Summary of discussion: systems engineering and CPS complexity

- We need to find out essence of complexity: is it about heterogeneity, humans, abstractions, numbers of components…? Only when we understand it fully, will we be able to come up with solutions.
- "Subsystems" is purely organisational issue – the organisation chart therefore has great impact on product structure
- What's the scope of "systems engineering"? Systems engineering is a collection of best practices to address some of these [above] concerns
- In systems engineering we focus on abstraction – it means that it's not the real thing – but there are issues when we refine. How to integrate effectively?
- Systems engineering is great for decomposition, but re-composing is not the direct inverse of this – putting them back together results in problems. There's something missing from SE to help us cope with this
- You can find a mapping between complexity and best practice – but now we merge many things together, including cross-cutting aspects. Can we just merge lots of best practices to get a new "global" best practice? Probably not. There is too much friction to combine things – e.g.. conflicting requirements, uncertainty about too much in the project
- Do we commit too early to designs? Should we explore tradeoffs more?
  We need to get the architecture in as good a shape as early as possible. Requests only start to be generated as the customer uses the products. Hardware is rigid and safety requirements mean it's difficult to change. Also there's an increasing focus on cyber-security and quick response to threats. This directly conflicts with functional safety, which is always a key focus for CPS. There are no fixed processes for how to solve these issues – resolving security with safety (for example). The more we go into automation the problem gets bigger – demand for more and more function, but conflicts with important cross-cutting performance such as safety.
- Ownership is really important behind the new business models; if a system is not owned there is no body of experience behind it, who is really accountable?

Communication

- There are lots of disciplines which don't communicate – functional safety, systems, cybersecurity etc. Traditionally they talk to each other very little. In fact, this problem is so huge it could be potentially unsolvable?
- We need to decompose the problem into component parts – addressing views of cross-cutting issues such as reliability, safety etc. We need to use this approach to decompose a wicked problem. But what's best practice for each of these? This will change for different domains, but at least we could select a challenging domain as a an example in order to provide a benchmark for ourselves. This will help identify the best technology to address that feature.
- Allowing disciplines to communicate is a key problem. What are some ways to improve sharing? E.g., one Swedish manufacturer of trucks insists all software developers have to have a truck license, in order to understand the truck drivers' job and environment and interactions with the system. It permits full system evaluation. This is also related to the question of a validity frame: we have very many models – but we need to store information about the context in which the model is valid. Without this we could end up with

models/components being redeployed or reused in inappropriate contexts and environments, that could have been avoided if this information was captured somewhere.

- The concept of an “intelligent middleman” is very important – we will always need someone who can step in quickly when needed. Although, what is a middle-man? Is this a person who has expertise in everything, e.g., in computer science, physics, electronics, safety etc? Or should we concentrate on middleman as a human operator who has the power to override automated CPS when necessary?
- It’s important to share the big picture. There is major fragmentation across the many different domains. It would be great to have a unified model or reference model. This is partly possible – e.g., in energy, it’s about energy flow. In telecoms, it’s about data flow. Should this be enforced?
- How do you tailor or adapt generic frameworks to CPS? It needs to be feasible to add something to the framework from time to time, so that cross-cutting concerns (e.g., safety) can be added to a framework. Frameworks are needed but must be extensible

Dependability

- Dependability is a problem as complexity increases. We can’t prove something is definitely dependable and we move to dealing with probabilities.
- Some systems cannot be designed deterministically – it depends on the specific environment. E.g., robotics can be designed in a deterministic way, but telecommunications can’t (because we can’t be overly deterministic about response time or SLA conformance).
- Look at the example of train failure caused by insufficient transponders installed outside stations. At design time the assumption was made that train drivers will slow down as they enter a station, but in one case the driver was over the legal alcohol limit and did not, and the lack of transponders meant that there was no failsafe to override him. Here finances play a role (don’t buy transponders to save money) and also human failures contribute.
- In fact, many major failures – Fukushima, Chernobyl, aviation crash, etc. can be attrited to human errors. We need to prevent humans violating safety state. It’s a tricky question: should human be able to override system? We make assumptions about reasonable human behaviour - we need more high level people taking a whole systems view.
- To resolve the problem of designing for human failures – we need a feedback loop (enabled by CPS) to examine human behaviour.

Evolution & long lifecycles

- Agile practice can stifle innovation – agile teams tend to focus on getting things done, not innovating.
- Are we equipped to handle DevOps style updates? CPSs are rarely “clean-sheet” developments, updates are deployed to live systems. Our design practices need to be adapted for change. Demand for new functions grows so rapidly that we basically overdesign existing architecture. This is very unpredictable and gives us very little forward view – e.g., in automotive industry, there’s a need to accommodate changes over the lifetime of the same architecture.
- We need to design for change. But safety standards and process standards do not support this.
- Difficulty of integration & change requests: traditionally in systems engineering an organisations might have requested all the subsystems to approve a change request, in order to check and validate impact across the system. That’s impossible if you want to do it quickly

- Dealing with legacy systems is a problem – no one knows or understands the old system, models don't exist. Modelling libraries, patterns etc could help but don't exist; companies don't want to share what they have