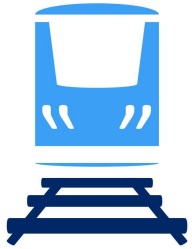
International standard IEC 61508-1



GICSAFe

GICSAFe

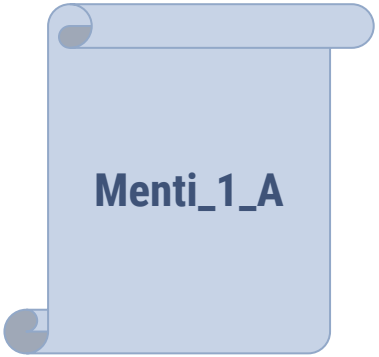
CONICET-GICSAFe

Grupo de Investigación y Control para la Seguridad y Aplicaciones Ferroviarias

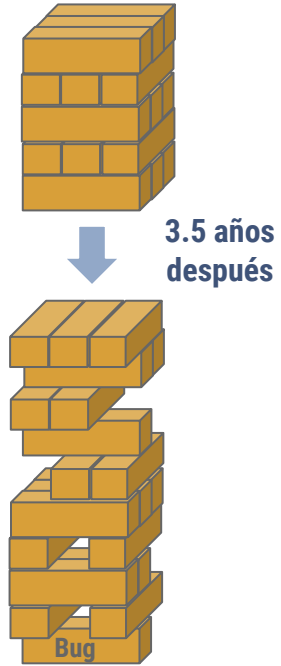
1

Calidad del código fuente

¿Por qué escribir buen código?



Cada 3.5 años se duplica el tamaño del código fuente



Received: 6 November 2015 | Revised: 2 December 2016 | Accepted: 22 December 2016

DOI 10.1002/smr.1847

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE

The long-term growth rate of evolving software: Empirical results and implications

Les Hatton[1] | Diomidis Spinellis[2] | Michiel van Genuchten[3]

[1]Kingston University, Kingston upon Thames, UK
[2]Athens University of Economics and Business, Patision 76, GR-104 34 Athens Greece
[3]VitalHealth Software, Ede, The Netherlands

Correspondence
Diomidis Spinellis, Athens University of Economics and Business, Patision 76, GR-104 34 Athens, Greece.
Email: dds@aueb.gr

Abstract

The amount of code in evolving software-intensive systems appears to be growing relentlessly, affecting products and entire businesses. Objective figures quantifying the software code growth rate bounds in systems over a large time scale can be used as a reliable predictive basis for the size of software assets. We analyze a reference base of over 404 million lines of open source and closed software systems to provide accurate bounds on source code growth rates. We find that software source code in systems doubles about every 42 months on average, corresponding to a median compound annual growth rate of 1.21 ± 0.01. Software product and development managers can use our findings to bound estimates, to assess the trustworthiness of road maps, to

Cada 3.5 años se duplica el tamaño del código fuente

Received: 6 November 2015 | Revised: 2 December 2016 | Accepted: 22 December 2016

DOI 10.1002/smr.1847

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE

The long-term growth rate of evolving software: Empirical results and implications

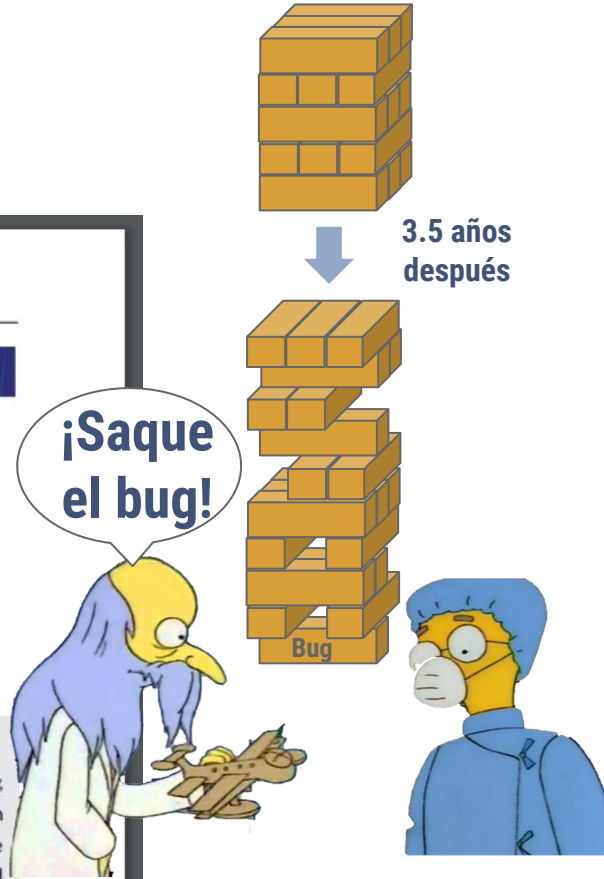
Les Hatton[1] | Diomidis Spinellis[2] | Michiel van Genuchten[3]

[1]Kingston University, Kingston upon Thames, UK
[2]Athens University of Economics and Business, Patision 76, GR-104 34 Athens Greece
[3]VitalHealth Software, Ede, The Netherlands

Correspondence
Diomidis Spinellis, Athens University of Economics and Business, Patision 76, GR-104 34 Athens, Greece.
Email: dds@aueb.gr

Abstract

The amount of code in evolving software-intensive systems appears to be growing relentlessly, affecting products and entire businesses. Objective figures quantifying the software code growth rate bounds in systems over a large time scale can be used as a reliable predictive basis for the size of software assets. We analyze a reference base of over 404 million lines of open source and closed software systems to provide accurate bounds on source code growth rates. We find that software source code in systems doubles about every 42 months on average, corresponding to a median compound annual growth rate of 1.21 ± 0.01. Software product and development managers can use our findings to bound estimates, to assess the trustworthiness of road maps, to



Cada 3.5 años se duplica el tamaño del código fuente

Received: 6 November 2015 | Revised: 2 December 2016 | Accepted: 22 December 2016

DOI 10.1002/smr.1847

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE

The long-term growth rate of evolving software: Empirical results and implications

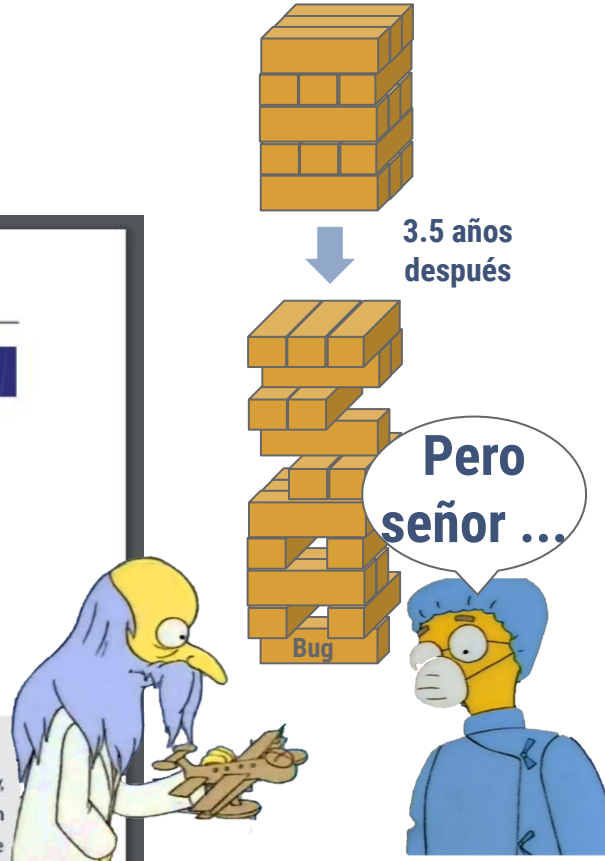
Les Hatton[1] | Diomidis Spinellis[2] | Michiel van Genuchten[3]

[1]Kingston University, Kingston upon Thames, UK
[2]Athens University of Economics and Business, Patision 76, GR-104 34 Athens Greece
[3]VitalHealth Software, Ede, The Netherlands

Correspondence
Diomidis Spinellis, Athens University of Economics and Business, Patision 76, GR-104 34 Athens, Greece.
Email: dds@aueb.gr

Abstract

The amount of code in evolving software-intensive systems appears to be growing relentlessly, affecting products and entire businesses. Objective figures quantifying the software code growth rate bounds in systems over a large time scale can be used as a reliable predictive basis for the size of software assets. We analyze a reference base of over 404 million lines of open source and closed software systems to provide accurate bounds on source code growth rates. We find that software source code in systems doubles about every 42 months on average, corresponding to a median compound annual growth rate of 1.21 ± 0.01. Software product and development managers can use our findings to bound estimates, to assess the trustworthiness of road maps, to



Cada 3.5 años se duplica el tamaño del código fuente

Received: 6 November 2015 | Revised: 2 December 2016 | Accepted: 22 December 2016

DOI 10.1002/smr.1847

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE

The long-term growth rate of evolving software: Empirical results and implications

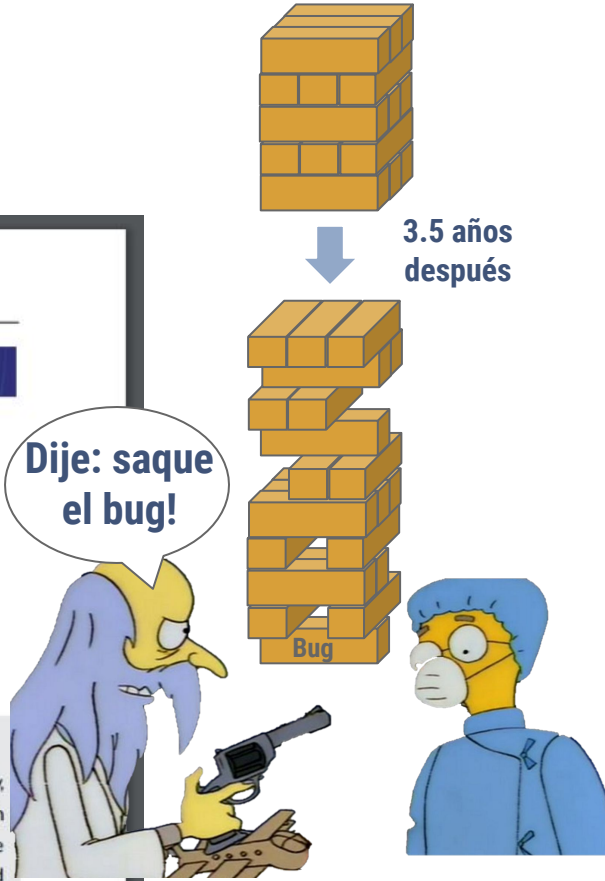
Les Hatton[1] | Diomidis Spinellis[2] | Michiel van Genuchten[3]

[1]Kingston University, Kingston upon Thames, UK
[2]Athens University of Economics and Business, Patision 76, GR-104 34 Athens Greece
[3]VitalHealth Software, Ede, The Netherlands

Correspondence
Diomidis Spinellis, Athens University of Economics and Business, Patision 76, GR-104 34 Athens, Greece.
Email: dds@aueb.gr

Abstract

The amount of code in evolving software-intensive systems appears to be growing relentlessly, affecting products and entire businesses. Objective figures quantifying the software code growth rate bounds in systems over a large time scale can be used as a reliable predictive basis for the size of software assets. We analyze a reference base of over 404 million lines of open source and closed software systems to provide accurate bounds on source code growth rates. We find that software source code in systems doubles about every 42 months on average, corresponding to a median compound annual growth rate of 1.21 ± 0.01. Software product and development managers can use our findings to bound estimates, to assess the trustworthiness of road maps, to



¿Cómo los programadores leemos un código?

2015 IEEE 23rd International Conference on Program Comprehension

How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking

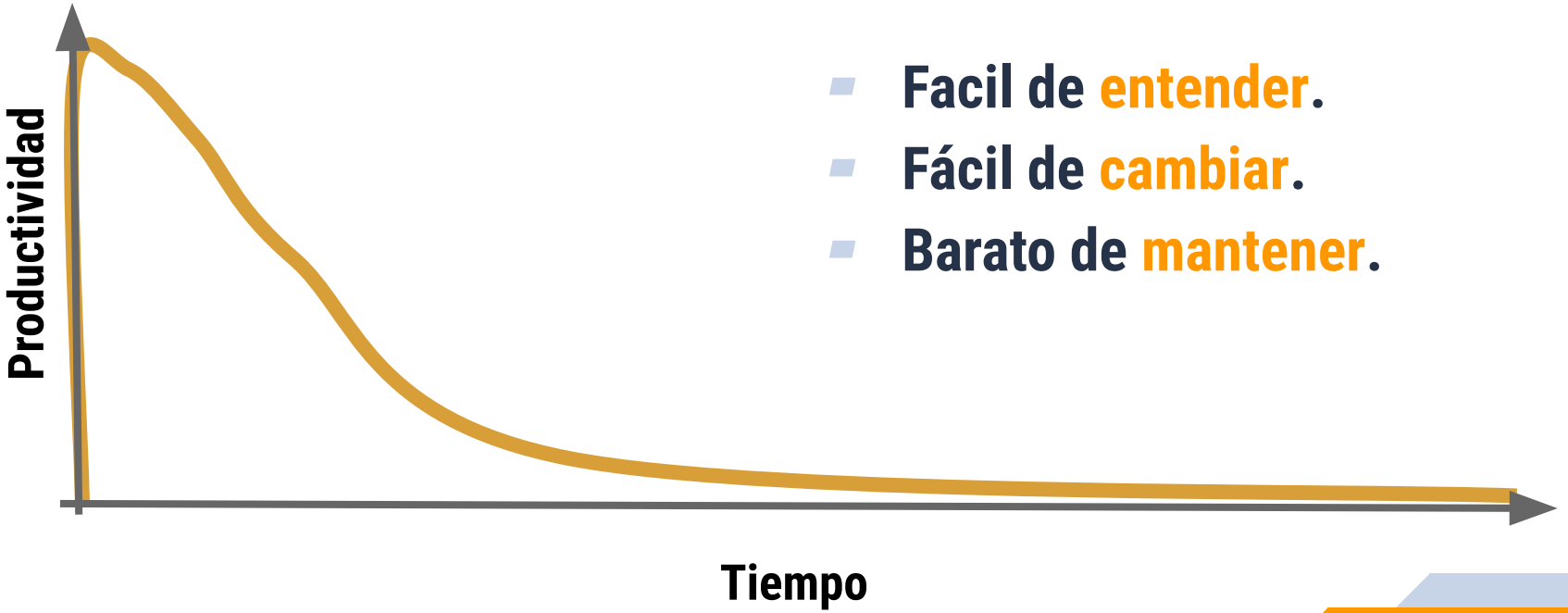
Ahmad Jbara[1,2] Dror G. Feitelson[2]
[1]School of Mathematics and Computer Science
Netanya Academic College, 42100 Netanya, Israel
[2]School of Computer Science and Engineering
The Hebrew University of Jerusalem, 91904 Jerusalem, Israel

***Abstract*—Regular code, which includes repetitions of the same basic pattern, has been shown to have an effect on code comprehension: a regular function can be just as easy to comprehend as an irregular one with the same functionality, despite being longer and including more control constructs. It has been speculated that this effect is due to leveraging the understanding of the first instances to ease the understanding of repeated instances of the pattern. To verify and quantify this effect, we use eye tracking** regular code, and to quantitatively measure the time and effort invested in the successive repetitions of such a code. The results indeed show that the time and effort are reduced as subjects progress from one repeated instance of a pattern to the next. This reduction can be modeled by an exponential or a cubic function.

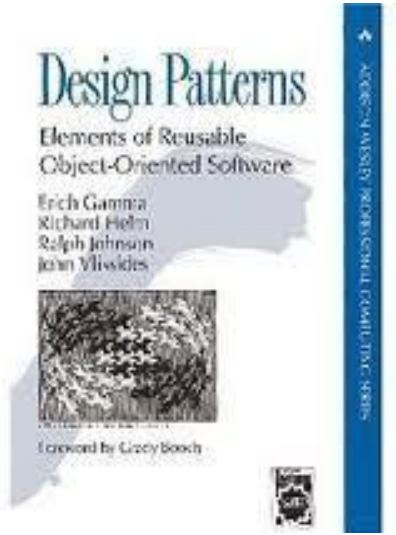
The consequence is that additive syntax-based metrics like

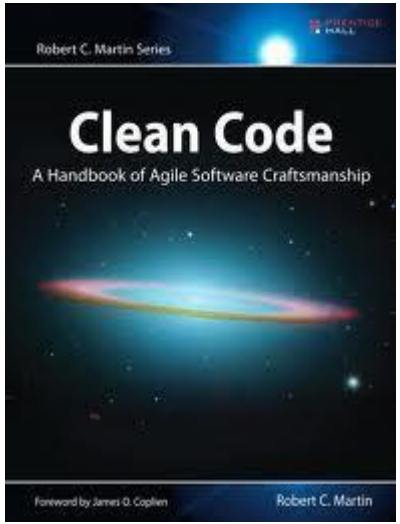
Son necesarias **métricas de complejidad** que tengan en cuenta el cambio en la **repetición** del código fuente

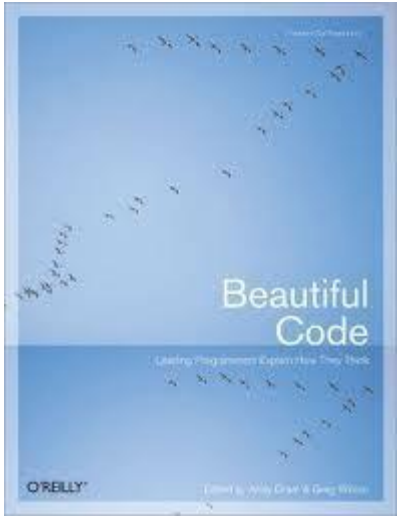
Costo de poseer código “no mantenible”

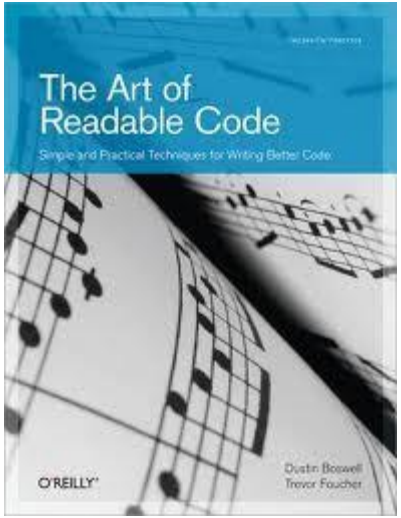


- Facil de entender.
- Fácil de cambiar.
- Barato de mantener.


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


Beautiful Code
O'REILLY


The Art of Readable Code
Simple and Practical Techniques for Writing Better Code
O'REILLY
Dustin Boswell
Trevor Foucher

Y muchos más…!

2

IEC 61508

Estándar de seguridad funcional

¿Cuando aplicamos la IEC 61508?

La norma los define como E/E/PE

- Si el proyecto involucra el desarrollo de sistemas electrónicos, eléctricos, o de electrónica programable que involucre **seguridad**.
- Si en otro estándar que apliques en tu proyecto la **seguridad** sea un factor relevante.

Seguridad funcional

Considerar al sistema COMPLETO y a su ENTORNO

Seguro: Ausencia de niveles inaceptables de riesgo que puedan provocar lesiones daños a la salud de las personas, ya sea directa o indirectamente.

(Aislamiento para soportar altas temperaturas)

Seguridad funcional: es parte de la seguridad general que depende de que un sistema o equipo funcione correctamente en respuesta a sus entradas

(Un dispositivo de protección contra sobrecalentamiento)

Objetivo

- Aumentar la **seguridad** y **eficiencia económica** de los sistemas E/E/PE.
- Brindar un marco general de seguridad para los desarrollos tecnológicos.
- Establecer un **enfoque flexible** basado en sistemas técnicamente sólido.
- Definir un **estándar** que pueda ser utilizado directamente por la industria.
- Mediar entre los usuarios y los reguladores para que ganen confianza en la tecnología
- Proporcionar un marco común de requisitos para facilitar:
  - Mejoras en la eficiencia en la cadena de suministro de componentes.
  - Mejoras en la comunicación de los requisitos.
  - El desarrollo de técnicas y medidas que puedan utilizarse.
  - El desarrollo de servicios de evaluación de la conformidad si es necesario.

Complejos, muy difíciles de determinar TODOS los modos de fallas

Fuente de los errores peligrosos:

Complejos, muy difíciles de determinar TODOS los modos de fallas

Fuente de los errores peligrosos:

Malas especificaciones

Errores aleatorios de hardware

Errores sistemáticos de hardware

Errores de software

Errores de causa común

Errores humanos

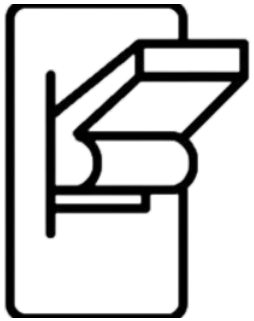
Influencias ambientales

Fuente de alimentación

Ejemplo

- Sistemas de apagado de **emergencia**.
- Señalamiento/Enclavamiento ferroviario.
- Plantas nucleares.
- Sistemas aeroespaciales (aeronaves, satélites, etc).
- Control del movimiento de un barco cuando está cerca de una instalación en alta mar.
- Luces indicadoras de vehículos, frenos **antibloqueo** y sistemas de gestión del motor

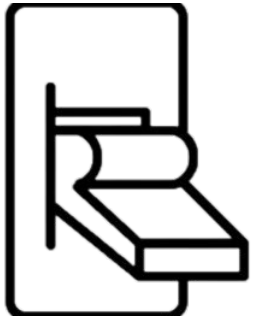
Sistemas críticos



Ejemplo

- Sistemas de apagado de **emergencia**.
- Señalamiento/Enclavamiento ferroviario.
- Plantas nucleares.
- Sistemas aeroespaciales (aeronaves, satélites, etc).
- Control del movimiento de un barco cuando está cerca de una instalación en alta mar.
- Luces indicadoras de vehículos, frenos **antibloqueo** y sistemas de gestión del motor

Sistemas críticos



Ejemplo

- Sistemas de apagado de **emergencia**.
- Señalamiento/Enclavamiento ferroviario.
- Plantas nucleares.
- Sistemas aeroespaciales (aeronaves, satélites, etc).
- Control del movimiento de un barco cuando está cerca de una instalación en alta mar.
- Luces indicadoras de vehículos, frenos **antibloqueo** y sistemas de gestión del motor

Nivel integral de seguridad

Cada nivel es más riguroso que el anterior

Severidad de la consecuencia

Frecuencia

	1	2	3	4	5
5	SIL3	SIL4	X	X	X
4	SIL2	SIL3	SIL4	X	X
3	SIL1	SIL2	SIL3	SIL4	X
2	X	SIL1	SIL2	SIL3	SIL4
1	X	X	SIL1	SIL2	SIL3

Menti_2_A

Si las partes deben cumplir diferentes SILs, todos deben cumplir el de mayor SIL

Niveles

PFD (Probabilidad de falla bajo demanda)	**Fallo máximo aceptado**	**PFH** (Probabilidad de falla peligrosa por hora)	**Fallo máximo aceptado**
10^0 > **PFD** >= ?	1 cada **?** años	?? > **PFH** >= ??	1 cada **?** horas

SIL 0

Niveles

PFD (Probabilidad de falla bajo demanda)	**Fallo máximo aceptado**	**PFH** (Probabilidad de falla peligrosa por hora)	**Fallo máximo aceptado**
10^-1 > **PFD** >= 10^-2	1 cada **10** años	10^-5 > **PFH** >= 10^-6	1 cada **100.000** horas

SIL 1

Niveles

PFD (Probabilidad de falla bajo demanda)	**Fallo máximo aceptado**	**PFH** (Probabilidad de falla peligrosa por hora)	**Fallo máximo aceptado**
10^-2 > **PFD** >= 10^-3	1 cada **100** años	10^-6 > **PFH** >= 10^-7	1 cada **1.000.000** horas

SIL 2

Niveles

PFD (Probabilidad de falla bajo demanda)	**Fallo máximo aceptado**	**PFH** (Probabilidad de falla peligrosa por hora)	**Fallo máximo aceptado**
10^-3 > **PFD** >= 10^-4	1 cada **1.000** años	10^-7 > **PFH** >= 10^-8	1 cada **10.000.000** horas

SIL 3

Niveles

PFD (Probabilidad de falla bajo demanda)	**Fallo máximo aceptado**	**PFH** (Probabilidad de falla peligrosa por hora)	**Fallo máximo aceptado**
10^-4 > **PFD** >= 10^-5	1 cada **10.000** años	10^-8 > **PFH** >= 10^-9	1 cada **100.000.000** horas

SIL 4

Partes de la norma

Consta de SIETE partes!

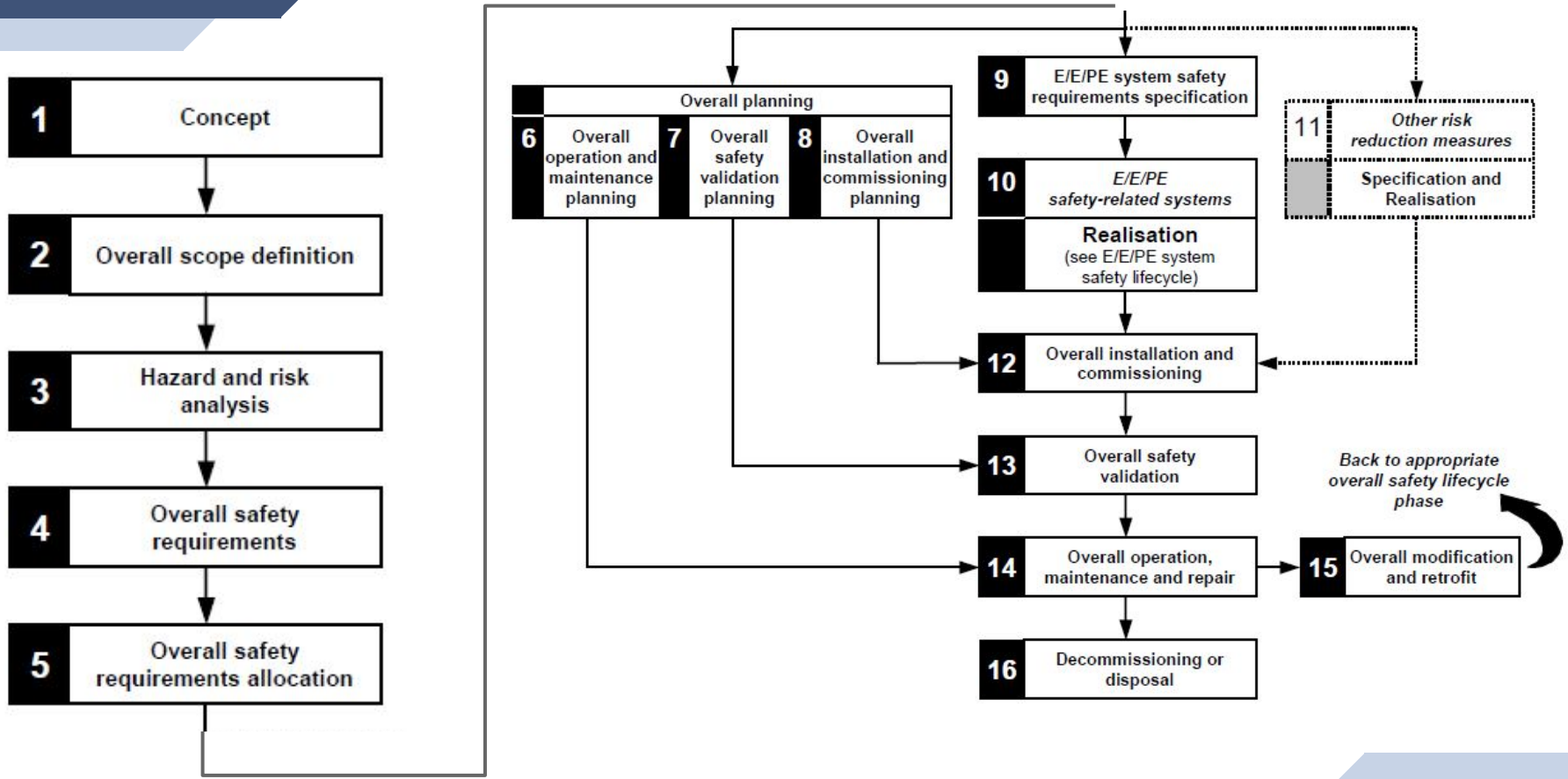
1. **IEC 61508-1**: Requerimientos generales.
2. **IEC 61508-2**: Requerimientos **E/E/PE**.
3. **IEC 61508-3**: Requerimientos de **software**.
4. **IEC 61508-4**: Definiciones y abreviaturas.
5. **IEC 61508-5**: Ejemplos de **cálculo de SIL**.
6. **IEC 61508-6**: **Guías para aplicar** IEC 61508-2 y IEC 61508-3.
7. **IEC 61508-7**: Mediciones y técnicas.

3

IEC 61508-1

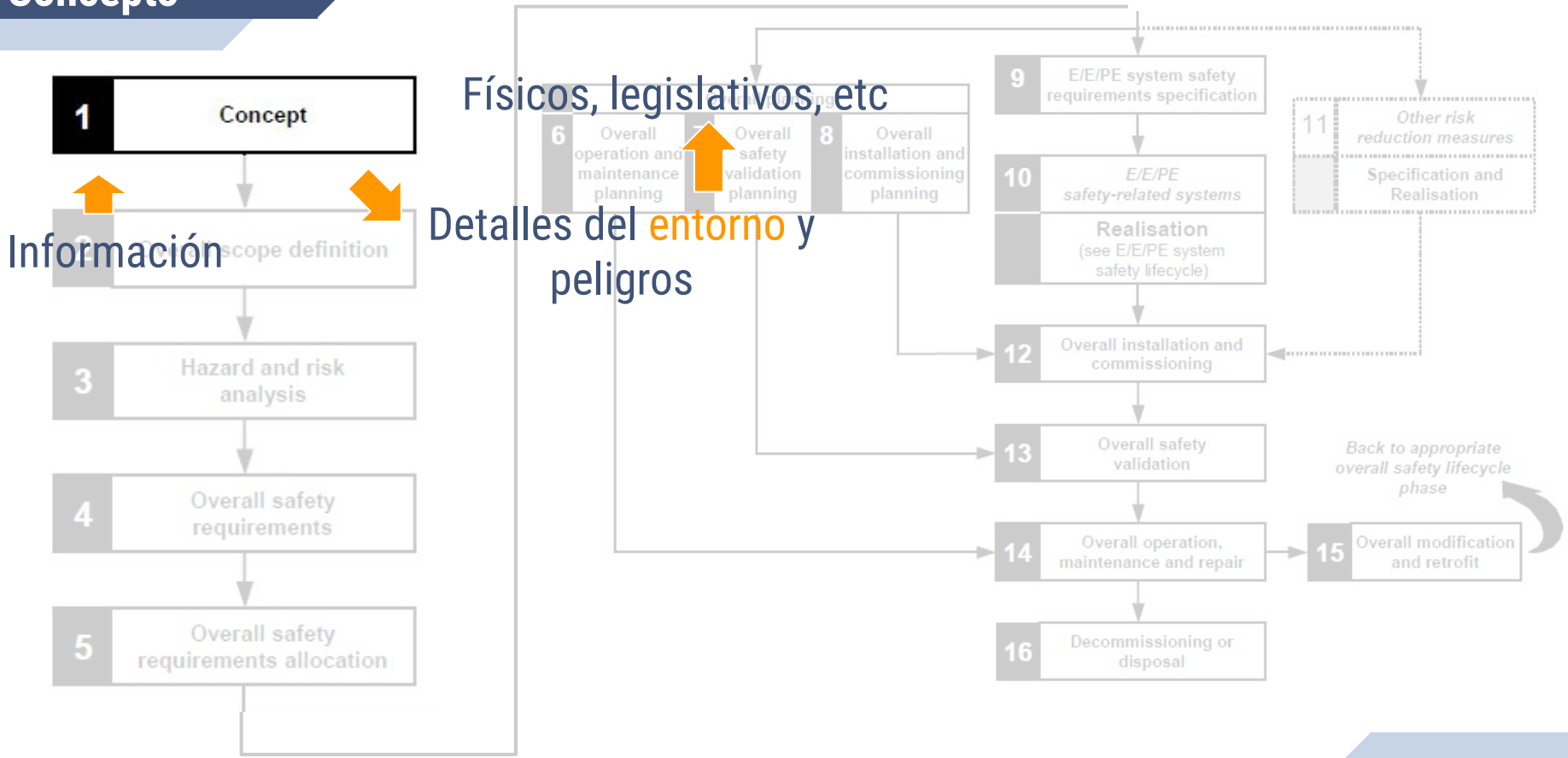
IEC 61508-1

Ciclo de vida de seguridad general

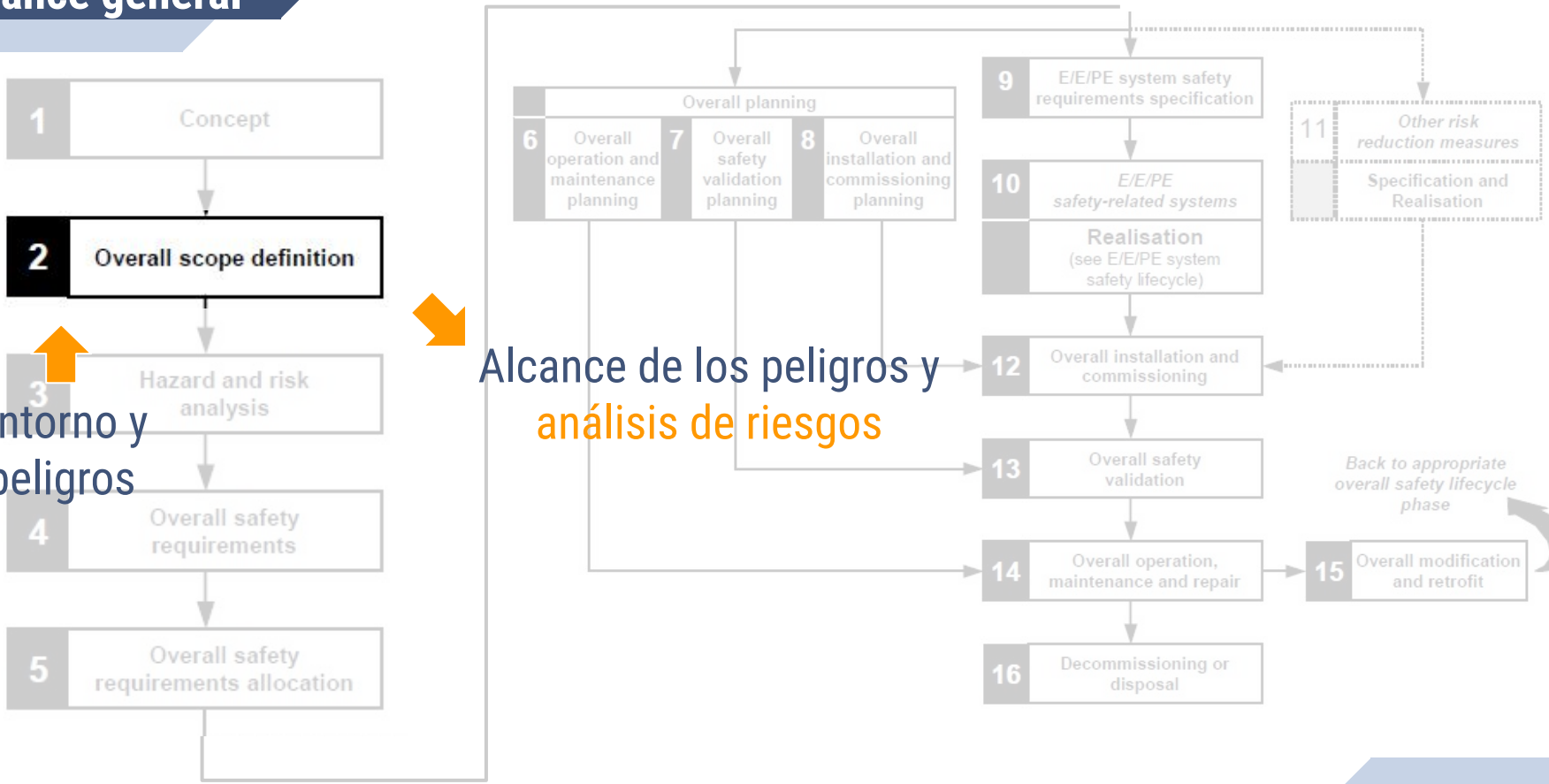


EUC: Equipment Under Control

Concepto


1 Concept
2 Scope definition
3 Hazard and risk analysis
4 Overall safety requirements
5 Overall safety requirements allocation
6 Overall operation and maintenance planning
7 Overall safety validation planning
8 Overall installation and commissioning planning
9 E/E/PE system safety requirements specification
10 E/E/PE safety-related systems
Realisation (see E/E/PE system safety lifecycle)
11 Other risk reduction measures
Specification and Realisation
12 Overall installation and commissioning
13 Overall safety validation
14 Overall operation, maintenance and repair
15 Overall modification and retrofit
Back to appropriate overall safety lifecycle phase
16 Decommissioning or disposal
Físicos, legislativos, etc
Información
Detalles del entorno y peligros

Alcance general



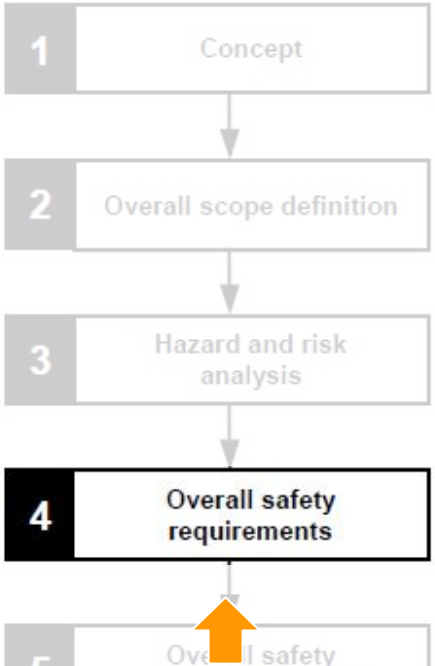
Entorno y peligros

Alcance de los peligros y análisis de riesgos

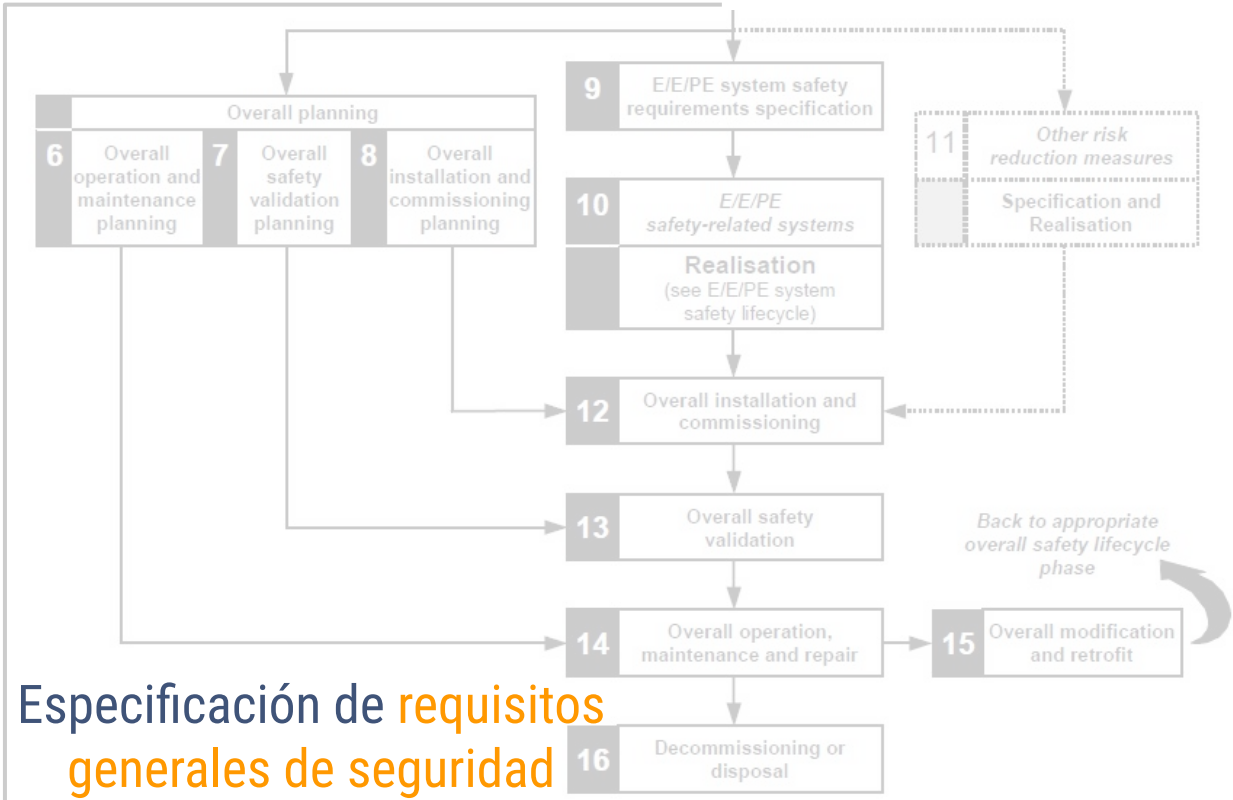
Análisis de riesgos

Alcance de los peligros y análisis de riesgos

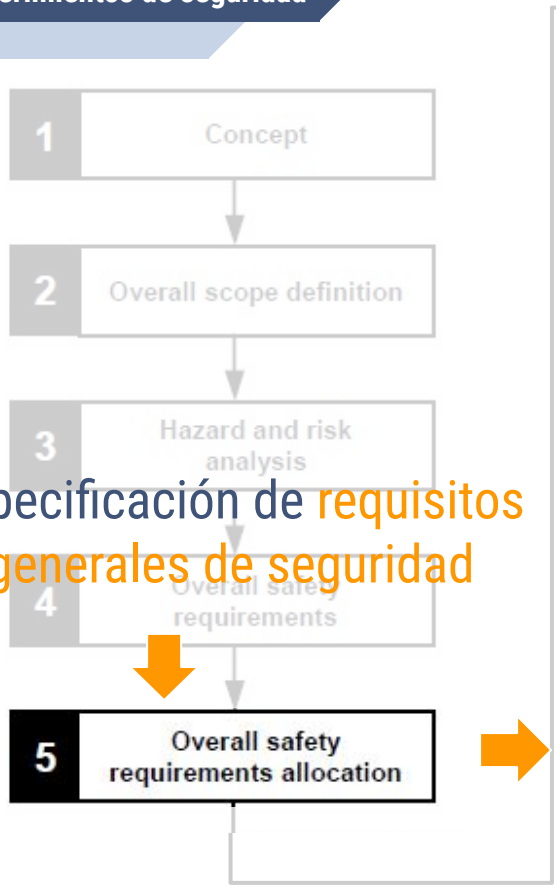
Descripción de los peligros y análisis de riesgos



Descripción de los peligros y análisis de riesgos



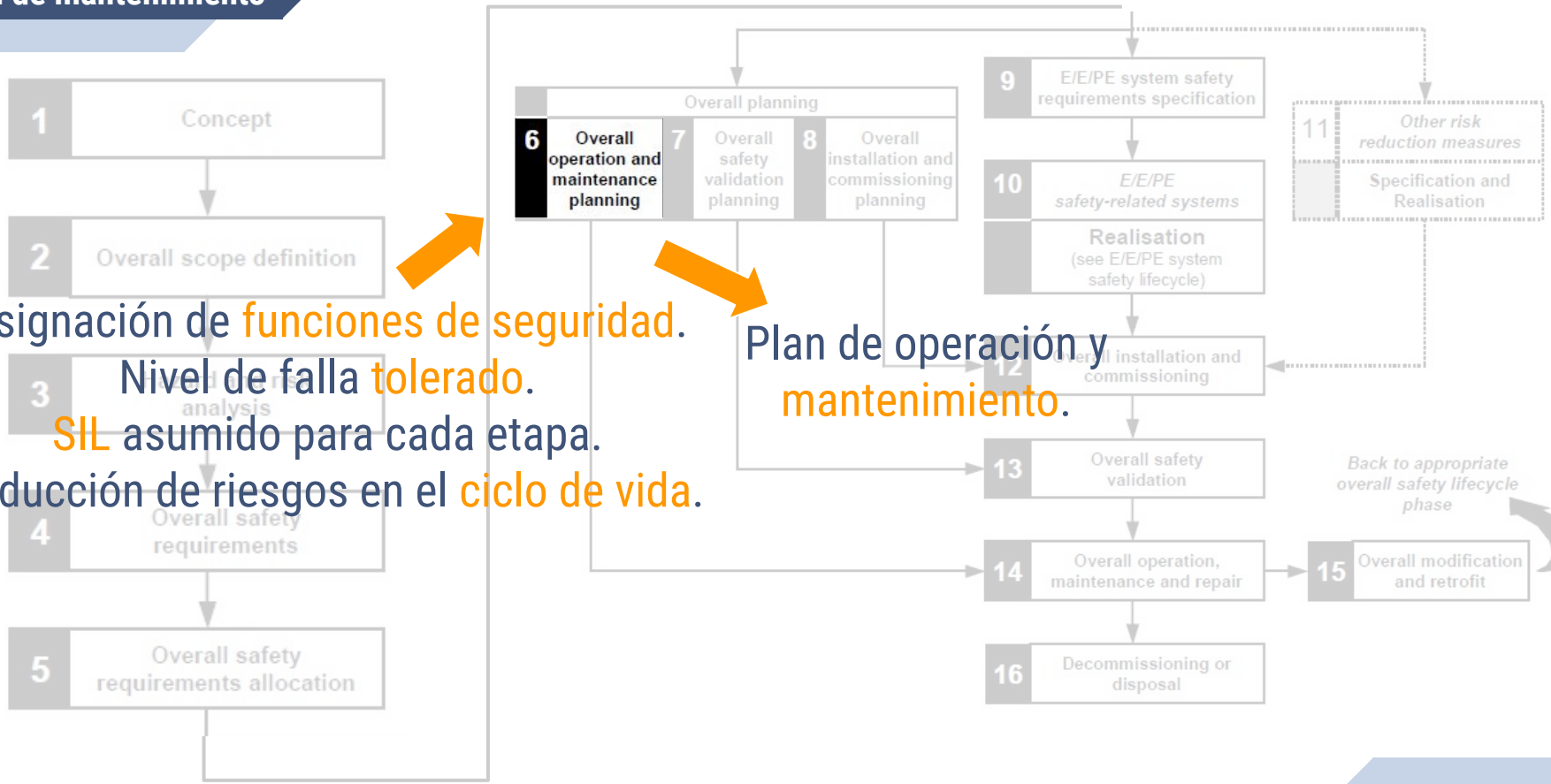
Especificación de requisitos generales de seguridad



Especificación de requisitos generales de seguridad

Asignación de funciones de seguridad.
Nivel de falla tolerado.
SIL asumido para cada etapa.
Reducción de riesgos en el ciclo de vida.

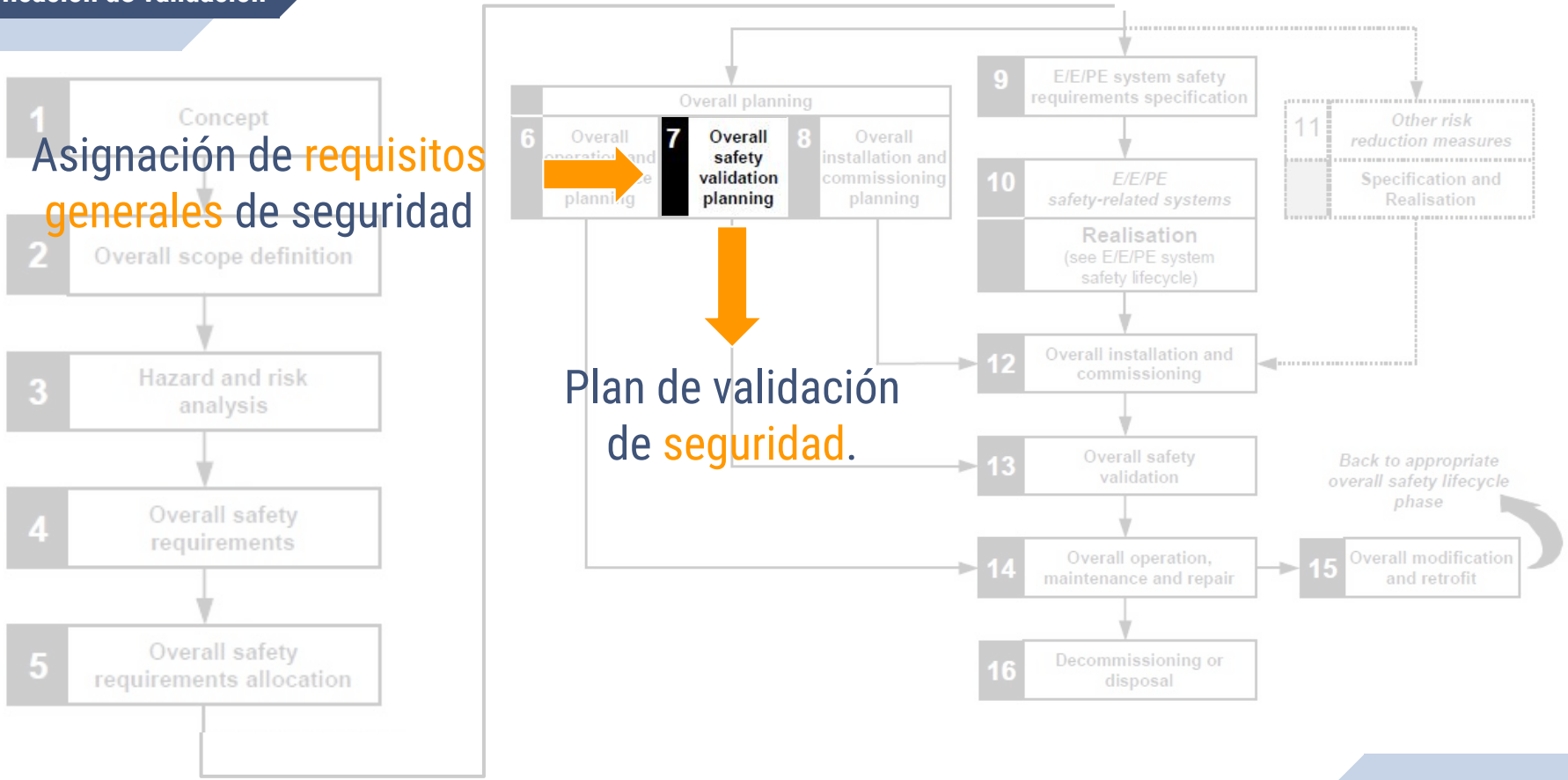
Plan de mantenimiento



Asignación de funciones de seguridad.
Nivel de falla tolerado.
SIL asumido para cada etapa.
Reducción de riesgos en el ciclo de vida.

Plan de operación y mantenimiento.

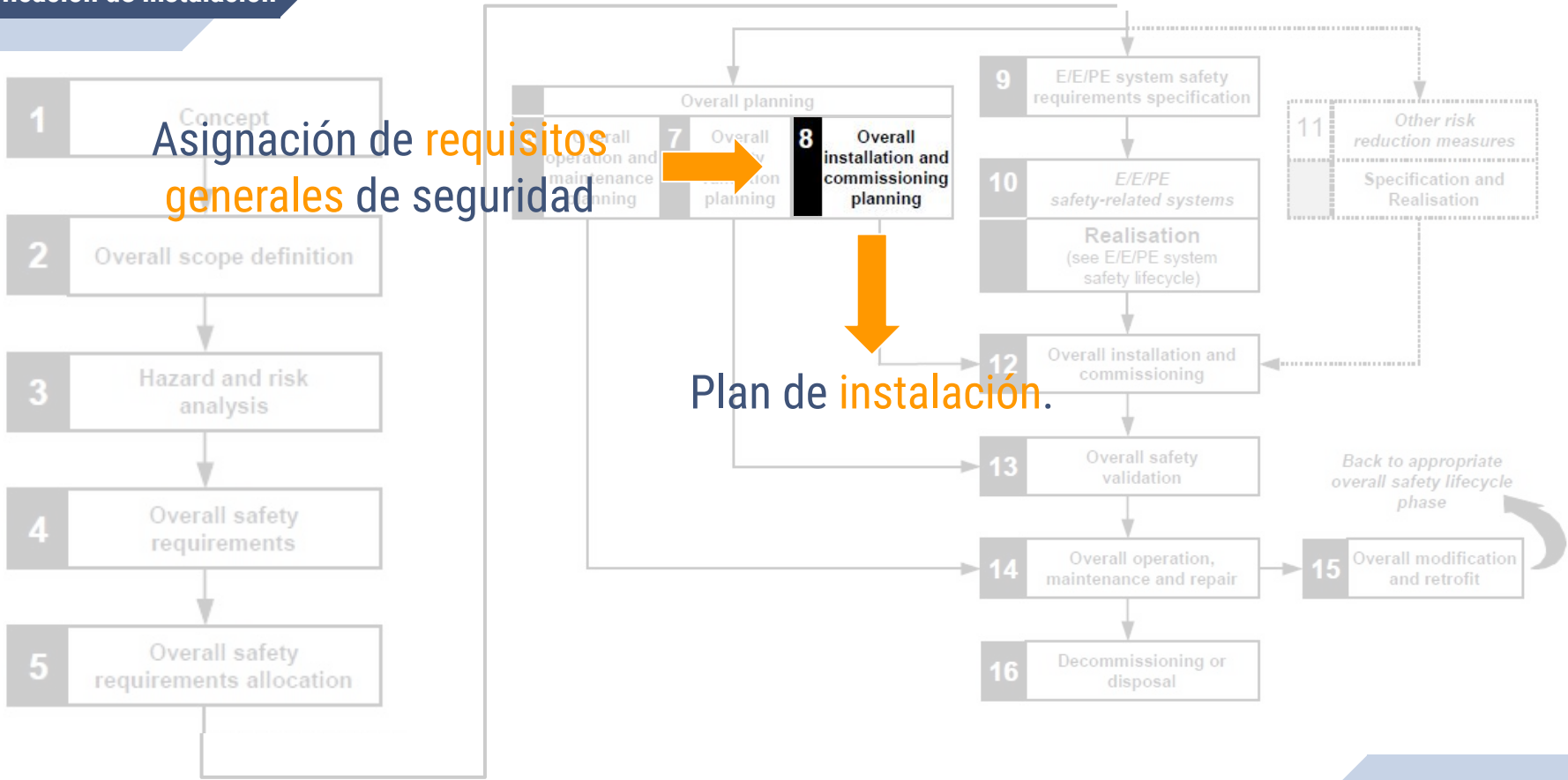
Asignación de requisitos generales de seguridad



Plan de validación de seguridad.

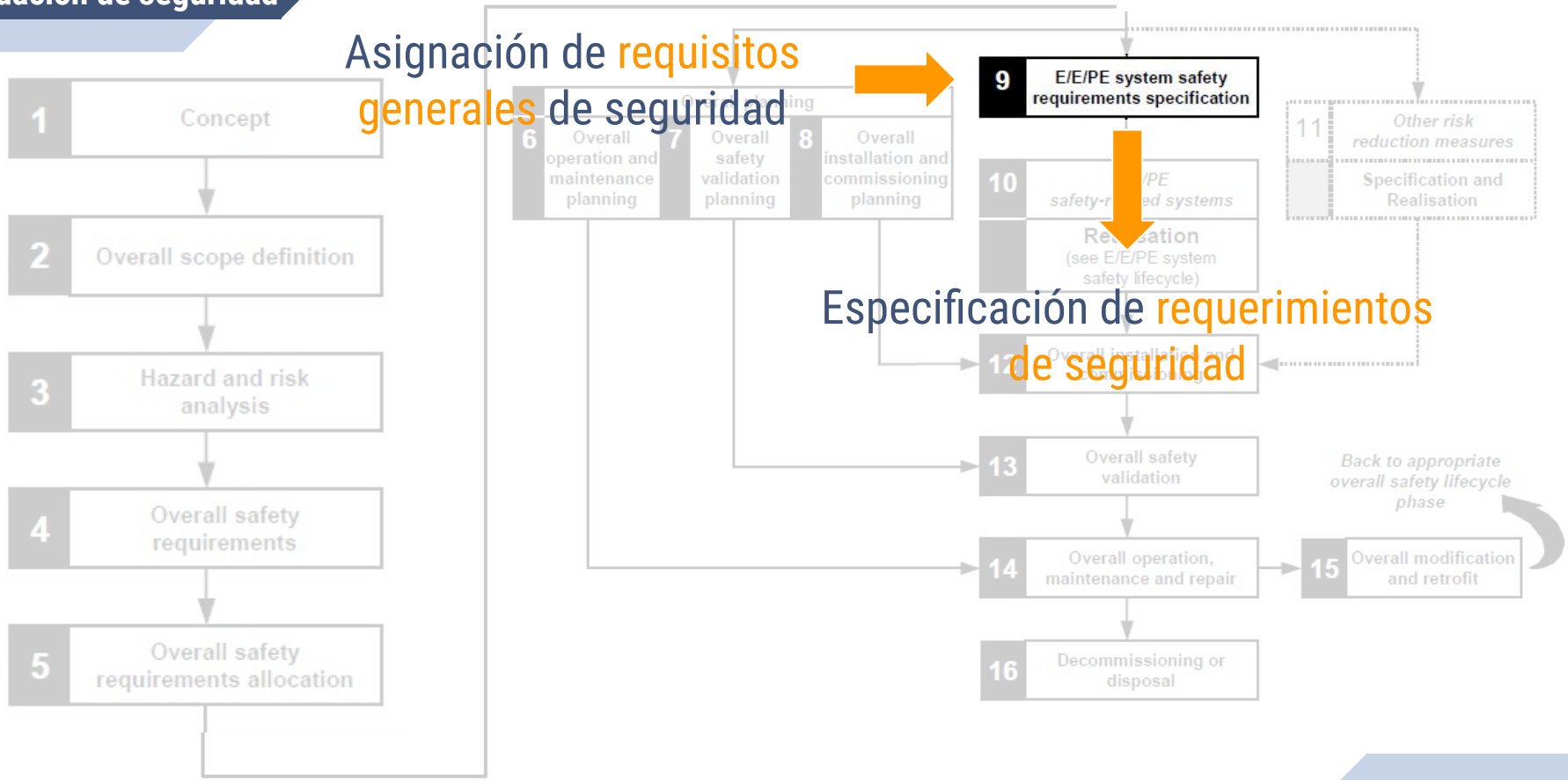
Asignación de requisitos generales de seguridad

Plan de instalación.

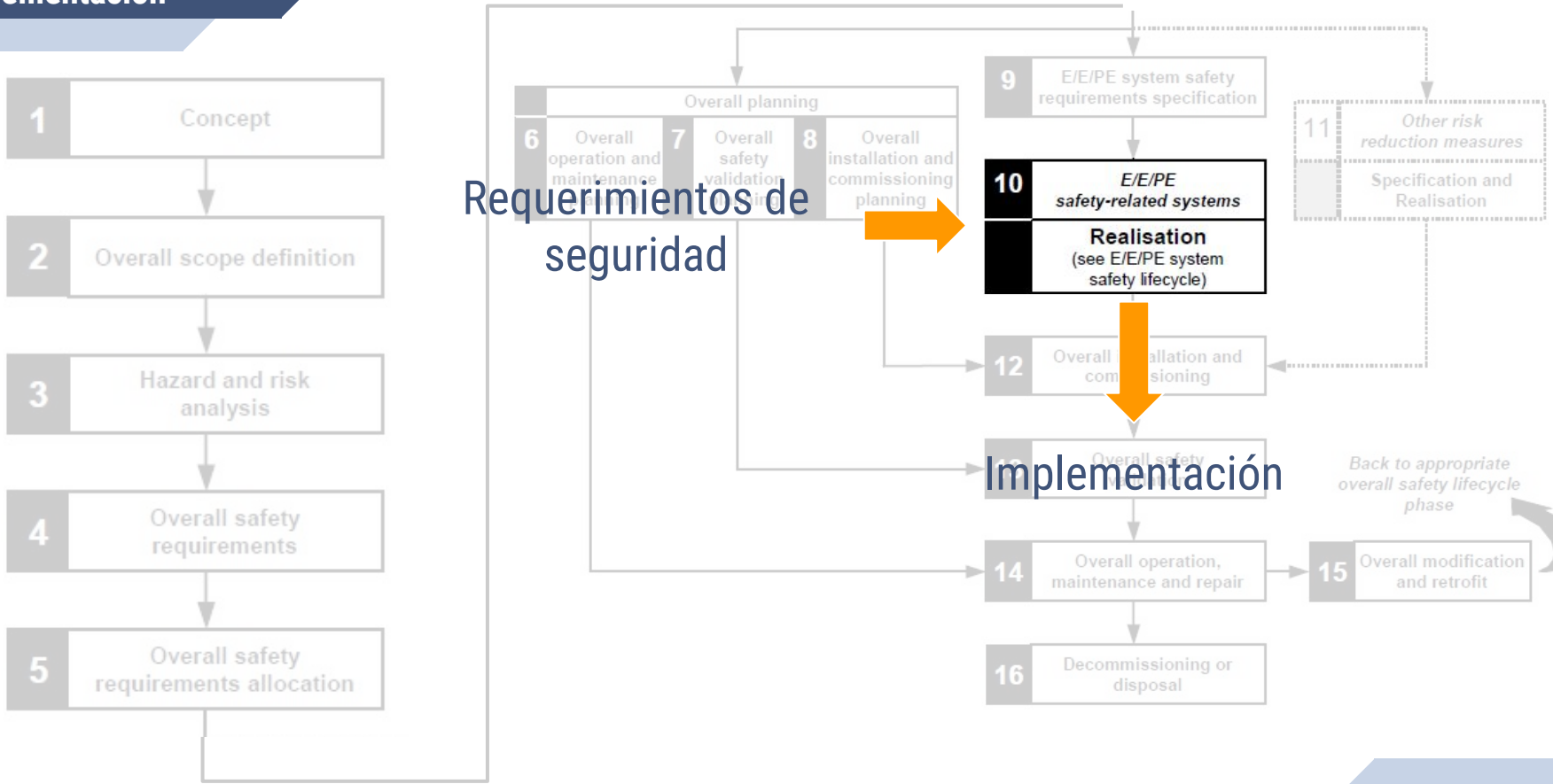


Asignación de requisitos generales de seguridad

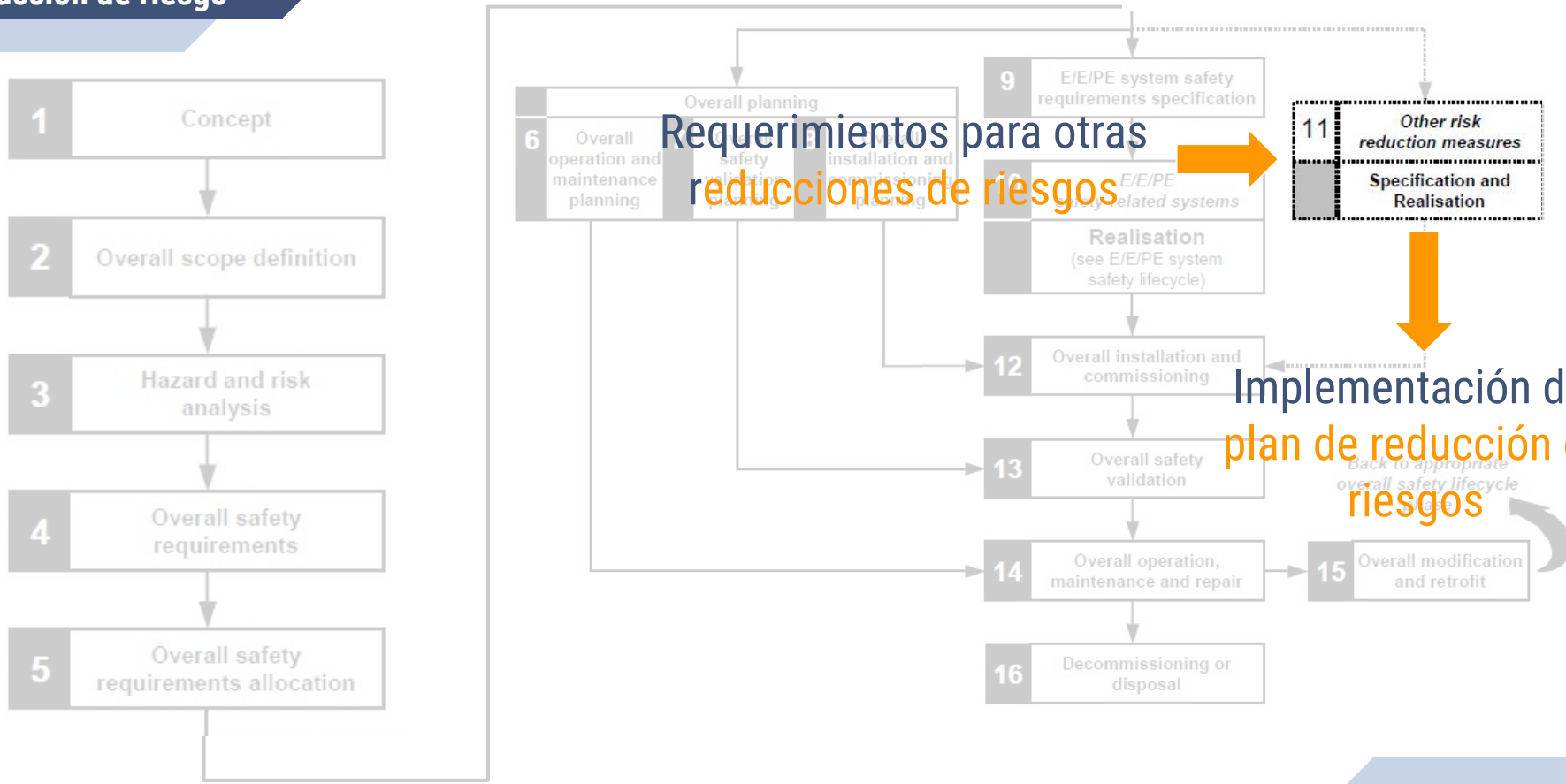
Especificación de requerimientos de seguridad



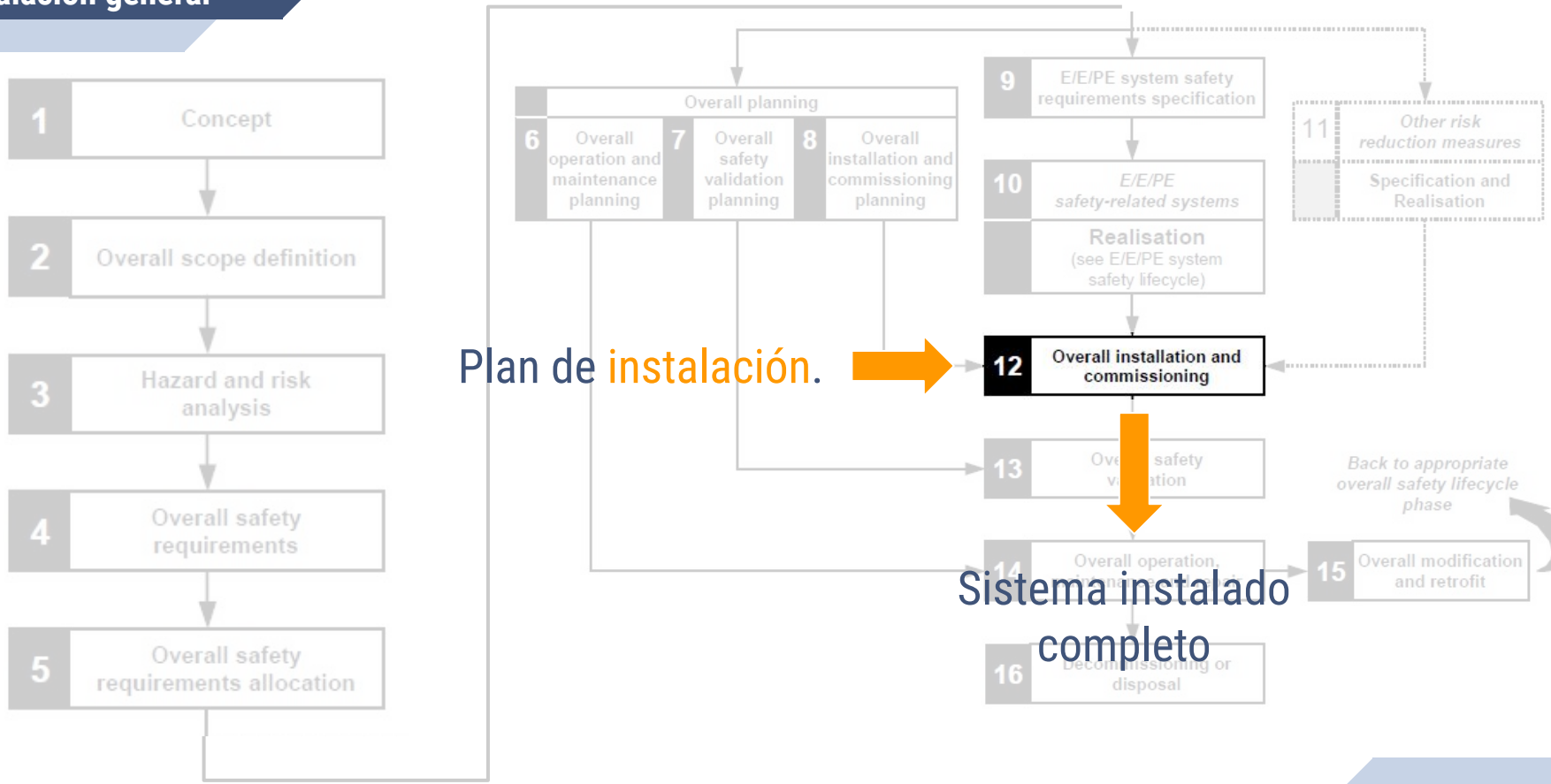
Implementación



Reducción de riesgo



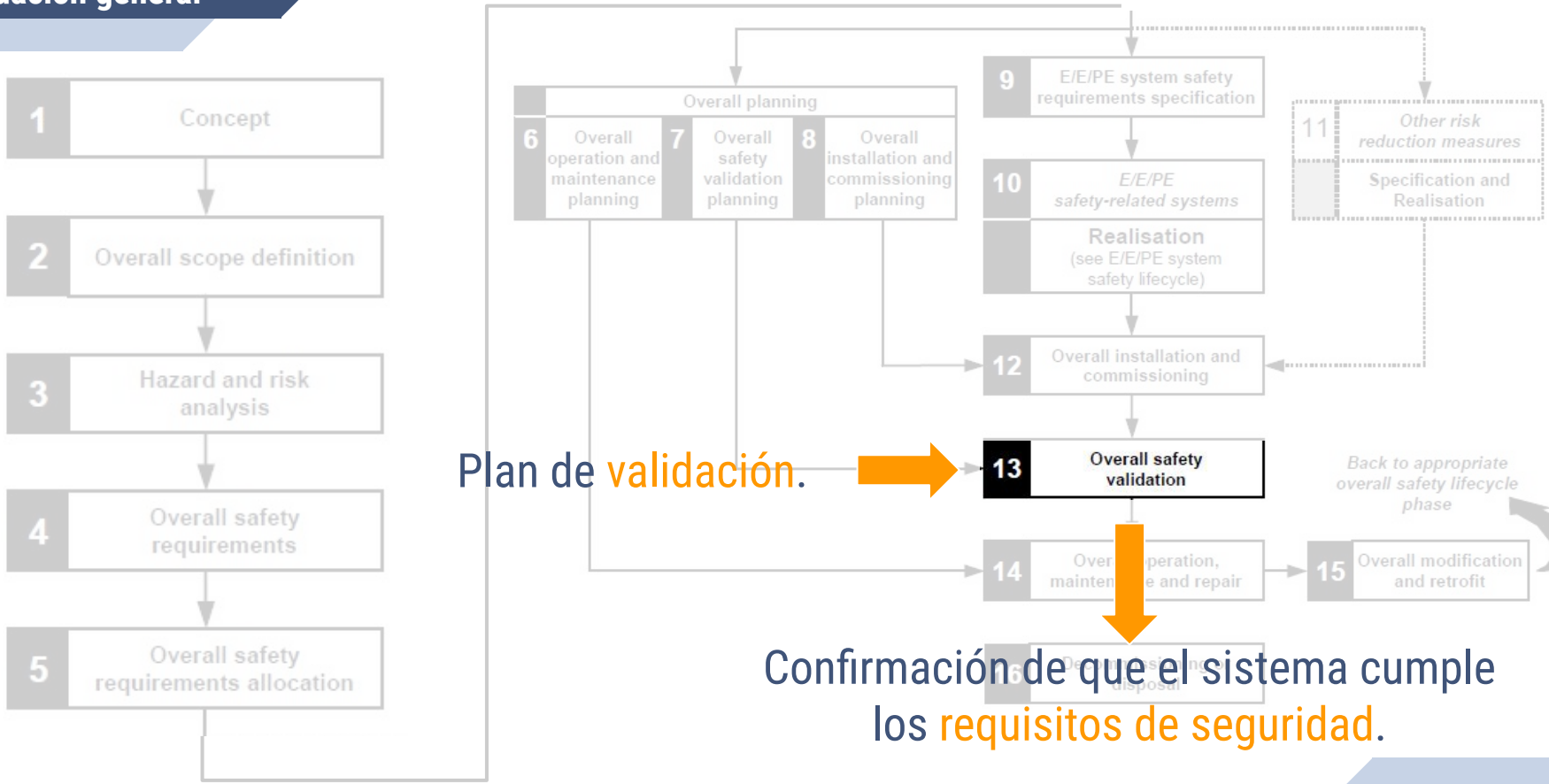
Instalación general



Plan de instalación.

Sistema instalado completo

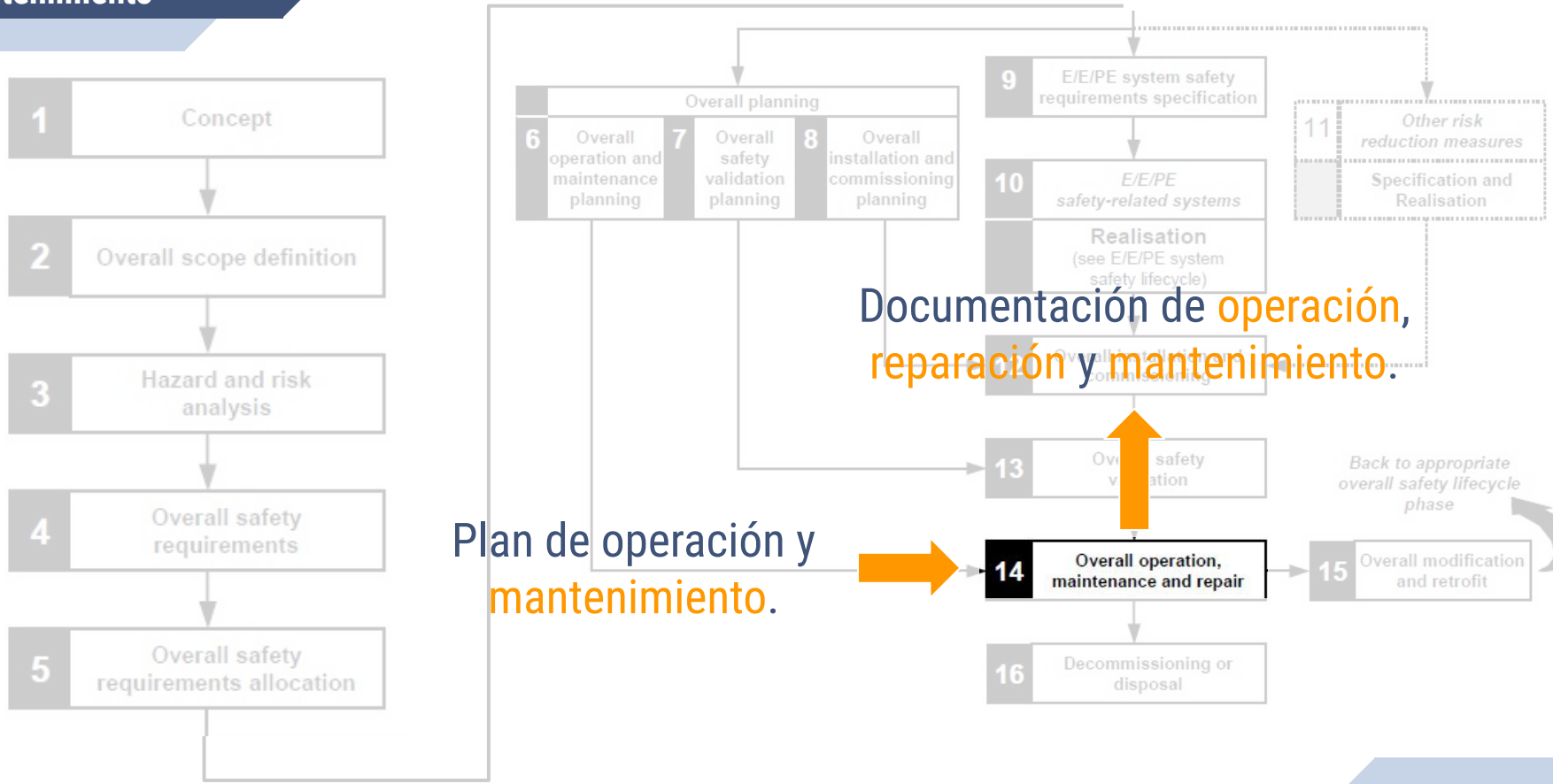
Validación general



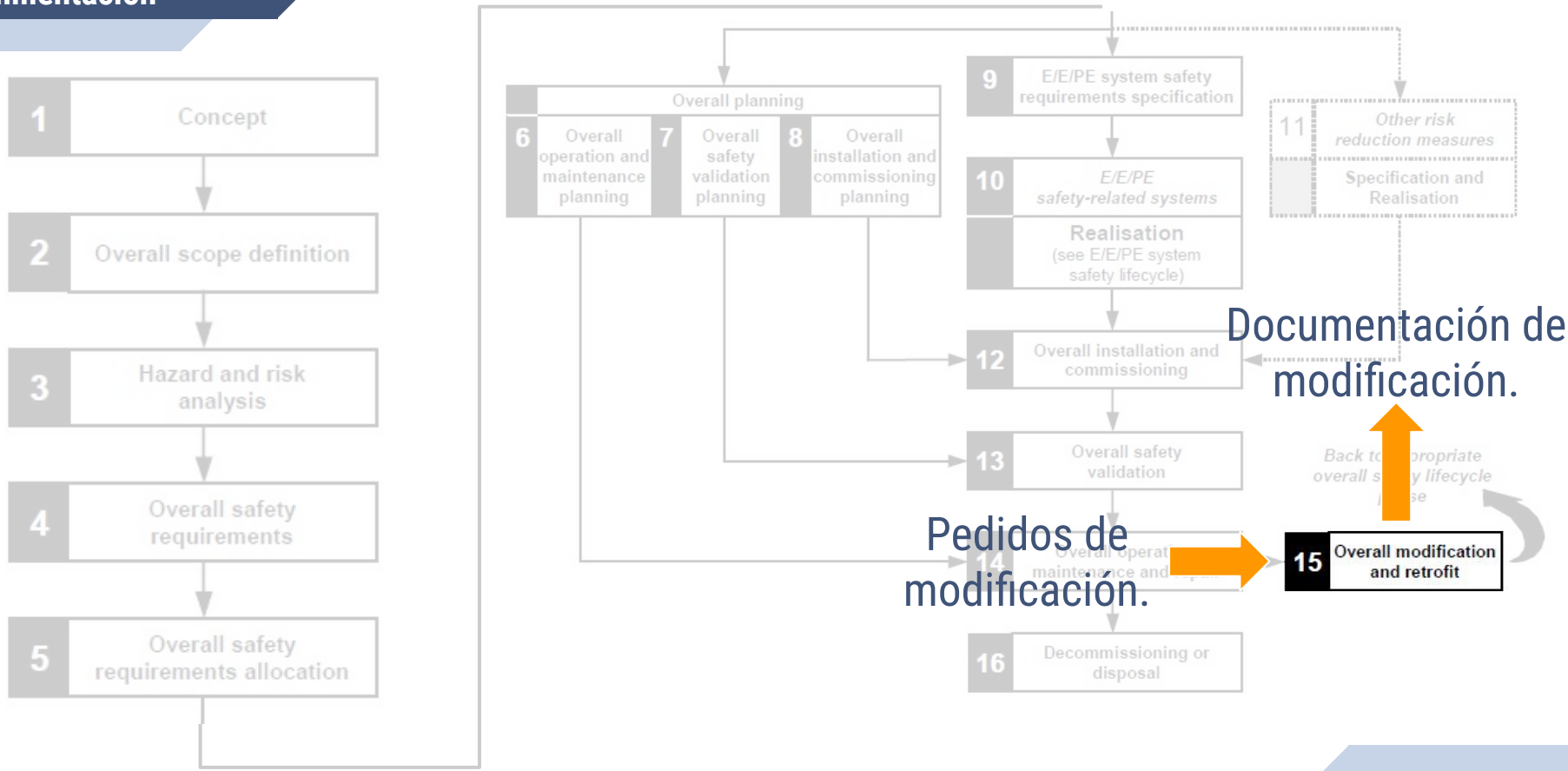
Plan de validación.

Confirmación de que el sistema cumple los requisitos de seguridad.

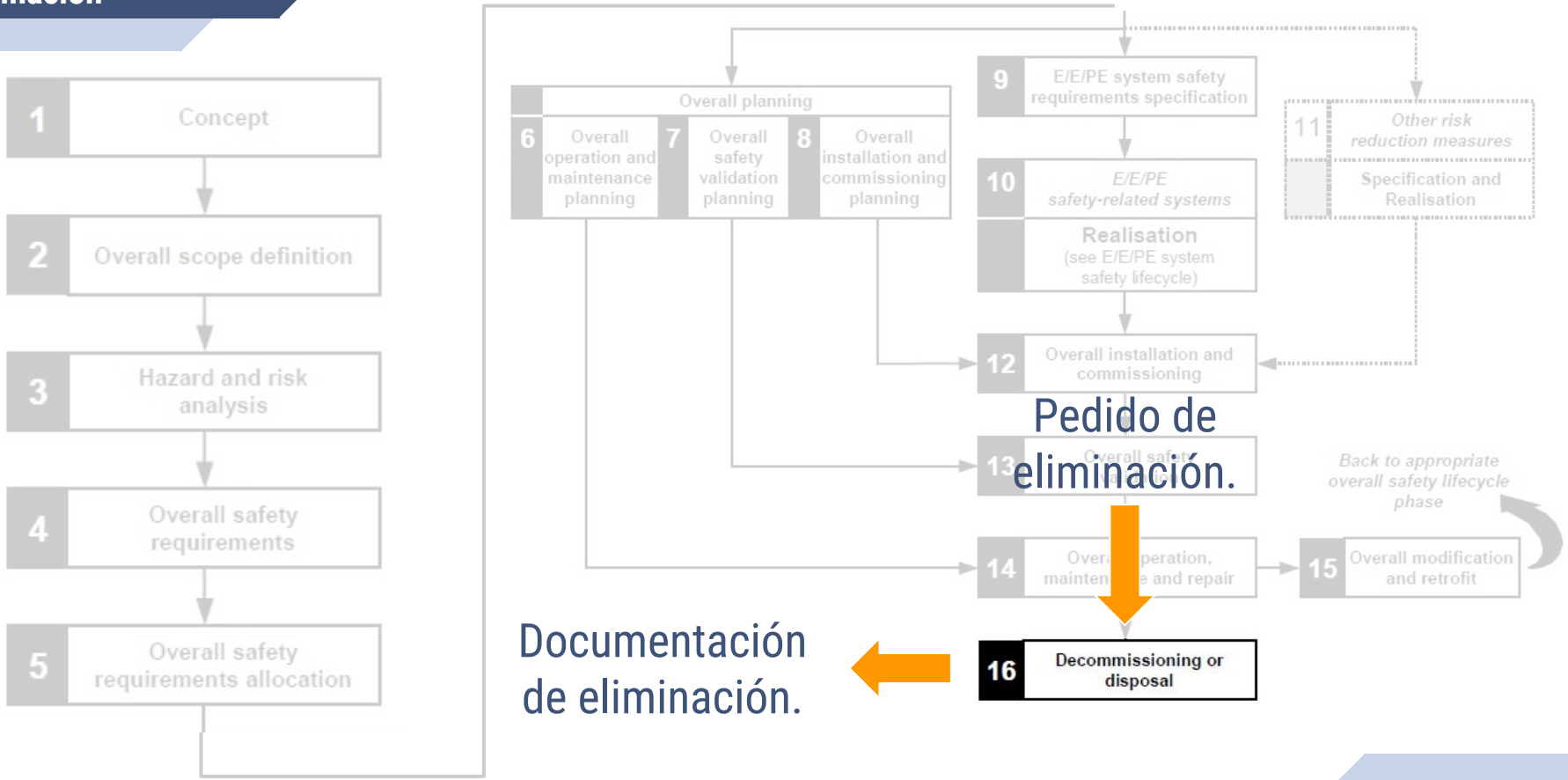
Mantenimiento



Realimentación



Eliminación



Niveles de independencia

Consecuencias

Nivel de independencia

	Lesión leve	Lesión severa / Una muerte	Muchas muertes	Masacre
Persona independiente	Suficiente	X1	Insuficiente	Insuficiente
Departamento independiente		X2	X1	Insuficiente
Organización independiente			X2	Suficiente

Factores para X_i :

Falta de experiencia

Sistemas complejos

Grado de innovación

X1 > X2

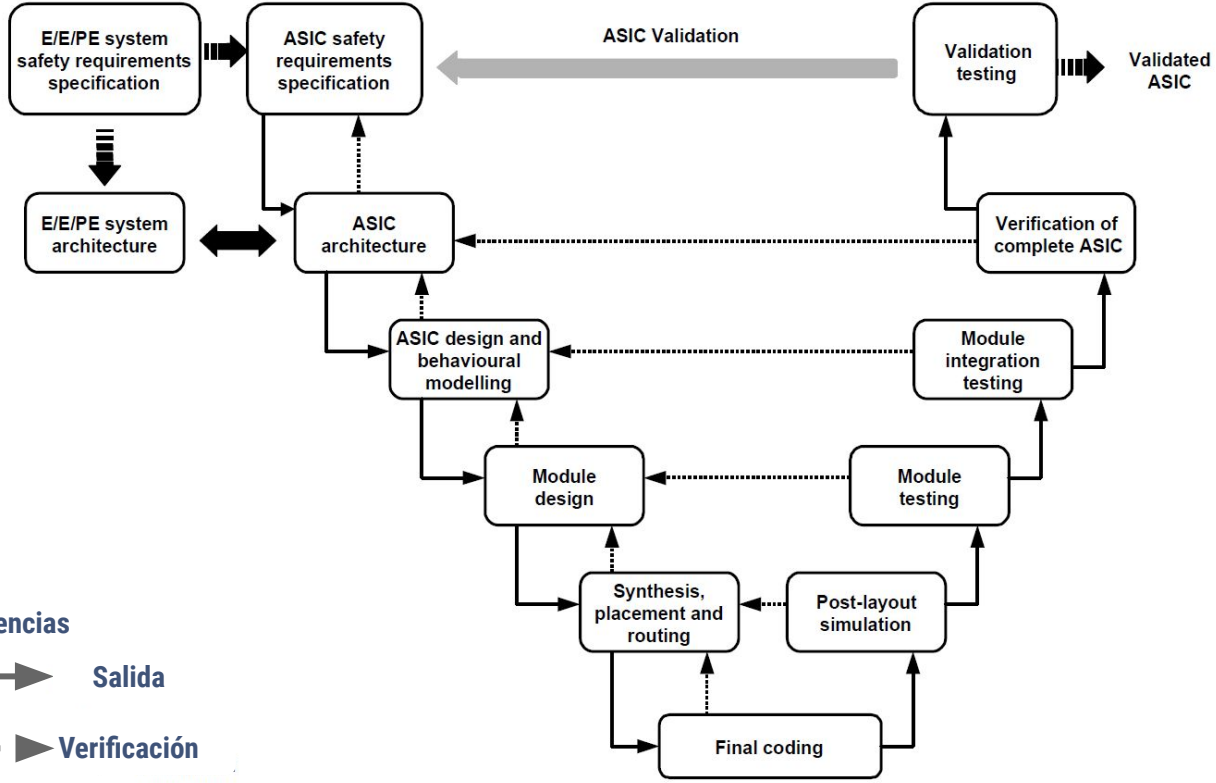
Si las partes deben cumplir diferentes SILs, todos deben cumplir el de mayor SIL

4

IEC 61508-2

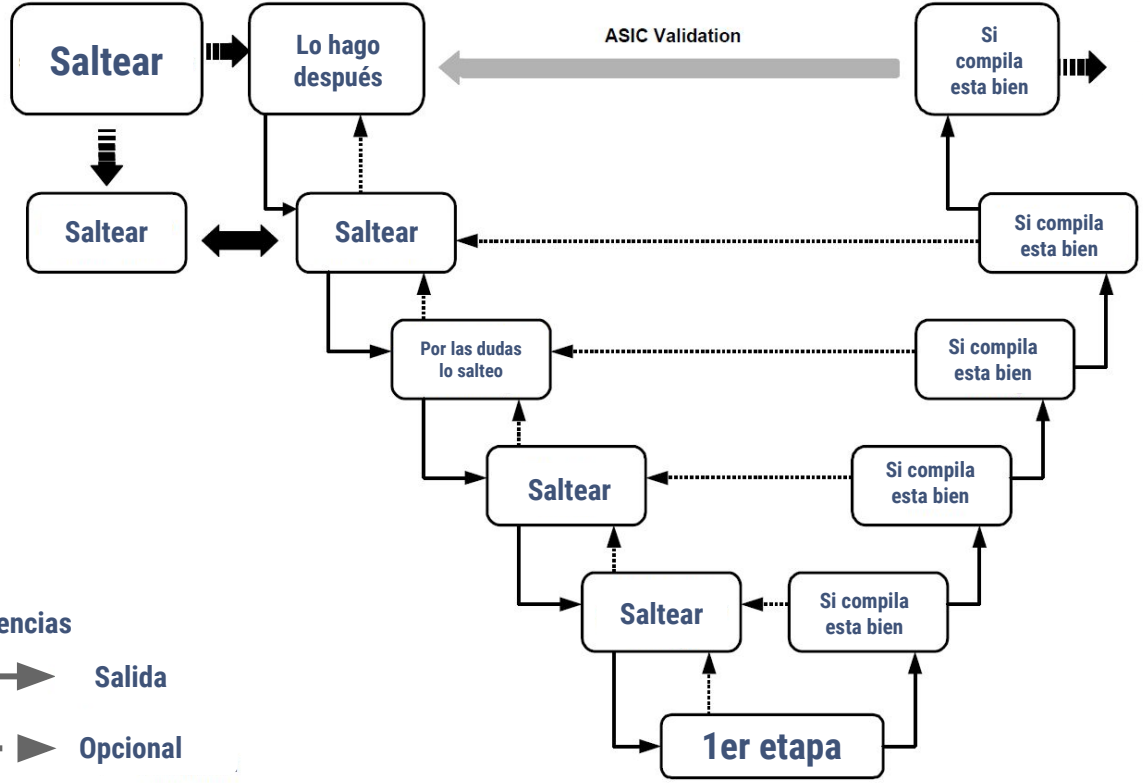
IEC 61508-2

Ciclo de vida (Lo que dice la norma)



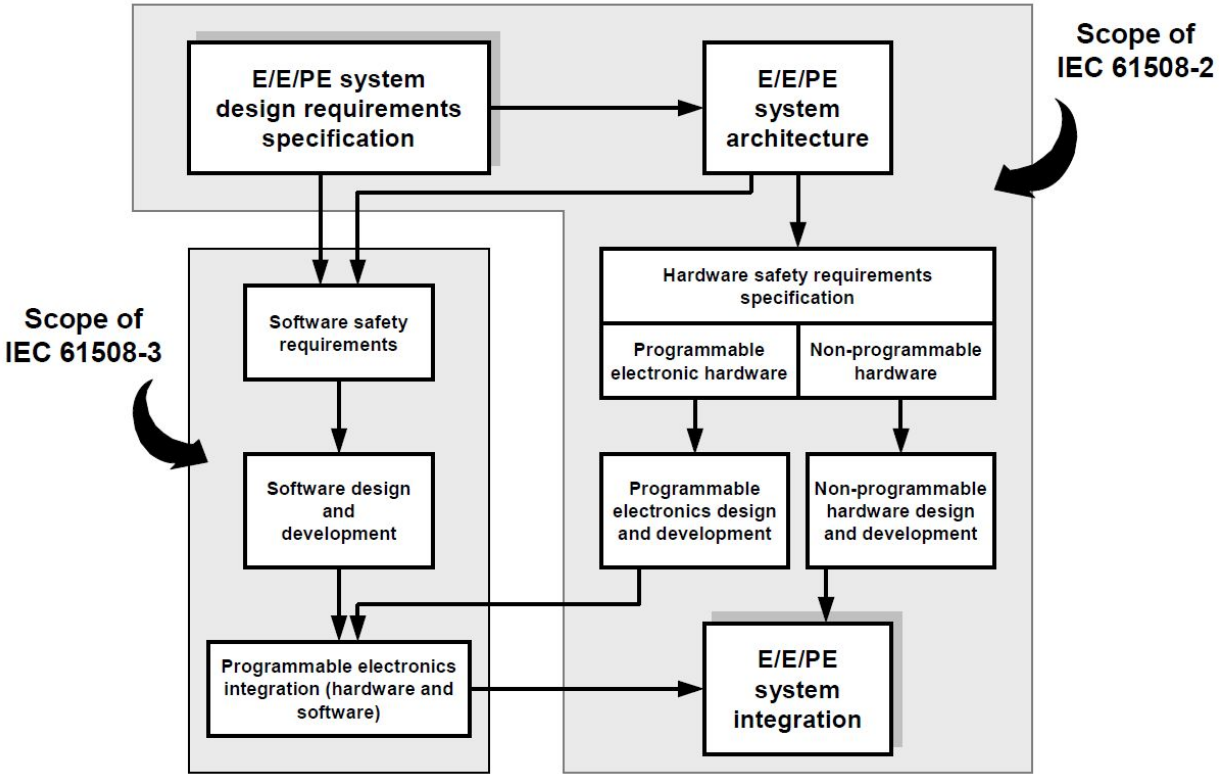
Menti_2_B

Ciclo de vida (Lo que la mayoría vemos)



Éxito asegurado!

Relación entre parte 2 y 3



Tolerancia a fallas en hardware (Tipo A vs tipo B)

Tolerancia a fallas de hardware en N
=
N+1 errores rompen TODO

Tipo A

- **Modos de falla** definidos para cada componente.
- Comportamiento de los elementos al fallar completamente determinado.
- Suficientes datos (confiables) para demostrar que se cumplen las tasas de fallas peligrosas (detectadas o no).

Tipo B

- Modo de falla indefinido para al menos UN componente.
- Comportamiento de algún elemento al fallar no está completamente determinado.
- Insuficientes datos para demostrar que se cumplen las tasas de fallas peligrosas (detectadas o no).

SFF(Safe Failure Fraction):

$$\frac{\Sigma\lambda S\ avg + \Sigma\lambda Dd\ avg}{\Sigma\lambda S\ avg + \Sigma\lambda Dd\ avg + \Sigma\lambda Du\ avg}$$

Tolerancia a fallas en hardware (Tipo A)

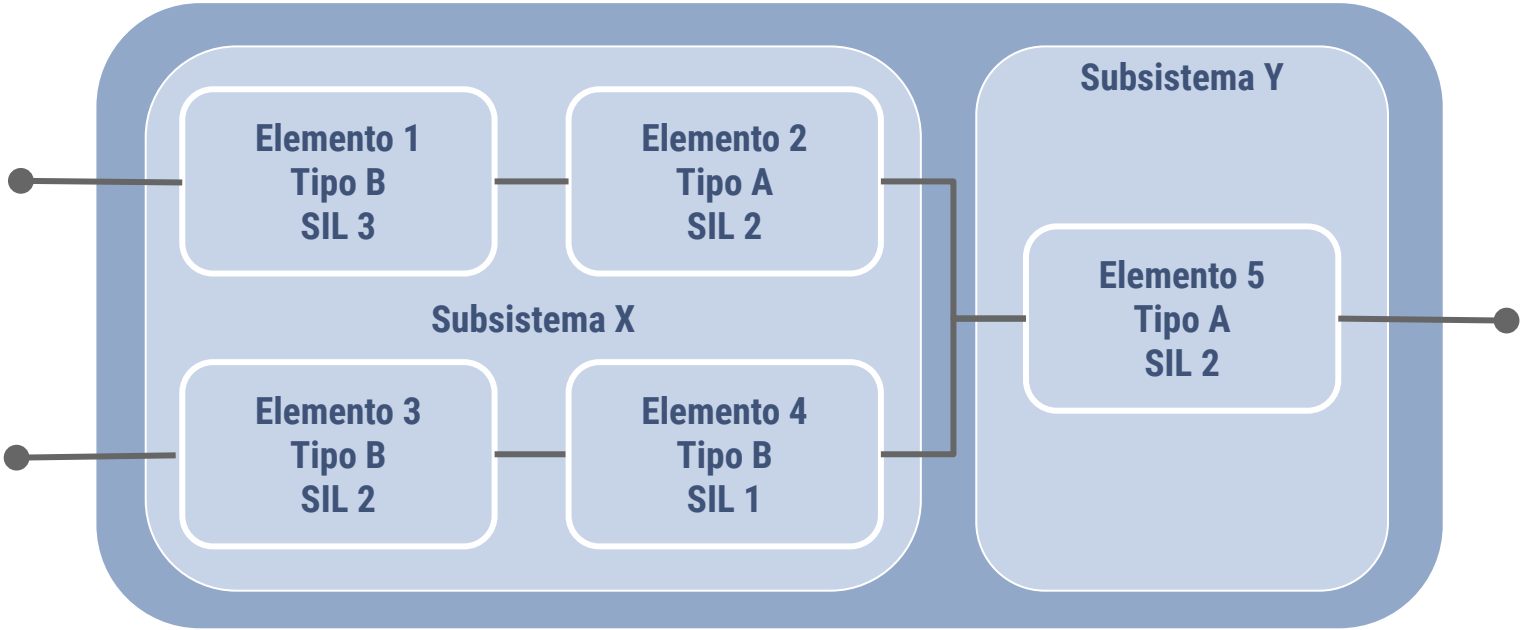
SFF (de un elemento)	Tolerancia a fallas en hardware		
	0	1	2
< 60%	SIL 1	SIL 2	SIL 3
60 a 90%	SIL 2	SIL 3	SIL 4
90 a 99%	SIL 3	SIL 4	SIL 4
>= 99%	SIL 3	SIL 4	SIL 4

Tolerancia a fallas en hardware (Tipo B)

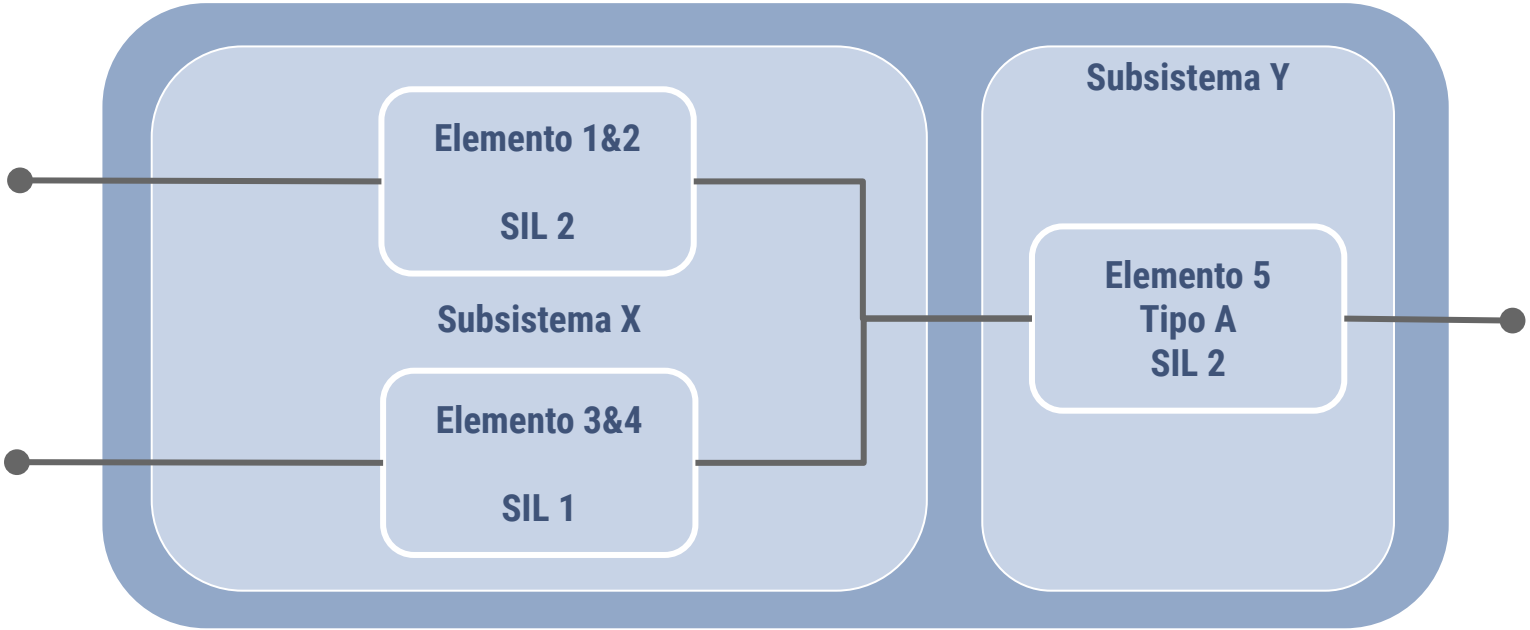
SFF (de un elemento)	Tolerancia a fallas en hardware		
	0	1	2
< 60%	JAMAS!	SIL 1	SIL 2
60 a 90%	SIL 1	SIL 2	SIL 3
90 a 99%	SIL 2	SIL 3	SIL 4
>= 99%	SIL 3	SIL 4	SIL 4

Reducciones de arquitecturas (Ejemplo 1)

Reducciones de arquitecturas (Ejemplo 2)

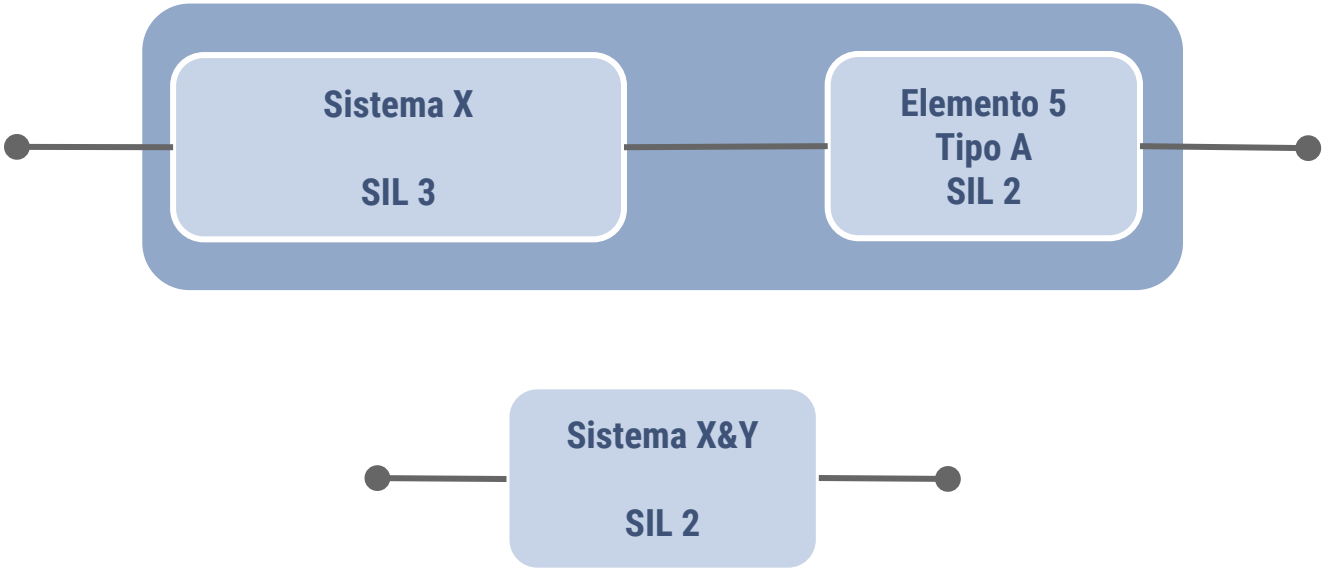


Reducciones de arquitecturas (Ejemplo 2)



Tolerancia a falla de UNO + Máximo SIL

Reducciones de arquitecturas (Ejemplo 2)



Técnicas de detección de errores sistemáticos (hardware)

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Supervisión de la secuencia del programa	AR	AR	AR	AR	Supervisar el comportamiento y la integridad de la secuencia del programa.
	Detección de fallas mediante monitoreo en línea	R	R	R	R	Monitorear estados del sistema remotamente
	Redundancia de hardware	R	R	R	R	Clase 4
	Protección de código	R	R	R	R	Por ejemplo **CRC**.
	Diversidad de hardware	-	-	R	R	Clase 4

Menti_C_1

Al menos uno de los grises

Técnicas de detección de errores sistemáticos (ambiente)

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Mediciones de tensión	O	O	O	O	Tensión de rotura, variaciones de tensión, sobretensión, baja tensión, variaciones de frecuencia de la fuente, etc.
Separación del camino de energía eléctrica y el camino de datos	O	O	O	O	Cursar **Diseño de Circuitos Electrónicos**
Inmunización a interferencias	O	O	O	O	Cursar **Diseño de Circuitos Electrónicos**
Medición de variables ambientales	O	O	O	O	Temperatura, humedad, agua, vibración, polvo, sustancias corrosivas, etc.
Supervisión de la secuencia del programa	AR	AR	AR	AR	Slide anterior
Medidas contra el aumento de temperatura	AR	AR	AR	AR	Baja efectividad: Detectar el aumento Alta efectividad: Corte por fusible térmico / alarmas / Enfriamiento forzado
Separación espacial de varias líneas	AR	AR	AR	AR	Cursar **Diseño de Circuitos Electrónicos**

Opcionales los blancos

Técnicas de detección de errores sistemáticos (ambiente)

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Principio de corriente inactiva	R	R	R	R	El sistema de seguridad se ejecuta al cortarse la energía
	Medición para detectar roturas o cortocircuitos en líneas de señal	R	R	R	R	Cursar **Diseño de Circuitos Electrónicos**
	Detección de fallas mediante monitoreo en línea	R	R	R	R	Ver slides anteriores
	Redundancia de hardware	R	R	R	R	Clase 5
	Protección de código	R	R	R	R	Ver slides anteriores
	Transmisión de señales antivalentes	R	R	R	R	Clase 5
	Diversidad de hardware	R	R	R	R	Clase 5
	Arquitectura de software	R	R	R	R	

Al menos uno de los grises y uno de los negros

Técnicas de detección de errores sistemáticos

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Protección contra modificaciones	O	O	O	O	Comprobaciones de integridad, aislar puertos, etc.
	Detección de fallas mediante monitoreo en línea	R	R	R	R	Ver slides anteriores
	Reconocimiento de entrada	R	R	R	R	Comprobar entradas, antirebotes, múltiples pedidos de confirmación, etc.
	Programación de aserción de fallas	R	R	R	R	Verificar condiciones pre-ejecución y post-ejecución de cualquier función.

Al menos uno de los grises

Técnicas para evitar errores durante la especificación

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4
	Gestión de proyectos	O	O	O	O
	Documentación	O	O	O	O
	Separar funciones de seguridad de las generales	AR	AR	AR	AR
	Especificación estructurada	AR	AR	AR	AR
	Inspección de la especificación	-	AR	AR	AR
	Métodos semiformales	R	R	AR	AR
	Listas de verificación	R	R	R	R
	Métodos formales	-	-	R	R
	Herramientas de especificación asistidas	-	R	R	R

Al menos uno de los grises

Cursar **Gestión de proyectos**

Cursar **Taller de trabajo final**

Una separación clara reduce el esfuerzo de testear los sistemas relacionados con la seguridad.

Reducir la complejidad e interferencia con una estructura jerárquica de requisitos parciales.

Evitar que la especificación esté incompleta o sea contradictoria. (OTRA persona)

Diagramas de bloques/secuencias/flujo, FSM, Redes de Petri, Modelo de datos, Tablas de verdad, etc.

Serie de preguntas genéricas. Usado en cualquier etapa

(matemáticamente)
Raise, Z, Etc. Sumar Petri y FSM si se usan estrictamente.

TestLink, **Visual Paradigm**, etc.

Técnicas para evitar errores durante el desarrollo

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Cumplimiento de pautas y estándares	O	O	O	O	
	Gestión de proyectos	O	O	O	O	Ver slides anteriores
	Documentación	O	O	O	O	Ver slides anteriores
	Diseño estructurado	AR	AR	AR	AR	Diseño de circuito estructurado jerárquicamente, uso de piezas fabricadas y probadas.
	Modularización	AR	AR	AR	AR	Subsistemas claramente definidos y restringidos, con interfaces simples, sección transversal mínima. (**sección transversal**: datos compartidos)

Opcionales los blancos

Técnicas para evitar errores durante el desarrollo

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Uso de componentes probados	R	R	R	R	Fabricantes con unidades patrón reconocidas, alto requisitos de seguridad y almacenamiento
	Métodos semiformales	R	R	AR	AR	Ver slides anteriores
	Listas de verificación	-	R	R	R	Ver slides anteriores
	Herramientas de diseño asistidas	-	R	R	R	**AutoCAD**, **KiCAD**, **Altium**, **Proteus**, etc.
	Simulación	-	R	R	R	Modelo de comportamiento de software. Modelo de comportamiento de a nivel componente y general.
	Inspección del hardware	-	R	R	R	Aplicar entradas representativas, medir salidas y rastrear manualmente a través de la lógica del programa
	Métodos formales	-	-	R	R	Ver slides anteriores

Al menos uno de los grises

Técnicas para evitar errores durante la integración

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Pruebas funcionales	O	O	O	O	Para evitar fallas en la integración de SW y HW. Comparar con especificaciones. Buscar fallas de modo común comparando contra HW ya validados.
	Gestión de proyectos	O	O	O	O	Ver slides anteriores
	Documentación	O	O	O	O	Ver slides anteriores
	Prueba de caja negra	R	R	R	R	No se utiliza ningún conocimiento previo de la estructura interna del sistema para guiar las pruebas
	Experiencia de campo	R	R	R	R	Reutilizar componentes o subsistemas que ya han probado en campo no tener fallas (o que no sean graves)
	Prueba estadística	-	-	R	R	Ingresar datos según la distribución estadística esperada de las entradas operativas reales

Al menos uno de los grises

Técnicas para evitar errores durante el mantenimiento

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4
	Instrucciones de operación y mantenimiento	AR	AR	AR	AR
	Facilidad de uso (User friendliness)	AR	AR	AR	AR
	Facilidad de mantenimiento	AR	AR	AR	AR
	Gestión de proyectos	O	O	O	O
	Documentación	O	O	O	O
	Posibilidades de operación limitadas	-	R	AR	AR
	Protección contra errores del operador	-	R	AR	AR
	Operación solo por operadores calificados	-	R	AR	AR

Instrucciones fácilmente entendibles de uso y mantenimiento, con esquemas para procesos complejos.

Mínima intervención humana (y fácil!), mínimo daño por error del operador, bien señalizado, operador capacitado

Poco o nulo mantenimiento, con herramientas de diagnóstico suficientes para reparaciones inevitables.

Ver slides anteriores

Ver slides anteriores

Limitar modos de funcionamiento

Detectar **entradas incorrectas** (valor, tiempo, etc.) mediante comprobaciones de aceptación. Indicar previamente qué entradas son posibles y permitidas

Capacitar personal adecuado

Al menos uno de los grises

Técnicas para evitar errores durante la validación

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Pruebas funcionales	AR	AR	AR	AR	Ver slides anteriores
	Ensayos funcionales en condiciones ambientales	AR	AR	AR	AR	Evaluar la fiabilidad de las funciones de seguridad ante diversas condiciones ambientales.
	Prueba de inmunidad a sobretensión de interferencias	AR	AR	AR	AR	Operar con todas las señales con ruido estándar (acercarse al límite de sobretensión con cuidado).
	Prueba de inserción de fallas (cuando se requiere una cobertura de diagnóstico ≥ 90%)	AR	AR	AR	AR	Abrir o cortocircuitar líneas (alimentación, datos, buses, etc.). Fallas unitarias y luego múltiples.
	Gestión de proyectos	O	O	O	O	Ver slides anteriores
	Documentación	O	O	O	O	Ver slides anteriores

Al menos uno de los grises B5

Técnicas para evitar errores durante la validación

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4
	Análisis dinámico y análisis de fallas	-	R	R	R
	Simulación	-	R	R	R
	Análisis del peor de los casos	-	-	R	R
	Análisis estático	R	R	NR	NR

Insuficiente si no se combina con análisis dinámico

Examinar todas las posibles fuentes de falla de un sistema determinando el impacto en su comportamiento.

Ver slides anteriores

Analizar posibles fallas sistemáticas que surgen de combinaciones desfavorables de las condiciones ambientales y las tolerancias de los componentes.

Analizar coherencia de datos, flujo de control, etc.

Al menos uno de los grises

Técnicas para evitar errores durante la validación

Color: Efectividad necesaria
O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
■	Prueba funcional ampliada	-	AR	AR	AR	Comportamiento frente a condiciones raras o fuera de lo especificado
■	Prueba de caja negra	R	R	R	R	Ver slides anteriores
■	Prueba de inserción de fallas (cuando se requiere una cobertura de diagnóstico < 90%)	R	R	R	R	Ver slides anteriores
■	Prueba estadística	-	-	R	R	Ver slides anteriores
■	Pruebas en el peor de los casos	-	-	R	R	Someter al sistema a los valores marginales ambientales más altos permitidos.
■	Experiencia de campo	R	R	R	NR	Ver slides anteriores

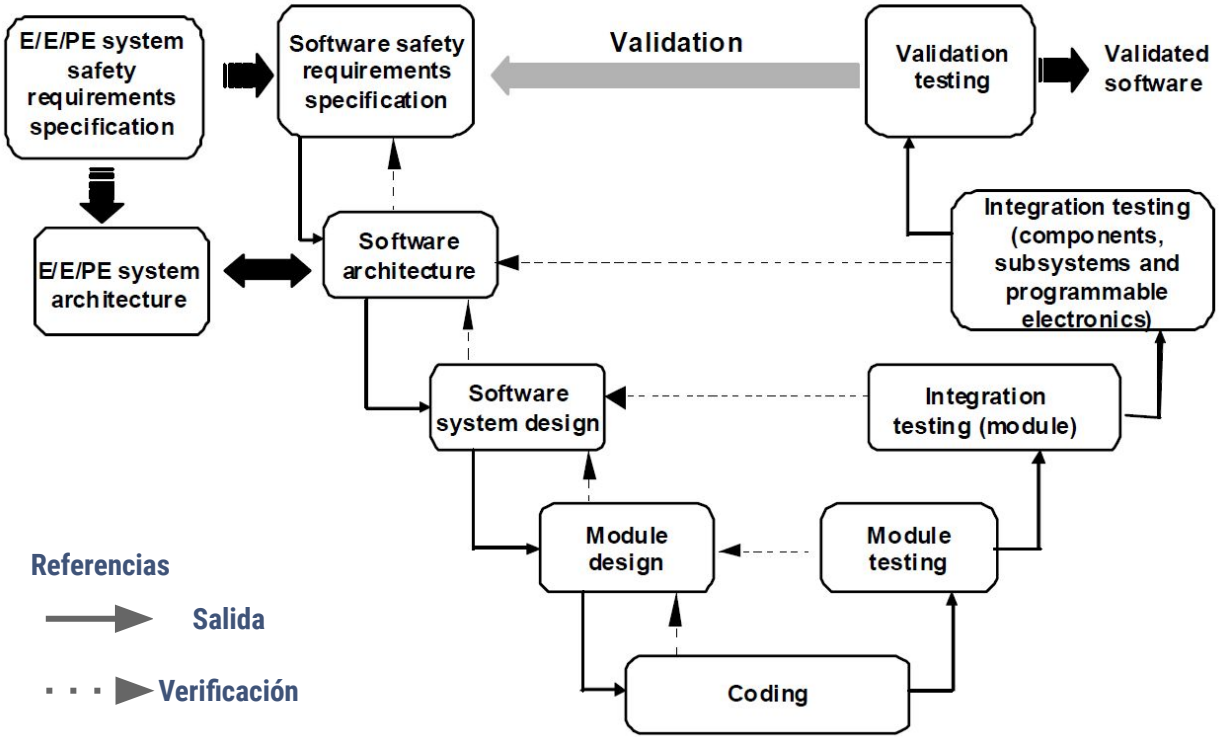
Al menos uno de los negros

5

IEC 61508-3

IEC 61508-3

Ciclo de vida (Software)



Técnicas para especificación de requerimientos (software)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Métodos semiformales	-	-	-	-	Ver slides anteriores
Métodos formales	-	-	-	-	Ver slides anteriores
Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y los requisitos de seguridad de software	-	-	-	-	Verificar que un requisito se corresponde adecuadamente en etapas posteriores de ciclo de vida
La trazabilidad hacia atrás entre los requisitos de seguridad y las necesidades de seguridad percibidas.	-	-	-	-	Verificar que cada decisión de implementación esté claramente justificada por algún requisito. Si no, la implementación es innecesariamente más compleja.
Herramientas de especificación asistidas	-	-	-	-	Ver slides anteriores

Opcionales

Técnicas para diseño de arquitectura (I)

Menti_C_2

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Detección de fallas	-	R	AR	AR
Error al detectar códigos	R	R	R	AR
Programación de aserción de fallas	R	R	R	AR
Diversas técnicas de monitorización (con independencia/separación entre el monitor y la función monitorizada en la misma computadora)	-	R	R	-/AR
Redundancia diversa (implementando la misma especificación de requisitos de seguridad de software)	-	-	-	R
Redundancia funcionalmente diversa (implementando diferentes especificaciones de requisitos de seguridad de software)	-	-	R	AR

Inhibir el efecto de resultados incorrectos. Basado en redundancia y diversidad. Aplicarse al nivel de subsistema más pequeño (más detallado).

Códigos de Hamming, códigos polinomiales

Ver slides anteriores

Computadora independiente con otra especificación que se ocupa de que el sistema principal no alcance estados inseguros o corregirlo si lo hace.

Clase 4

Clase 4

Técnicas para diseño de arquitectura (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Recuperación hacia atrás	R	R	-	NR
Diseño de software sin estado (o diseño de estado limitado)	-	-	R	AR
Mecanismo de reintentar ante fallas	R	R	-	-
Degradación elegante	R	R	AR	AR
Inteligencia artificial - corrección de fallas	-	NR	NR	NR
Reconfiguración dinámica	-	NR	NR	NR
Uso de software confiable/verificado (si está disponible)	R	AR	AR	AR
Enfoque modular	AR	AR	AR	AR

Si se detecta una falla, el sistema pasa a un estado interno anterior, cuya consistencia ya fue probada.

Minimizar la complejidad del comportamiento del software evitando o minimizando la cantidad de estados.

Ante fallas, re ejecutar el código, ejecutar un reinicio de rutinas o tareas. Común en telecomunicaciones.

Si no hay recursos para todas las funciones, las de mayor prioridad se llevan a cabo con preferencia a las inferiores.

Pronóstico y corrección de fallas, mantenimiento predictivo y acciones de supervisión respaldados por **IA**.
(Evitar modelos pre pensados)

Asignar al sistema una parte de los recursos, si colapsa, derivar el sistema a la otra parte.

Analizar si el software ha sido usado en sistemas críticos y si cumple esta norma.

Ver slides anteriores

Técnicas para diseño de arquitectura (III)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
	Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y la arquitectura de software	R	R	AR	AR	Ver slides anteriores
	Trazabilidad hacia atrás entre los requisitos de seguridad del sistema y la arquitectura de software	R	R	AR	AR	Ver slides anteriores
	Métodos esquemáticos estructurados	AR	AR	AR	AR	Dividir el problema, documentar todo, atomizar funciones, listas de comprobación, subsistemas simples, etc.
	Métodos semiformales	R	R	AR	AR	Ver slides anteriores
	Métodos formales de diseño y refinamiento	-	R	R	AR	Ver slides anteriores
	Generación automática de software	R	R	R	R	Convertir en código un modelado en alto nivel. Elimina tareas manuales de codificación propensas a errores. Diseños más complejos a un nivel abstracto superior.
	Herramientas de diseño y especificación asistidas por computadora	R	R	AR	AR	Ver slides anteriores

Técnicas para diseño de arquitectura (IV)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Comportamiento cíclico, con tiempo de ciclo máximo garantizado	R	AR	AR	AR
Arquitectura activada por tiempo (TTA: Time-triggered architecture)	R	AR	AR	AR
Orientación a eventos (con tiempo de respuesta máximo garantizado)	R	AR	AR	-
Asignación de recursos estáticos	-	R	AR	AR
Sincronización estática de acceso a recursos compartidos	-	-	R	AR

Basado en una base de tiempo sincronizada globalmente. Transparente a tolerancia a fallos, muy recomendable para sistemas críticos.

Las actividades del sistema se desencadenan por eventos arbitrarios en momentos impredecibles.

Evitar uso de memoria dinámica

Evitar uso de memoria dinámica

Técnicas para diseño e implementación (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Métodos estructurados	AR	AR	AR	AR
Métodos semiformales	R	AR	AR	AR
Métodos formales de diseño y refinamiento	-	R	R	AR
Herramientas de diseño asistidas por computadora	R	R	AR	AR
Programación defensiva	-	R	AR	AR

Ver slides anteriores

Ver slides anteriores

Ver slides anteriores

Ver slides anteriores

Verificar rango, tipo, dimensión, validez y plausibilidad (si son físicas) de variables, existencia de HW y accesibilidad, completitud del SW, etc. **Observación de las salidas**.

Técnicas para diseño e implementación (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Enfoque modular	AR	AR	AR	AR
Estándares de diseño y codificación	R	AR	AR	AR
Uso de software confiable/verificado (si está disponible)	AR	AR	AR	AR
Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y el diseño del software	R	AR	AR	AR
Programación estructurada	R	R	AR	AR

Ver slides anteriores

Estas reglas están diseñadas para facilitar el desarrollo, la verificación, la evaluación y el mantenimiento.

Ver slides anteriores

Ver slides anteriores

Módulos pequeños y simples con interacciones explícitas. Flujo de control con pocos caminos y restricciones según las entradas. Evitar ramificaciones en base a cálculos complejos y saltos incondicionales (**goto**).

Técnicas para testing e integración (detalles) (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Ensayo probabilístico	-	R	R	R
Análisis y ensayos dinámicos	R	AR	AR	AR
Registro y análisis de datos	AR	AR	AR	AR
Ensayos de caja negra	AR	AR	AR	AR
Ensayos de rendimiento	R	R	AR	AR

Estimación estadística de la fiabilidad del software: probabilidad de falla bajo demanda (**PFD**), probabilidad de falla durante un cierto período de tiempo y probabilidad de contención de errores (**CFP**).

Someter un prototipo a datos de entrada previstos. Es satisfactorio si el comportamiento observado se ajusta al requerido. Corregir y volver a analizar en caso contrario.

Detallar: Pruebas realizadas en cada módulo, decisiones y justificaciones, problemas y sus soluciones.

Ver slides anteriores

Ensayos de carga, ensayos de estrés, ensayos de resistencia, ensayos de picos, ensayos de volúmen, ensayos de escalabilidad.

Técnicas para testing e integración (detalles) (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Ensayos basados en modelos	R	R	AR	AR
Ensayos de interfaz	R	R	AR	AR
Herramientas de automatización y gestión de pruebas	R	AR	AR	AR
Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y el módulo y las especificaciones de ensayos de integración	R	R	AR	AR
Verificación formal	-	-	R	R

Los ensayos se aplican sobre el modelo

Combinaciones de valores extremos y normales en las variables internas y de interfaz.

Ver slides anteriores

Ver slides anteriores

Requiere modelo abstracto del sistema y su comportamiento en un lenguaje con significado matemático preciso: **CSS**, **CSP**, **HOL**, **Raise**, **Z**, etc.

Técnicas para integración de electrónica programable

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Ensayos de caja negra	AR	AR	AR	AR	Ver slides anteriores
Ensayos de rendimiento	R	R	AR	AR	Ver slides anteriores
Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y los requisitos de diseño de SW para la integración de HW/SW y las especificaciones de prueba de integración de HW/SW	R	R	AR	AR	Ver slides anteriores

Técnicas para validación de seguridad

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Ensayo probabilístico	-	R	R	AR	Ver slides anteriores
Simulación del proceso	R	R	AR	AR	Probar el sistema de software, junto con su interfaz con el mundo exterior, sin permitirle modificar el mundo real.
Modelado	R	R	AR	AR	Ver slides siguientes
Ensayos de caja negra	AR	AR	AR	AR	Ver slides anteriores
Trazabilidad hacia adelante entre los requisitos de seguridad del sistema y el plan de validación de seguridad del software	R	R	AR	AR	Ver slides anteriores
Trazabilidad hacia atrás entre el plan de validación de seguridad del software y la especificación de requisitos de seguridad del software	R	R	AR	AR	Ver slides anteriores

Técnicas para modificación (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Análisis de impacto	AR	AR	AR	AR
Reverificación del módulo de software cambiado	AR	AR	AR	AR
Reverificación de los módulos de software afectados	R	AR	AR	AR
Revalidación completa del sistema	-	R	AR	AR
Validación de regresión	R	AR	AR	AR

Analizar efecto de un cambio/mejora en un subsistema en otros subsistemas y en el sistema total.

Requiere mucho esfuerzo y recursos. Usarlo de forma restringida.

Técnicas para modificación (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Gestión de configuración de software	AR	AR	AR	AR	Documentar la producción de cada versión de cada entregable significativo y de cada relación entre las diferentes versiones de los diferentes entregables.
Registro y análisis de datos	AR	AR	AR	AR	Ver slides anteriores
Trazabilidad hacia adelante entre la especificación de los requisitos de seguridad del software y el plan de modificación del software (incluida la reverificación y la revalidación)	R	R	AR	AR	Ver slides anteriores
Trazabilidad hacia atrás entre el plan de modificación del software (incluida la reverificación y la revalidación) y la especificación de los requisitos de seguridad del software	R	R	AR	AR	Ver slides anteriores

Técnicas para verificación

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Verificación formal	-	R	R	AR	Ver slides anteriores
Animación de especificación y diseño.	R	R	R	R	Animar interfaz de usuario para que aquellos sin conocimientos técnicos puedan probar el sistema con el que trabajarán.
Análisis estático	R	AR	AR	AR	Buscar objetos que **no** mantengan su valor, código **no accesible**, **fugas** de memoria, etc.
Análisis y ensayos dinámicos	R	AR	AR	AR	Ver slides anteriores
Trazabilidad hacia adelante entre la especificación de diseño del software y el plan de verificación del software (incluida la verificación de datos)	R	R	AR	AR	Ver slides anteriores
Trazabilidad hacia atrás entre el plan de verificación del software (incluida la verificación de datos) y especificación de diseño del software	R	R	AR	AR	Ver slides anteriores
Análisis numérico sin conexión	R	R	AR	AR	Errores de **truncamiento**, **redondeo**.

Técnicas para evaluación de seguridad funcional

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Listas de verificación	R	R	R	R
Tablas de decisión / verdad	R	R	R	R
Análisis de fallas	R	R	AR	AR
Análisis de fallas de causa común de software diverso (si se usa software diverso)	-	R	AR	AR
Diagrama de bloques de confiabilidad	R	R	R	R
Trazabilidad hacia adelante entre los requisitos de la evaluación de seguridad funcional y el plan para la evaluación de la seguridad funcional del software	R	R	AR	AR

Ver slides anteriores

Describiendo circuitos de compuertas lógicas.

Analizar modo de falla de cada componente, causas y efectos (locales y generales), formas de **detección** y **recomendaciones**. Documentar medidas correctivas tomadas.

Clase 4

Modelar, en forma de diagrama, el conjunto de eventos que deben tener lugar y las condiciones que deben cumplirse para el funcionamiento exitoso de un sistema o una tarea.

Ver slides anteriores

Estándares de diseño y codificación (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Uso de estándar de codificación para reducir la probabilidad de errores	AR	AR	AR	AR
No usar objetos dinámicos	R	AR	AR	AR
No usar variables dinámicas	-	R	AR	AR
Comprobación de la instalación de variables dinámicas	-	R	AR	AR
Limitar el uso de interrupciones	R	R	AR	AR

Evitar variables globales y GoTo. **Usar** encabezados estándar para módulos, convención uniforme de nombres, nombres significativos, indentación, funciones cortas, retornos y manejo de errores.

No se puede predecir el uso de memoria con precisión mediante algún análisis estático (antes de la ejecución del programa), **tampoco** se puede garantizar la ejecución predecible del programa.

Idem anterior

Usar solo si simplifican el sistema. Anular en **zonas críticas** de tiempo calculado **acotado**. **Documentar** las interrupciones.

Estándares de diseño y codificación (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

	Técnica	SIL 1	SIL 2	SIL 3	SIL 4
	Limitar el uso de punteros	-	R	AR	AR
	Limitar el uso de recursividad	-	R	AR	AR
	Sin flujo de control no estructurado	R	AR	AR	AR
	Sin conversión de tipo automática	R	AR	AR	AR

Verificar tipo y rango antes. La comunicación entre tareas **no** debe realizarse por referencia. El intercambio de datos **debe** realizarse a través del sistema operativo.

Si se utiliza la recursividad, debe haber un criterio claro que haga predecible la profundidad de la recursividad.

Análisis dinámico y testing

Menti_D_1

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Ensayos de análisis de valor límite	R	AR	AR	AR
Ensayos de adivinación de errores	R	R	R	R
Ensayos de siembra de errores	-	R	R	R
Ensayos basados en modelos (Model-based Testing (MBT))	R	R	AR	AR
Modelado de desempeño	R	R	R	AR
Ensayos de clases de equivalencia y partición de entrada	R	R	R	AR
Ensayos de cobertura estructurales (puntos de entrada / declaraciones / sucursales / condiciones, MC / DC) 100%	AR/R/R/R	AR/AR/R/R	AR/AR/AR/R	AR

Valores máximos/mínimos, división por cero, caracteres ASCII nulos, pila o lista vacía, matriz/cola llena, entradas nulas, etc.

¿Y si presiono los botones muy rápido o muy seguido? ¿Qué pasa si los presiono simultáneamente?

Algunos tipos de errores conocidos se insertan (siembran) en el programa y se ejecuta en **modo de prueba**.

Tablas de verdad, FSM, cadenas de Markov, diagramas de estados, diagrama de flujo, autómatas finitos, redes de Petri, etc.

Determinar para cada función el rendimiento, uso de recursos por cada proceso (tiempo de procesamiento, ancho de banda, etc) en condiciones promedio y peor caso.

Probar el software usando un mínimo de datos de prueba, seleccionando las particiones del dominio de entrada necesarias.

Si no es posible cobertura del 100%, documentar las razones.

Ensayos de caja negra

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Ensayos de diagramas de causa y consecuencia	-	-	R	R
Ensayos basados en modelos (Model-based Testing (MBT))	R	R	AR	AR
Prototipos / animación	-	R	R	R
Ensayos de clases de equivalencia y partición de entrada	R	AR	AR	AR
Simulación del proceso	R	R	R	R

La técnica se puede considerar como una combinación de árbol de fallas y análisis de árbol de eventos.

Ver slides anteriores

Acotar el sistema y armar un prototipo de alto nivel. **No** hay que considerar plataforma, lenguaje, capacidad, confiabilidad, disponibilidad. Lo evalúa el cliente y **pueden modificarse requisitos** a posteriori.

Ver slides anteriores

Ver slides anteriores

Análisis de fallas

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Diagramas de causa y consecuencia	R	R	R	R	Ver slides anteriores
Análisis de árbol de eventos (Event tree analysis (ETA))	R	R	R	R	Diagramar la secuencia de eventos que pueden desarrollarse en un sistema para indicar qué consecuencias graves pueden ocurrir.
Análisis de árbol de fallas (Fault tree analysis (FTA))	R	R	R	R	Método gráfico de símbolos estandarizados que representa la función lógica que vincula las fallas de componentes con la falla general del sistema
Análisis de fallas funcionales de software	R	R	R	R	Ver slides anteriores

Modelado

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Diagramas de flujo de datos	R	R	R	R
Máquinas de estados finitos (Finite state machines, FSM)	-	R	AR	AR
Métodos formales	-	R	R	AR
Redes de Petri temporales	-	R	AR	AR
Modelado de desempeño	R	AR	AR	AR
Diagramas estructural	R	R	R	R
Prototipos / animación	R	R	R	AR

Muestra como la **entrada** de datos se transforma en **salida**, con cada etapa del diagrama representa una transformación distinta.

Debe tener: **integridad** (una acción y un estado por entrada/estado), **coherencia** (una transición por estado/entrada), **accesibilidad** (posibilidad de recorrer todos los estados) y **ausencia de bucles** sin fin o **estados sin salida**.

Ver slides anteriores

Red de nodos (marcados o no) y transiciones para flujo de control. Las marcas "circulan" por la red en función de **transiciones** (temporales o enable/disable) y entradas. Eficiente para **Monte Carlo** en cálculo de probabilidad de falla.

Modelo de procesos e interacciones. Determina uso de recursos por proceso, distribución de la demanda y rendimiento medio en condiciones promedio y peor caso.

Complementa los **diagramas de flujo de datos**. Describen el sistema de programación y una jerarquía de partes como un árbol. Muestra relaciones entre los módulos sin el orden de activación.

Ver slides anteriores

Ensayos de rendimiento

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Ensayos de avalancha / estrés	R	R	AR	AR
Limitaciones a los tiempos de respuesta y memoria	AR	AR	AR	AR
Requisitos de desempeño	AR	AR	AR	AR

Si funciona por pooling o bajo demanda, inyectar muchas entradas rápidamente. Dispositivos influyentes a su máxima/mínima velocidad. Factores influyentes en sus condiciones límites.

Requiere estimaciones del uso de recursos y el tiempo de cada función. Por ejemplo, mediante la comparación con un sistema existente o prototipos.

Se realiza un análisis comparativo del sistema y de las especificaciones de requisitos.

Métodos semiformales (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Diagramas de bloques lógicos / funcionales	R	R	AR	AR	Ver slides anteriores
Diagramas secuenciales	R	R	AR	AR	Ver slides anteriores
Diagramas de flujo de datos	R	R	R	R	Ver slides anteriores
Máquinas de estados finitos (Finite state machines, FSM)	R	R	AR	AR	Ver slides anteriores
Redes de Petri temporales	R	R	AR	AR	Ver slides anteriores

Métodos semiformales (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Modelos de datos entidad-relación-atributo	R	R	R	R
Diagramas de secuencia de mensajes	R	R	R	R
Tablas de decisión / verdad	R	R	AR	AR
Lenguaje de modelado unificado (Unified Modeling Language,UML)	R	R	R	R

Ayudar al usuario a escribir una buena especificación centrándose en las entidades dentro del sistema y las relaciones entre ellas.

Describe el comportamiento en términos de la comunicación que tiene lugar entre los actores del sistema (ser humano, elemento u objeto de software, etc).

Ver slides anteriores

Diagramas de clases, casos de uso, diagramas de actividad, diagramas de transición de estado (Statecharts), diagramas de secuencia del sistema.

Análisis estático (I)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Análisis de valor límite	R	R	AR	AR	Ver slides anteriores
Listas de verificación	R	R	R	R	Ver slides anteriores
Análisis de flujo de control	R	AR	AR	AR	Ver slides anteriores
Análisis de flujo de datos	R	AR	AR	AR	Ver slides anteriores
Adivinación de errores	R	R	R	R	Ver slides anteriores
Inspecciones formales (incluidos criterios específicos)	R	R	AR	AR	Ver slides anteriores

Análisis estático (II)

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4
Walk-through (software)	R	R	R	R
Ejecución simbólica	-	-	R	R
Revisión de diseño	AR	AR	AR	AR
Análisis estático del comportamiento de error en tiempo de ejecución	R	R	R	AR
Análisis del tiempo de ejecución en el peor caso	R	R	R	R

Para revelar **discrepancias** entre la especificación y la implementación.

Para mostrar **coincidencias** entre el código fuente y la especificación.

Revelar defectos en el diseño del software.

Ver slides anteriores

Ver slides anteriores

Enfoque modular

O: Obligatorio
AR: Altamente Recomendado
R: Recomendado
NR: No Recomendado

Técnica	SIL 1	SIL 2	SIL 3	SIL 4	
Límite de tamaño del módulo de software	AR	AR	AR	AR	Ver slides anteriores
Control de complejidad de software	R	R	AR	AR	Ver slides anteriores
Ocultación / encapsulación de información	R	AR	AR	AR	Ver **modularización**
Límite de número de parámetros / número fijo de parámetros de subprograma	R	R	R	R	Ver slides anteriores
Un punto de entrada / un punto de salida en subrutinas y funciones	AR	AR	AR	AR	Ver slides anteriores
Interfaz completamente definida	AR	AR	AR	AR	Ver slides anteriores

Menti_D_2

¡Muchas gracias!

¿Alguna pregunta?