Systems Integration

White Paper Resulting from Architecture Forum Meeting

November 18, 19, 2014, Kongsberg Group, Kongsberg, Norway

Edited by:

Dr. Gerrit Muller, Buskerud University College, Embedded Systems Institute

Mr. Eirik Hole, Stevens Institute of Technology

Input was provided by the following participants in the Architecture Forum:

Name	Organization
Leandre Adifon	Ingersoll Rand
Jonas Andersson	HBV
Maarten Bischoff	FEI Company
Paola Geromel	Kongsberg Protech Systems
Jan Morten Gimse	Kongsberg Defense
Anders Gustavsson	Mycronic
Pål Hellum	Kongsberg Protech Systems
Eirik Hole	Stevens Institute of Technology
Bjørn Victor Larsen	Kongsberg Defense
Hugo van Leeuwen	FEI Company
Gerrit Muller	HBV-NISE/TNO-ESI

Name	Organization
Jurjen Nicolai	Bosch
Per Erik Nissen	Kongsberg Protech Systems
Teade Punter	TNO-ESI
Solve Raaen	Kongsberg Maritime
Mats Rosling	Mycronic
Martin Simons	Daimler
Andy Turner	Microsoft Devices Group
Egil Vassend	Kongsberg Maritime
Arne Wegger	Kongsberg Defense
Klaas Wijbrans	Philips PINS

Published April, 2015

1. Introduction

Systems Integration is the activity where components, concepts, technologies, and systems are combined to validate (do we build what is needed) and verify (do we build what we promised) early to mitigate risks of uncertainties and unknowns in specification and design.

In practice, many nasty failures pop-up during systems integration, causing delays and budget overruns. Managers, engineers, and academics poorly understand Systems Integration. It is the mirror-side of systems architecting (which they in general poorly understand too). Any unforeseen interaction or dependency across disciplinary or component borders surfaces during systems integration.

2. Experiences with Systems Integration

To get a feeling for the relevance of the topic, we made a quick inventory of experiences in systems integration

- Do you have examples of excellent or disastrous systems integration?

In this group of 21 participants, we had examples of delays of years, cost overruns of tens of millions of dollars, aborted projects, and even a business unit of a company that collapsed. Architects know from experience that systems integration is a painful phase, where often surprises pop-up causing significant delays and cost overruns. Examples of problems that pop-up during systems integration are:

- Customer expectations that were not yet known
- Negotiation of interfaces, while integrating
- Misunderstanding of interfaces
- Poor robustness, due to insufficient design analysis
- No planning for failing test runs
- Late allocation of integration responsibility
- Late confrontation of components, especially of hardware and software
- Poor understanding of environment where the system will operate in

In general, the focus of an organization is dominantly on parts. A system is partitioned in subsystems, subsubsystems, etc. to facilitate the development organization (work break down structure, project organization), purchasing and logistics (structure of Product Data and Life Cycle Management systems (PDM, PLM)), and manufacturing, service, and sales (using PDM and PLM). Partitioning implies interfaces between parts; interfaces get a great deal of attention, since they facilitate distributed work and later assembly.

However, many integration problems come from problems exceeding a single part. Especially, few designers understand the dynamic behavior, where many parts interact. Dynamic behavior is a major source of problems. The qualities of the system (performance, safety, reliability, etc.) are the ultimate concerns of system stakeholders. These qualities typically emerge from the implementation of dynamic behavior and the allocation of behavior to parts. Designers understand qualities even less than the dynamic behavior. Many problems found in integration show up as undesired emerging qualities. Focus of systems integration is therefore on a limited set of (end-to-end) key performance parameters.

We also asked participants to write down what they see as Systems Integration. The answers are in Appendix A.

3. Case Study Smartphones

Microsoft Devices Group, phones unit, presented the integration approach and challenges in the Lumia development. Lumia smartphones appear in many variants and configurations. A single product is actually many different instances of the same system. What is an efficient way to get sufficient coverage? Next challenge is the dynamics of the product once users have bought it. System Integration continues in the market, in the hands of the user. After 1 year of use an end user's devices is quite different from the one that left the factory.. Examples of ongoing integration are software updates, 3rd party applications, Internet services evolution, new devices and accessories. How to ensure forward compatibility of the system, and backward compatibility of new components? Microsoft instruments their software extensively, called telemetry, to support systems integration before and after release.

Vendors build smartphone devices from “off the shelf” components which are used throughout the industry. The components align with the high-level system architecture of the devices but often have a number of smaller incompatibilities on a detailed technical level. Managing these incompatibilities is a repeating challenge when taking frequent new releases of the core functionality from the vendor. Offsets between development cycles of product and platform bring additional challenges. Product developers only know what they really need at the time the platform supplier is done with development and no new functionality or changes can be added. Product integration happens late in the platform development cycle. Critical bugs may be rejected by the platform, as they claim that the platform behaves as agreed originally and the platform cannot accept new requirements found so late in their development cycle. There is a need for fast frequent integration cycles, where reuse of test results is possible. Handling of bugs and latency in handling bugs is challenging across platform product boundaries.

Challenge in the smartphone domain is the pace of innovation and the need for differentiation. This is often in conflict with existing system drivers and architecture: how to handle the consequences of incompatible changes? Most innovation forces changes that break the existing architectures, since architectures tend to be finitely scalable, flexible and extensible.

Microsoft MDG uses several systems architecting and integration approaches:

- API and interface management; fundamental to avoid integration surprises. Part of the interface management approach is releasing of interfaces.
- Standardization; subsystems and interfaces via standards body or de facto.
- Strategic alignment; across companies, as part of an ecosystem. In this case, three large companies Microsoft, MDG phones unit (former Nokia), and the chipset vendor have to align their strategies to minimize systems integration problems later.
- Move from delivering monolithic firmware to delivering robust components
    - In the past, components relied on being tested as part of an embedded system

    - Nowadays, components cannot assume the exact system, hence they must adapt to the environment.
    - Consequently, components have to be tested against a wide range of possible parameters.
- Release early and regularly:
    - Use short, fast release cycles
    - Constantly prove compatibility with the existing system
    - Apply early non-functional testing
- Extend the integration scope to the user by Beta SW and telemetry
    - This is needed, since you cannot know all usage scenarios and environments. In addition, even if you did then you cannot afford to test them all.
    - By instrumenting the system such that you can follow the data while the user uses it, you can use the "field" for integration and test of all (evolving) variants, configurations, applications, and circumstances.
- Automated code merge, where the machine resolves code-line differences originating from various sources; this is fast and (to a point) reliable.

Systems architecture plays a crucial role for systems integration. Its contributions are:

- Definition of the system, its components and its context
- Design the component subsystems to work together (system design, architecture design)
- Manage subsystem interfaces
- Define, manage, and track non-functional requirements
- Guide systems integration to focus on areas with high system complexity, syste level, risk, weakness etc.
- Preempt systems integration problems as early as possible, for example by documentation, models, and mental integration

4. Context and Life Cycle Evolution

The Microsoft MDG case shows clearly that validation of the system depends on the context of the system. In addition, both the context and the system itself evolve over time; consequently, systems integration turns into a continuous activity! We discussed these aspects in a breakout session, using the following questions:

- How to cope with the context of the system?
- How to cope with life cycle evolution in systems integration?

An outcome of the discussion is that design and integration are not consecutive phases. Instead of an instantaneous transition, it is a gradual changeover from design into integration. Integration as activity should start already at the start of a project. A related observation is that in today's world, the system boundaries have become fuzzy, since everything is connected to everything.

Principle 15.1 Systems Integration as activity starts at the beginning of development to identify unknowns and consequences of uncertainties as early as possible. Systems integration includes integration with the context of the system-of-interest.

Principle 15.2 Systems Integration continues after product deployment, since both context and system keep evolving. The system needs instrumentation to support this ongoing integration.

A fundamental challenge is that time and effort available for integration before release and delivery is limited, while the usage space of the system in its context in unlimited. How to achieve a reasonable coverage and how to ensure that the integration team will hit major problems during integration? One suggestion to cope with integration of system and context is to "eat your own food"; use the system internally to increase coverage and to get "real-life" load on the system.

Challenge in the lifecycle evolution is that a company controls some lifecycle aspects, while other aspects are outside the company's control. Some domains, such as defense and electron microscopes have long lifetimes that make obsolescence management an issue. A further

complication is that regulations change during the lifetime, which may have impact on fielded systems.

One team proposes architecture measures to cope with life cycle changes, for example, localization of impact in (upper) layers, modularity, and anticipation of changes for example by roadmapping. Anticipation in architecture immediately triggers a discussion about risks and limitations of anticipation. Some measures make sense and provide actual options for future change. However, there are also numerous examples, where the anticipating measures only added complexity and worsened the situation.

5. Case Study Remote Weapon Station

Kongsberg Group develops and manufactures a range of remote weapon stations that are mounted on top of platforms such as armed vehicles, see Figure 1. The product portfolio contains many variations in terms of weapons, sensors, vehicle integration, and applications. The product itself consists of multiple units that connect via a network. These units integrate into the vehicle (mechanical, electrical, logical). At next level, they integrate further with domain applications and services creating to create desired military capabilities. Finally, the systems are used in a wide variety of circumstances.

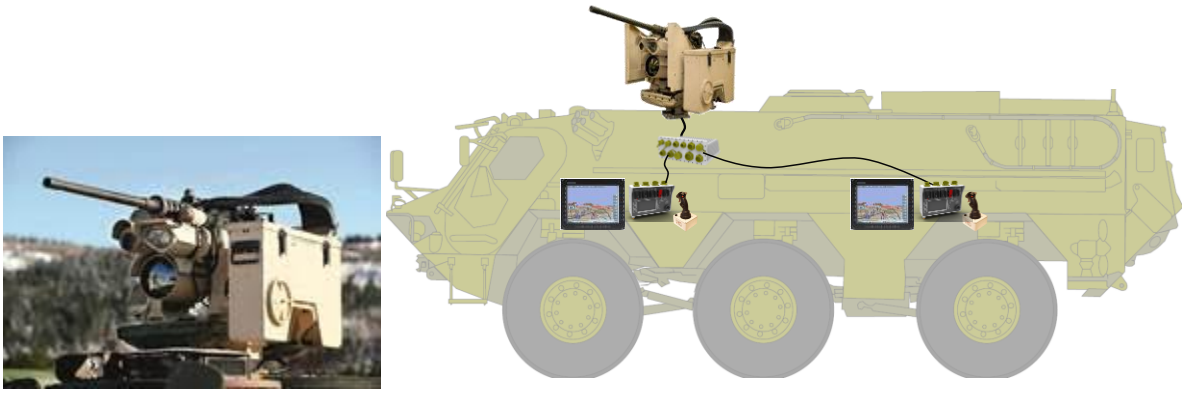


Figure 1. Remote Weapon Station, weapon and optics on top of the vehicle, controls inside the vehicle.

Originally, the development team used a conventional V-model based integration strategy. However, recently, the organization decided to move to an incremental approach. The main drive for this change is an increase of internal and external complexity: more interfaces that are external, more platform (e.g. vehicle) interaction, and more functionality in component-based software.

Figure 2 visualizes the system in layers to explain the logical order of integration. The circled numbers 1 to 8 indicate this logical order. The integration starts with testing the electronics units forming the basis to run any software. Step 2 is adding the software infrastructure, such as operating systems, middleware, and communication. Step 3 integrates the electronics units and the software infrastructure and verifies internal connectivity of the core system. Step 4 uses external simulators simulating the platform to verify external connectivity.

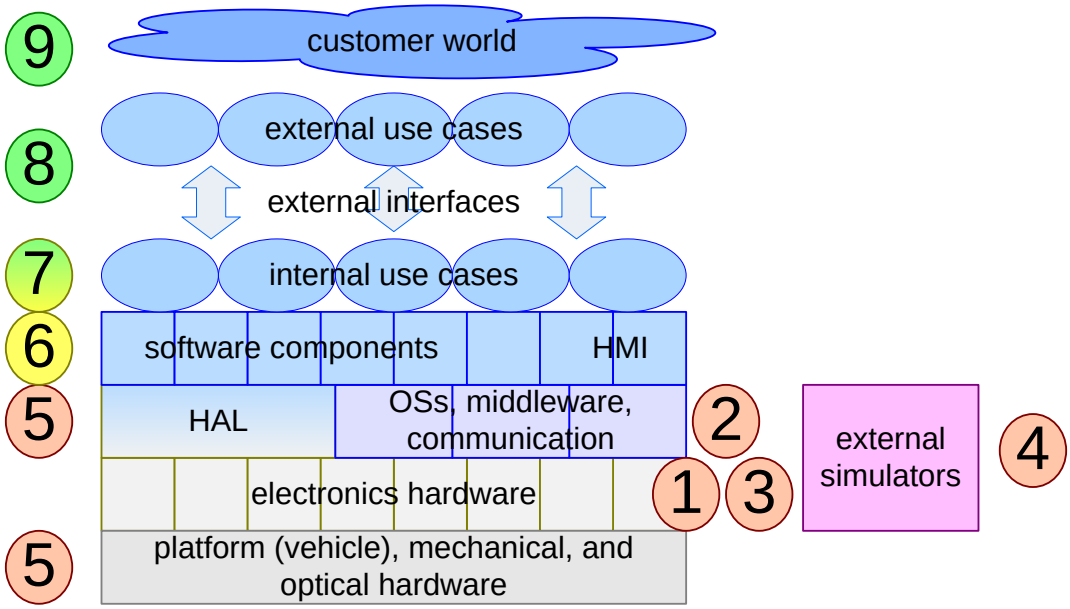


Figure 2. The RWS system shown as layered system and the logical order of integration.

Step 1 runs into a logical dependency problem: integration needs some initial (test) software, while the software resources work on the software layers. Steps 2 and 3 are time-consuming

do to variation of components. A problem in steps 2 to 4 is that the focus is on functionality, while performance and robustness get insufficient attention.

Step 5 is integration of Hardware Abstraction Layer (HAL) with core system and mechanical and optical hardware. The idea is that this happens incrementally in preparation of the software functionality and internal use case that will be integrated in next increment. Practical problem is that hardware components and use cases do not have a clear relation. The other fundamental challenge is that hardware has a production time that dominates the planning and may conflict with desired functional increments.

Step 6, integration and test of software functionality, and step 7 functional and performance test of use cases at RWS system level, fit better in the incremental approach. However, there is a tension between architecture development (holistic view) and the increments (fragmented view, often with time pressure to realize short increments). The iterative approach depends on automated testing, among others unit testing. One of the problems is that only part of the software has automated testing provisions. Especially older software lacks unit testing and provisions for test automation. In these steps, the SysML sequence diagrams prove to be a good foundation for integration.

Steps 8 and 9 in the integration into the customer's system and context. Interestingly, these steps are known to be challenging, however, they run well. Experienced software developers and testers perform this integration. This area with known high risks is followed up so closely that few problems arise.

6. How to do systems integration?

The RWS case shows clearly that the main tension in systems integration is the need to validate key performance early, while hardware and infrastructure tends to appear late in the project plan. The order described in the case is typical for integration. Figure 3 shows a more generic diagram from [Muller 2011].

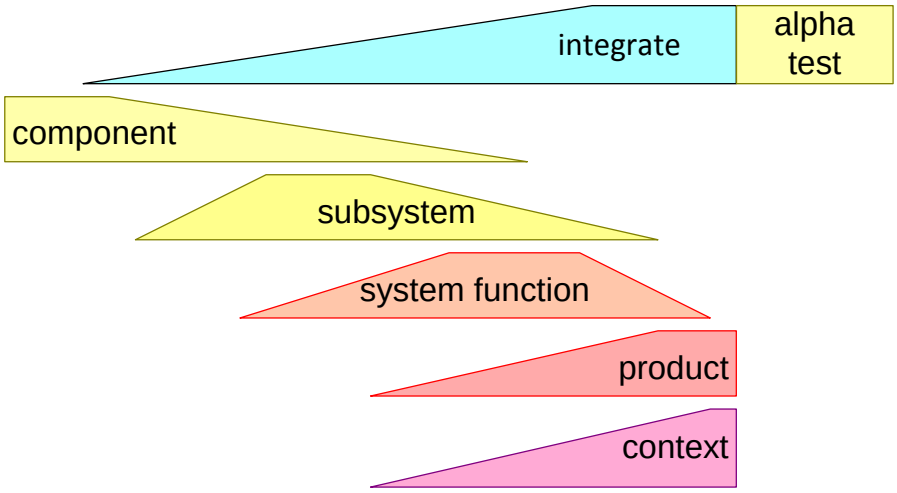


Figure 3. Logical Order of Systems Integration is bottom-up. However, the goal of early validation requires early confrontation of the whole system and its context.

One of the ways to achieve early validation, despite late delivery of some hardware is using alternative implementations, ranging from modified old hardware, to models and simulators. Figure 4 shows a spectrum of alternatives to validate earlier when hardware is not yet available.

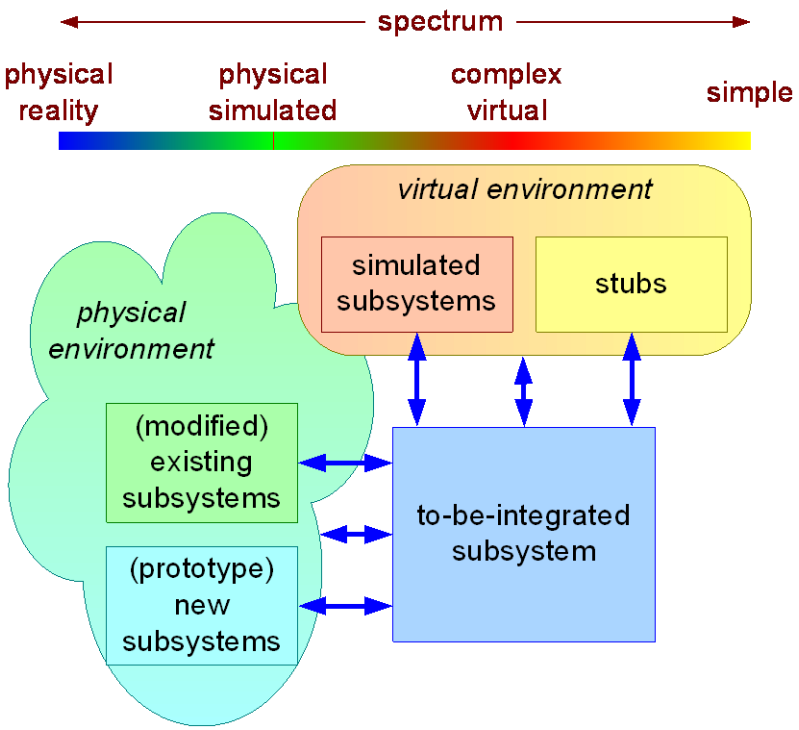


Figure 4. Spectrum of alternate environments to validate early.

Many organizations use the existing version of a system to integrate modified or new components and functions. For example, the project runs new software on existing hardware. If this integration reveals problems, then often it does not make sense yet to run the new software on the scarce new hardware. One of the participants used the term “canary build” for this way of working; the canary that detects problems early.

In general, systems integration requires an understanding of dependencies, ultimate (customer-level) performance goals, and risks. Integrators and project leaders transform these ingredients into an integration sequence (what to test, on what configuration, in what order). The integration sequence in turn transforms into an integration schedule to manage resources. In practice, integration schedules change continuously, due to trouble shooting (integration fails) and due to project issues (delays of deliveries). Integration requires an

agile mindset. Many organizations have frequent short meetings (for example, 15 minutes at the beginning of each day) once integration is in full flow.

1. Determine integration views
2. Determine integration risks
3. Determine integration sequence
4. Determine increments
5. Determine test environment
6. Match development-, integration- & test planning
7. Provide planning per increment
8. Capture all details, results, decisions
9. Iterate and check assumptions

Figure 5. The TNO-ESI cookbook for Integration and Test

TNO-ESI developed an integration and test cookbook as shown in Figure 5[1]. This cookbook is similar to the approach we discussed.

The best paper of IS2013 by Papke [Papke 2013] describes the "last Planner" as used in civil engineering. In this approach, planners use a look-ahead window to assess maturity of planned deliveries. When deliveries are assessed to be too immature, then the planning is adapted to mitigate disruptions by immature deliveries (or in practice, also non deliveries). Typical look-ahead windows are 2 and 6 weeks.

[1] As part of the System integration and test course http://www.esi.nl/innovation-support/competence-development/courses/system-integration-and-test.dot

7. What competence is required for systems integration?

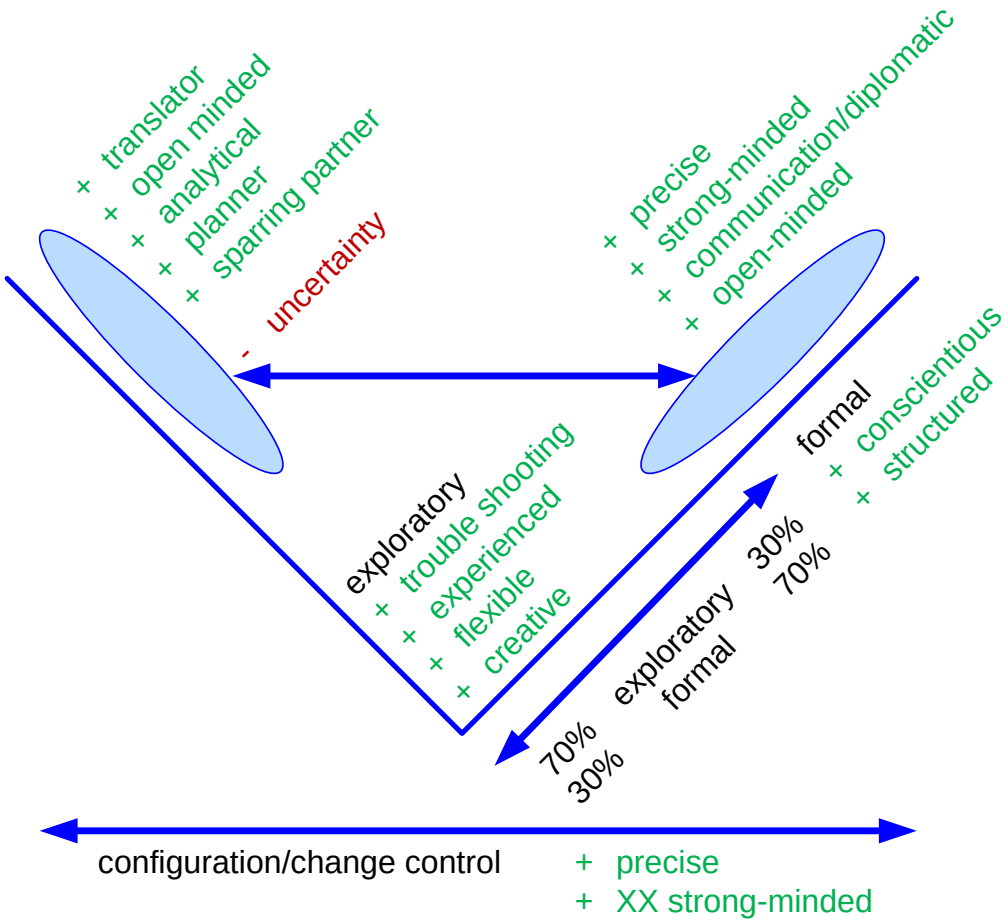


Figure 6. Competences for systems integration related to phase and role in the V-model.

In the last breakout session, we discussed the question "What competence is required for systems integration?" Figure 6 shows the context dependence of this question. Role and phase of the project impact desired competences for systems integration. In the beginning of the V-model, a system integrator is a sparring partner for systems architects. The integrator translates integration issues into needs or requirements and translates architectural ideas into systems integration consequences. In this phase, an open-mind and analytical skills are

essential to be a serious sparring partner for architects. We observe in practice, that systems integrators have difficulties in coping with the inherent uncertainties early in the project.

In the late phases of the project, the systems integrator becomes a finisher, which requires precision and strong-mindedness. However, similar to architects, systems integrators work mostly by influencing and recognized authority. Hence, they need strong communication and diplomatic skills. At the bottom of the V-model, integrators have foremost an exploratory role. They are heavily involved in trouble shooting (early in integration, nearly nothing works as expected), which tends to require experience. Flexibility and creativity help them in this phase. Gradually the exploratory role transforms into a formal verification role, where conscientious and structured work dominates.

A complementary role is configuration management and change control, which require precision and extra strong-mindedness. Both aspects are crucial for systems integration. Ill-defined configurations cause unexpected results and associated delays. Changes and change propagation are essential to make progress; however, at the same time they complicate configuration control and disrupt ongoing testing.

Other competences that came up in the discussion are:

- High tolerance of physical and mental pain
- Ability to see the big picture on all (most) use cases
- Ability to switch between large and small (jo jo) and know and like details
- Domain knowledge (technical depth and application knowledge)
- Ability to prioritize
- Operational capabilities; they are foot soldiers and machine owners
- Ability to report to senior management
- Know critical aspects and, hence, what to test
- Directness in telling the truth (prevent convolution of truth in every reporting level)
- Focus on solving problems (rather than blame assignment)
- Being able to drag in relevant experts (knowing who they are)

- Sense of urgency
- Motivated by seeing things work and work together, hands-on
- Communicative, collaborative, and humble
- Ability to manage risks
- Cross discipline
- Planning and adaptable to change
- Respected and trusted in the organization
- Ability to deal with pressure

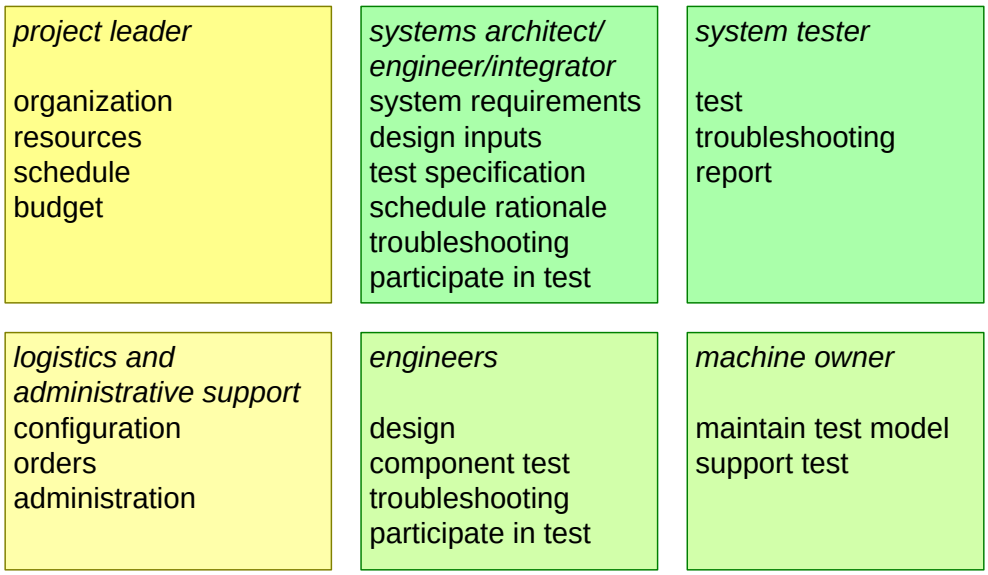


Figure 7. Multiple roles contributing to systems integration.

From this long list and Figure 6 emerges a profile for a systems integrator that is not realistic. Typically, the integration work is allocated to multiple employees with complementary characteristics. Figure 7 from [Muller 2011] shows multiple complementary roles for systems integration.

Principle 15.3 The systems integrator role is complementary to the systems architect role. Systems integration requires many competences that in practice cannot be found in

a single person; Hence, systems integration needs complementary roles to cover all required competences.

8. Discussion and conclusions

We started the forum with a number of preparatory questions, and we raised several question for the breakout sessions. We will briefly discuss these questions here, and formulate answers as far as our discussion resulted in them.

When does your organization start integrating? How does integration influence the early architecting phase?

Systems integration needs to start at the beginning of the project. System integration concerns, such as instrumentation and design of dependencies, are inputs to the architects. Architects own the validation of key performance parameters, and as such, need to strive for early validation.

What people in your organization understand systems integration? What role do they play in projects?

In many organizations, there is insufficient understanding of systems integration. Key players as project leaders and (lead) engineers sometimes underestimate systems integration. An approach is to involve managers early in the cross cutting aspects, the "ilities" early. Unfortunately, here a gap stats already; when people do not recognize cross cutting concerns, how can they understand systems integration? Finally, participants managed a "horizon effect": managers that ignore problems that are beyond their current (short-term) horizon.

Software managers and engineers often claim that continuous and incremental integration solves all integration issues. In practice, we see many opposite effects; the purely incremental focus tends to ignore cross cutting concerns, such as performance and other qualities, which "emerge" from increments. Another effect in continuous integration is that it often also means continuous change, which can hamper the desired feedback. Continuous integration and incremental work is highly relevant for systems integration. However, it

requires focus on cross-cutting concerns (for example, end-to-end performance) to benefit from an incremental approach.

How to cope with the context of the system? How to cope with life cycle evolution in systems integration?

Systems integration extends into the customer space, the operational context for the system. Many system problems arise from unforeseen interaction between internal system design choices and external circumstances. Major causes of system failure are the complex and unpredictable characteristics of nature and human beings. Many forms of testing strive to hit such problems early, for instance by stress, load, and accelerated lifetime testing. However, there is no clear and generic answer on how to achieve acceptable (and affordable) coverage. Continuation of systems integration in the field, as done by Microsoft MDG, is clearly part of the solution. Note that the aerospace industry has been doing this for decades by using black box recorders.

Awareness of the continuous change of systems and their context is a starting point to cope with lifecycle evolution. Life cycle evolution is an increasing issue, due to increased interoperability of systems. Our discussion did not reveal more solutions than the Microsoft approach. The field of Systems of Systems Engineering recognizes this as one of the major challenges.

How to do Systems Integration?

Summary of Section 6 on this topic is that systems integration requires an integration and plan that strives to fail early. The way to achieve this is to integrate early, and to use old systems, models, simulators, and prototypes creatively. Focus especially on key performance parameters, at customer level (end-to-end). This necessitates that the system context (environment) is included in the integration sequence. Thinking about risks (unknowns, uncertainties) helps to determine what needs early integration attention.

What makes someone a good systems integrator?

Summary of section 7 is that a good system integrator is another sheep with 7 legs (similar to systems architects [Muller 2011]). Hence, a team of complementary roles does the systems integrator work. Complicating factor is that few managers understand systems integration; even less managers understand the competence of good systems integrators.

One of the participants phrased it as “System integration (and Architecture) is a discipline, and there is a whole lot to think about!”

Global trends

The scope of applications is increasing quickly. Systems that contribute to applications get more and more interoperable. All these systems (cooperating as Systems of Systems) evolve over time, as well as their context. Consequently, system boundaries get less clear, interfaces and behavior gets more dynamic, systems, stakeholders and concerns get more heterogeneous, and less and less people have overview and insight of end-to-end functionality and performance. The sum of these trends makes systems integration even more important and challenging.

Acknowledgements

Kongsberg group hosted this meeting. They provided an inspirational demonstration of bridge systems, and organized a tour through the Kongsberg silver mines; an engineering marvel of the past.

Literature

Muller, G.J. (2011). “Systems Architecting; A Business Perspective”, CRC Press, Boca Raton, FL, USA; see chapter 7

Papke, B. and Dove, R. (2013) "Combating Uncertainty in the Workflow of Systems Engineering Projects." Proceedings of INCOSE 2013 in Philadelphia, USA.

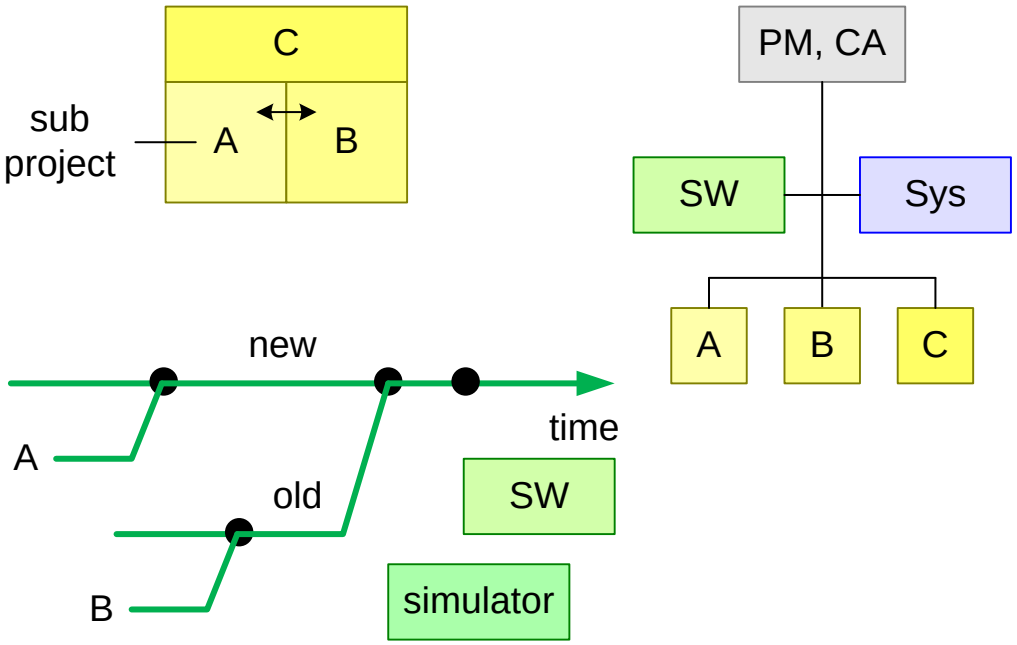
Appendix A, What is Systems Integration?

Answers of participants on the question "What is Systems Integration?"

- Assembly of predefined elements to deliver an expected outcome, i.e. a desired function.
- - Bringing the pieces together to a working whole.

  - Debugging & troubleshooting

  - Agile alignment of realization
- Planned assembly and verification of system components into sub assemblies and finally the whole system removing risks early.
- Task and verification of an architectural decomposition of a product/system



- System integration is about creating working systems from (numerous) subsystems where interfaces sometimes are given and sometimes can still be defined. In addition, some "glue" (eg in behavioural terms) needs to be applied from the systems side.

- Integrating parts defining in an architecting phase. Costumer level + design level.
- The architect's grumpy, unpredictable spouse.
- The ultimate proof /truth /"revealers" of all design efforts.
- The lifecycle of a project from a practical engineering point of view, focusing on the technical aspects of making systems play well together.
- Making "independent" systems working together to perform the functions of a large system.
- Planning and execution of how to make the HW parts and SW from different suppliers work together to deliver a complete system solution.
- Steering development by assembly & test by looking at qualities & testing.
- Realizing a new system by combining products functions and sensors with each other.
- "Putting" together parts to see if it works as designed.
- Complex and large products cannot be developed as one big project: it will consist of many sub modules (supplied by other projects). System integration means:

- taking care that all sub modules are well specified.

- taking care that the interactions between the sub modules are well defined.

- Taking care that the end-user specifications are met.

- - Definition of Sub System Interfaces

- Management of Sub System Interfaces

- Hierarchical Test of Sub system composition (structure based)

- Phased Test of System Composition MIL->SIL->HIL->Mules->System. (6)

- - *Closing* the design look (balancing reductionism)

- *Aligning* minds, descriptions technology.

- Bringing sub systems together and verify the system.
- - Integration of our different products.

- Integration of 3rd party suppliers.

- Standard interfaces
    - With standardization body
    - Without a standardization body