

## 1. Software Engineering Overview

1. Software Engineering Overview .....	1
1.1 Total programme structure .....	1
1.2 Topics covered in module.....	2
1.3 Examples of SW eng. practice in some industrial sectors .....	4
1.3.1 European Space Agency (ESA), <a href="#">software engineering standards</a> .....	4
1.3.2 Civil Aviation: software within a system, software levels, Level D & C summaries.....	5
1.3.3 <a href="#">IEC 61508</a> - "Functional safety of electrical/electronic/programmable electronic safety-related systems.....	15
1.3.4 Guidance related to software contained in medical devices .....	16
1.4 A preliminary note on software life cycle: .....	17
1.5 Concluding note and caution .....	18

### 1.1 Total programme structure

CA582	Computer architecture & operating systems
CA583	Computing in business
CA589	Database design
CA591	Object-oriented programming
CA596	Information systems framework
CA586	Software engineering
CA587	Networks & Internets
CA592	Algorithms & data structures
CA593	User interface development
CA597	Introduction to e-commerce

The main prerequisite module is CA591.

## 1.2 Topics covered in module

(a) Indicative CA586 syllabus (as listed on school web site)

*(May not cover all, or in this order)*

The software engineering discipline		
System analysis, Requirements Analysis		
Object modelling with UML		
Design		<i>Using UML to document. Continuous assessment will require some programming inter alia</i>
Software construction		
<del>File and database design</del>		<i>Won't cover in fact</i>
Verification - reviews and testing		
Project management including planning		
Software Quality Assurance		
Software Configuration Management		

(b) Recommended textbook<sup>1</sup>

*Using UML, Software engineering with objects and components* by P. Stevens with R. Pooley, Addison-Wesley, 2000

Main elements of recommended textbook:

<b>Part I: Conceptual background</b>	Software engineering with components Object concepts Introductory case study The development process	
<b>Part II: The Unified Modeling Language</b>	<b>Essentials of</b> class models use case models interaction diagrams state and activity diagrams	<b>More on</b> class models use case models interaction diagrams state and activity diagrams <i>Design Reqs</i>
<b>Part III: Case studies</b>	Implementation diagrams Packages, subsystems, models CS4 administration Board games Discrete event simulation <i>Probably will omit this case study</i>	
<b>Part IV: Towards practice</b>	Reuse: components, patterns <b>Product</b> quality: verification, validation, testing <b>Process</b> quality: management, teams, QA	

Will follow text book fairly closely for much of the time but sometimes will add or omit material, or cover it differently.

<sup>1</sup> In addition, there are several books in the library (especially classmark 005.1 and neighbouring shelves) that are useful for background and further reading. There are also some journals in the library that have articles of interest (e.g. IEEE transactions on software engineering (a bit dry sometimes!), Software engineering journal, Software engineering notes, ACM transactions on software engineering and methodology, etc)

### 1.3 Examples of SW eng. practice in some industrial sectors<sup>2</sup>

Note: Some of following includes sector-specific terminology - not important for us.

#### 1.3.1 European Space Agency (ESA), [software engineering standards](#)

A summary of the main elements of SW eng. standards used by ESA for many years.

*Part 1 PRODUCT STANDARDS: Standards, recommendations, guidelines about the product (software) to be defined, implemented, operated and maintained.*

*Part 2 PROCEDURE STANDARDS: Procedures used to manage a software project.*

*Part 3 APPENDICES: Glossary, Software project documents, Document templates, Summary of mandatory practices, Form templates.*

(a): High-level structure of ESA (1991) SW Engineering Standards

1. SOFTWARE LIFE CYCLE
2. USER REQUIREMENTS DEFINITION PHASE
3. SW REQUIREMENTS DEFINITION PHASE
4. ARCHITECTURAL DESIGN PHASE
5. DETAILED DESIGN & PRODUCTION PHASE
6. TRANSFER PHASE
7. OPERATIONS & MAINTENANCE PHASE

where each phase (2 to 7) is further subdivided into

- X.1 Introduction
- X.2 Inputs to the phase
- X.3 Activities
- X.4 Outputs from the phase

(b): PRODUCT STANDARDS part of (1991) ESA SW Eng. Standards

1. MANAGEMENT OF THE SW LIFE CYCLE
2. SW PROJECT MANAGEMENT
3. SW CONFIGURATION MANAGEMENT
4. SW VERIFICATION & VALIDATION
5. SW QUALITY ASSURANCE

where each process (2 to 5) is subdivided into

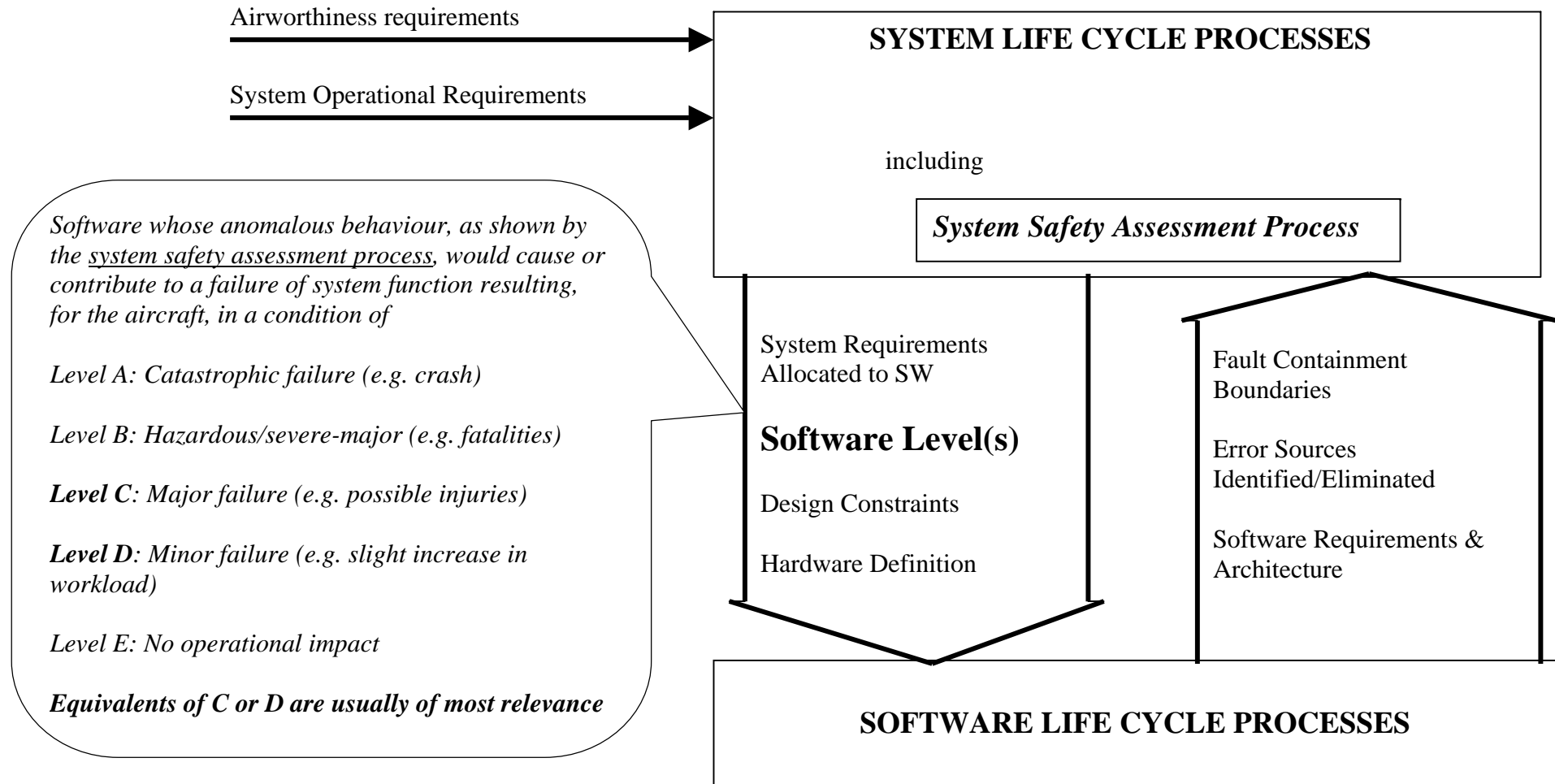
- X.1 Introduction
- X.2 Activities
- X.3 The <process> Plan
- X.4 Evolution of the <process> plan throughout the life cycle

(c): PROCEDURE STANDARDS part of (1991) ESA SW Eng. Standards

<sup>2</sup> Several of diagrams in this section are based on Software Quality Journal (Volume 10, Issue 1, July 2002, pages 47-68).

1.3.2 Civil Aviation: software within a system, software levels, Level D & C summaries

(a) Following (*Information Flow between System and Software Life Cycle Processes*) illustrates (1) *SW is often part of a wider system*; (2) there can be *different requirements depending on how critical the SW is*. (from "Software considerations in airborne systems ...", [RTCA/DO-178B](#)).

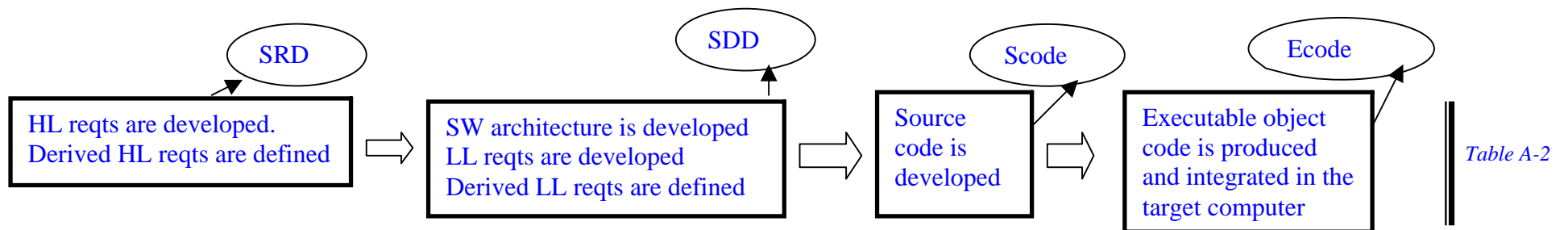


(b) Next we present a series of diagrams depicting the main constituent processes of Level D software (in civil aviation terms). Aim is to build up a picture of the roles and responsibilities involved.

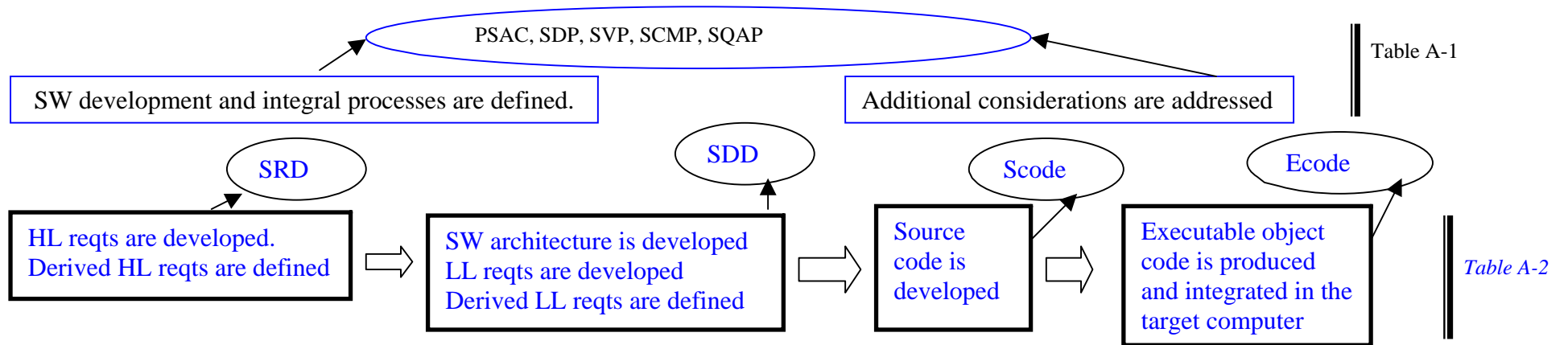
Note in reading these diagrams:

- Ignore the references to "tables" (in fact, these are tables within document RTCA/DO-178B).
- The oval shapes depict documentation.
- There is some specialised terminology here (that can be largely ignored), especially regarding high-level (HL) and low-level (LL) requirements. Roughly, can regard HL as being "requirements proper" and LL as being part of software design.

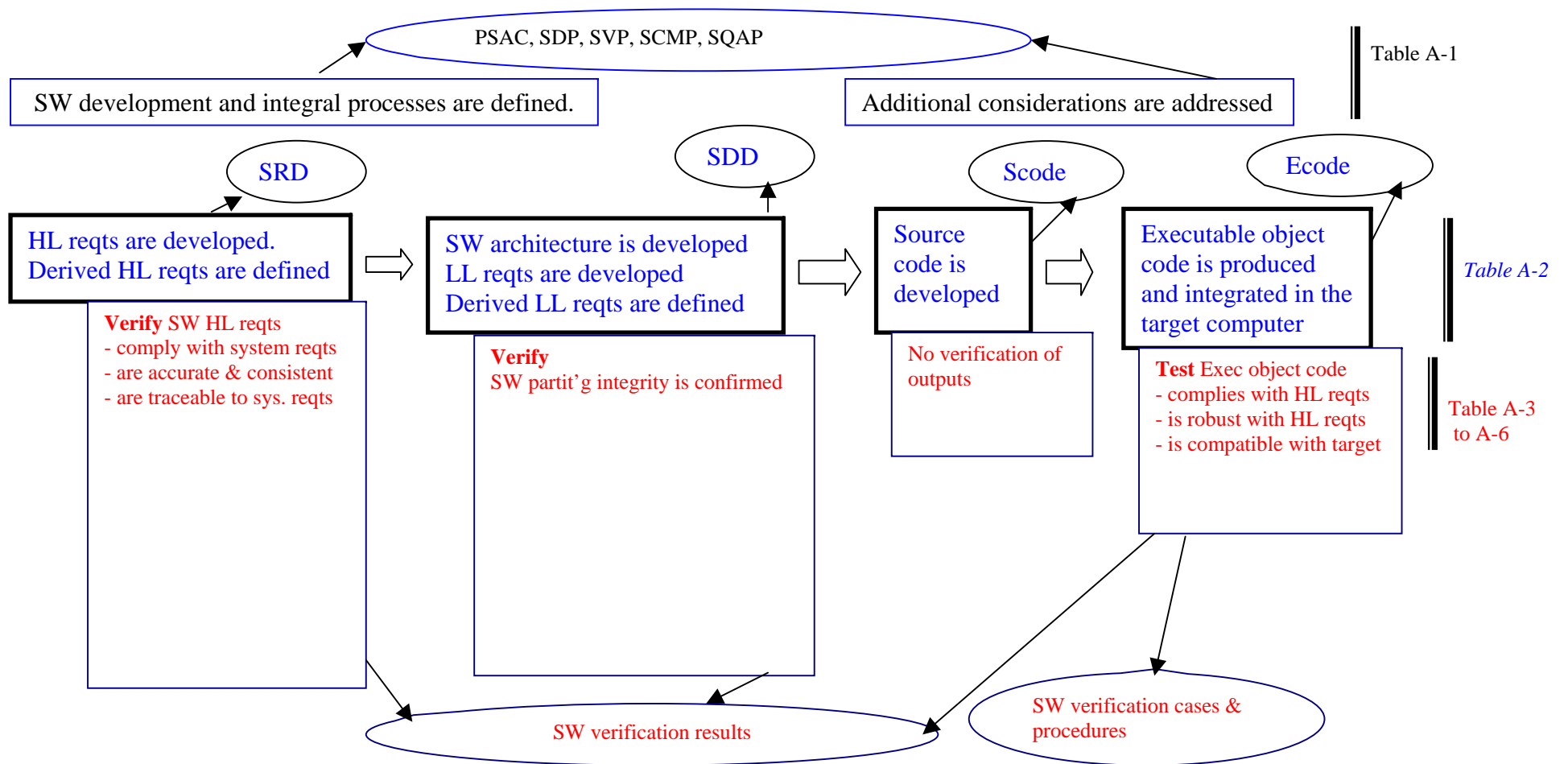
**LEVEL D SUMMARY-1 (SW "development process" only):**



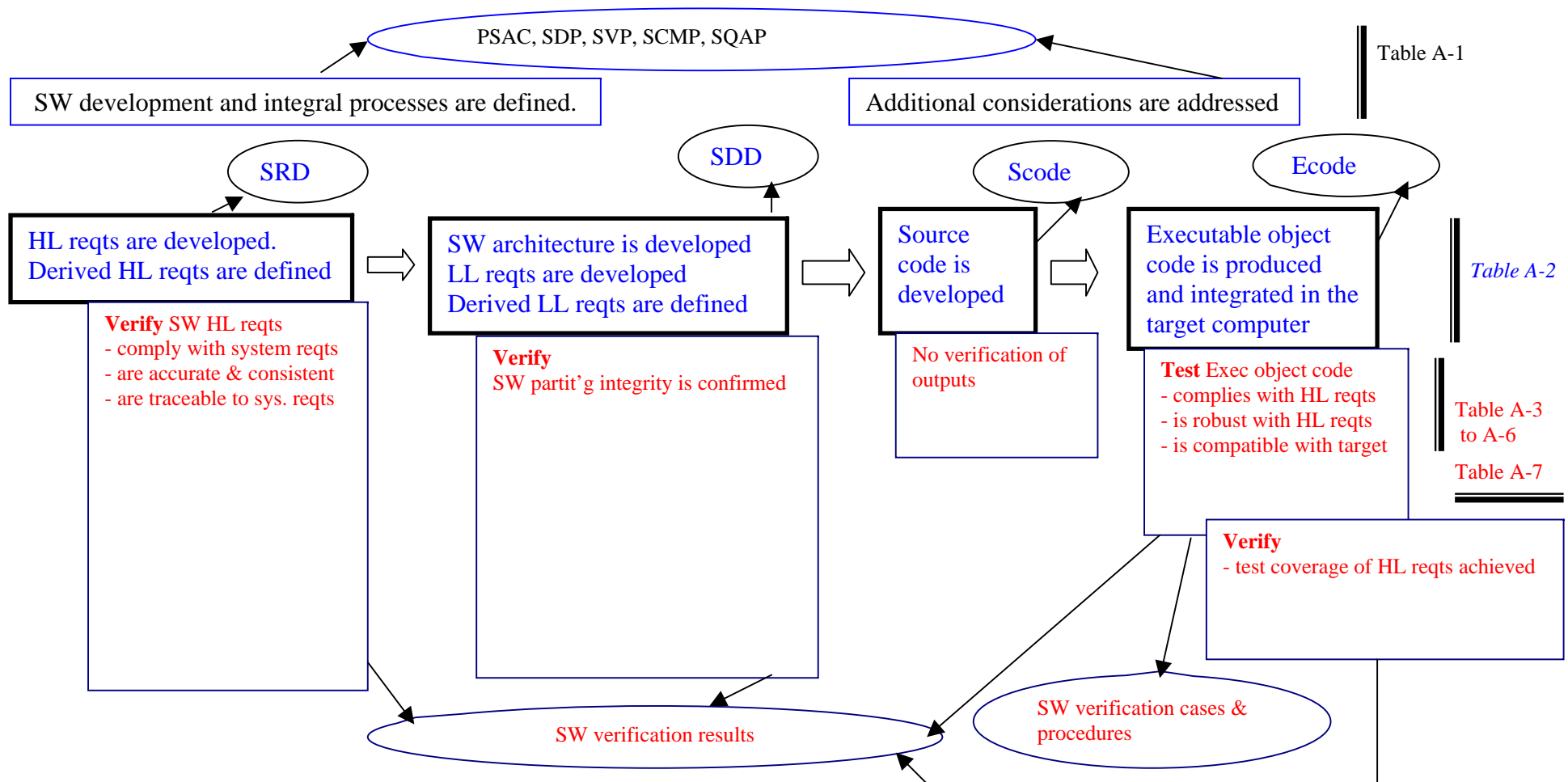
**LEVEL D SUMMARY-2 (SW "planning process" added):**

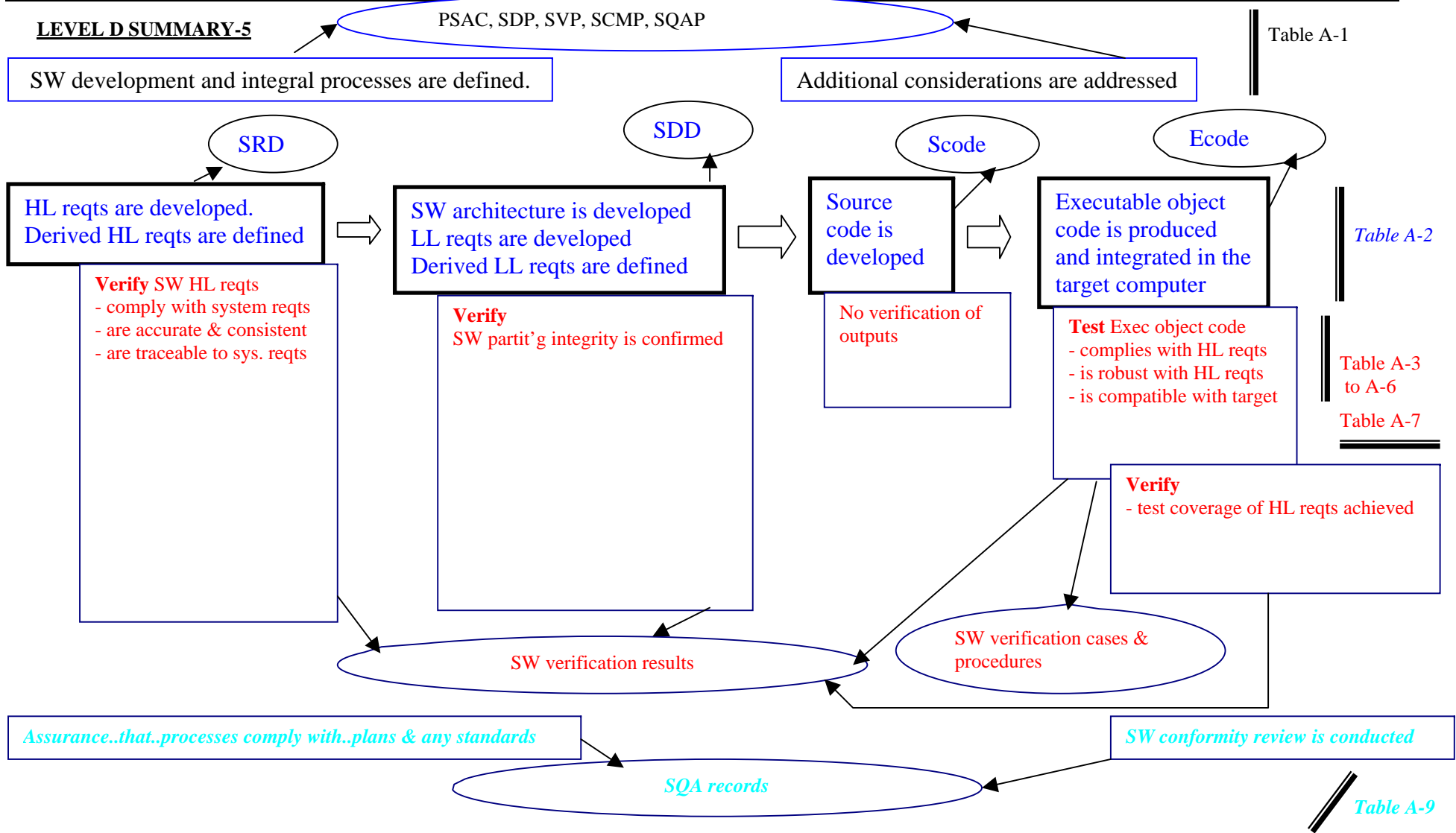


**LEVEL D SUMMARY-3 (SW "verification process" added):**



**LEVEL D SUMMARY-4 (Verification of verification!!!):**

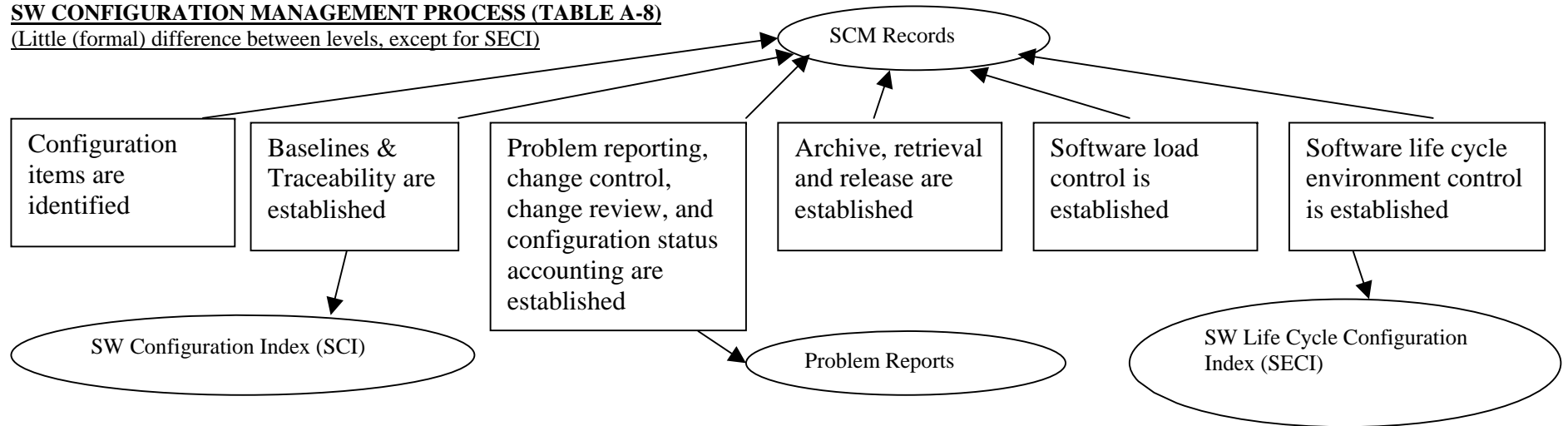




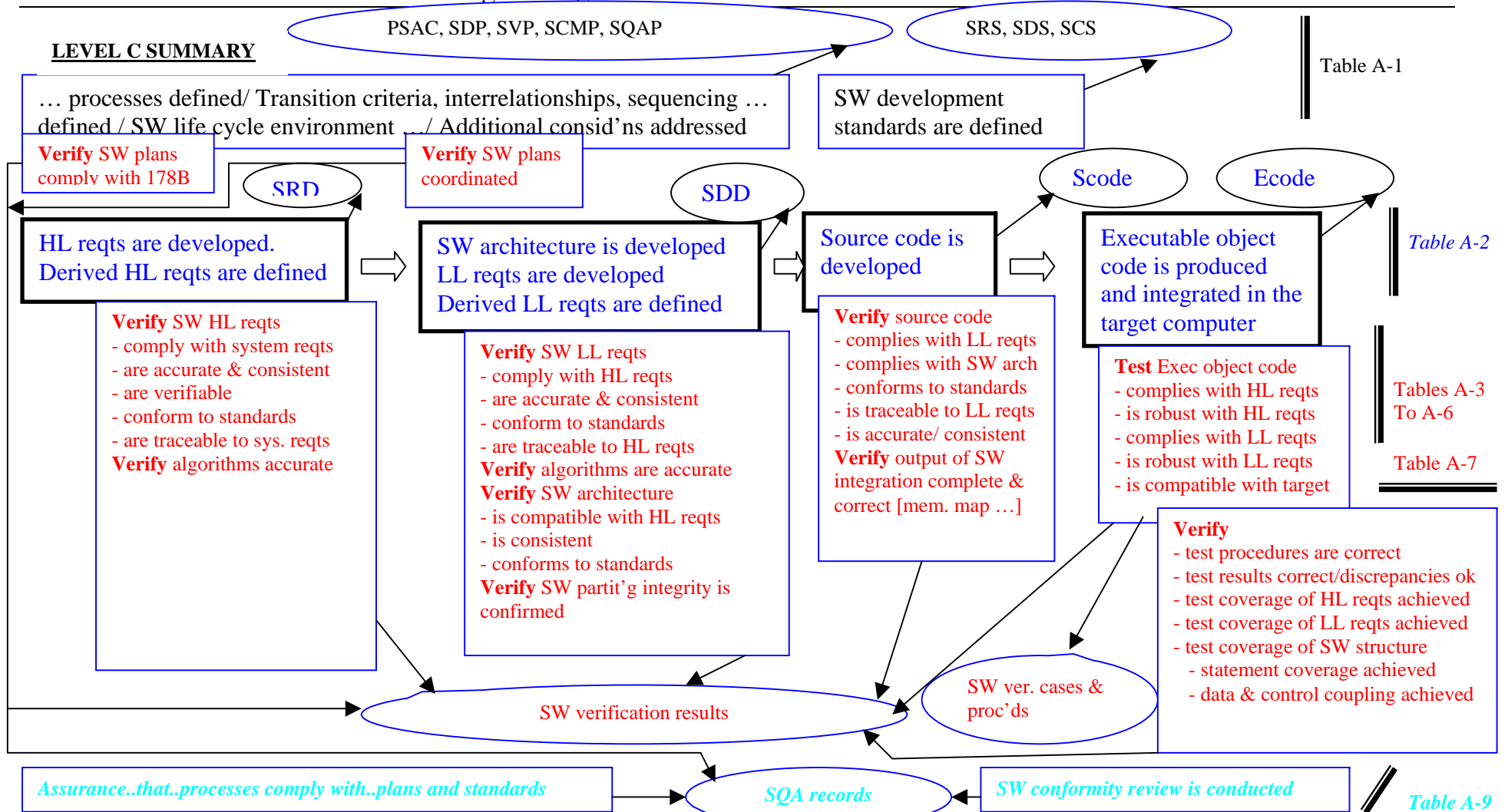
(c) The next diagram depicts the level D software "*configuration management*" process. (In fact, there is another process also called "certification liaison" but we can ignore this as being quite specialised). On the other hand, the elements of software configuration management are very important at all levels; the only distinction is that more rigorous controls are introduced for more critical software.

**SW CONFIGURATION MANAGEMENT PROCESS (TABLE A-8)**

(Little (formal) difference between levels, except for SECI)

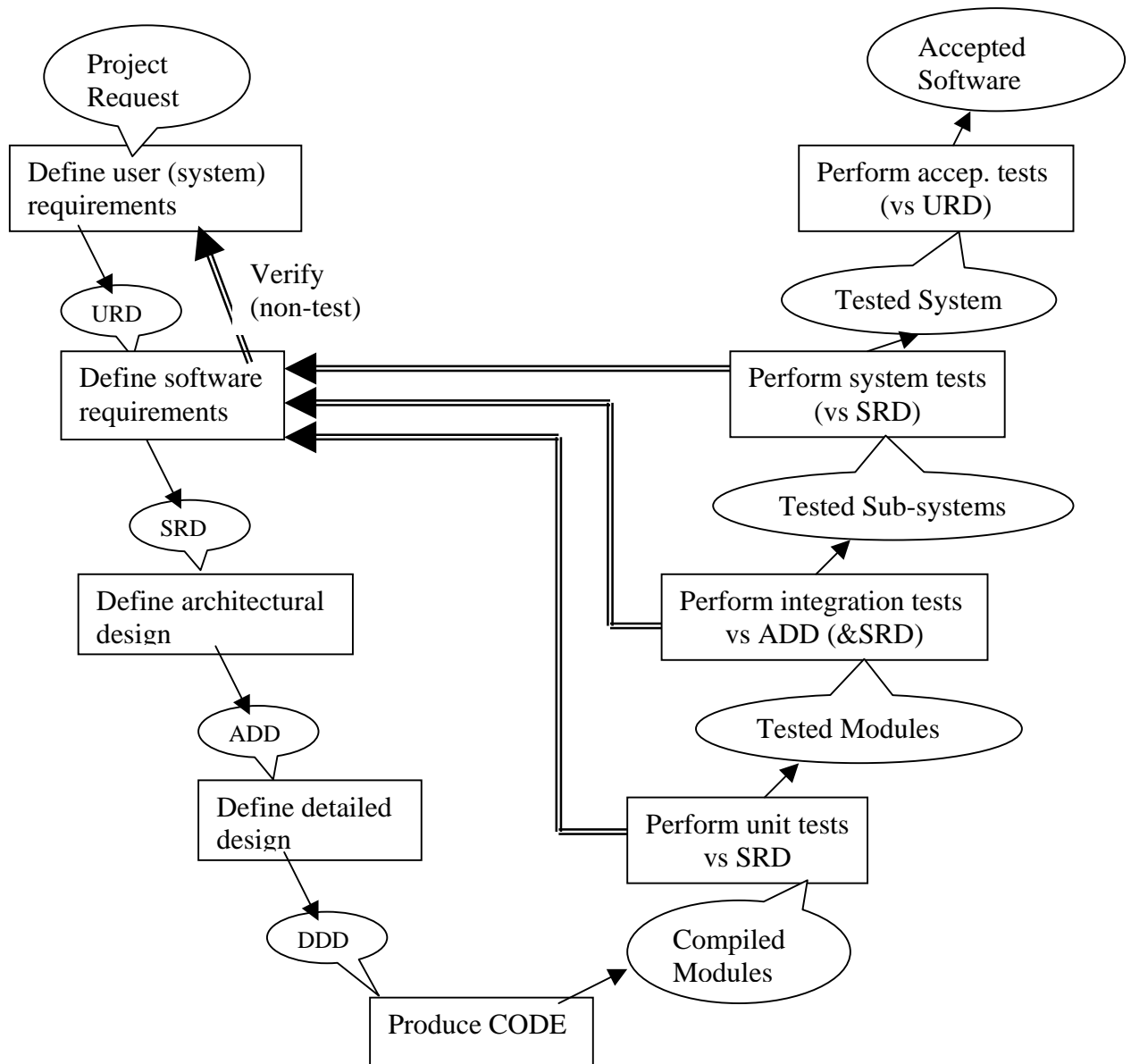


(d) Next, to show the more demanding requirements for a Level C development - especially in terms of verification though there are also more stringent requirements in configuration control and other areas - we have



(e) The following two diagrams provide a different way of viewing & contrasting SW levels D and C, through use of the classical "V" diagram:

Level D "mapped to" Classic Life Cycle Verification Approach:



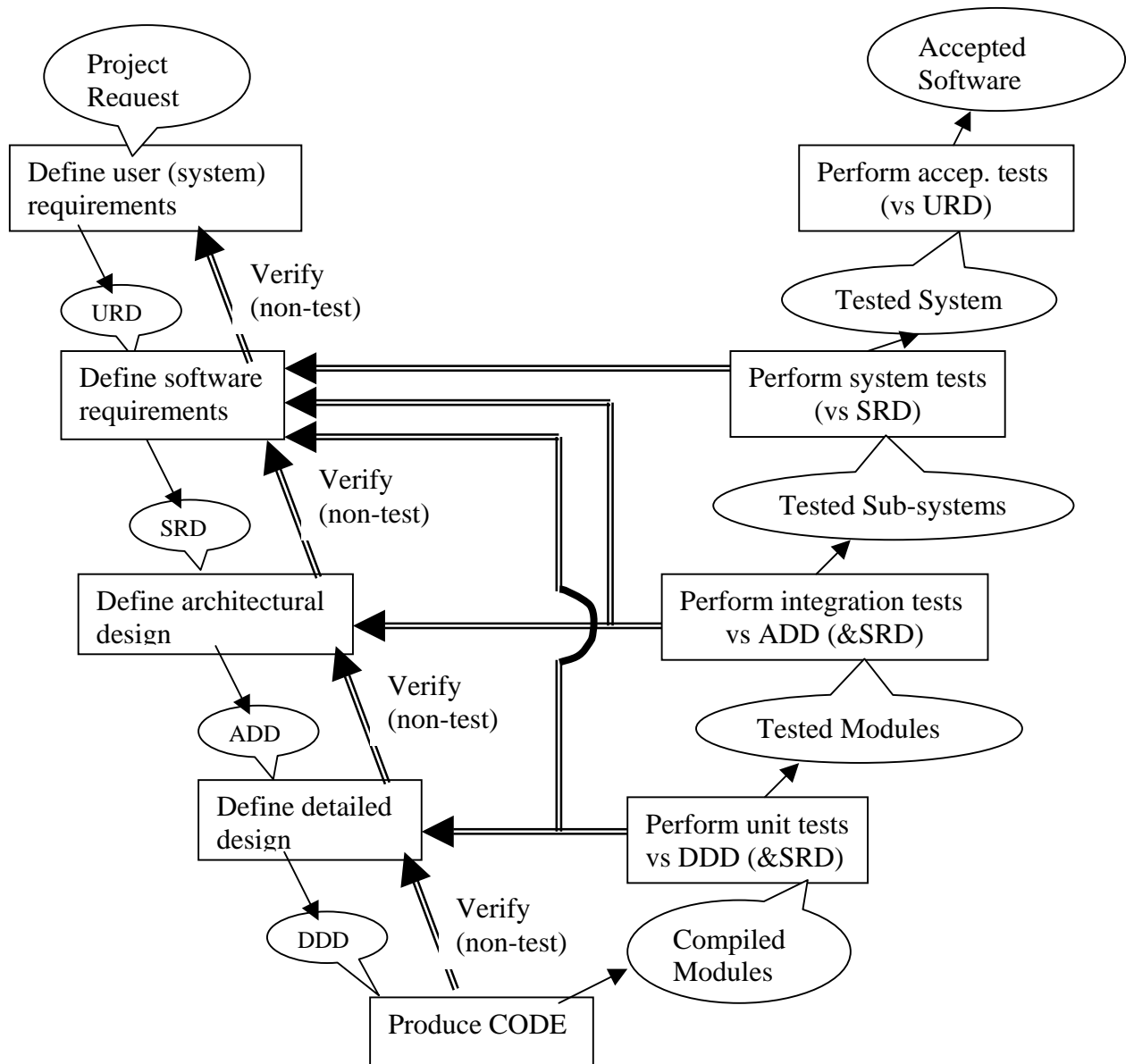
Legend:

URD	User requirements document/data
SRD	SW requirements document/data
ADD	Architectural design document/data
DDD	Detailed design document/data

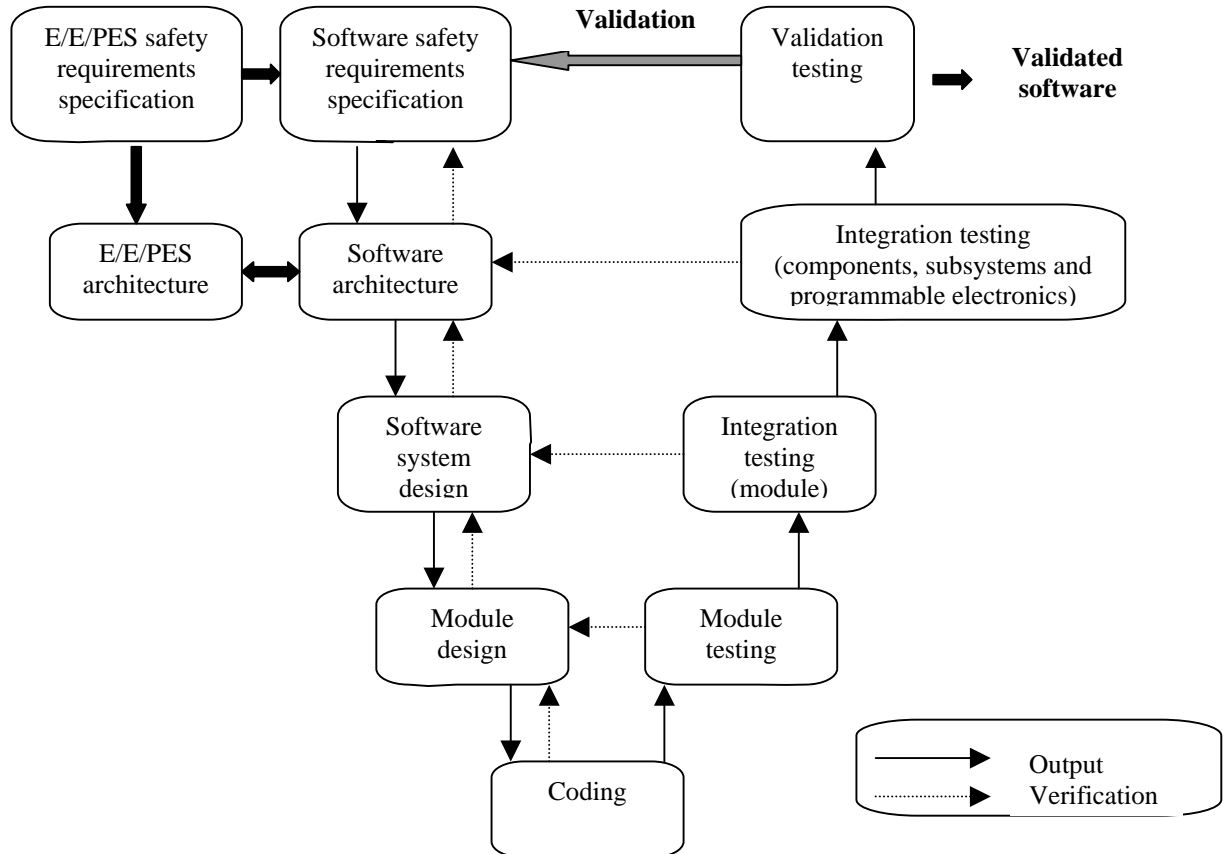
*In some circumstances might be combined*

*May be put into a single "SW design document" (SDD)*

Level C "mapped to" Classic Life Cycle Verification Approach:



1.3.3 [IEC 61508](#) - "Functional safety of electrical/electronic/programmable electronic safety-related systems.



V-model (IEC 61508-3, 1998)

(E/E/PES = electrical/electronic/programmable electronic system)

**Note:** IEC 61508 is an international standard for the "Functional safety of electrical/electronic/programmable electronic safety-related systems.

## 1.3.4 Guidance related to software contained in medical devices

<b>SOFTWARE DOCUMENTATION</b>	<b>MINOR CONCERN</b>	<b>MODERATE CONCERN</b>	<b>MAJOR CONCERN</b>
Level of Concern (determination)	All levels of concern		
Software Description	All levels of concern		
Device Hazard Analysis	All levels of concern		
Software Requirements Specification (SRS)	Software functional requirements from SRS	SRS	
Architecture Design Chart	A chart depicting the partitioning of the software system into functional subsystems	A chart depicting the partitioning of the software system into functional subsystems, listing of the functional modules and a description of how each fulfills the requirements.	
Design Specification	No documentation is necessary in the submission.	Software design specification document	
Traceability Analysis	No documentation is necessary in the submission.	Traceability among requirements, design specifications, identified hazards, and Verification and Validation testing.	
Development	No documentation is necessary in the submission.	Summary of software life cycle development plan, including a summary of the configuration management and maintenance activities.	Summary of software life cycle development plan. Annotated list of control documents generated during development process. Include the configuration management and maintenance plan documents.
Validation, Verification and Testing (VV&T)	Software functional test plan, pass / fail criteria, and results	Description of VV&T activities at the unit, integration and system level. System level test protocol including pass/fail criteria, and tests results.	Description of VV&T activities at the unit, integration and system level. Unit, integration and system level test protocols including pass/fail criteria, test report, summary, and tests results.
Revision Level History	No documentation is necessary in the submission.	Revision history log	
Unresolved anomalies (bugs)	No documentation is necessary in the submission.	List of errors and bugs which remain in the device and an explanation how they were determined to not impact safety or effectiveness, including operator usage and human factors.	
Release Version Number	Version number and date for all levels of concern.		

## Documentation in a Pre-market Submission

(in Guidance for FDA reviewers and industry. Guidance for the content of [pre-market submissions for software contained in medical devices](#).)

**1.4 A preliminary note on software life cycle:**

The previous sections identify the processes involved in SW engineering as well as many of the key outputs (SW code, design data, requirements documentation). However, they do not depict what is called the "software life cycle" (except in a very general sense).

Very different plans and processes may be called for depending on circumstances. For example,

(A) Develop a new implementation of a standard set of requirements (such as a well-established communication protocol or the definition of a programming language) -> go straight to design process

(B) Configure a standard package to the needs of a particular customer -> formulate requirements and map them to the functions & interface provided by the package -> No design or coding; only system or acceptance level tests.

(C) Some or all of requirements are uncertain ("researchy") -> do the development cycle a number of times before getting a satisfactory solution

etc, etc

### **1.5 Concluding note and caution**

The foregoing preliminary notes are intended to give an overview of several of the main aspects and processes that make up software engineering.

It was convenient for the purpose of this overview to take examples from industrial sectors where there are reasonably well-defined, standardised approaches to software development.

*However, it is emphasised that many, if not most, software development organisations work in sectors where such standards are not imposed. It is up to the software development organisations themselves to define a software engineering approach that works for them and their customers.*

Often, over-formalised approaches may be too restrictive, inhibiting of initiative and creativity, and need too much time and effort. *The trick is to develop software in time and within budget and also of high enough quality.*

In 1995, Yourdon wrote in [When good enough software is best](#) (*IEEE Software, should be in library*) that

'The mathematics of the relationships between X[number of people], Y[units of time], Z[cost], P[units of functionality], and Q[bugs per function point] are something we don't know enough about at our present level of software engineering'.

***There's still a lot to learn about these issues.***

A very recent (February 2005) call for contributions to a workshop on software quality notes that

“The goal of software engineering is to achieve high-quality software in a cost-effective, timely, and reproducible manner. Advances in technology offer us reductions in cost and schedule, but their effect on software quality often remains unknown. Object-oriented concepts, component technology, components off the shelf (COTS), and open source software can dramatically reduce development time; however, assuring the quality of systems using these technologies is problematic.

New software development processes also complicate quality assurance. Agile methods explicitly allow customers to change requirements very late in the project. Agile methods also take a "light weight" approach to project documentation and software testing. Reducing process overhead can improve response to change and speed product delivery, but may also adversely affect the project's risk profile. Little data exists on the quality of industrial systems developed using Agile methods.

The job of measuring, assuring, and improving the quality of software systems is getting harder, not easier. The goal of this workshop is to bring together researchers, engineers, and practitioners to discuss and evaluate the latest challenges and breakthroughs in the field of software quality. The main focus of the workshop will be on software quality assurance.”