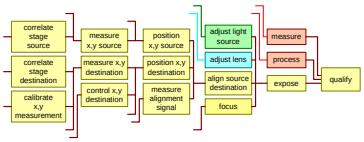
System Integration How-To

-



Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

This paper has been integrated in the book "Systems Architecting: A Business Perspective", http://www.gaudisite.nl/SABP.html, published by CRC Press in 2011.

Abstract

In this document 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.

version: 0.2 status: concept September 9, 2018

1 Introduction

Quality problems and delays are one of the symptoms of the troublesome relation between software and system The integration of software and hardware is in many organizations taking place when both hardware and software are nearly finished. Organizational boundaries propagate into the schedule causing too late integration of crucial technologies. Systems architects have to ensure that software-hardware integration starts very early.

System Integration is one of the activities of the Product Creation Process. The Product Creation Process starts with 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. Decomposition is an universal method used in organization, documentation and design. 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.

Integration is an ongoing flow of activities during the entire product creation. The nature of integration activities, however, shifts over time. Early in the project technologies or components are integrated, while at the end of the project the entire system is built and verified. In formal process descriptions[1] the description of product integration is mostly limited to the very last phase of the total integration flow, with a focus on the administrative and process aspects. We use the term integration in the broader meaning of all activities where decomposed parts are brought together.

In practice projects hit many problems that are caused by decomposition steps. Whenever an activity is decomposed the decomposed activities normally 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.

[1] for example NASA Procedural Requirements (NPR)

1.1 Goal of Integration

The goal of integration is 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 of 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.

1.2 Product Integration as part of Product Creation Process

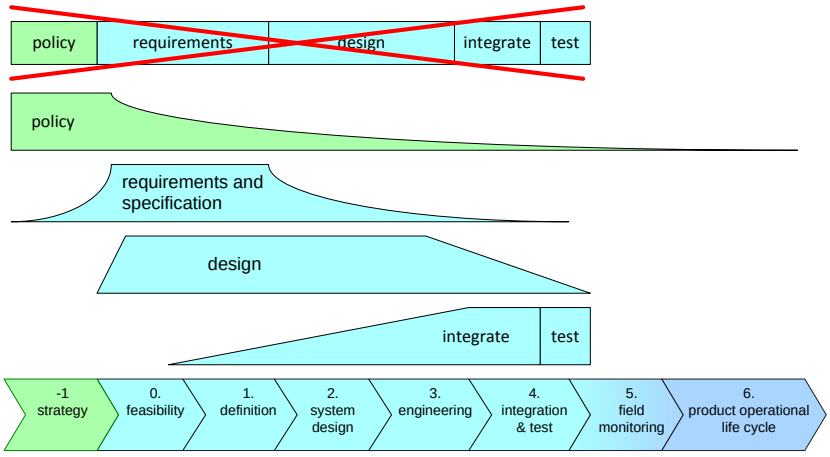


Figure 1: Typical Product Creation Process and the concurrency of engineering activities.

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. Much more practical is an approach with a shifting emphasis as shown in Figure 1. A comparable approach is Rational Unified Process (RUP), see [2] and for integration [3]. Note especially the long ramp-up of the integration, the focus of this chapter.

1.3 Integration in Relation to Testing

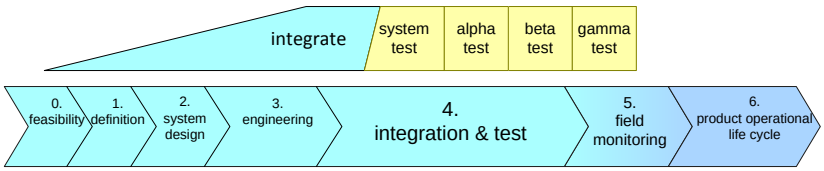


Figure 2: Zooming in on Integration and Tests

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

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. Before integration starts testing is applied at component level. 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 systems 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 tests, 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 chapter on integration,

with the main purpose to reduce risks in the testing phase by identifying (potential) problems as early as possible.

2 What, How, When and Who of Integration

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; the scope must be small enough to allow diagnosis in case of failure. If we bring thousands of components together into a system, then this system will fail for certain. But it is nearly impossible to find the sources of this failure, due to the multitude of unknowns, uncertainties, and ill-functioning parts.

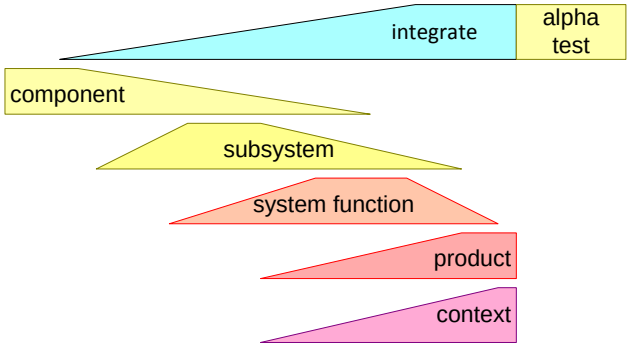


Figure 3: Integration takes place in a bottom-up fashion, with large overlaps between the integration levels.

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 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.

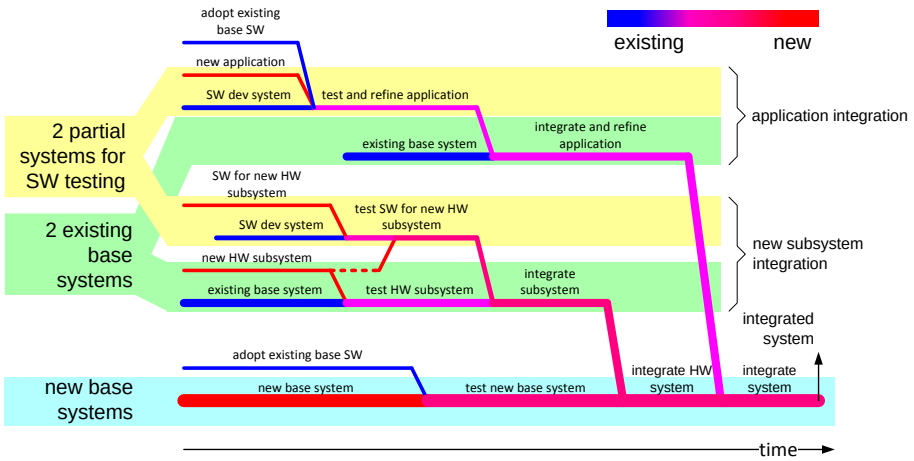


Figure 4: During integration a transition takes place from using previous systems and partial systems to the new system configuration.

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, dynamic positioning experts for instance, are needed for the different configurations. 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.

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 needs to be invested in this intermediate step? Special attention

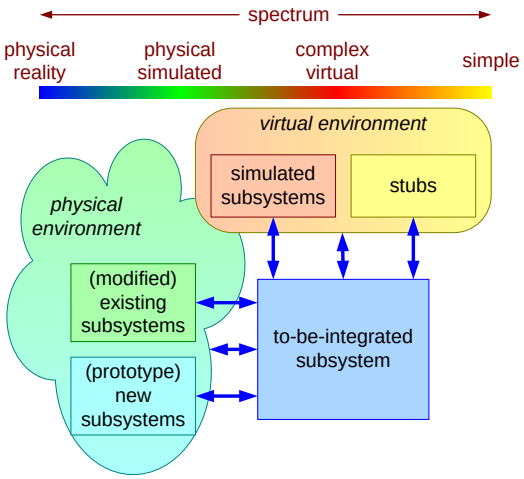


Figure 5: Alternatives to integrate a subsystem early in the project.

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?

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. 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.

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.

It is important to monitor the system performance regularly, since many engineers are still changing many parts of the total system. 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.

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

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

The approach described above 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 integration order of components and functions. The integration order is designed such that the desired critical system performance parameter can be measured as early as possible.

Figure 7 shows an example of a specific order of functions required to determine the image quality system performance parameter of a wafer stepper. 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.

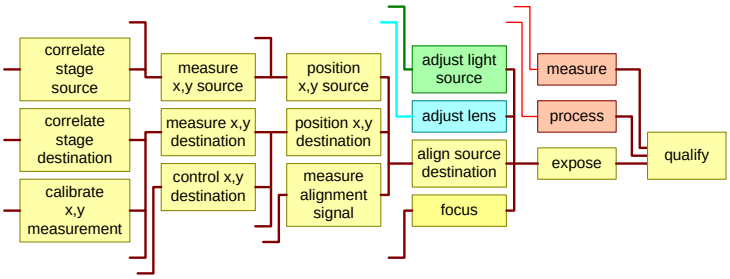


Figure 7: Example of small part of the order of functions required for the image quality system performance parameter of a wafer stepper.

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.

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 does not 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.

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

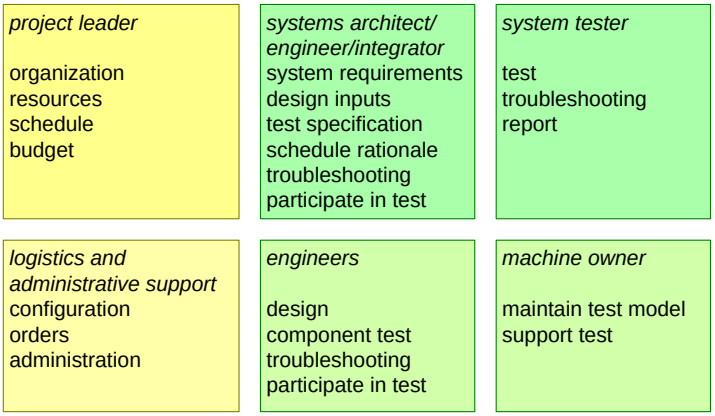


Figure 8: Roles and responsibilities during the integration process.

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 chapter.

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*, *systems 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[2]. 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 systems engineer. The integral perspective of this role results in a natural contribution to the troubleshooting.

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

[2]Critical characteristics for architects are the balance theoretical versus hands-on, conceptual versus implementation, creative and diverging versus result driven and converging. During integration the emphasis must be on hands-on, implementation, and result driven an converging.

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.

3 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.

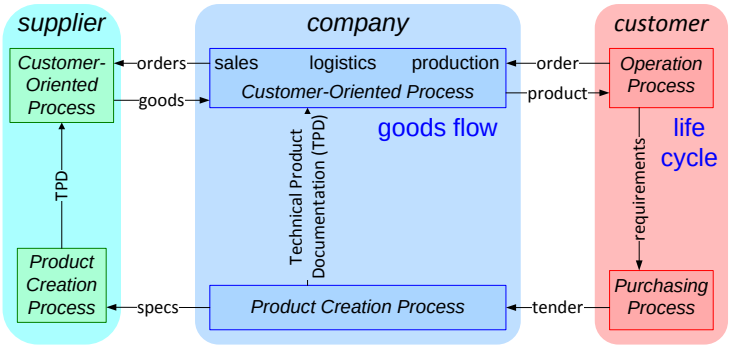


Figure 9: Simplified Process diagram that shows processes that are relevant from configuration management perspective.

Basically, the internal *Customer Oriented* and *Product Creation* processes are linked to the related supplier and customer processes. 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 flow with life cycle considerations. The final product as used operationally by customers also has its own life cycle.

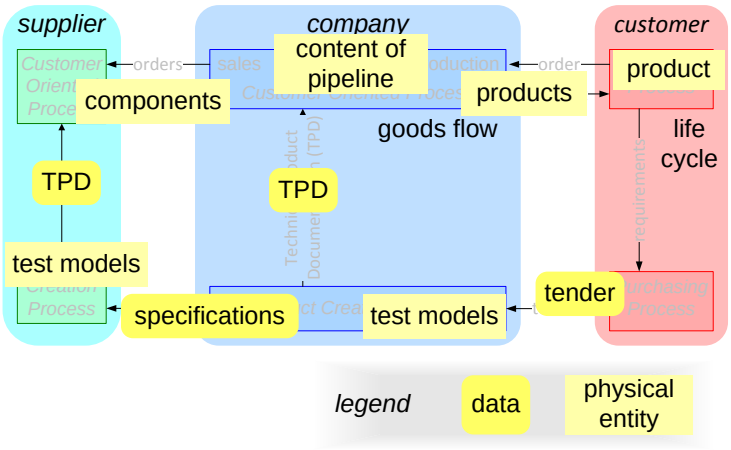


Figure 10: The simplified process diagram annotated with entities that are under configuration management.

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: information and physical items. The information entities are normally managed by procedures and computer based 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 effort in processes helps, however, sometimes more effort in processes results in more latency and more work-around behavior; unfortunately, there is no silver bullet for configuration management processes.

The main configuration management entities during integration are the test models. 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 single provisionally 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.

4 Typical Order of Integration Problems Occurring in Real Life

Experience in many integration phases resulted in the observation of a typical order when integration problems occur. This typical order is shown in Figure 11.

1. The (sub)system does not build.
2. The (sub)system does not function.
3. Interface errors.
4. The (sub)system is too slow.
5. Problems with the main performance parameter, such as image quality.
6. The (sub)system is not reliable.

Figure 11: Typical Order of Integration Problems

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.

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 than assumed behavior of the called subsystem may 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.

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 exercised more 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.

5 Acknowledgements

Dinesh Verma stimulated me to write this paper. Roland Mathijssen provided feedback and citations.

References

[1] Gerrit Muller. The system architecture homepage. http://www.gaudisite.nl/index.html, 1999.

[2] Wikipedia. Rational unified process (rup). http://en.wikipedia.org/wiki/Rational_Unified_Process, 2006.

[3] Wikipedia. Rup test discipline. http://en.wikipedia.org/wiki/Rational_Unified_Process#Test_Discipline, 2006.

History

Version: 0.2, date: August 6, 2010 changed by: Gerrit Muller

- textual updates
- changed status to concept

Version: 0.1, date: July 18, 2006 changed by: Gerrit Muller

- Added explanation of abbreviations
- added references
- added component testing before integration
- added footnote about critical characteristics of system architect during integration

Version: 0, date: July 4, 2006 changed by: Gerrit Muller

- Created, no changelog yet