Integrated Requirements Baseline Management for Complex Software System

Description of Methods and Tools for improving the Software Modules Design processes for complex Systems in a Project Management and System Engineering Integrated Environment

Sergio Funtò
Engineering Ingegneria Informatica S.p.A.
Via S. Martino della Battaglia 56 Rome, Italy
sergio.funto@eng.it

***Abstract*—**This paper reports on experiences from managing the requirements baseline in regards to Complex Software System. The requirements management of these starts from the architectural structure of the whole system by defining the high-level functionalities. The complex systems are architecturally organized in separate modules and each of them gives support to the development of one or more functionalities of the whole systems. The flow down of the whole system requirements towards the requirements of each module and the monitoring of the related traceability are the core activities within the baseline management. Moreover, the development plan of a complex system foreseen more than one deliveries each one characterized by new features and functionalities compared to the previous one. Each system version is defined by the set of the corresponding modules. This scenario requires a project development environment where Project Management (PM) and System Engineering (SM) activities are strictly connected and integrated. The presented approach takes into account the system development life cycle identified by quality standard adopted for the software development and documentation. Adopted methods, tools and artifacts are presented in order to describe the proposed processes taking into account PM and SE activities integration mechanisms episodic and pervasive.

Keywords—Requirements Management Processes, Scope Management, Project Management and System Engineering Integration

I. INTRODUCTION

This paper presents experiences from defining and managing the requirements baseline related to Complex Software System (CSS).

Starting from the general definition of a Complex System, are introduced the main concepts and elements to be taken into account in order to describe the proposed methods and tools.

In this regards, a specific focus is dedicated to identify:

- the design process/model adopted as references for the system development process;
- the artifacts and documents used to describes the adopted software development life-cycle;
- the methods and tools used in regards to the Project Management (PM) and System Engineering (SM) integrated activities to support the design process phases.

The next Section II describes the background of the work, and introduces the Military Standard 498 (MIL-STD-498), the Waterfall Model and the referred integration mechanism in regards to PM-SE activities.

Section III introduces the operative scenario used as reference to describe the proposed requirements baseline methods and tools as reported in Section IV.

II. BACKGROUND

A. *What is a Complex Software Systems*

The definition of a CSS is strictly related to the identification of the properties of a Complex System and to the features of a Software System.

1) The Complex Systems

A *Complex System* is composed of many components which may interact with each other. In many cases it is useful to represent such a system as a network where the nodes represent the components and the links their interactions. The behavior of a Complex Systems is intrinsically difficult to model due to the dependencies, relationships, or interactions between their parts or between a given system and its environment. Systems that are *complex* have distinct properties that arise from these relationships, such as: non-linearity, emergence, spontaneous order, adaptation, and feedback loops, among others. Because such systems appear in a wide variety of fields, the commonalities among them have become the topic of their own independent area of research.

2) Properties of a Complex Systems

Abstracting from the quotations related to Complex Systems and drawing on the culture of complexity science as expressed through a wide range of popular as well as academic sources, the following list of properties can be associated with the idea of a complex system [1]:

- *Nonlinearity* - is often considered to be essential for complexity. A system is linear if one can add any two solutions to the equations that describe it and obtain another, and multiply any

solution by any factor and obtain another. Nonlinearity means that this superposition principle does not apply;

- *Feedback* - is an important necessary condition for complex dynamical systems. A part of a system receives feedback when the way its neighbors interact with it at a later time depends on how it interacts with them at an earlier time;

- *Spontaneous order* - given the above it is clear that a fundamental idea in complex systems research is that of order in a system's behavior that arises from the aggregate of a very large number of uncoordinated interactions between elements;

- *Robustness and lack of central control* - the order in complex systems is said to be robust because, being distributed and not centrally produced, it is stable under perturbations of the system. A centrally controlled system on the other hand is vulnerable to the malfunction of a few key components;

- *Emergence* - is a notoriously murky notion with a long history in the philosophy of science. People talking about complexity science often associate it with the limitations of reductionism. A strong, perhaps the strongest, notion of emergence is that emergent objects, properties or processes exhibit something called 'downwards causation';

- *Hierarchical organization* - In complex systems there are often many levels of organization that can be thought of as forming a hierarchy of system and sub-system. Emergence occurs because order that arises from interactions among parts at a lower level is robust;

- *Numerosity* - the kind of hierarchical organization that emerges and gives rise to all the features listed above, only exists if the system consists of a large number of parts, and usually, only if they are engaged in many interactions;

- *Remarks* - The above discussion makes it clear that the definition of complexity and complex systems is not straightforward and is potentially philosophically interesting. The notions of order and organization introduced above and the idea of feedback are suggestive of an information-theoretic approach to complexity, since complex systems can be often helpfully be construed as maintaining their order and hierarchical organization by the exchange of information among their parts.

3) *The features of the Software Systems*

A *Software System* is characterized by inter communicating components based on software forming part of a computer system (a combination of hardware and software). It consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system.

The term *Software System* should be distinguished from the terms 'computer program' and 'software'. The term computer program generally refers to a set of instructions (source, or object code) that perform a specific task. However, a software system generally refers to a more encompassing concept with many more components such as specification, test results, end-user documentation, maintenance records, etc.

The use of the term software system is at times related to the application of systems theory approaches in the context of software engineering. A software system consists of several separate computer programs and associated configuration files, documentation, etc., that operate together [2]. The concept is used in the study of large and complex software, because it focuses on the major components of software and their interactions. It is also related to the field of software architecture.

B. *The Software System Design Process*

Once introduced the definition and the characteristics/properties of the CSS, the next step is to identify the adopted system design process.

1) *The Waterfall Model*

The *Waterfall Model* can be described through a sequential (non-iterative) design process, used in software development for large systems, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production, implementation and maintenance.

In the original waterfall model [3], the following phases are followed in order: *System and software requirements*: captured in a product requirements document; *Analysis*: resulting in models, schema, and business rules; *Design*: resulting in the software architecture; *Coding*: the development, proving, and integration of software; *Testing*: the systematic discovery and debugging of defects; *Operations*: the installation, migration, support, and maintenance of complete systems.

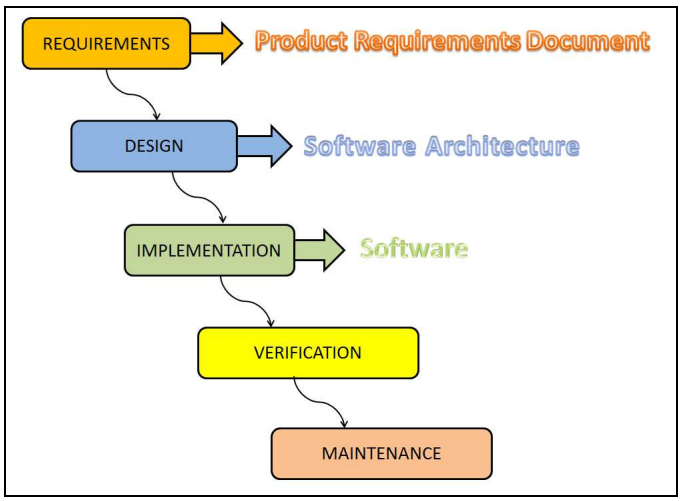

Figure II-1: The Waterfall phases [3]

The United States Department of Defense (DOD) captured this approach in the DOD-STD-2167A, their standards for working with software development contractors, which stated that "the contractor shall implement a software development cycle that includes the following six phases: Preliminary Design, Detailed Design, Coding and Unit Testing, Integration, and Testing. The DOD-STD-2167A is the precursor of the MIL-STD-498 introduced hereafter.

C. The Software System Description

Following the identification of the adopted system design process, another core step is the specification of the artifacts and documents used to describes the adopted software development life-cycle.

1) The Military Standard 498

The Military-Standard-498 (MIL-STD-498) is a United States military standard whose purpose was to establish uniform requirements for software development and documentation. It replaced the DOD-STD-2167A, DOD-STD-7935A, and DOD-STD-1703. It was meant as an interim standard, to be in effect for about two years until a commercial standard was developed.

Unlike previous efforts the MIL-STD-498 was the first attempt at a truly comprehensive description of the systems development life-cycle. It was the baseline that all of the ISO, IEEE, and related efforts after it replaced. It also contains much of the material that the subsequent professionalization of project management covered in the *Project Management Body of Knowledge* (PMBOK) [7].

2) MIL-STD-498 Data Idem Descriptions

The MIL-STD-498 standard specifies [4] the development and documentation in terms of 22 Data Item Descriptions (DIDs) from which an effort will select to conduct the system development and support efforts. Each DID generically describes the required content of a data item, a file or document that describes the system or some aspect of the system life-cycle. These documents could take many forms, from source code, to installation scripts, to various electronic and paper reports, and the contracting party (e.g., the government) is encouraged to specify acceptable formats. Any data item description is tailored for a specific contract, meaning sections not desired for a particular effort are identified as not to be provided as part of identifying the Contract Data Requirements List (CDRL) of what items are to be produced and delivered by a contractor. Exactly which DIDs and what parts of the DIDs are required for a particular system depends on the nature of the project and how parts of it are being produced by contract(s).

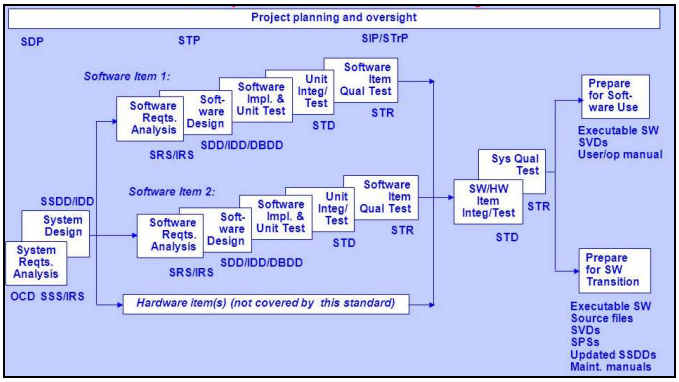

Figure II-2: MIL-STD-498 DID [4]

3) Data Idem Description related to System Specification

The MIL-STD-498 DIDs covers all the phases of a software system development life-cycle. In regards to system design and requirements specification, the involved DIDs are listed hereafter:

- *Operational Concept Description (OCD)*: describes a proposed system in terms of the user needs it will fulfill, its relationship to existing systems or procedures, and the ways it will be used. The OCD is used to obtain consensus among the acquirer, developer, support, and user agencies on the operational concept of a proposed system;

- *System/Subsystem Specification (SSS)*: specifies the requirements for a system or subsystem and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the system or subsystem's external interfaces may be presented in the SSS or in one or more Interface Requirements Specifications (IRSs) referenced from the SSS;

- *System/Subsystem Design Description (SSDD)*: describes the sytem- or subsystem-wide design and architectural design of a system or subsystem. The SSDD may be supplemented by Interface Design Descriptions (IDDs) and Database Design Descriptions (DBDDs);

- *Interface Requirements Specification (IRS)*: specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCIs), Computer Software Configuration Items (CSCIs), manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces.

D. PM and SE Integrated Environment

The design and development of a CSS, taking into accounts the elements introduced before, takes advantage from a working environment where Project Management (PM) and System Engineering (SE) activity are strictly integrated.

All aspects of integration are about individuals and how they coordinate the application of their collective knowledge, expertise and capabilities to deliver results. Effective integration efforts are accomplished through the application of processes, practices and tools that help to enable important abilities [5]:

- enable communication and common understanding on key objectives,
- provide framework for defining specific work activities,
- document approaches for coordinating and tracking work effort;
- establish expectation of each person's contribution;
- identify critical point where focalized the work effort;
- facilitate problem identification and resolution;
- apply generally accepted approaches that have demonstrate effective results under similar circumstances in the past;
- support and accelerate the accomplishment of specific work activities.

Some processes, practices and tools are designed for individual use while others may be structured fro group activities. Both uses have appropriate application within complex program.

The process, practices and tools can be organized by the timeline of their impact on integration: episodic or pervasive [5]. Episodic

integration emerges as the need requires. Pervasive integration tends to be synchronous with the daily work of the program or its component project.

1) Episodic Integration Mechanism

Episodic integration mechanisms are applied occasionally to specific activities or at specific intervals within a program/project:

- *Program Gate Reviews*: require that all program aspects (cost, schedule, performance, risks, requirements, testing) be presented in their current state of maturity individual gates in order to: receive approval to proceed to the next project gate (Go), repeat the review after addressing specific concerns (recycle) or terminate the project (No Go);

- *Joint Planning*: There are a variety of tools, templates and software application that organizations can use to support scoping and planning activities. Moreover, there are a number of planning related practices that help to integrate PM and SE. These include: *program kick-off workshops* (that brings together all key stakeholder including engineers, project managers and factory teams) and *model based program planning* (artifacts describing the program/project including product breakdown structure, work breakdown structure, system engineering process models);

- *Dedicated Team Meeting Space*: The creation and use of dedicated team meeting space and stand-up meeting is a proven process in a variety of domains;

- *Pulsed Product Integration and Iterative Development*: (PPIID) it is sometime described as the 'daily build' of product components into more complex component or into complete products. Iterative development comprises the use of short cycles to create and deliver product increments.

2) Pervasive Integration Mechanism

There are process, practices and tools integral to program design and development. They are continuous in nature and thus are 'pervasive'. The application of pervasive integration mechanism can help program teams realize such potential benefits for their organization:

- *Standard, Methodologies and Assessment*: A methodology is a documented approach for integrating interacting or interdependent practices, techniques, procedures, and rules to determine how best to plan, develop, control and deliver a defined objective. A standard, on the other hand, reflects broadly accepted principles of what represents good practices or common guidelines. As the methodology is implemented, executive leaders and user must evaluate (assess) the specific practices and the extent to which the company is adhering to its methodology;

- *Integrated Product and Process Development*: (IPPD) also known as 'simultaneous engineering' or 'design build', uses multidisciplinary co-located teams in design to jointly derive requirements and schedules with equal emphasis on product and process development. The teams often use requirement breakdown structure (RBS) and work breakdown structure (WBS) to facilitate their scoping and planning activities. RBS are used where complex system dictate significant attention be paid to requirements and when integration is crucial. The WBS serves as the key framework for organizing the program and the system engineering effort as well as for estimating and allocating cost, schedule and performance requirements;

- *Work Design Process*: work design process as *configuration management* can help to increase communication and collaboration across the program. Another work design process is standardized work that foreseen rigorous design standardization supports platform reusability. The work processes are deliberately designed so that integration is a natural outcome of the work itself;

- *Requirements Management*: it forces conversation between program/project manager and systems engineers. Effective requirements management practices helps to align the work so that customer receive ideal solution and desired program/project benefits and value is realized for the business. Requirements management often start at the concept level as a high level view associated with investment or business opportunities. The project manager and the chief system engineer build on the hight level view by eliciting, documenting, and validating high level requirements from customer and stakeholders. The project may further cascade elements of the high level requirements for more detailed development;

- *Risk Management*: Effective risk management ensure that the sponsoring organization realized its desired benefits. Ideally risk management processes should be fully integrated into all program/project activities (management, technical, design, development, procurement, planning;

- *Technical Performance Measurements*: is an analysis and control technique that is used to anticipate the probable performance of a selected technical parameter, record the actual performance observed of the selected parameter and assist the manager in decision making through comparison of planned versus actual performance;

- *Governance*: is a structured mechanism through which individuals with oversight responsibility and authority provide guidance and decision making for important organizational activities. The governance serves the following key functions [6]: Oversight, Control, Integration and Decision Making.

III. The Operative Environment Scenario

The previous Section II has introduced the main elements that characterized the proposed system requirements baseline management. This section specifies the environment scenario related to the target program/project.

A. The Complex Software System Decomposition

According to the Software Design DIDs identified by the MIL-STD-498, a first description of the Complex System is given in the *OCD* (see Section II.C.2). Taking into account the *Pulsed Product Integration and Iterative Development* (PPIID) and the *Integrated Product and Process Developmen*t (IPPD) integration mechanism (see Section II.D.2), the first requirements breakdown consist in the decomposition of the complex system OCD in three different OCD in regards to complex system: *Hardware:* user needs related to hardware elements; *Software*: user needs related to software

functionalities; and *Interfaces*: requirements related to the external interfaces to be managed.

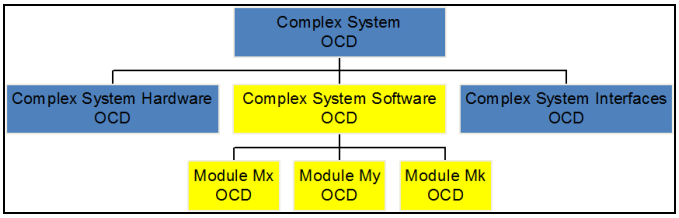

Figure III-1: Complex System Decomposition

Taking into account only the *Software component* the further breakdown of the related OCD (applying PPIID and IPPD) is given by the definition of the OCDs related to the single software modules that compose the this component of the complex system.

B. *The Software System Description*

Each software module from the design point of view is fully defined by (see Section II.2) by the following artifacts (documents and models): *System/Subsystem Specification* (SSS), *System/Subsystem Design Description* (SSDD) and *Interface Requirements Specification* (IRS).

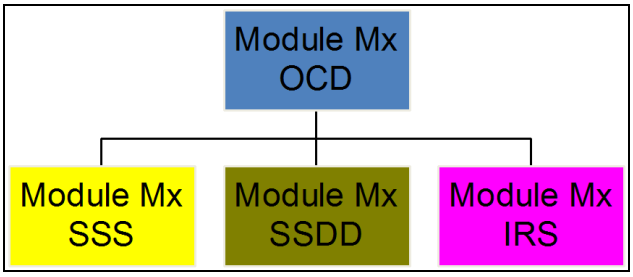

Figure III-2: Software System Module Design

Specifically the IRS related to each software module is part of the breakdown of the CID related to the CSS Interfaces. In other terms the CID relates to the interface of the global system are "implemented" through the IRS of each module of the software component of the system itself.

This requirements organization takes into account the *Requirements Management* integration mechanism (see Section II.D.2).

C. *The Design Process*

As stated in the Section II.B.1, the Waterfall model approach was captured in the MIL-STD-498. The related DIDs have an immediate correspondence in the adopted system software life-cycle (see Figure III-3) identified within the model in regards to the *Work Design Process* integration mechanism (see Section II.D.2).

Specifically, the system requirements baseline management is focused on the System Engineering activities (see Figure III-4) related to: user requirements Analysis (described in the CID), System Requirements Analysis (defined in the SSS/IRS) and System Design (detailed in the SSDD).

Note: The Software Engineering activities are reported in the corresponding DIDs: Software Requirements Specification (SRS), Software Design Description (SDD), Interface Data Description (IDD).

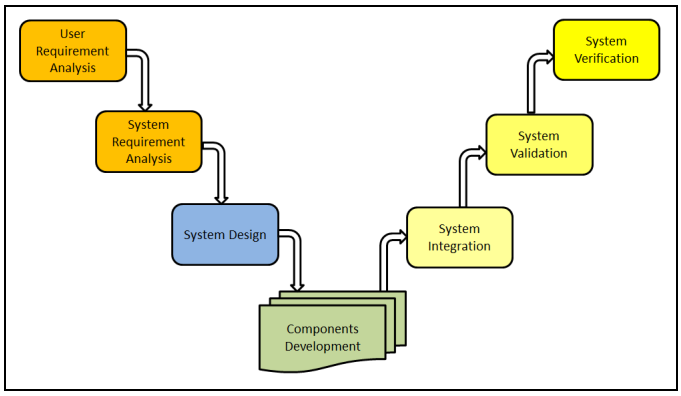

Figure III-3: System Software adopted life-cycle

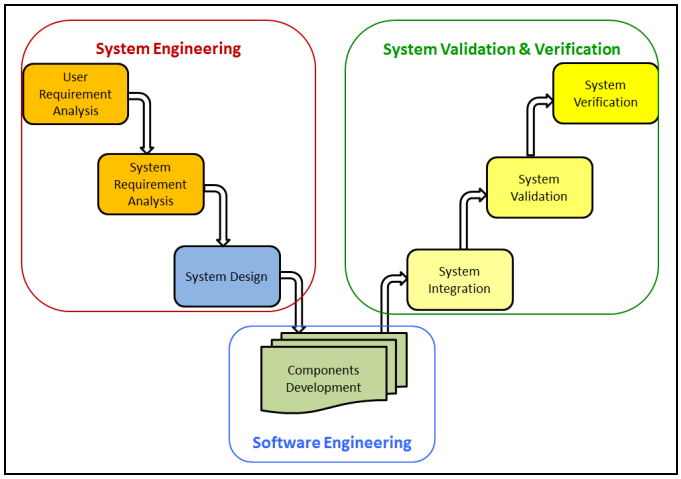

Figure III-4: Software System life-cycle activities

Additional input to the System Engineering comes from the System Validation and Verification activities where, for example, failures in the system behavior are related to anomalies or errors in the system specification and design (see next Section III.D for further details).

Moreover, also within Software Requirements baseline management uses MIL-STD-498 documents managed in the Software Engineering activities (see [4]).

D. *System Versions approach*

Starting from the complexity of the system and the related functionalities and taking into account the suggestions of the integration mechanism (see section II.D), the approach foreseen more than one system delivery according to (contractual) milestone each one corresponding to functional increments of the system according to the following schema:

- the Milestone ***Mk*** foreseen (at ***Tk***) the delivery of the version ***Vk*** of the system according to the documents: ***SSSk***, ***SSDDk***, ***IRSk***;

- after the *Mk*, the activities related to the delivery of the milestone ***Mk+1*** are started in order to deliver the version ***Vk+1*** at ***Tk+1***. This is related to the corresponding evolution (*k→k+1*) of the systems design and specification (SSS, SSDD and IRS).

This versions approach schema taken into account in the RBS and in the WBS of the project (see *Pervasive Integration Mechanism*, Section II.D.2).

Within the *k→k+1* evolution, in addition to the activities related to the design and specification of the functional increments of the system, are taken into account the following input, including outcomes from System Verification and Validation activities:

- failures coming from the final customer highlighted during the system verification activities on the ***Vk*** version;
- not blocking remarks on the ***Vk*** version coming from the system validation team;
- evolution of the user needs (e.g new operating rules, blaws, STANAG,..).

1) Impact on Software System life-cycle

As reported in the previous Section III.C, the adopted Software System life-cycle corresponds to a sequential process. At the same time, the System Versions approach foresees activities iterations each one corresponding to a specific version.

This results in sequence of design phases corresponding to the System Versions where a specific phase takes inputs from the previous one and gives input to the next one. The activities foreseen for each phase are the same (according to Waterfall model):

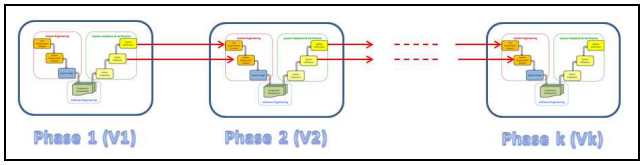

Figure III-5: System Versions design phases

IV. REQUIREMENTS BASELINE MANAGEMENT

Starting from the Operative Environment Scenario identified in Section III and taking into account the PM-SE integration mechanisms specified in Section II.D, this section reports the methods and tools related to the proposed requirements management activities.

The management of the requirements baseline in regards to complex software is focused on the concept of the changes management embedded on the system versions approach (see Section III.D). Within the V*k→Vk+1* evolution of the system, the corresponding functional increment is specified by the upgrade of the Requirements Baseline (RB): *RBk→RBk+1*. This requirements evolution is regulated by a set of change requests managed through an integrated change control.

A. Integrated Change Control

Perform Integrated Change Control [7] is the process of:

- reviewing all change requests;
- approving changes and managing changes deliverables, project technical documents and project plans;
- communicating their disposition.

It reviews all requests for changes or modification to project documents, deliverables, baselines and project management plan. Moreover, it approves or rejects the changes. The key benefit of this process is that it allows the documented changes within the project to be considered in an integrated fashion while reducing project risk, which often arises from changes made without consideration to the overall project objectives and/or plans.

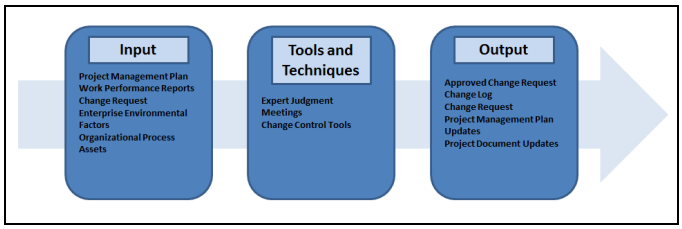

Figure IV-1: Integrated Change Control – Inputs, Tools and Techniques and Outputs [7]

Changes may be requested by any stakeholder involved in the project. They should be recorded in written form and entered into the change management and/or configuration management system. Change requests are subject to the process specified in the change control and configuration control system. Those change request process may require information on estimated time impacts and estimated cost impacts.

1) Change Control Board

Every documented change request need to be either approved or rejected by a responsible individual (project or program manager). The integrated change control process includes a Change Control Board (CCB) which is a formally chartered group responsible for reviewing, evaluating, approving, delaying or rejecting changes and for recording and communicating such decisions. Approved change requests can require new or revised cost estimates, activity sequences, schedule dates, resource requirements and analysis of risk response alternatives. Customer/sponsor approval may be required for certain changes after the CCB approval, unless they are part of the CCB.

2) Configuration Control

Configuration Control is focused on the specification of both the deliverable and the process, while change control is focused on identifying, documenting and approving or rejecting changes to the project documents, deliverables and baselines.

Some of the configuration management activities included in the integrated change control are configuration:

- identification: identification and selection of a configuration item to provide the basis for which the product configuration is defined and verified, products and documents are labeled, changes requests are managed and accountability is maintained;

- status accounting: information is recorded and reported as to when appropriate data about the configuration item should be provided;
- verification and audit: ensure the composition of a project's configuration item is correct and that corresponding changes are registered assessed, approved, tracked and correctly implemented.

B. *Change Request Description*

According to previous Section III.D and Section IV.A, the change requests are documented through an artifacts named Change Request Description (CRD).

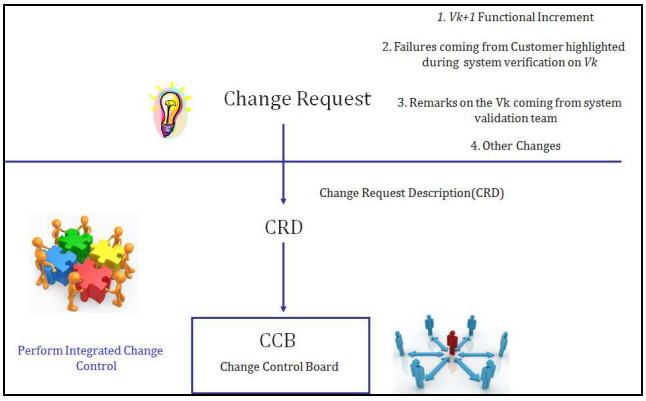

Figure IV-2: CRD Management

Each Change Request on the Vk version of the system (see Section III.D) is related to: a function increment requested for the *Vk+1* version, failures coming from the customer on the *Vk* version, non-blocking remarks on the *Vk* version, other needs.

Each CRD is labeled with an Identifier (CRD_ID) defined according to the configuration control and report the following information:

- the authors of the CRD;
- the figures in the project team responsible for the acceptance of the CRD;
- the module of the system affected (see Section III.A);
- the deliverables, documents and artifacts affected by the changes (if approved);
- identification of the items affected by the change in regards to design documents (see Section II.C.3) and artifacts (e.g. models);
- detail description of the reasons related to the change;
- detailed description of the proposed changes (requirements, models, drawings).

C. *Approving the CRD*

Each Change Request on the *Vk* version of the module *Mx*, described by the corresponding CRD, in order to be approved for application to the *Vk+1* version, must to be reviewed by the involved individuals (specified in the CRD itself) within a CCB:

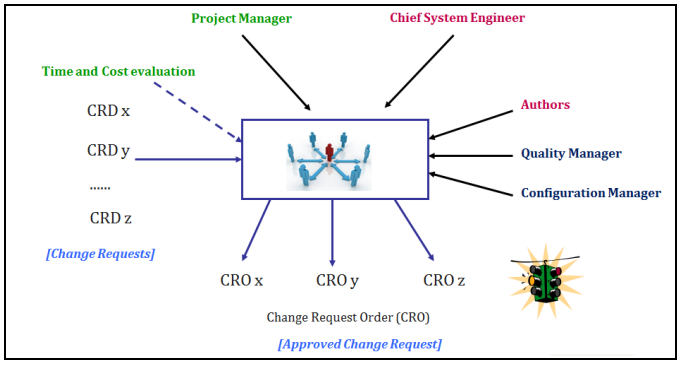

Figure IV-3: Approving CRD through CCB

An important input to the CCB, in addition to the CRD to be reviewed, is the time and cost impact evaluation for each change request in regards to the implementation of the changes on the version *Vk+1* of the affected system module. These evaluations must to take into account system engineering and system verification and validation activities (see Section III.C).

The main outcomes of the CCB are the approved Change Requests that became Change Request Order (CRO) applicable to the *Vk+1* version.

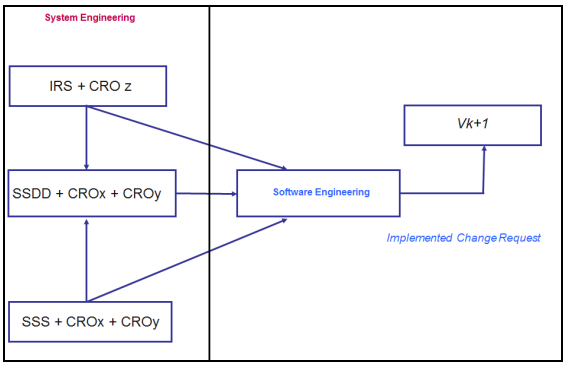

Figure IV-4: Software baseline evolution according CRO

Each CRO must to be applied to the system description documents (see Section II.C.3) in order to define the new software baseline, as defined in the CCBs, to be applied by the software engineering activities.

D. *System Failure Description*

The Change Request mechanism is applied also to the management of the System Failure (SF) highlighted during the system verification and validation phase of the specific version (*Vk*) on one module (*Mx*).

Similar to the change request, the system failures are documented through a System Failure Description (SFD). Each SFD is labeled with an Identifier (SFD_ID) defined according to the configuration

control and reports at list: the authors, the module of the affected system (see Section III.A), identification of the functionalities affected by the failure, detailed description of the failure including the operative condition and references to supporting artifacts.

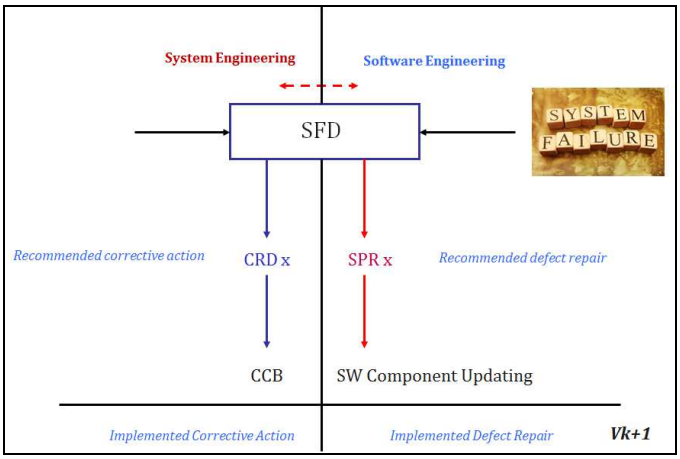

Figure IV-5: System Failure vs Change Request

Each system failure may be related to:

1. an anomaly of the system specifications or design that affect the SSS/SSDD or IRS (see Section II.C.3).

2. an anomaly of the software specifications/design or implementation.

In the first case, starting from the SFD is produced a change request (documented by a CRD that specifies the *recommended corrective action* [7]) to be reviewed/approved in a CCB. The corresponding CRO specifies the *implemented corrective action*.

In the second case, a Software Problem Report (SPR) is defined in order to trace the *recommended defect repair* that will be implemented in the Vk+1 version of the affect module of the system.

V. Conclusion

The paper presents experiences from defining and managing the requirements baseline related to Complex Software System (CSS).

The definition of a CSS is strictly related to the identification of the properties of a Complex System and to the features of a Software System. A complex system is composed of many components which may interact with each other. A software system is based on inter communicating components based on software forming part of a computer system (a combination of hardware and software).

Starting from the identification of the features of a CSS, it is identified the adopted system design process, the Waterfall Model, and the artifacts and documents used to describes the software development life-cycle. In this regards, it is introduced the MIL-STD-498 Data Item Descriptions (DIDs) that covers all the identified phases giving a focus on the DIDs involved in the system design and requirements specification.

The design and development of a CSS takes advantage from the working environment where Project Management (PM) and System Engineering (SE) activity are strictly integrated. All aspects of integration are about individuals and how they coordinate the application of their collective knowledge, expertise and capabilities to deliver results. Effective integration efforts are accomplished through the application of processes, practices and tools These can be organized by the timeline of their impact on integration: episodic or pervasive. Episodic integration emerges as the need requires. Pervasive integration tends to be synchronous with the daily work of the program or its component project.

Starting from a first description of the CSS (given in the *Operational Concept Description OCD*) is introduced the proposed environment scenario related to the target, taking into account the *Pulsed Product Integration and Iterative Development* (PPIID) and the *Integrated Product and Process Development* (IPPD) integration mechanism. The output of this process is the identification of the software modules that compose the software component of the CSS.

The proposed System Version approach (based on the suggestions of the integration mechanism) foreseen more than one system delivery according to (contractual) milestones each one corresponding to functional increments of the system. This results in sequence of design phases corresponding to the System Versions where a specific phase takes inputs from the previous one and gives input to the next one.

The management of the requirements baseline in regards to CSS is focused on the concept of the changes management embedded on the system versions approach. Within each System Version evolution, the corresponding functional increment is specified by the corresponding upgrade of the Requirements Baseline. This requirements evolution is regulated by a set of change requests managed through an integrated change control.

VI. References

1. James Ladyman, James Lambert (Department of Philosophy, University of Bristol, U.K), Karoline Wiesner (Department of Mathematics and Centre for Complexity Sciences, University of Bristol, U.K.) - “What is a Complex System” (2012).

2. Sommerville, Ian - "What is software?". Software Engineering, 8th ed. (2007).

3. Winston Royce - "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON (1970).

4. Joint Logistics Commanders and Joint Policy Coordinating Group on Computer Resources Management - “Military Standard 498 (MIL-STD-498) Overview and Tailoring Guidebook” (1994);

5. Eric Rebentish, PMI, INCOSE – “Integrating Program Management and System Engineering. Methods, Tools and Organizational Systems for Improving Performance”, Wiley (2017);

6. PMI – “Governance of Portfolio, Program and Projects: A Practice Guide” (2016);

7. PMI – “A guide to the Project Management Body of Knowledge” Fifth Edition (2013);