# Exercise A

*List your 3 top strategies for how to control compatibility between objects (HW or SW) as well as backwards and forward compatibility.*

Gruppernas svar i korthet:

* Parent-child relationer räcker inte. Krävs relationsdatabas som kan hålla reda på fler beroenden.
* Inför intelligenta enheter som kan presentera sig och vet vad de behöver och var de skall instantieras.
* Styr med namnsättning av versioner. Byt artikelnr om compatibiliten bryts.
* Inför versioneringsregler: när man bryter bakåtkomp på HW får man byta artikelnummer
* Tydliggörande av processer.
* Standardisera interface och inför modularisering (som lego)
* Håll koll på interfacen, dessa bör hanteras som Configuration items, beakta:
	+ Funktionsnivå med interface
	+ Systemnivå med interface
	+ Realiserings (engineering) med interface
	+ Produktionsnivå med interface
	+ Funktionsspecar och interfacespecar
* Det krävs en stark SW Arkitektur som inte är så funktionsberoende utan är service orienterad
* Inför Semantic versioning (Semver.org) dynamiska strukturer, styr om interfaceförändring förändrar överordnad produkt. Kräver systemarkitektur som styr CM systemet.
* Inför Parameterstyrd variant-konfigurering
* Kompatibilitiet med kunden/mottagaren, vi bör se intressentperspektivet och beräkna risk

# Exercise B

*List your 3 top CM-strategies for the ”Robot case”: “When some change in the electronics needs major changes in SW, we define a new platform, which is a kind of label. The platform number is what decides which SW release will be loaded onto the robot. OTA updates check the compatibility before attempting to load a given release onto the robot. Each platform is built separately when a release is made.*

*Individual robots of the same platform may have differences which can be handled by configuration files. E.g. type of charger. These files define a group of attributes which can customise the behaviour of a robot. These can be set individually or in groups at production, to create different "variants" of a platform, or possibly while the robot is in the field, if for example we were to enable individual SW functions as aftermarket updates. The configuration files for different variants are applied after the platform-appropriate SW is installed on a robot.*

*We want as few active branches of code as possible, so if two platforms can be supported from a single branch, this is good. If two platforms are so different that maintaining a single branch leads to difficult (to maintain or write) code, we can create a new branch for one platform. This means there is no hard rule- it is a case of individual judgement by the team.*

*Some HW (electronic and mechanical) changes are necessary during the life of the robot- quality issues, end-of-life, etc. As far as possible, we aim to make these backward compatible, when it comes to SW. If they cannot be made compatible, then a new platform (or at least a variant attribute) needs to be defined so the SW knows how to act on this HW.*

*Occasionally a new platform or variant will need to be handled differently during production at the factory. Sometimes the differences require us to change the suite of SW we provide the manufacturer to configure and calibrate the robot. This suite is known as PT (production test) SW, and to produce a robot with a given SW means that only certain PT versions can be used.”*

Gruppernas svar i korthet:

* Branchning bör inte vara personberoende, inför dokumenterad strategi/regelverk för branchning
* Begränsa varianthanteringen
* Viktigt att känna till på vilka faktorer man fattar besluten på, det är inte alltid ett tekniskt beslut utan även ett affärsbeslut
	+ Detta kan bero på vad produktägaren har för kommersiellt ansvar
	+ Finns med i agile manifesto
* Hållbarhetsmässigt och långsiktigt återanvändningstänk
* Det ärinte hållbart att använda brancher för olika versioner (detta är vanligt men funkar inte så bra när varianterna blir fler)
	+ Då dör aldrig brancherna
	+ Modularisera istället
	+ Matriser i form av baselines där vi håller ihop hela produkten
* Det måste vara möjligt att läsa av HW-versionen på varje kort
* Konfigfil är bra för flexibilitet
* CM bör se till helheten, vilka områden som påverkas av en ändring
* Klargör alla faktorer som bör påverka beslut, säkerställ att dessa ingår i beslutsforumet (Kombinera CM med affärsbeslut)
* Versionsnummer-system där kompatibiliteten framgår
* Hålla ihop med baselines där ni plockar hela moduler
* Jobba kontinuerligt med akritektur och arkitektur-roadmappen
* Arbeta emot Single branch strategi av software, sträva efter så få brancher som möjligt
* Utgå ifrån en livscykelmodell där man ärver med sig information från tidigare fas vilken kontinuerligt underhålls och utvecklas (se bilder)
	+ ISO/IEC 15288 Livscykelprocesser fö system
	+ ISO/IEC 12207 Systems and SW engineering





# Exercise C

*List your 3 top CM-strategies for the user documentation: Robot case part II: “Originally there was one version of the robot. The second one shares most of its construction with the first one. Do we:*

*- make two sets of documents, containing much duplicated information (hard to maintain)*

*- make one set of documents, containing information for both robots (hard to read and very hard to manage when robot number 3 comes along)*

*- try to split them so that there are common documents, and a few which are specialised. This is what we chose, but it will also be hard if/when robot 3 comes along and it shares some parts from number 1 and other parts from number 2. The "interfaces" between the documents become very hard to manage.”*

Gruppernas svar i korthet:

* Gör en Informations analys och arkitektur. Förslag på hantering i existerande verktygsmiljö:
	+ Bryt ner informationen i mindre delar
	+ Splitta dokumenten i generiska delar och specifika delar (variant specifika).
	+ Länka från produkterna i Teamcenter till dokumenten. Kopiera aldrig information så det behöver uppdateras på fler ställen än ett.
* Förslag på uppdatering av verktygsmiljö: Använd ett  verktyg som stödjer hantering av informations objekt/moduler för textdokument. Dokument kan sedan genereras fram vid behov (t ex pdf:er)
* Modularisering med Xml och S1000D – då kan man använda ”applicability”. Se tex Combitechs ”uptime”
* Använd verktyget ”Mark-down” (för kodnära dokumentation) hänger med kodes brancher
* Specifika eller gemensamma block taggade med tillhörighet
* Onlinedokumentation om möjligt kopplad till den SW-version man kör
* Model based engineering
* För mjukvara kan man checka in dokumentationen tillsammans med koden för att versionerna ska följa varandra
* Hantera information inklusive underlagen i källsystemen. Producera dok nära källan.

# HW dev Question 1

*Problem: We need to release non-verified designs*

*Question: What is the proper way to handle this deviation from CM point of view?*

*Process clearly say we need to test before release but this does not always work because of …*



Gruppernas svar i korthet:

* Inför mognadsklassning
* Att ta reda på resultat ifrån testning är en projektfråga
* Varje version bör ha sin livscykel
* Releaseplanering i högre utsträckning
* Större förserie så sluttest kan göras tidigare
* Ny plan skall göras vid leverantörsbyte -> tydlighet vad som får ändras när
* Hantera orealistisk tidplan/ riskanalys
* Tvinga ihop leverantörerna i samma rum

# HW dev question 2

*Problem:*

* *Rev D is in production (released in Teamcenter), Rev E under update (not released)*
* *Difference D->E is updated Layout (PCB) and updated component values (BOM) – HW update only*
* *Alternative components requested for Rev D by supplier but Teamcenter does not allow update until Rev E is released*
* *Rev E is blocked by pending prototype build, pre-series and long test time until it can be released.*

*Question:*

*What is the best way to handle this kind of deviation that is not supported by PLM system, from a CM perspective?*

*Input – We cannot change Teamcenter setup, Supplier request BOM to be updated or production will stop, All alternative components are verified and approved by Electrolux. MASA audit will look at this…*

Gruppernas svar i korthet:

* Göra en minor revidering. Dvs gör en d1 under det att e lever kvar
* Ändra processen och implementera i verktyget. Ericsson har gjort det, har flera releaser på gång samtidigt.
* Modularisera och i mindre block, då möjliggörs fler ändringar
* Hantera med en deviation-process
	+ Skapa en deviation i systemet som man sparar (ett medvetet beslut) och knyt i teamcenter
	+ Ändra sedan i BOM i SAP, eller supplier som säkerligen har något CM-stöd
* Ta fram en strategi och process för hantering av alternativa componenter. Förslag på hantering i existerande verktygsmiljö:
	+ Suppliern skriver en Deviation Request som Electrolux granskar och godkänner.
	+ Den godkända ”Deviation” knyts till produkten i Teamcenter (Vi antar att man kan addera dokument till en releasad produkt)
	+ BOMen i SAP uppdateras enligt ”Deviation”.
* Förslag på uppdatering av verktygsmiljö: Möjlighet att ”brancha” i Teamcenter, dvs göra en version D1 samtidigt som E inte är releasad.