Organisation and Communication

The pace and flexibility of development possible in Software Intensive Systems requires an organisation to match. From initial customer contact, through requirements capture, prototype, production solution, delivery, customer training, support, maintenance and obsolescence, communication flow and unambiguous shared understanding is key.

As more of the system is left to be defined in software (allowing more flexible programmes, delaying the point at which a design has to be committed, and enabling more arbitrary relationships to be defined) so the system becomes more complex. With many businesses looking for improved speed of development, or a shorter time-to-market, as a competitive advantage there is an increasing need to get the solution right first time. As products take on more safety/security emphasis, the development requires more rigour and robustness – and evidence to support

This requires 'migration' of key skills across a number of boundaries at different phases of development. The final system solution is still dependent on carefully crafted components – but programme performance is dependent on information flow.

Complex systems require Models

To achieve communication that is complete (in as much detail as is necessary to be portrayed, for attributes that are significant to the design) requires a rigour of process and a structured, methodical approach.

For any substantial project, it is necessary to convey this information across many parties (i.e. share the description). In the real world, that description changes for a large portion of the development. System Engineering starts when the customer describes his problem... and probably doesn't finish until the system is taken out of service (but hopefully the effort levels will change over that lifetime)!

For truly large systems, or those with a long service life, it will often require a group of system architects to collaborate, or 'pass the baton' as careers progress. Rigour and Shared understanding, with Rich detail => Modelling. N.B. This may not be necessary if the system is small, or being developed by a pair of domain experienced engineers... but in that case it isn't a big problem anyway!

Concurrent Component Engineering

To enable many parties to start work on 'their part of the solution' needs a well-articulated, clearly partitioned (functional) architecture, with well-defined interfaces (functionally, logically, temporally).

In most real projects, because the system engineering is not a single pass through a design cascade (waterfall model), then you will need that information to be available as a single source, and for the content that is shared to be unambiguous, even when undergoing change (and it will change!).

Again, in the real world the system architect and system engineers will have many rounds of discussions about proposed solutions, the capabilities and performance of the individual components, to achieve an 'optimal' solution.

Even with carefully designed prototypes and trade-studies, the fidelity of the final product cannot be guaranteed until the product is built. At these late development stages, new constraints (e.g. the amount of effort spent on achieving a particular component) may bias any further trades, or accommodation of change.

Abstract Modelling – Deliberately

Most system engineers rely heavily on previous experience, either personal or researched. The ability to re-use or re-purpose information (and experience) from one application to another is VERY high value. With an eye on the future re-use, a system engineer will often attempt to keep his solution abstract. He may detail mechanisms without identifying labels that imply component solutions; functionality without implying a technology. To convey these requirements, yet still leave the implementation 'open' requires the design to be recorded in a sufficiently layered way as to minimise detailed application propagation.

N.B. The above is not limited to Component re-use – it can be applied to analysis, structure, architecture, subsystem, validation, specification, processes, simulation, interfaces…

When describing architectures that apply to portfolios of products (i.e. variant applications having some shared attributes, features, functions, structure, processes etc) it is quite possible that the engineering may be organized as non-aligned phases of development. In these cases it is necessary for the information (model) to be able to be shared at varying points in development (e.g. phased projects, geographically dispersed team, customer specific application variants).

The modeling system must have the ability o output phased, yet coherent, sets of information and yet allow multiple simultaneous developments.

Collaboration – the System Skills in Embedded Systems

By now it should be clear that system engineering is a collaborative effort, requiring many individual and multi-party discussions to arrive at an optimal solution. It is clear therefore that the organization to achieve such an objective must deliberately break-down boundaries between classically described skill groups (e.g. Electronics, Software).

In the very best projects, engineers will migrate from one type of work to another, playing different roles while retaining their specialist domain.

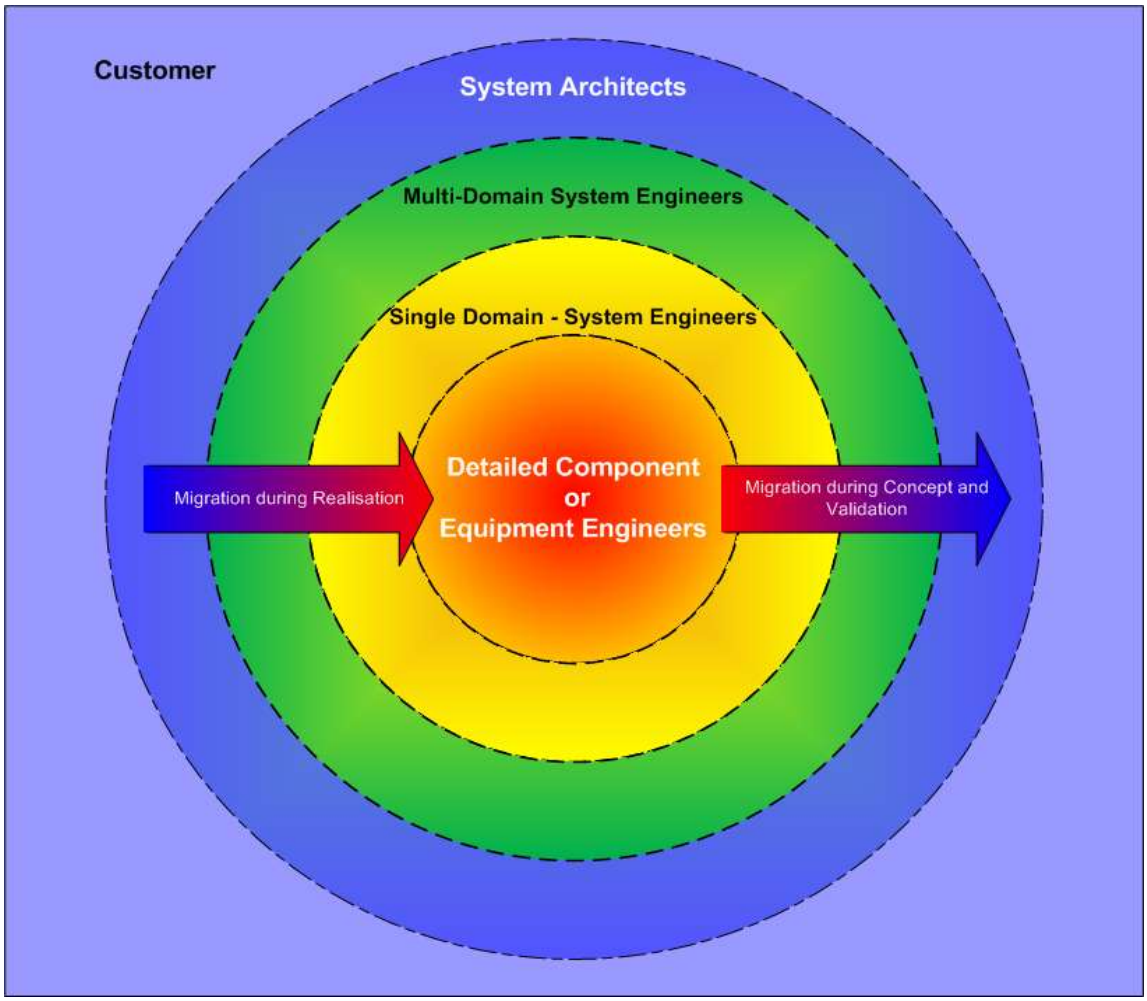
At the outset, more engineers contribute to system engineering; as the engineering progresses through prototyping, concept phases and trade studies, various individuals may experiment locally to contribute capability, performance and flexibility information back to the system architects, or even in demonstrations to the customer.

At the conclusion of component developments, detailed engineers will once again migrate towards systems engineering, through system integration; helping assure that component interface correctly and work together to achieve the expected outcomes.

This migration should not be seen as a denigration of specialization. There will be some engineers who naturally aspire to very niche skills, whilst a number of (initially) specialists, may learn more about aother roles and even product domains and naturally migrate their careers.

There is an increased need for a 'continuous' development method, a passage of information from system architect to component engineer and back to system integration and test, which needs to be a seamless 'embellishment' of information – not re-interpretation.

In general, System engineers are not taught… well at least not much more than the rudimentary methods and methods... but what they do apply is their acquired experience. If you want a good system engineer, look for one who has plied his career in many different domains, acquired some truly awesome skills of abstraction… and is able to hold a topic of conversation with anyone in any walk of life, or specialization!



Please Note that on this diagram none of the boundaries are hard – but should be seen as 'blurred' continuums. Imagine a continuous rainbow colour!

System Architect:

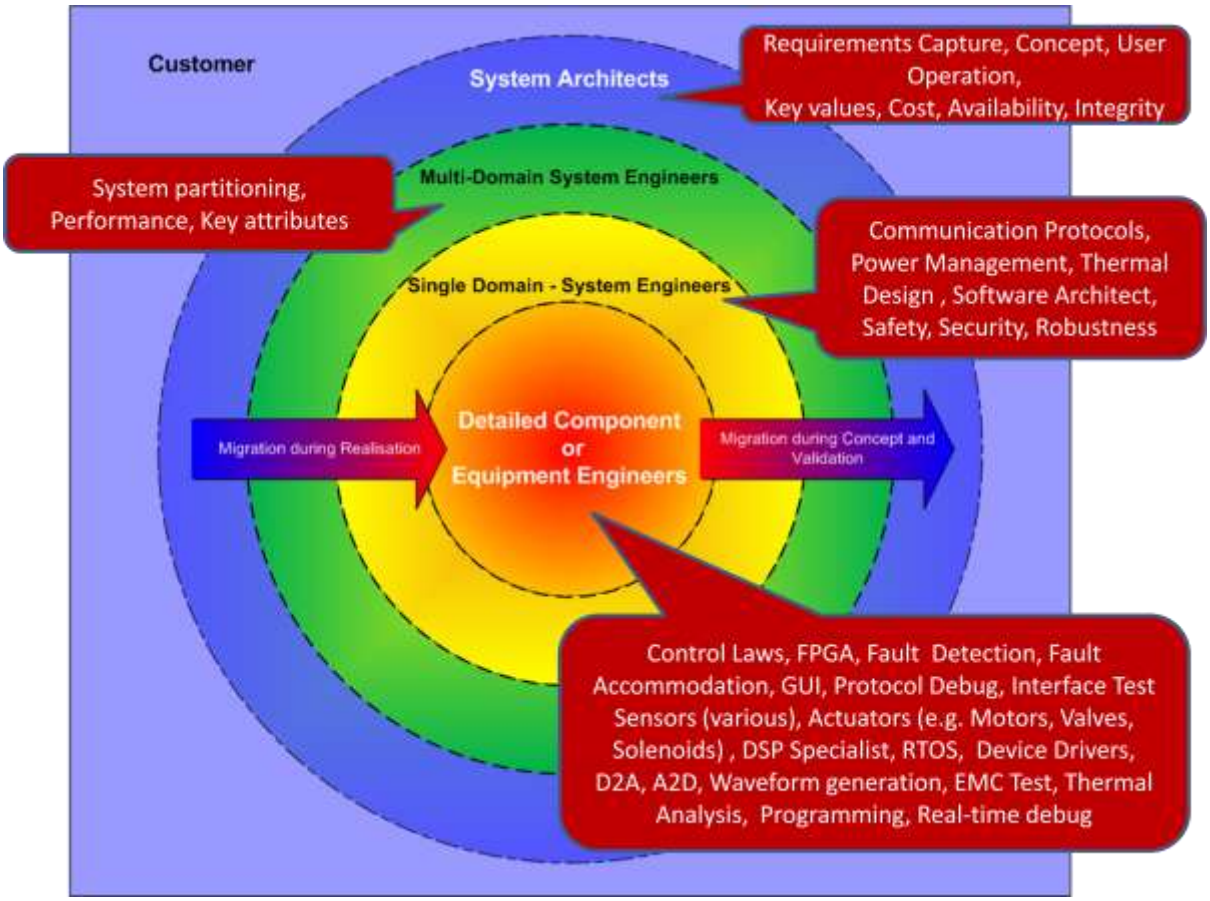
a) Speaks language of the Customer, understand his domain and Application needs
b) Understand the "Art of the Possible" from the multiplicity of specialist system domains at his disposal
c) Is able to speak the language of the system engineer

Systems Engineers

a) Multi-domain engineers – expert across more than 1 domain
b) Single domain engineers – expert in a single domain
c) Able to trade solution options to some suitably optimal set of component requirements/definitions
d) Sees components as potential technologies for solutions to his problem

Component Engineers

a) Specialist in the component implementation
b) Sees system engineering problem as an application of his components
c) E.g. DSP software engineer, FPGA designer, Electric Motor specialist



Please Note that on this diagram none of the boundaries are hard – but should be seen as 'blurred' continuums. Imagine a continuous rainbow colour!

Different Viewpoints

Although we are considering migration skills, each of the classically defined roles carries a different viewpoint, a set of attributes that he is trying to identify, record and assure matches the design intent, or delivers the required functionality.

Component engineers may be abstract from Customer/Application (e.g. Electric Motor designer – may evolve motor for a range of duties rather than singular purpose)

Systems Architects may be abstract from the components (e.g. The specifics of the detailed Electric Motor design) save for the key attributes in that application (e.g. Mass, Volume, Torque delivery, Efficiency)

Viewpoints are not Industry, Business or Application specific

LOCKING component specialist to CUSTOMER DOMAIN may not provide best in class solution.

SHARED KNOWLEDGE of component capabilities in OTHER APPLICATIONS (and their limitations) may enhance solution value/cost.

An electric motor specialist may be able to transfer his knowledge of the optimisation of magnetic circuits, electronic control, electrical waveforms, windings and lay-ups, noise, rotational forces, thermal design, core materials, bearing loads from rail traction motor to rolling mill drive to hybrid electric vehicle.