • System requirements base • Use Case model (optional, recommended)	
	3	System Design	• SSDD • ICD (external and internal)	
	4	Software Planning and Risk Analysis	• SDP • STP	
CSCI level	5	Software Requirements	• SRS • Software requirements base • Use Case model	Elaboration / Construction
	6	Software Analysis and Design	• SAD • CASE based design model • STD	
	7	Coding and Unit Testing	• Successfully tested code	
	8	CSCI approval	• Approved CSCI	
System level	9	System Integration and Testing	• An approved build of the system, ready for delivery	

4.2.1 System Level Stages

The following stages are applied at system level, as shown in Table 4.1 above. Stages 1 through 4 comprise the inception phase and are performed exactly once. Stage 9 is performed at each iteration, throughout the elaboration and construction phases, in sequence to stages 5 through 8 at the CSCI level.

Stage No. 1: System Boundary Definition. The purpose of this stage is to provide a common ground and understanding about the system for the client, the developers and any other stakeholders. The product of this stage is a brief, but exhaustive, document named System Charter or Vision Document. It is a complete document (that is, it is written and approved once, and not iteratively), which contains system purpose and goals, its relationships with external systems and its primary requirements. Although this document is optional, it is strongly recommended, based on feedback from clients, project managers and system engineers who reported this basic document to be a clear and well-defined focus on the purposes of the system and its main requirements. However, the best benefit of the system charter is gained by its authors – system engineers and software team leaders – since it leads them to concentrate on the main features and document them in writing. In many cases it is more important to define what is **excluded** from the system, rather than what is **included** in it. Once this is

performed properly, development planning is easier and the entire course of development is clearer.

Stage No. 2: **System Requirements.** The purpose of this stage is to define in detail the system requirements, at the current level of knowledge. Two products are produced at this stage: SSS document and the systems requirement base. The SSS (System / Subsystem Specification) contains external interfaces and functional and other requirements. Although not mandatory, it is strongly recommended to specify the functional requirements in the form of Use Cases [7,8], which will be the form in which each CSCI's software requirements will be defined. The SSS document is expected to be complete, although a few "isolated" topics may be left incomplete for further elaboration. The SSS document is written in MIL-STD-498 based format, using UML annotations and diagrams. The system requirements base is a database containing the system requirements in an enumerable and well defined form, together with other attributes, defined by the system engineer in charge. The system requirements base is subject to change management throughout development, and therefore it is strongly recommended to be implemented in a computerized tool. For small projects, a simple tool, such as an electronic spreadsheet, may be sufficient.

Stage No. 3: System Design. The purpose of this stage is to provide a stable proposed solution for implementing the system, including its architecture and its breakdown to software and hardware configuration items (CSCIs and HWCI, respectively). The main product of this stage is an SSDD (System / Subsystem Design Description) document (MIL-STD-498 format), containing alternatives for system solutions, the selected system architecture, CSCI/HWCI breakdown and the allocation of system requirements to the configuration items. The interfaces (both external and internal) should be defined as part of the SSDD document, or as separate ICD (Interface Control Definition) documents.

Stage No. 4: Software Planning. Based on the system requirements and design a Software Development Plan (SDP) is produced, prior to commencing software development. The SDP document contains the details of software development stages for all the CSCIs in the form of an iteration plan. The overall iteration plan is constructed such that each iteration is expected to mitigate at least one major risk, based on the table of risks included in the SDP. A detailed plan of the first iteration is included in the SDP. Although the overall iteration plan is complete, the SDP, as a whole, is expected to be modified throughout the project, by adding a detailed iteration plan before the development of each individual iteration commences (see below). In addition, the general Software Test Plan (STP) is defined at this stage, either as a separate document (MIL-STD-498 format) or as part of the SDP. In order to adapt the SDP for iterative development a special SDP template has been designed on top of the basic MIL-STD-498 format.

Stage No. 9: System Integration and Test. This stage is performed at each iteration, after the CSCI level stages (No. 5-8) has been performed for each CSCI (see next section). Since a basic concept of the iterative software development is the delivery of an executable version of the system (a "build"), the purpose of this stage is to integrate the approved versions of the CSCIs (see below) into a complete system and validate it,

according to system test specifications, up to the current iteration. It is emphasized, that the resulted build is an extension of the previous one, and must comply to all the previous requirements, unless changed by request. This means that the tests performed on each built contains a significant amount of regression testing.

4.2.2 CSCI Level Stages

The following stages (5 through 8) are performed separately for each CSCI, on the basis of the system documents and other products produced in the system level stages (1 through 4) above. The products of these stages are produced iteratively, that is, at each iteration additional details, associated with the iteration in view, are added to those produced in previous iterations. Stage No. 5 of each iteration is preceded by a detailed iteration plan (an appendix to the SDP), which defines clearly which of the CSCIs participate in the current iteration and the selection of its allocated systems requirements which will be implemented in this iteration.

Stage No. 5: Software Requirements. At this stage the software requirements for the current iteration are defined and specified in detail, on top of the software requirements from previous iterations. If changes to requirements have been received, as conclusion of operating the previously delivered iteration, they are incorporated into the definition of the requirements of the current iteration. The main product of this stage is the SRS (Software Requirements Specification) document, containing functional and other requirements. The software requirements base is generated (in the first iteration) or extended (in following iterations) by extracting requirements from the SRS. The software requirements base is derived from the systems requirements base, and is recommended to be implemented on a computerized tool (such as RequisitePro or Doors). However, the best practice is to manage both system and software requirements in a common tool, if applicable. This avoids unnecessary duplications and keeps better integrity of the entire requirements database. An additional product of the requirements phase is the Use Case model, constructed using a CASE tool, capturing the functional software requirements of the CSCI. In order to incorporate the Use Cases into the SRS, a special template has been designed, on top of the basic MIL-STD-498 format. The SRS contains traceability of all its requirements, up to and including the current iteration.

Stage No. 6: Software Analysis and Design. At this stage the requirements of the CSCI are analyzed in order to generate/update the software design for the CSCI. The software architecture acts as the top level design of the software, interpreted in various views (usuall the “4+1” viewes defined in [6]), including the identification of those requirements affecting explicitly the software architecture. Various design models may be used for the software design, such as Class Diagrams, Sequence Diagrams, Activity Diagrams and Statechart Diagrams [7,8]. The main document produced at this stage is the SAD (Software Architecture Description) document, which is arranged according to a special template combining both the SDD (Software Design Description) format of MIL-STD-498 and concepts taken from [4] and [9]. The SAD document captures the top level (architectural) design, whereas the details of the design reside in the models themselves, which undergo further refinement. It is emphasized that documents are not written independently, but are rather extracted, as snapshots, from the models. In

addition, the detailed tests of the CSCI, at the current iteration level, are designed, in a STD (Software Test Description) document.

Stage No. 7: Coding and Unit Testing. At this stage the software modules are coded and tested according to the design. According to RAFAEL's software development procedures unit testing is performed at the discretion of the programmer and is documented informally. It is the responsibility of the software team leader to approve the code module and its test coverage prior to inserting it into the configuration management library. The approved code modules are the products of this stage.

Stage No. 8: CSCI Validation and Approval. Each CSCI is validated and approved upon successful completion of the tests specified in the STD (Software Test Description). These tests are usually performed in an environment containing a simulator, in which test scenarios are defined and test results are recorded and analyzed. For large CSCIs partial integration and testing may be performed in several steps. The product of this stage is a validated and approved CSCI, ready for system integration. When all the required CSCIs are ready, stage 9 (see above) is performed at the system level.

4.3 Reviews

Reviews are the means by which work products are verified. Proceeding from one development stage to the following is dependent upon successful accomplishment of the corresponding review. Reviews are categorized into two types: external reviews and peer reviews, as described in the following. Reviews are planned in advance for the entire project, and the review plan is documented in the SDP. Reviews are performed according to RAFAEL procedure for software reviews.

4.3.1 External Reviews

External reviews are performed for the clients and with their participation, according to the contract. Their main purpose is to enable the client to get an insight into the system and its design at significant milestones during development. Recommended external reviews are enlisted in Table 4.2.

Table 4.2: Recommended external reviews

External Review	When?	Reviewed work products	Comments
SRR (System Requirements Review)	End of Stage 2	• SSS	
SDR (System Design Review)	End of Stage 4 of the Inception Phase	• SSDD • SDP	
SSR (Software Specification Review)	End of Stage 5 of selected/significant iterations	• SRS of relevant CSCIs	

CDR (Critical Design Review – for the software)	End of Stage 9 of the Elaboration phase	• SAD • An operative system	At this stage an operative skeletal system, with stable architecture, is expected, with all its architecture components tested for their appropriate function.
TRR (Test Readiness Review)	End of Stage 8 of client deliverable iteration	• STR (Software Test Reports)	Approving the readiness of each CSCI for system integration and testing.

4.3.2 Peer Reviews

There are four peer reviews associated with each iteration. Their main role is to objectively review the work products, in order to alert the developers for any defects anticipated in the products according to design errors, deficiencies or misinterpretation of requirements. Peer reviews are internal, that is, without the presence of the client. Peer review teams often include, besides software engineers, system engineers and engineers of other relevant disciplines. Recommended peer reviews are enlisted in Table 4.3. Note that reviews may be unified, as applicable.

Table 4.3: Recommended peer reviews

Peer Review	Timing	Reviewed work products	Comments
Iteration plan review	Before Stage 5 of **each** iteration	• SDP appendix containing the detailed plan of the current iteration	Emphasis on the goals and risk mitigation of the current iteration
SSR (Software Specification Review)	End of Stage 5 of **each** iteration	• SRS of relevant CSCIs	
SDR (Software Design Review)	End of Stage 6 of **each** iteration	• The appropriate design models • STD of relevant CSCIs	
Iteration completion review	End of Stage 9 of **each** iteration	• Demonstration of the system developed so far • Updated risk status	The results of this review is the input for the next iteration plan

4.4 An Example of Iterative Software Development

Table 4.4 describes an example of the entire product development life cycle of a system containing 3 CSCIs. Life cycle contains two elaboration iterations and two constuction iterations. Note that not necessarily **all** mentioned CSCI participate in each system iteration. Furthermore, CSCI level reviews not necessarily align, and may be performed separately or unified, as applicable.

Table 4.4: Iterative Software Development

Inception		1. System Boundary
		2. System Requirements
		SRR – System Requirements Review
		3. System Design
		4. Software Planning
		SDR – System Design Review

		CSCI A	CSCI B	CSCI C
Elaboration	Iteration 1	Detailed Iteration Planning		
		Iteration Plan Review		
		5. Software Requirements	5. Software Requirements	5. Software Requirements
		SSR – Software Specification Review(s)		
		6. Software Design	6. Software Design	6. Software Design
		SDR – Software Design Review(s)		
		7. Coding and Unit Testing	7. Coding and Unit Testing	7. Coding and Unit Testing
		8. CSCI Validation	8. CSCI Validation	8. CSCI Validation
		Iteration Completion Review		
	Iteration 2	Detailed Iteration Planning		
		Iteration Plan Review		
		5. Software Requirements	5. Software Requirements	5. Software Requirements
		SSR – Software Specification Review(s)		
		6. Software Design	6. Software Design	6. Software Design
		SDR – Software Design Review(s)		
		7. Coding and Unit Testing	7. Coding and Unit Testing	7. Coding and Unit Testing
		8. CSCI Validation	8. CSCI Validation	8. CSCI Validation
		Iteration Completion Review		
Construction	Iteration 3	Detailed Iteration Planning		
		Iteration Plan Review		
		5. Software Requirements	5. Software Requirements	5. Software Requirements
		SSR – Software Specification Review(s)		
		6. Software Design	6. Software Design	6. Software Design
		SDR – Software Design Review(s)		
		7. Coding and Unit Testing	7. Coding and Unit Testing	7. Coding and Unit Testing
		8. CSCI Validation	8. CSCI Validation	8. CSCI Validation
		Iteration Completion Review		
	Iteration 4	Detailed Iteration Planning		
		Iteration Plan Review		
		5. Software Requirements	5. Software Requirements	5. Software Requirements
		SSR – Software Specification Review(s)		
		6. Software Design	6. Software Design	6. Software Design
		SDR – Software Design Review(s)		
		7. Coding and Unit Testing	7. Coding and Unit Testing	7. Coding and Unit Testing
		8. CSCI Validation	8. CSCI Validation	8. CSCI Validation
		Iteration Completion Review		

5 Conclusions

During the last couple of years the iterative software development process has been applied to a small number of pilot projects. The experience gained through these projects has been analyzed and recorded into a formal process documentation, which is currently in the process of discussion and formal approval.

The results reported so far show significant satisfaction of engineers, managers and clients, who feel more confidence in the product, which is obtained by having operational demonstration of parts of the system in early stages of the project. The early mitigation of major risks play a major role in the adoption of the iterative development process to an increasing number of projects.

6 Bibliography

[1] W. W. Royce, Managing the development of large software systems: Concepts and techniques, 1970 WESCON Technical Paper, Western Electronic Show and Convention, Los Angeles, 1970, pp. A/1-1–A/1-9, reprinted in: *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, May 1989, pp. 328–38.

[2] DOD-STD-2167A, Military Standard, Defense System Software Development, February 1988.

[3] MIL-STD-498, Military Standard, Software Development and Documentation, December 1994.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, Reading, MA, 1999

[5] I. Jacobson, M. Ericsson and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology*, Addison-Wesley, Reading, MA, 1995.

[6] P. Kruchten, *The Rational Unified Process – an Introduction*, Addison-Wesley, Reading, MA, 1998.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison Wesley, Reading, MA, 1999.

[8] OMG Unified Modeling Language Specification, Version 1.4, September, 2001.

[9] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000, IEEE Computer Society, September 2000.