Deployment Package
Design with Security Considerations
Basic Profile + Security

Notes:

This document is the intellectual propriety of its author's organization. However, information contained in this document is free of use.

This material is furnished on an "as-is" basis. The author(s) make(s) no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material.

The processes described in this Deployment Package are not intended to preclude or discourage the use of additional processes that Very Small Entities may find useful.

Authors	P. Maciel – CIMAT A.C. Jezreel Mejía – CIMAT A.C. (México)
Editors	Perla Maciel – CIMAT AC. (México) Jezreel Mejía – CIMAT A.C. (México)
Creation date	24/06/21
Last update	24/06/21
Version	0.1

Version History

Date	Version	Description	Author
2009-07-08	0.1	Document creation	Perla Maciel

Abbreviations/Acronyms

Abre./Acro.	Definitions
DP	Deployment Package - a set of artefacts developed to facilitate the implementation of a set of practices, of the selected framework, in a Very Small Entity.
VSE	Very Small Entity – an enterprise, organization, department or project having up to 25 people.
VSEs	Very Small Entities
TL	Technical Leader
AN	Analyst
DES	Designer
SA	Security Advisor

Table of Contents

1. Technical Description

Purpose of this document

This Deployment Package (DP) supports the Basic Profile as defined in ISO/IEC 29110 Part 5-1-2: Management and engineering guide[1]. The Basic Profile is one profile of the Generic profile group. The Generic profile group is composed of 4 profiles: Entry, Basic, Intermediate and Advanced. The Generic profile group is applicable to VSEs that do not develop critical software. The Generic profile group does not imply any specific application domain.

A DP is a set of artefacts developed to facilitate the implementation of a set of practices in a Very Small Entity (VSE). A DP is not a process reference model (i.e. it is not prescriptive). The elements of a typical DP are: description of processes, activities, tasks, roles and products, template, checklist, example and reference to standards and models, and tools.

The purpose of this document, entitled "*Deployment Package - Software Architectural and Detailed Design*" is to provide VSEs with tailorable and easily usable guidelines and materials in order to implement a good *Software Design*.

The content of this document is entirely *informative*.

This document has been produced by Perla Maciel and Jezreel Mejía of CIMAT A.C. (México).

Why Design with Security Considerations?

Investing effort in the design **activity** ensures that the proposed solution (e.g. software to be built) will have been given some thought prior to implementation (e.g., coding). Building something without designing it typically yields a solution that doesn't meet the requirements, is delivered late, exceeds the budget or is of poor quality.

Investing effort in explicitly **documenting** the design enables communication and negotiation among project stakeholders, more specifically those that have an interest in the design. Capturing a design in some form (electronic document, paper document, models,...) is not only useful while a software project is active, but also for future maintenance and enhancements.

The figure below presents data from a real company. It shows that more than 20% of the defects are produced during the design phase.

The Software Architectural and Detailed Design activity produces a document termed the *Software Design* that enables stakeholders to understand the interactions in the software, and the tracing of design elements to the requirements. This provides a way to verify that each requirement has been addressed (e.g., design completeness). The Software Design is also used when maintaining software because it describes the components and their interfaces.

[2] ISO/IEC 29110-5-1-2 is available at no cost from ISO: **http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html**

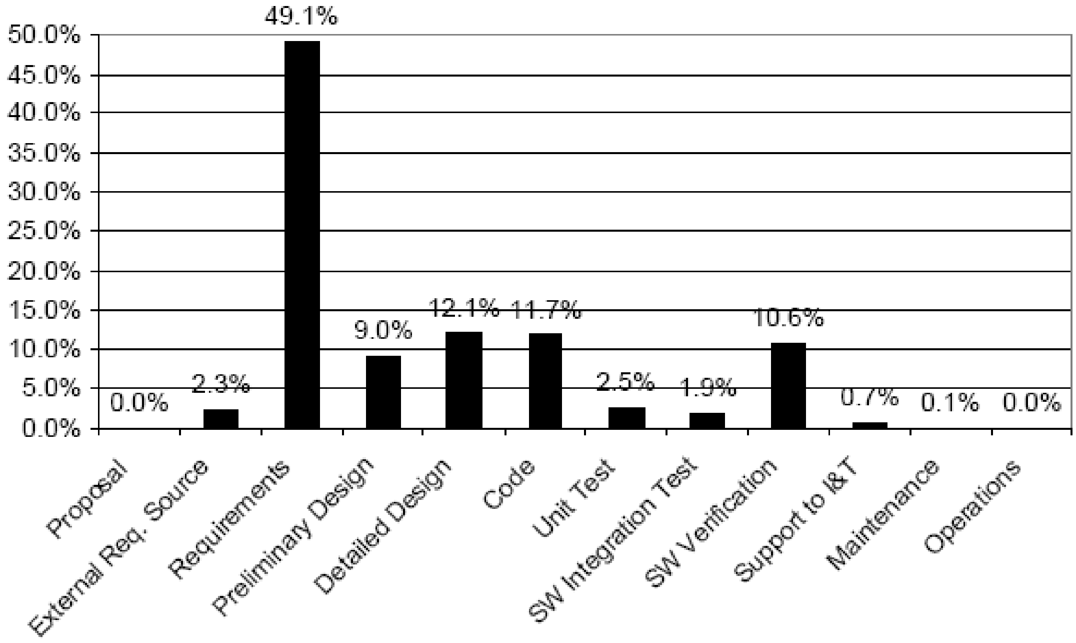


Figure 1 Origins of software defects ([2])

[2] Selby, P., Selby, R.W., Measurement-Driven Systems Engineering Using Six Sigma Techniques to Improve Software Defect Detection, Proceedings of 17th International Symposium, INCOSE, June 2007, San Diego.

2. Definitions

In this section, the reader will find two sets of definitions. The first set defines the terms used in all Deployment Packages, i.e. generic terms. The second set defines terms used in this Deployment package, i.e. specific terms.

Generic Terms

Process: set of interrelated or interacting activities which transform inputs into outputs [ISO/IEC 12207].

Activity: a set of cohesive tasks of a process [ISO/IEC 12207].

Task: required, recommended, or permissible action, intended to contribute to the achievement of one or more outcomes of a process [ISO/IEC 12207].

Sub-Task: When a task is complex, it is divided into sub-tasks.

Step: In a deployment package, a task is decomposed in a sequence of steps.

Role: a defined function to be performed by a project team member, such as testing, filing, inspecting, coding. [ISO/IEC 24765]

Product: piece of information or deliverable that can be produced (not mandatory) by one or several tasks. *(e. g. design document, source code)*.

Artefact: information, which is not listed in ISO/IEC 29110 Part 5, but can help a VSE during the execution of a project.

Specific Terms:

Cryptographic design: Cryptographic technologies such as encryption, digital signatures, key management, and secret sharing schemes are critical elements in the implementation of any security service. [Chan Yeob Yeun, Khalifa University]

Threat Model: *Steps to identify targets and vulnerabilities, identify measures to prevent or minimize the impact of threats on your system, and optimize the security of an application, system, or process Your Business*. [Michael Cobb]

3. Tasks, Steps, Roles and Products in ISO/IEC29110

Process: Software Implementation (SI)

As defined in ISO/IEC 29110, the purpose of the Software Implementation process is the systematic performance of the analysis, design, construction, integration and tests activities for new or modified software products according to the specified requirements.

Activity: SI.3 Software Architectural and Detailed Design

The Software Architectural and Detailed Design activity transforms the software requirements to the system software architecture and software detailed design.

Task List	**Roles**
SI.3.3 Document or update the Software Design	**AN, DES**

Tasks

- **Satisfy Minimum Cryptographic Design Requirements**
- **Complete Threat Models**

Satisfy Minimum Cryptographic Design Requirements

Objectives:	Satisfy the minimal cryptographic design requirements established for your product when you established Security Requirements.
Roles:	AN – Analyst
	DES – Designer
	SA – Security Advisor
Products:	Software Design
	Requirement specifications
	Test Cases and Test Procedures
	Software Configuration
	Validation Results
Artifacts:	Minimum Cryptographic Design
Steps:	1. Identify all the Cryptographic Standards
	2. Select the Cryptographic Standards that apply to the project
	3. Document the minimum Cryptographic Standards to consider in the Design phase
Step Description:	**Step 1. Identify all the Cryptographic Standards** Identify all the Cryptographic Standards that can be introduced in the project, i.e. as the Microsoft Cryptographic Standards for SDL-covered products, at high-level are: • Use AES for symmetric enc/dec.

	• Use 128-bit or better symmetric keys. • Use RSA for asymmetric enc/dec and signatures. • Use 2048-bit or better RSA keys. • Use SHA-256 or better for hashing and message-authentication codes. • Support certificate revocation. • Limit lifetimes for symmetric keys and asymmetric keys without associated certificates. • Support cryptographically secure versions of SSL (must not support SSL v2). • Use cryptographic certificates reasonably and choose reasonable certificate validity periods. ***Step 2.* Select the Cryptographic Standards that apply to the project** After Identifying all the Cryptographic Standard that apply to the project, a minimum must be selected and specified as such. *Tip: Every language has their own libraries to implement the different Cryptographic Standards, and also their recommendation of which to use.* ***Step 3. Document the minimum Cryptographic Standards to consider in the Design phase*** The selected Cryptographic Standards must be documented and taken as the minimum in regard of security implementation. So, in the design phase that minimum is taken as a baseline a used in any data that requires a higher level of security.

Complete Threat Models

Objectives:	Execute a complete Threat models
Roles:	AN – Analyst
	DES – Designer
	SA – Security Advisor
Products:	Software Design
	Requirement specifications
	Test Cases and Test Procedures
	Software Configuration
	Validation Results
Artifacts:	Threat Models

Steps:	1. Create the Complete threat models for all functionality identified
	2. Ensure that all threat models meet minimal threat model quality requirements
	3. Have all threat models and referenced mitigations reviewed and approved
	4. Threat model documentation.
Step Description:	**Step 1. Create the complete threat models for all functionality identified.** Threat models typically need to consider the following areas: - All projects. Code exposed to the attack surface and code created or licensed by a third party. - New project. All functionalities and features. - Updated version of an existing project. New features added in the updated version. **2. Ensure that all threat models meet minimal threat model quality requirements.** All threat models should include data flow diagrams, assets, vulnerabilities, and mitigations. Threat modeling can be done in a variety of ways, including defining approaches using tools or documentation / specifications. **3. Have all threat models and referenced mitigations reviewed and approved.** Ask architects, developers, testers, program managers, and other users to understand the software contributing and review the threat model. Seek extensive feedback and reviews to ensure that our threat model is as complete as possible. **4. Threat model documentation** Threat model data and associated documentation (functional/design specs) should be stored using the document control system used by the product team.

Role Description

This is an alphabetical list of the roles, abbreviations and list of competencies as defined in Part 5-1-2.

	Role	Abbreviation	Competency
1.	Analyst	AN	Knowledge and experience eliciting, specifying and analyzing the requirements. Knowledge in designing user interfaces and ergonomic criteria. Knowledge of the revision techniques and experience on the software development and maintenance. Knowledge of the editing techniques.
2.	Designer	DES	Knowledge and experience in the software components and architecture design. Knowledge of the revision techniques and experience on the software development and maintenance. Knowledge of the editing techniques. Knowledge and experience in the planning and performance of integration and system tests.
3.	Security Advisor	SA	Knowledge to advise the businesses to identify potential security weaknesses, create security policies, and reduce risks to their IT systems.
4.	Technical Leader	TL	Knowledge and experience in the software development and maintenance.

Product Description

This is an alphabetical list of the input, output and internal process products, its descriptions, possible states and the source of the product.

IMPORTANT NOTE: Part 5.1.2 of ISO/IEC 29110 describes an example of Software Design content. This Deployment Package intentionally uses a different Software Design product content.

	Name	Description	Source
1.	*Requirements Specification*	Includes an introduction and a description of the development and security requirements. It may contain: - Introduction –general description of software and its use within the scope of the customer	Software Implementation

		business; - Requirements description: - functionality – established needs to be satisfied by the software when it is used in specific conditions. Functionality must be adequate, accurate and safe.	
2.	*Software Configuration*	A consistent set of software products including: - *Requirements Specification* - *Software Design* - *Traceability Record* - *Components* - *Software* - *Test Cases and Test Procedures* - *Test Report* - *Product Operation Guide* - *Software User Documentation* - *Maintenance Documentation* The applicable statuses are: delivered and accepted.	Software Implementation
3.	*Software Design*	This document includes textual and graphical information on the software structure and behaviour. This may include the following parts: Software Architectural Design – Describes the overall *Software* structure and behaviour: - Identifies architectural design stakeholders and their concerns - Identifies relevant architectural mechanisms (patterns, tactics, heuristics, rules of thumb, ...) - Identifies the types of views that are relevant to convey the software architecture, taking into consideration the stakeholders concerns and the various requirements (functional and non-functional) - Provides relevant software architectural views in various forms (diagrams, models, tables, plain text, ...) - Identifies and describes the main elements of the software architecture (subsystems, layers, modules) and their relationships - Identifies and describes the required software *Components,* their interfaces and the relationships among them - Describes rationale, provides any analysis used to produce the solution, identifies known risks and inconsistencies. Detailed Software Design – includes details of the *Components* to facilitate their construction and testing within the programming environment:	Software Implementation

		- Provides detailed design (could be represented as a prototype, flow chart, entity relationship diagram, pseudo code, etc.) - Provides format of input / output data - Provides specification of data storage needs - Establishes required naming conventions - Defines the format of required data structures - Defines the data fields and purpose of each required data element - Provides the specifications of the program structure The applicable statuses are: verified and baselined.	
4.	*Test Cases and Test Procedures*	Test Case may include: - Identifies the test case - Test items - Input specifications - Output specifications - Environmental needs - Special procedural requirements - Interface dependencies Test Procedures may include: - Identifies: test name, test description and test completion date - Identifies potential implementation issues - Identifies the person who completed the test procedure - Identifies prerequisites - Identifies procedure steps including the step number, the required action by the tester and the expected results The applicable statuses are: verified and baselined.	Software Implementation
5.	*Traceability Record*	Relationship among the requirements included in the *Requirements Specification, Software Design* elements*, Components, Test Cases and Test Procedures.* - Identifies requirements of *Requirements Specification* to be traced - Provides forward and backwards mapping of requirements to *Software Design* elements, *Components*, *Test Cases and Test Procedures*. The applicable statuses are: verified, baselined and updated.	Software Implementation
6.	*Validation Results*	May include the record of: - Participants - Date	Project Management Software

		- Place - Duration - Validation check-list - Passed items of validation - Failed items of validation - Pending items of validation - Defects identified during validation	Implementation

Artefact Description

This is an alphabetical list of the artefacts that could be produced to facilitate the documentation of a project. The artefacts are not required by Part 5, they are optional.

	Name	Description
1.	Cryptographic Design	In this artefact a minimum of cryptographic considerations are established to help in the protection of the information.
2.	Threat Models	This models shows the threats that can put in danger the information managed by the software system developed.

5. Template

Software Design Template

This Table of Content is adapted from [IEEE 1471], [IEEE 1016] and the SEI's "Views and beyond" template for software architecture[3].

1 INTRODUCTION
1.1 Purpose
1.2 Scope
1.3 Definitions
1.4 References documents

2 DESIGN STAKEHOLDERS AND CONCERNS
2.1 Design stakeholders and their concerns
2.2 Design views and relationships to design concerns

3 SOFTWARE ARCHITECTURE DESCRIPTION
3.1 Overview of software architecture
3.2 Software architecture view 1
3.2.1 Overview
3.2.2 Design constraints that apply to this view
3.2.3 Design concerns and requirements addressed by this view
3.2.4 Description of view elements and their interfaces
3.2.5 Rationale
3.2.6 Other relevant views
3.3 Software architecture view 2
3.3.1 ...
...
3.x Architectural information that is relevant to multiple views

4 DETAILED DESIGN DESCRIPTION
4.1 Overview of detailed design
4.2 Detailed design of element 1
4.2.1 Structural view
4.2.2 Behavioural view
4.2.3 Other relevant views
4.2.4 Rationale
4.3 Detailed design of element 2
4.3.1 ...
...
4.x Detailed design information that is relevant to multiple elements

5 ANNEXES

[3] http://www.sei.cmu.edu/architecture/tools/viewsandbeyond/index.cfm (consulted 2010-Jul-22).

6. Example

Example of Software design Practice Lifecycle

To be developed

7. Checklists

The following checklists were adapted from the OpenUP[4] checklists for work products "Architecture Notebook" and "Design" [OPENUP]:

Note: The elements of the checklist can be tailored to the needs of the project

Elements common to both Software Architecture and Detailed Design (SADD)

SADD (UNDERSTANDABLE)	The design is understandable.
SADD (LEGEND)	All diagrams have an appropriate key.
SADD (TEXT)	All views are supported with text, minimally describing the rationale behind the view, assumptions, responsibilities of elements in the view, interfaces to the elements in the view, pointers to other relevant views.
SADD (MAINTAINABLE)	The design is maintainable.
SADD (USEFUL)	Each view helps the designer reason about the design, or communicates key design decisions to the team.
SADD (RELATIONSHIPS)	The relationships between views are clear when several views are used to describe structure and behaviour.
SADD (NAVIGABLE)	It is easy to navigate between related views.
SADD (COHERENCE)	Each view focuses on a relevant perspective.
SADD (COMPLETENESS)	Each view is complete and minimal. It shows everything relevant to that view and nothing more.
SADD (READABLE)	The views are tidy and easy to interpret, with a minimum of clutter.
SADD (VERIFIED)	The *Software Design* document has been verified and corrected.
SADD (APPROVED)	The *Software Design* has been approved and signed by the customer.
SADD (Repository)	The *Software Design*, *Traceability Record* and *Test Cases and Test Procedures* have been baselined and stored in the project repository.

Elements specific to Software architecture (SA)

SA (SCOPE)	The architectural goals, constraints and requirements are adequately described and handled.
SA (MECAHNISMS)	Necessary architectural mechanisms are identified, described and justified.
SA (TRACEABLE)	All architecture elements are traceable to requirements.
SA (BOUNDARIES)	The system partitions are adequately defined.
SA (KEY_ELEMENTS)	The key architectural elements are adequately defined.
SA (INTERFACES)	The interfaces between architectural elements and to external systems are adequately represented.

[4] http://epf.eclipse.org/wikis/openup/

SA (EVOLVABLE)	The architecture is built to evolve.
SA (BUILDABLE)	The architecture can be delivered by the team.
SA (HARDWARE)	The software elements are adequately mapped to hardware elements.

Elements specific to detailed design (DD)

DD (COMPLETENESS)	The detailed design, derived from the software architecture, is complete and correct.
DD (TRACEABLE)	All detailed design elements are traceable to architectural elements.
DD (CONSISTENCY)	The detailed design is consistent with the architecture and requirements.
DD (CONFORMITY)	The detailed design reflects the architectural objectives of the system.
DD (MODULARITY)	The detailed design elements are modular (high cohesion, low coupling, appropriate use of abstract interfaces).
DD (BUILDABILITY)	The system can be implemented from the information in the detailed design.
DD (TESTABILITY)	The design provides enough information for developer testing.

8. Tools

Version Control tools

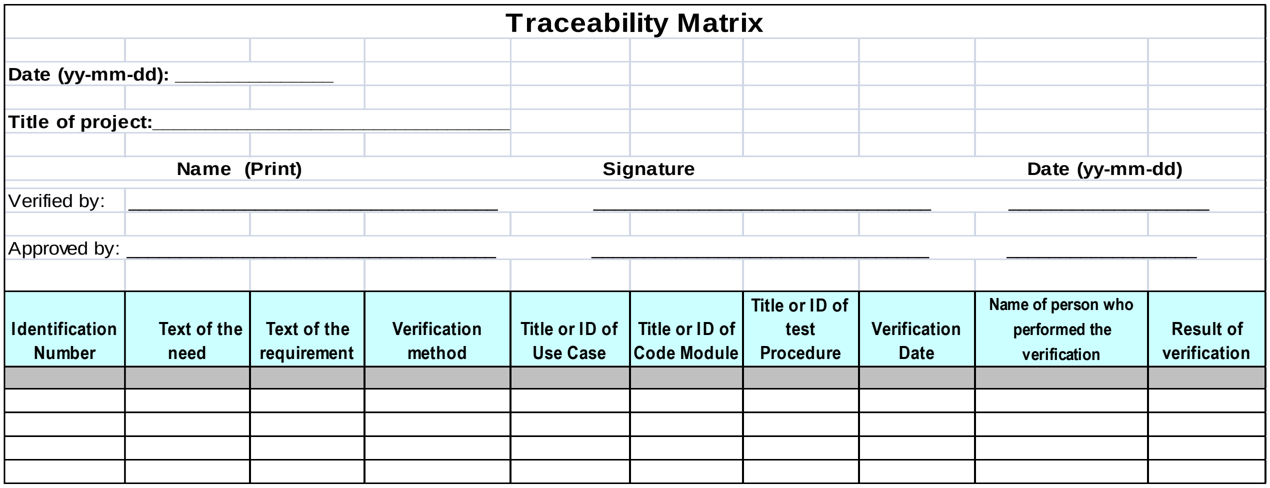
- Subversion (SVN)
- Concurrent Version System (CVS)
- GIT
- Perforce
- Microsoft Visual Source Safe (VSS)

Design description software

- UML tools: Visual paradigm UML, Smart Draw, Omondo,...
- Standard office tools: OpenOffice Writer, Microsoft Visio, Microsoft Word,...
- Full development environments with modelling capabilities: Eclipse, Sparx Systems Enterprise Architect (Windows), Microsoft Visual Studio, ...

Traceability Tool

- Objectives:
  - To maintain the linkage from the source of each requirement through its decomposition to implementation and test (verification).
  - To ensure that all requirements are addressed and that only what is required is developed.
  - Useful when conducting impact assessments of requirements, design or other configured item changes.



Traceability Matrix

Date (yy-mm-dd): _______________

Title of project:___________________________________

	Name (Print)	Signature	Date (yy-mm-dd)
Verified by:	________________________________	______________________________	__________________
Approved by:	________________________________	______________________________	_________________

Identification Number	Text of the need	Text of the requirement	Verification method	Title or ID of Use Case	Title or ID of Code Module	Title or ID of test Procedure	Verification Date	Name of person who performed the verification	Result of verification

Instructions	
The above table should be created in a spreadsheet or database such that it may be easily sorted by each column to achieve bi-directional traceability between columns. The unique identifiers for items should be assigned in a hierarchical outline form such that the lower level (i.e. more detailed) items can be traced to higher items.	
Unique Requirement Identification (ID)	The Unique Requirement ID / System Requirement Statement where the requirement is referenced, and/or the unique identification (ID) for decomposed requirements
Requirement Description	Enter the description of the requirement (e.g., Change Request description).
Design Reference	Enter the paragraph number where the CR is referenced in the design documentation
Module / Configured Item Reference	Enter the unique identifier of the software module or configured item where the design is realized.
Release Reference	Enter the release/build version number where the requirement is fulfilled
Test Script Name/Step Number Reference	Enter the test script name/step number where the requirement is referenced (e.g., Step 1)

Guideline

Requirements traceability should:

- Ensure traceability for each level of decomposition performed on the project. In particular:
  - Ensure that every lower level requirement can be traced to a higher level requirement or original source
  - Ensure that every design, implementation, and test element can be traced to a requirement
  - Ensure that every requirement is represented in design and implementation
  - Ensure that every requirement is represented in testing/verification
- Ensure that traceability is used in conducting impact assessments of requirements changes on project plans, activities and work products
- Be maintained and updated as changes occur.
- Be consulted during the preparation of Impact Assessments for every proposed change to the project
- Be planned for, since maintaining the links/references is a labor intensive process that should be tracked/monitored and should be assigned to a project team member
- Be maintained as an electronic document

9. Reference to Other Standards and Models

This section provides references of this deployment package to selected ISO and ISO/IEC Standards and to the Capability Maturity Model Integration^{SM} version 1.2 of the Software Engineering Institute (CMMI® [5]).

Notes:

- This section is provided for information purpose only.
- Only tasks covered by this Deployment Package are listed in each table.
- The tables use the following convention:
  - Full Coverage = F
  - Partial Coverage = P
  - No Coverage = N

ISO 9001 Reference Matrix[6]

Clause of ISO 9001	Coverage F/P/N	Title of the Task and Step	Comments
7.3.1 Plan product design and development.	P	Software Architectural and Detailed Design Step 1,4,6	No planning in the deployment package except for the initial project plan
7.3.2 Identify design and development inputs.	F	Software Architectural and Detailed Design Step 2,3	
7.3.3 Generate design and development outputs.	P	Software Architectural and Detailed Design Step 2,3	No acceptance criteria in design phase
7.3.4 Carry out design and development reviews.	F	Software Architectural and Detailed Design Step 4,8	
7.3.5 Perform design and development verifications.	F	Software Architectural and Detailed Design Step 6,7,8	
7.3.6 Conduct design and development validations.	N		No formal validation steps, just verification steps
7.3.7 Manage design and development changes.	F	Software Architectural and Detailed Design Step 7,8	

[5] SM CMM Integration is a service mark of Carnegie Mellon University.

® Capability Maturity Model, CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[6] ISO 9001 clauses are extracted from: http://www.praxiom.com/iso-9001.htm

ISO/IEC 12207 Reference Matrix

Clause of ISO/IEC 12207	Coverage F/P/N	Title of the Task and Step	Comments
7.1.3.3.1.1 The implementer shall transform the requirements for the software item into an architecture that describes its top-level structure and identifies the software components. It shall be ensured that all the requirements for the software item are allocated to its software components and further refined to facilitate detailed design. The architecture of the software item shall be documented.	F	Software Architectural and Detailed Design Step 2 and 3	
7.1.3.3.1.2 The implementer shall develop and document a top-level design for the interfaces external to the software item and between the software components of the software item.	F	Software Architectural and Detailed Design Step 2	
7.1.3.3.1.3 The implementer shall develop and document a top-level design for the database.	F	Software Architectural and Detailed Design Step 3	
7.1.3.3.1.4 The implementer should develop and document preliminary versions of user documentation.	F	Software Architectural and Detailed Design Step 3	
7.1.3.3.1.5 The implementer shall define and document preliminary test requirements and the schedule for Software Integration.	P	Software Architectural and Detailed Design Step 5	Except scheduling

Clause of ISO/IEC 12207	Coverage F/P/N	Title of the Task and Step	Comments
7.1.3.3.1.6 The implementer shall evaluate the architecture of the software item and the interface and database designs considering the criteria listed below. The results of the evaluations shall be documented. a) Traceability to the requirements of the software item. b) External consistency with the requirements of the software item. c) Internal consistency between the software components. d) Appropriateness of design methods and standards used. e) Feasibility of detailed design. f) Feasibility of operation and maintenance.	P	Software Architectural and Detailed Design Step 2, 3,4, 7,8	Except feasibility of operation and maintenance
7.1.3.3.1.7 The implementer shall conduct review(s) in accordance with subclause 7.2.6 7.2.6 Conduct Software Review	F	Software Architectural and Detailed Design Step 4	
7.1.4.3.1.1 The implementer shall develop a detailed design for each software component of the software item. The software components shall be refined into lower levels containing software units that can be coded, compiled, and tested. It shall be ensured that all the software requirements are allocated from the software components to software units. The detailed design shall be documented.	F	Software Architectural and Detailed Design Step 3	
7.1.4.3.1.2 The implementer shall develop and document a detailed design for the interfaces external to the software item, between the software components, and between the software units. The detailed design of the interfaces shall permit coding without the need for further information.	F	Software Architectural and Detailed Design Step 3	
7.1.4.3.1.3 The implementer shall develop and document a detailed design for the database.	N		Nothing about SHALL develop

Clause of ISO/IEC 12207	Coverage F/P/N	Title of the Task and Step	Comments
7.1.4.3.1.4 The implementer shall update user documentation as necessary.	F	Software Architectural and Detailed Design Step 4	
7.1.4.3.1.5 The implementer shall define and document test requirements and the schedule for testing software units. The test requirements should include stressing the software unit at the limits of its requirements.	P	Software Architectural and Detailed Design Step 5 and 6	Nothing specific about stress testing
7.1.4.3.1.6 The implementer shall update the test requirements and the schedule for Software Integration.	F	Software Architectural and Detailed Design Step 5 and 6	
7.1.4.3.1.7 The implementer shall evaluate the software detailed design and test requirements considering the criteria listed below. The results of the evaluations shall be documented. a) Traceability to the requirements of the software item; b) External consistency with architectural design; c) Internal consistency between software components and software units; d) Appropriateness of design methods and standards used; e) Feasibility of testing; f) Feasibility of operation and maintenance.	P	Software Architectural and Detailed Design Step 3,4,5,6,7	Except feasibility of operation and maintenance
7.1.4.3.1.8 The implementer shall conduct review(s) in accordance with subclause 7.2.6. 7.2.6 Conduct Software Review	F	Software Architectural and Detailed Design Step 4,6	Not all in accordance with clause 7.2.6

CMMI Reference Matrix

Objective/ Practice of CMMI V1.2	**Coverage F/P/N**	**Title of the Task and Step**	**Comments**
TS(SG 2) SP 2.1 Design the Product or Product Component Develop a design for the product or product component.	P	Software Architectural and Detailed Design Step 3,4	No specifics verifications included in the deployment package
TS(SG 2) SP 2.2 Establish a Technical Data Package Establish and maintain a technical data package.	P	Software Architectural and Detailed Design Step 2,3	Not a real project package is created
TS(SG 2) SP 2.3 Design Interfaces Using Criteria Design product component interfaces using established criteria.	P	Software Architectural and Detailed Design Step 2,3	Nothing about rationale for selected design or alternative
TS(SG 2) SP 2.4 Perform Make, Buy, or Reuse Analyses Evaluate whether the product components should be developed, purchased, or reused based on established criteria.	N		Nothing about reuse or buy a product instead of building it

10. References

Key	Reference
[CLEMENTS 03]	P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, "Documenting Software Architectures - Views and Beyond", Addison Wesley, 512 pages, 2003, ISBN 0-201-70372-6.
[ISO/IEC 12207]	ISO/IEC 12207:2008 Systems and software engineering - Software life cycle processes.
[ISO/IEC 24765]	ISO/IEC 24765:2010, Systems and Software Engineering Vocabulary. (http://pascal.computer.org/sev_display/index.action)
[ISO/IEC 29110]	ISO/IEC 29110:2011, Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 5-1-2: Management and engineering guide – Generic profile group - Basic Profile.
[IEEE 1471]	IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems
[IEEE 1016]	IEEE recommended Practice for Software Design Description.
[OPENUP]	Open Unified Process (OpenUP), available online at http://epf.eclipse.org/wikis/openup/.

11. Evaluation Form

Deployment Package - Design Description Version 0.4

Your feedback will allow us to improve this deployment package, your comments and suggestions are welcomed.

1. How satisfied are you with the CONTENT of this deployment package?

☐ *Very Satisfied* ☐ *Satisfied* ☐ *Neither Satisfied nor Dissatisfied* ☐ *Dissatisfied* ☐ *Very Dissatisfied*

2. The sequence in which the topics are discussed, are logical and easy to follow?

☐ *Very Satisfied* ☐ *Satisfied* ☐ *Neither Satisfied nor Dissatisfied* ☐ *Dissatisfied* ☐ *Very Dissatisfied*

3. How satisfied were you with the APPEARANCE/FORMAT of this deployment package?

☐ *Very Satisfied* ☐ *Satisfied* ☐ *Neither Satisfied nor Dissatisfied* ☐ *Dissatisfied* ☐ *Very Dissatisfied*

4. Have any unnecessary topics been included? (please describe)

5. What missing topic would you like to see in this package? (please describe)

- Proposed topic:
- Rationale for new topic

6. Any error in this deployment package?

- Please indicate:
  - Description of error :
  - Location of error (section #, figure #, table #) :

7. Other feedback or comments:

8. Would you recommend this Deployment package to a colleague from another VSE?

☐ *Definitely* ☐ *Probably* ☐ *Not Sure* ☐ *Probably Not* ☐ *Definitely Not*

Optional

- Name: ____
- e-mail address :____

Email this form to: claude.y.laporte@etsmtl.ca or Avumex2003@yahoo.com.mx