Coping With System Integration Challenges in Large Complex Environments

Gerrit Muller
Embedded Systems Institute
Gerrit.muller@embeddedsystems.nl

Abstract. The increasing scope of systems integration poses many challenges for the system integrators. This paper contends that in conventional projects the Systems Integration phase is systematically underestimated. During this phase all unexpected and unforeseen problems surface, where most problems are cross-organizational. When the project size increases with more suppliers, more users, more enterprise processes, and more (intelligent) functionality, then the system integration phase gets tremendously more difficult.

In this paper we will bring order to the integration process and discuss an approach and best practices to cope with systems integration in conventional projects. Such level of integration approach is a prerequisite to deal with all added integration complexity in systems with a scope of (intelligent) enterprises.

System Integration as part of Systems Engineering Process

Introduction System Integration is one of the activities of the Product Creation Process. The Product Creation Process is triggered by a set of product needs and ideas and results in a system that:

- fits in customer's needs and context
- can be ordered, manufactured, installed, maintained, serviced, disposed
- fits the business needs

During Product Creation many activities are performed, such as: feasibility studies, requirements capturing, design, engineering, contracting suppliers, verification, testing, et cetera. A universal organizational instrument is *decomposition*. *Decomposition* enables the distribution of work in a concurrent fashion. The complement of *decomposition* is *integration*. Every activity that has been decomposed in smaller steps will have to be integrated again to obtain the single desired outcome.

System Integration Definitions. System integration constitutes an ongoing flow of activities during the entire systems engineering process. The nature of these ongoing integration activities, however, changes with the changing system maturity and development phase. Early in the project technologies or lower level components are integrated, while at the end of the project the entire system is built from the higher-level subsystems and then verified. As an example of this process, the NASA NPR 7123.1[1] defines product integration as "*The process used to transform the design solution definition into the desired end product of the WBS model through assembly and integration of lower-level validated end products in a form consistent with the product-line life-cycle phase exit criteria and that satisfies the design solution definition requirements*." The

[1] NASA NPR 7123.1 – NASA Systems Engineering Processes and Requirements

Defense Acquisition Guide (DAG) defines system integration as the "*process of incorporating lower level system elements into higher-level system elements in the design solution or the physical architecture*".

Challenges with complex system integration. In practice projects hit many problems that are caused by decomposition steps. Whenever an activity is decomposed the decomposed activities run well, however crosscutting functionality and qualities suffer from the decomposition. Lack of ownership, lack of attention, and lack of communication across organizational boundaries are root causes for these problems. The counter measure for these problems is to have continuous attention for the integration.

In this paper we will discuss the full integration flow. We will discuss the goal of integration, the relation between integration and testing, what is integration and how to integrate, an approach to integration, scheduling and dealing with disruptive events, roles and responsibilities, configuration management aspects, and typical order of integration problems occurring in real life.

Goal of Integration. The goal of integration is to reduce the project risk of being late or the risk of creating an ill performing system. During integration the project team tries to find unforeseen problems as early as possible, in order to solve these problems in time. Integration plays a major role in risk reduction. The word *unforeseen* indicates the main challenge of integration: How to find problems when you don't know that they exist at all?

Problems can be unforeseen because the knowledge of the creation team is limited. May be nobody on earth did have the knowledge to foresee such a problem, simply because the creation process enters new areas of knowledge. Problems can also be unforeseen due to invalid assumptions. For instance many assumptions are being made early in the design to cope with many uncertainties. The limited intellectual capabilities of us, humans, limit also the degree in which we can oversee all consequences of uncertainties and assumptions we make. A common source of unforeseen problems is interference between functions or components. For example two software functions running on the same processor may perform well individually, but running concurrently may be way too slow, due to cache pollution or memory trashing.

System Integration as part of Product Creation Process. The Product Creation Process (PCP) is often prescribed as a sequence of phases with increasing level of realization and decreasing level of risk. This is a useful high-level mental model, however one should realize that most activities have much more overlap in the current dynamic world. The pure waterfall model, where requirements, design, integration and test are sequential phases, is not practical anymore. The problem in today's complex systems is that the because it relies on a high level of end product knowledge and understanding at the start of the Product Creation Process which is not not yet present, nor practical for most complex projects. Much more practical is an approach with a shifting emphasis as shown in Figure 1. A comparable approach is RUP. Note especially the long ramp-up of the integration, the focus of this paper. The bar at the bottom of this figure shows a typical product creation phasing used in industry. Typical 5 to 8 phases are present, where the naming suggests that integration is an activity nearly at the end. Figure 1 brings the message that system integration actually requires some effort from very early in the project onwards.

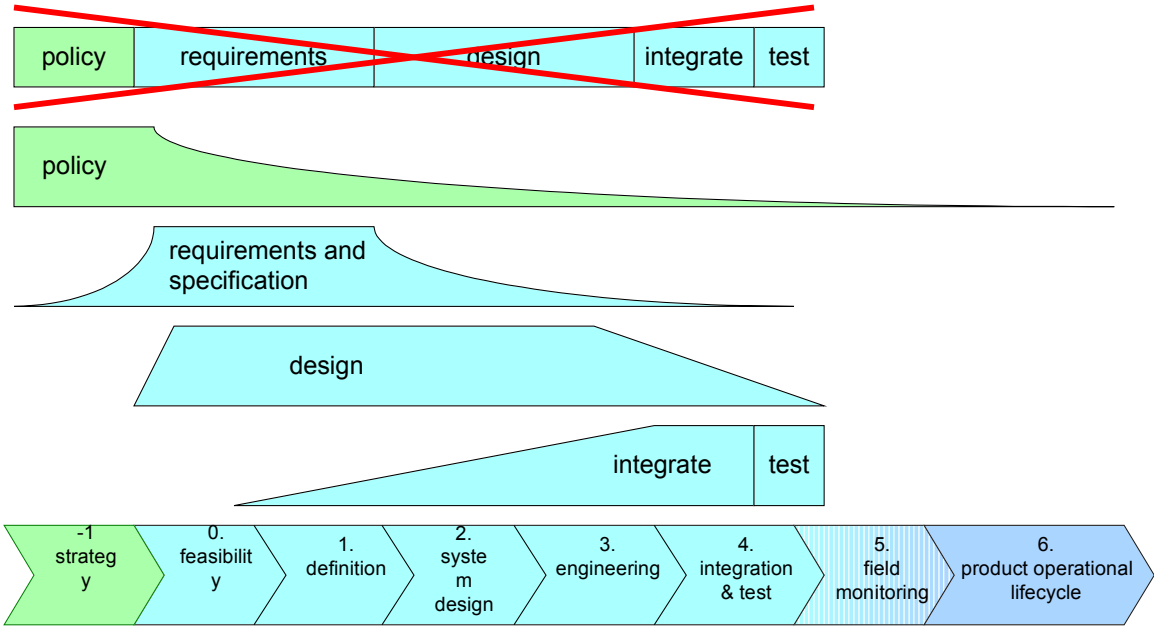

Figure 1. Typical Product Creation Process and the concurrency of engineering activities.

Integration and Testing. Integration and testing is often used as identical activities. However, the two activities are related and completely different at the same time.

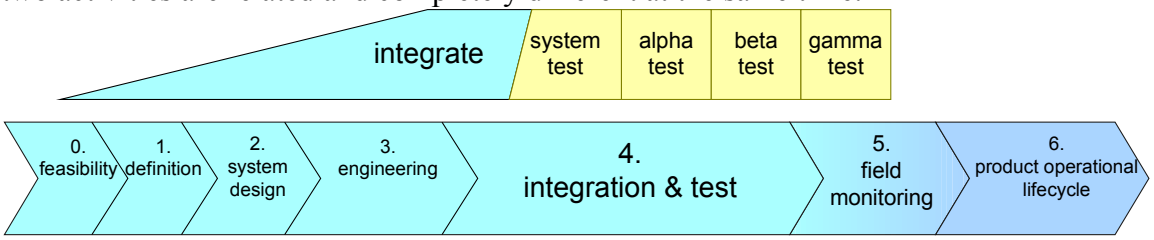

Figure 2. Zooming in on integration and tests.

Figure 2 zooms in on the integration and test activities. Integration is the activity where we try to find the unknowns and where we resolve the uncertainties. Testing is an activity where we operate a (part of a) system in a predefined manner and verify the behavior of the system. A test passes if the result fits the specified behavior and performance and otherwise it fails. During integration many tests may be applied as part of the integration. These tests during integration are applied to find these unknowns and to resolve the uncertainties. When the milestone is passed that the system is perceived to be ready, then the system engineers will run an entire system level test suite. Normally this run still reveals unknowns and problems. The system test verifies both the external specification, as well as the internal design. When sufficient stability of the system test is achieved a different working attitude is taken: from problem solving to verification and finishing. The alpha test starts with a hard milestone and is also finished at a well-defined moment in time. The alpha test is the formal test performed by the product creation team itself, where the specification is verified. The beta test is also a well-defined time-limited formal test, performed by the "consuming" internal stakeholders: marketing, application, production, logistics and service. The beta test also verifies the specification, but the testers have not been involved in product creation. These testers are not blinded by their a priori know-how. Finally the external stakeholders, such as actual users, test the product. Normally problems are still

found and solved during these test, violating the assumption that the system is stable and unchanged during testing. In fact, these alpha, beta, and gamma testers hit problems that should have been found during integration. We will focus the rest of this paper on integration, with the main purpose to reduce risks in the testing phase by identifying (potential) problems as early as possible.

System Integration Strategies and Approaches

Integration from Components to Systems. By necessity the integration of a system starts bottom-up with testing individual components in a provisional component context. The purpose of the bottom-up steps is to find problems in a sufficiently small scope to diagnose them. If we bring thousands of components together into a system, then this system will fail for certain. But it is nearly a mission impossible to find the sources of this failure, due to the multitude of unknowns, uncertainties, and ill-functioning parts.

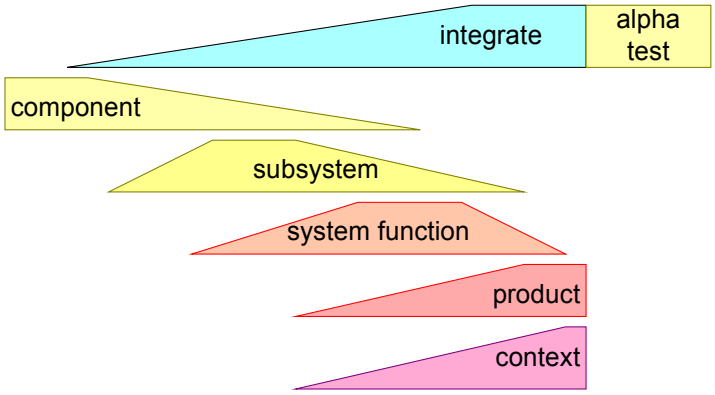

Figure 3. Integration takes place in a bottom-up fashion, with large overlaps between the integration levels.

The focus of the integration activity is shifting during the integration phase. Figure 3 shows the bottom-up levels of integration over time. Essential for integration is that the higher levels of integration start already, when the lower levels of integration are not yet finished. The different levels of integration are therefore overlapping. Early during integration the focus is on functionality and behavior of components and subsystems. Then the focus is shifting to system level functionality: do the subsystems together operate in the right way? The last step in integration is to focus on the system qualities, such as performance and reliability. The system qualities can often only be tested in a context, such as other systems providing inputs or consuming outputs. A context of the system has to be realized in parallel with the system and the system and the context are integrated stepwise. We will discuss later in this paper that the context might be simulated or improvised, as well as be created by the more actual operational environment.

Integration Approaches. The integrator tries to integrate subsystems or functions as early as possible with the purpose of finding unforeseen problems as early as possible. This means that integration already takes place, while most of the new components, subsystems, and functions are still being developed. Normally partial systems or modified existing systems are used in the early phases of integration as substitute of the not yet available parts. Figure 4 shows this transition from using partial and existing subsystems to systems based on new developed parts.

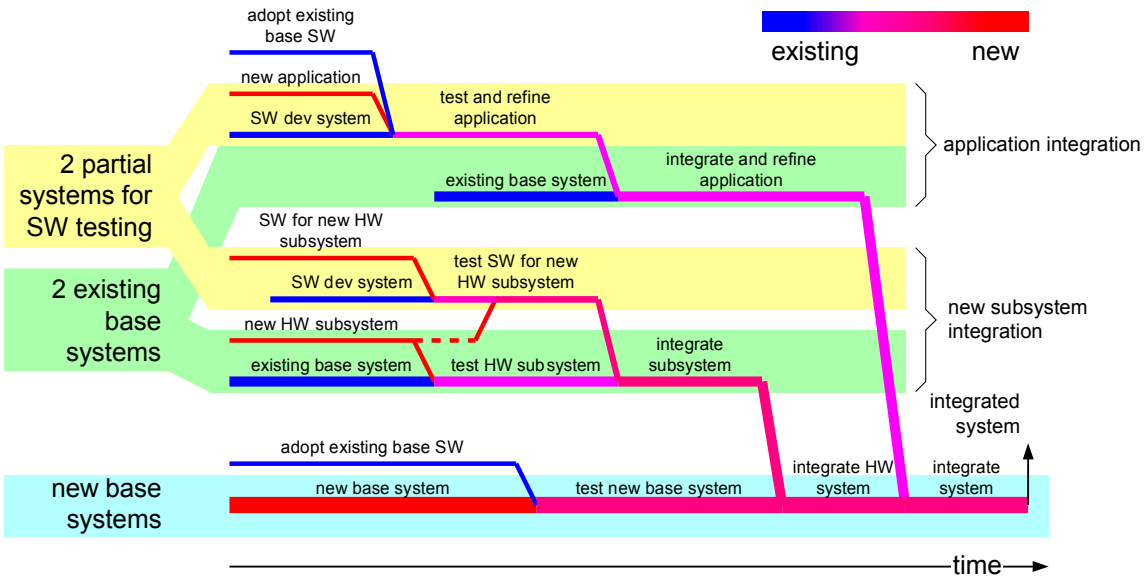

Figure 4. During integration a transition takes place from using previous systems and partial systems to the new system configuration.

The unavailability of subsystems or the lack of stability of new subsystems forces the integrator to use alternatives. Figure 5 shows a classification of alternatives. Simple stubs in a virtual environment up to real physical subsystems in a physical environment can be used. In practice multiple alternatives are combined. As function of time the integration shifts from the use of stubs and a virtual environment to as close as possible to the final physical reality.

The challenge for the project team is to determine what intermediate integration configurations are beneficial. Every additional configuration adds costs: creation costs as well as costs to keep it up-to-date and running. An even more difficult conflict is that the same critical resources, an imaging expert for instance, are needed for the different configuration. Do we focus completely on the final product, or do we invest in intermediate steps? Last but not least is the configuration management problem that is created with all integration configurations. When hundreds or thousands of engineers are working on a product then most of them are in fact busy with changing implementations. Strict change procedures for integration configurations may reduce the management problem, but this conflicts often with the troubleshooting needs during integration.

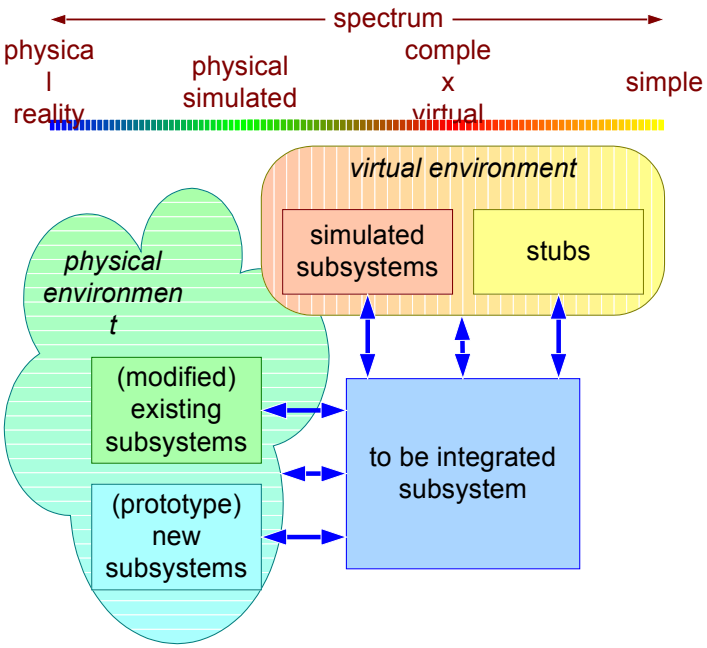

Figure 5. Alternatives to integrate a subsystem early in the project.

Crucial questions in determining what intermediate configurations to create are:

- How critical or sensitive is the subsystem or function to be integrated?
- What are the aspects that are sufficiently close to final operation so that the feedback from the configuration makes sense?
- How much need to be invested in this intermediate step? Special attention is required for critical resources.
- Can we formulate the goal of this integration system in such a way that it guides the configuration management problem?

1	determine most critical system performance parameters
2	identify subsystems and functions involved in these parameters
3	work towards integration configurations along these chains of subsystems and functions
4	show system performance parameter as early as possible; start with showing "typical" system performance
5	show "worst-case" and "boundary" system performance
6	rework manual integration tests in steps into automated regression tests
7	monitor regression results with human-driven analysis
8	integrate the chains: show system performance of different parameters simultaneously on the same system

Figure 6. Stepwise integration approach

Based on these considerations we propose a stepwise integration approach as shown in Figure 6. The first step is to determine a limited set of the most critical system performance parameters, such as image quality, productivity or power consumption. These system performance parameters are the outcome of a complicated interaction of system functions and subsystems; we call the set of functions and subsystems that result in a system parameter a *chain*. The complementary activity of system architecting and design has to decompose the system, to define interfaces and to allocate contributions of system level performance parameters to the elements of the decomposition. Component and subsystem testing is used to verify compliance with the specification. The integration itself has to focus on the aggregate performance of the elements. In this way it is verified that the composition behaves and performs in the intended way. We start to define partial system configurations as integration vehicles once we have identified critical chains. The critical chains serve as guidance for the integration process.

Performance parameters and their related chains can be critical for different reasons: the performance parameter is highly critical for system success in its context, or we have a priori knowledge, for instance based on historic data, that this performance parameter is very sensitive, vulnerable or uncertain. As example we can look again at software performance, where experience shows that individual functions tend to perform good, however the composition of many functions performs bad, due to low level resource conflicts.

We strongly recommend focusing on showing the critical system performance parameters as early as possible. In the beginning the focus is on “typical” performance. Once the system gets somewhat more stable and predictable, then we get room to also study “worst-case” and “boundary” performance.

Regression Testing and Analysis. Since many engineers are still changing many parts of the total system it is important to monitor the system performance regularly. The early integration tests are manual tests, because the system circumstances are still very premature and because integrators have to be responsive to many unexpected problems. In due time the chain and the surrounding system gets more stable, allowing automation of tests. We can migrate the early manual integration steps into automated regression test. The results of regularly performed regression tests must be monitored and analyzed by system engineers. This analysis does not focus on pass or fail, but rather looks for trends, unexplained discontinuities, or variations.

Later during integration we have to integrate the chains themselves and to show the simultaneous performance of the critical performance parameters.

All test results and test conditions have to be archived. In this way a set of historical system data is created. Historical data is helpful during trouble shooting: “Did we see this effect before?”, “How and when does the effect occur?”, et cetera.

Integration Plan

Integration Planning and Scheduling. The approach described in the previous section requires quite some logistics support. The project leader will therefore make integration schedules in close cooperation with the system engineers. Integration schedules have two conflicting attributes:

- Predictability and stability to ensure timely availability of resources
- Flexibility and agility to cope with the inherent uncertainties and unknowns.
- The starting point to create a schedule is to determine a specific and detailed order of

integrating components and functions to finally measure the desired critical system performance parameter.

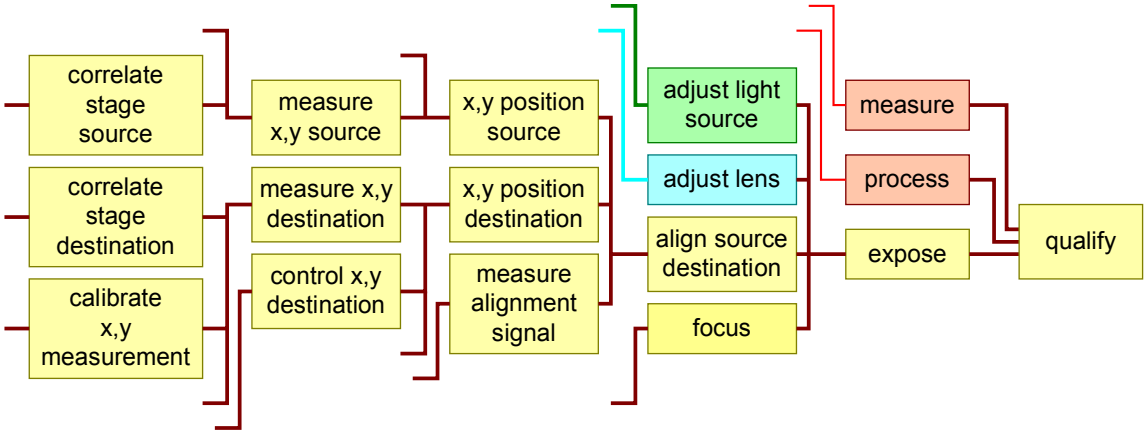

Figure 7. Example of small part of the order of functions required for the image quality system performance parameter of a waferstepper.

Figure 7 shows an example of a specific order of functions required to determine the image quality system performance parameter of a waferstepper. Such a diagram starts often at the right hand side: what is the desired output parameter to be achieved? Next the question "What is needed to achieve this output?" is asked recursively. This very partial diagram is still highly simplified. In reality many of these functions have multiple dependencies.

Worse is that often circular dependencies exist, for instance in order to align source and destination we need to be in focus, while in order to find the focal plane we need to be aligned. These dependencies are known during design time and already solved at that moment. For example a frequently used design pattern is a stepwise refined: coarse and fine alignment, and coarse and fine focusing. The creation of a detailed integration schedule provides worthwhile inputs for the design itself. Making the integration schedule specific forces the design team to analyze the design from integration perspective and often results in the discovery of many (unresolved) implicit assumptions.

Dealing with Disruptive events. The existence of this integration schedule must be taken with a grain of salt. It has a large value for the design and for understanding the integration. Unfortunately, the integration process itself turns out to be poorly predictable: it is an ongoing set of crises and disruptive events, such as late deliveries, breaking down components, non-functioning configurations, missing expertise, wrong tolerances, interfering artifacts, et cetera. Crucial to the integration process are capabilities to improvise and to troubleshoot.

The integration schedule is a rather volatile, and dynamic entity. It doesn't make sense to formalize the integration heavily, neither to keep it updated in all details. Formalization and extensive updating takes a lot of effort with little or no benefits. The recommended approach is to use the original integration schedule as kind of reference and to use short cyclic planning steps to guide the integration process. Typical meeting frequency during integration is once per day. Every meeting results and problems, required activities and resources, and short-term schedule are discussed.

Roles and Responsibilities. During integration many project team members are involved with

different roles and responsibilities:

- Project leader
- System architect/engineer/integrator
- System tester
- Logistics and administrative support personnel
- Engineers
- Machine owner

Figure 8 shows these roles in relation to their responsibilities. Note that the actual names of these roles depend on the organization, we will use these generic labels in this paper.

The *project leader* is the organizer who takes care of managing resources, schedule and budget. Based on inputs from the system engineer the project leader will claim and chase the required resources. The project leader facilitates the integration process. This contribution is critical for the project timing.

The *system architect*, *system engineer* and *system integrator* role is in fact a spectrum of roles that can be performed by one or more persons, depending on their capabilities. A good system architect is sometimes a bad system integrator and vice versa. This role is driven by content, relating critical system performance parameters to design and to test. In this role the rationale of the integration schedule is determined and the initial integration schedule is a joint effort of project leader and system engineer. The integral perspective of this role results in a natural contribution to the troubleshooting. The system architect/engineer is responsible for the success of the integration test.

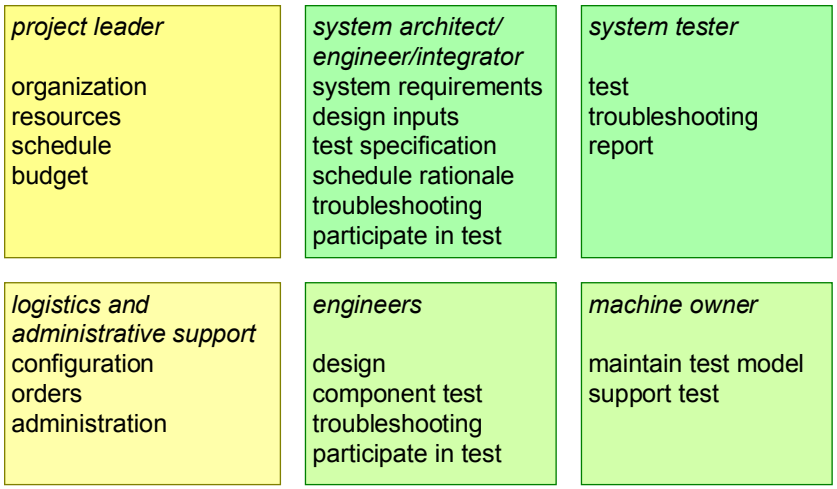

Figure 8. Roles and responsibilities during the integration process.

The *system tester* is the practical person actually performing most of the tests. During the integration phase lots of time of the system tester is spent in troubleshooting, often of trivial problems. More difficult problems are escalated to engineers or system integrator. The system tester documents test results in reports.

The *machine owner* is responsible for maintaining a working up-to-date test model. In practice this is a quite challenging job, because many engineers are busy with making updates and performing local tests, while system integrator and system tester need undisturbed access to a stable test model. We have observed that explicit ownership of one test model by one machine

owner increases the test model stability significantly. Organizations without such role lose a lot of time due to test model configuration problems.

Engineers deliver locally tested and working components, functions or subsystems. However, the responsibility of the engineers continues into the integration effort. Engineers participate in integration tests and help in troubleshooting.

The project team is supported by all kinds of support personnel. For integration the *logistics and administrative support* is crucial. They perform configuration management of test models as well as the products to be manufactured. Note that integration problems may induce changes in the formalized product documentation and the logistics of the final manufacturing, which can have significant financial consequences due to the concurrency of development and preparation of production. The logistics support people also have to manage and organize unexpected but critical orders for parts of test models.

Supply Chain

Configuration Management. Configuration management and integration are intimately related as discussed in the previous sections. We should realize that configuration management plays a role in many processes. Figure 9 shows a simplified process decomposition of those processes that are related to configuration management.

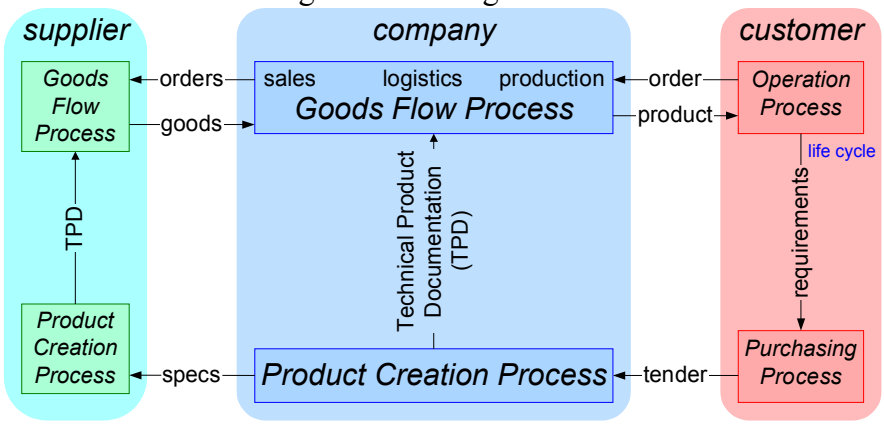

Figure 9. Simplified Process diagram that shows processes that are relevant from configuration management perspective.

We will discuss two different cases:

- Products that require a Goods Flow Process, because repeated ordering, assembly and testing will take place
- One-of-a-kind products

Goods Flow Process. The main entities involved in configuration management are suppliers, the organization or company itself and customers. There are two main flows where configuration management plays a role:

- Creation flow, from customer requirements to component specifications to technical product documentation to be used in the other flow.
- Goods flow, a repeating set of processes where orders are fulfilled by a logistics and production chain.

In principle the *creation flow* is a one-time project activity. This flow may be repeated to create successor products, but this is a new instantiation of this flow. The *goods flow* is a

continuous process with life cycle considerations. The final product as used operationally by customers also has its own life cycle.

Many entities have changing configurations and therefore need configuration management. Figure 10 shows the same process decomposition as Figure 9, but now annotated by entities under configuration management. Two classes of configuration management entities exist: *data* and *physical items*. The data entities normally fit well in existing procedures and tools. However for physical entities the challenge is to maintain consistency between the actual physical item and the data in the configuration management administration. Especially during the hectic period of integration the administration sometimes differs from the physical reality, causing many nasty problems. Sometimes more process helps, sometimes more process results in more latency and more work-around behavior; unfortunately, there is no silver bullet for configuration management processes.

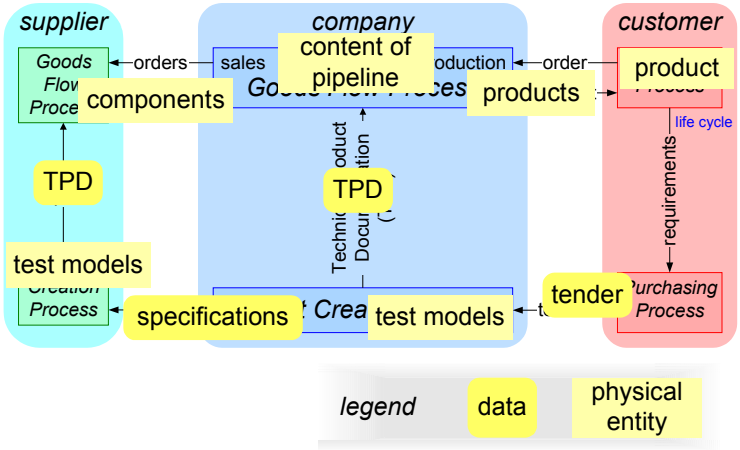

Figure 10. The simplified process diagram annotated with entities that are under configuration management.

The main configuration management entities during integration are the test models. However, changes in test models may have to propagate to other entities, such as specifications, technical product documentation, and, due to concurrency, also to components and products in the goods flow processes.

One particular area of attention is the synchronization of components, subsystems and test models. All these entities exist and change concurrently. A certain pull to use latest versions is caused by the fact that most problems are solved in the latest version. However, integrators and testers need a certain stability of a test model. This makes integrators and testers hesitant to take over changes. One should realize that only a limited amount of test models exist, while all these engineers create thousands of changes. On top of this problem comes a logistics problem: from change idea to availability of changed component or function may take days or weeks. Sometimes one provisional changed component is available early.

One way of coping with the diversity of test model configurations is to clearly formulate the integration goals of the different test models. Note that these integration goals may change over time, according to Figure 3.

One-of-a-kind Systems. The previous section discusses configuration management from the perspective of products that are repeatedly produced. Different issues and concerns pop-up for

one-of-a-kind systems, such as scientific experimental set-ups or turnkey projects. In this case the system used for integration is one and the same as the delivered system, although partial prototypes may still be used for early integration steps.

Beneficial for one-of-a-kind systems is that the creation chain is shortened by one difficult transfer step from creation to repetition. However, one should realize that within the single system a repetition of subsystems still requires repeated production, with all its integration consequences. Examples are the antennas and electronics for the LOFAR telescope, the detectors for high-energy experiments et cetera.

Destructive or potentially damaging integration steps are unwanted for one-of-a-kind systems. Alternative environments, as shown in Figure 5, get even more important. Because of the "virtual" nature of most of the integration environment, a lot of attention is required for the potential differences between the actual future reality and the current integration environment. Techniques such as Failure Mode Effect Analysis (FMEA) applied in the broader context of the environment help to identify potential problems early.

Impact and Propagation of Changes. The integration phase is a period in which many design and implementation changes are taking place. Problems discovered due to the integration are one of the main sources of these changes. The theory of requirements traceability suggests that the change impact can be predicted and managed. Unfortunately the situation in practice is much nastier. The left hand side of Figure 11 shows the formal situation as defined by the traceability graph. The effect of a low-level change can be traced to the impact on system level requirements. The right hand-side of figure 11 shows the more actual situation: a number of undocumented relations between design choices pop-up, causing an avalanche of additional impacted requirements. Every additional relationship triggers its own avalanche of impacted higher-level requirements. The counteraction to repair the design in itself will trigger a new avalanche in the opposite direction.

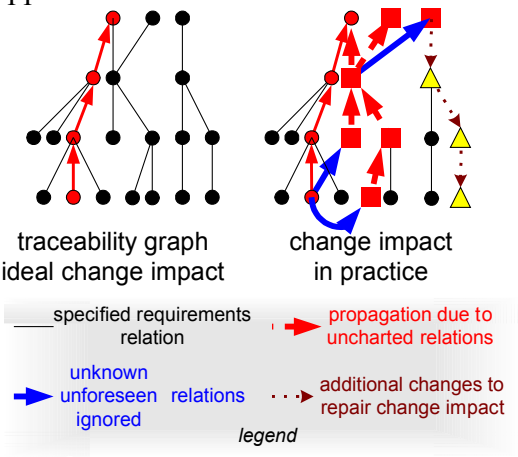

Figure 11. Impact of changes according to the documentation and in actual practice.

These kinds of effects make the integration into a blood, sweat and tears activity. The system is constantly changing while it is being integrated. These changes also continuously change subsystem and system performance, undoing the previous integration progress. Availability of sufficient regression tests helps to cope with this problem.

COTS, Systems-of-Systems, and Outsourcing. Extensive use of Commercial-Of-The-Shelve (COTS) products makes integration somewhat easier, if these products are transparent, mature, stable and if the products fit in the overall architecture. The disadvantage from design and

integration point of view is that the internals of these products are often unknowns. In other words most acquired external products are not transparent from integration perspective. These products are used as black boxes. The designer, nevertheless, will need information about functionality and performance beyond the black box specification. Many designers build in many implicit assumptions about the used product, without even being aware of it themselves. A feasible alternative is to document assumptions and to provide a basis for these assumptions by characterizing the product.

Characterization of a product is done by measuring and modeling the product to capture the required information. This characterization should be done as early as possible to determine COTS system characteristics or to confirm assumptions about COTS systems. Ideally this information is fed back into the supplier's specification. In the COTS market, however, suppliers are often unwilling to extend or to dedicate their specifications. Without anchoring this information at the supplier it is a vulnerability of the system design. The supplier might change the product implementation, without changing the specification, but impacting the earlier characterization.

The systems in systems-of-systems (SoS) look a lot like COTS subsystems, including the above mentioned issues. However, the individual systems in SoS often maintain their individual functions and often follow their own life cycle. In other words the individual systems evolve (change from function, form and performance), without explicitly managed change control at SoS level. The SoS level functionality and performance tends to emerge. If SoS functionality and performance is critical, then some creation process at SoS level is required, including integration. The challenge is to obtain acceptable predictability at SoS level, while maintaining sufficient autonomy at system level.

We have observed that in SoS situations where nobody takes responsibility for integral performance at SoS level, the outcome is indeed emerging. However, when one of the involved parties fills the integration vacuum, then key performance targets are better controlled. In the healthcare and manufacturing domains this integration role is often taken by work or process flow oriented suppliers. The equipment suppliers themselves are more focused on the systems level rather than the SoS level.

In outsourcing also the same issues play as in COTS. The integration strategy depends on the chosen outsourcing model: black-box supplier, co-designer, co-manufacturer, et cetera. In general the main challenge is to agree on early deliverables for integration. Most common mistake is to wait until the entire deliverable is finished and tested; this results in discovering integration problems late and lots of late rework as a consequence. Even in the case of black box outsourcing intermediate (grey box) deliverables must be organized for integration.

System Integration Lessons Learned

Typical System Integration Problems. Experience in many integration phases resulted in the observation of a typical order when integration problems occur. This order is:

- The (sub) system does not build
- The (sub) system does not function
- Interface errors
- The (sub) system is too slow
- Problems with the main performance parameter, such as image quality
- The (sub) system is not reliable

- The system does not operate well in its context

System does not build. Typically none of these problems should occur, but despite mature processes all of them occur in practice. The failure to build the system at all is often caused by the use of implicit know-how. For example a relatively addressed data file that resides on the engineers workbench, but that is not present in the independent test environment. As a side remark we observe the tension between using networked test models. Network connections shorten software change cycles and help in troubleshooting, however at the same time the type of problems we discussed here may stay invisible.

System does not function, interface problems. The next phase in integration appears to be that individual components or functions work, but cease to function when they are combined. Again the source of the problem is often a violated implicit assumption. This might relate to the third problem, interface errors. The problem might be in the interface itself, for instance different interpretations of the interface specification may result in failures of the combination. Another type of problem in this category is again caused by implicit assumptions. For example the implementation of the calling subsystem is based on assumed functionality of the called subsystem. It will be clear that different behavior of the called subsystem will cause problems for the caller. These types of problems are often not visible at interface specification level, because none of the subsystem designers realized that the behavior is relevant at interface level.

Performance Challenges. Once the system gets operational functionally, then the non-functional system properties become observable. The first problem that is hit in this phase by integrators is often system performance in terms of speed or throughput. Individual functions and components in isolation perform well, but when all functionality is running concurrently sharing the computing resources then the actual performance can be measured. The mismatch of expected and actual performance is not only caused by concurrency and sharing, but also by the increased load of more realistic test data. On top of these problems non-linear effects appear when the system resources are more heavily loaded, worsening overall performance even more. After some redesigns the performance problems tend to be solved, although continuous monitoring is recommended. Performance tends to degrade further during integration, due to added functionality and solutions for other integration problems.

When the system is both functional and well performing, then the core functionality, the main purpose of the product, is tested extensively. In this phase the application experts are closely involved in integration. These application experts use the system differently and look differently at the results. Problems in the critical system functionality are discovered in this phase. Although these problems were already present in the earlier phases, they stayed invisible due to the dominance of the other integration problems and due to the different perspectives of technical testers and application experts.

During the last integration phase the system gets used more and more intensively. The result is that less robust parts of the design are touched causing system crashes. A common complaint in this phase is that the system is unreliable and unstable. Part of this problem is caused by the continuous influx of design changes triggered by the earlier design phases; every change also triggers new problems.

The system does not operate well in its context. This may be caused by misunderstanding of the context, misrepresentation of the context during integration, unforeseen emergent behavior, or changes in the context during the project. This emphasizes the need to integrate with a

(realistic) context as early as possible during the integration.

From System to (Intelligent) Enterprise

The systems integration process we discussed so far works for typical projects with hundreds to thousands of engineers, tens of suppliers and several departments of the customer. If we increase the system scope to enterprise level, then all these quantities change with and order of magnitude: more engineers, more suppliers, more departments (more users, more enterprise processes). On top of this increased scope more (intelligent) functionality is added. In terms of uncertainties, unknowns and unforeseen aspects this translates in increases of several orders of magnitude. As a consequence no human being, nor any small team of chief system engineers, is capable of overseeing such a scope. These observations increase the importance of the integration process even more. All measures provided in this paper are not yet sufficient to deal with the increased integration complexity. Learning in practice seems to be the only way forward. Where learning implies: doing, critical reflection and improving.

Acknowledgements

Dinesh Verma stimulated the creation of this paper and provided useful inputs. The reviewer comments on the submitted draft provided valuable inputs for improvement.

Author Biography

Gerrit Muller.

Gerrit Muller received his Master's degree in Physics from the University of Amsterdam in 1979. He worked from 1980 until 1997 at Philips Medical Systems as system architect. From 1997 to 1999 he was manager System Engineering at ASML. From 1999 - 2002 he worked at Philips Research. Since 2003 he is working as senior research fellow at ESI (Embedded Systems Institute). In June 2004 he received his doctorate. The main focus of his work at ESI is on System Architecture methods and on education of future System Architects. Special areas of interest are:

- ways to cope with the exponential growth of size and complexity of systems. Examples of methods to address the growing complexity are product lines and composable architectures.
- the human aspects of systems architecting (which in itself is a crucial factor in coping with the above mentioned growth)

All information (System Architecture articles, course material, curriculum vitae) can be found at:

Gaudí systems architecting http://www.gaudisite.nl/