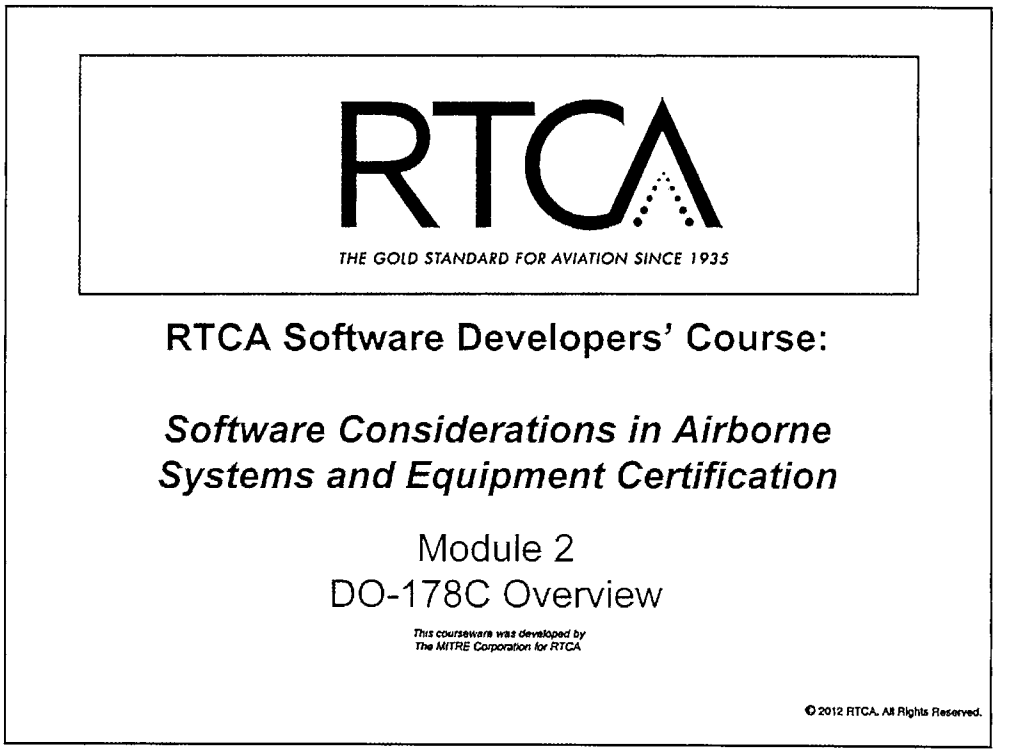
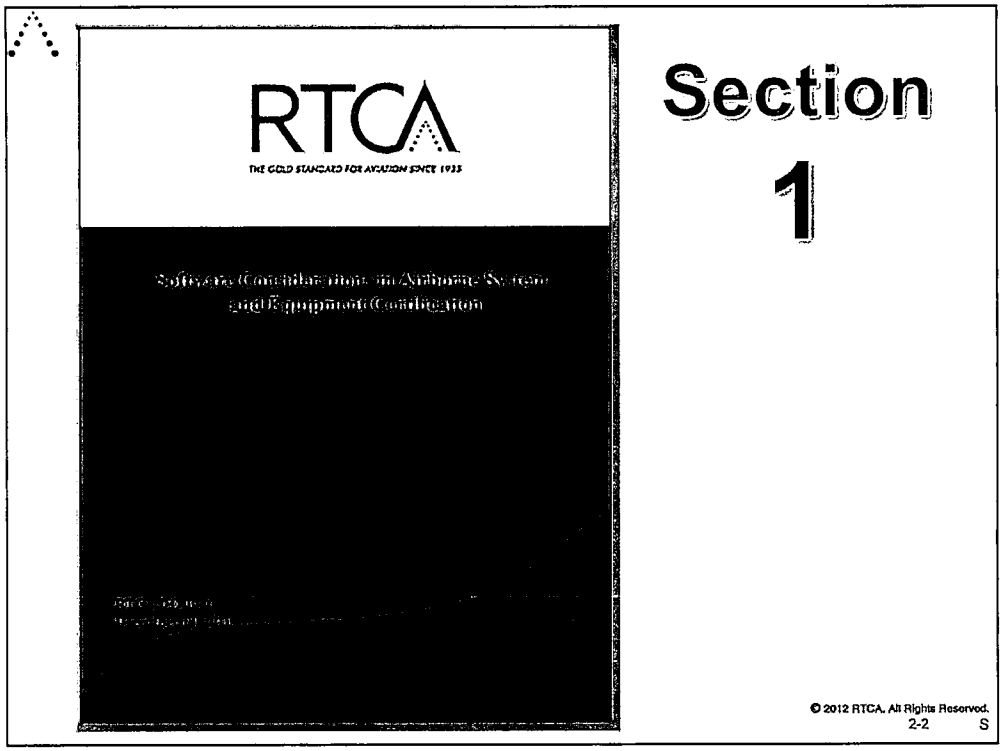


This module consists of a 1-hour presentation intended to provide an overview of the DO-178C document. The presentation includes the document's scope, relationship to other documents, the purpose of DO-178C, and the definition of a few key terms. It includes instruction on how to use DO-178C and describes how it is organized, provides a broad overview of the four Supplements and a module summary. Review questions are provided at the end of each of the modules to ensure that the course materials are covered but more importantly to provide a means to ensure that the course material is understood by participants.



DO-178C Section 1 slides

Module Objectives

- Understand organization and layout of DO-178C

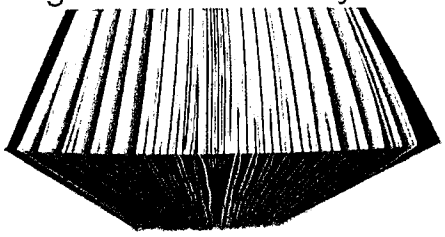


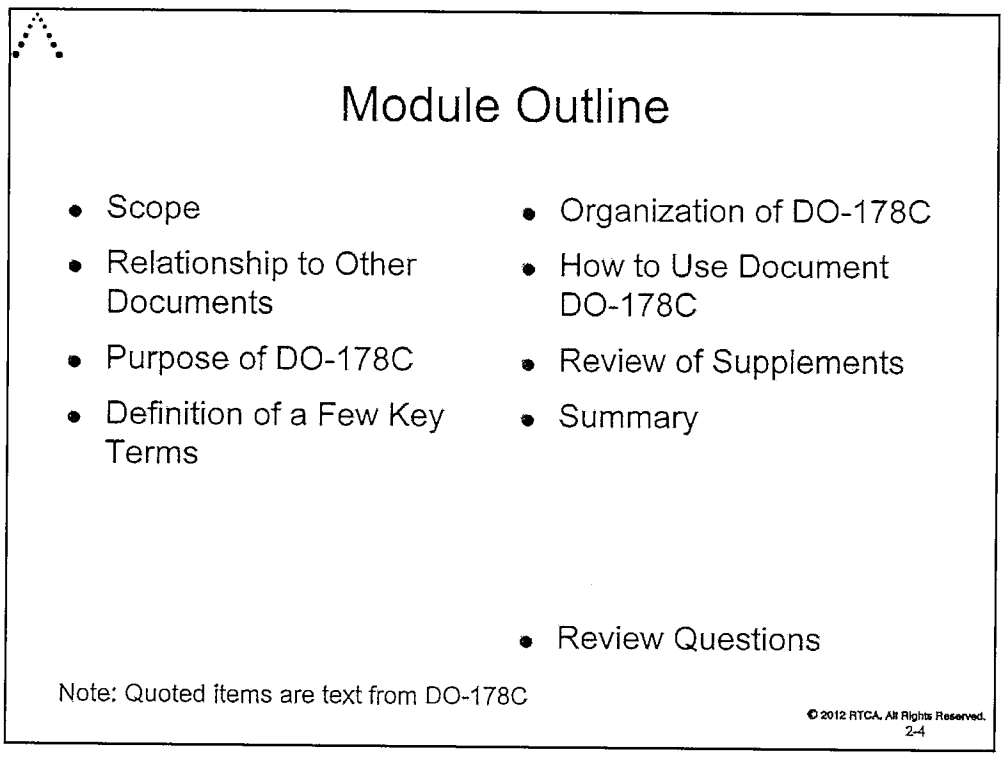
Image Source: MS

- Know the relationship of the four Supplements to DO-178C:
  - DO-330, *Software Tool Qualification Considerations*
  - DO-331, *Model-Based Development and Verification Supplement to DO-178C and DO-278A*
  - DO-332, *Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A*
  - DO-333, *Formal Methods Supplement to DO-178C and DO-278A*

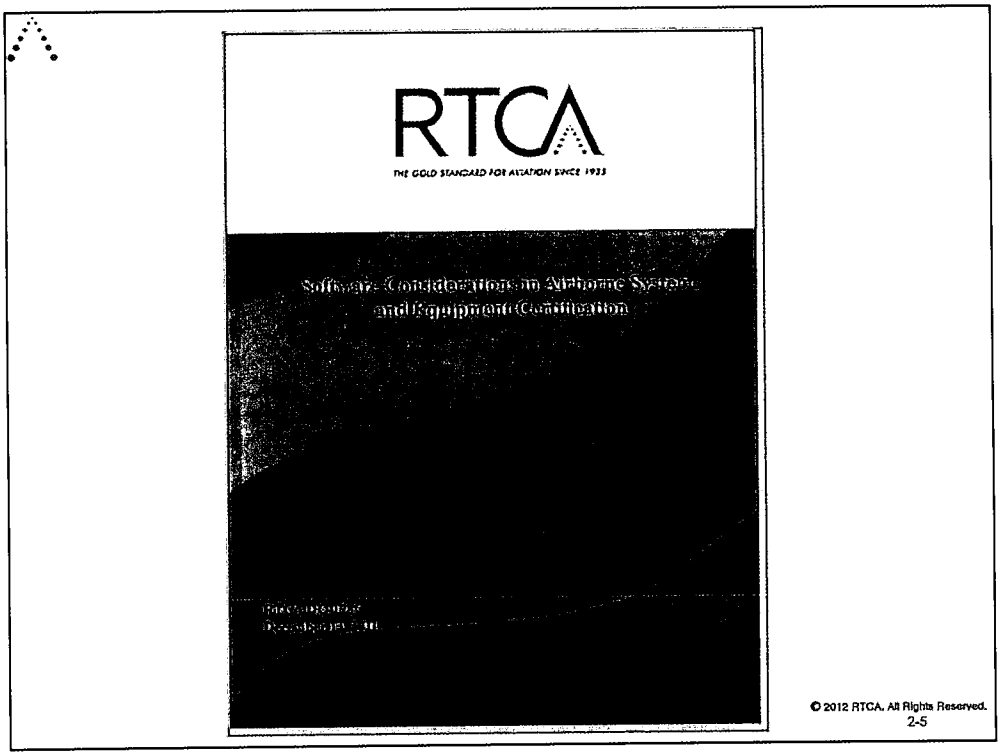
2-3

This slide identifies the module objectives. This module will review the organization and layout of the document, define a few key terms and provide a brief overview of the four supplements.

How to use multiple supplements with the core document on a project will be mentioned.



This slide provides the module outline.



This is the cover of DO-178C.

The course is about this document.

There are many different adjectives used on the word "Considerations" through out the document. For example:

- System Considerations
- Architectural Considerations
- Language and Compiler Considerations

There are many other types of considerations not mentioned within this document.

For example:

- Cost Considerations
- Schedule Considerations
- Maintainability considerations

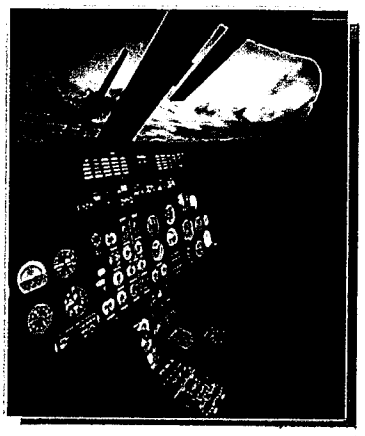
The use of the word Airborne implies the main focus of this document is on software in airplanes and other types of aircraft.

And finally this document was written primarily to provide a means to achieve Certification.

Scope of DO-178C (1 of 2)

What is in Scope?

- Production of software for airborne systems and equipment certification
  - Software Life Cycle processes
  - Relationship to System Life Cycle
  - Relationship to Hardware Life Cycle
- Airborne systems and equipment can be on:
  - aircraft
  - engines
  - propellers
  - by region, auxiliary power units



Section 1.2

2-6

This chart identifies what is within the scope of DO-178C.

Weight and balance on the aircraft is in the scope. Weight and balance used on the ground is not in scope.

Production of software means the creation of the ones and zeros running in the airborne system and equipment. This would include using previously developed software such as COTS or software from other airborne systems and equipment.

The next slide identifies what is out of the scope of DO-178C.

Scope of DO-178C (2 of 2)

What is out of Scope?

- Complete description of *system* life cycle processes, including *system* safety assessment, validation processes, and certification processes
- Level of certification authority involvement
- Operational aspects
- Hardware including firmware
- Structure of applicant's organization
- Commercial relationships between applicant(s) and supplier(s)
- Personnel qualification criteria
- Software not certified, example passenger laptops

Image Source: iStock

Section 1.2

2-7

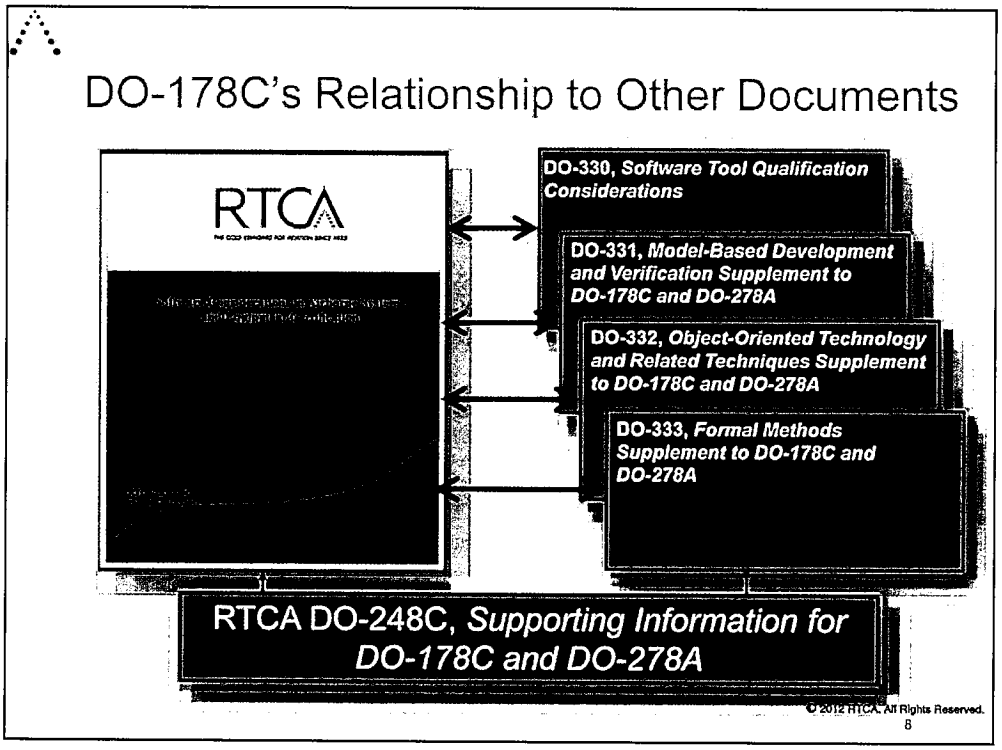
This chart identifies what is out of the scope of DO-178C.

We will not talk about what the FAA must do.

We don't talk about operations (i.e. when to put on a seat belt).

Why isn't firmware in the document? It is out of scope and considered hardware (see DO-254).

Section 2 may also have other "Out of Scope" criteria.



This slide indicates the relationship of DO-178C to:

- The four DO-178C Supplements (new to DO-178C)
- RTCA DO-248C, *Supporting Information for DO-178C and DO-278A, released Dec 2011*

Supplements: Covered in more detail in "Review of Supplements" slide

Tools – This supplement is referenced in DO-178C section 12.2 as containing the objectives, activities, guidance, and life cycle data required for each Tool Qualification Level

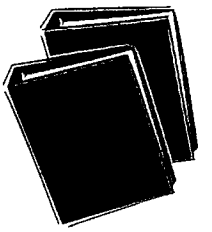
Formal Methods, Object-Oriented technology, Model-Based Development – Contain the additions, deletions, and changes to DO-178C objectives and activities for each of the three software development technologies.

RTCA DO-248C: contains FAQs and Discussion Papers that provide additional information about DO-178C/DO-278A. The same committee developed both docs.

Relationship to Other Documents

It does *not* define a relationship to:

- Airworthiness requirements
- National and international software standards
- Contractual standards
  e.g. from an engine or aircraft manufacturer



NOTE: DRAFT "Advisory Circular AC No: 20-115C", Available 2012
Subject: RTCA, Inc., Document RTCA/DO-178C

2-9

Airworthiness requirements are in the regulations and not defined by DO-178C. DO-178C maybe a means to achieve airworthiness certifications defined in the Advisory Circulars and TSOs (Ex: AC 20-115C – Draft expected 2012).

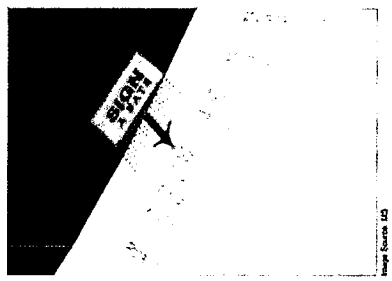
DO-178C does not reference any National or International software standard.

DO-178C does not define any contractual standards. If contractual standards would prevent an applicant from achieving compliance to DO-178C then a certification may not be granted by the certification authority.

Intent for Using DO-178C

Show Compliance with DO-178C:

Compliance is achieved when all applicable objectives have been satisfied by performing all planned activities and capturing the related evidence.



Section 1.4 P

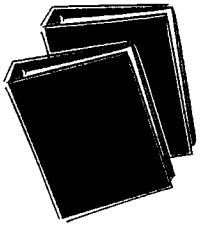
2-10

This slide discusses the intent for using DO-178C.

Relationship to Other Documents

It does *not* define a relationship to:

- Airworthiness requirements
- National and international software standards
- Contractual standards
  e.g. from an engine or aircraft manufacturer



NOTE: Advisory Circular 20-115C "Airborne Software Assurance", July 19, 2013 is available from WWW.FAA.GOV - Search for "AC 20-115C"

2-9

Airworthiness requirements are in the regulations and not defined by DO-178C. DO-178C maybe a means to achieve airworthiness certifications defined in the Advisory Circulars and TSOs (Ex: AC 20-115C "Airborne Software Assurance" July 19, 2013).

DO-178C does not reference any National or International software standard.

DO-178C does not define any contractual standards. If contractual standards would prevent an applicant from achieving compliance to DO-178C then a certification may not be granted by the certification authority.

Purpose of DO-178C (1 of 2)

"Provide *guidance* for production of software for airborne *systems* and equipment

that performs its intended function

with a level of *confidence*[1] in safety that *complies* with airworthiness requirements."

Image Source: iStockphoto

[1] Ref: slide titled "Five Software Levels"

Section 1.1

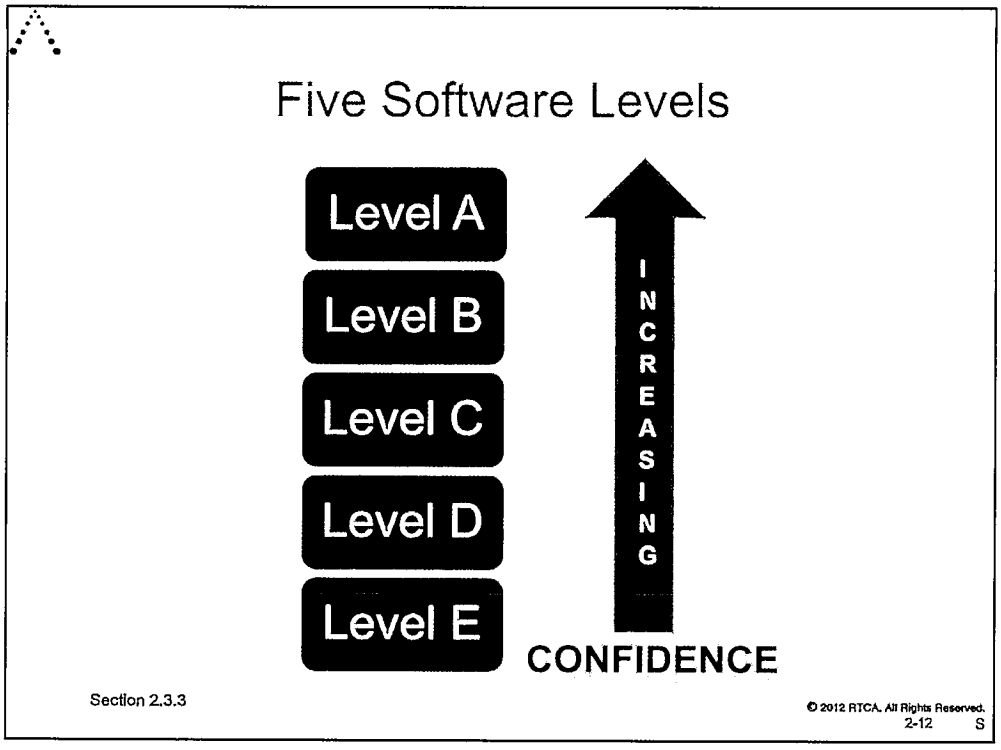
This slide explains the main purpose of DO-178C document.

Have students go to section 1.1 of DO-178C

The guidance is to the applicant/developer and it is for certification.

The second screen slide "Five Software Assurance Levels" should be brought up on the second screen.

The next slide lists six more themes that are the foundation of DO-178C.



Examples of Software Levels for Avionics Systems:

Level A = primary flight displays, altimeters, and fly-by-wire

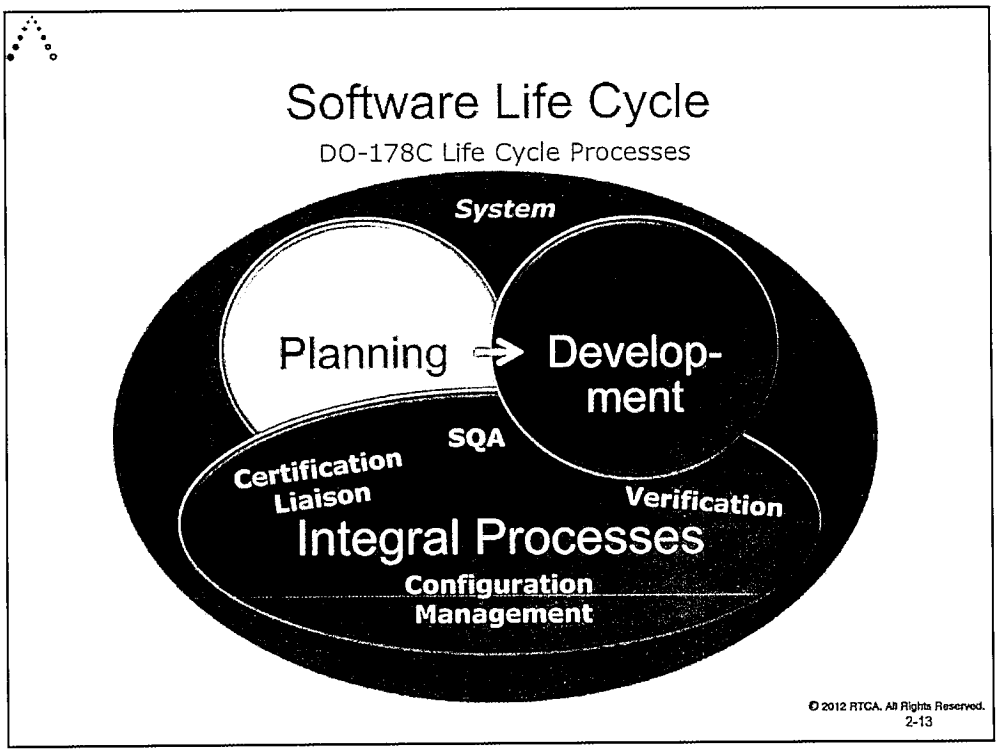
Level B = Traffic Alert and Collision Avoidance System (TCAS)

Level C = Navigation and radio voice communications

Level D = Aircraft Communications Addressing and Reporting System (ACARS)

Level E = software to operate the on board entertainment systems

Level E: Software whose anomalous behavior, as shown by the *system* safety assessment process, would cause or contribute to a failure of *system* function with no effect on aircraft operational capability or pilot workload. **If a software component is determined to be Level E and this is confirmed by the certification authority, no further guidance contained in this document applies.**



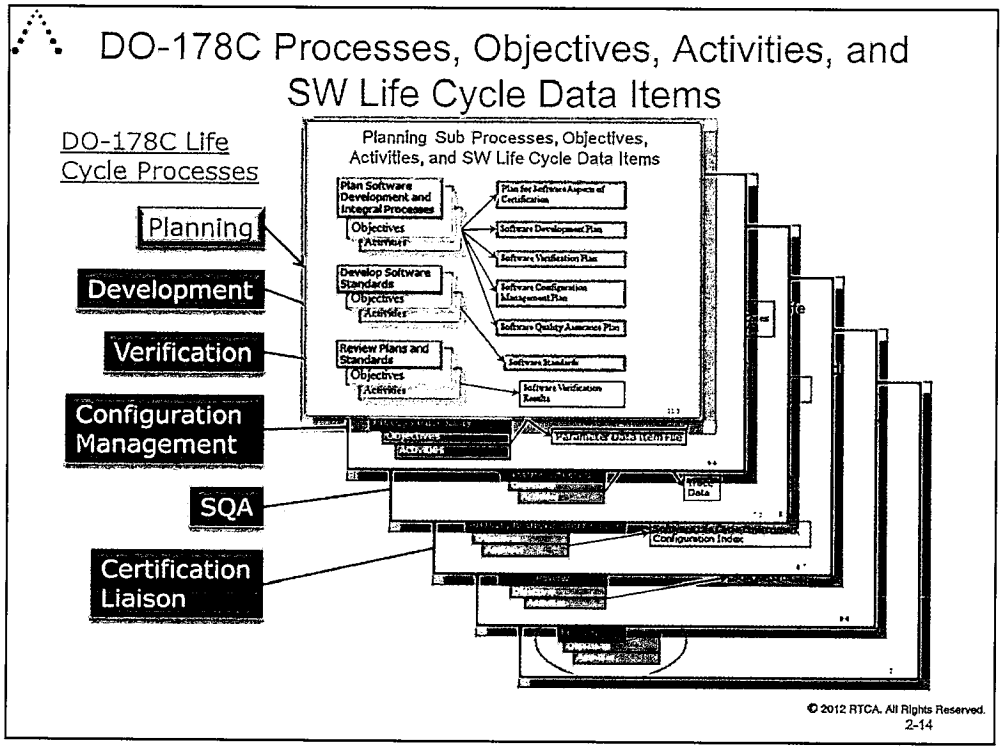
There are many life cycle process charts. This particular slide depicts the life cycle processes in DO-178C that are used for software.

Note: the integral processes interact with the *system*, planning and development processes.

The *system* spans all three software life cycle processes.

The integral processes span the planning and development processes.

This chart is also following the same color-coded scheme as the previous chart in order for the reader to follow the processes all the way through the course.

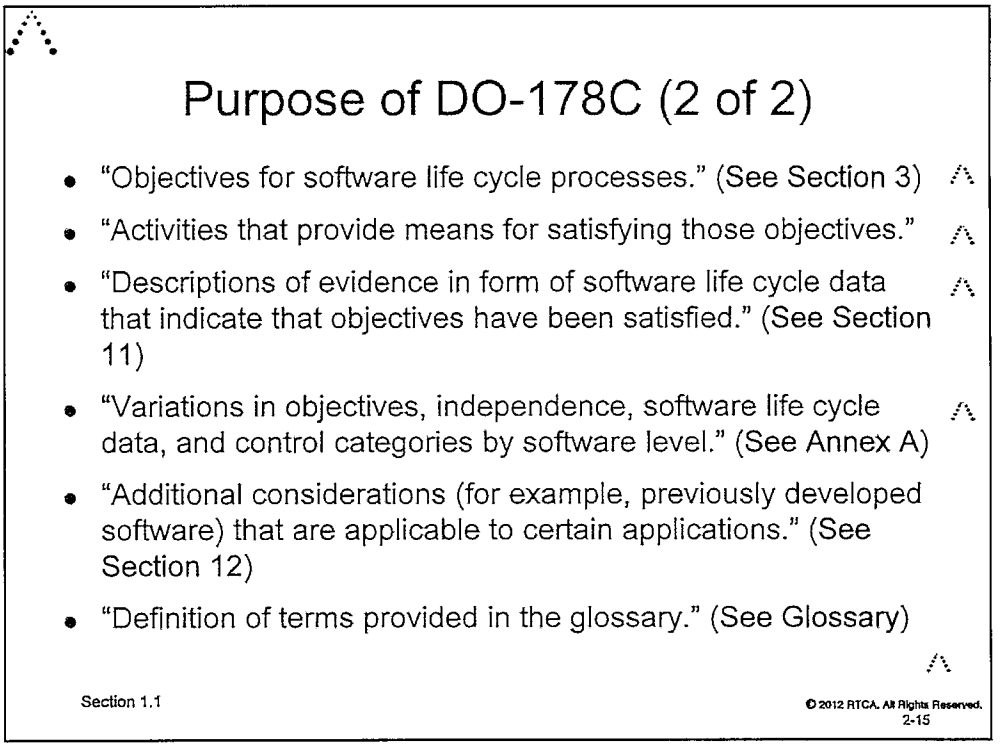


Three sub-processes in the software planning process are:

- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Discuss "activities": Use backup slide defining "activities"

Plans, when followed, need to produce products compliant to DO-178C objectives.

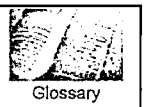


This course will cover all these items in more detail in the subsequent modules.

This module will define key terms used in this course, namely: (Second screen slides will be brought up for each of these items)

- Objectives
- Means
- Activities
- Evidence
- Variations

Understanding these 5 key definitions will be the basis for understanding the entire DO-178C document and are fundamental to the understanding of this course.



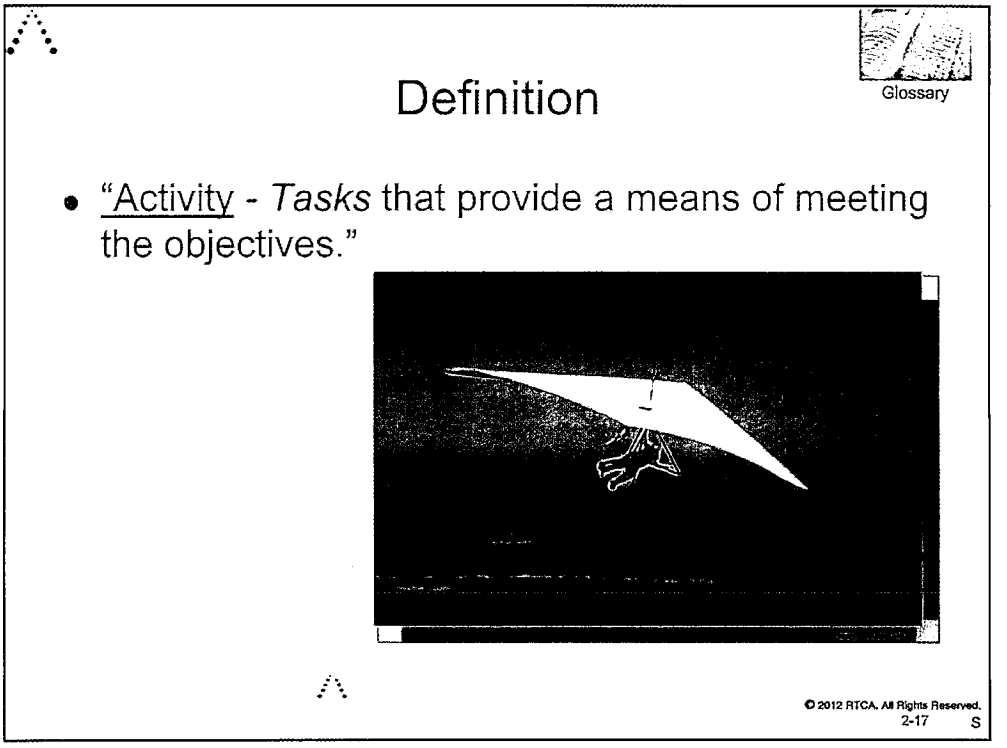
Definitions

- "Objective – When this document is identified as a means of compliance with the regulations, the objectives are *requirements that should be met to demonstrate compliance."*
- "Means of compliance – The intended method(s) to be used by the applicant to satisfy the requirements stated in the certification basis for an aircraft, engine, propeller, or, by region, auxiliary power unit."

Note: There is a difference between the *objectives* and *system/software requirements.*

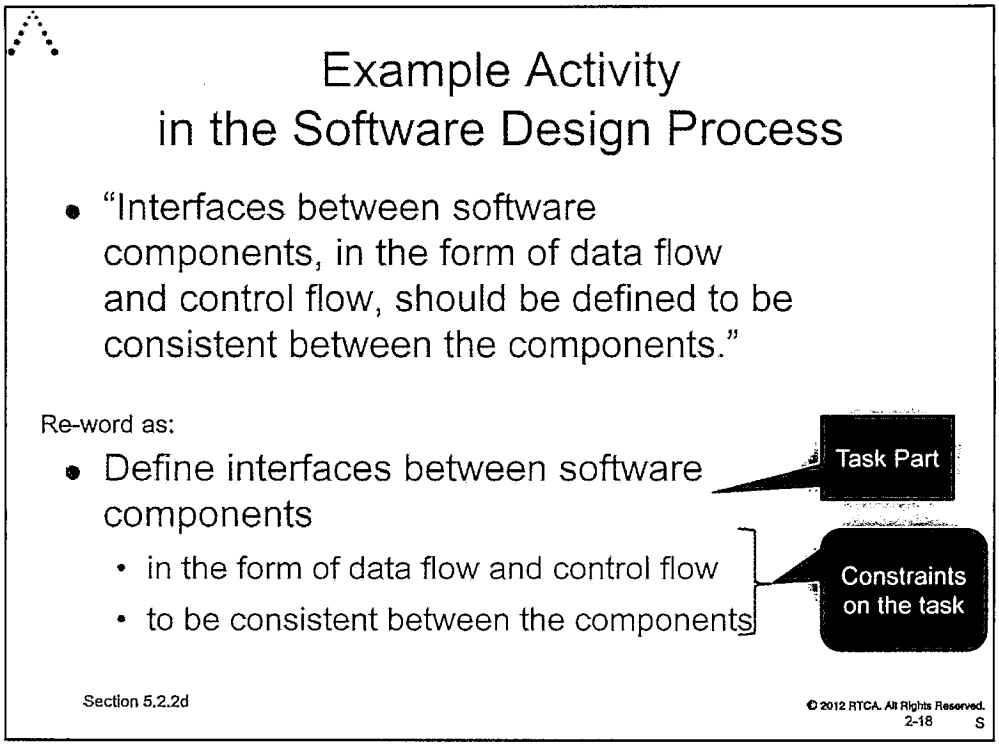
This slide defines "Objectives" as requirements that should be met to demonstrate compliance

NOTE: There is a difference between the objectives as requirements and the *system*/software requirements.



Definition of "Activity" is: Tasks that provide a means of meeting the objectives.

The slide titled "Example Activity In The Design Process" contains a discussion of how activities are stated in DO-178C.



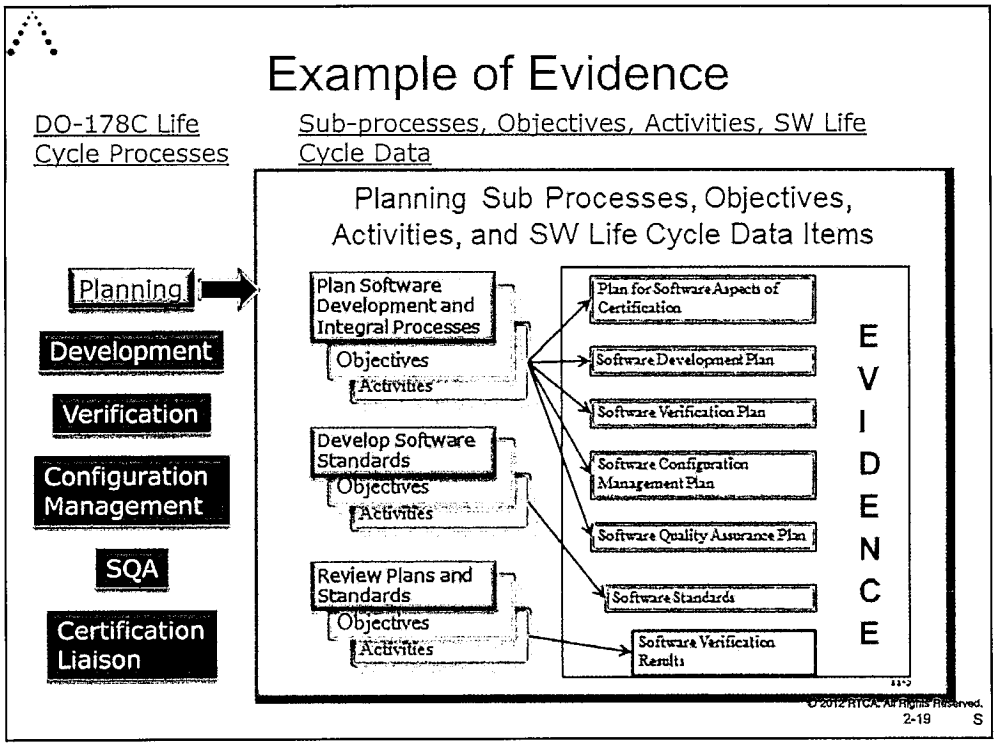
To understand the meaning of activities written in DO-178C, it is useful to think of them as having two parts.

- A Task description that defines an action to be performed
- One or more constraints on the task

The activity may have to be re-worded to highlight the different parts.

Any activity that cannot be re-worded into an actionable part should be considered as one or more constraints on the whole process, sub-process, or sub-sub process containing that activity.

Example of Evidence



This slide show the relationship of the life cycle processes, objectives, activities, and Software Life cycle Data Items (i.e. evidence).

More on these data items will be discussed in future modules.

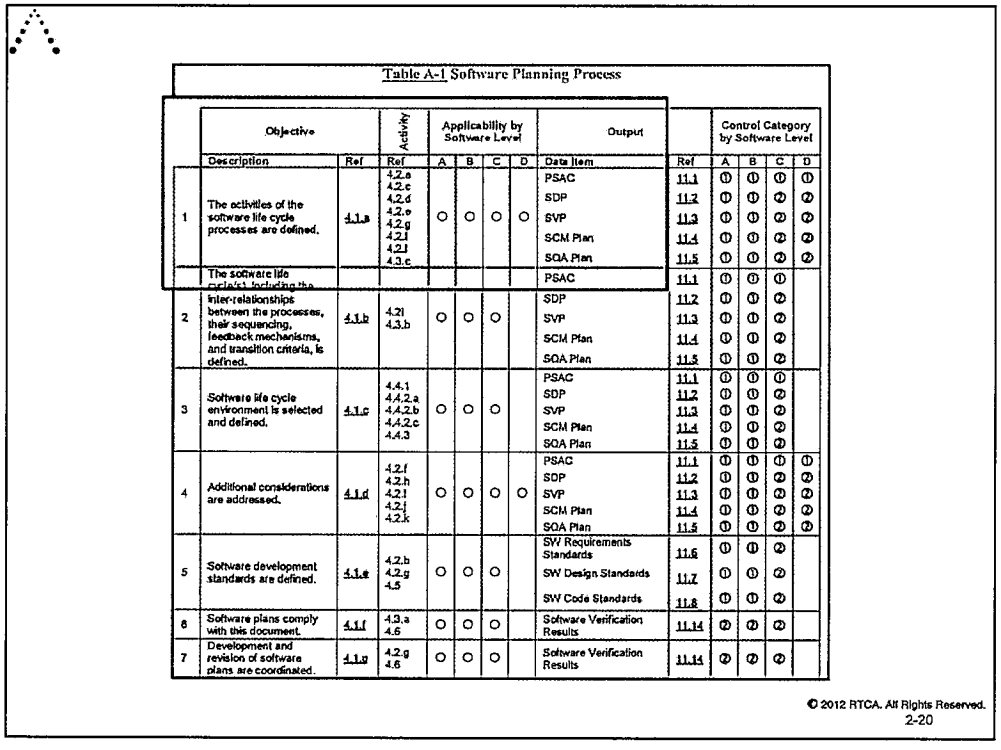
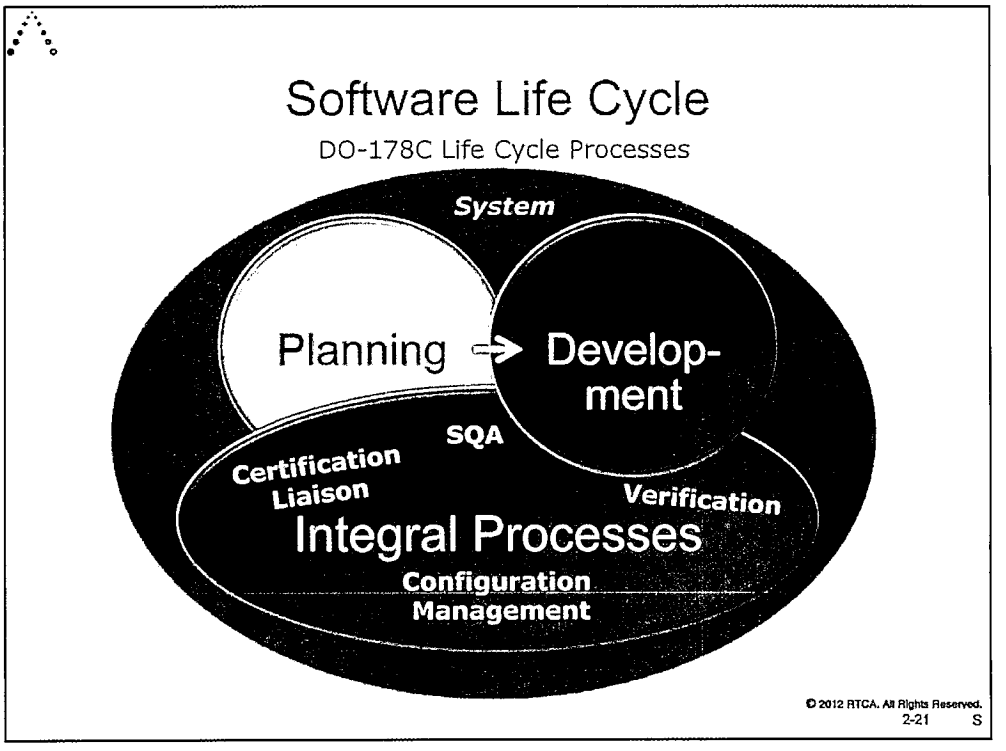


Table A-1 Software Planning Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	The activities of the software life cycle processes are defined.	4.1.a	4.2.a 4.2.c 4.2.d 4.2.e 4.2.g 4.2.i 4.2.l 4.3.c	○	○	○	○	PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ② ②	① ① ② ②	① ② ② ②	① ② ② ②
2	The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is defined.	4.1.b	4.2l 4.3.b	○	○	○		PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ①	① ① ① ①	① ② ② ②	
3	Software life cycle environment is selected and defined.	4.1.c	4.4.1 4.4.2.a 4.4.2.b 4.4.2.c 4.4.3	○	○	○		PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ① ①	① ① ① ① ①	① ② ② ② ②	
4	Additional considerations are addressed.	4.1.d	4.2.f 4.2.h 4.2.l 4.2.j 4.2.k	○	○	○	○	PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ① ①	① ① ① ① ①	① ② ② ② ②	① ② ② ② ②
5	Software development standards are defined.	4.1.e	4.2.b 4.2.g 4.5	○	○	○		SW Requirements Standards SW Design Standards SW Code Standards	11.6 11.7 11.8	① ① ①	① ① ①	② ② ②	
6	Software plans comply with this document.	4.1.f	4.3.a 4.6	○	○	○		Software Verification Results	11.14	② ② ②	② ② ②	② ② ②	
7	Development and revision of software plans are coordinated.	4.1.g	4.2.g 4.6	○	○	○		Software Verification Results	11.14	② ② ②	② ② ②	② ② ②	

Annex A Tables show the variation of objectives by software Level

Open circle is "achieve the objective" while a filled in circle (not shown) is "achieve the objective with independence"



There are many life cycle process charts. This particular slide depicts the life cycle processes in DO-178C that are used for software.

Note: the integral processes interact with the *system*, planning and development processes.

The *system* spans all three software life cycle processes.

The integral processes span the planning and development processes.

This chart is also following the same color-coded scheme as the previous chart in order for the reader to follow the processes all the way through the course.

DO-178C Document Organization

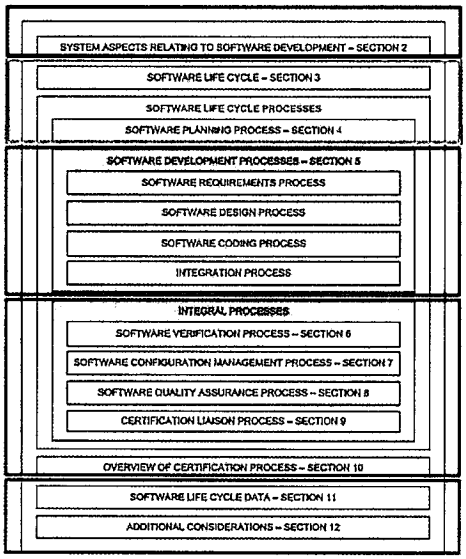


Figure 1-1

This slide explains that DO-178C can be broken into five parts (color-coded throughout the course).

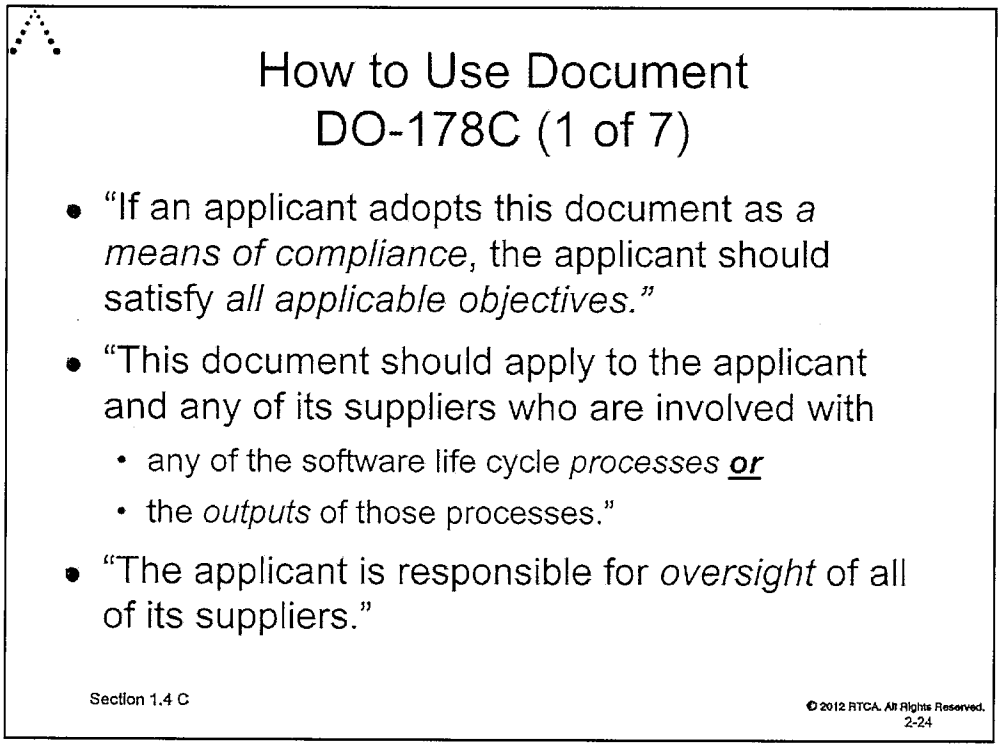
- Interfacing with *system* processes, including system data items (**Red**)
- Planning – defining processes and activities for the other parts of this document (**Orange**)
- Software Development (**Green**)
- Integral processes, including verification, Configuration management, SQA, and Certification liaison (**Blue**)
- Other topics, including Software Life Cycle Data and Additional Considerations (for example: previously developed software, when to do tool qualification, service history,...) (**Black**)

- Data items coming from a single process are color-coded the color of that process.
- Data items coming from multiple processes are color-coded **Black**.

HOW TO USE DOCUMENT DO-178C

2-23

The following 7 charts describe how to use the document.



The two topics of objectives and activities will now be examined in more depth.

First, all applicable objectives need to be satisfied by the applicant.

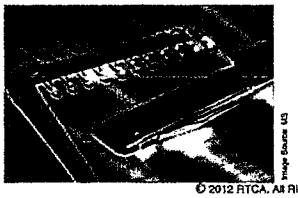
Do-178C also applies to any suppliers and the *applicant* is responsible to oversee the suppliers.

The next chart discusses activities.

How to Use Document DO-178C (2 of 7)

- "The applicant should plan a set of activities that satisfy the objectives."
  - "This document describes activities for achieving those objectives."
  - "The applicant may plan and, *subject to the approval of the certification authority*, adopt alternative activities to those described in this document."
  - "The applicant may also plan and conduct additional activities that are determined to be necessary."

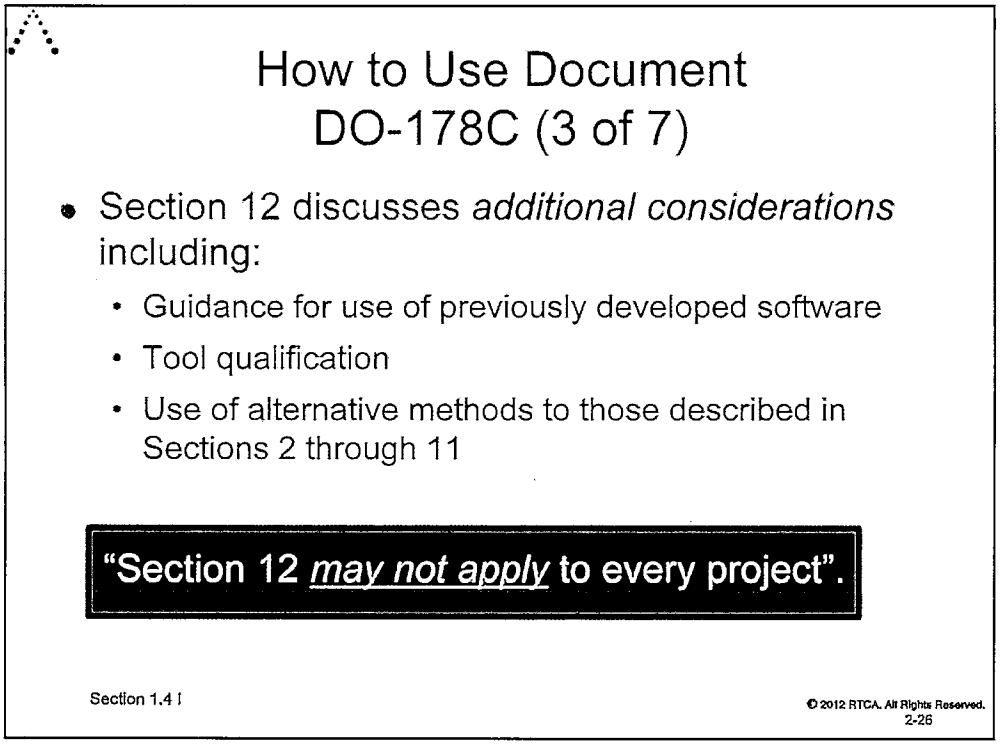
The activities in the software plans, provide direction to the personnel performing the software life cycle processes. A plan for an activity should be in place before the activity starts.



Section 1.4 D

2-25

- Some or all of activities list in sections 4 – 10 may be used by the applicant. These activities may not be the complete set of activities needed by the applicant in order to be compliant with DO-178C.
- The software plans should be developed that provide direction to the personnel performing the software life cycle processes. **All** planning does not have to be complete before any Software Life Cycle process may be started. An example would be 'verification planning does not have to be complete before development starts'. The applicant may do life cycle processes and the activities of a life cycle process in the order that best suits their own software life cycle. However it is sound practice that a software life cycle process activity not be undertaken before that activity is described in a software plan.
- The applicant may add delete, or change activities in order to meet the objectives of DO-178C and get approval for the applicant's set of activities by means of the Plan for Software Aspects of Certification (PSAC).
- Additional activities will be explained next in more detail.



The topic, *Additional Considerations*, is covered later on in Module 12 of this course. The three areas we will be discussing when we cover this topic include:

- Guidance for use of previously developed software
- Tool qualification
- Use of alternative methods to those described in sections 2 through 11

The next chart reviews how to use the charts in Annex A.

How to Use Document DO-178C (4 of 7)

- "Annex A specifies the applicability of the objectives, activities, and software life cycle data for each software level as well as the variation in the independence and control categories for each software level."
- "In order to fully understand the guidance, the full body of this document should be considered."

Section 1.4 J

2-27

See slide "Variations in Objectives, ...

The body of DO-178C is written for Level A software. Compliance with this document for other software levels must be tailored using Annex A.

Variations in Objectives, …

Table A-3 Verification of Outputs of Software Requirements Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	High-level requirements comply with system requirements.	6.3.1.a	6.3.1	●	●	○	○	Software Verification Results	11.14	②	②	②	②
2	High-level requirements are accurate and consistent.	6.3.1.b	6.3.1	●	●	○	○	Software Verification Results	11.14	②	②	②	②
3	High-level requirements are compatible with target computer.	6.3.1.c	6.3.1	○	○			Software Verification Results	11.14	②	②		
4	High-level requirements are verifiable.	6.3.1.d	6.3.1	○	○	○		Software Verification Results	11.14	②	②	②	
5	High-level requirements conform to standards.	6.3.1.e	6.3.1	○	○	○		Software Verification Results	11.14	②	②	②	
6	High-level requirements are traceable to system requirements.	6.3.1.f	6.3.1	○	○	○	○	Software Verification Results	11.14	②	②	②	②
7	Algorithms are accurate.	6.3.1.g	6.3.1	●	●	○		Software Verification Results	11.14	②	②	②	

LEGEND:
- ● The objective should be satisfied with independence.
- ○ The objective should be satisfied.
- Blank Satisfaction of objective is at applicant's discretion.
- ① Data satisfies the objectives of Control Category 1 (CC1).
- ② Data satisfies the objectives of Control Category 2 (CC2).

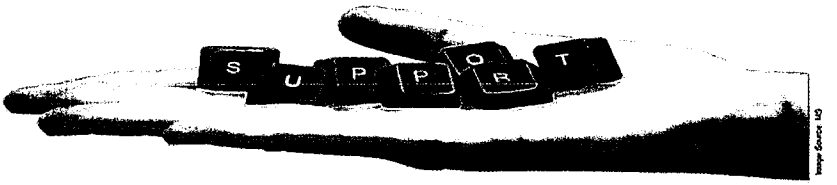
Annex A

- For each objective there is an entry in an Annex A Table. The one pictured here is from Table 4, Rows 1 and 2 (An Objective of the Verification of Outputs of Software Design Process).
- The variation is shown in the main column title "Applicability by Software Level". In this example, if the software is Software Level D, then this objective is not required to be met. If the Software Level is Software Level A, then it is required to be met.
- This example shows a filled circle which means that independence is required in the activities used to meet this objective for software level A and B.

> "Independence – Separation of responsibilities which ensures the accomplishment of objective evaluation. (1) For software verification process activities, independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified, and a tool(s) may be used to achieve equivalence to the human verification activity. (2) For the software quality assurance process, independence also includes the authority to ensure corrective action."

How to Use Document DO-178C (5 of 7)

- "In cases where examples are used to indicate how the guidance might be applied, either graphically or through narrative, the examples are **not** to be interpreted as the *preferred method.*"
  - "In these cases, the examples are considered ***supporting information.***"



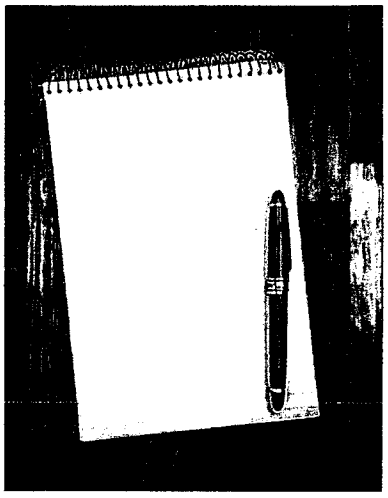
Section 1.4 K

2-29

This chart discusses the use of examples in DO-178C and cautions the reader to not miss-apply them as a preferred method. Rather, they are just meant to be used as supporting materials.

How to Use Document DO-178C (6 of 7)

"A list of items does ***not*** imply the list is all-inclusive."



Section 1.4 L

Image Source: MS

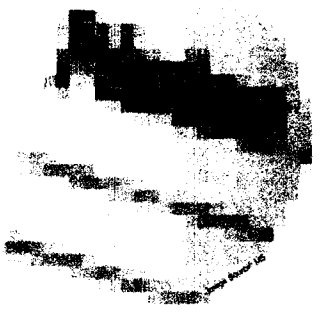
2-30

This chart discusses the use of the word "list" and how it is applied in DO-178C.

This will also hold true for lists in this course.

How to Use Document DO-178C (7 of 7)

- "Notes are *supporting information* used to:
  - Provide explanatory material,
  - Emphasize a point, or
  - Draw attention to related items which are not entirely within context."

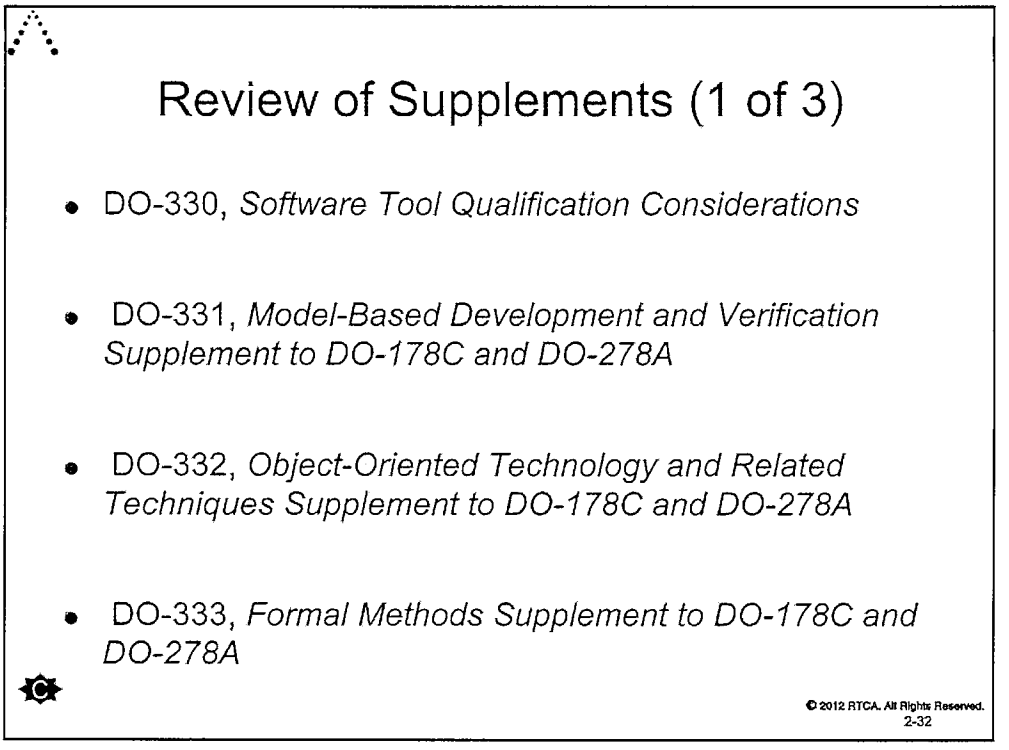


Section 1.4 M

2-31

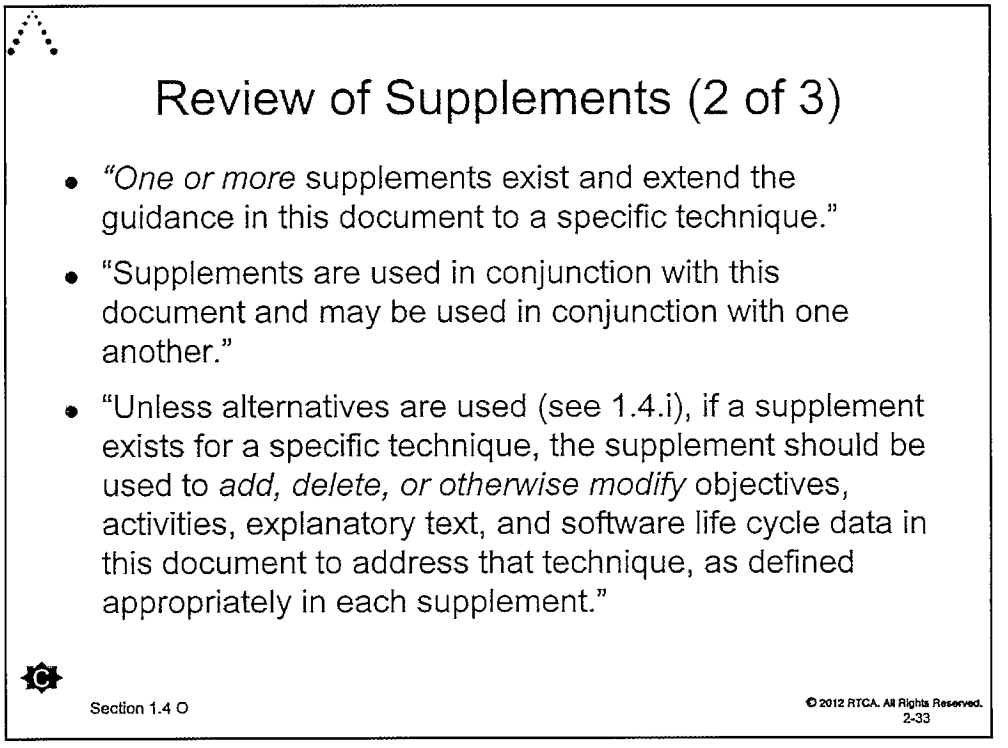
This chart discusses the use of DO-178C's notes. Mainly NOTES are to emphasize a point and provide more information about items.

The four supplements to DO-178C are reviewed next.

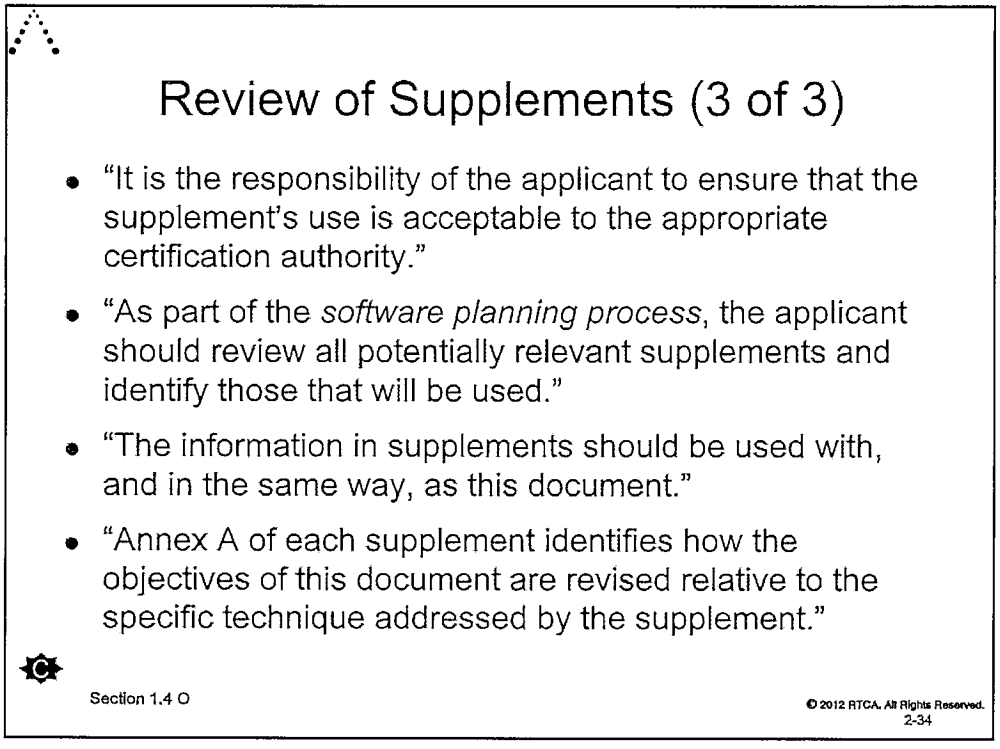


This slide and the following 2 slides review the four supplements to DO-178C.

DO-278A *"Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems"* is similar to DO-178C but for ground-based CNS/ATM systems.



Refs of supplement objectives are form MB1-2, verses DO-178C simply 1-2. It at least will give the participants an effective method for handling the multitude of DO-178C and supplement objectives.



The supplements used on a program must be indicated by the applicant in the Plan for Software Aspect of Certification (PSAC). The PSAC will be described in Module 5, Planning.

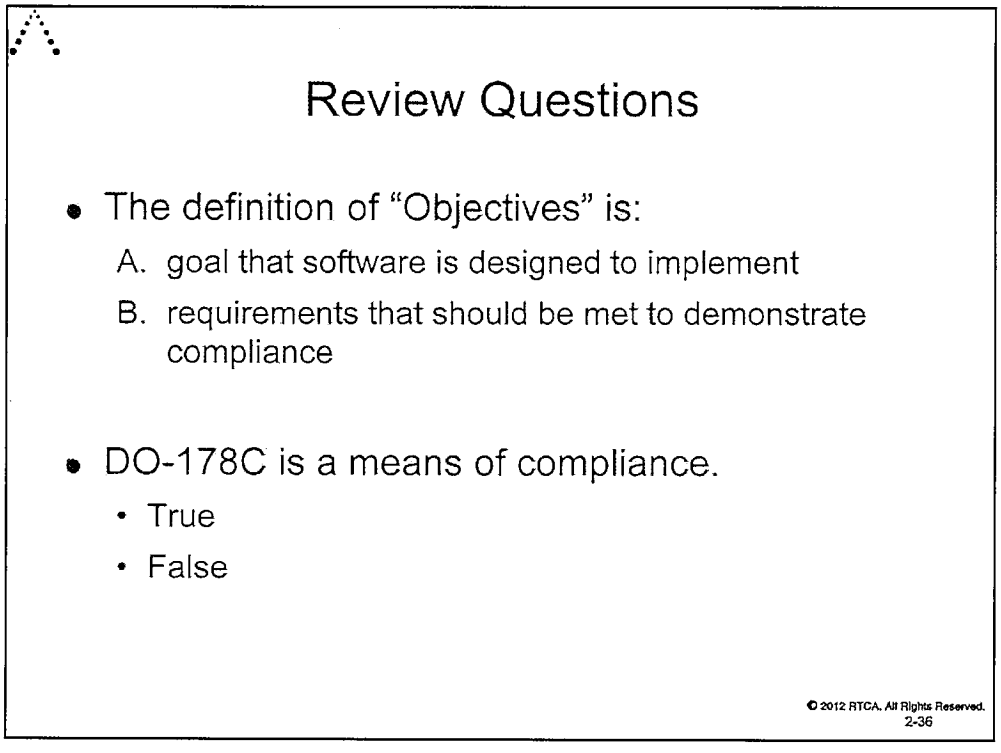
Summary

Compliance with DO-178C is "achieved when all applicable objectives have been satisfied by performing all planned activities and capturing the related evidence."

Section 1.4 P

2-35

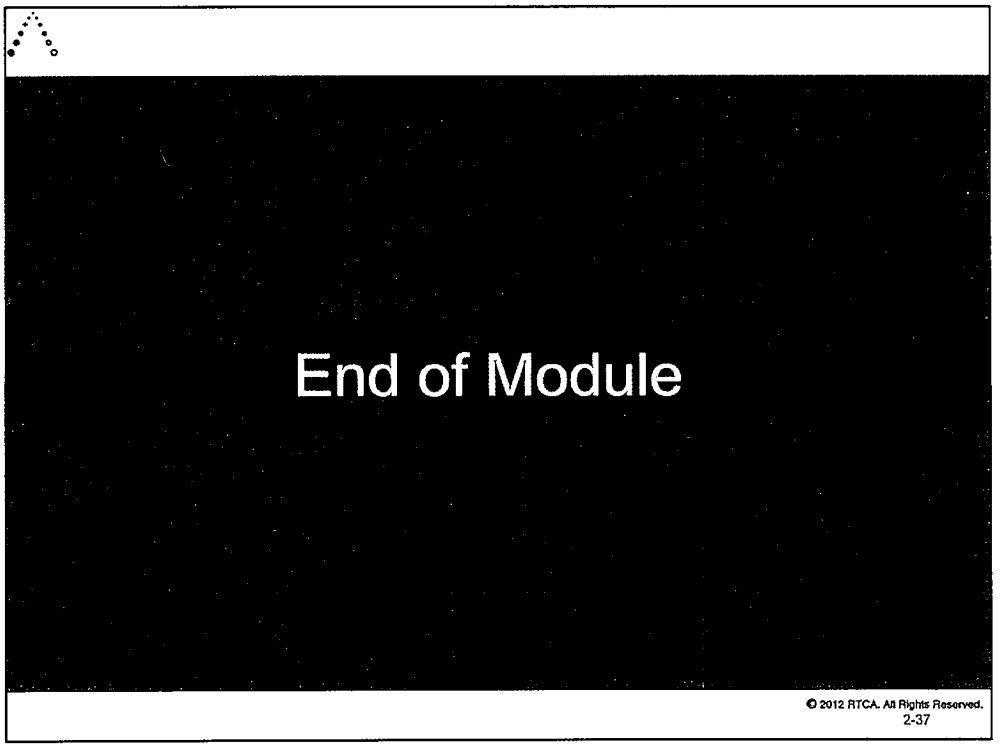
Module Summary

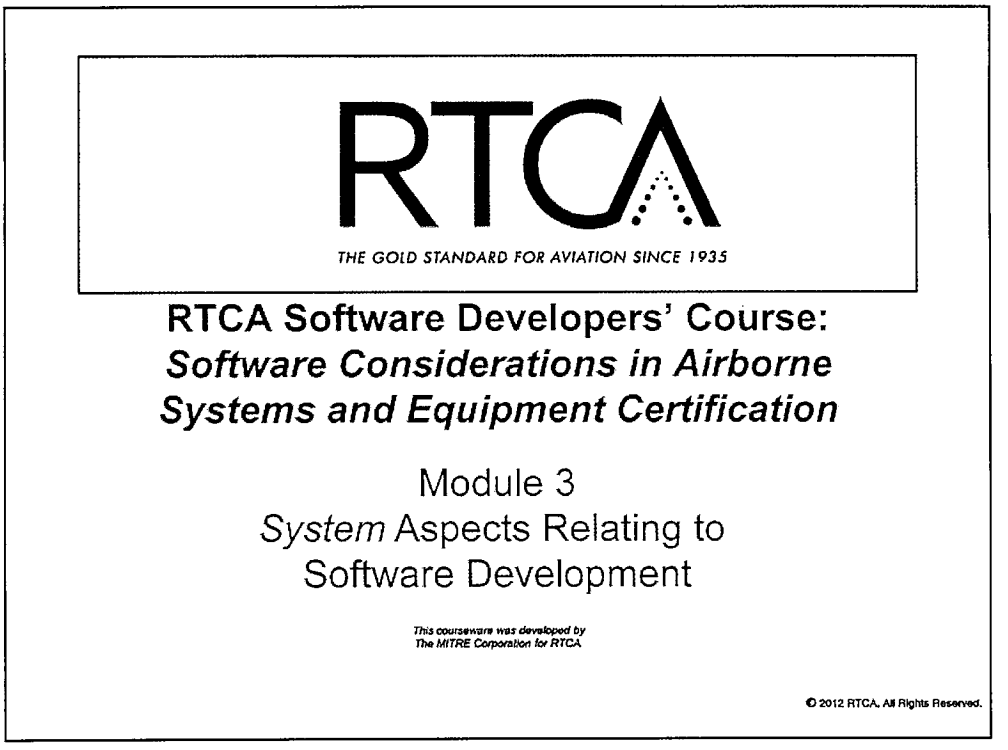


Answers:

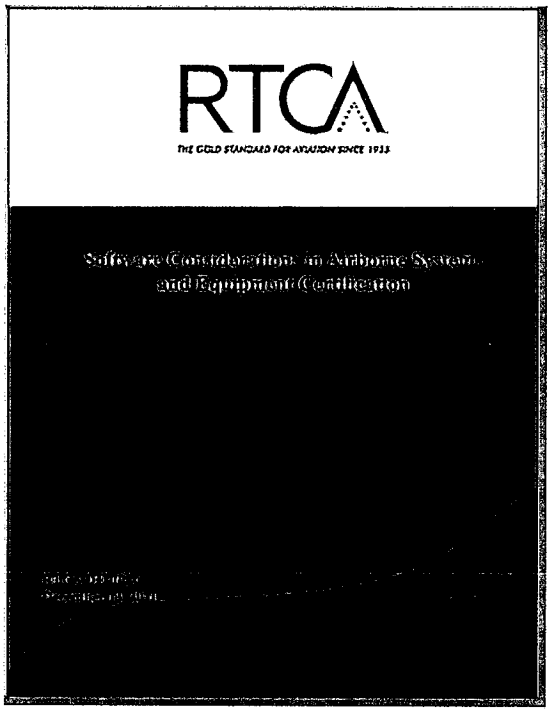
B.

True.

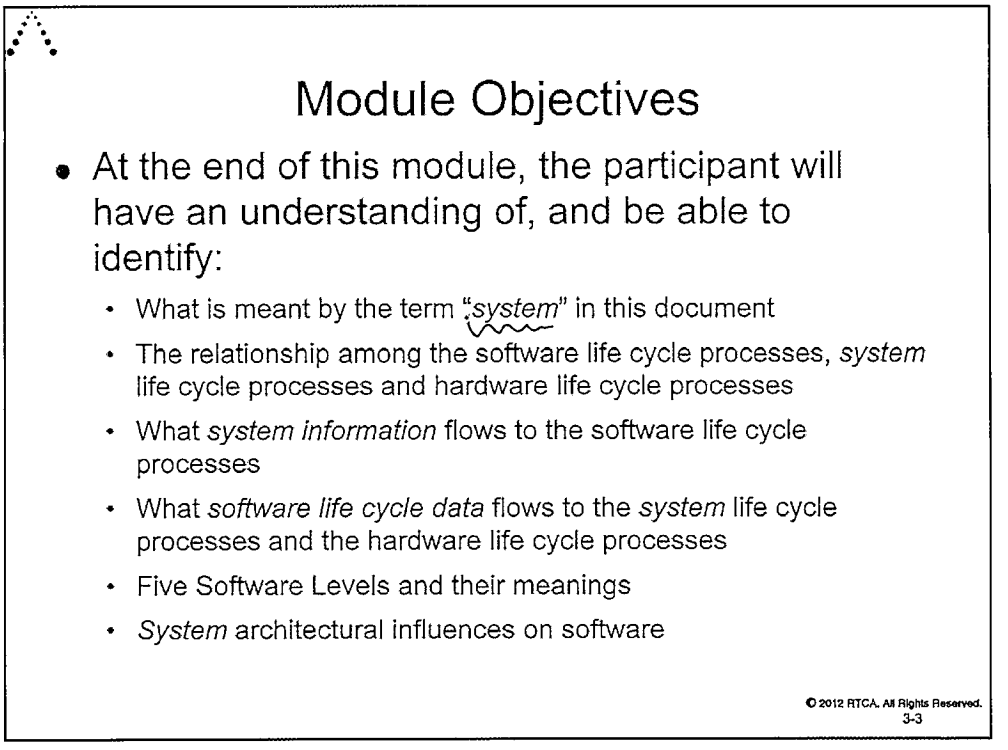

End of Module
© 2012 RTCA. All Rights Reserved.
2-37



This 1-hour 30 minute presentation introduces the definition of "*system*" as used in this document. The word "*system*" is italicized throughout the entire course to ensure that it is clear that it is separate but influences the software considerations. *System* processes and *system* requirements are distinguished. The information flow among the three life cycle processes, *system*, software, and hardware are presented. The relationship of *system* safety and software levels is detailed along with *system* architectural considerations. Finally, software considerations in *system* life cycle processes and *system* considerations in software life cycle processes are described.



Section 2



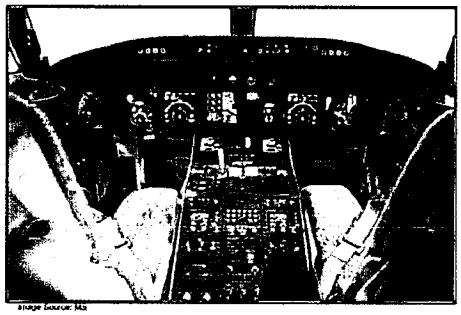
After completing module 3, the participant will have an understanding of these topics. This is also the outline of the module.

Scope of "*system*"

Refers to the airborne *systems* and equipment ***only***

System life cycle processes can be found in other documents, e.g. SAE ARP4754A

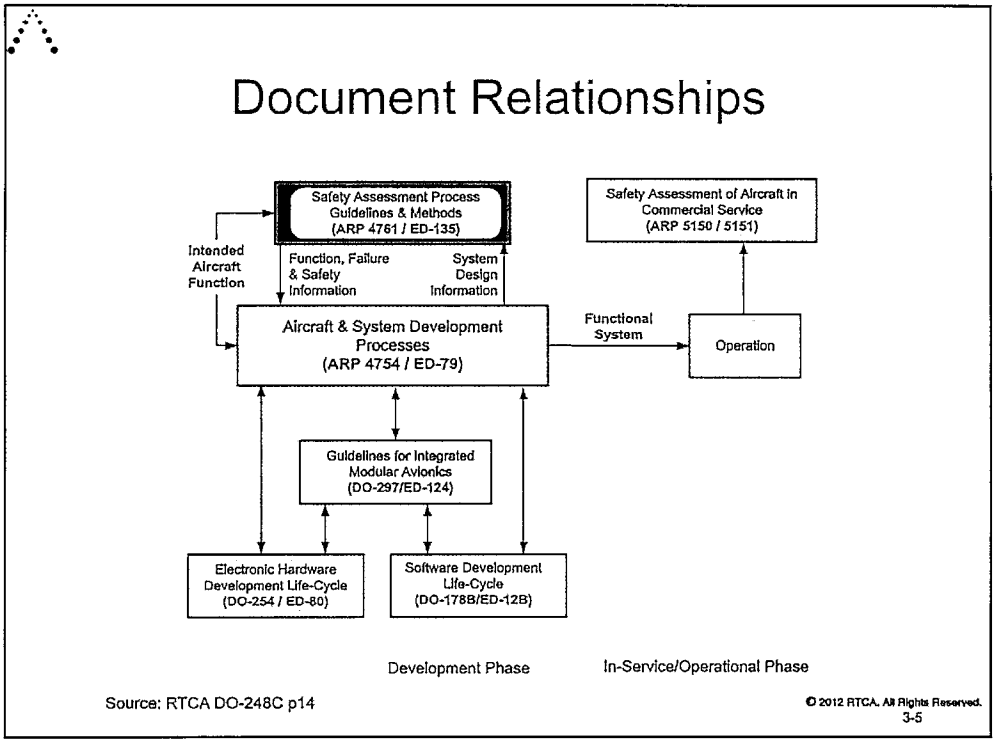
*Does **not** include the wider definition of a system that might include operators, operational procedures, etc.*



Aerospace Recommended Practices (ARP) are recommendations for engineering practice.

3-4

DO-178C defines system as "a collection of hardware and software components organized to accomplish a specific function or set of functions." The "*system*" takes into account the user interfaces, procedures, etc. when determining the requirements allocated to software. Hence, these have already been considered and are not an issue for the software life cycle processes. *System* requirements validation are not within the scope of DO-178C.



This diagram from DO-248C, which originated in ARP 4754A, shows the differentiation between aircraft functions and the software development life cycle. Other documents, including other RTCA documents, may drive requirements for a particular *system*. The "Guidelines for Integrated Modular Avionics (DO-297/ED-124)" is an example.

DO-248C refers to DO-178B, though that reference should now be to DO-178C.

Note that SAE's ARP 4754's current version is "A". Also SAE's ARP 4761 is currently undergoing revision.

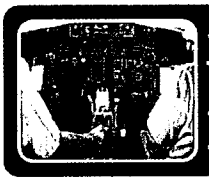
Aerospace Recommended Practices (ARP)

This module discusses those aspects of the *system* life cycle processes necessary to understand the software life cycle processes.

3-6

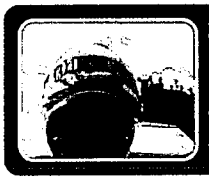
The intent of this course is not to train students in *system* life cycle processes, but it is important for those involved with the software life cycle processes to have a basic understanding of how the *system* life cycle processes can affect their activities.

System requirements allocation to software



System requirements are developed from:

- ***System*** **operational requirements**
- **Safety, security and performance requirements**



Safety related requirements may include:

- **Functional, integrity and reliability requirements**
- **Design constraints**

Image Source: M3

3-7

The *system* processes decides which requirements need to be implemented by software and passes these requirements to the software processes. These requirements are called 'system requirements allocated to software' by DO-178C.

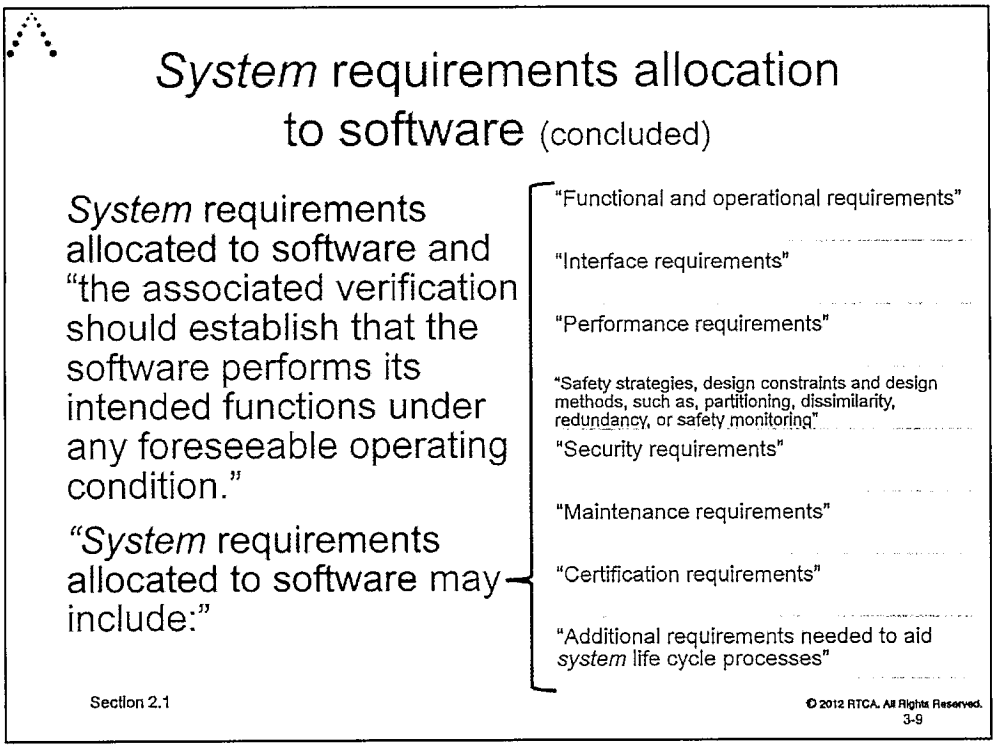
System requirements allocation to software (continued)

- "The *system* safety assessment process determines and categorizes the failure conditions of the *system*"
- "Safety-related requirements are defined to ensure the integrity of the *system* by specifying the desired immunity from, and *system* responses to, these failure conditions"
- "These requirements are identified for hardware and software to preclude or limit the effects of faults, and may provide fault detection, fault tolerance, fault removal, and fault avoidance"
- "The *system* processes are responsible for the refinement and allocation of *system* requirements to hardware and/or software as determined by the *system* architecture"

Section 2.1

3-8

The *system* safety assessment determines how the *system* might fail, and the consequences of those failures. Protections from undesired failures are developed and are allocated to software as appropriate.



There are various types of requirements developed to protect the *system* from undesirable failures.

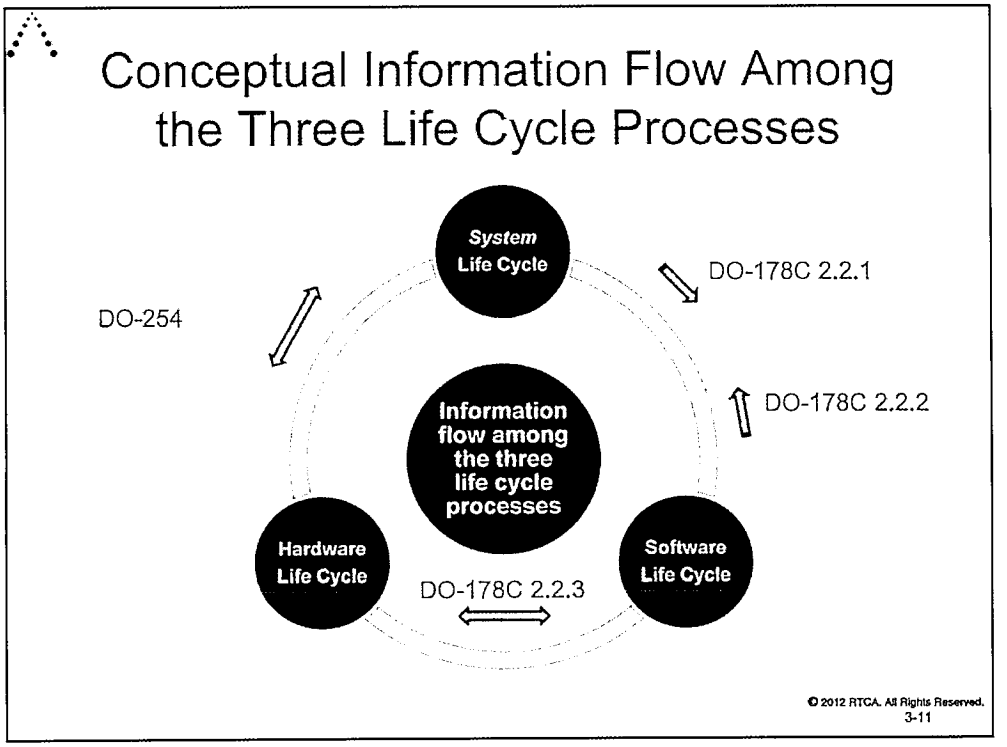
Upcoming… Exercises!

- *System* requirements to be used in the upcoming exercises:

1) *System* requirements allocated to software:
   - The equipment shall read pressure from the pressure sensor in pounds per square inch.
   - The equipment will calculate altitude in hundreds of feet from pressure in pounds per square inch.
   - The equipment shall display altitude in hundreds of feet.

2) Assumptions for the exercises:
   - A software function exists that can read pressure in PSI
   - An output to display altitude in 100s of feet function exists

In later modules, the students will work an exercise through the software life cycle process. These are the *system* requirements allocated to software that will be used in the exercise. These will be repeated later for the students' use.



This is a conceptual diagram illustrating the flow of information between life cycle processes. Also illustrated are the DO-178C sections further describing the information flows. Note that DO-178C does not address information flow between the *system* life cycle and the hardware life cycle processes.

Information Flow Among the Three Life Cycle Processes

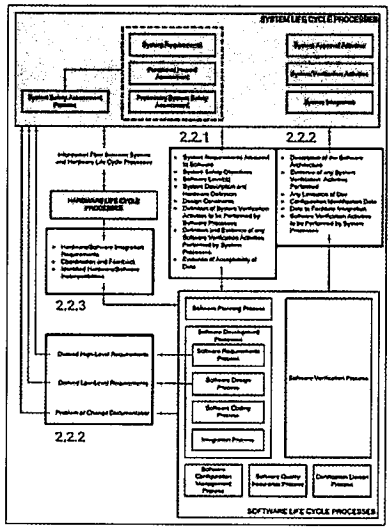
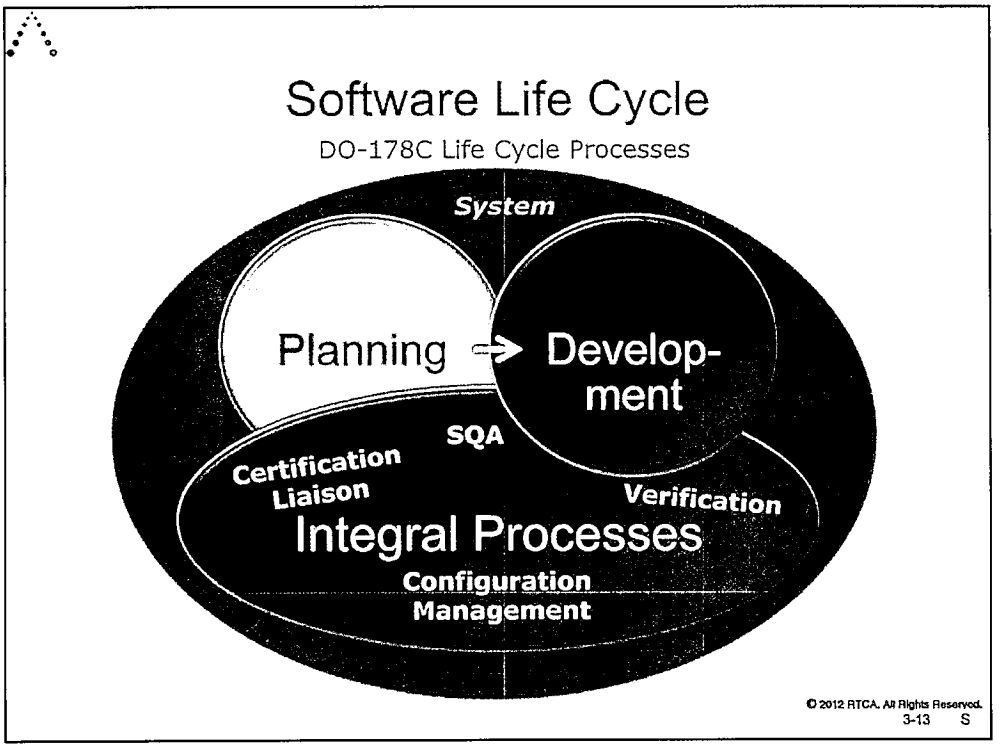


Figure 2-1, page 9

3-12

This is a more detailed diagram from DO-178C where we have highlighted the document sections addressing the different information flows.



There are many life cycle process charts. This particular slide depicts the life cycle processes in DO-178C that are used for software.

Note: the integral processes interact with the *system*, planning and development processes.

The *system* spans all three software life cycle processes.

The integral processes span the planning and development processes.

This chart is also following the same color-coded scheme as presented in module 2 to allow the reader to follow the processes all the way through the course.

Information Flow From *System* Processes to Software Processes (2.2.1)

"The following data is passed to the software life cycle processes by the *system* processes either as part of the requirements allocation or during the development life cycle:"

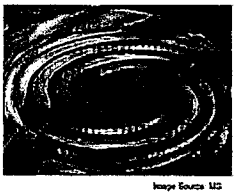
- "*System* requirements allocated to software"
- "*System* safety objectives"
- "Software level for software components and a description of associated failure condition(s), if applicable"
- "*System* description and hardware definition"
- "Design constraints, including external interfaces, partitioning requirements, etc."
- "Details of any *system* activities proposed to be performed as part of the software life cycle. **Note that *system* requirement validation is not usually part of the software life cycle processes"**
- "Evidence of the acceptability, or otherwise, of any data provided by the software processes to the *system* processes on which any activity has been conducted by the *system* processes"
- "Evidence of software verification activities performed by the *system* life cycle processes, if any"

Section 2.2.1

3-14

System requirement validation is normally a *system* life cycle process and is not usually part of the software life cycle process. The *system* level personnel should validate and verify that the requirements they "flow down" are accurate and complete. The software level personnel then endeavor to meet those requirements, as given. The software process does not validate the *system* requirements allocated to software. The software process will verify the *system* requirements allocated to software.

Information Flow From Software Processes to *System* Processes (2.2.2)



- "The software life cycle processes analyze the *system* requirements allocated to software as part of the software requirements process"
- "If such an analysis identifies any *system* requirements as inadequate or incorrect, the software life cycle processes should capture the issues and refer them to the *system* processes for resolution"
- "Furthermore, as the software design and implementation evolves, details are added and modifications made that may affect *system* safety assessment and *system* requirements"

Section 2.2.2

Software level personnel may determine that certain requirements received from the *system* level are inadequately or incorrectly specified. In this case, it is the responsibility of the software level personnel to refer those issues back to the *system* level personnel for reconciliation.

Information Flow From Software Processes to *System* Processes (2.2.2)

To aid the evaluation of the evolving design and changes to the design, the software life cycle processes should make data available to the *system* processes. Such data includes:

- "Details of **derived requirements** created during the software life cycle processes"
- "A description of the software architecture, including software partitioning"
- "Evidence of *system* activities performed by the software life cycle processes, if any"
- "Problem or change documentation, including problems identified in the *system* requirements allocated to software and identified incompatibilities between the hardware and the software"
- "Any limitations of use"
- "Configuration identification and any configuration status constraints"
- "Performance, timing, and accuracy characteristics"
- "Data to facilitate integration of the software into the *system*"
- "Details of software verification activities proposed to be performed during *system* verification, if any"

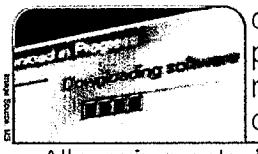
Section 2.2.2

3-16

The DO-178C/DO-278A glossary defines derived requirements as: *"Requirements produced by the software development processes which (a) are not directly traceable to higher level requirements, and/or (b) specify behavior beyond that specified by the system requirements or the higher level software requirements."*

During the software development process, requirements may be developed that do not trace directly to higher level requirements. Development of these derived requirements is typically based upon design, performance, or architectural decisions and therefore not directly traceable to higher level requirements. Even though a derived requirement may not be traced to higher level requirements, the derived requirement may affect the higher level requirements. DO-178C/DO-278A section 5 gives the following example: *"the need for interrupt handling software to be developed for the chosen target computer."* In this example, the need for interrupt handling software will not appear in the higher level requirements, but it will be specified in the lower level requirements as derived requirements.

Information Flow Between Software Processes and Hardware Processes (2.2.3)



"Data is passed between the software life cycle process and the hardware life cycle process either as part of the *system* requirements allocation or during the development life cycles. Such data includes:

- All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the interface between hardware and software
- Instances where hardware and software verification activities require coordination
- Identified incompatibilities between the hardware and the software"

Section 2.2.3

Derived requirements produced by the software development processes may affect hardware/software integration issues and should be communicated to the hardware processes personnel upon identification.

System Safety Assessment and Software Levels

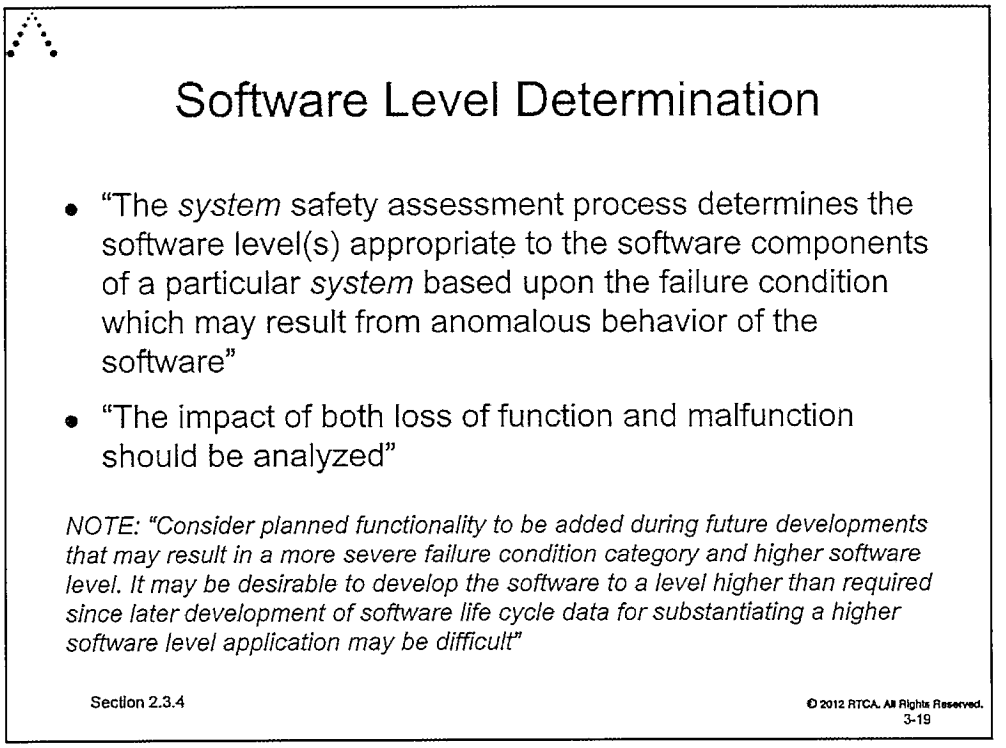
- The *system* safety assessment process requires that software be associated with a failure condition category and be assigned a software level
- Software levels are associated with an appropriate design assurance rigor
- Since software associated with lower hazard categories does not require the same rigor as higher hazard categories, DO-178C/DO-278A provides varying levels of rigor

Section 2.3

3-18

The next few slides provide a brief introduction to how the software level for software components is determined and how architectural considerations may influence the allocation of a software level. It is not the intent of this document to prescribe how these activities are performed; this has to be established and performed as part of the *system* life cycle processes.

The software level of a software component is based upon the contribution of software to potential failure conditions as determined by the *system* safety assessment process by establishing how an error in a software component relates to the *system* failure condition(s) and the severity of that failure condition(s). The software level establishes the rigor necessary to demonstrate compliance with DO-178C.



If the anomalous behavior of a software component contributes to more than one failure condition, then the software component should be assigned the software level associated with the most severe failure condition to which the software can contribute, including combined failure conditions.

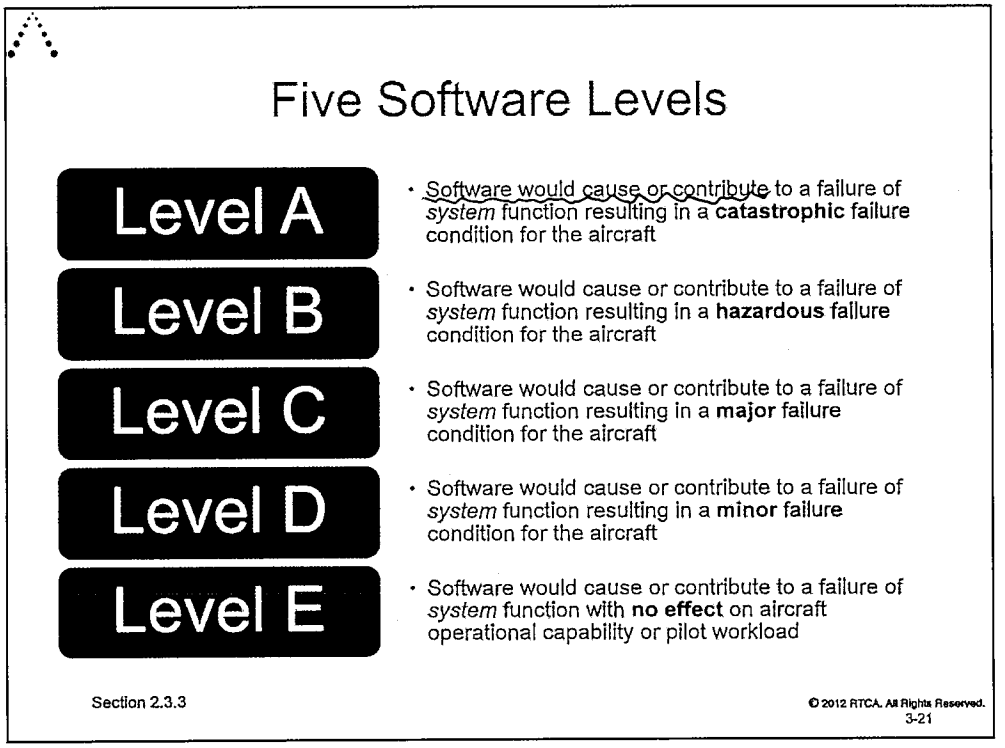
Five Failure Condition Categories

Category	Description
Catastrophic	Failure Conditions, which would result in **multiple fatalities**, usually with the loss of the airplane
Hazardous	Failure Conditions, which would reduce the capability of the airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be: • **A large reduction in safety margins** or functional capabilities; • Physical distress or excessive workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely, or • **Serious or fatal injury** to a relatively small number of the occupants other than the flight crew
Major	Failure Conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a **significant reduction in safety margins** or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to the flight crew, or physical distress to passengers or cabin crew, possibly including injuries
Minor	Failure Conditions which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities. Minor Failure Conditions may include, for example, a **slight reduction in safety margins** or functional capabilities, a slight increase in crew workload, such as routine flight plan changes, or some physical discomfort to passengers or cabin crew
No Safety Effect	Failure Conditions that would have **no effect on safety**; for example, Failure Conditions that would not affect the operational capability of the airplane or increase crew workload.

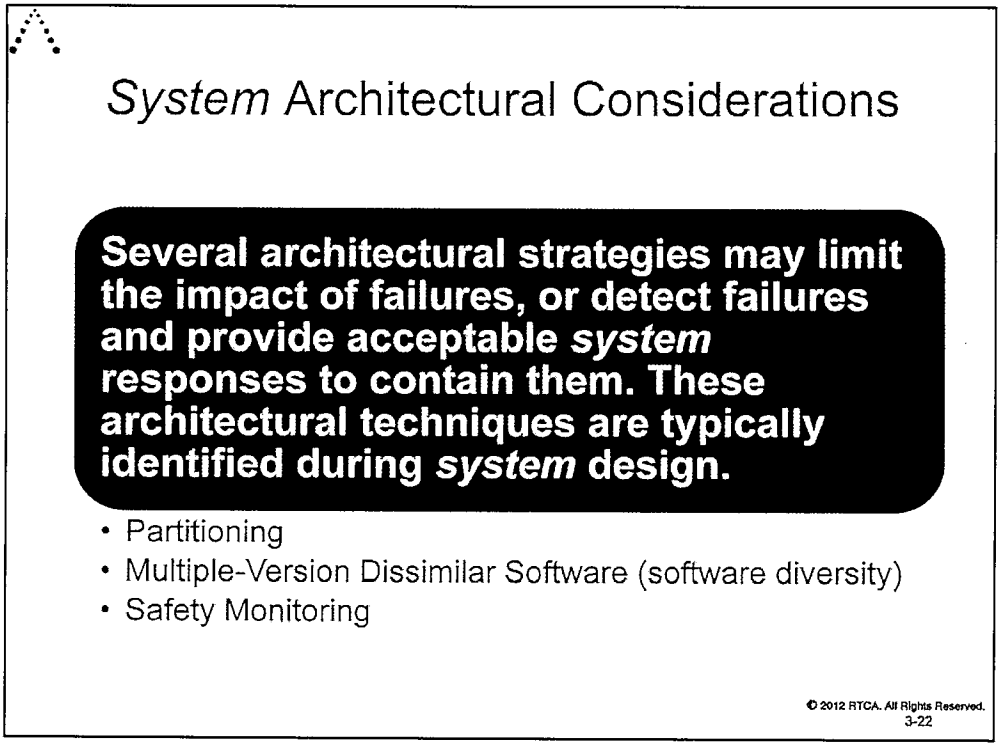
Section 2.3.2

For a complete definition of failure condition categories, the applicant should refer to applicable regulations and guidance material issued by the relevant certification authority. The failure condition categories listed in this slide are valid for large transport aircraft based on established advisory material for the *system* safety assessment process.

Multiple FAA Advisory Circulars can be referenced for additional information.



Level E: Software whose anomalous behavior, as shown by the *system* safety assessment process, would cause or contribute to a failure of *system* function with no effect on aircraft operational capability or pilot workload. **If a software component is determined to be Level E and this is confirmed by the certification authority, no further guidance contained in this document applies.**



These Architectural Considerations might be used to lower the Software Level for some or all of a *system's* software. The level would be specified by the *system* processes along with the architecture. Software design, especially the architecture part, may influence the Software Level but agreement for any change must be given by the *System's* processes.

Partitioning

- Partitioning is a technique for:
  - providing isolation between software components to contain and/or isolate faults
  - potentially reducing the software level
- Partitioning between software components may be achieved by allocating unique hardware resources to each component
- Alternatively, partitioning provisions may be made to allow multiple software components to run on the same hardware platform

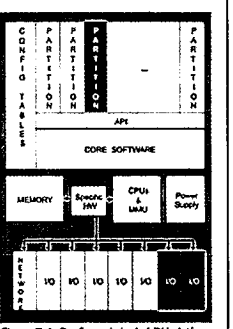


Figure D-1: Configured single LRU platform
Diagram Source: RTCA DO-297

Section 2.4.1

3-23

The intent of partitioning is to prevent a fault in one component from affecting other components. This may allow for a reduction in the software level. The illustration from DO-297 is meant to illustrate the partitioning of software only. The hardware aspects of the diagram are not considered here.

Partitioning Requirements

The following should be ensured for partitioned software components:

- "A partitioned software component should not be allowed to contaminate another partitioned software component's code, input/output (I/O), or data storage areas"
- "A partitioned software component should be allowed to consume shared processor resources only during its scheduled period of execution"
- "Failures of hardware unique to a partitioned software component should not cause adverse effects on other partitioned software components"
- "Any software providing partitioning should have the same or higher software level as the highest level assigned to any of the partitioned software components"
- "Any hardware providing partitioning should be assessed by the *system* safety assessment process to ensure that it does not adversely affect safety"

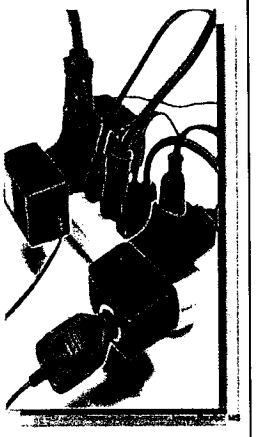
Section 2.4.1

3-24

The software life cycle processes should address the partitioning design considerations. These include the extent and scope of interactions permitted between the partitioned components and whether the protection is implemented by hardware or by a combination of hardware and software.

Multiple-Version Dissimilar Software

- "Multiple-version dissimilar software is a *system* design technique that involves producing two or more components of software that provide the same function in a way that may avoid some sources of common errors between the components"
- "The degree of dissimilarity, and hence the degree of protection, is not usually measurable"
- "Dissimilar software versions are usually used, therefore, as a means of providing additional protection after the software verification process objectives for the software level have been satisfied"



Section 2.4.2

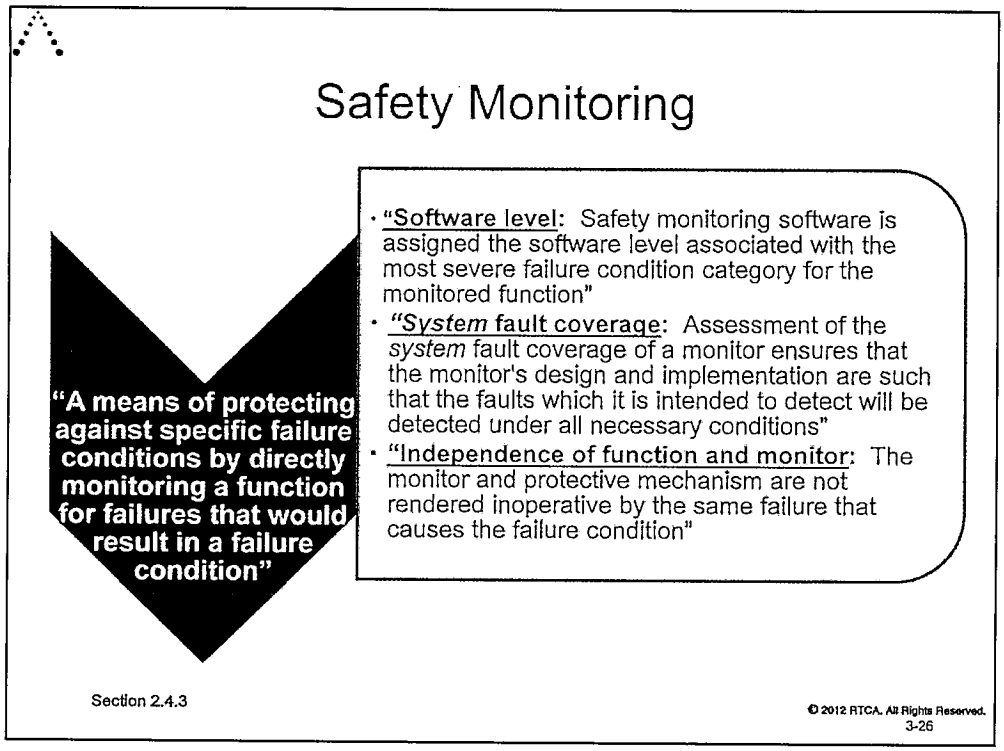
3-25

Multiple-version dissimilar software is also referred to as multi-version software, multi-version independent software, dissimilar software, N-version programming, or software diversity.

System requirements may specify a hardware configuration that provides for the execution of multiple-version dissimilar software.

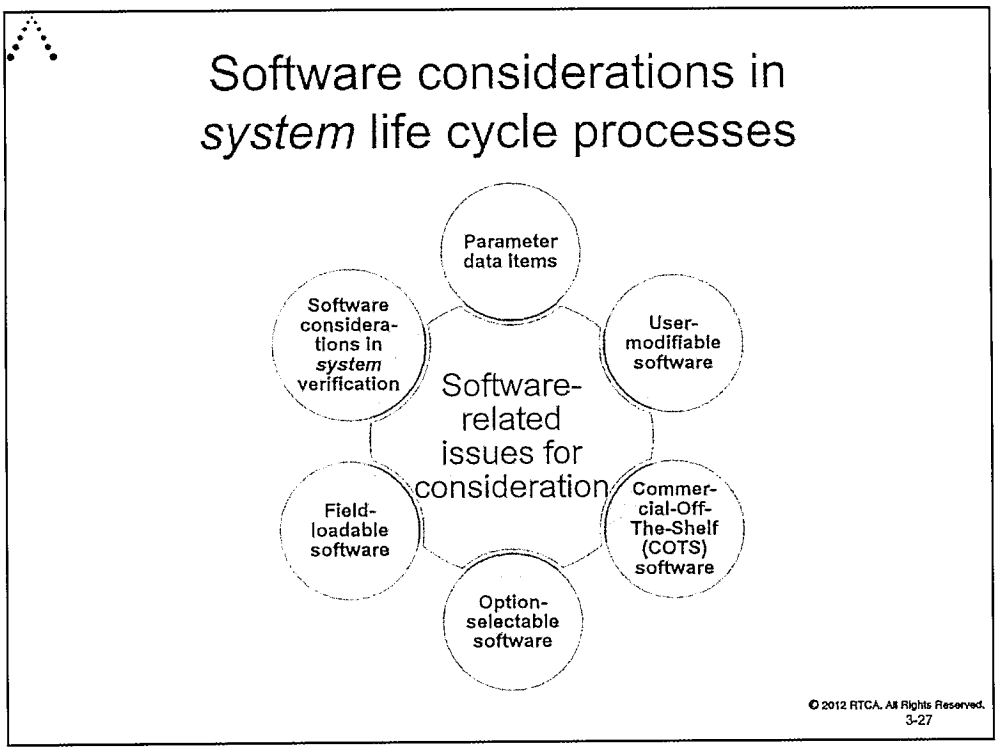
Dissimilar software verification methods may be reduced from those used to verify single version software if it can be shown that the resulting potential loss of *system* function is acceptable as determined by the *system* safety assessment process.

Multiple-version dissimilar software is not used much.



Monitoring functions may be implemented in hardware, software, or a combination of hardware and software.

Through the use of monitoring techniques, the software level of the monitored software may be assigned a software level associated with the loss of its related *system* function.

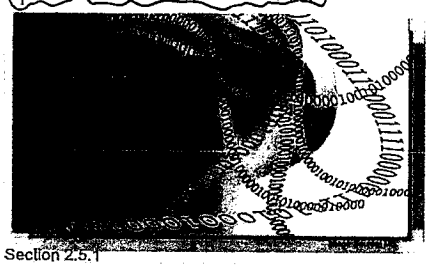


The next few slides provide an overview of those software-related issues, which are not necessarily mutually exclusive, and that should be considered, as appropriate, by the *system* life cycle processes.

New in DO-178C.

Parameter Data Items (1 of 3)

- "Software consists of Executable Object Code and/or data, and can comprise one or more configuration items"
- "A data set that influences the behavior of the software without modifying the Executable Object Code and is managed as a separate configuration item is called a parameter data item"



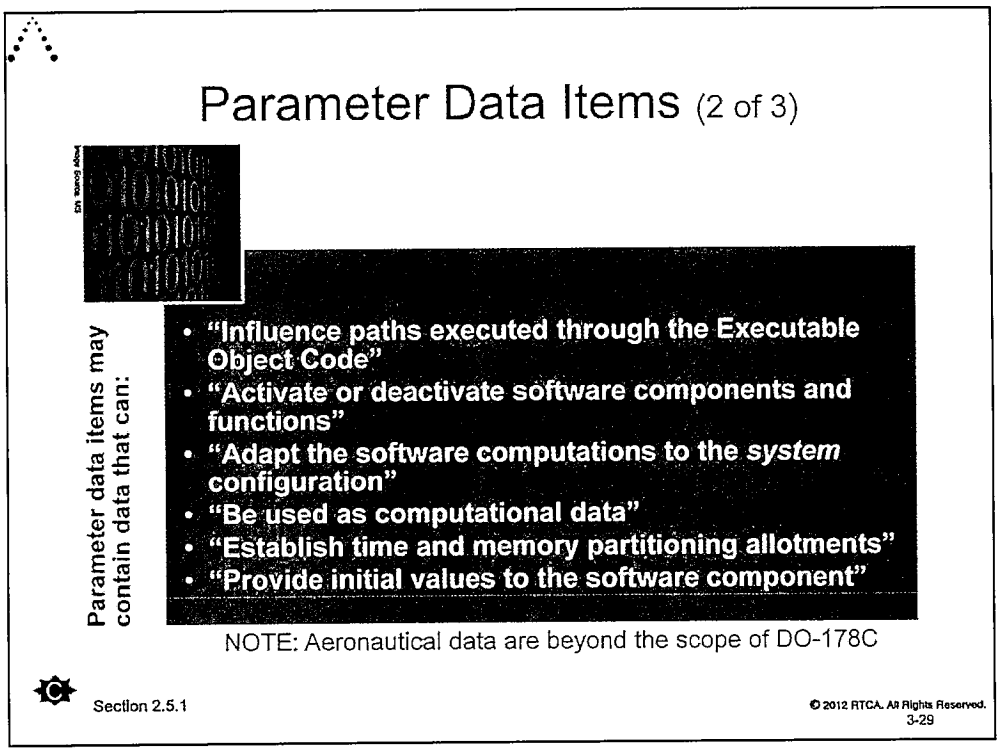
- It is implied that the parameter data items are not part of the Executable Object Code

Section 2.5.1

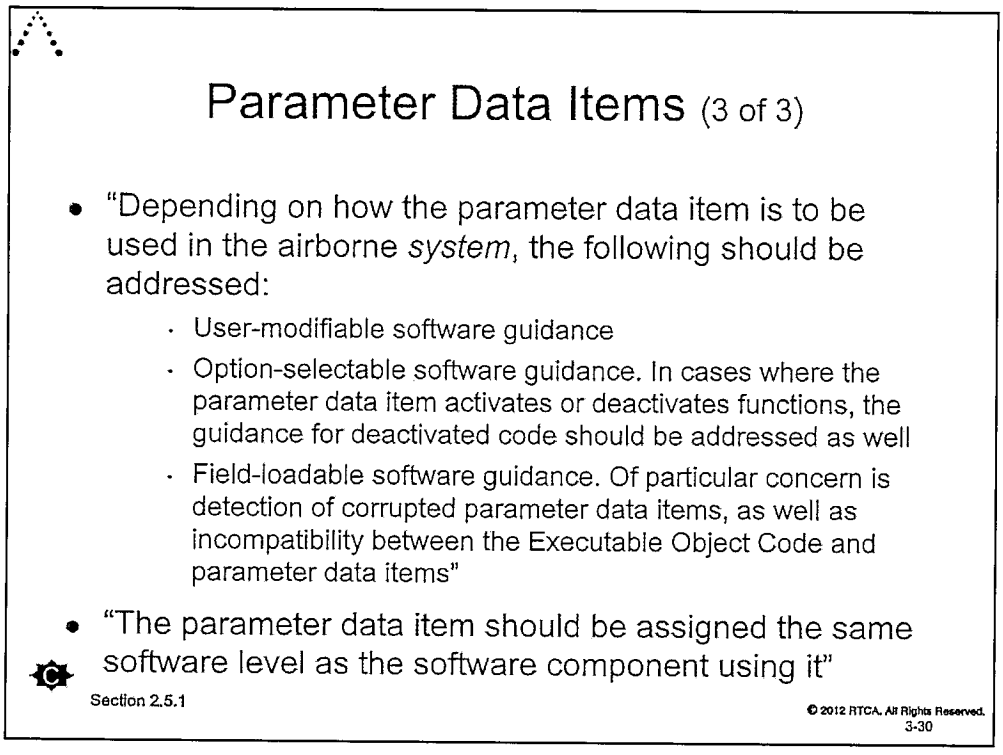
3-28

A parameter data item comprises a structure of individual elements where each element can be assigned a single value. Each element has attributes such as type, range, or set of allowed values.

An example is a personality module. A tail number is an example.



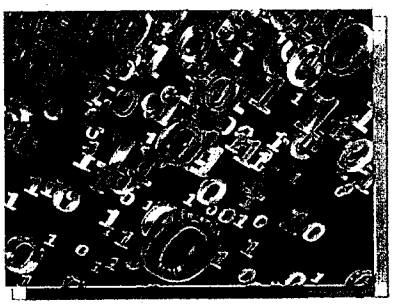
- Examples of parameter data items include configuration tables and databases but not aeronautical data because they are beyond the scope of DO-178C.
- An example of aeronautical data is the world wide navigation data base that must be updated every 28 days.



Verification of parameter data items is covered in Module 7.

It is difficult to come up with an example of user-modifiable software.

User-Modifiable Software (1 of 2)



"A user-modifiable component is that part of the software that may be changed by the user within the modification constraints without certification authority review"

"The potential effects of user modification are determined by the *system* safety assessment process and used to develop the software requirements, and then, the software verification process activities"

Section 2.5.2

3-31

This is software that is allowable to be changed by the user. The possible effects of changes have to be verified by the *system* safety assessment process.

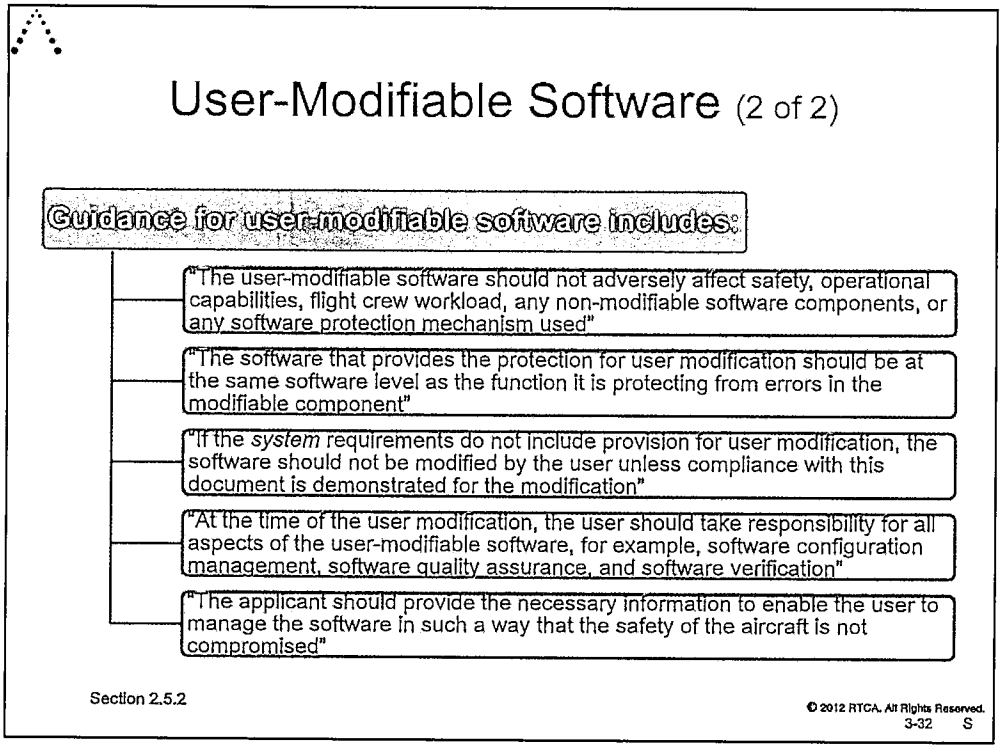
It is difficult to come up with many examples of user-modifiable software

ACARS is one such example.

Weight and balance will not be user-modifiable.

A non-modifiable component is that which is not intended to be changed by the user.

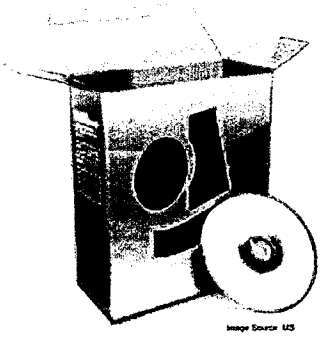
A change that affects the non-modifiable software, its protection, or the modifiable software boundaries is a software modification and is discussed in Module 12.



If the *system* requirements do not include provision for user modification, the software should not be modified by the user unless compliance with this document is demonstrated for the modification.

Commercial-Off-The-Shelf Software

"COTS software included in airborne *systems* or equipment should satisfy the objectives of this document"



"If deficiencies exist in the software life cycle data of COTS software, the data should be augmented to satisfy the objectives of this document"

Section 2.5.3

3-33

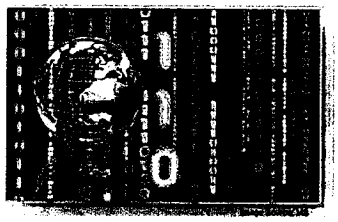
The guidance in section 12.1.4, Upgrading a Development Baseline, and section 12.3.4, (Alternate Method) Product Service History, may be relevant when needing to augment the life cycle data.

How can an i-pad be used in weight and balance? It is just replacing paper-and-pencil methods, which may also lead to many errors. In this case the i-pad does not transmit data to the aircraft. If you connect them up, then they have to be certified.

Example, portable ADS-B device. It does not transmit to the aircraft.

Option-Selectable Software

"Some airborne *systems* and equipment may include optional functions that may be selected by software-programmed options rather than by hardware connector pins"



"The option-selectable software functions are used to select a particular configuration within the target computer"

"When software programmed options are included, a means should be provided to ensure that inadvertent selections involving non-approved configurations for the target computer within the installation environment cannot be made"

Section 2.5.4

3-34

See DO-178C sections 4.2.h, 5.2.4, and 6.4.4.3.d for guidance on deactivated code.

Field-Loadable Software (1 of 2)

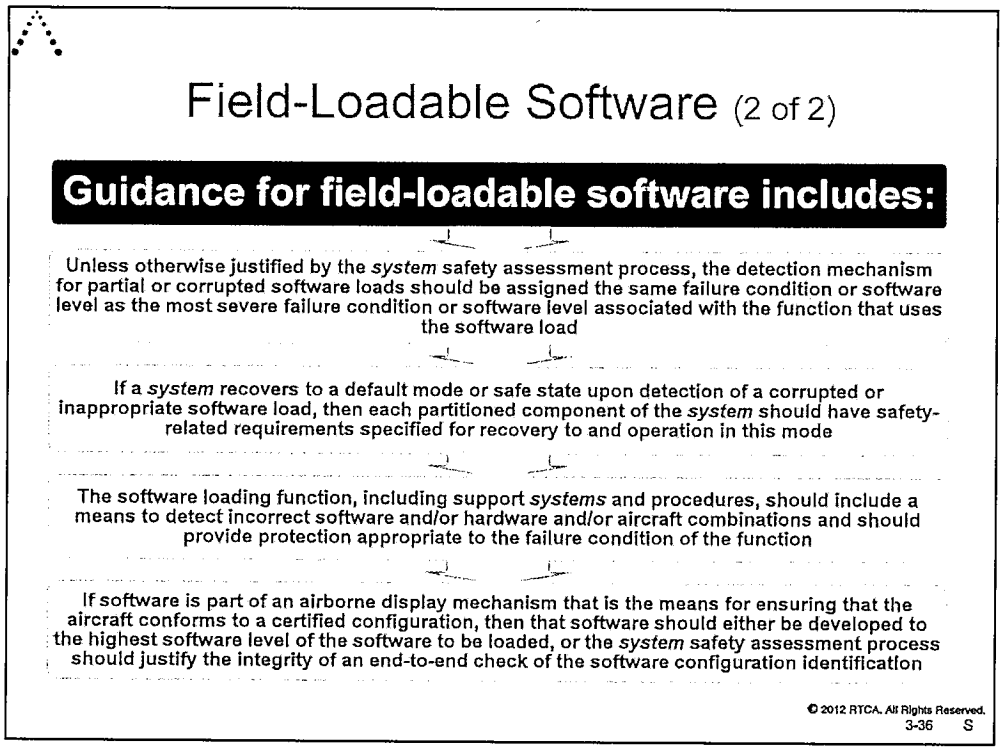
- "Field-loadable airborne software refers to software that can be loaded without removing the *system* or equipment from its installation"
- "The safety-related requirements associated with the software loading function are part of the *system* requirements"
- "*System* safety considerations relating to field-loadable software include:
  - Detection of corrupted or partially loaded software
  - Determination of the effects of loading the inappropriate software
  - Hardware/software compatibility
  - Software/software compatibility
  - Aircraft/software compatibility
  - Inadvertent enabling of the field-loading function
  - Loss or corruption of the software configuration identification display"

Section 2.5.5

3-35

If the inadvertent enabling of the software loading function could induce a *system* failure condition, then a safety-related requirement for the software loading function is specified in the *system* requirements.

FAA, DER's, etc. will conduct stringent surveillance of field-loadable software.



This is a common practice and occurs frequently. Almost any box can be upgraded with downloadable software on the ground.

Software considerations in *system* verification

Guidance for *system* verification is beyond the scope of DO-178C

"However, the software life cycle processes aid and interact with the *system* verification process and may be able to satisfy some *system* verification process objectives"

Image Source: MS

"Software design details that relate to the *system* functionality need to be made available to aid *system* verification"

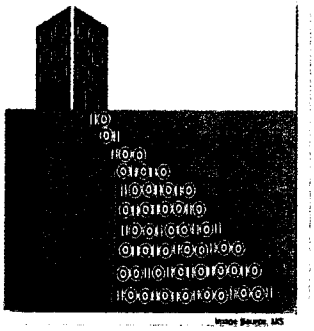
Section 2.5.6

System verification is out of scope.

Software life cycle processes may be able to assist in the *system* verification, but that must be determined by the *system*.

System considerations in software life cycle processes

Credit may be taken from *system* life cycle processes for the satisfaction, or partial satisfaction, of the software objectives as defined in DO-178C



"In such cases, the *system* activities for which credit is being sought should be shown to meet the applicable objectives of DO-178C with evidence of the completion of planned activities and their outputs identified as part of the software life cycle data"

Section 2.6

3-38

Review Questions

- What is the scope of the term "*system*" as it is used in DO-178C?
  - Refers to the airborne *system* and equipment ***only***
  - Does not include the wider definition of a *system* that might include Operators, operational personnel, etc.

Review Questions

- What *system* data flows to the software life cycle processes?
  - *System* Requirements Allocated to Software
  - *System* Safety Objectives
  - Software Level(s)
  - *System* Description and Hardware Definition
  - *System* Design Constraints
  - Definition of *System* Verification Activities to be Performed by Software Processes
  - Definition and Evidence of any Software Verification Activities Performed by *System* Processes
  - Evidence of Acceptability of Data

Review Questions

- What software data flows to the *system* life cycle processes?
  - Description of the Software Architecture
  - Evidence of any *System* Verification Activities Performed
  - Any Limitation of Use
  - Configuration Identification Data
  - Data to Facilitate Integration
  - Software Verification Activities to be Performed by *System* Processes
  - Derived High-Level and Low-Level Requirements
  - Problem or Change Documentation

Review Questions

- What data flows between the software life cycle processes and the hardware life cycle processes?
  - Hardware/Software Integration Requirements
  - Coordination and Feedback
  - Identified Hardware/Software Incompatibilities

Review Questions

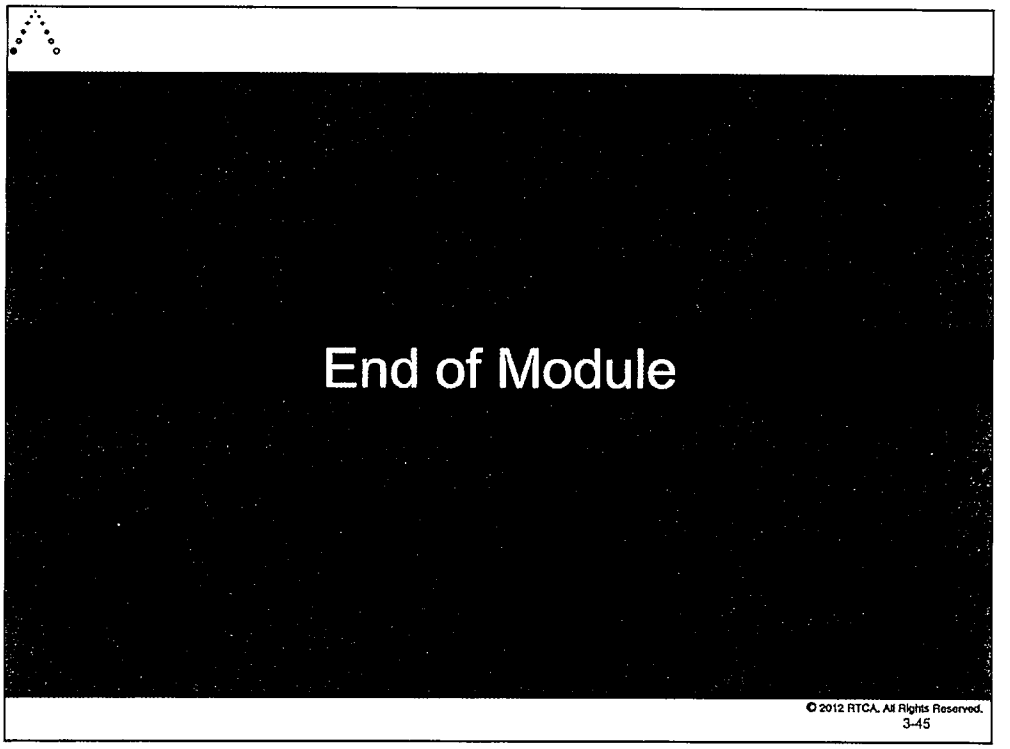
- Which Software Level is the least severe in terms of failure condition for an aircraft: A, B, C, D or E?

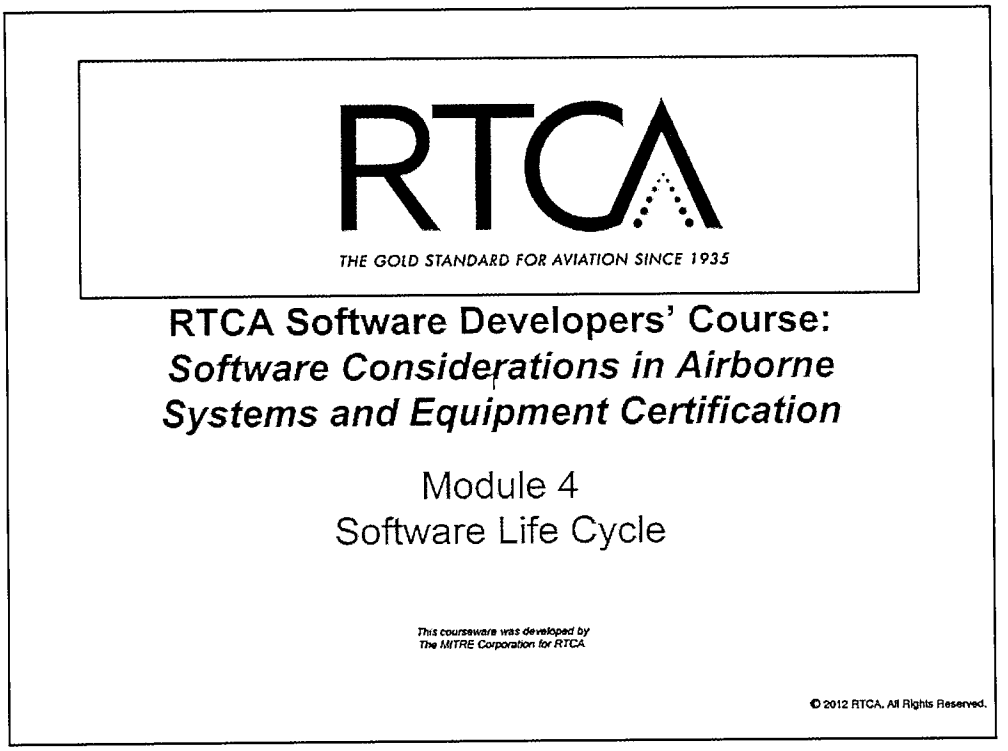
Level E

- Software would cause or contribute to a failure of *system* function with **no effect** on aircraft operational capability or pilot workload

Review Questions

- Name 3 of the *system* architectural influences on software:
  - Partitioning
  - Multiple-Version Dissimilar Software (software diversity)
  - Safety Monitoring

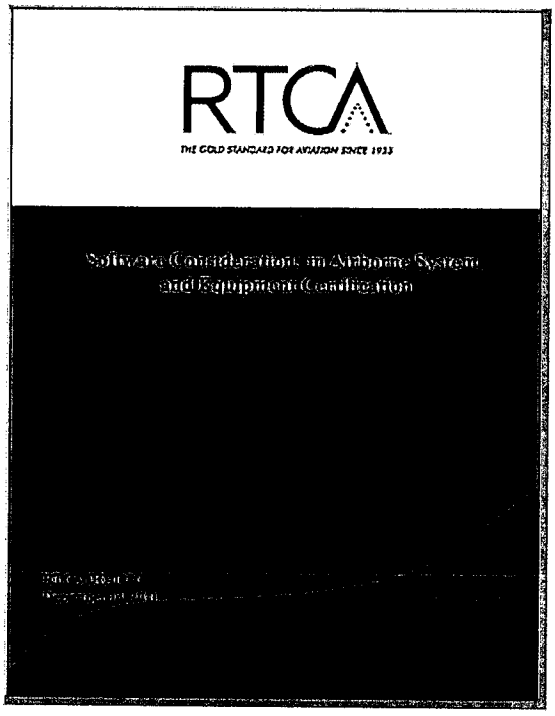




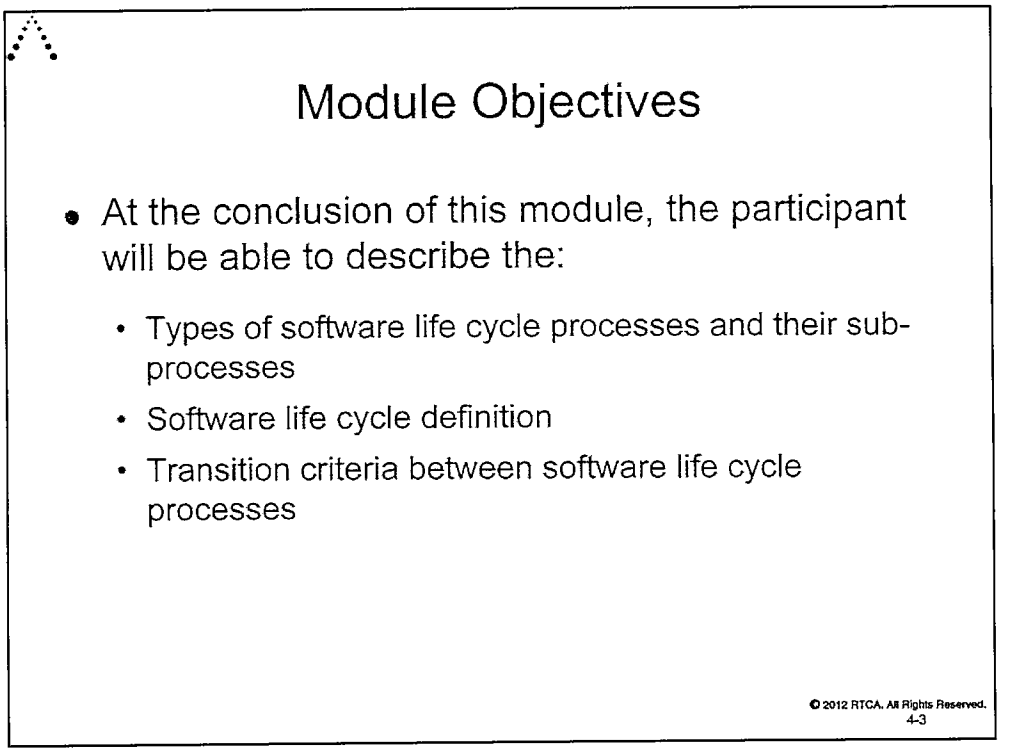
This 30-minute module introduces the Software Life Cycle processes, the Software Life Cycle definition and the transition criteria between processes. The transition criteria between processes determine whether a process may be entered depending upon the sequence of software development and integral processes.

DO-178C does not dictate a particular life cycle process. Choice of a life cycle process is left to the applicant in order to allow him/her to optimize his/her own life cycle processes.

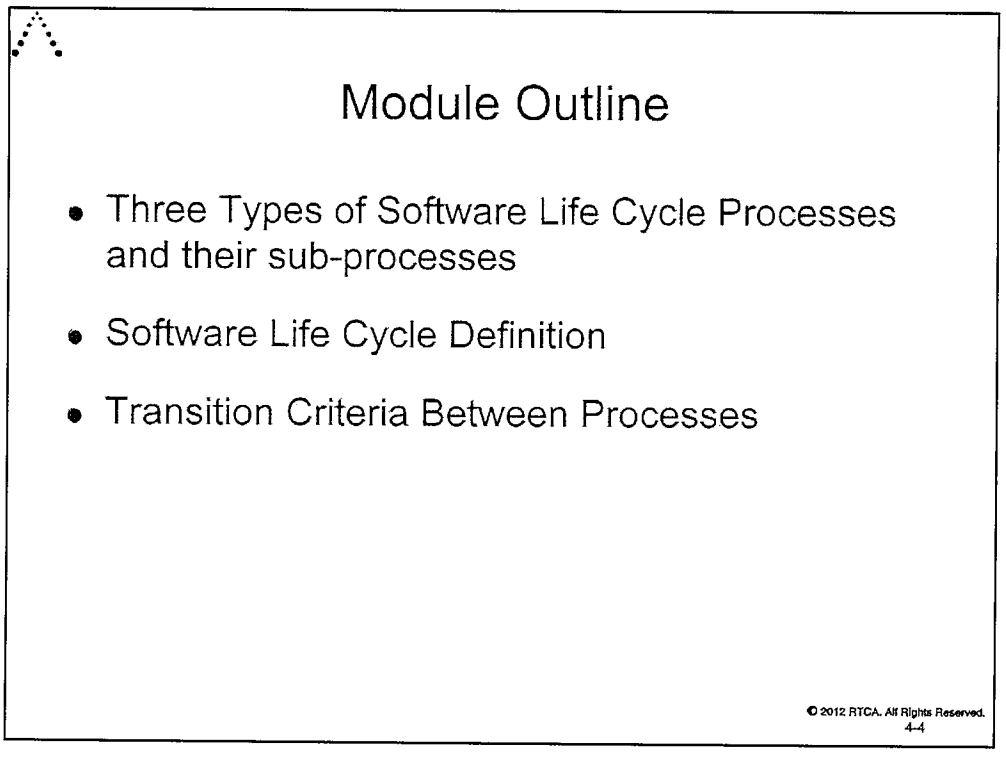
Your instructor does not know of a single change from DO-178B to DO-178C to this section.



Section 3



This chart provides the Module Objectives.



This chart is the Module Outline.

Philosophy
Choosing a Software Life Cycle

- Define a software life cycle to suite the type of development required
  - For example, if a prototype *system* is needed for simulation before fielding the final *system* then a rapid prototyping life cycle might be appropriate
- Define transition criteria

4-5

This slide explains the core philosophy of DO-178C – an applicant has a lot of freedom in selecting their own life cycle processes.

DO-178C does not define an order to the software life cycle processes, their sub-processes, and the activities within them. DO-178C does require the applicant to define the order by specific transition criteria between processes and/or sub-processes.

Software Life Cycle

- Software development is done within a software life cycle

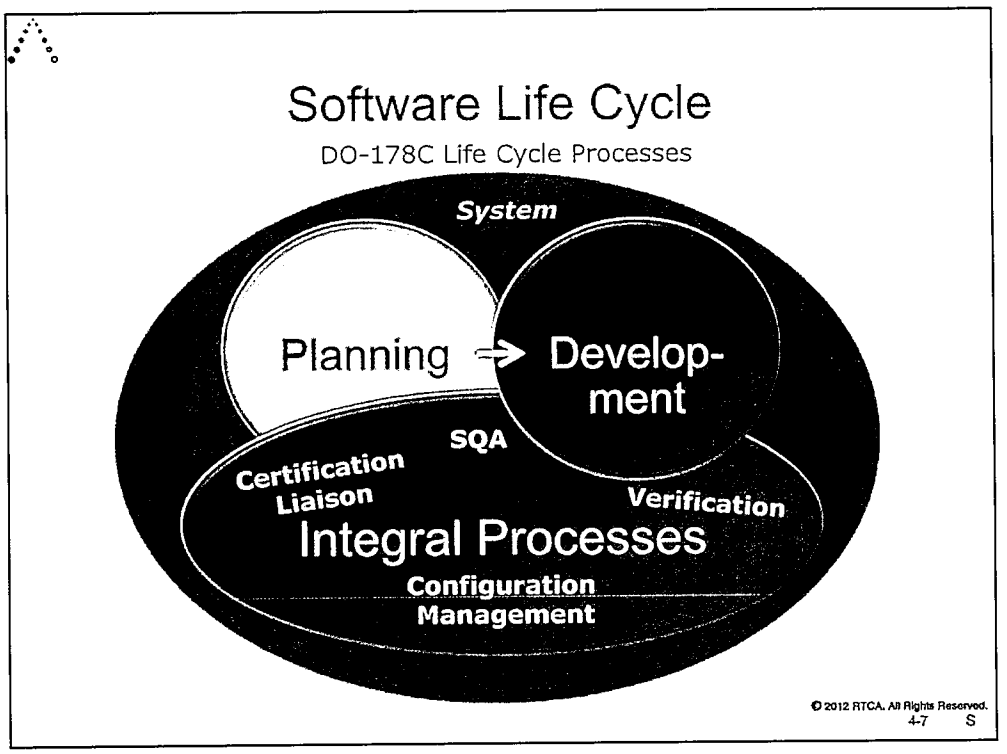
- A software life cycle is made up of planning, development and integral processes

4-6

DO-178C does not prescribe preferred software life cycles and interactions between them. The separation of the processes is not intended to imply a structure for the organization(s) that perform them. For each software product, the software life cycle(s) is constructed that includes these processes.

Integral is defined as pertaining to, or belonging as a part of the whole.

Integral process – A process which assists the software development processes and other integral processes and, therefore, remains active throughout the software life cycle. The integral processes are the software verification process, the software quality assurance process, the software configuration management process, and the certification liaison process.



There are many life cycle process charts. This particular slide depicts the life cycle processes in DO-178C that are used for software.

Note: the integral processes interact with the *system,* planning and development processes.

The *system* spans all three software life cycle processes.

The integral processes span the planning and development processes.

This chart is also following the same color-coded scheme in order for the reader to follow the processes all the way through the course.

Three Software Life Cycle Processes

1. Software **planning** process

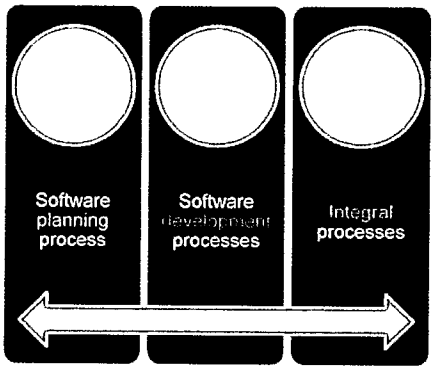
   defines and coordinates activities of software development and integral processes for a project

2. Software development processes

   Produce software product(s)

3. Integral processes

   Ensure correctness and control of, and confidence in software life cycle processes and their outputs



Integral processes are performed concurrently with software planning and development processes throughout software life cycle

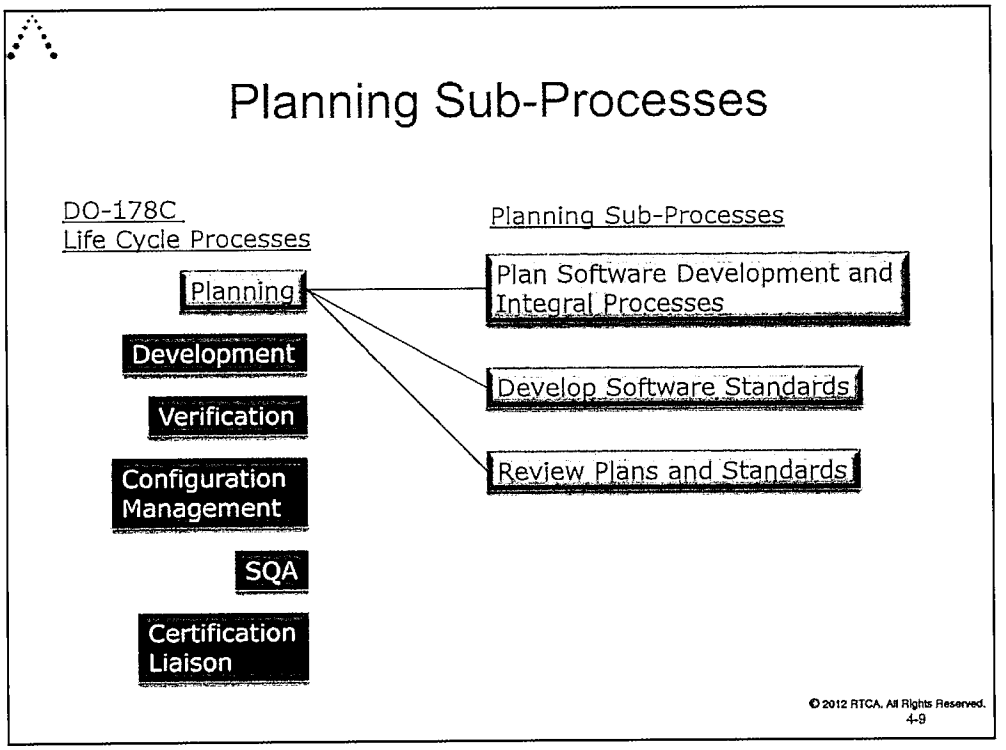
Section 3.1

4-8

The software development processes are the software requirements process, the software design process, the software coding process, and the integration process.

The integral processes are the software verification process, the software configuration management process, the software quality assurance process, and the certification liaison process.

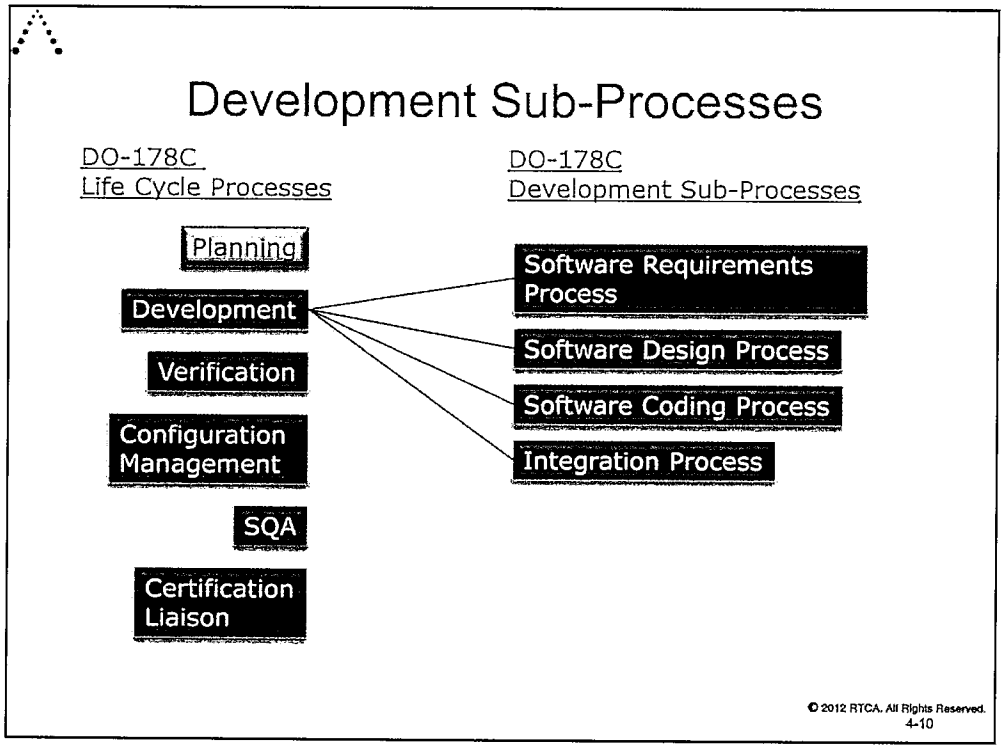
Remember the three Cs: Correctness, Control and Confidence



Three sub-processes in the software planning process are:

- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

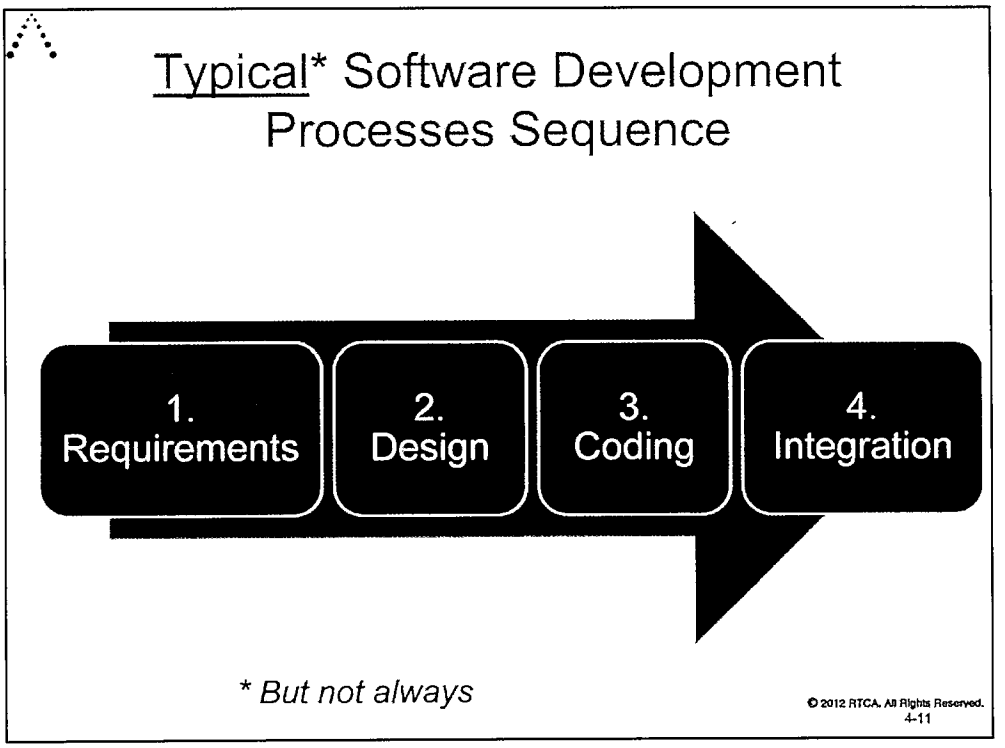
Plans, when followed, need to produce products compliant to DO-178C objectives.



The DO-178C Software Development Process has 4 Sub-Processes:

- Software requirements sub-process
- Software design sub-process
- Software coding sub-process
- Software integration sub-process

DO-178C does not impose a timing or ordering constraint on these sub-processes. Any timing or ordering constraints are imposed by the transition criteria defined in the Software Development Plan. Do what makes sense within your development processes. Processes may be done in parallel or sequentially.



This is a typical sequence, called a waterfall, but it is not required. We will see other sequence examples.

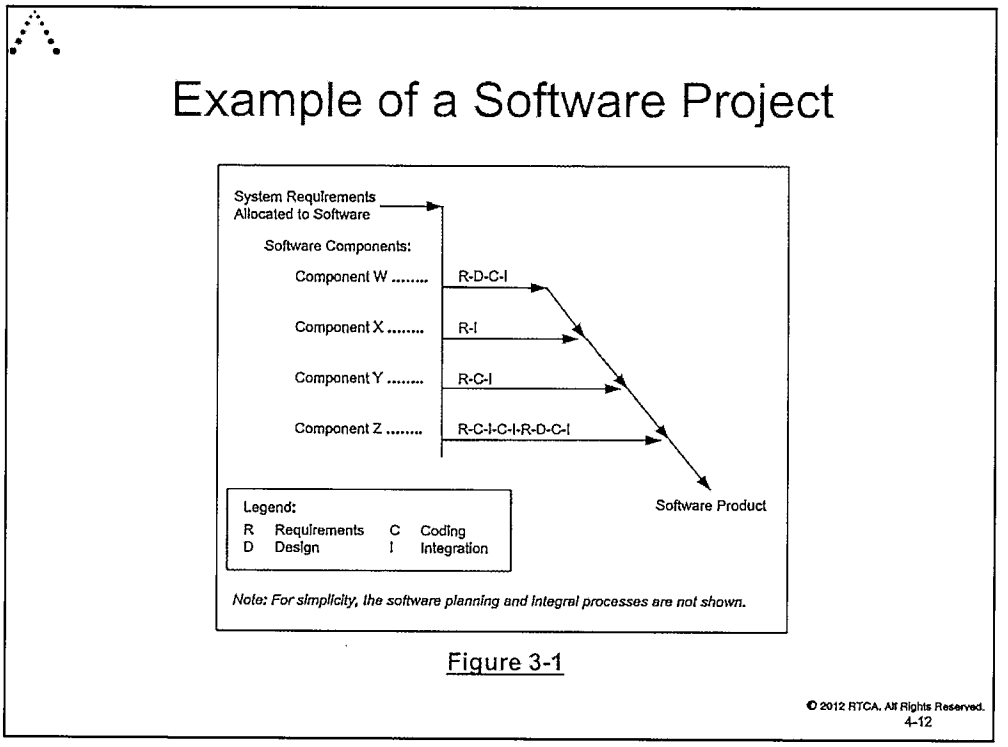
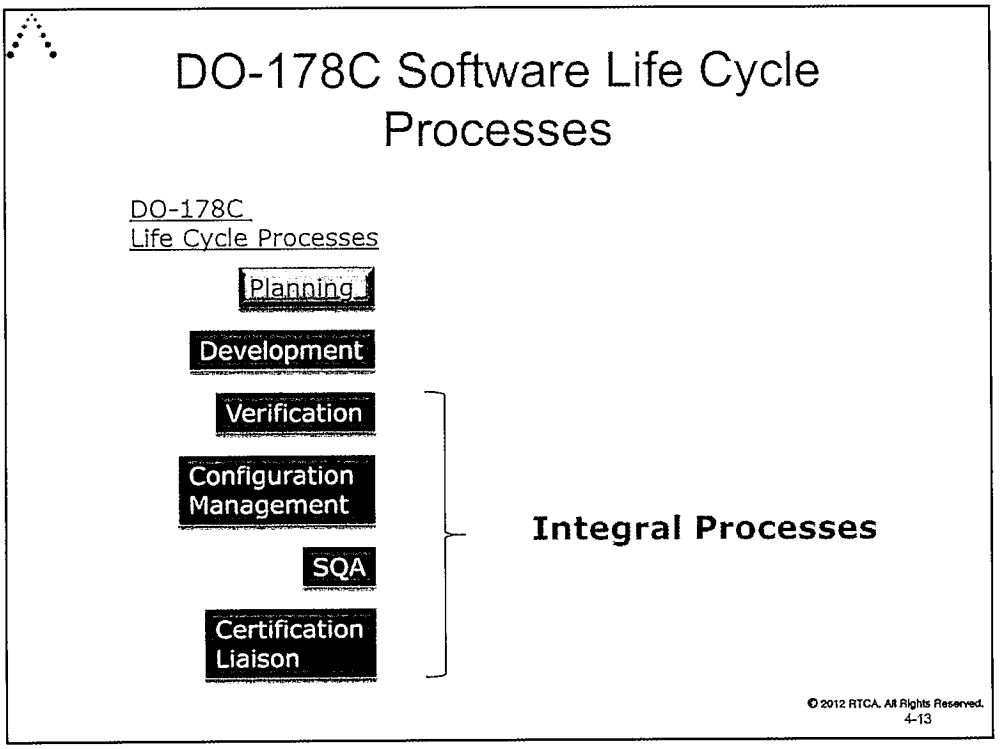


Figure 3-1 illustrates the sequence of software development processes for several components of a single software product with different software life cycles.

- Component W implements a set of *system* requirements by developing the software requirements, using those requirements to define a software design, implementing that design into Source Code, and then integrating the component into the hardware.
- Component X illustrates the use of previously developed software used in a certified product.
- Component Y illustrates the use of a simple, partitioned function that can be coded directly from the software requirements.
- Component Z illustrates the use of a prototyping strategy. Usually, the goals of prototyping are to better understand the software requirements and to mitigate development and technical risks. The initial requirements are used as the basis to implement a prototype. This prototype is evaluated in an environment representative of the intended use of the *system* under development. Results of the evaluation are used to refine the requirements.

The structure of DO-178C may be appear to be a waterfall model, but it is not intended to be a waterfall model.



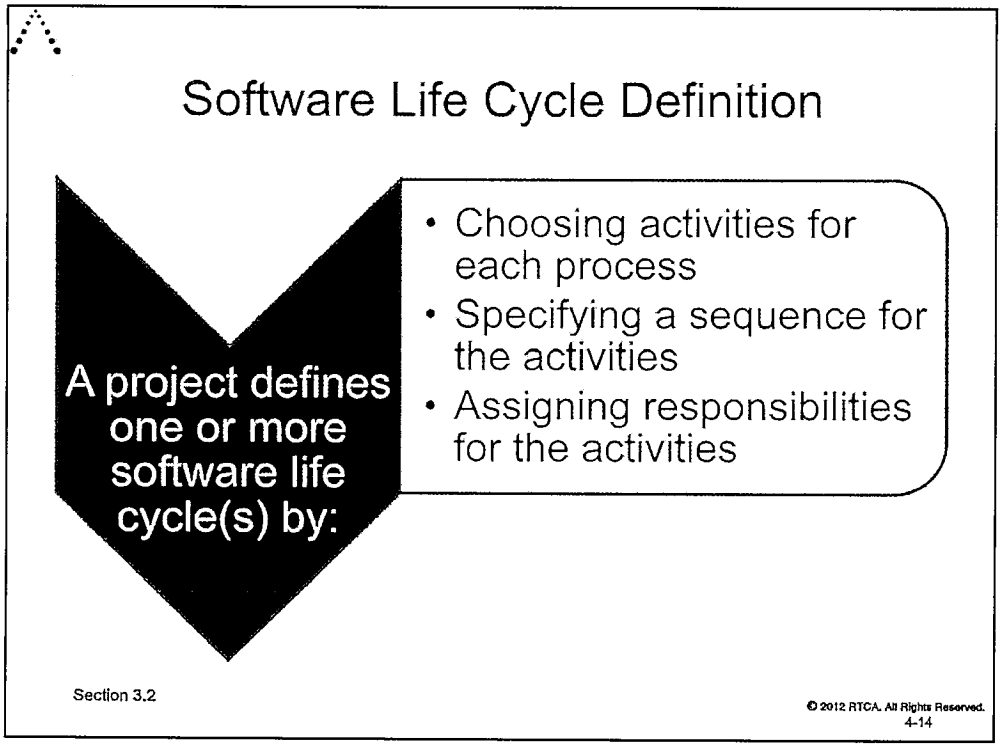
The 3 software life cycle processes.

Planning is made up of three sub-processes, not shown here.

Four sub-processes (not shown) make up the development process.

Four processes make up the integral processes.

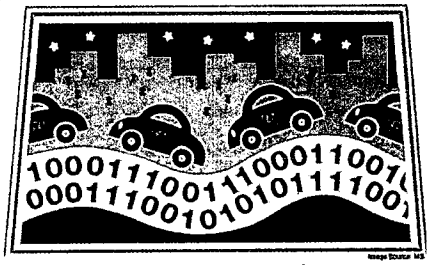
Each of these integral sub-processes will be covered in detail in later modules. The intent here is to show their relationship in the life cycle.



Choosing sub-processes and activities for each process, specifying a sequence for the activities, and assigning responsibilities for the activities is how a project defines one or more software life cycles.

Sequencing of Processes

- Determined by project's attributes, such as:
  - *System* functionality and complexity
  - Software size and complexity
  - Requirements stability
  - Use of previously developed software
  - Development strategies
  - Hardware availability



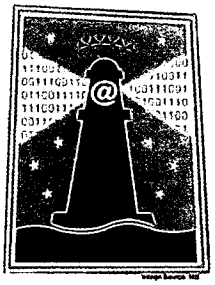
Section 3.2

4-15

Processes are sequenced by determining a project's attributes.

Notes on Sequencing

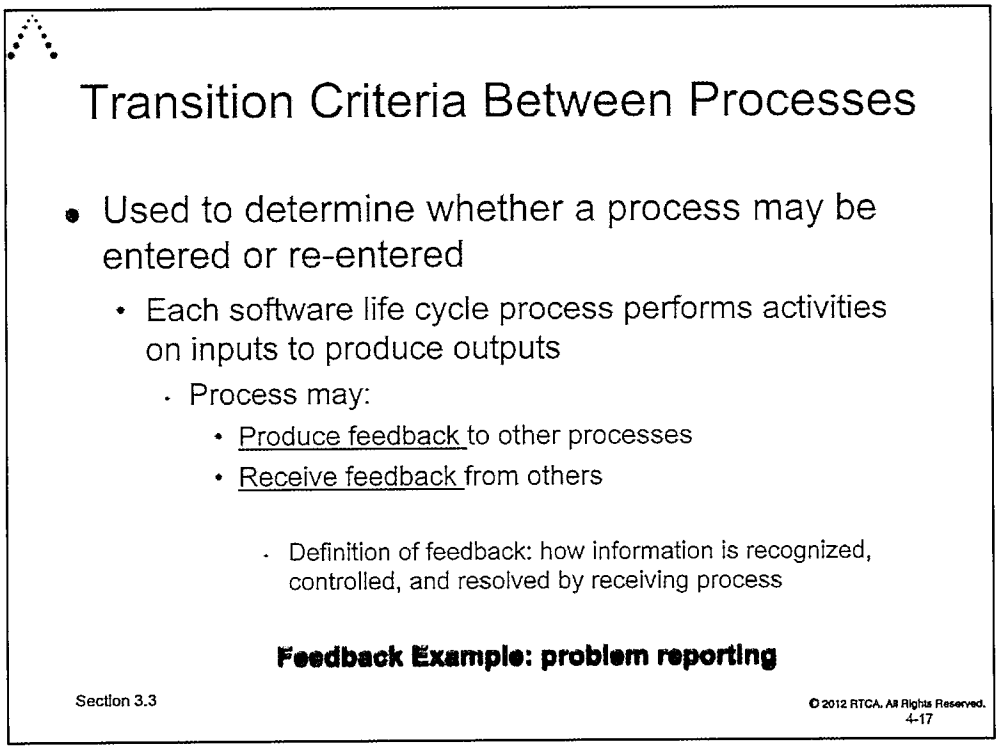
- Processes of software life cycle may be iterative
  - Entered and re-entered
  - Timing and degree of iteration varies due to
    - Incremental *system* functions development
    - Complexity
    - Requirements development
    - Hardware availability
    - Feedback to previous processes
    - Other project attributes
- Various parts of selected software life cycle are tied together with a combination of
  - Incremental integration process
  - Software verification process activities



Section 3.2

4-16

The key idea on this slide is the iterative process that may be used in sequencing.



The transition criteria will depend on the planned sequence of software development processes and integral processes, and may be affected by the software level.

Transition Criteria

Examples of transition criteria:

- Software verification process reviews have been performed
- The input is an identified configuration item
- Traceability analysis has been completed for the input

"Not <u>every</u> input to a process need be complete before that process can be initiated, if the transition criteria established for the process are satisfied"

Section 3.3

4-18

If a process acts on partial inputs, the inputs to the process should be examined to ensure that they meet transition criteria. Also, subsequent inputs to the process should be examined to determine that the previous outputs of the software development and software verification processes are still valid.

Review Questions

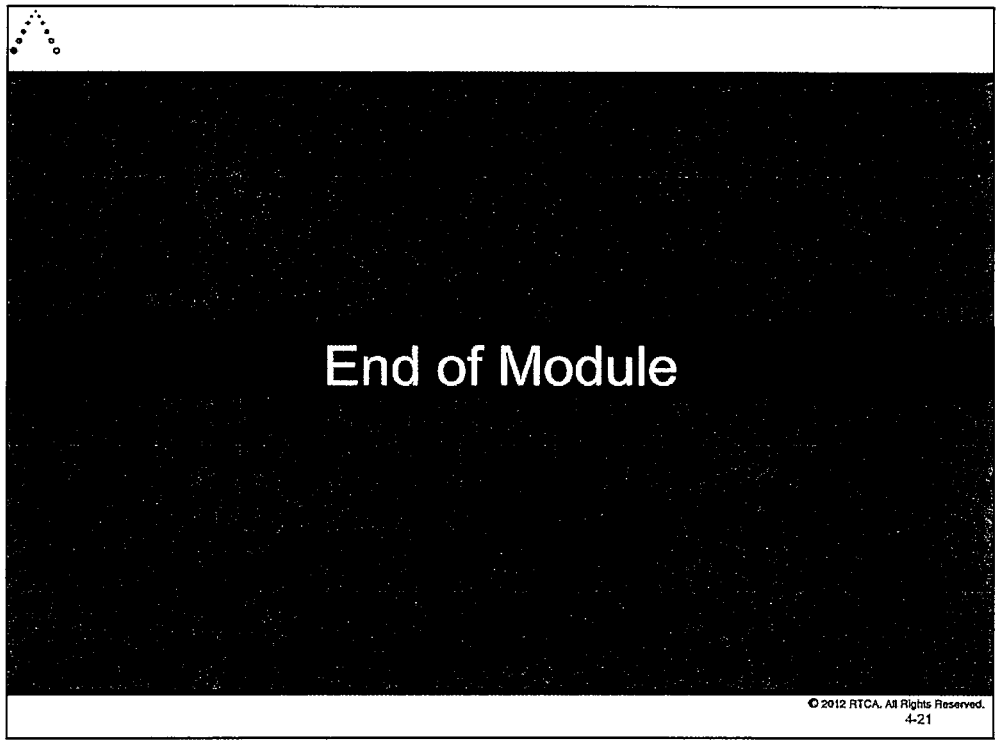
- Which software process "Defines and coordinates the activities of the software development and integral processes for a project"?

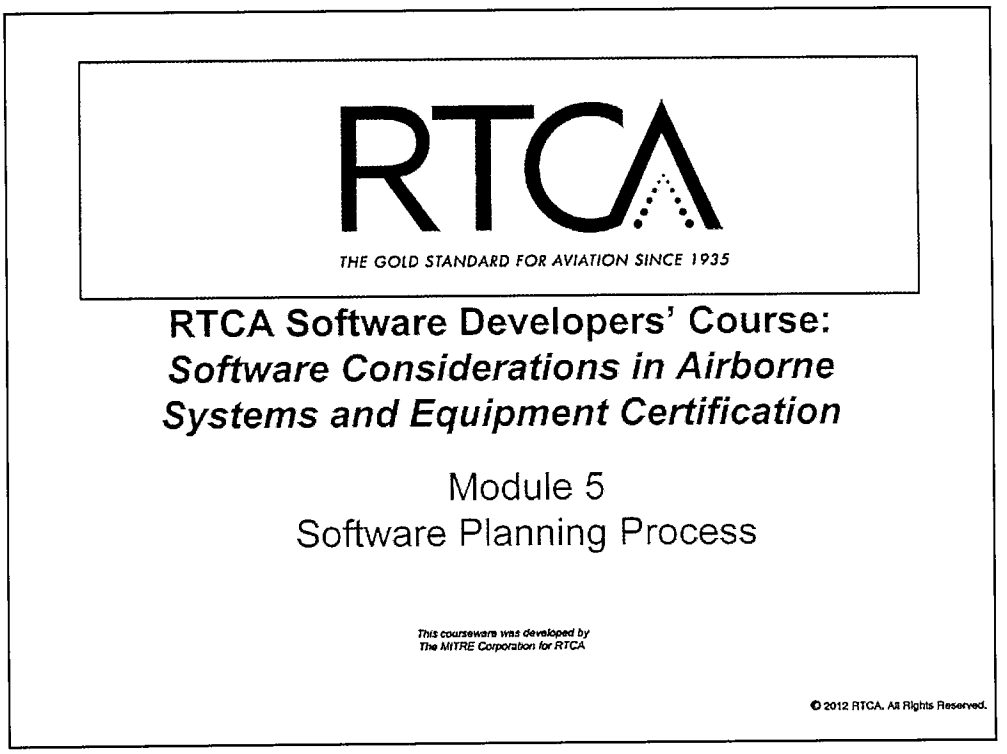
Software Planning Process

Review Questions

- What planning data item controls the order and timing of the Software Life Cycle processes and their activities?

Transition Criteria


End of Module
© 2012 RTCA. All Rights Reserved.
4-21



This 1-hour, module covers planning of software development and integral processes necessary to completely and correctly implement *system* requirements which strongly contributes to *system* safety. The module sets the context for the rest of the course. The objectives and products of the Software Planning Process are introduced. These process objectives are:

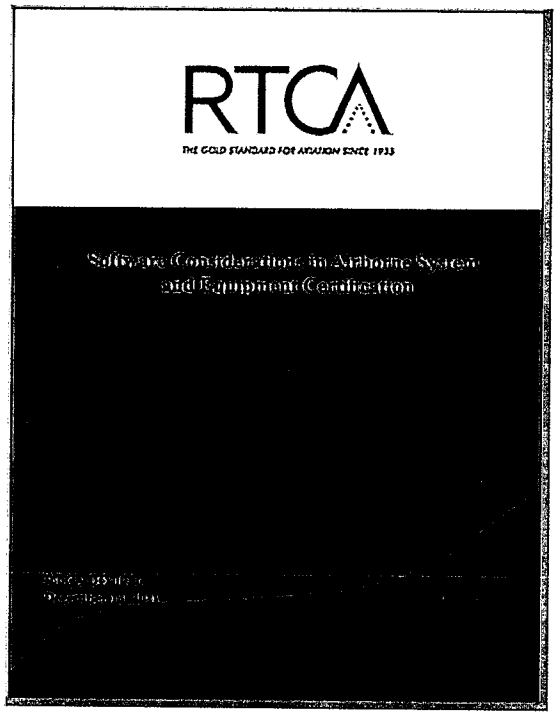
Describe software development and integral software life cycle processes needed to address *system* requirements and software levels

Define software life cycle environment including methods and tools to be used for each activity and testing of integrated process outputs

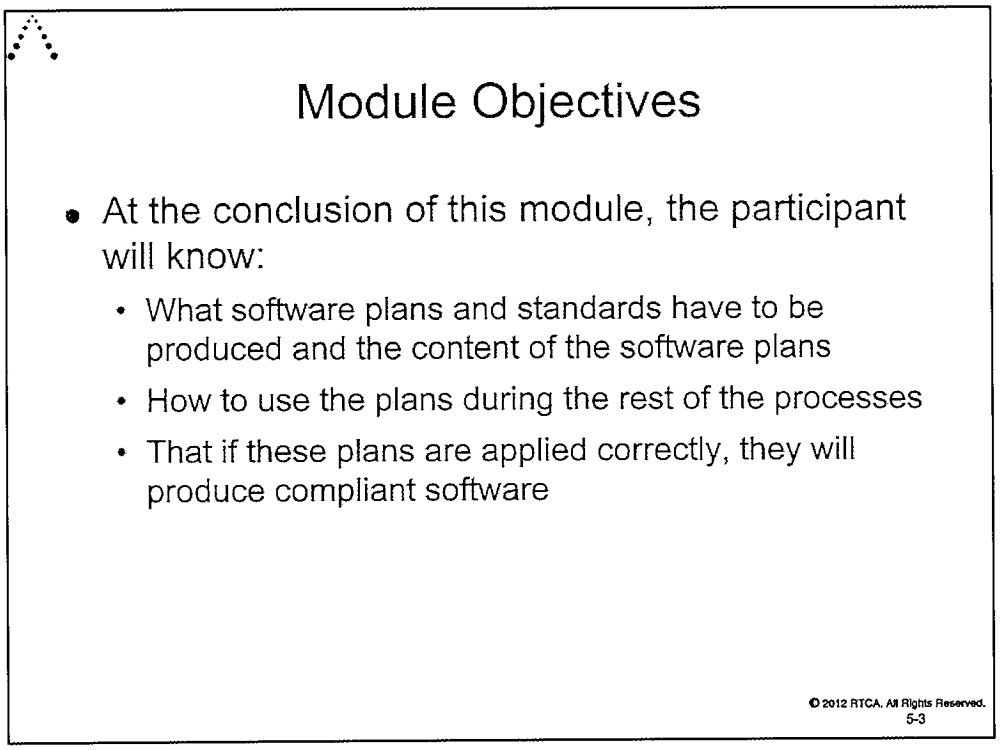
Enable software development standards consistent with *system* safety objectives, software requirements, design and code standards for software verification process and compliance.

Next, the products of the planning process (Software Development Plan, Software Verification Plan, Software Configuration Plan, Software QA Plan) and the standards required by DO-178C (Requirements, Design, and Code) are described. The Software Life Cycle Environment will cover Development Tools including Programming Language, and Verification Tools.

An exercise designed to review the Software Planning Process teaches the participants how to apply knowledge acquired in this module in a case study.

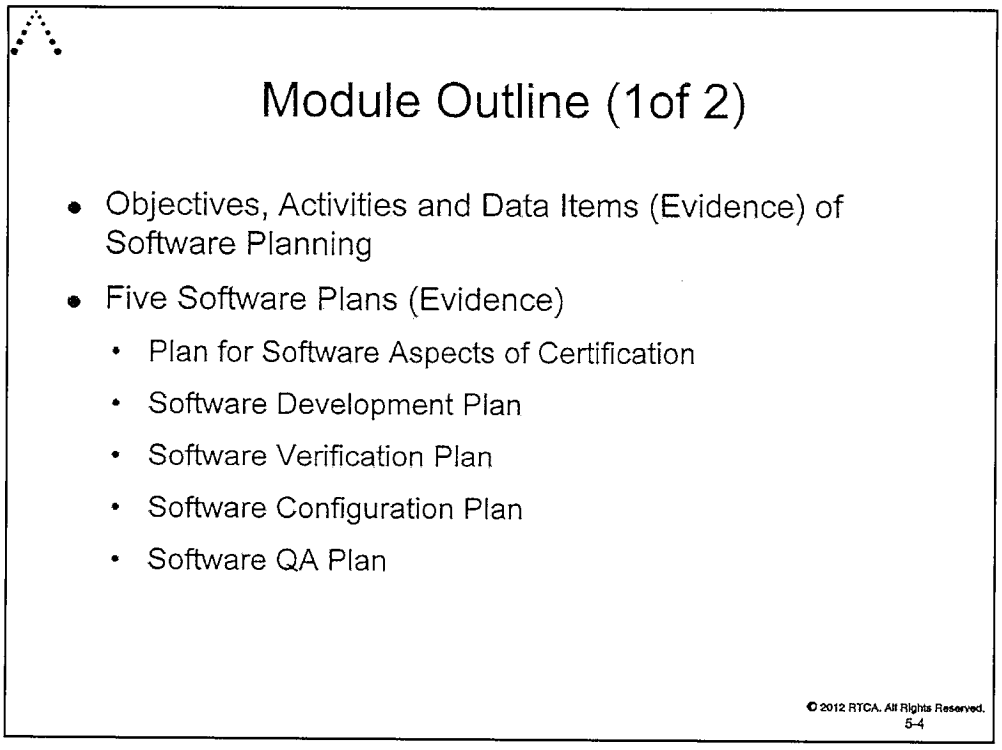


Section 4

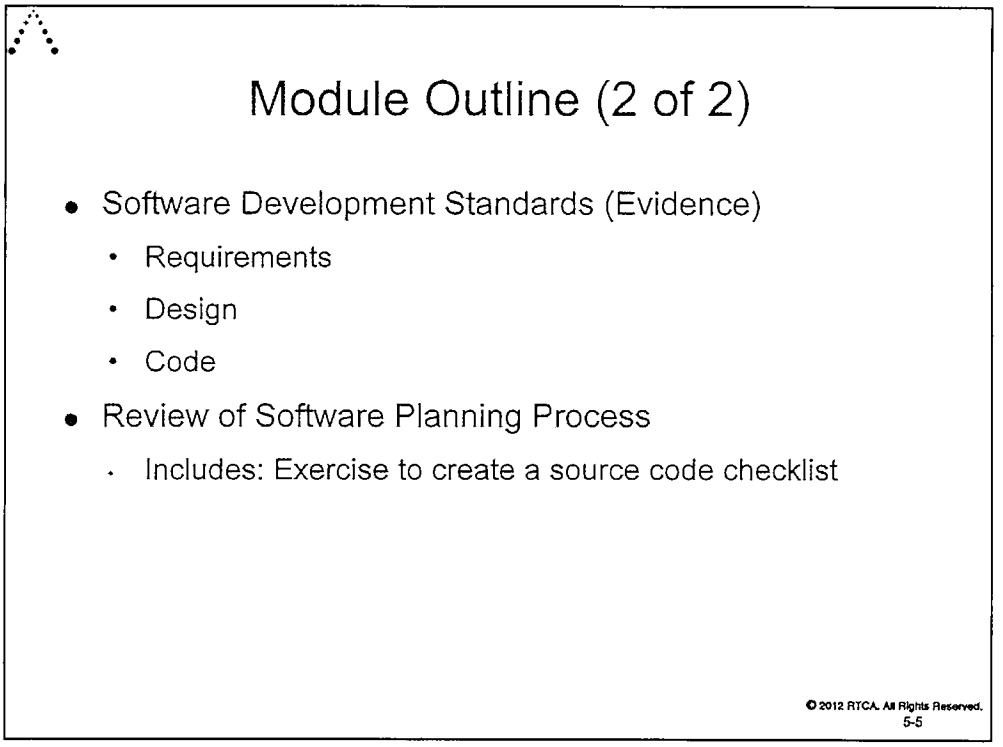


The plans describe the processes and the activities within a process that if followed correctly will develop, verify, control, and assure that the airborne *system* and equipment being developed will achieve certification. The plans must be followed or modified if the processes and activities are not working.

The plans must be correct.

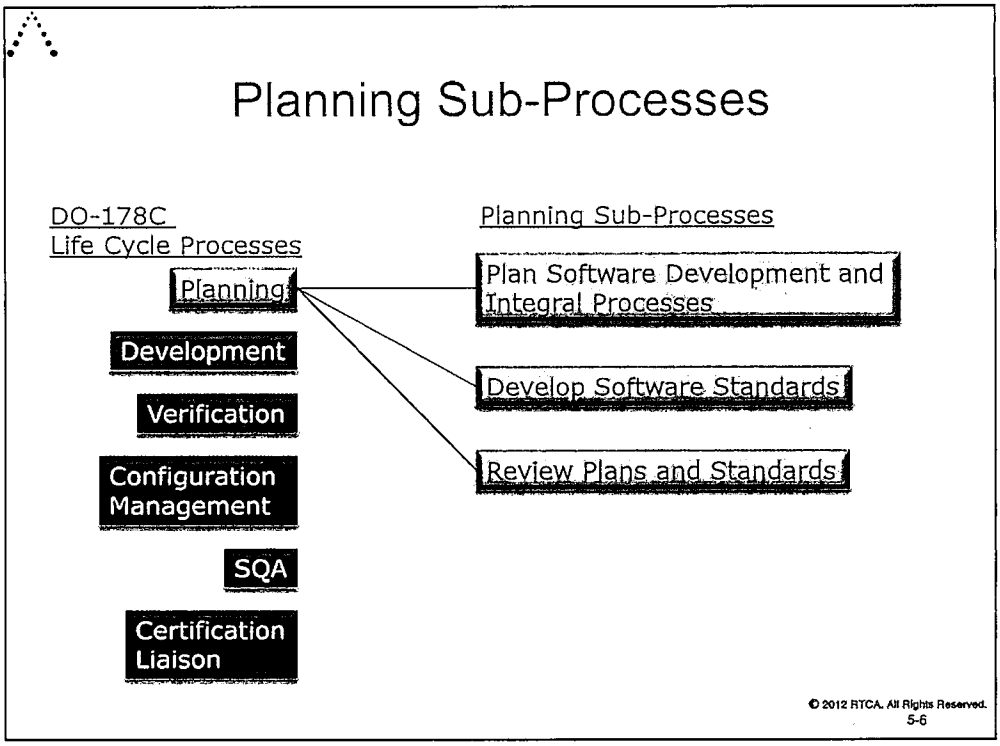


The software planning process along with the 5 resulting plans will be covered.



The sub process that develops the three software standards will be covered.

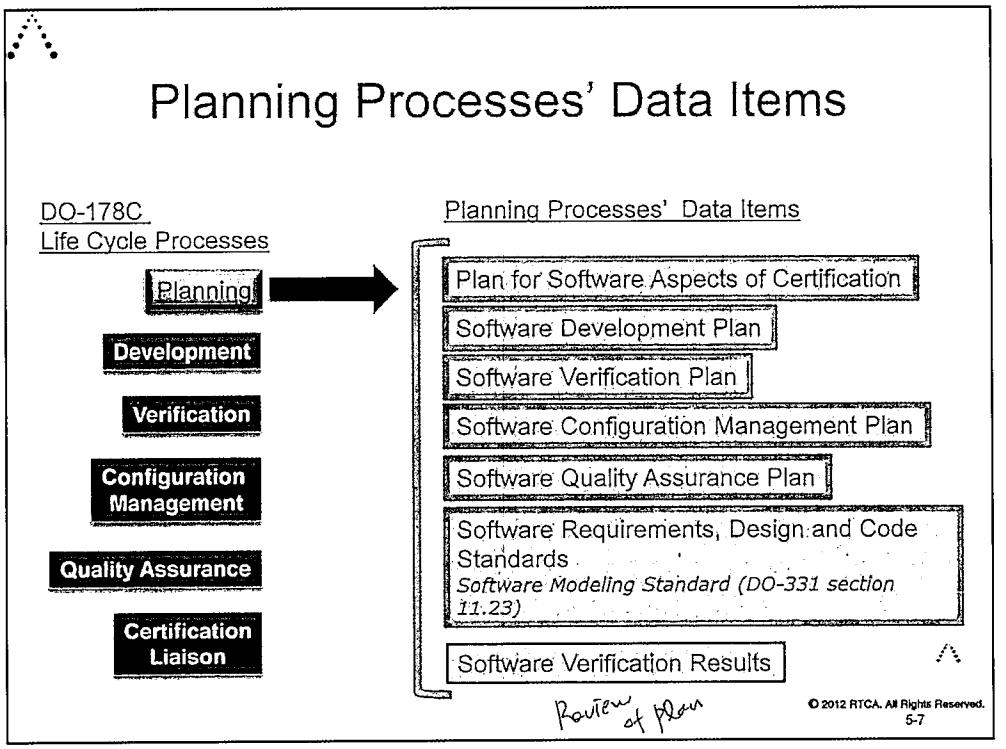
And finally the very important process that reviews the plans and standards to verify that they if followed certifiable airborne *system* and equipment software will be developed.



Three sub-processes in the software planning process are:

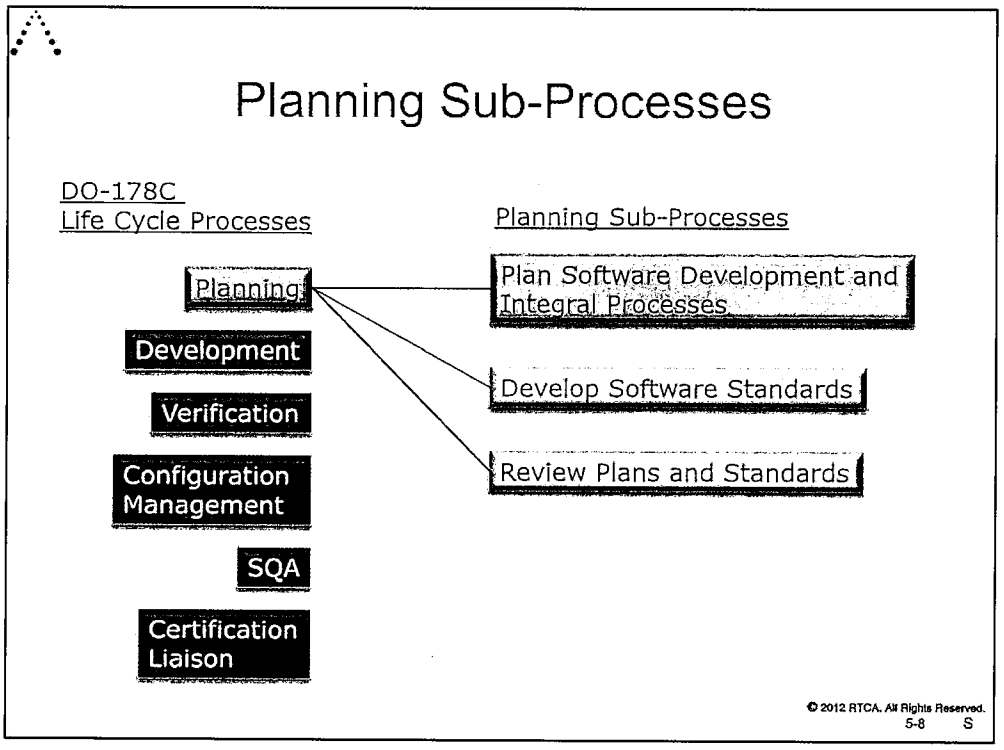
- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Plans, when followed, need to produce products compliant to DO-178C objectives.



This slide reviews Modules 5 (Planning process) Software Life Cycle Data.

Note that verification results are output of an integral process call verification

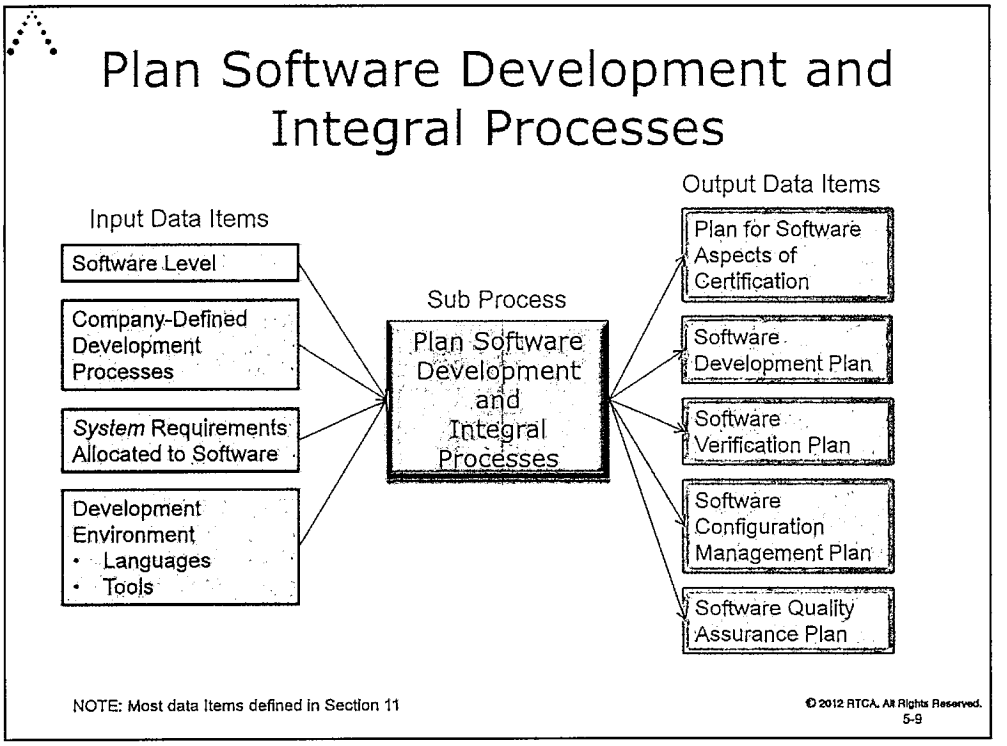


Three sub-processes in the software planning process are:

- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Discuss "activities": Use backup slide defining "activities"

Plans, when followed, need to produce products compliant to DO-178C objectives.



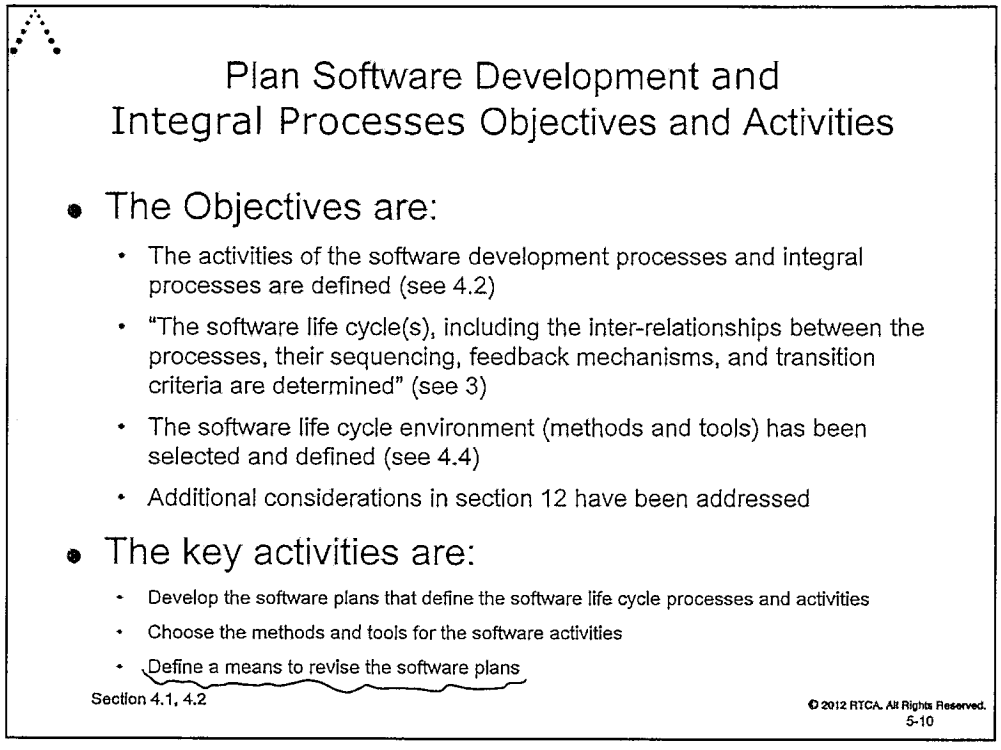
Inputs to the planning process include:

- Software Level – This is a ***must have*** because it affects which processes and activities are required to build a certifiable product. Additionally it is needed to complete the review of the plans.
- Company-Defined Development Processes – If your company has plans, they can be used. Additions to them may be needed. They can be referenced in part or in whole from a program specific plan. The program-specific plan may tailor the company plans depending on assurance level or specific needs of the program.
- *System* Requirements – It may or may not be important to know what is being developed. It will be important to know if there are additional considerations like partitioning, user-modifiable, ...
- Development Environment – The plans will need know this especially if there are different/special tools not normally used in a company's software processes.

The five plans ' output data items are: (they will be discussed individually in other slides)

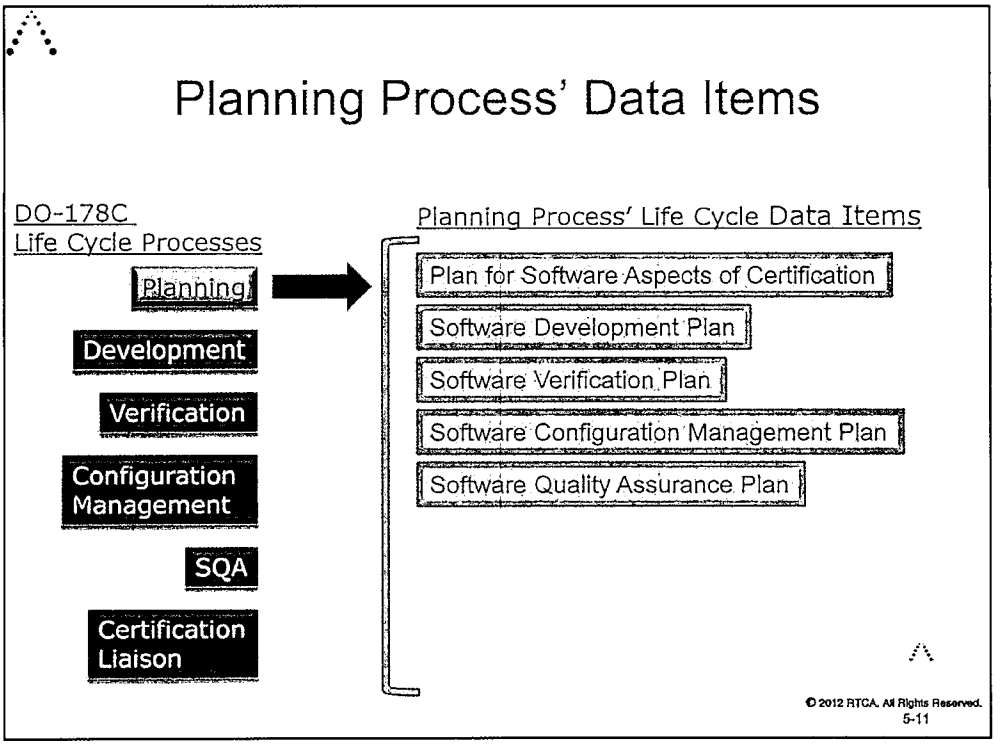
- PSAC, SDP, SVP, SCMP, SQAP

There are no objectives on this chart. This is a typical input output chart.



- The first objective implies existence of planned activities for each process/sub-process/sub-sub--process... The activities may include the activities described in DO-178C. There may be different activities from those described in DO-178C. There may be additional activities to those described in DO-178C. **The correct choice of processes and their activities is a "determining factor in producing software that satisfies the guidance of this document".**
- Second objective implies the existence of transition criteria for all process/sub-process/sub-sub--process... Transition Criteria may be needed for some tasks and should support the processes not hinder them. Do not forget SCM and SQA when defining transition criteria. Other software life cycle processes may begin before completion of the software planning process if transition criteria for the specific process activity are satisfied (for example: Configuration Management of the plans).
    - Every one of these things has to be defined, i.e. static analyzers.
- Third objective implies that the tools and computers used during the software life cycle are listed. This is especially important since some tools may need qualification and use of an unqualified tool will negate the results from that tool. Also consider calibration of any hardware tools.
    - Go to Section 12 partitioning
- If any additional considerations (Ex: re-usable software) from section 12 are to be used they must be included in the plans.

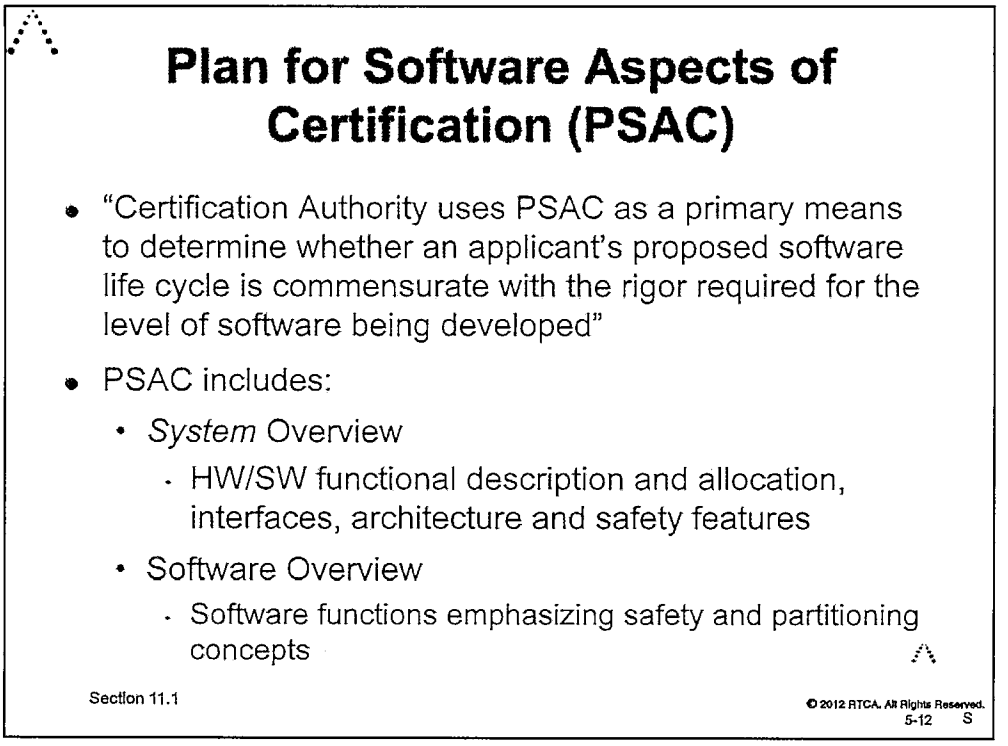
Key activities for the software planning process are listed. - You are in the middle of coding and you want to use a different style, you must say so in your plan. If you don't want to allow it, you must say so. Plans can be revised, but effects must be studied.



These 6 software life cycle data items are plans produced from the Software Planning process. Since the Plan for Software Aspects of Certification is submitted to the FAA (discussed in Module 10) it would probably be a standalone document.

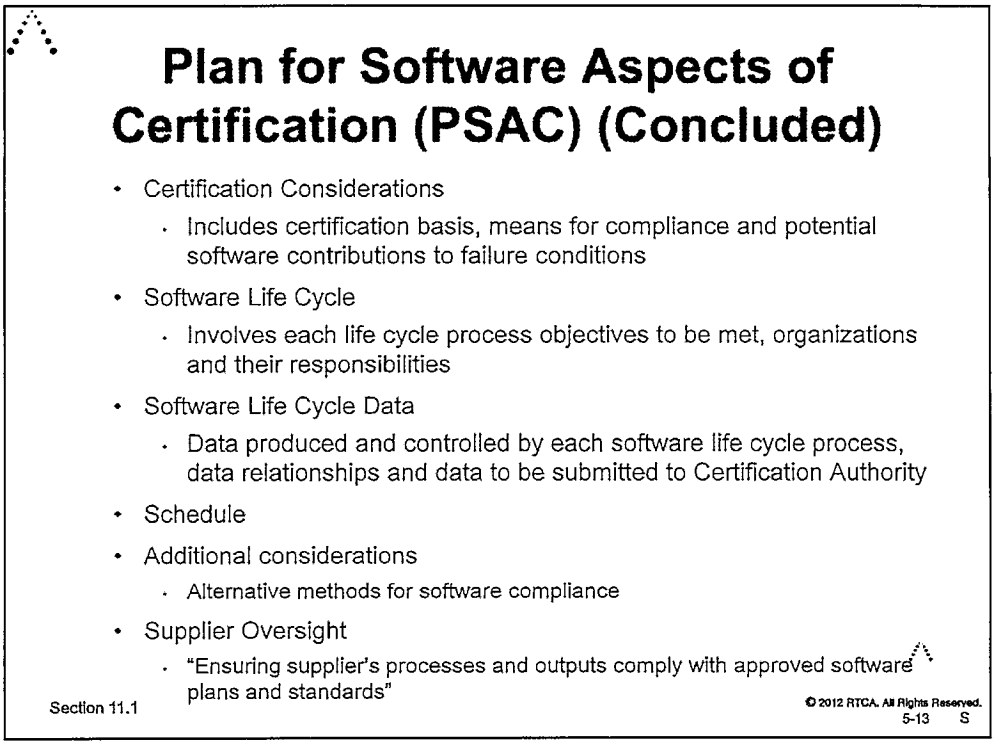
The other data items may each be a document, or many documents, or one or two documents, or be electronic information, or ...

In any case the individuals involved in the project will need access to these planning data items to do their processes and activities correctly.

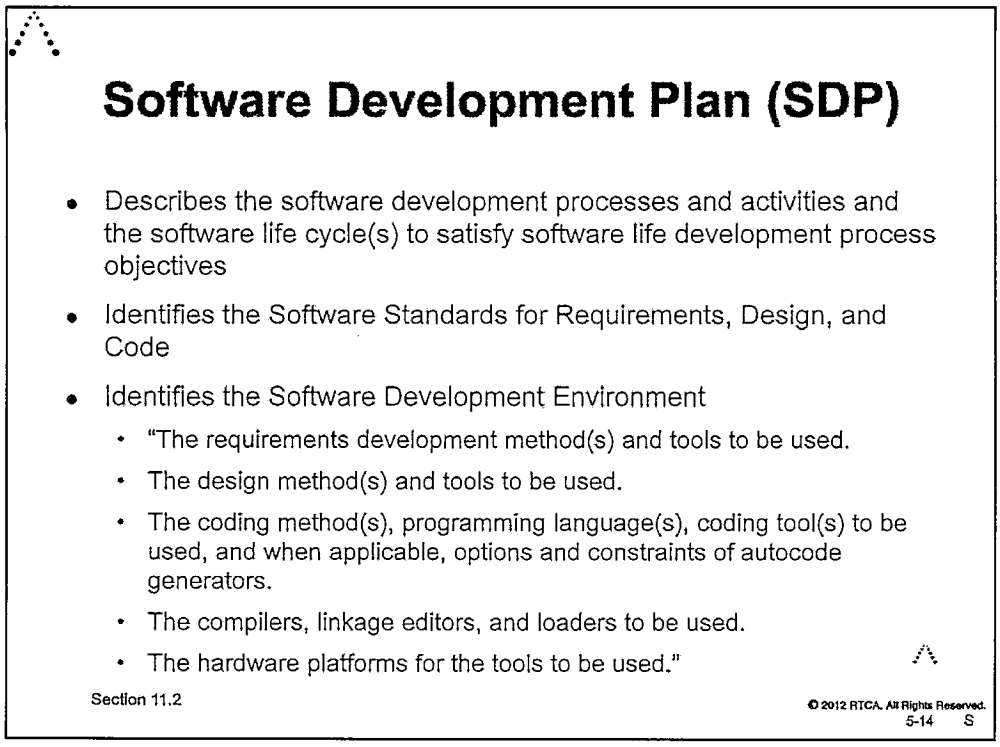


The PSAC will be submitted to the FAA and is used to start a certification project.

The PSAC may reference other plans. They would have to then be submitted to the FAA with the PSAC. "Overview" is what is needed in the PSAC and not a large detailed description.



A short PSAC is good. However, it needs sufficient detail to give the reader an understanding of what will happen.

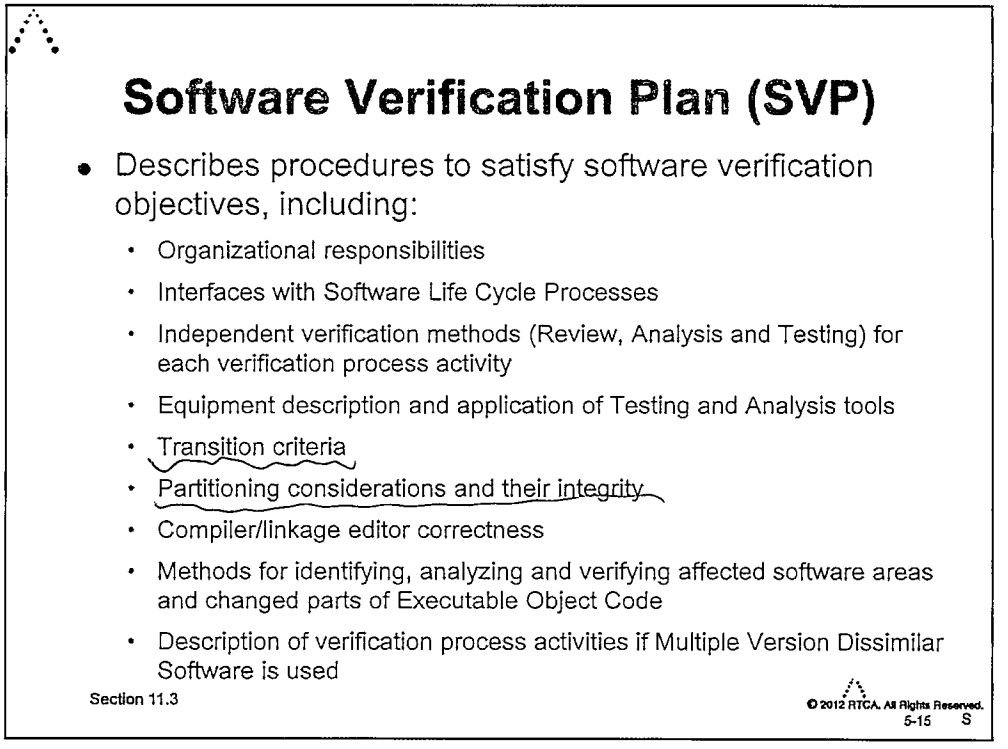


A description of the software life cycle processes to be used to form the specific software life cycle(s) to be used on the project, including the transition criteria for the software development processes.

This data item and the software standards may be in a single document. In fact, this plan as a document may contain all other plans except for the PSAC which should probably be a document in and of itself.

See slide "Software Life Cycle Environment"

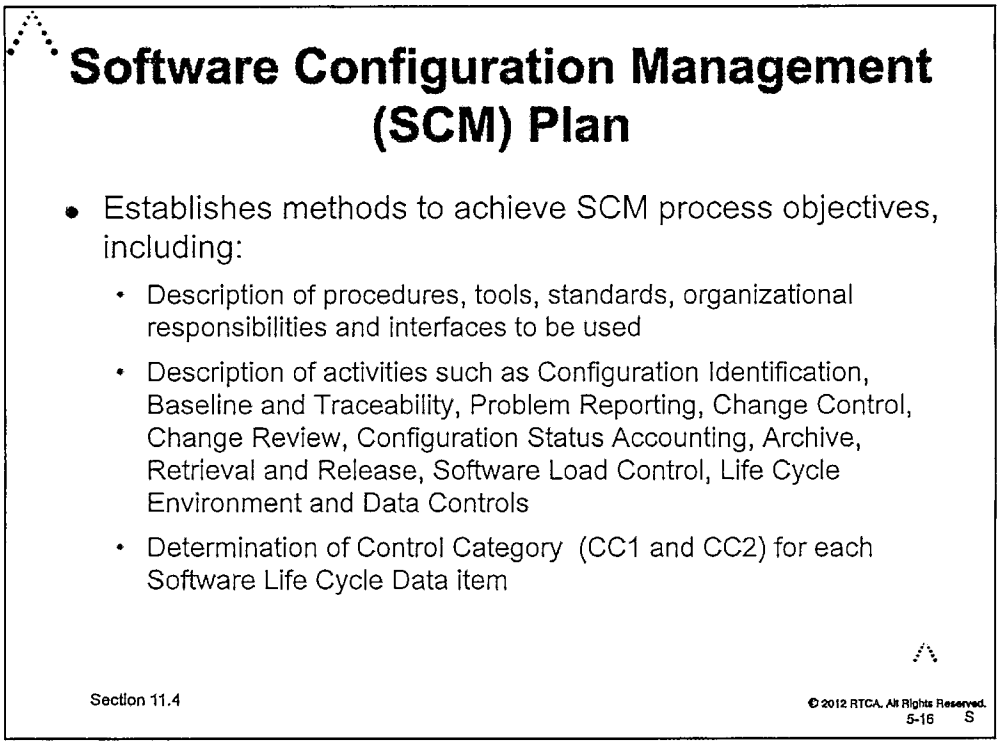
Some people think the PSAC should be three or four pages, some think it should be thousands of pages. It should be detailed enough to show what the applicant plans to do.



Not a classic "Test Plan". The classic "Test Plan" is the "Test Cases And Procedures" data item.

May be included as part of a Software Development Plan document.

Transition Criteria are defined and examples given in Module 4.



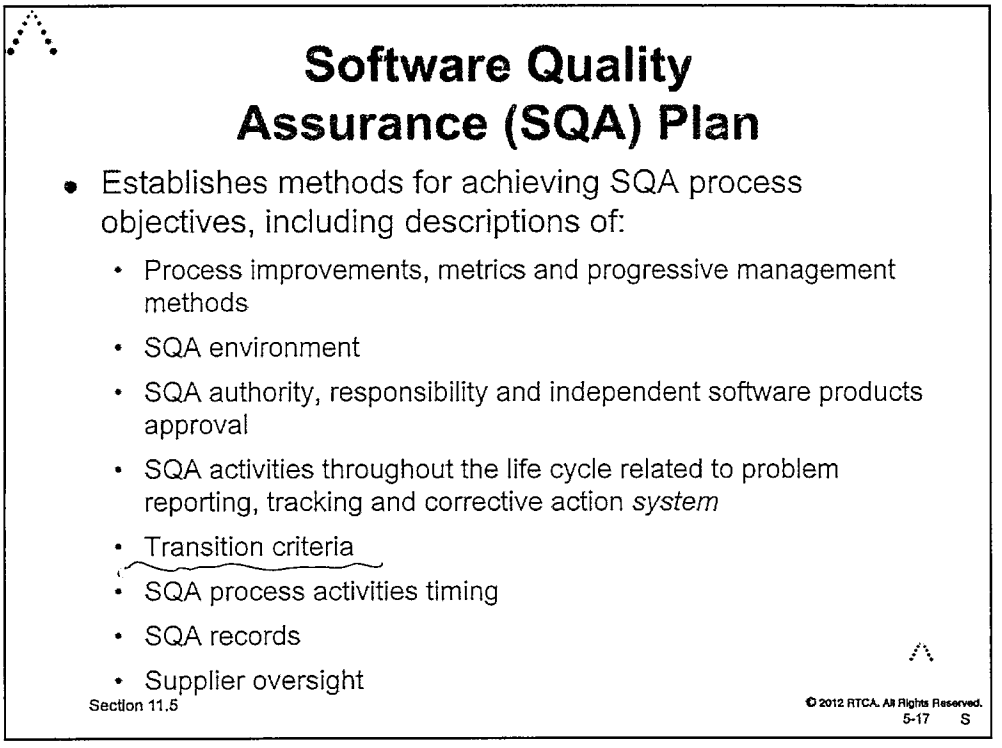
May be included as part of a Software Development Plan document.

CC1 and CC2 are described in Module 8 and Module 11. Also see DO-178C section 7.3

A difference between CC1 and CC2:

- Control Category 1 - use a problem report when changing a configuration controlled data item
- Control Category 2 – no problem report is necessary when changing a configuration controlled data item

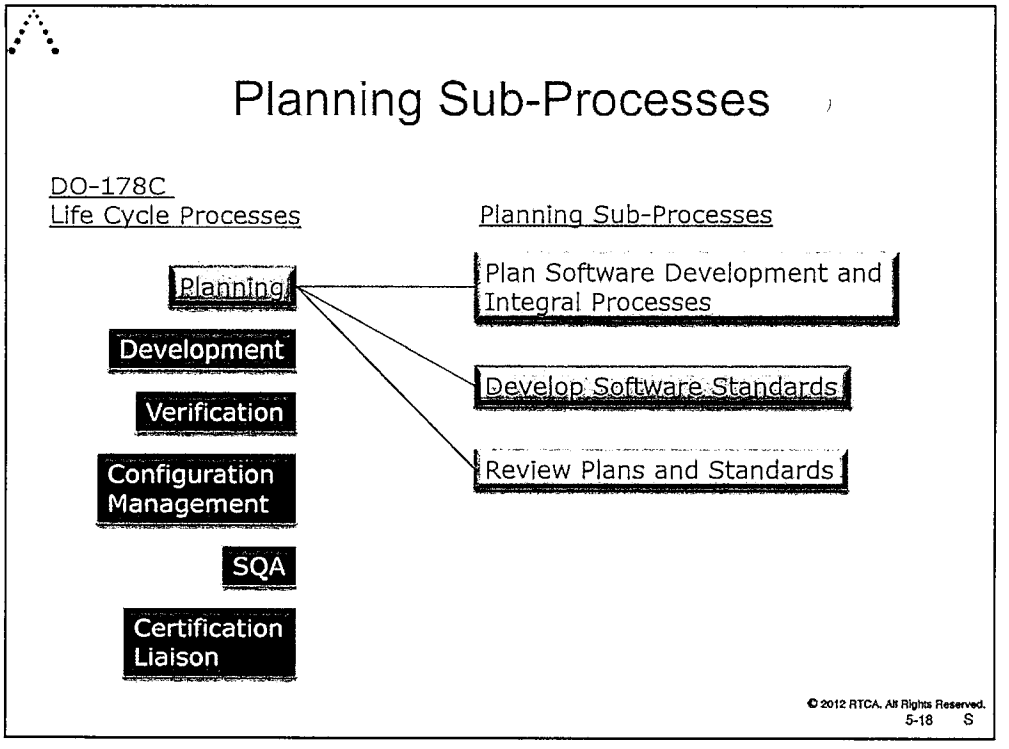
Annex A indicates CC1 or CC2 for all Life Cycle Data items. Software Level A can have CC2 items.



May be included as part of a Software Development Plan document.

Independence is important since SQA's independence is required at all Software Levels. SQA independence needs to be at a level within an organization to stop completion of the program until a process or product issue has been resolved.

Planning may be scattered in many documents, but developers must know how to obtain the plans.

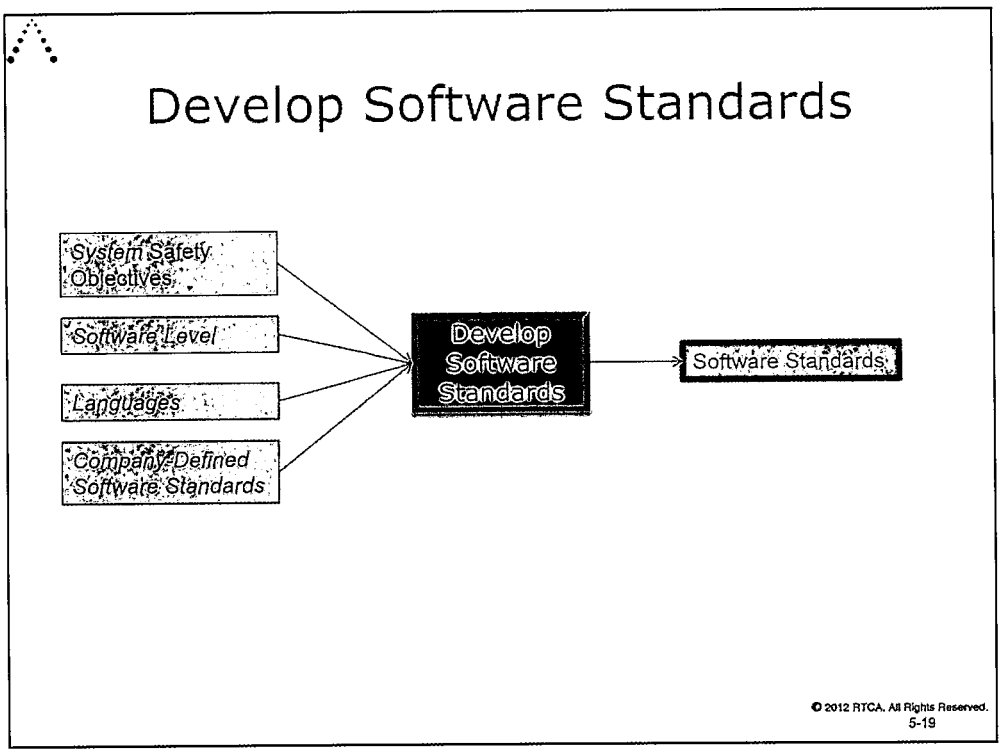


Three sub-processes in the software planning process are:

- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Discuss "activities": Use backup slide defining "activities"

Plans, when followed, need to produce products compliant to DO-178C objectives.



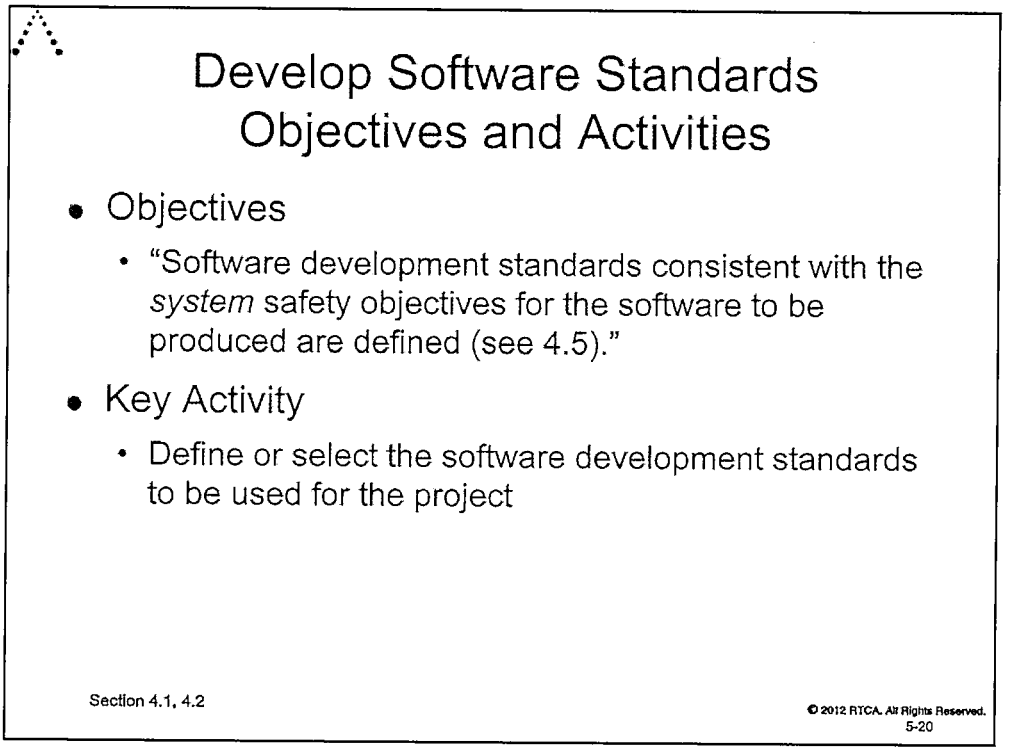
There are three software standards:

- Requirements standard
- Design Standard
- Coding Standard

Company-Defined Software Standards – If your company has standards they can be used. Additions to them may be needed. They can be referenced in part or in whole from a program specific standard. The program-specific standard may tailor the company standards depending on assurance level or specific needs of the program.

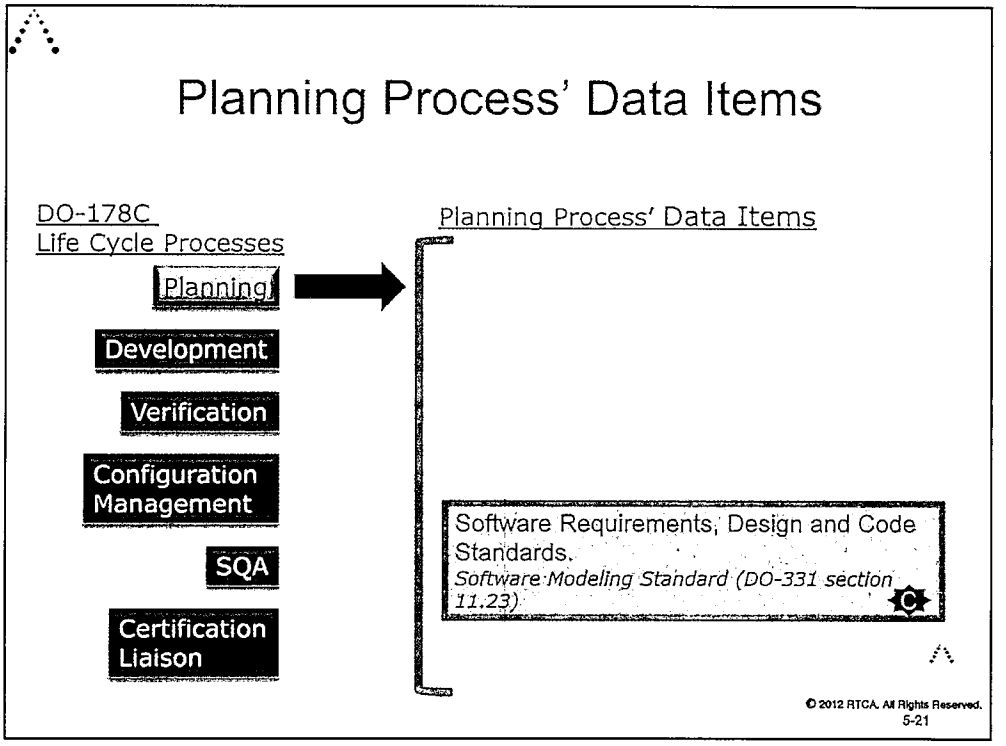
Safety objectives may affect the use of certain allowable architectures (for example – Partitioning) or language constructs (for example – Tasking).

Choice of languages will influence the software standards.



Both the objective and key activities imply existence of software standards.

Note: for Level D design and coding, standards do not have to exist, since design and source code does not have to exist. (see the slide in module 6 titled "Traces During Development" for more information. Just because you do not have to show the Certification Authorities this software life cycle data does not mean that you might not want to actually produce it and use it.



These data items are covered in a slide titled "Software Standards" in the backup slides. It will be discussed next.

The software modeling standard is defined in the Model-Based Supplement.

Software Standards

- **Requirements Standards:** Define methods, rules and tools to be used to develop high level requirements including:
  - Structured methods, notations expressing requirements, constraints on tool use and method to provide derived requirements to *system* processes
- **Design Standards:** Define methods, rules and tools to be used to develop software architecture and low-level requirements including:
  - Design description method(s), naming convention, conditions imposed on permitted design methods, constraints on design and related tool use and complexity restrictions
- **Code Standards:** Define programming languages, methods, rules, and tools to be used to code software including:
  - Source code presentation standards, naming conventions and associated constraints and conditions
- **Software Model Standards** (DO-331): define modeling techniques for each type of model including:
  - Methods and tools, modeling languages, style guidelines and complexity restrictions, constraints on use, requirements identification methods, model element identification, and rational for suitability

Section 11.6, 11.7, 11.8, MB.11.23
5-22 S

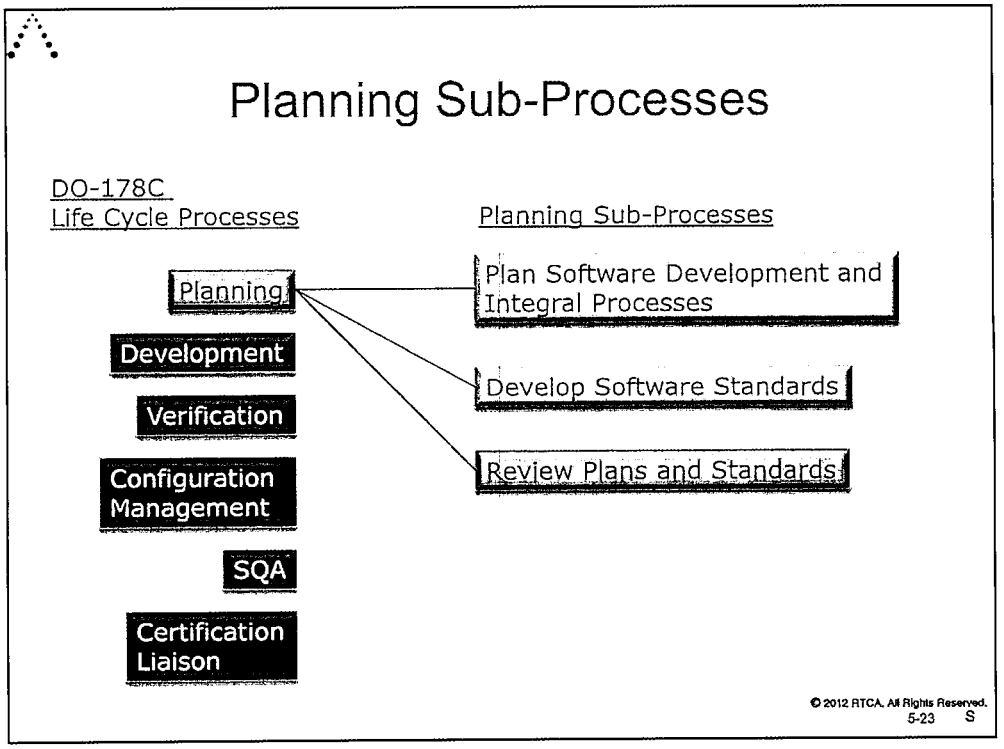
compiler bug list.

Pre-developed standards may be used, they don't have to create them from scratch.

The standards should deal with errors in the compiler?

Don't be foolish choosing your standards.

The standards may be in separate documents, the same document, in the SDP, … They don' t have to be in separate documents.

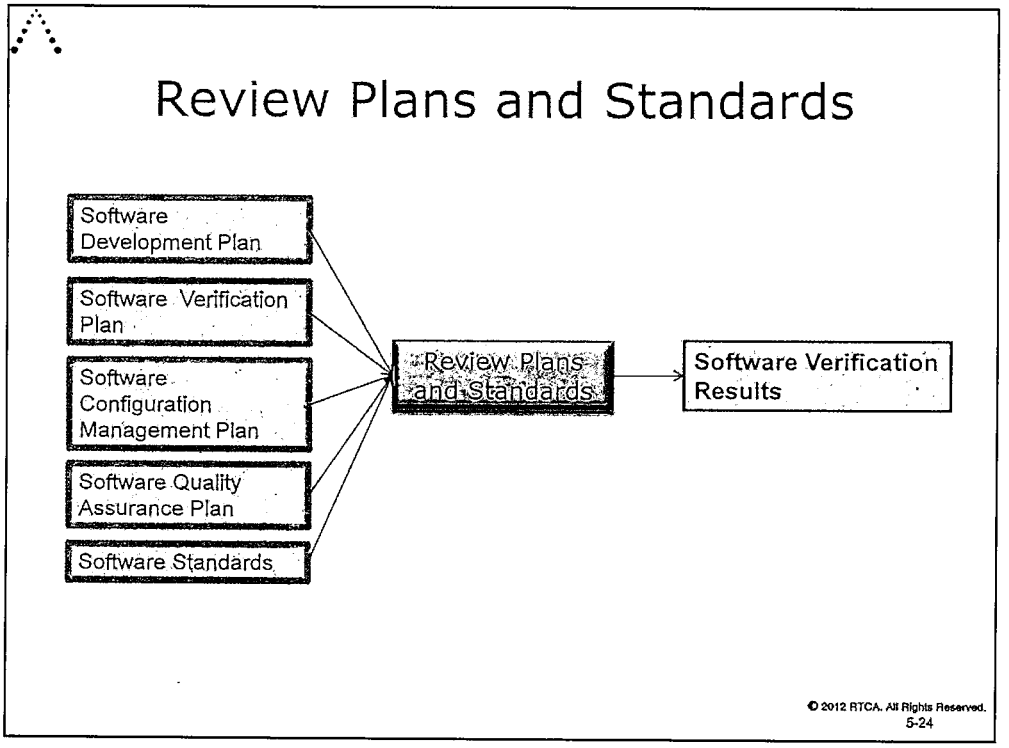


Three sub-processes in the software planning process are:

- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Discuss "activities": Use backup slide defining "activities"

Plans, when followed, need to produce products compliant to DO-178C objectives.

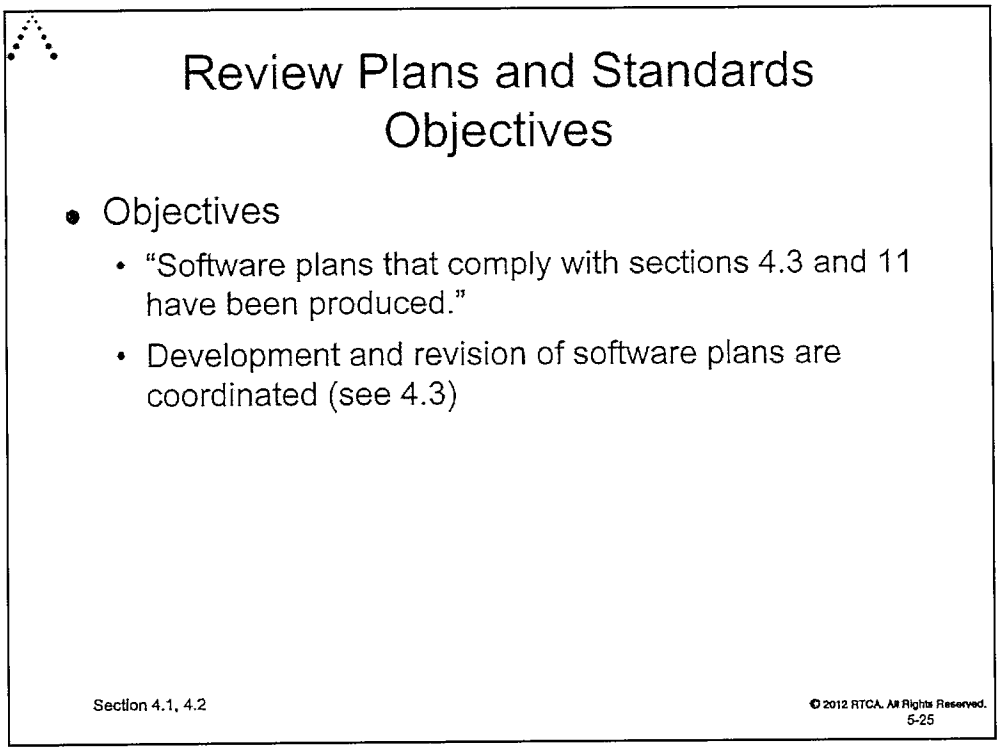


This is a review of the plans and standards to determine that the plans and standards are adequate to achieve compliance with DO-178C (i.e. certifiable airborne *system* and equipment software will be developed).

All organizations involved in the software life cycle processes need to be involved and agree with the plans and standards (includes SCM and SQA). This is the coordination part.

Any changes to the plans in the future should prompt another review.

A good method to accomplish this review might involve listing all the objectives that must be satisfied based on the software level and determining which process and activity produces the evidence of compliance.



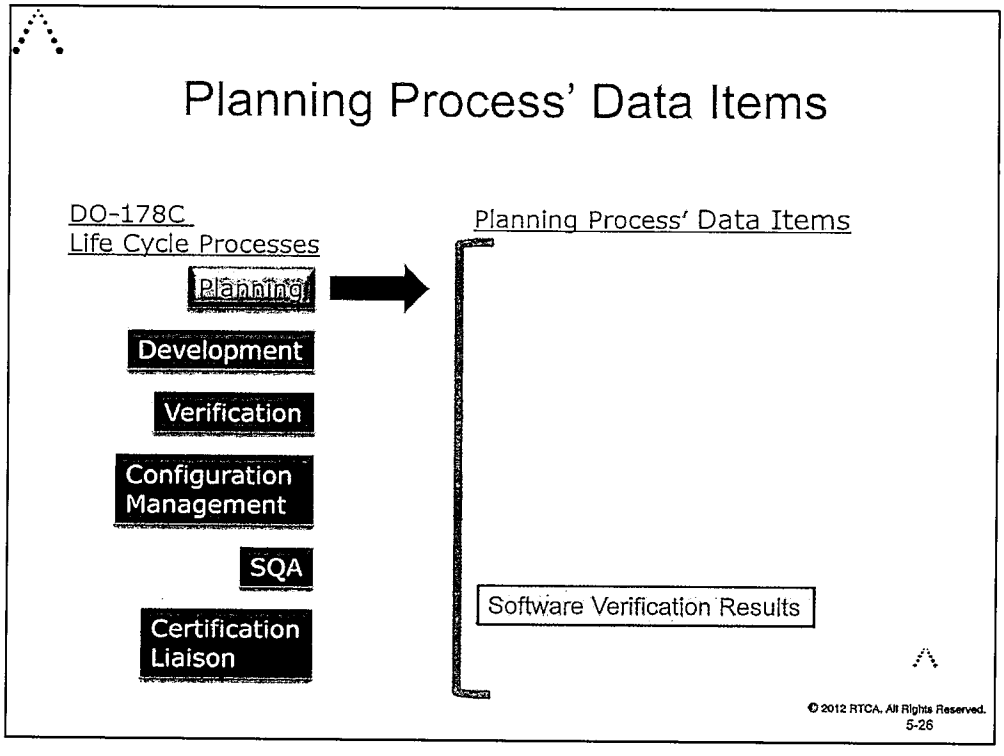
First objective calls for a review to determine that certifiable airborne *system* and equipment software will be developed.

Second objective wants all organizations involved in the Software Development Life Cycle processes be involved in the determination of all the plans and standards and not just the ones for which they are responsible. All the plans must work together.

Constraints on the task: (verification criteria)

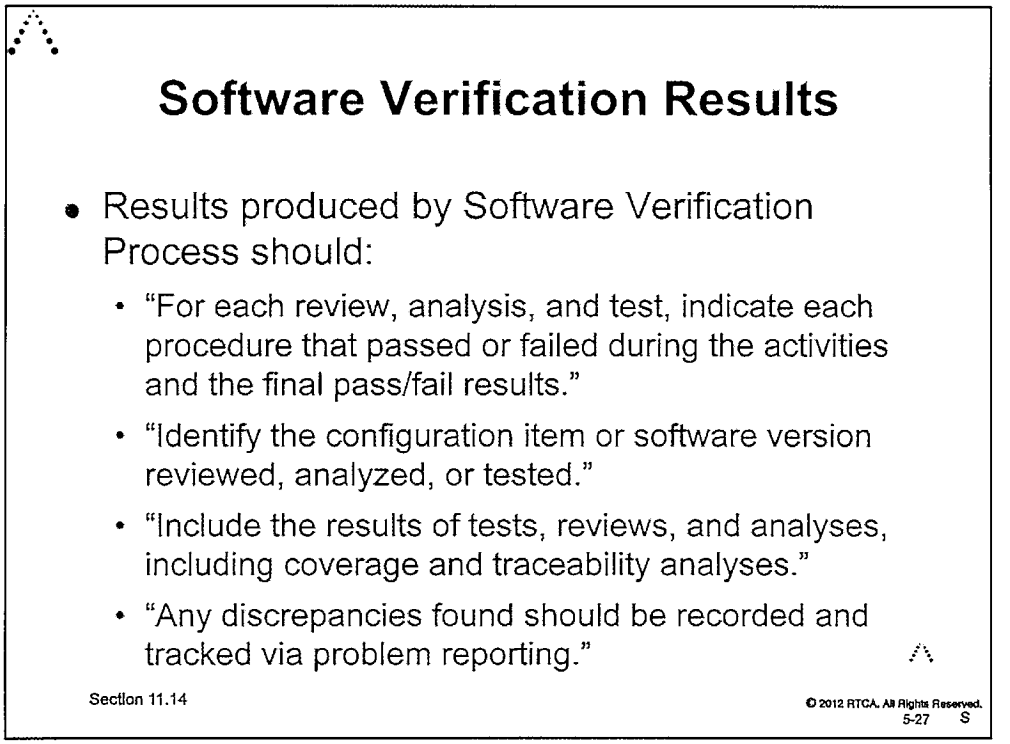
- The software plans should comply with this document.
- The software plans should define the transition criteria for software life cycle processes by specifying:
    - The inputs to the process, including feedback from other processes.
    - Any integral process activities that may be required to act on these inputs.
    - Availability of tools, methods, plans, and procedures.
- The software plans should state the procedures to be used to implement software changes prior to use on a certified product. Such changes may be as a result of feedback from other processes and may cause a change to the software plans.

Plans may need to be modified. Especially if during the software life cycle it is determined that the plan is preventing com0pliance to DO-178C



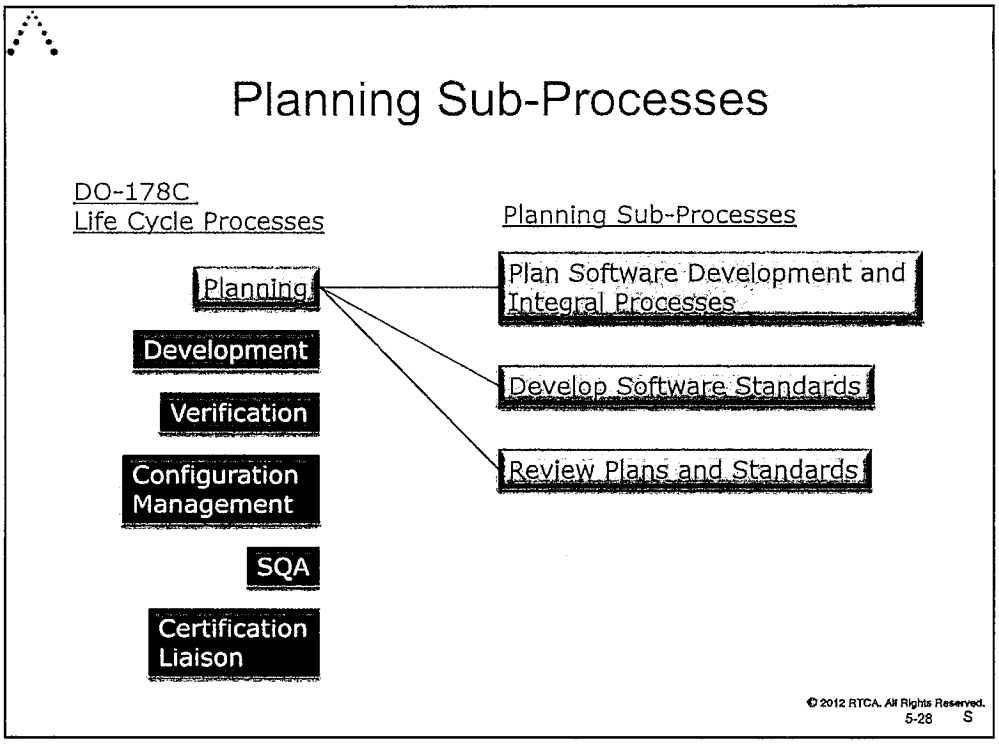
The output of the review will be a Software Verification Data Item. Note: this is not simply an SQA activity so the evidence of completion is not considered an SQA Data Item. However SQA may keep its own records of an audit of this activity.

Control your CM Plan. The plans have to control themselves. Put into your plans a mechanism to allow that.



Catch-all Item.

Planning has to tell you how you will review the plans

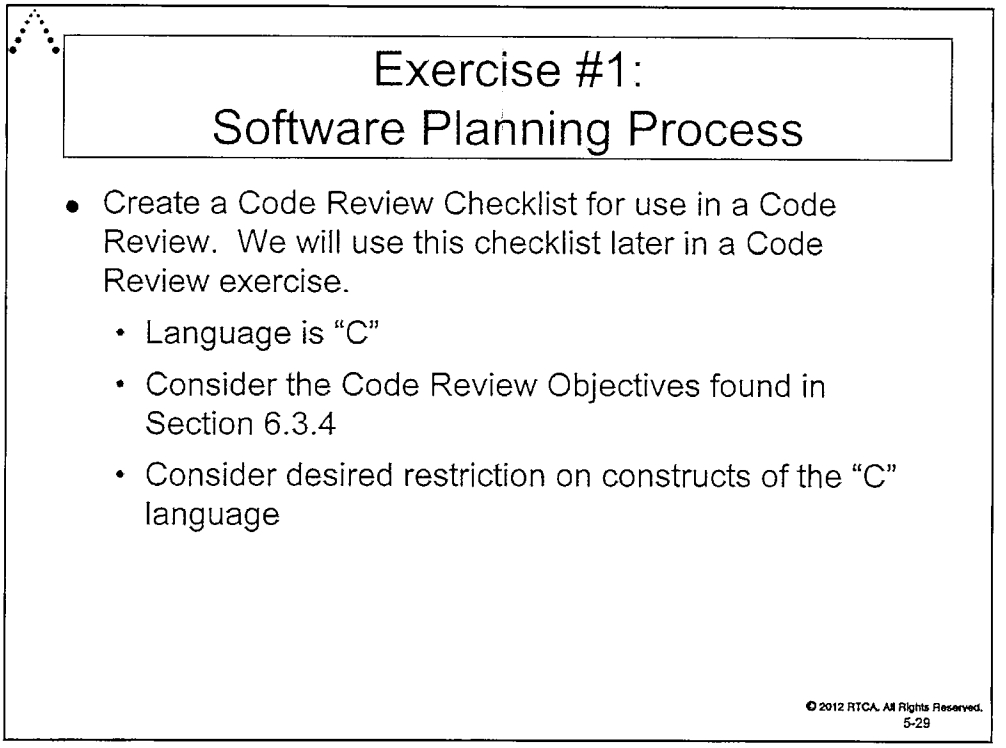


Three sub-processes in the software planning process are:

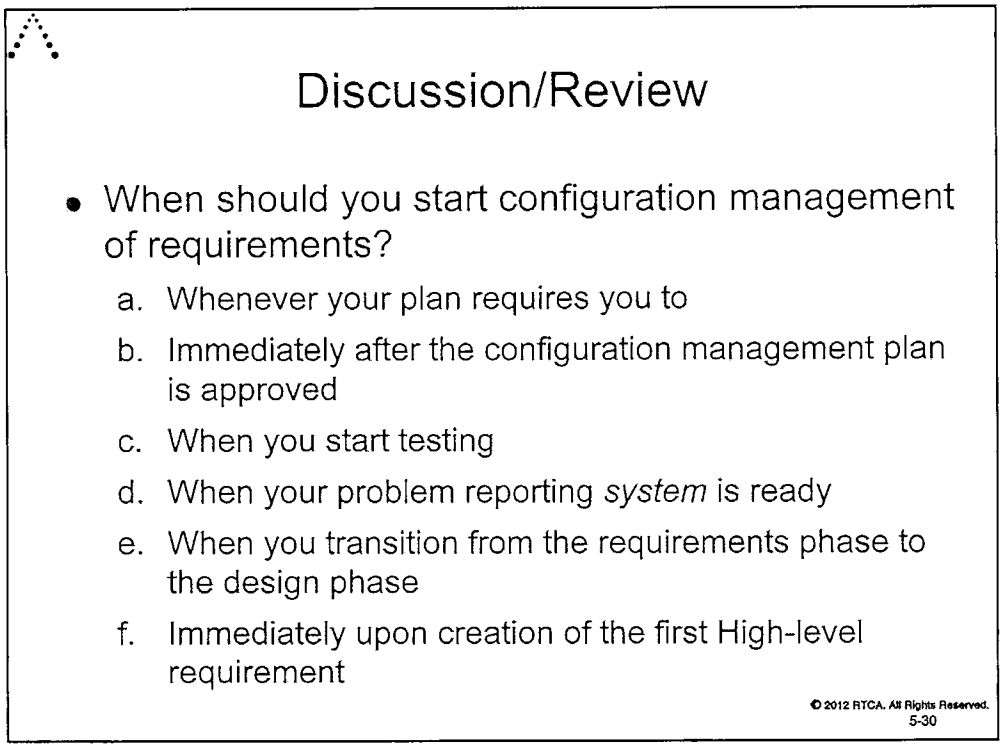
- Plan SW Development and Integral Processes
- Develop Software Standards
- Review Plans and Standards

Discuss "activities": Use backup slide defining "activities"

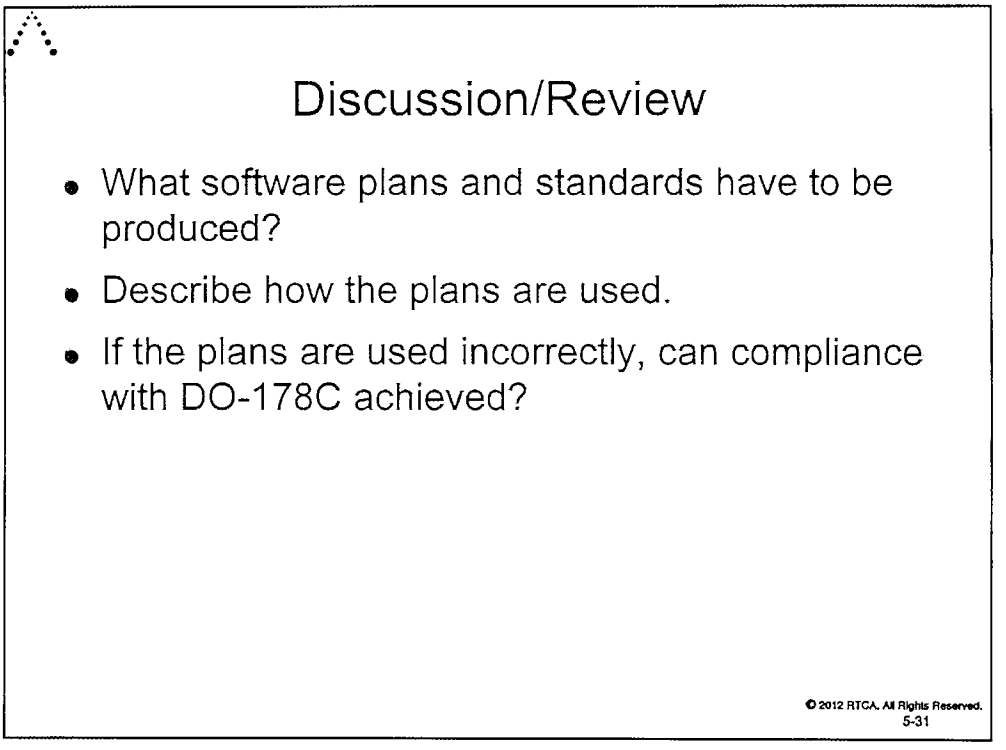
Plans, when followed, need to produce products compliant to DO-178C objectives.



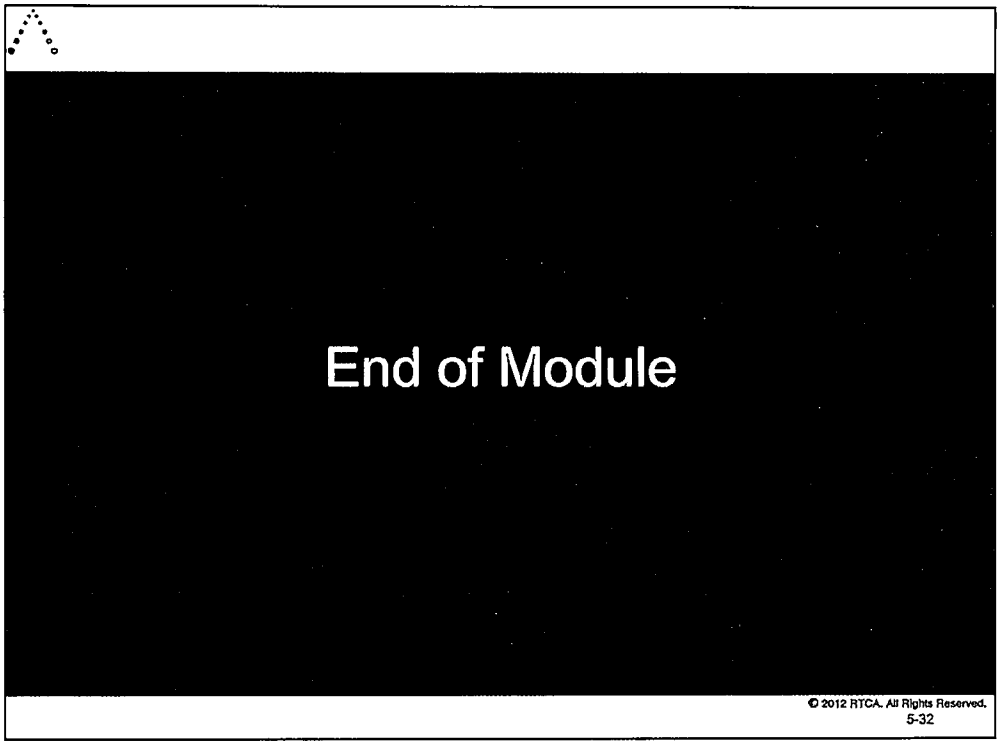
Section 6.3.4 is not a checklist for a review.



The answer is F. The plans **may** chose one of the A-E alternatives so any of these may be considered correct. The plans should start configuration management of requirements at a time that is best for the capturing of requirement errors and their resolution. Having a history trail of requirement changes and the reasons for those changes is invaluable to complete understanding of issues in the later stages of the software life cycle.



- All software plans have to be produced. If the Software Level is D, then the design and coding standards do not have to be produced.
- The plans should be followed since they specify which and when each process and activity within a process starts and completes.
- Failure to do an activity, or doing an activity incorrectly, would be considered "applying the plans incorrectly". By definition this would produce non-compliant software (See module 9). However the plans may not have considered a particular development or verification issue and the correct activity differs from the planned activity. In this case the plans need to be revised and re-reviewed.
- The FAA may or may not approve any plans.


End of Module
© 2012 RTCA. All Rights Reserved.
5-32

Exercise # 1:
System Requirements Allocated to Software

Altitude Display V1.0

1. The equipment shall read pressure from the pressure sensor in pounds per square inch.
2. The equipment will calculate altitude in hundreds of feet from pressure in pounds per square inch.
3. The equipment shall display altitude in hundreds of feet.

Create a Code Review Checklist for use in a Code Review. We will use this checklist later in a Code Review exercise.

Language is "C"

Consider the Code Review Objectives found in Section 6.3.4

Consider desired restriction on constructs of the "C" language

The DO-178C (section 6.3.4) objectives for a Source Code Review are:

a. Compliance with the low-level requirements: The objective is to ensure that the Source Code is accurate and complete with respect to the low-level requirements and that no Source Code implements an undocumented function.

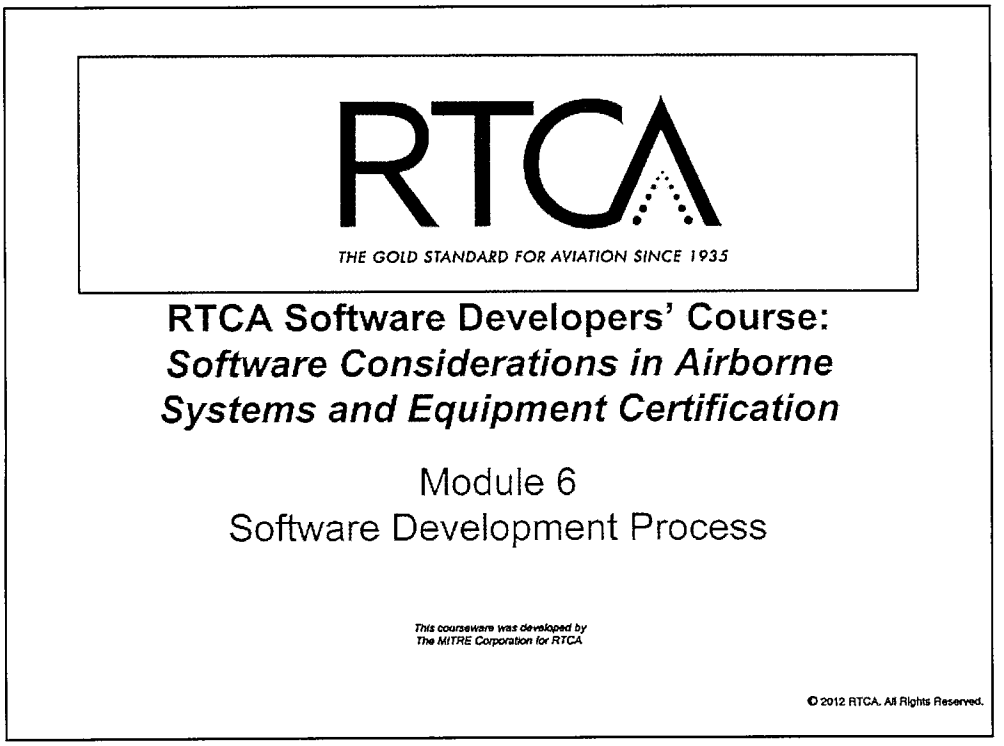
b. Compliance with the software architecture: The objective is to ensure that the Source Code matches the data flow and control flow defined in the software architecture.

c. Verifiability: The objective is to ensure the Source Code does not contain statements and structures that cannot be verified and that the code does not have to be altered to test it.

d. Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, for example, complexity restrictions and code constraints. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified.

e. Traceability: The objective is to ensure that the low-level requirements were developed into Source Code.

f. Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, memory usage, fixed point arithmetic overflow and resolution, floating-point arithmetic, resource contention and limitations, worst-case execution timing, exception handling, use of uninitialized variables, cache management, unused variables, and data corruption due to task or interrupt conflicts. The compiler (including its options), the linker (including its options), and some hardware features may have an impact on the worst-case execution timing and this impact should be assessed.



This 2-hour module introduces the Software Development Processes. It also introduces the participant to the differences between High-level and lower level requirements. The Software Design process, Coding and Integration Processes, and concept of traceability will be presented.

The software requirements process uses output of *system* life cycle processes to develop High-level requirements including functional, performance, interface and safety requirements

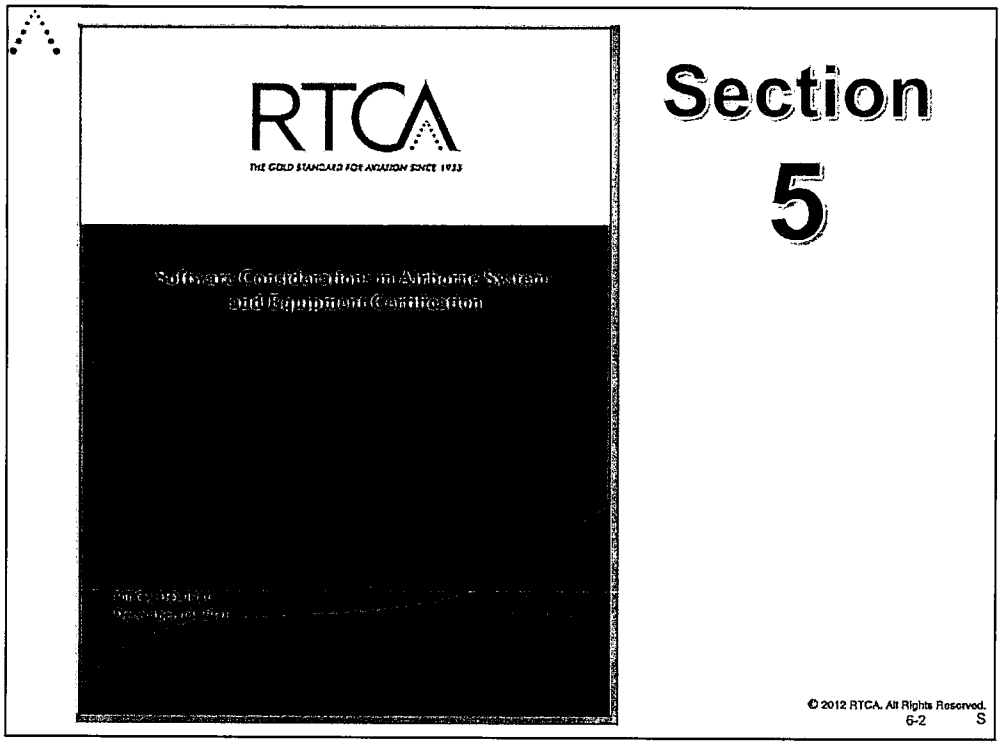
The objectives of the software design process are to develop Low-level requirements from High-level requirements and provide them to the *system* processes. This process provides software design description and software architecture.

The software coding process supports development and implementation of source code from software architecture and Low-level requirements.

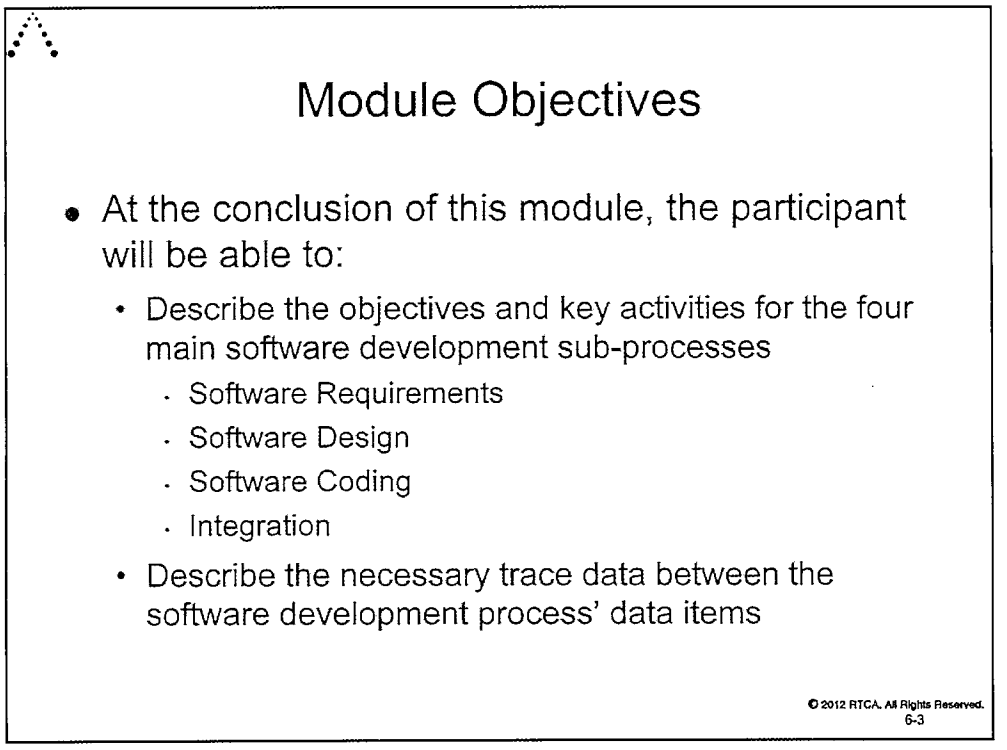
The integration process produces Executable Object Code and associated parameter data item files and loads them into target hardware/computer for hardware/software integration.

The software development process traceability provides tracing of data showing bi-directional association between: 1) *system* requirements allocated to software and High-level requirements to verify implementation of *system* requirements; 2) High-level and Low-level requirements to verify complete implementation of High-level requirements; and 3) Low-level requirements and source code to verify that no source code implements an undocumented function.

This is a straightforward module.



SHOW Figure 2-1 on the second screen

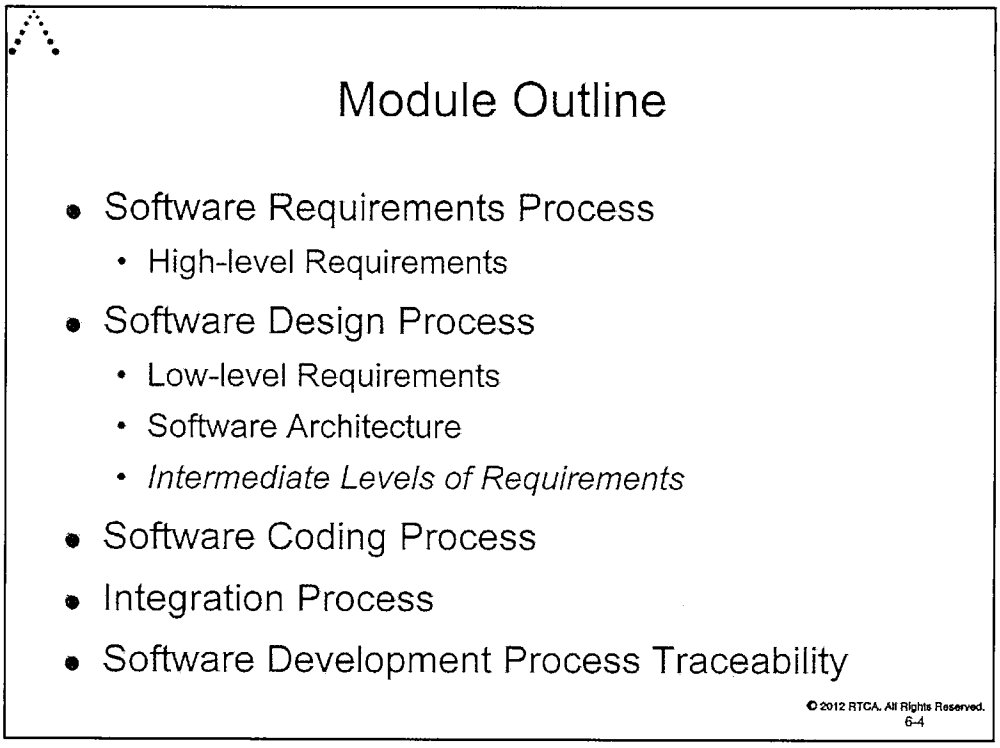


This slide covers the objectives of Module 6.

When you are done, these things will exist.

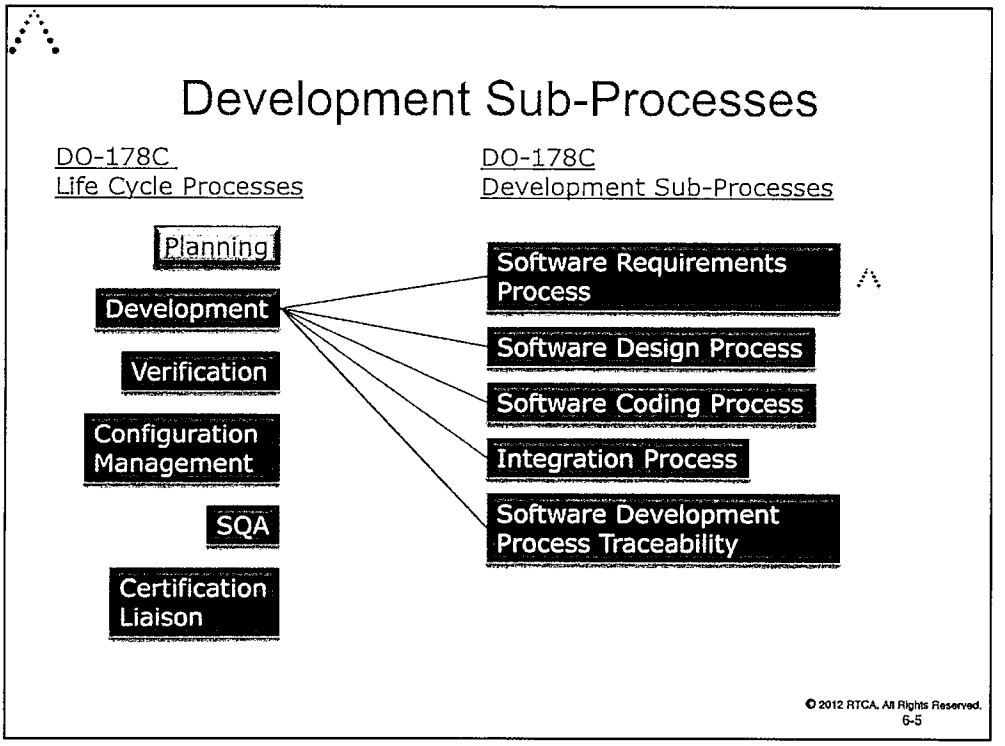
Software Requirements are High-level Requirements and derived High-level Requirements.

Software Design will include Low-level requirements, derived Low-level Requirements (both of which will be considered as software requirements for verification testing), and software architecture.



In this module we will cover inputs and outputs for each process as well as objectives and key activities.

Intermediate levels of requirements are described in Section 6.1.b. They are included here since they may be part of a software design process which refines High-level requirements into Low-level requirements through several refinement steps. They should be part of Software Requirements Data or the Design Description (section 11.9 or 11.10) and should be verified like any other High-level or Low-level requirement.

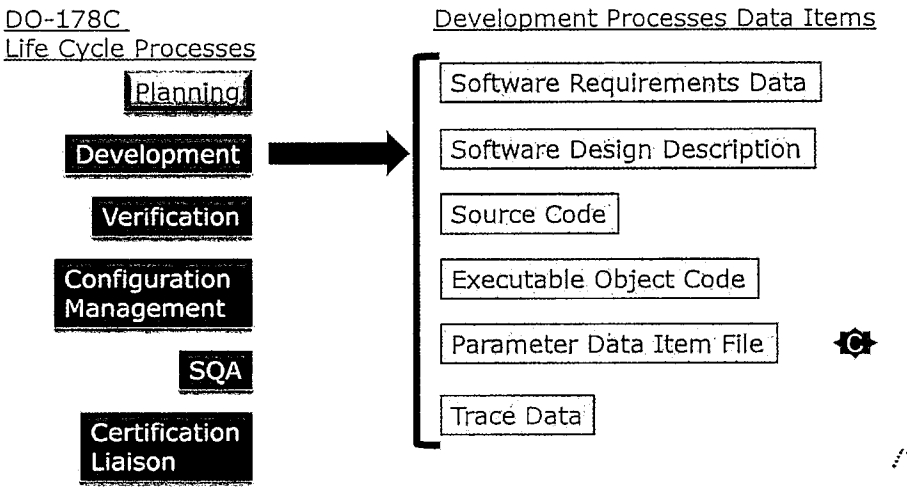


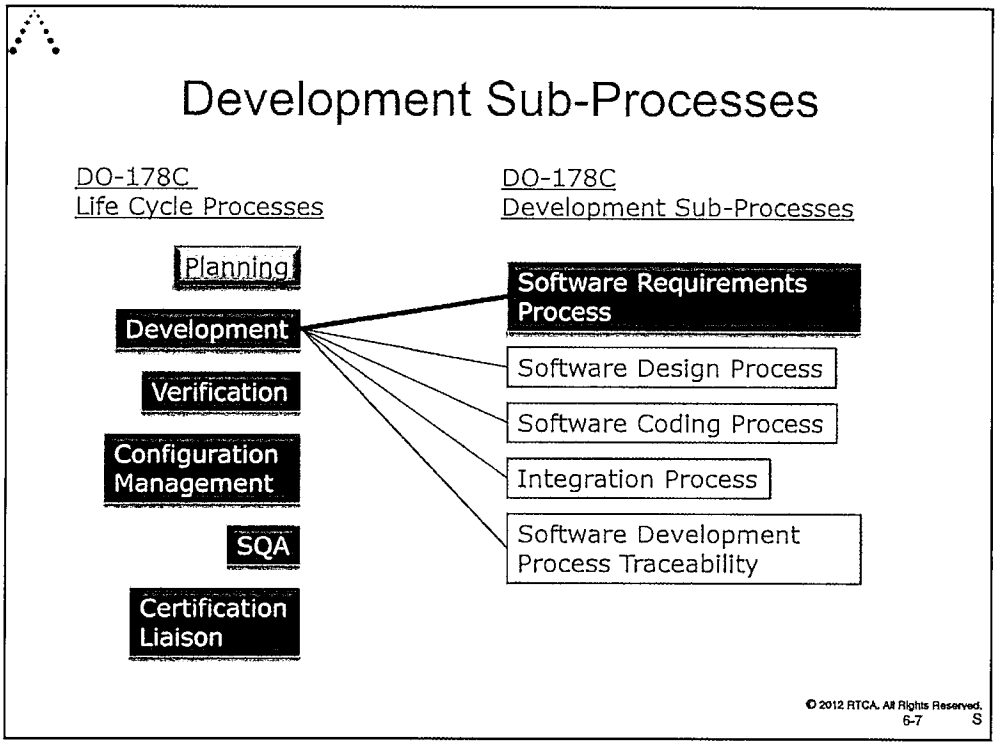
The DO-178C Software Development Process has 5 Sub-Processes:

- Software requirements sub-process
- Software design sub-process
- Software coding sub-process
- Software integration sub-process
- Software traceability sub-process

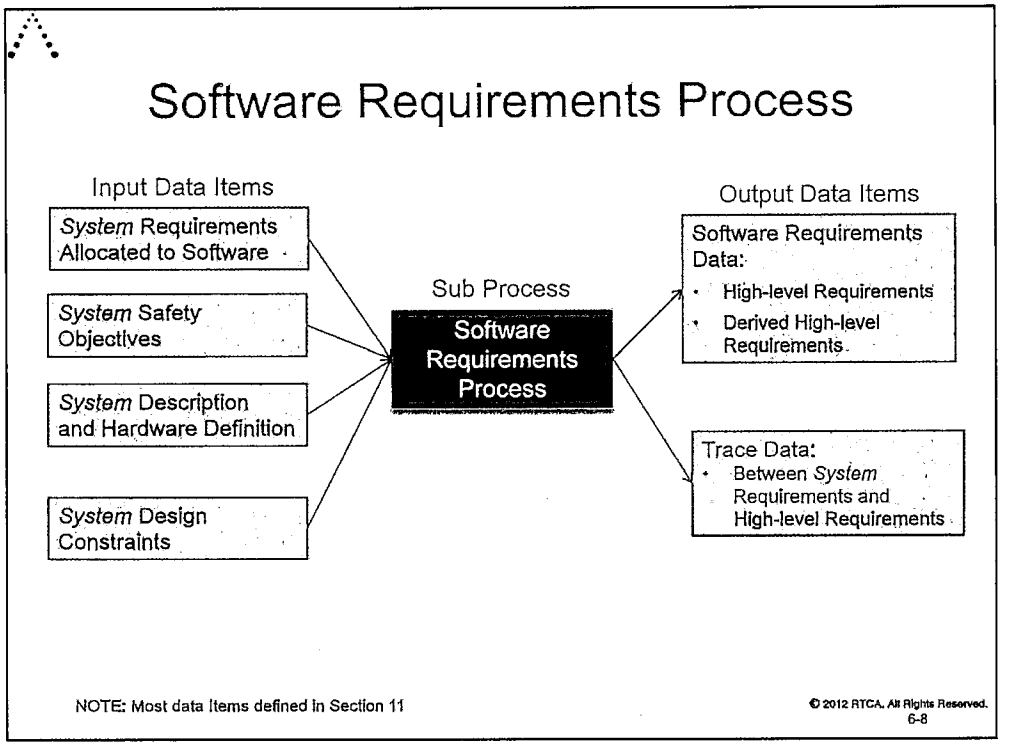
DO-178C does not impose a timing or ordering constraint on these sub-processes. Any timing or ordering constraints are imposed by the transition criteria defined in the Software Development Plan. Do what makes sense within your development processes. Processes may be done in parallel or sequentially.

Development Process Data Items





The DO-178C Software Development Process has 5 Sub-Processes; the Software Requirements Process is the first development Sub-Process to be covered in this session.

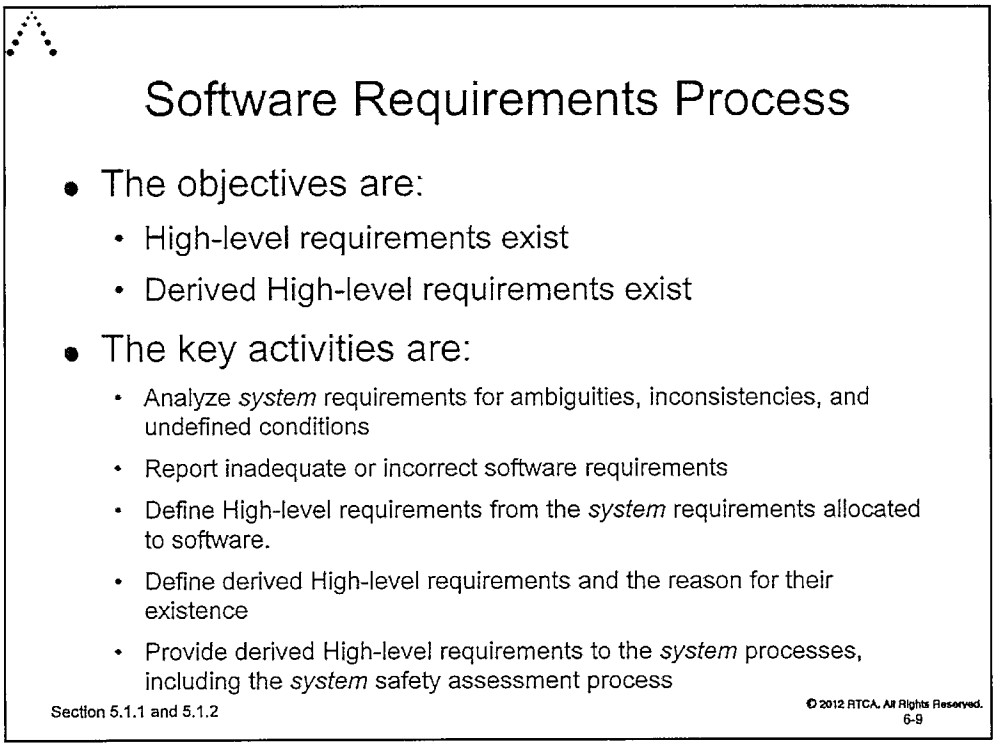


Most important input is the *System* Requirements Allocated to Software. The other inputs influence the decomposition of them into High-level and Derived High-level Requirements.

Design constraints, *System* description and hardware definition, and *System* Safety Objectives may include (not an inclusive list):

- Use of Partitioning
- Use of Multiple-Version Dissimilar Software
- Use of COTS (Commercial-Off-The-Shelf)
- Use of specific external interfaces,
- Safety Monitoring
- Timing and Memory Margins
- Languages
- Target Hardware

The Software Development Plan and the Software Standards could be considered inputs to this and all other development sub-processes. They are not shown here since they guide the sub-processes in the transformation of input data to output data.



These objectives imply existence of two Software Life Cycle Data Items:

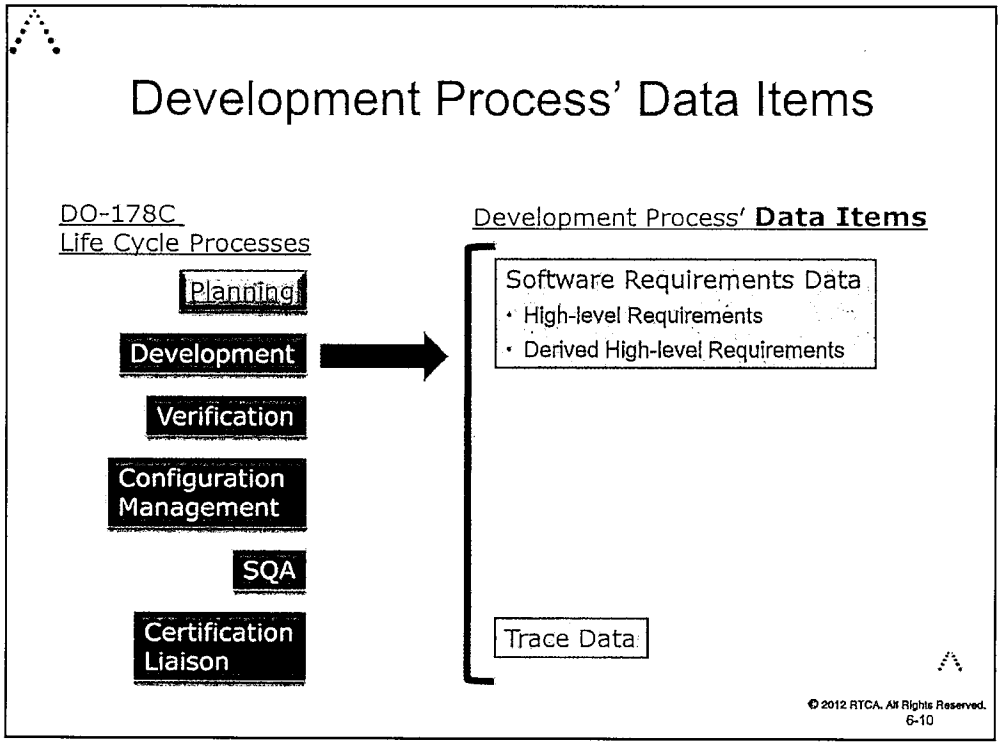
- High-level requirements
- Derived High-level requirements

They are both considered to be the "Software Requirements Data" of the Software Life Cycle Data Items.

This slide does not list all the activities in section 5.1.2. These are activities that should be completed. The other activities in section 5.1.2 may be applicable to this sub-process or may offer information on ways to meet the objectives of DO-178C when activities for the development process are being defined in the Planning process. This concept applies to all the other sub-processes in the development process and will not be repeated in the notes.

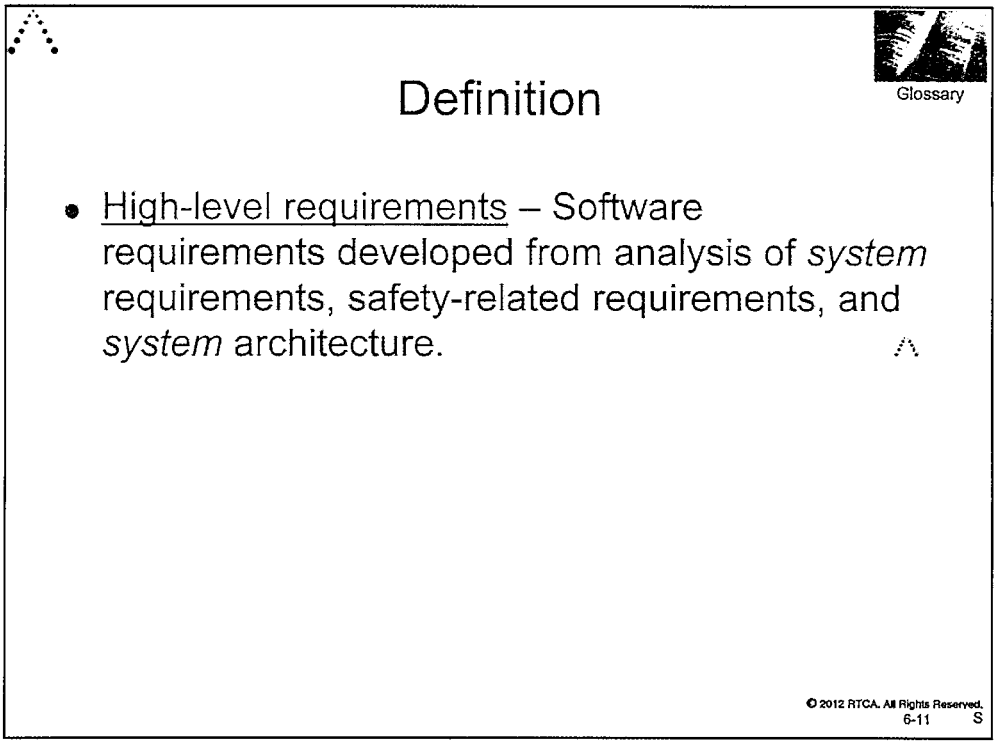
Possible example derived requirement:

> We had issues with ranges (restricted ranges may be a derived requirement; so system safety can decide whether safe flight can be achieved.)
>
> Negative 2,300 feet is the correct answer to the question of what is the lowest altitude that can be safely flown. Software developers did not know this height. They thought it was 200 feet. The software said this is a nonsensical value. The instrument needed to be looked at from a safety perspective.



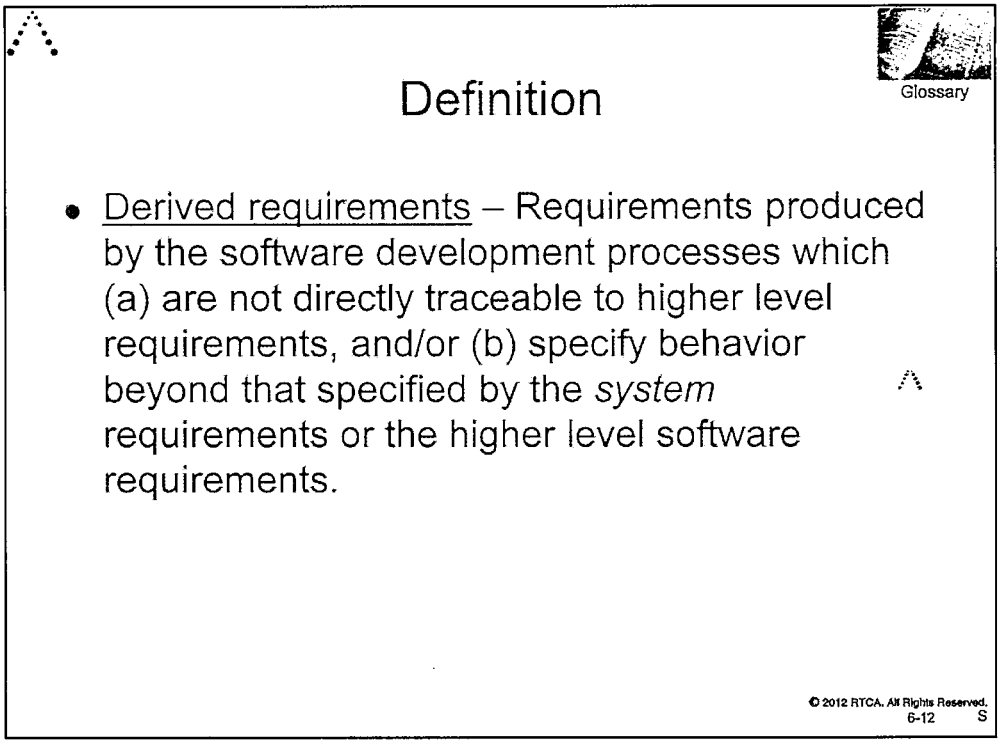
Two Software Life Cycle Data items are created by this sub-process: (see Second Screen slide titled "Software Requirements Data")

- Software Requirements Data, which includes:
  - High-level requirements
  - Derived High-level requirements
- Trace data



System requirements allocated to software

- look for inconsistencies
- may need architecture
- and do derived requirements.



FAQ #36: What is the definition or interpretation of derived requirements in DO-178C/DO-278A?

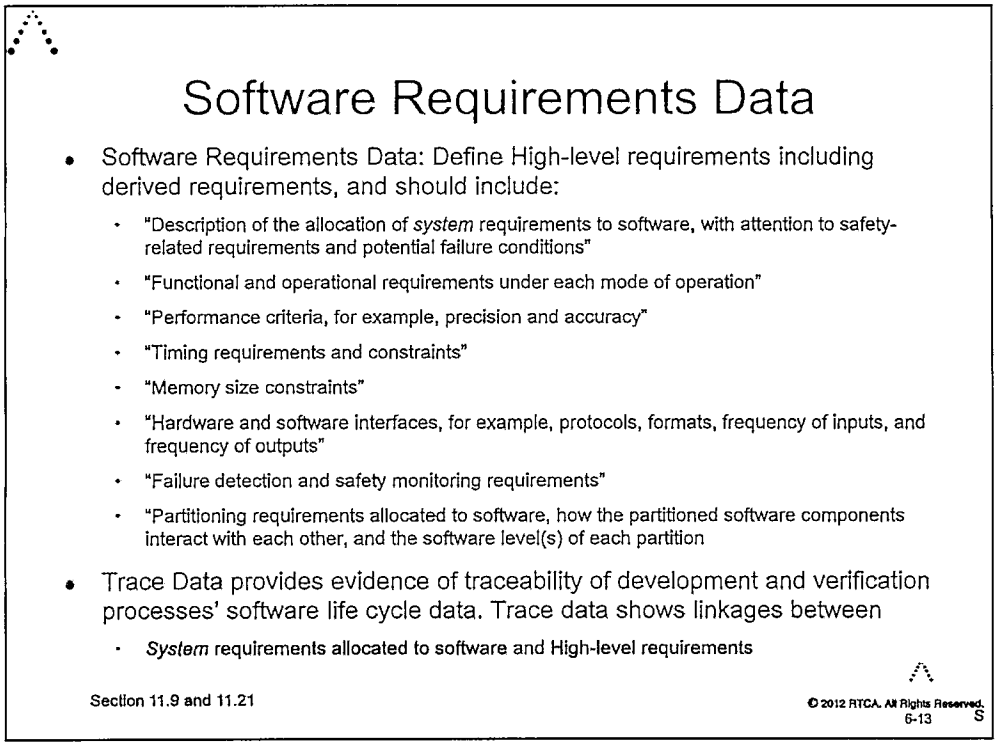
There are two definitions.

Example, test software that is software Level E in a software Level A environment (the software controlling the coffee pot warmer was embedded in the fly-by-wire control system) .

If partitioning of those two sets of functions are not implemented, a safety hazard may have been created. Safety needs to review the system's observable behavior for every requirement. Partitioning may not have been a requirement at the system level; and therefore partitioning was not traceable.

Example of traceability: Inertial Navigation / GPS Navigation Equipment.

Functionality was added by the implementer that used values which weren't defined – Identity Matrix of up to 12 values for magnetic deviation.
The coder said, if in my input device, my value is not between 8 and 12 coefficients . He loaded 12 values. We have no idea what's in there, it could be zero. You have a derived requirement that did not go back up (to system safety). That's a functionality that was not specified.

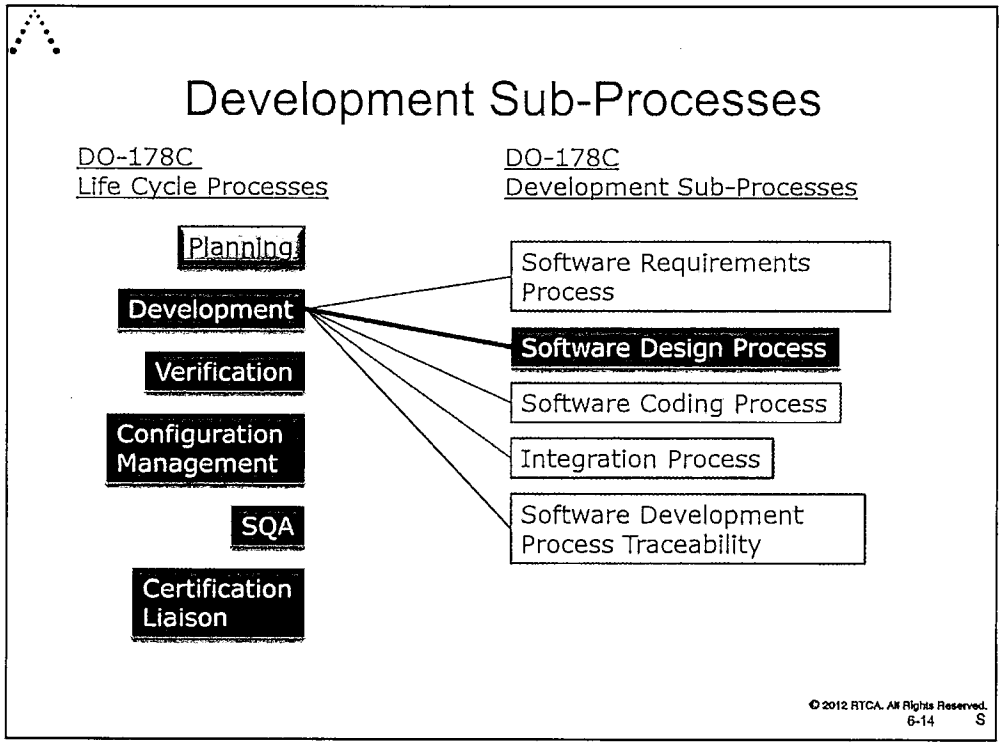


Keep the requirements positive. DO NOT have a requirement that uses the phrase "shall not". For purposes of DO-178C, requirements should specify functionality.

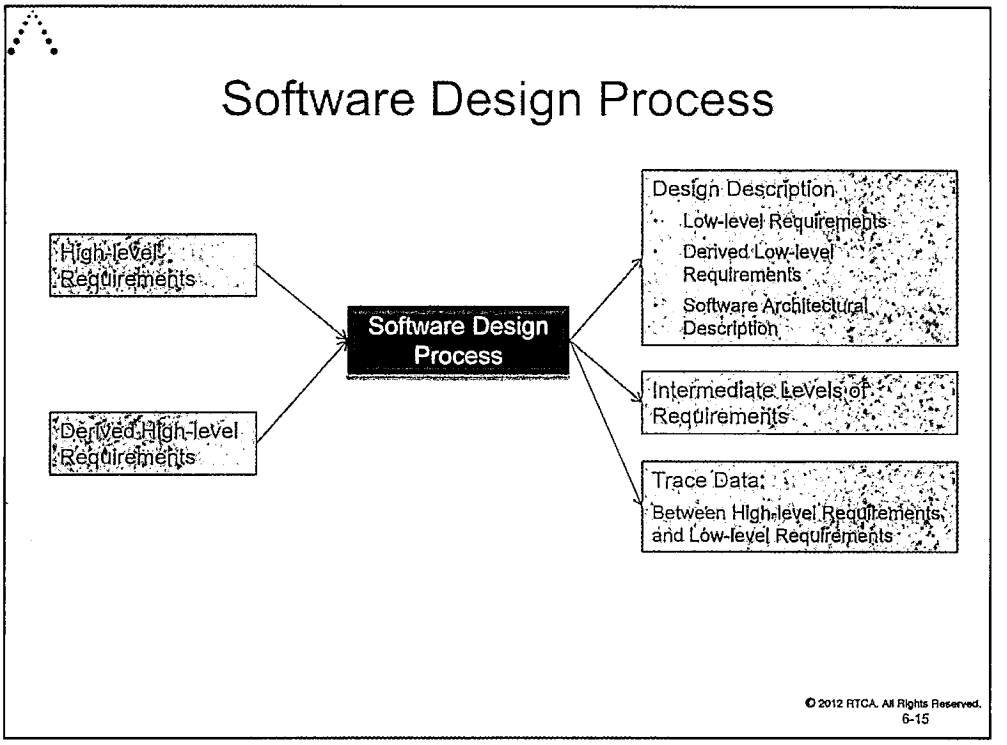
Trace data will be discussed in the last "Software Development Process Traceability" charts at the end of this module. The information here is included for completeness.

The trace data specified in this slide may be created during the sub-process that creates the Design Description or it may be later in a separate sub-process.

V2 for all failure mode -



The DO-178C Software Development Process has 5 Sub-Processes; the Software Design Process is the second development Sub-Process to be covered in this session.



The High-level requirements are developed into software architecture and Low-level requirements that satisfy the High-level requirements.

In the next sub-process, source code is created from Low-level requirements.

High-level requirements are also the Low-Level requirement:

In the next sub-process, source code is created from Low-level requirements. However, if High-level requirements are simple enough, source code can be created from them. (DO-248C, section 3.81) If source code is generated directly from High-level requirements, then High-level requirements are considered to be both High-level and Low-level requirements. (DO-248C, section 3.84) Therefore, this sub-process is not necessary.

Intermediate levels of requirements (optional)

- "If one or more levels of software requirements are developed between High-level requirements and Low-level requirements, the successive levels of requirements are developed such that each successively lower level satisfies its higher level requirements." Section 6.1.b.
- They are included here since they may be part of a software design process which refines High-level requirements into Low-level requirements through several refinement steps. They should be part of Software Requirements Data or the Design Description (section 11.9 or 11.10) and should be verified like any other High-level or Low-level requirement. They do not have to be included in the traceability from High-level to Low-level requirements

Software architecture

In the next sub-process, the software architecture and Low-level requirements are developed into Source Code that satisfies the Low-level requirements and software architecture.

Annex A Table A-2
Low-Level Requirements

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
4	Low-level requirements are developed.	5.2.1.a	5.2.2.a 5.2.2.e 5.2.2.f 5.2.2.g 5.2.3.a 5.2.3.b 5.2.4.a 5.2.4.b 5.2.4.c 5.5.b	O	O	O		Design Description Trace Data	11.10 11.21	① ①	① ①	① ①	

LEGEND:
- ● The objective should be satisfied with independence.
- O The objective should be satisfied.
- Blank Satisfaction of objective is at applicant's discretion.
- ① Data satisfies the objectives of Control Category 1 (CC1).
- ② Data satisfies the objectives of Control Category 2 (CC2).

Annex A

6-16

Just because Annex A Table A-2 indicates for Software Level D that Low-level requirements don't have to exist doesn't mean you shouldn't create them if they are needed for your project.

Can you do everything at Level A?

Yes. And there are **advantages**:

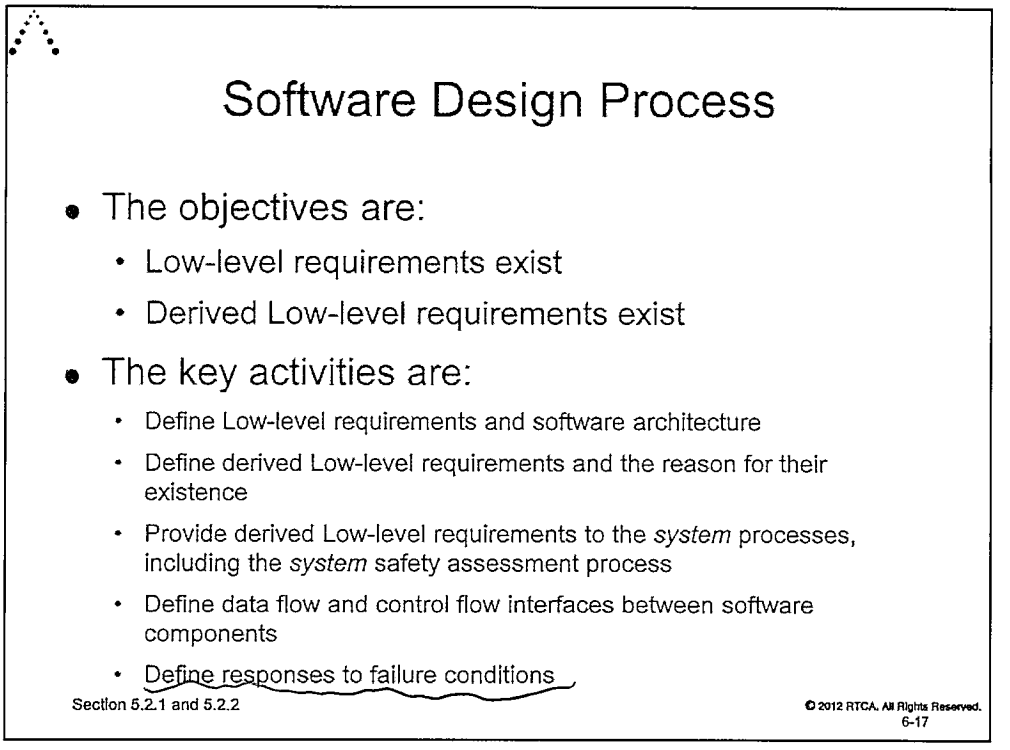
1) If your product later needs higher assurance, it is already done.

2) Level A software has been shown to require less maintenance after release.

Disadvantages might include:

1) Initial costs would be more

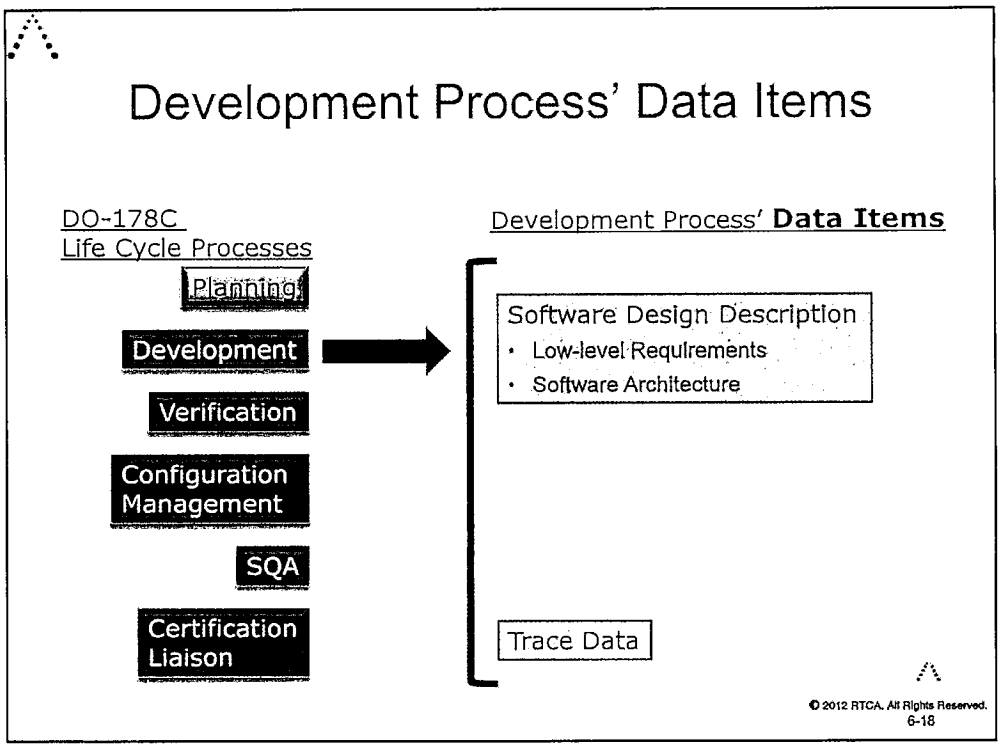
2) Longer development schedule would be required.



These objectives imply existence of two Software Life Cycle Data Items. These objectives are:

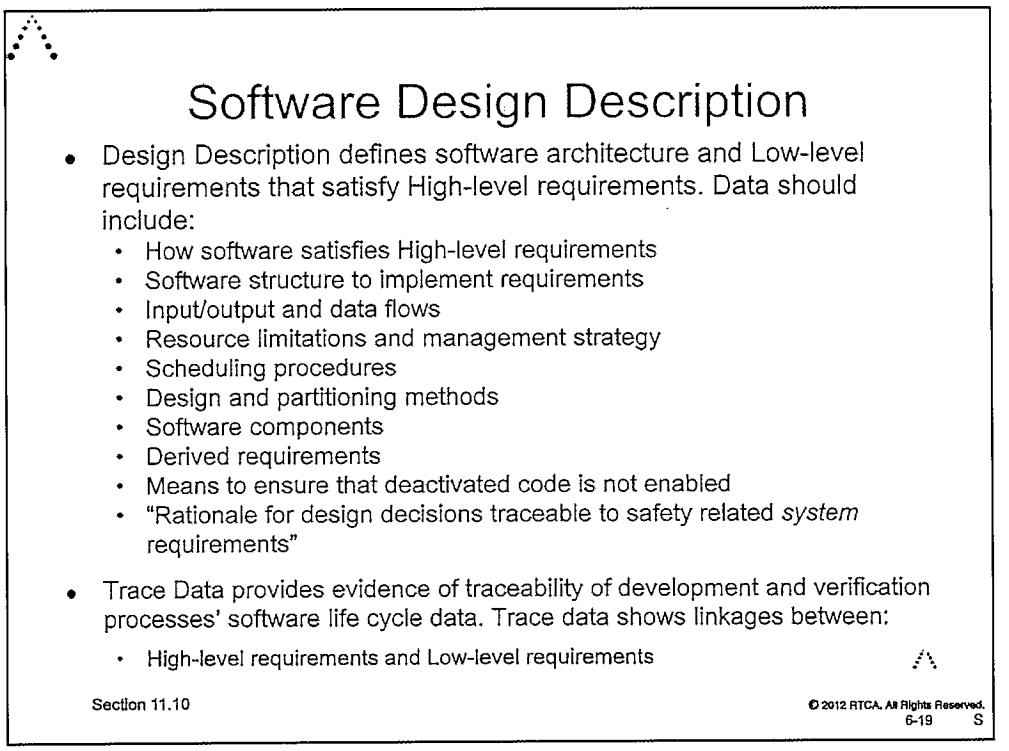
- Low-level requirements
- Derived Low-level requirements

They are both considered to be the "Design Description" of the Software Life Cycle Data Items.



Two Software Life Cycle Data items are created by this sub-process: (see Second Screen slide titled "Software Design Description")

- Software Design Description:
  - Low-level Requirements
  - Software Architecture
- Trace data



Software architecture is the structure of the software elements (for example tasks, procedures, functions, etc...) including inputs/outputs and calling /invocation methods for each software element.

Low-level requirements specify the behavior of each software element or small group of elements. This is not to imply that there must be a Low-level requirements for each element nor does it imply that a software element might not implement more than one Low-level requirement. Keep the requirements positive. DO NOT have a requirement that uses the phrase "shall not". For purposes of DO-178C, requirements should specify functionality.

Trace data will be discussed in the last "Software Development Process Traceability" charts at the end of this module. The information here is included for completeness.

The trace data specified in this slide may be created during the sub-process that creates the Design Description or it may be later in a separate sub-process.

Glossary

Definition

- Low-level Requirements – Software requirements developed from High-level requirements, derived requirements, and design constraints from which Source Code can be directly implemented without further information.

Glossary

Definition

- Software architecture – The structure of the software selected to implement the software requirements.

Exercise #2: Create Low-Level Requirements

- Create Low-level requirements for a function to display altitude, from pressure, on a secondary cockpit display.

Assumptions:

1. High-level requirements will be supplied.
2. A software architecture will be supplied.
3. A software function exists:
   - that can read pressure in PSI.
   - to output altitude in 100s of feet to a cockpit display.

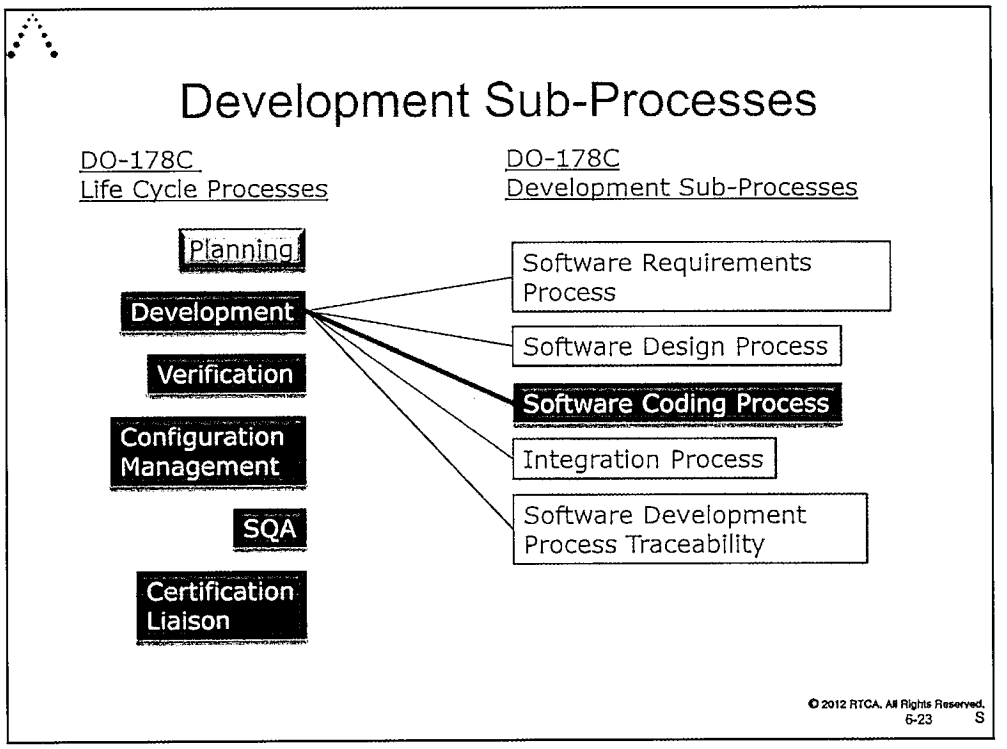
6-22

One answer is to use a look up table, example below, with interpolation between the bracketing Barometric Pressures.

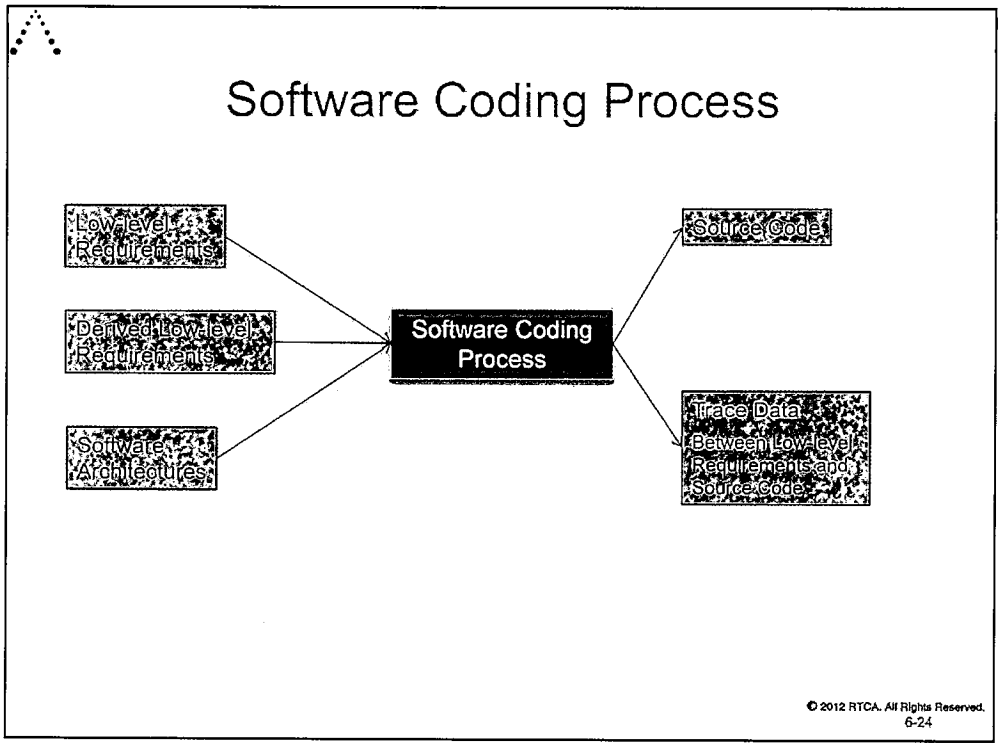
Example of Pressure vs. Altitude Table. (40 entries in full table)

Feet	PSI	
-5000	17.48	
-4500	17.19	
-4000	16.9	
-3500	16.62	
-3000	16.34	
-2500	16.06	
-2000	15.78	
-1500	15.51	
-1000	15.23	
-500	14.96	
0	14.696	
500	14.43	
1000	14.16	etc.

See exercise sheet for full information



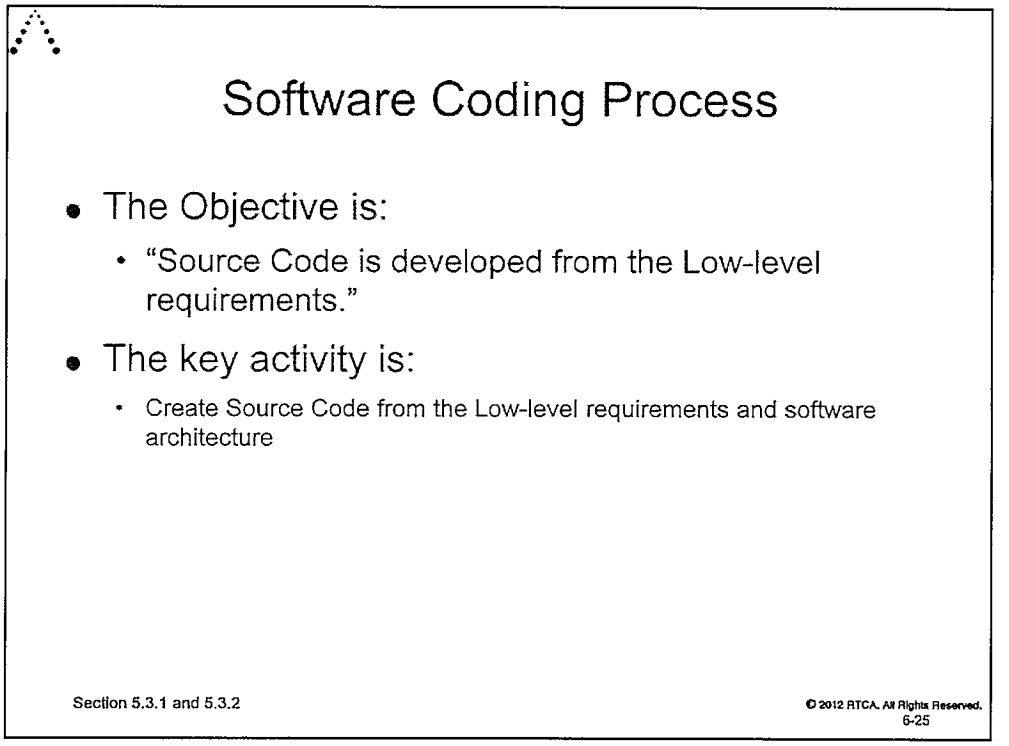
The DO-178C Software Development Process has 5 Sub-Processes; This session covers the Software Coding Process of the Development sub processes.



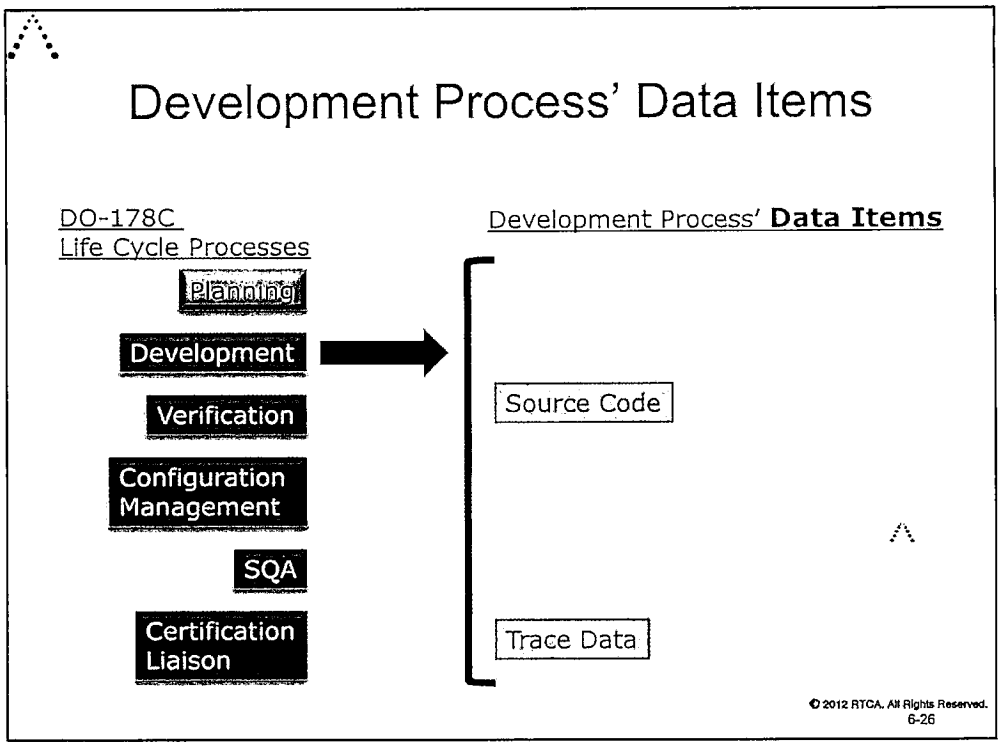
This sub-process produces the Source Code from the Low-level requirements within the software structure defined by the software architecture.

The Source code is a computer language that is complied/assembled in the next sub-process. Note: a complier/assembler may produce object code and a linker/loader may be needed to produce Executable Object Code.

DO-178C states: "The Source Code is used with the compiling, linking, and loading data in the integration process to develop the integrated *system* or equipment." The compiling, linking, and loading data maybe created in the coding sub-process or in the integration sub-process. It is not shown as an output of this sub-process as there is no specific Life Cycle Data Item for it, therefore it might be considered part of the Source Code data Item or the Executable Data Item.

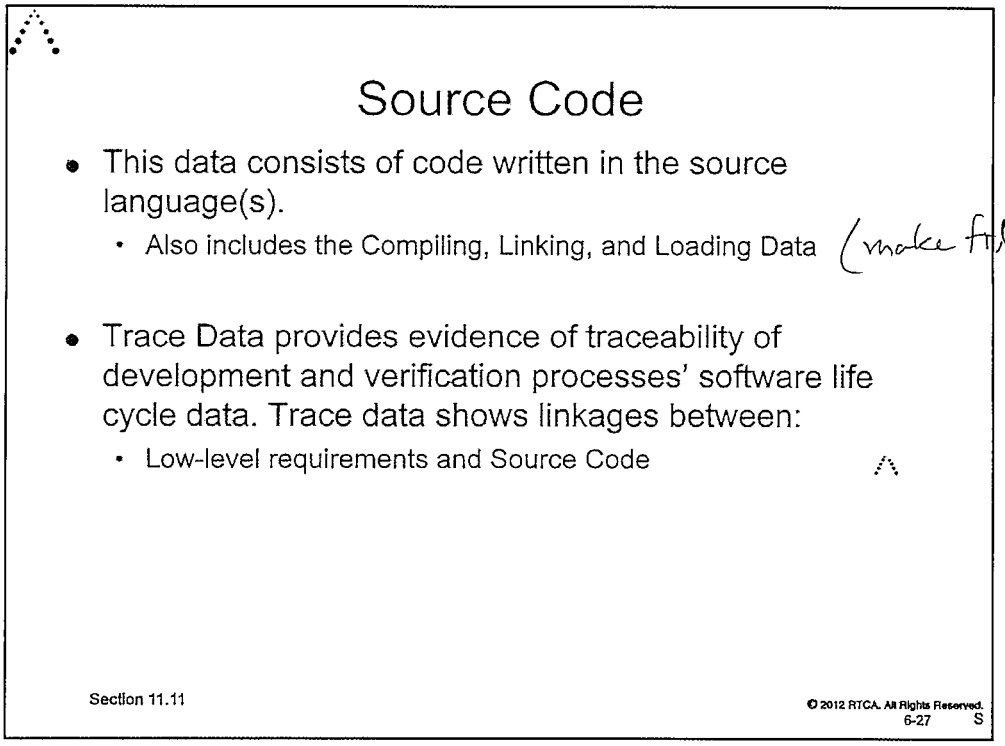


These objectives imply existence of one Software Life Cycle Data Item, Source Code.



Two Software Life Cycle Data items are created by this sub-process: (see Second Screen slide titled "Source Code")

- Source Code
- Trace data

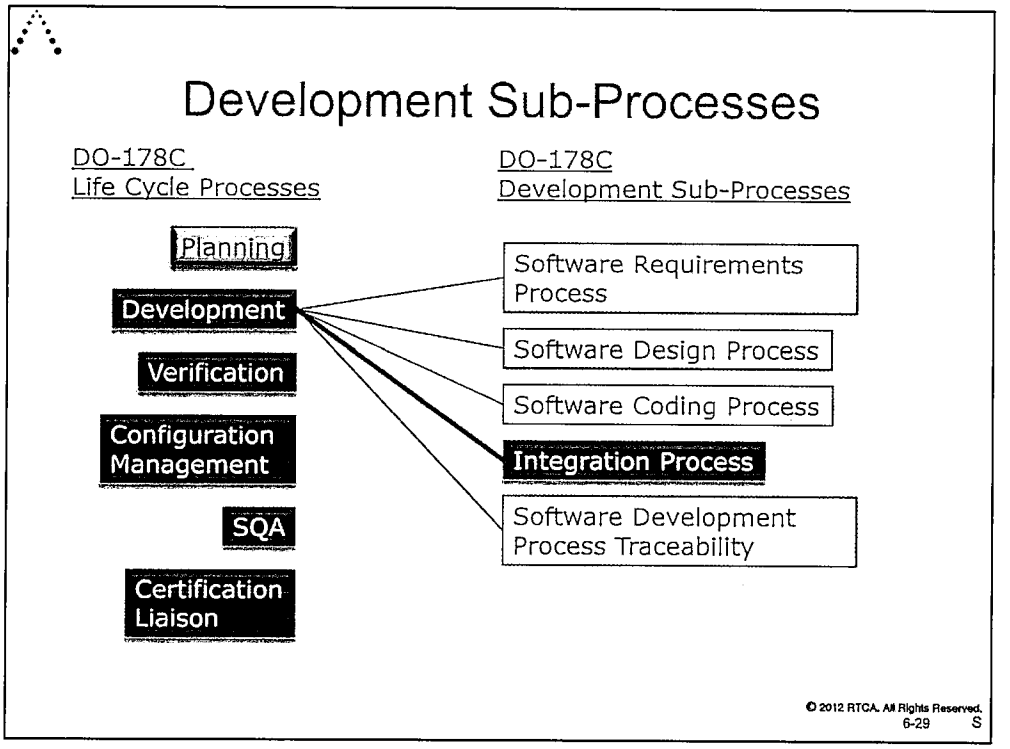


"The Source Code is used with the compiling, linking, and loading data in the integration process to develop the integrated *system* or equipment." The compiling, linking, and loading data maybe created in the coding sub-process or in the integration sub-process.

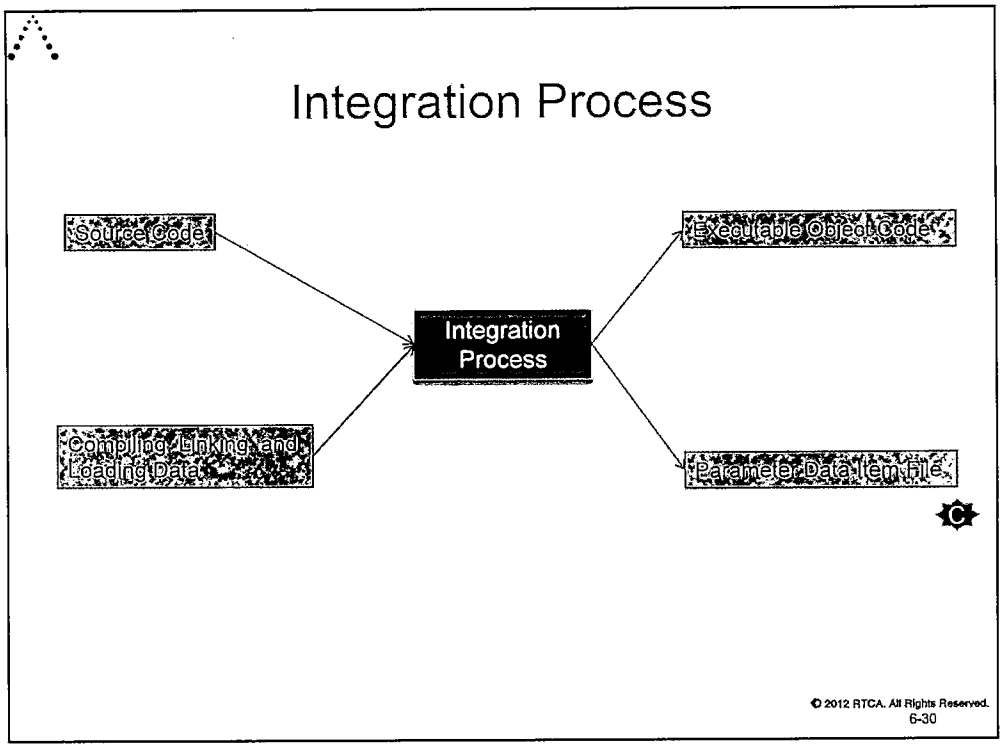
Glossary

Definition

- Source Code – Code written in source languages, such as assembly language and/or High-level language, in a machine-readable form for input to an assembler or a compiler.



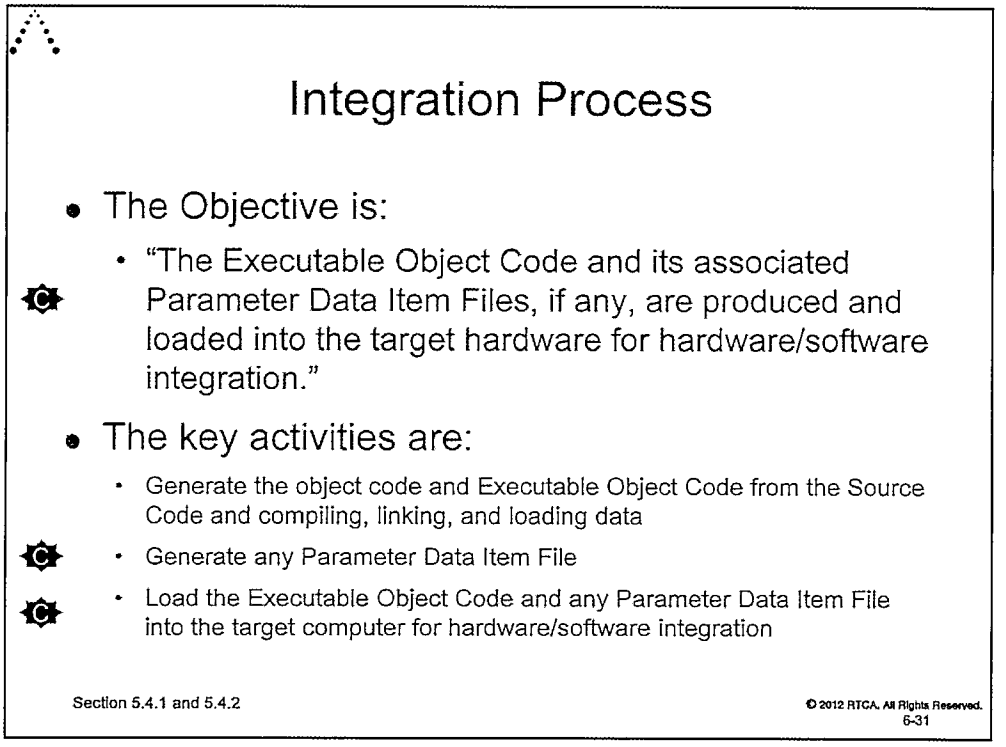
The DO-178C Software Development Process has 5 Sub-Processes; this section covers the Integration sub process.



This process brings all the software elements together into the Executable Object Code and Parameter Data Item File that will be used in the airborne *system* or equipment.

The Compiling, Linking, and Loading Data is created in this sub-process, if necessary and if not done in the coding sub-process, to tell the compiler/assembler and Linker/loader how to put all the software elements together.

The compiling, linking, and loading data maybe created in the coding sub-process or in the integration sub-process.

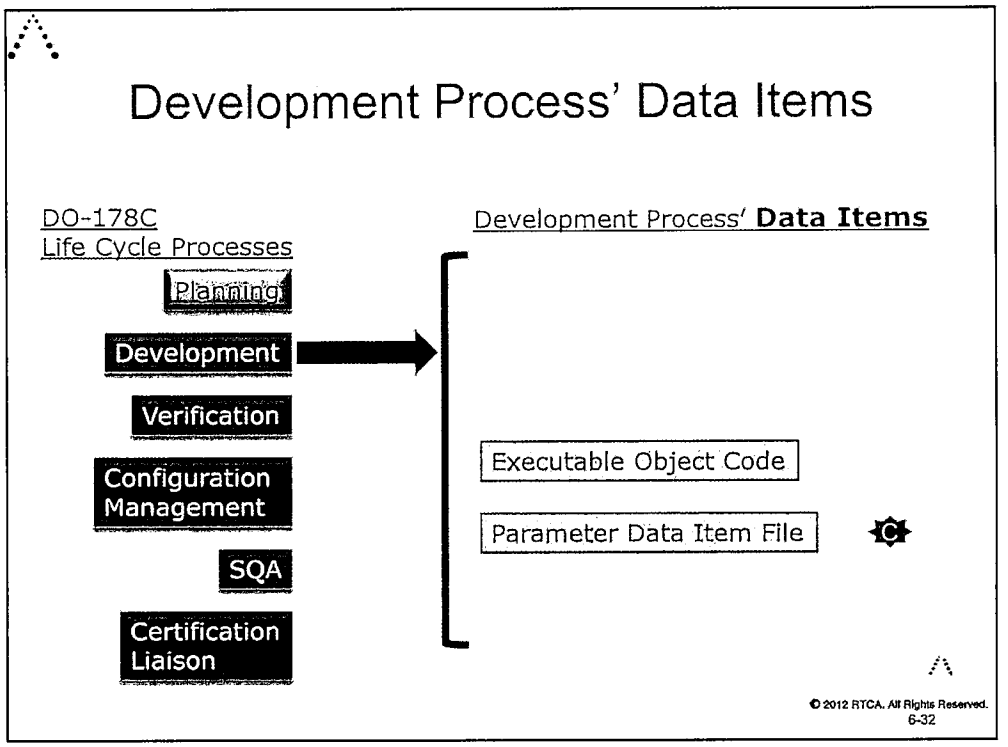


These objectives imply existence of two Software Life Cycle Data Items:

- Executable Object Code
- Parameter Data Item File

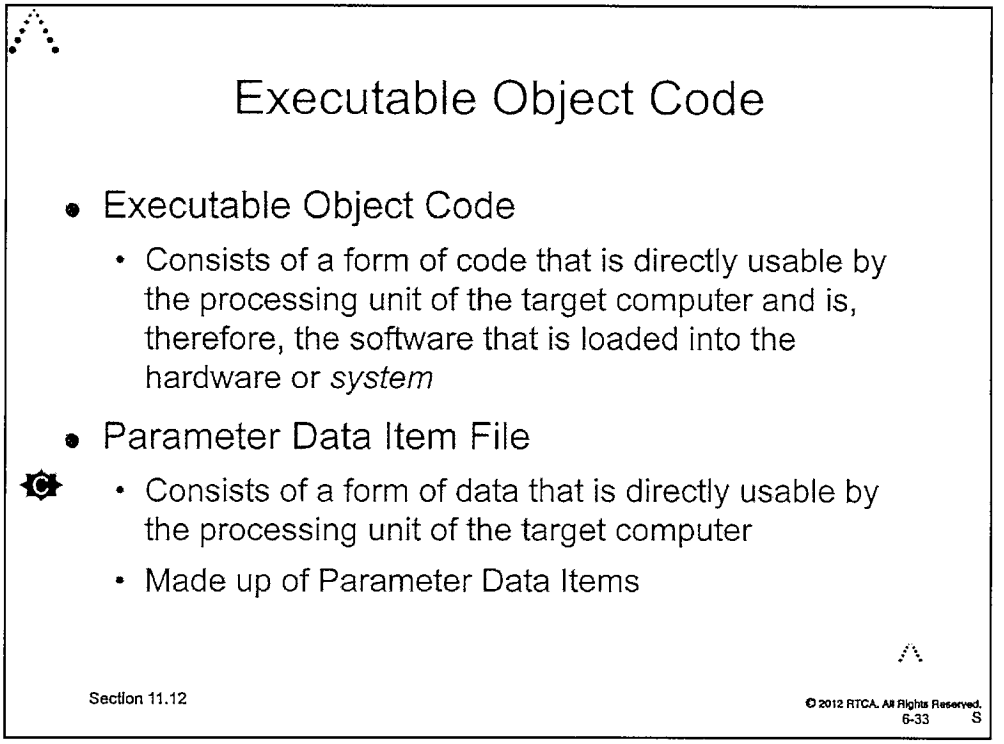
See Second Screen slide "Executable Object Code.

See Second Screen slide "Parameter Data Item File"



Two Software Life Cycle Data items are created by this sub-process (see Second Screen slide titled "Executable Object Code")

- Executable Object Code
- Parameter Data Item File will be dealt with later



Executable Object Code is the implementation of the *system*/software requirements that must be tested during the verification process (section 6.4). It ends up on the airplane.

Parameter Data Item File is verified as specified in DO-178C section 6.6. It ends up on the airplane associated with the Executable Object Code.

Glossary

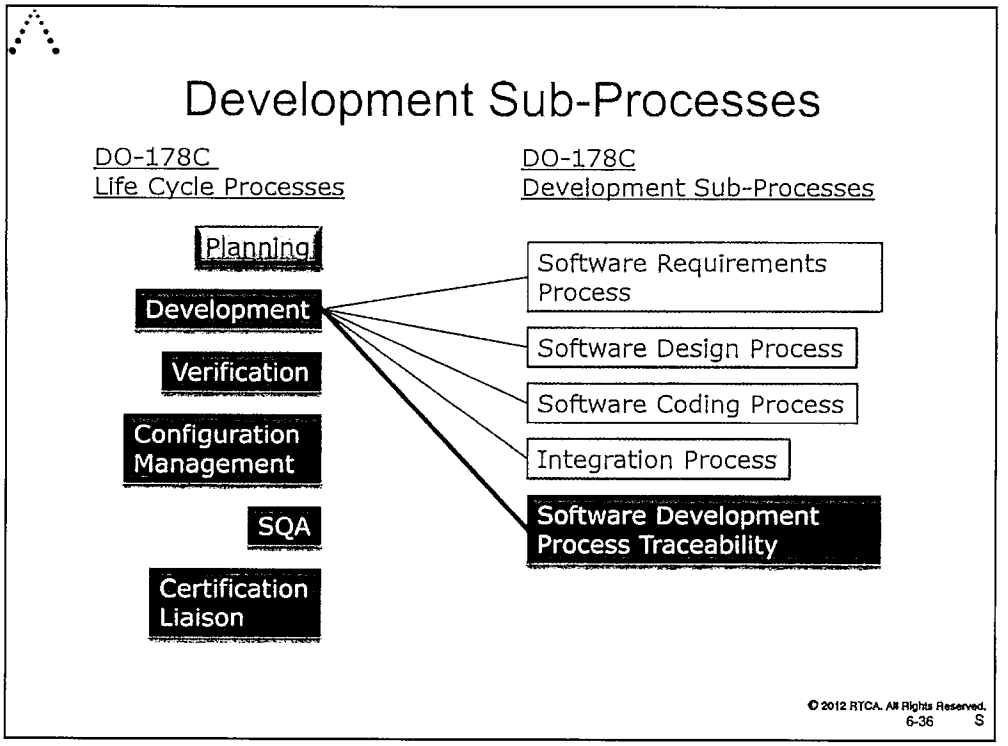
Definition

- Executable Object Code – A form of code that is directly usable by the processing unit of the target computer and is, therefore, a compiled, assembled, and linked binary image that is loaded into the target computing hardware.

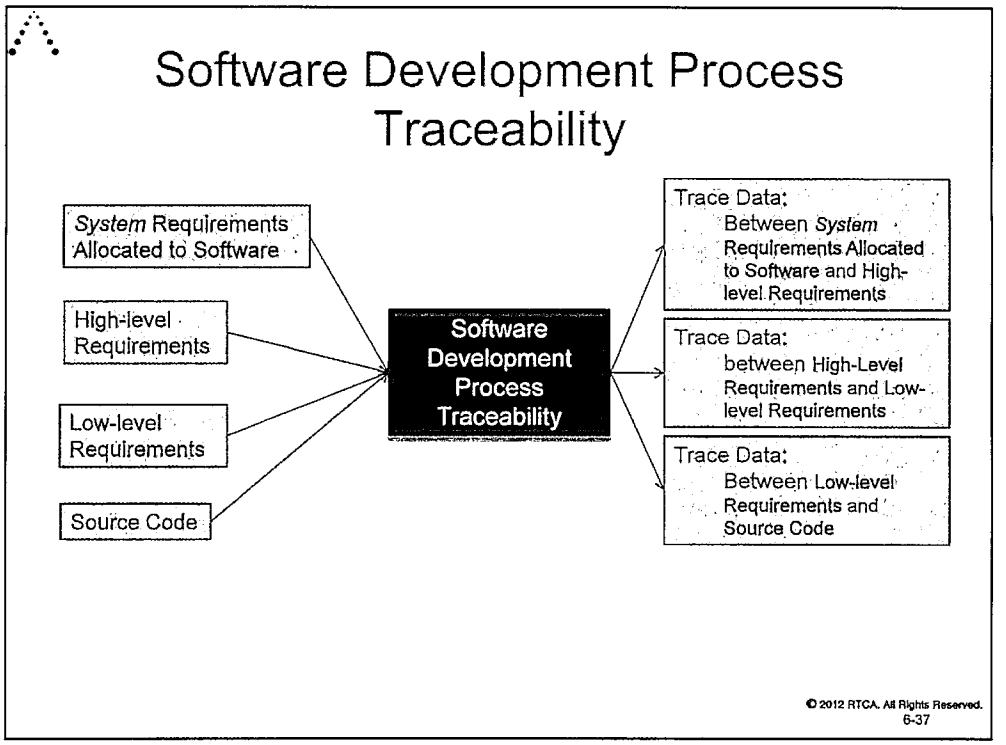
Glossary

Definition

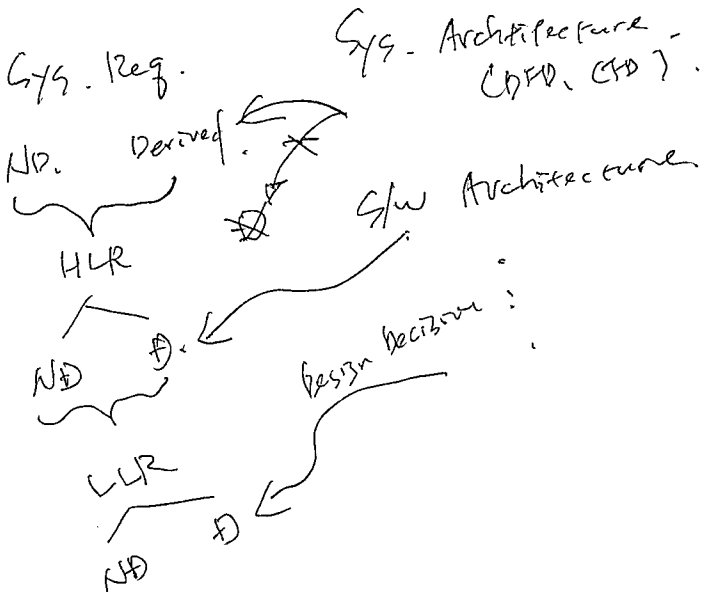
- Parameter Data Item – A set of data that, when in the form of a Parameter Data Item File, influence the behavior of the software without modifying the Executable Object Code and that is managed as a separate configuration item. Examples include databases and configuration tables.
- Parameter Data Item File – The representation of the parameter data item that is directly usable by the processing unit of the target computer. A Parameter Data Item File is an instantiation of the parameter data item containing defined values for each data element.

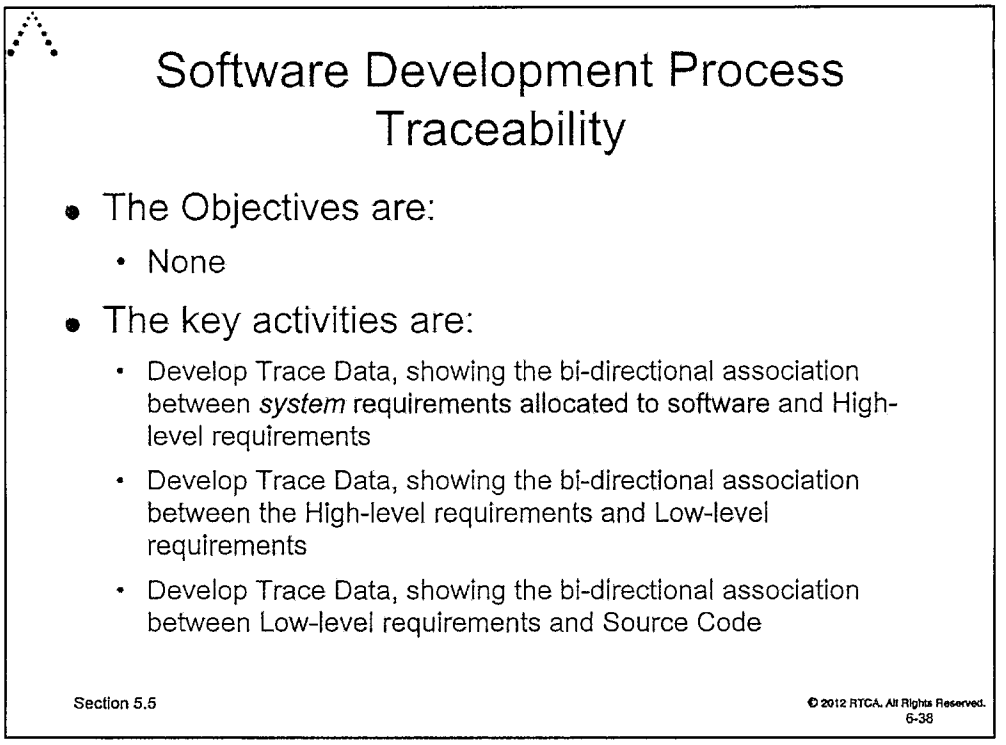


The DO-178C Software Development Process has 5 Sub-Processes; this section covers the software development sub process – Traceability.



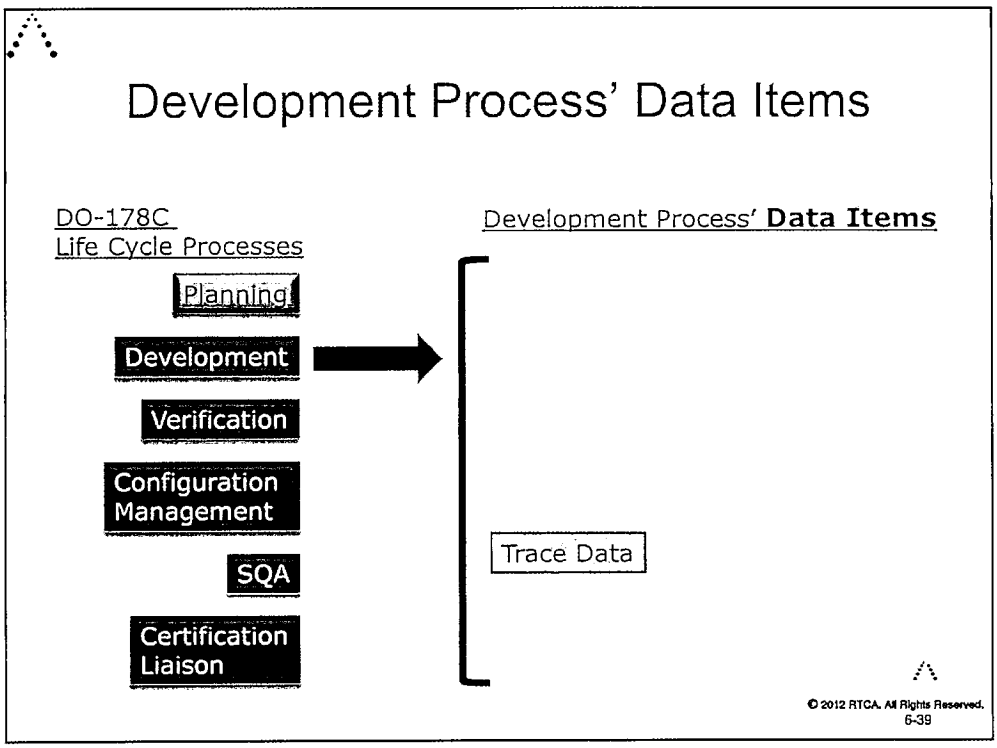
If trace data is not produced during the software requirements, software design and coding processes, then this process must be accomplished by this sub-process. It is up to the applicant's life-cycle definition.





Note: there are no objectives for this sub-process.

However, Trace Data is a Software Life Cycle Data Item and therefore must exist.



One Software Life Cycle Data items are created by this sub-process: (see Second Screen slide titled "Trace Data")

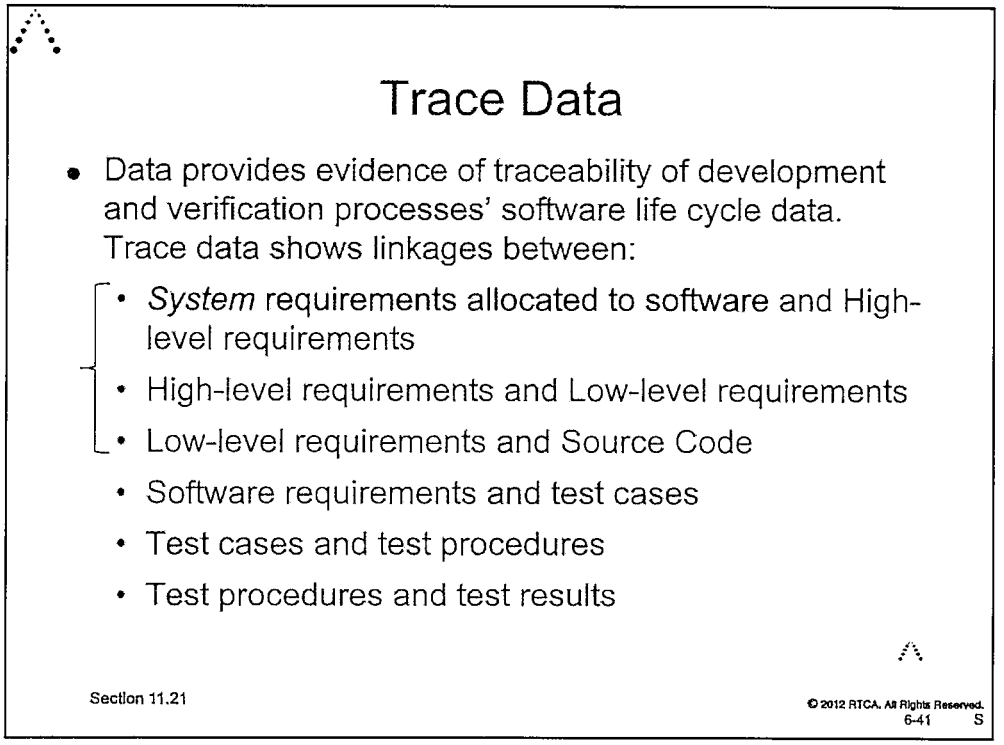
- Trace Data

There are three groupings of trace data for the Development process

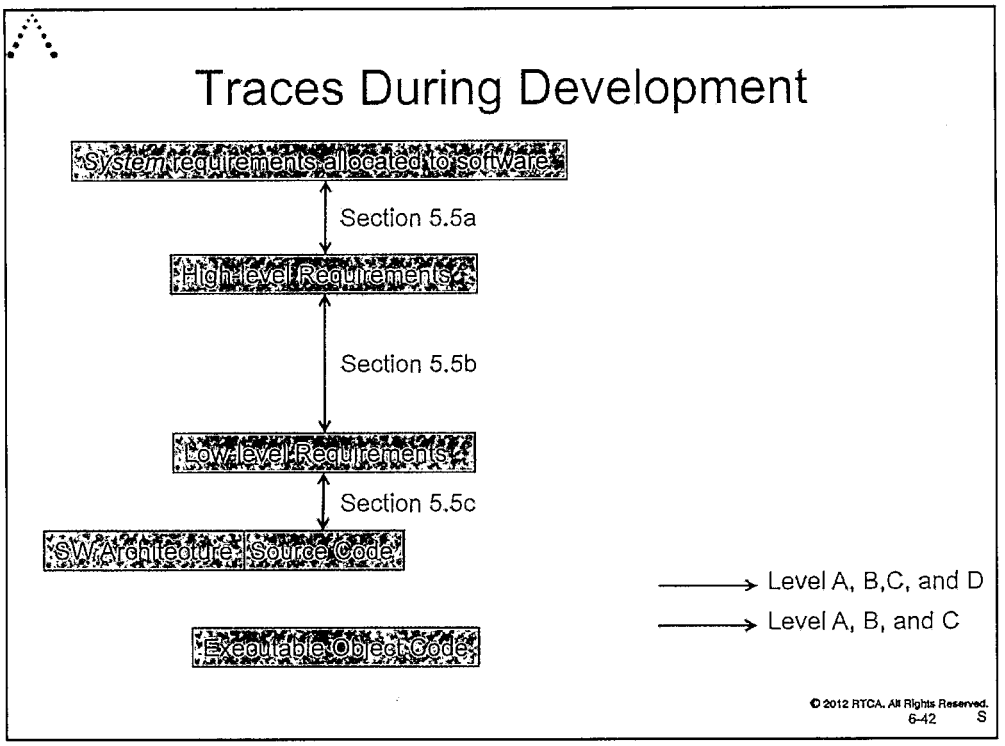
Glossary

Definition

- Trace data – Data providing evidence of traceability of development and verification processes' software life cycle data without implying the production of any particular artifact. Trace data may show linkages, for example, through the use of *naming conventions* or through the use of *references* or *pointers* either embedded in or external to the software life cycle data.

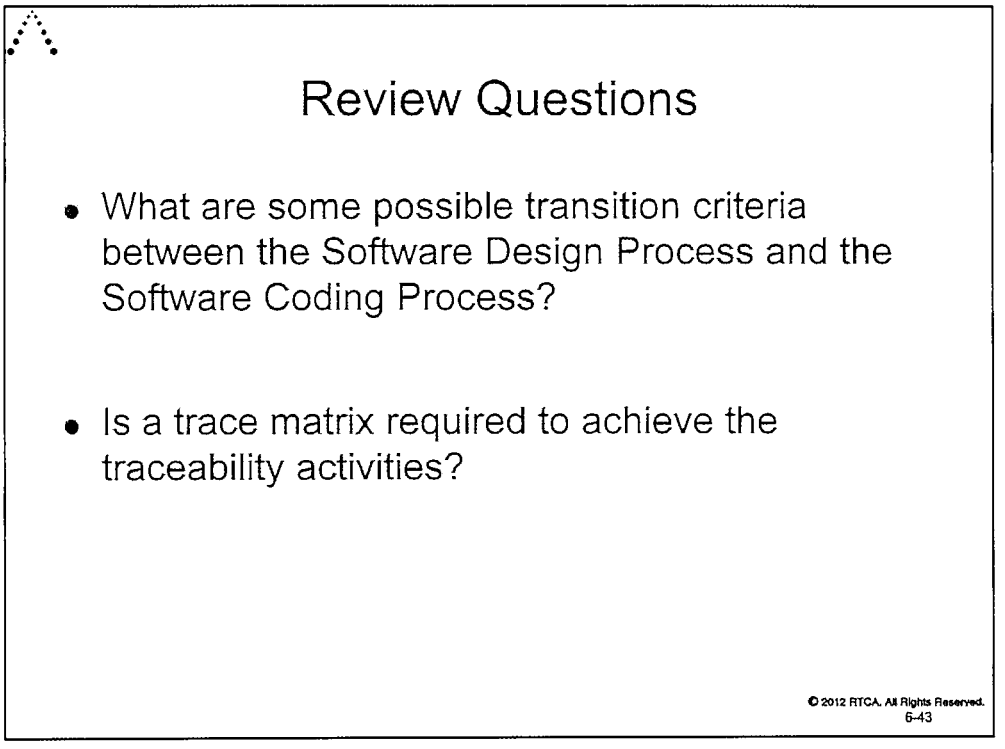


The first three bullets show the linkages between the various outputs of the development process.



Since Low-level requirements and Source Code are not required for Software Level D the trace data shown by the green arrows above would not be required. See the notes in slide "Annex A Table A-2 Low-Level Requirements" for more information.

If High-level are considered to be Low-level requirements then the green arrow labeled "Section 5.5b" is not required. However the green arrow label "Section 5.5c" would now link the High-level/Low-level requirements to Source Code. This a rare thing and would only occur on *systems* with very simple software. See the slides titled "Software Design Process" above for more information.



Question 1

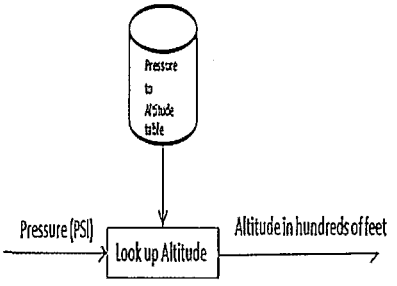
- All software design work is completed and Configuration Management identified
- All software design work is completed, Configuration Management identified, and verified by a review or analysis
- A Low-level requirement and its architecture is completed and configuration management identified
- A Low-level requirement and its architecture is completed, configuration management identified, and verified by a review or analysis
- A Low-level requirement and its architecture exists in some state
- None – the Source Code already exists and the Low-level requirements were created to document the portion of the High-level requirements that the source code implemented

Question 2

No. The definition of trace data allows the use of *naming conventions* or the use of *references* or *pointers* embedded in the software life cycle data.

Exercise #3: Coding

- Create Source Code for a function, in the language of your choice, that implements a table look up of altitude using pressure.

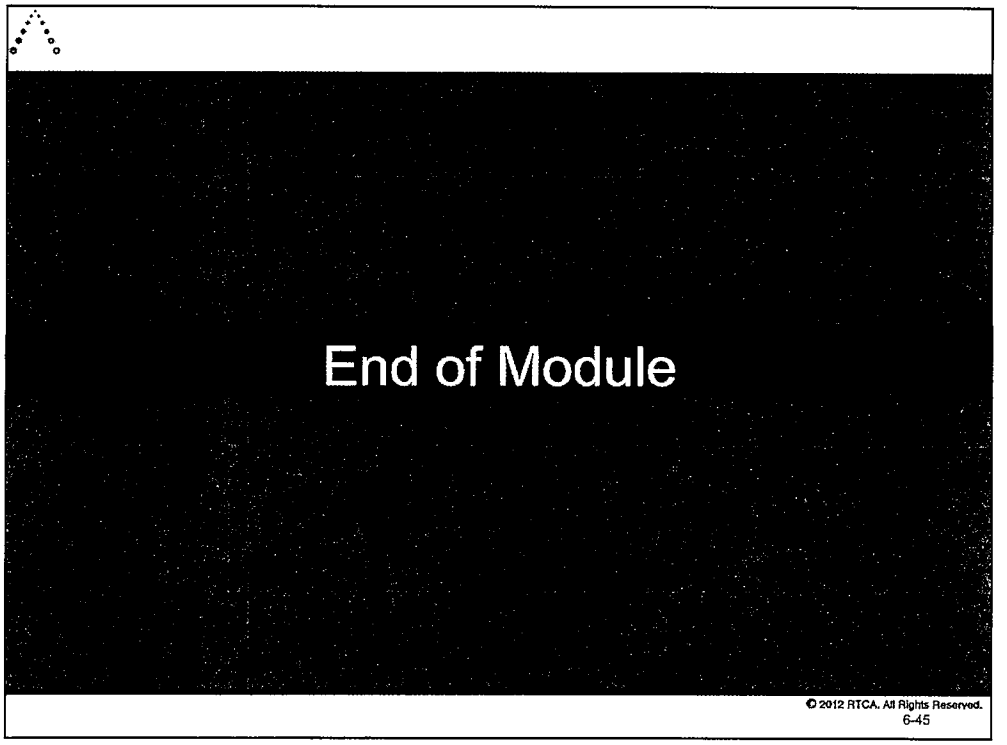


Assumptions:

1. Low-level requirements will be supplied. (Created in the exercise above) For example: Given a pound-force per square inch (PSI) pressure return an altitude in hundreds of feet
2. The software architecture is a single function that accepts a PSI input and returns an altitude in hundreds of feet.

6-44

Give interpolation algorithms and describe solution. Not all DO-178Cers are coders....



Exercise # 2
Low-level requirements

Create Low-level requirements for a function to display altitude, from pressure, on a secondary cockpit display.

Assumptions:

1. High-level requirements will be supplied.
2. A software architecture will be supplied.
3. A software function exists:
   - that can read pressure in PSI.
   - to output altitude in 100s of feet to a cockpit display.

High-Level Requirements V1.1

Software shall use the table below to determine altitude to the nearest 100 feet using PSI from the pressure sensor.

Barometric Pressure(PSI) vs. Altitude Table(feet). (40 entries)

Feet	*PSI*
-5000	17.48
-4500	17.19
-4000	16.9
-3500	16.62
-3000	16.34
-2500	16.06
-2000	15.78
-1500	15.51
-1000	15.23

-500	14.96
0	14.696
500	14.43
1000	14.16
1500	13.91
2000	13.66
2500	13.41
3000	13.17
3500	12.93
4000	12.69
4500	12.46
5000	12.23
6000	11.78
7000	11.34
8000	10.91
9000	10.5
10,000	10.1
15,000	8.29
20,000	6.76
25,000	5.46
30,000	4.37
35,000	3.47
40,000	2.73
45,000	2.15
50,000	1.69
55,000	1.33
60,000	1.05
70,000	0.651
80,000	0.406
90,000	0.255
100,000	0.162

Derived Requirements Discussion

This will be discussed in class.

The class will determine if any derived requirements are imbedded in Instructor's Low-level requirements.

Exercise # 3
Source Code

Create Source Code for a function, in the language of your choice, that implements a table look up of altitude using pressure.

Assumptions:

1. Low-level requirements will be supplied. (Created in the exercise above) For example: Given a pound-force per square inch (PSI) pressure return an altitude in hundreds of feet
2. The software architecture is a single function that accepts a PSI input and returns an altitude in hundreds of feet.