Selecting a suitable system architecture for testing and integration

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2007-13

Selecting a suitable system architecture for testing and integration

I.S.M. de Jong, R. Boumen,
J.M. van de Mortel-Fronczak and J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2007-13
Eindhoven, February 2008
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

A system architecture is selected in the early design phases of a product. A trade-off is made between the most important architectural views during this selection process. The required system functionality is realized in an architecture which is maintainable, extendible, manufacturable, testable, integratable, etc., etc. This work investigates how an architecture can be selected, such that it is testable and integratable. The elements of an architecture which is suitable for testing and integration are introduced first. These elements are: components, interfaces between components and a layering.
The division of the system into components determines the how the system can be integrated and how many integration steps are required. Next to that, not all components need to be selected for system level integration and testing. Some, low-risk, components are integrated and tested on a lower level or not tested at all. The selection of components to be considered for integration and testing also influences which interfaces are considered.
The selection of an interface infrastructure influences integration and testing, next to the interfaces which result from component and interface selection. The interface infrastructure can reduce or increase the number of interfaces in the system. An interface infrastructure could also introduce that specific connectors need to be developed resulting in additional risk and more required testing.
And finally, a layering defines how the system, consisting of components and interfaces, is clustered. This layering reduces the complexity of the system and therefore the complexity of the integration and test plan. The layering for integration and testing can be defined fairly late in the development process just before integration and testing begins. Next to that, the layering for integration and testing can be different than the normal organizational or functional layerings of a system. More layerings can be defined and used next to each other.

1 Introduction

System architecting [1] is the process of creating an architecture for a system. The resulting architecture is a trade-off between the most important architectural views, like functionality, maintainability, extendability, etc. A list of these architectural views, so called quality attributes, that need balancing is given in the ISO-9126 standard [2]. Two of these quality attributes are *testability* and *manufacturability*. Testability fits within our definitions and defines if a system is well testable or not. Manufacturability in the context of ISO-9126 is a much broader term than the term integratability[1], which we prefer. Only a limited number of architectural views is considered when an architecture and design are defined. Testability and integratability are often not the most important views that are considered. Consequently, the resulting architecture does not reflect testability and integratability very well. Two solutions exist for this problem. The first solution defines integratability and testability as the most important architectural views of a system and the entire system architecture and design are centered around this view. Although, this solution results in a system that is designed for integration and testing, it requires a specific business case where this is beneficial. For most systems, integration and testing are not the most important architectural views.
The second solution uses an existing architecture, design and resulting integration and test sequence. The architecture and design are then improved based on the performance of the integration and test sequence. This, more iterative, solution is the basis for this paper. The structure of this report is as follows. First, definitions of an 'architecture' are discussed in Section 2. Then, Section 5 defines methods and guidelines for the selection of components, interfaces and layerings. These guidelines are illustrated with example systems. Conclusions are given in Section 9.

2 Architectures

According to the Oxford English Dictionary, the term *'Architecture'* is defined as: 1) the art or practice of designing and constructing buildings, 2) the style in which a building is designed and constructed, 3) the complex structure of something. The first definition describes the *process* of designing buildings, or systems, also called *architecting* in [3]. The second definition reflects the most commonly known definition of an architecture that is the style of a building. The, most relevant, third definition broadens the second definition, such that it is applicable to complex structures of 'something'. If we start with the first, building oriented, definition, then we see that the low-level components of most buildings are equal. For instance, bricks, mortar, nails and glue are used to build any type of house. The architectural 'style' defines if a building is functional and beautiful.
For systems, as defined as 'something' in the third definition, the same holds. The components (bricks) can be defined for a family of systems. This is also the case for the interfaces (mortar, nails, glue, etc.). Complex manufacturing machines are also built up using components and interfaces. The *style*, if applied to these systems determines if the system is functional and beautiful. The style determines how the components interact and are able to function together. The architectural style also determines how well the system can be split up into smaller sub-systems and how well this work-break-down-structure can be communicated and maintained throughout the development process: the beauty of the architecture.
The architectural style also determines how well the system can be split up into smaller sub-systems and how well this work-break-down-structure can be communicated and maintained throughout the development process: the beauty of the architecture.

[1] The word integratability is first used by SUN microsystems to indicate the easy integration capabilities of the Java™ platform. The word integratability is not defined in a dictionary at this moment. Other words describing integratability could be synthesizability or assembleability. Both words have different meanings in different domains and are also not defined in a dictionary.

3 System integration model

The system integration models describe a system. These models are used for sequencing and improvement in the integration and test planning method. The architecture of a system consists of components and interfaces. Both the components and interfaces are represented by the system integration model.
The system integration models should fulfill the input-output requirements of the integration and test-diagnose-fix tasks. The output of an integration and test-diagnose-fix task is the input of the next task. Integration and test-diagnose-fix tasks are combined into integration and test sequences. For instance, an assembly task can be followed by another assembly task or a test-diagnose-fix task. A component in the system integration model that is the output of the assembly task should be of the same type as the input of the next task.
System integration models should enable performance modeling of the integration and test sequence. Analyzing the performance of an integration and test sequence is one of our goals. Therefore, the integration models should support this goal.
System integration models should support abstraction if a system contains many components and interfaces. Integration and test sequencing is a problem for large systems in particular. These systems consist of many components and interfaces. The integration and test models should support the combination of smaller models into large models, such that large systems can be modeled easily.
The system integration model A is modeled as a five-tuple containing: models of the components in the system Γ, models of the interfaces in the system $\mathcal{X}_F$, the relation between the components and interfaces $R_{\gamma,\mathcal{X}_F}$, a so-called *layering* $\mathcal{L}$ and the objective and constraints of the integration and test sequence Obj. Thus, $A = (\Gamma, \mathcal{X}_F, R_{\gamma,\mathcal{X}_F}, \mathcal{L}, Obj)$, where Γ represents a set of component models γ. The elements of the system integration model are described in detail below.

The component model

A component in the system is modeled as a *system test model* D introduced in Section 4, and properties: $\gamma = (D, \varphi_{dev}, C_{dev})$. The system test model D describes the component in terms of test cases and fault states and is described in Section 4. The duration and cost of developing a component γ is modeled as φ_{dev} and C_{dev} respectively. The development duration is modeled as the duration between the start of the integration and test sequence and the moment that component development is ready. The end moment defines when the integration and test sequence can progress. The cost of development can also be modeled. This cost is added to the overall cost of the integration sequence. Often, the cost is set to 0, because only the cost of integration and test tasks is of interest when integration and test sequences are analyzed and compared and not the cost of developing a module that is just an offset in the total cost.

The interface model

An interface is also modeled as a system test model. The duration and cost of developing the interface is also taken into account. The interface is modeled as a triple: $\mathcal{X}_F = (D, \varphi_{dev}, C_{dev})$. The interface risk is modeled explicitly. Even for interfaces with very low failure probabilities and risk, the risk needs to be modeled. The combined risk of many low risk interfaces results in an overall, system level, remaining risk that could be too high. The low risk interfaces are also modeled by explicitly modeling the interfaces. Note that a broad multi-disciplinary definition of interfaces is considered.

Relationship between components and interfaces

The relationship between components and interfaces is modeled as a set of triples that contain the models of two components and their interface $R_{\gamma,\mathcal{X}_F} \subseteq \Gamma \times \mathcal{X}_F \times \Gamma$, which is a set of triples representing the connections that can be made between the components in the system. Two components can be connected by two or more interfaces. This is modeled either as separate relations or by combining the system test models of the interfaces into a single model. The interfaces between one component and many other components are modeled as separate relations.

Layering

The minimal set of models that is required to create integration and test sequences consists of the component model, the interface model and the model of the relationship. The model of the relationship between components is used to obtain the sequence. In this way, any combination of tasks can be made. The number of sequences obtained using these five tasks and the component models is infinite, because of the possibly infinite combinations of disassembly and assembly tasks, the copy task and the repeatable test-diagnose-fix tasks. Even if the copy and disassembly tasks are not taken into account and it is assumed that test-diagnose-fix are not repeated, then the number of possible sequences leading to a complete and tested system is huge for systems with many components and interfaces. The number and type of possible sequences can be constrained by using *layering*.
For this reason, a *layering* $\mathcal{L}$ in the system, consisting of sets of components Γ, is defined: $\mathcal{L} \subseteq \mathcal{P}(\Gamma)$. The relation between the components in the groups and between the groups are derived using the component-interface relation $R_{\gamma,\mathcal{X}_F}$. A set of components in a layering can contain a component that is also present in another set of components. In other words, sets of components can overlap. A layering is often used to describe a functional clustering opposed to the hierarchical or mechanical clustering. The physical components can contribute to one or more of these functional clusters. In this way, functional integration and test sequences can be derived from the system integration model.

Objectives and constraints

The objective for an integration and test sequence describes whether the resulting system should be integrated and tested quickly (T), cheaply (C) or with a high quality (Q). The constraints of the integration and test sequence describe if the executed integration and test sequence should not exceed a fixed duration, a fixed cost level or a risk level. Objectives and constraints are both expressed in terms of time, cost and remaining risk. The time, cost and remaining risk objectives are described relative to each other, such that it is indicated which objective is most important. The constraints of an integration and test sequence determine the limits that may not be exceeded. A constraint on duration means that the sequence should be finished at a certain deadline. A constraint on cost means that a certain cost level may not be exceeded and a constraint on quality means that the integration and test sequence should lead to a certain quality level. The objectives of a complete integration and test sequence are described in a similar fashion.

4 System test models

Components and interfaces in a system are modeled as a *system test model*. A system test model describes a component or interface as a combination of possible fault states, test cases, the coverage of the test cases on the fault states and the properties of the fault states and test cases. In this section, the system test model is described in detail. The practical implication

of every modeling element is discussed. The model completeness is described and stepwise improvement of the model is explained.
The test cases, fault states and their properties are modeled as a so called *system test model*. The *system test model* is a ten-tuple consisting of those elements that are relevant for a test-diagnose-fix task. The ten-tuple is derived from a basic system test model used by [4] for sequencing diagnosis tasks. Boumen [5] uses a system test model, based on the basic test model, for probabilistic test sequencing. The basic test model is defined as $D = (T, S, C, P, R_{ts})$. Our *system test model* D is defined as a ten-tuple $(T, S, (C_T, \varphi_T), (C_D, \varphi_D), (C_F, \varphi_F), (C_{AF}, \varphi_{AF}), P, I, U, R_{ts})$, where:

- T is a finite set of k tests.
- S is a finite set of l fault states.
- $C_T : T \to \mathbb{R}^+$ gives for each test in T the associated cost of performing that test.
- $\varphi_T : T \to \mathbb{R}^+$ gives for each test in T the associated duration of performing that test.
- $C_D : T \to \mathbb{R}^+$ gives for each test in T the associated cost of diagnosing the failed test.
- $\varphi_D : T \to \mathbb{R}^+$ gives for each test in T the associated duration of diagnosing the result of the test if the result is *fail*.
- $C_F : S \to \mathbb{R}^+$ gives for each fault state which is to be fixed the cost of fixing that fault state.
- $\varphi_F : S \to \mathbb{R}^+$ gives for each fault state in S the associated duration of developing a fix for the fault state.
- $C_{AF} : S \to \mathbb{R}^+$ gives for each fault in S the associated cost of applying the fix on the system under test.
- $\varphi_{AF} : S \to \mathbb{R}^+$ gives for each fix which is to be applied the associated duration.
- $P : S \to [0..1]$ gives for each fault state in S the *a priori* probability that the fault state is present, which is the probability that a certain fault is present.
- Impact can be a value relative to each fault state or a specific value related to a property of the system under test.$I : S \to \mathbb{R}$ gives for each fault state in S the impact of the fault state if the fault state exists in the system under test. Impact can be a value relative to each fault state or a specific value related to a property of the system under test.
- $U : S \to [0..1]$ gives for each fault state in S the uncertainty that the fault state is present, which is 100% if no test case has been executed that covers the fault state.
- $R_{ts} : T \times S \to [0..1]$ gives for each test t and fault state s the coverage of the test t on fault state s.

4.1 Fault states

Fault states describe *possible* faults in the system, including the associated failure probability and impact. Fault states are assumed to be independent of each other. Additional properties are modeled per fault state, like the uncertainty about the fault state (used for reliability qualification), the duration and cost of fixing the fault state and the duration and cost of applying the fix on the system under test. A set of fault states can be modeled as a single combined fault state with adjusted properties [6] for the combined fault state, like for instance a higher failure probability. A broad definition of fault states is used when modeling a system. Fault states could be based on existing requirements, known failures from previous system releases, failure mode effect analysis (FMEA/FMECA) [7], expert knowledge and even more detailed fault state sources like manufacturing failure databases or static code analysis tools for software systems.

4.2 Test cases

The detailed test procedure is a valuable source to determine the coverage of the test case.A test case is described in terms of its coverage on the defined fault states. The coverage of the test case on the fault states determines the capability of the test case on system level. Note that the detailed test procedure is a valuable source to determine the coverage of the test case. The other properties that are modeled for a test case are execution duration, execution cost, diagnosis duration and cost. A set of test cases can be modeled as a single test case with a higher coverage, a longer duration and higher cost.

4.3 Coverage

The coverageis modeled as the probability that a test case t can detect a fault state s. The coverage of a test case on a fault state can be estimated by counting for a test case how many of the functions are covered by the test case. This number of covered functions divided by the total number of functions in the system that this test case covers, such that the coverage is calculated. The coverage is used to determine the reduction of the failure probabilities of the fault states that are covered by a test case when the test case *passes*.

4.4 Uncertainty

The uncertainty is defined as the so-called *subjective* uncertainty [8] and describes the lack of knowledge about the system. This lack of knowledge is a property of the analyst that analysis the test-diagnose-fix task or integration and test task. The *stochastic* uncertainty, which is another definition of uncertainty that is often used, is a property of the system itself and describes the variability of the behavior of a system. The stochastic uncertainty of a test-diagnose-fix task (or a complete integration and test sequence) is measured in terms of the performance indicators: duration, cost and remaining risk. The subjective uncertainty is used in the reliability test planning method.

4.5 Risk

Our measure for product (or system) quality is risk. The risk that is present in a system is the sum of the risk of the fault states that are possibly in the system. The risk that a fault state is in the system is defined as the product of the probability that the fault state is in the system and the impact of the fault state when it is present: $R(s) = P(s)I(s)$ [9–11]. Since fault states are assumed to be independent, the risk of all fault states in the system can be added such that the system risk is obtained: $R = \sum_{s \in S} R(s)$.

The required probability can be estimated or obtained from historical data.The required probability, a property of the product, can be estimated or obtained from historical data. The required impact value is not a property of the product. The impact of the presence of a fault state is estimated, in case the fault state occurs after the (integration) and test sequence is finished. If the product is delivered to customers once the integration and test sequence is finished, then the impact for customers is estimated. If the product is released for the next task in the integration and test sequence, then the impact for the remainder of the integration and test sequence is estimated.

The impact of a number of fault states in a system must be estimated in relation to each other. For instance, if a system is modeled for overlay testing, then the impact of all fault states in the system should be modeled in terms of overlay [nm]. In this way, test cases can be selected that reduce the overlay risk as much as possible.

4.6 Example telephone system

As an example, a test model is defined for a common telephone. The telephone consists of a handset, a cable and a device. Each of these modules can contain a fault. The interfaces between the modules can also contain a fault. Hence, in total, 5 fault states are defined. Additionally, 6 test cases are defined covering the 5 fault states. A graphical view of the telephone is given in Figure 1.

The set of five possible fault states in the telephone is: $S = \{s_1, s_2, s_3, s_4, s_5\}$.

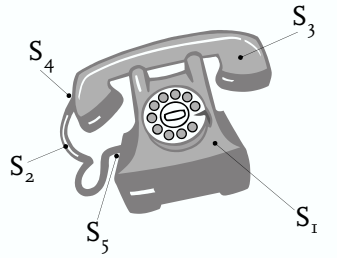


Figure 1: Telephone system

- s_1 the device contains a fault
- s_2 the cable contains a fault
- s_3 the handset contains a fault
- s_4 the interface between the cable and the device contains a fault
- s_5 the interface between the handset and the cable contains a fault

The set of six tests is defined to cover these fault states is: $T = \{t_0, t_1, t_2, t_3, t_4, t_5\}$.

- t_0 tests the complete phone system
- t_1 tests the device
- t_2 tests the cable
- t_3 tests the handset
- t_4 tests the device and the cable
- t_5 tests the handset and the cable

A matrix representation of the system test model D, including the properties per fault state and test, is given in Table 1. The coverage of a test case on a fault state is placed in the matrix.

S / T	t_0	t_1	t_2	t_3	t_4	t_5	P	I	U	C_F	φ_F	C_{AF}	φ_{AF}
s_1	0.2	0.5	0	0	0.3	0	10%	7	1.0	2	2	1	1
s_2	0.2	0	0.5	0	0.3	0.3	10%	5	1.0	2	2	1	1
s_3	0.2	0	0	0.5	0	0.3	10%	3	1.0	2	2	1	1
s_4	0.2	0	0	0	0.3	0	10%	3	1.0	2	2	1	1
s_5	0.2	0	0	0	0	0.3	10%	3	1.0	2	2	1	1
C_T	3	1	1	1	2	2							
φ_T	6	2	2	2	4	4							
C_D	3	1	1	1	2	2							
φ_D	10	1	1	1	6	6							

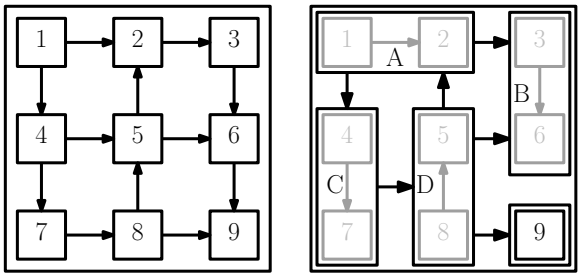
Table 1: A test model for the telephone example

Components, interfaces and a layering are the elements of an architecture that are relevant for integration and testing. The remainder of this section describes how components, interfaces and a layering can be selected such that they are suitable for integration and testing.

5 Selection of a suitable set of components

A suitable set of components, in the context of integration and testing, means that the components are integratable and testable. The aspects that influence the integratability and testability of an architecture are the number of components and the size of the components. The break-down of the system into components determines how many components need to be integrated. Not all components need to be taken into account for the system level integration and test sequence. Small risk, low-level components could be ignored in the system level integration and test sequence. It is then assumed that these components are integrated and tested on sub-system level. How to select the number of components suitable for integration and testing is discussed below.

The number of components in the system determines how many integration tasks are required to build the complete system. More integration tasks could lead to longer integration and test sequences, depending on the possible parallelism in the sequence. An example system is broken down into nine components in Figure 2(a) and into five components in Figure 2(b). The resulting integration sequences for the nine and five component system are depicted in Figure 3 and Figure 4 respectively. The number of assembly tasks in the sequence of the nine-component system is larger and therefore probably takes more time, depending on the duration of the individual tasks.



(a) Example system split up into nine components

(b) Example system split up into five components

Figure 2: An example system split up into nine and five components

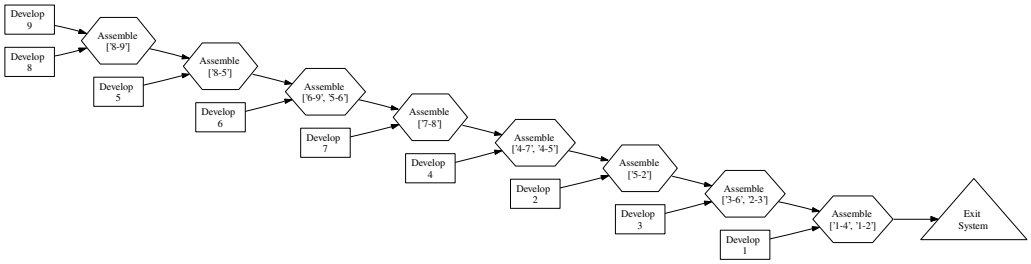


Figure 3: Integration sequence of the system depicted in Figure 2(a)

Splitting up a system into more components leads to longer integration sequences. A small number of components results in components that are too complex, too large, hence contain too much risk. Higher risk results in lengthy and costly test-diagnose-fix tasks, because the risk in the system needs to be reduced. Exclusion of risk can be done in two ways. First, *passed* test cases reduces the risk in the system because it is proven, by a passed test case, that

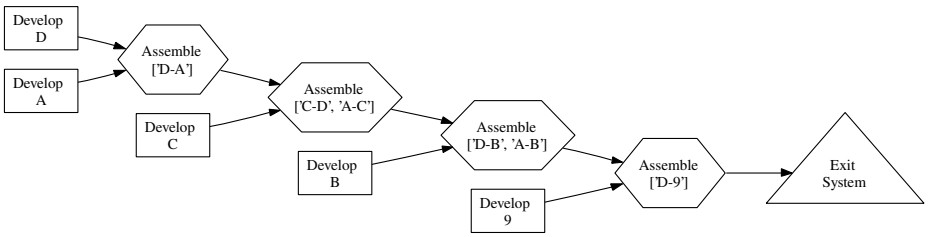


Figure 4: Integration sequence of the system depicted in Figure 2(b)

some risk is not in the system. Second, *failed* test cases result in a diagnosis of the failure and a fix of the fault. Fixing a failure also reduces the risk, so does the diagnosis that a fault is not in the system. Both ways of reducing risk cost more time/money when the risk in the system is high. Consequently, a suitable number of components is a balancing act between the number of components to select and the size of the resulting integration sequence. Some guidelines for this balancing act are given below.

5.1 Guidelines for selecting components

Selecting the number of components to accommodate the integratability and testability is a balancing act. An algorithm that determines the optimal integration and test sequence, given the components and interfaces is NP-hard [12, 13]. Comparing a number of component selections is therefore also NP-hard. Nevertheless, some guidelines for component selection are given below. The first guideline is most important, the other guidelines are exceptions on the first guideline.

- The risk of a component should guide the split up of a component. High risk components should be split up to accommodate: parallel integration, parallel testing and a shorter test-diagnose-fix task because less faults are found.
- Parallel integration and testing is only beneficial when the resulting integration and test sequence has a shorter duration than the original sequence. The resulting integration and test sequence can be longer in one of the following cases:
  - The development duration of both new components is longer, because additional time is spent on interface definition and development. This can be the case if two development groups are involved and/or the interface decisions are more difficult to make.
  - The assembly of the two components is difficult (because the interface agreements are more difficult to make).
  - The combination of components and interfaces is the risky part of the development, while the individual components are relatively simple. The additional parallel test-diagnose-fix tasks have no benefit in this case, because reducing the risk is done in the last test-diagnose-fix task and is therefore on the critical path in the integration and test plan.
- Splitting up components is only beneficial if the interfaces between the new components 'allow' splitting up these components. The properties of interfaces that 'allow' splitting up components are:
  - The number of (different) interfaces between the split up components should not increase, because the increase in interfaces could result in an increase of the duration of the assembly phase.
  - The interface risk after splitting up a component into two components should be low. A component split-up that results in high risk interfaces results in a longer

test phase after assembly. Examples of high risk interfaces after component split up are interfaces that are to be developed newly, because of the component split up, or interfaces with extreme requirements in terms of speed, tolerance, etc.

6 Selection of a suitable set of interfaces

Two approaches can be followed to select the suitable (integratable/testable) set of interfaces. The first approach is directly related to the selection of components, while the second approach involves the selection of a different interface paradigm. The first approach is a result of the component selection as discussed in the previous section. Splitting up components leads in general to more interfaces. E.g. dividing a single component with two external interfaces into two components leads to at least one additional interface between the two components. The selection criteria for suitable interfaces are similar for component selection in this setting. Dividing a component into two smaller components reduces the risk in the component. Testability is increased by reducing the number of interfaces, interface usage and reducing the risk involved with this type of interfaces. Integratability is increased by reducing the time required to connect, disconnect and reconnect two components to each other, i.e., the assembly time and cost are minimal. Dividing the component into two smaller components with (too) many interfaces between these components increases the risk, because of possible problems with definition, implementation and utilization of these interfaces.
The second approach involves the selection of a different *interface paradigm* that increases the integratability and testability. Testability is increased by reducing the number of interfaces, interface usage and reducing the risk involved with this type of interfaces. Integratability is increased by reducing the time required to connect, disconnect and reconnect two components to each other, i.e., the assembly time and cost are minimal.

6.1 Example: A mechanical interface in the ASML wafer scanner

An example of a *mechanical* interface in an ASML wafer scanner that is specially chosen to increase the testability and integratability is the so called 'cable slab' between the body of the wafer scanner and a wafer stage. Figure 5 depicts the mechanical interface. The wafer stage is

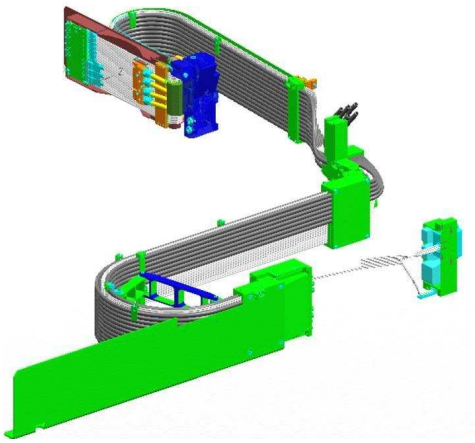


Figure 5: Cable slab

a sub-system of the wafer scanner, that needs to move in six degrees of freedom. Any contact of the wafer stage with the environment (the body) results in vibrations in the wafer stage system resulting in overlay and imaging problems. The required power, signal and airflow

to control and move the wafer stage is supplied via the 'cable slab'. A special interface has been designed such that the stage can easily be replaced and tested. A specially designed interface imposes additional risk, because no common-of-the-shelf (COTS) interface is used. Additional 'cable slab' and sub-system testing was required for the 'cable slab', while the cable slab itself reduced risk of overlay and imaging problems on system level. The cable slab is an example of an interface where additional time and cost is spent to reduce the higher level system risk.
Next, some guidelines for selecting an interface paradigm are given. An interface paradigm that is suitable for integration and testing:

- reduces the number of interfaces or the risk involved with interfaces,
- reduces the bandwidth used by the interfaces,
- reduces the assembly or disassembly duration and cost,
- improves the testability and diagnosability by adding additional measurement capabilities,
- improves the ability to assemble and disassemble components.

7 Selection of a suitable set of layerings

Components and interfaces are the only elements that are necessary to create an integration and test plan. However, the number of possible integration and test plans increases dramatically when the number of components (and interfaces) increases. The selected *layering* for a system reduces the complexity of creating an integration and test plan. The functional and organizational layering can differ from the layering that is chosen for the benefit of integration and testing. The functional and organizational layering can differ from the layering that is chosen for the benefit of integration and testing. Selecting a layering suitable for integration and testing means that the set of components and interfaces are grouped such that a good integration and test sequence can be created.

7.1 Example: Layering a car axle

An example car axle with wheels is depicted in Figure 6 and modeled as a four component architecture, depicted in Figure 7. The architecture consists of two wheels (W_1 and W_2) and two steering knuckles (SK_1 and SK_2). The axle itself is modeled as the interface between the two steering knuckles. The functional layering for this system could be defined as a layer for

Figure 6: Example car axle system

each side of the car as depicted in Figure 8(a). An organizational layering could be grouped

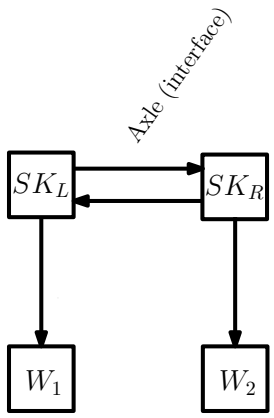
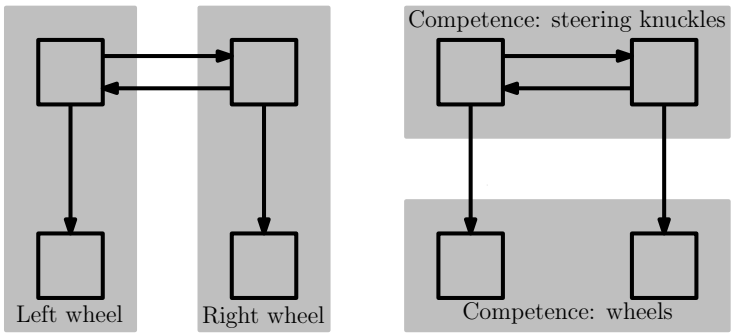


Figure 7: Car axle architecture

according to the two competences involved in this system: wheels and steering knuckles. The organizational layering is depicted in Figure 8(b). The integration sequences for the



(a) Functional layering of the car axle system

(c) Organizational layering of the car axle system

Figure 8: Two layerings of the car axle system

functional and organizational layering have a different duration, assuming that assembling each interface costs the same time in this example. Figure 9 depicts the resulting integration sequence for the layering according to the competence. The first two assembly tasks assemble a wheel and a steering knuckle. The last assembly task assembles both wheels and steering knuckles with their common interface, the axle. A total of four interfaces is created. However, the first two interfaces are created in parallel. If the creation duration of *each* interface is one time unit, then the total duration of this sequence is three time units.
The integration sequence according to the organizational axis is depicted in Figure 10. Now, the steering knuckles are assembled first, followed by an assembly of the wheels to the steering knuckles. The duration of this integration sequence is four time units, assuming again that the creation of *each* interface costs one time unit and testing is not taken into account.
This example does not include test-diagnose-fix tasks. It is assumed that the components are available at the start of the integration sequence. Development durations and cost of components need to be taken into account as well as the test-diagnose-fix tasks.
Some guidelines are given such that a suitable layering can be chosen.

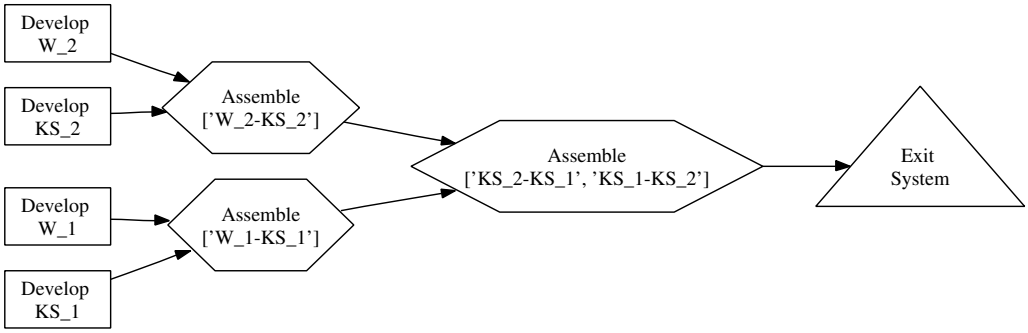


Figure 9: Integration sequence of the axle according to the functional layering

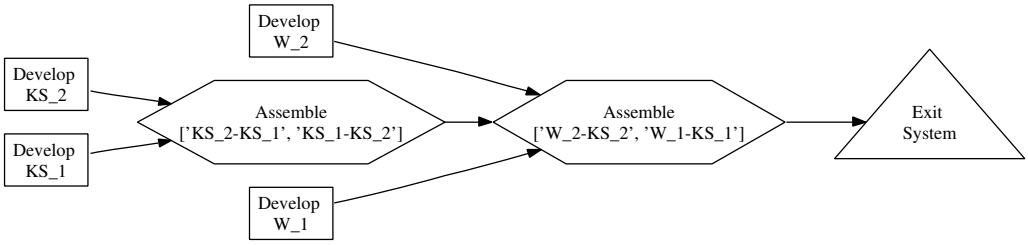


Figure 10: Integration sequence of the axle according to the organizational layering

- The layering chosen during design of the system is not necessarily the best layering for integration and testing.
- Selecting a new layering for integration and testing can be done late in the process. See the next sub-section for details.
- If all possible sequences of components and interfaces that are able to form a system are considered, the optimal sequence can be selected. However, this approach makes planning an integration and test sequence for large systems complex and time consuming. To reduce this complexity, layering can be applied.
- Take the interfaces between components into consideration when choosing a layering. A layering that divides the system, such that the integration and test sequence can be executed as much as possible in parallel is beneficial.
- Layers containing components that are not connected result in less optimal integration and test plans.

8 Illustration: TWINSCAN wafer scanner integration

The TWINSCAN™ wafer scanner platform of ASML was designed and developed from 1997 onwards. In 2000-2001, the first TWINSCAN™ systems were shipped to customers. The initial design was set up such that the (relatively large) sub-systems are easily recognizable within the system and that these sub-systems are evolvable. The sub-systems are recognizable for the purpose of system level training and understandability. Evolvability was important for the purpose of creating a platform that served as a basis for the quick development of new system types. The technical design and layering was centered around the most important aspect of the TWINSCAN™ system at that time: productivity.
The TWINSCAN™ system was designed using a sub-system oriented architecture/layering. The system level specifications were broken down into sub-system level specifications and so

on. The sub-systems could be identified easily in the system architecture. This architecture was suitable for breaking the high level problem into smaller problems and for managing this work-break-down structure.
However, system level integration requires cooperation of all sub-systems, such that the system function, run production, could be performed. A single sub-system contributes to the performance of many other sub-systems that together result in a system level performance. 'Life-of-a-wafer' describes in terms of *sub-functions* how production is run on a wafer scanner. The *sub-systems* in the wafer scanner contribute to one or more of the *sub-functions* in the 'Life-of-a-wafer' process.The goal at integration time, is to combine all these sub-systems, with their individual performance requirements, into a complete and performing system.
Around one year before the first shipment, the intended function, run production, of the wafer scanner was split up in around 15 sub-functions. These functions were related to each other by a process description called 'life-of-a-wafer'. 'Life-of-a-wafer' describes in terms of *sub-functions* how production is ran on a wafer scanner. The *sub-systems* in the wafer scanner contribute to one or more of the *sub-functions* in the 'Life-of-a-wafer' process. The focus during integration and testing shifted from sub-system development to sub-function and function integration and qualification. The integration and test sequence reflected this new layering, because milestones for these functions were planned and function owners were assigned.
Shifting from a sub-system development approach to sub-function integration, a new layering was not an easy task. Many developers were convinced or forced to use the 'new' way-of-working. Functional milestones did not seem like progress for upper management and customers who wanted to see the results in terms of throughput, overlay or imaging performance. However, the developed systems were shipped on time and a record breaking number of additional system types were shipped in the next year. Furthermore, the system reliability and availability was brought to the required levels twice as fast if compared with the previous platform. Afterward, it can be concluded that changing the layering very late in the process, with the purpose of creating a better integration and test plan, was successful for the TWINSCAN™ platform.

9 Conclusions

The selection of an architecture that is suitable for testing and integration is important, because this choice influences the integration and test sequences that can be created. By this selection, the performance of an integration and test sequence is influenced. Often, a single architecture is considered for a system under test. Testability and integratability aspects are balanced together with other architectural aspects. A suitable architecture for integration and testing is chosen, if testability and integratability are aspects of high importance. Otherwise, the resulting architecture leads to a sub-optimal integration and test sequence. Selecting an architecture that is suitable for integration and testing requires a component selection, an interface selection and a selection of layers.
Component selection is a balancing act between the number of selected components and the risk of these components. Selecting too many components leads to a large integration and test sequence, while selecting too few components results in an integration and test sequence with too much risk in the individual components and long and unpredictable test-diagnose-fix tasks. Component selection determines what level of abstraction is used for the system. Consequently, an integration and test sequence is obtained with more or fewer integration tasks.
Interface selection is related to component selection, because the selection of components determines what interfaces are selected. Interface selection by selecting a different interface paradigm could reduce the number of interfaces and the interface usage. Additionally, selecting an interface that does not fit in the interface paradigm could increase the number of interfaces and the usage of the interfaces.
Component and interface selections are, in principle, the only required aspects to consider

for a suitable integration and test architecture. The components and interfaces are the inputs for the integration and test planning process. However, the number of possible integration and test sequences for real life systems is large. Selecting a layering for a system reduces the number of possible integration and test sequences that are to be considered. Therefore, layering is an important selection mechanism.

Component, interface and a layering can be selected late in the development process, when the system architecture does not change because of the selection. Additionally, selecting a different interface paradigm should be done as early as possible, because a change in the interface paradigm often requires that the individual components need to be changed as well. Implementing a new interface paradigm could introduce additional risk. The additional risk can be minimized by selecting an implementation of the interface paradigm that is stable already.

Bibliography

[1] G. Muller, *System architecting*, G. Muller, Ed. Embedded Systems Institute, 2007. [Online]. Available: http://www.gaudisite.nl/SystemArchitectureBook.pdf

[2] ISO-9126-1, "Information technology - software product quality - part 1: Quality model," International Organization for Standardization, 2000-03-20.

[3] G. Muller, "CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity," Ph.D. thesis, Technische Universiteit Delft, Jan. 2004.

[4] K. Pattipati, S. Deb, M. Dontamsetty, and A. Maitra, "START: System testability analysis and research tool," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 6, no. 1, pp. 13–20, Jan 1991.

[5] R. Boumen, I. de Jong, J. Vermunt, J. van de Mortel-Fronczak, and J. Rooda, "Test sequencing in complex manufacturing systems," *IEEE Transactions on Systems,Man,and Cybernetics-Part A: Systems and Humans*, vol. 38, no. 1, pp. 25–37, January Jan. 2008.

[6] R. Boumen, "Integration and test plans for complex manufacturing systems," Ph.D. thesis, Eindhoven University of Technology, August 2007.

[7] J. Bowles, "The new SAE FMECA standard," *Reliability and Maintainability Symposium, 1998. Proceedings., Annual*, pp. 48–53, 19-22 Jan 1998.

[8] J. Helton, "Uncertainty and sensitivity analysis in the presence of stochastic and subjective uncertainty," *Journal of Statistical Computation and Simulation*, vol. 57, no. 1 - 4, pp. 3–76, April 1997.

[9] S. Amland, "Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study," *Journal of Systems and Software*, vol. 53, no. 3, pp. 287–295, 2000.

[10] S. Kaplan, "The words of risk analysis," *Risk Analysis*, vol. 17, no. 4, pp. 407–417, August August 1997. [Online]. Available: http://www.blackwell-synergy.com.janus.libr.tue.nl/doi/abs/10.1111/j.1539-6924.1997.tb00881.x

[11] S. L. Pfleeger, "Risky business: what we have yet to learn about risk management," *Journal of Systems and Software*, vol. 53, no. 3, pp. 265–273, Sept. 2000.

[12] R. Boumen, I. de Jong, J. Mestrom, J. van de Mortel-Fronczak, and J. Rooda, "Integration sequencing in complex manufacturing systems," Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, SE Report 2006-02, 2006. [Online]. Available: http://se.wtb.tue.nl/sereports

[13] ——, "Integration and test sequencing for complex systems," Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, SE Report 2007-07, 2007. [Online]. Available: http://se.wtb.tue.nl/sereports

Authors biography

I.S.M. de Jong has a B.Sc. in Laboratory Informatics and Automation from Breda Polytechnic. He has been a software engineer in various companies in the USA and The Netherlands. Since 1996 he has worked with ASML in systems testing, integration, release and reliability projects. His specialization is in the field of test strategy. Since 2003 he is an active member in the TANGRAM project and a Ph.D. student at the Eindhoven University of Technology. His research concerns integration and test strategies.

R. Boumen received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2004. During his work as a master student he worked in the field of supervisory machine control of lithographic machines. Since 2004 he is a Ph.D. student at the Eindhoven University of Technology. His research, embedded in the TANGRAM project, concerns test strategies.

J.M. van de Mortel-Fronczak graduated in computer science at the AGH University of Science and Technology of Cracow, Poland, in 1982. In 1993, she received the Ph.D. degree in computer science from the Eindhoven University of Technology, the Netherlands. Since 1997 she has worked as assistant professor at the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include specification, design, analysis and verification of supervisory machine control systems.

J.E. Rooda received the M.S. degree from Wageni+ngen University of Agriculture Engineering and the Ph.D. degree from Twente University of Technology, The Netherlands. Since 1985 he is Professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering of Eindhoven University of Technology, The Netherlands. His research fields of interest are modelling and analysis of manufacturing systems. His interest is especially in the control of manufacturing lines and in the supervisory control of manufacturing machines.