Tutorial Software as Integrating Technology in Complex Systems

-

logo
TBD

Gerrit Muller
Buskerud University College
Frogs vei 41 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

This tutorial describes the integrating value of software in complex systems. The extensive use of software technology to integrate other technologies has a significant impact on the product characteristics and on the product creation organization and process. This tutorial provides insight in the relation between software and the system, and it provides insight in the consequences for the product and the organization. Some recommendations are provided to cope with these consequences.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

All Gaudí documents are available at:
http://www.gaudisite.nl/

version: 0.1 status: concept June 23, 2016

1 Introduction

In this tutorial we will address the role of software in a complex system by using a waferstepper as an illustrating case. The material of this tutorial reuses previous articles and presentations of the Gaudí project. However about 50% of the material is new.

2 Case: the waferstepper and it's context

disclaimer

The case material is based on actual data, from a complex context with large commercial interests. The material is ***simplified*** to increase the accessibility, while at the same time ***small changes*** have been made to remove commercial sensitivity. Commercial sensitivity is further reduced by using relatively ***old*** data (between 5 and 10 years in the past). Care has been taken that the illustrative value is maintained

ASML builds wafersteppers, lithography equipment used by semiconductor manufacturers. The lithography equipment determines to a high degree the performance and cost of the semiconductor manufacturing.

Figure 1: ASML Twinscan AT1100

Figure 1 shows one of the most recent ASML products, the Twinscan AT1100. This is an 193nm high NA scanner, capable of handling 300 mm wafers.

The main function of the waferstepper is to "print" the electronic circuit information on the wafer. The waferstepper is only exposing the wafer, the actual circuit is formed by many processing steps in the semiconductor fab. Many (typical hundreds) dies, identical electronic circuits, fit on one wafer. A few dies are

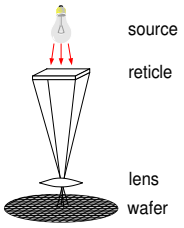

Figure 2: What is an waferstepper

exposed at a time. The original information for the exposure resides on the mask or reticle. This mask or reticle is 4 or 5 times larger than the final circuitry. Via an extremely high quality, but expensive lens subsystem the original is projected on the wafer, see figure 2.

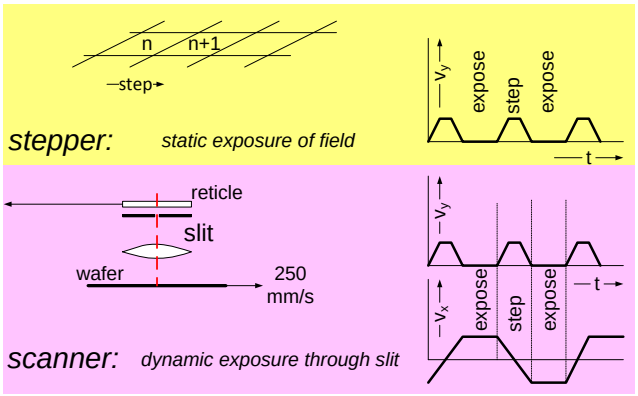

Figure 3: From stepping to scanning

Modern wafersteppers actually do the exposure scan-wise, where both reticle and wafer move and the light is passing through a narrow slit, see figure 3. Scanning is using the lens more effectively than static exposure of the entire area.

Lithography customers use a few key specifications for the lithography operation, see also figure 4:

- Critical Dimension (CD) control or imaging
- Overlay

The Critical Dimension (CD) control defines how accurate the linewidth of structures can be controlled. This parameter strongly influences the final performance (speed, power) of the electronic circuitry.

The overlay is defined as the repositioning accuracy of successive exposures. Electronic circuitry is build by exposing and processing layer by layer. Hence the

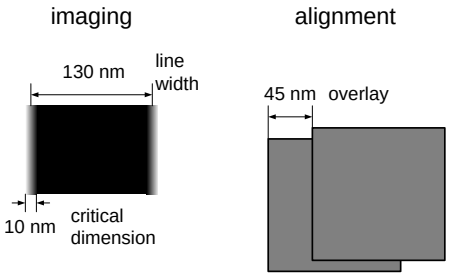

Figure 4: Key specifications waferstepper

same wafer is exposed many times, with days to weeks in between, where the next layer must be at (nearly) the same position. The overlay amongst others strongly influences the density of electronic components that can be obtained.

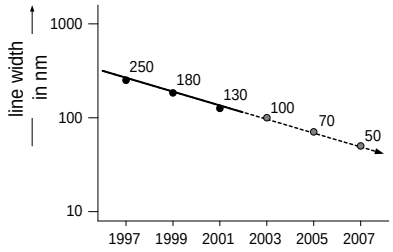

Figure 5: Moore's law

The entire semiconductor industry is driven by Moore's law, see the visualization in figure 5. Most competitors try to leapfrog each other by being faster than Moore's law, creating an extremely competitive environment, with large stakes.

In order to achieve the required performance figures technical budgets are used, see figure 6. Such a budget is a decomposition of the allowed performance figure into subsystem or component level contributions. Note that the addition of contributions is not always linear; systematic effects add linear, stochastic effect add quadratic.

These budgets are based on models of the system. Of course every model is a simplification of reality. Figure 7 shows the many components in the system that in one way or the other influence the overlay. It is immediately clear that the overlay budget takes only a limited set of influences into account, the "significant" ones.

When the performance requirements of the system increase (as dictated by Moore's law) more and more components start to fall into the *significant* category, causing an exponential increase of adjustment and control complexity.

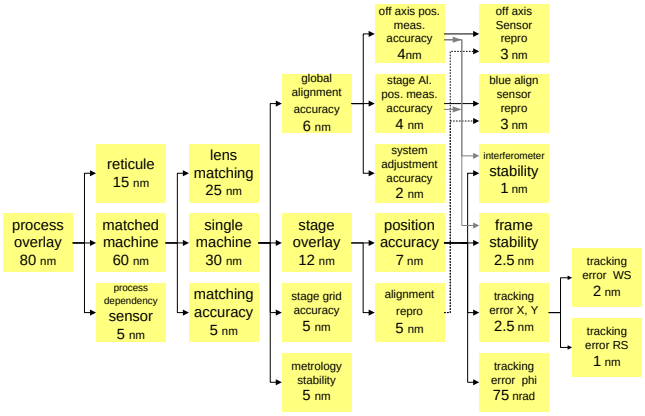

Figure 6: Overlay budget (1999)

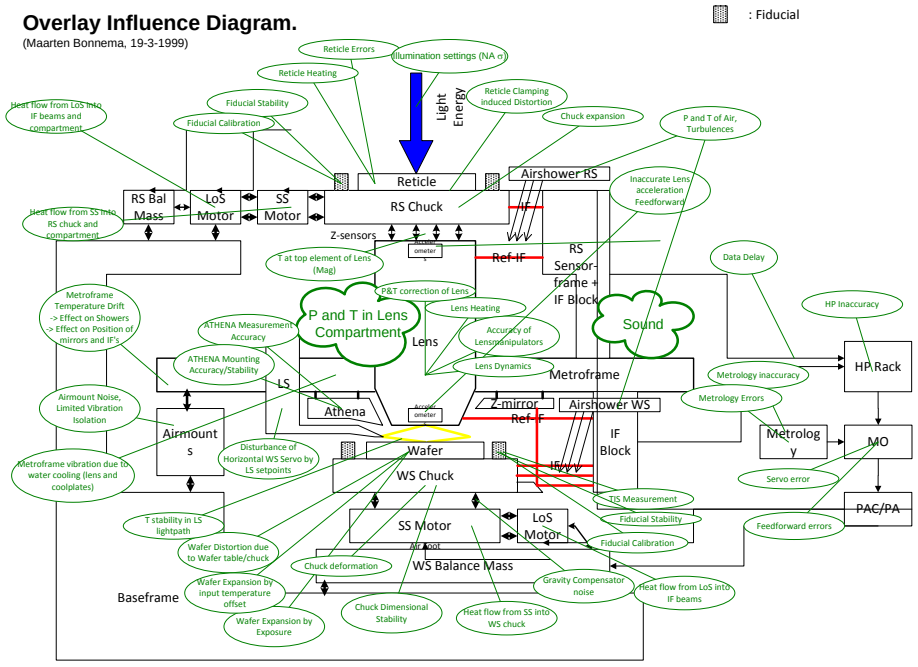

Figure 7: Everything influences overlay

Exercise 1, 10 minutes

Make a 3 picture description (What, How, biggest challenge) of your own system.

The key performance of the waferstepper, in terms of CD control, overlay and productivity and the design choices depend on many context aspects, such as the production environment, the business, the human stakeholders, and the many involved technical disciplines.

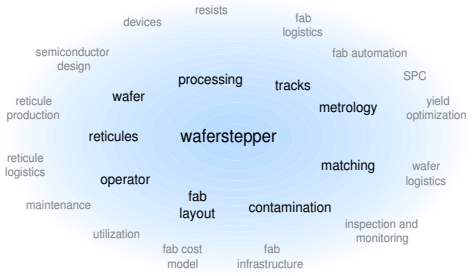

Figure 8: Fab Context of Waferstepper

The key performance in the production environment depends on the waferstepper itself, but also on many other aspects in the context of the waferstepper, as shown in Figure 8. For example, the wafer and the reticule themselves influence the performance as well as the measurement, processing and logistics of wafers and reticules.

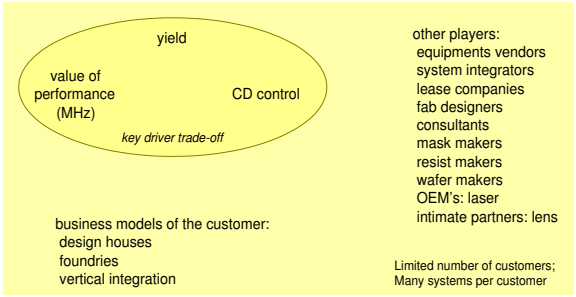

Figure 9: Business Context

In the business context, Figure 9 a balancing act is performed between yield and CD control with a significant impact on the final chip performance (power and speed). The business context is a complex playing field with many players, such as equipments vendors, system integrators, lease companies, fab designers, consultants, mask makers, resist makers, and wafer makers and many different kinds of customers: design houses, foundries, and vertical integrated companies.

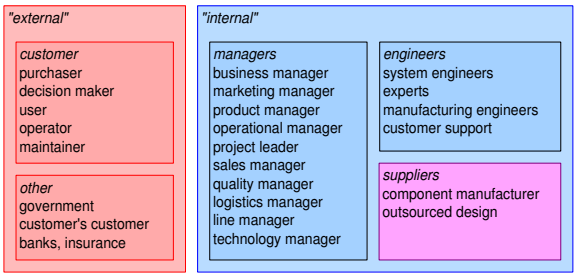

Figure 10: Human Context: Stakeholders

Figure 10 shows the stakeholders in the human context. The stakeholders are both internal as well as external. All stakeholders have their particular concerns, interests, rhythms, and contributions. The design emerges from a complex psychosocial interaction between all these stakeholders.

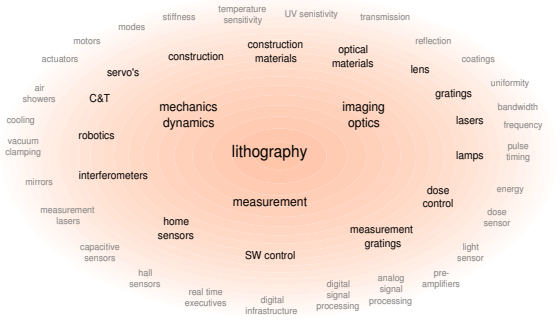

Figure 11: Multitude of Disciplines

The system is developed by a large number of different experts. Figure 11 shows many of the involved technologies, and therefor the involved technical disciplines.

Figure 12 summarizes all the mentioned different contexts of the waferstepper. The waferstepper is in itself a very complex system, operating in a heterogeneous and complex context.

Problems arising from the complexity of this context become visible in a rather late stage of development: during integration or worse in the customer's fab. The problems are often detected during the integration, when (often well tested) components have to work together, see Figure 13. The solution of these problems takes more time than planned, causing project delays. The dynamics, the uncertainties, the unknowns and the heterogeneity of these systems engineering aspects makes complex systems engineering into a challenge.

Exercise 2, 10 minutes

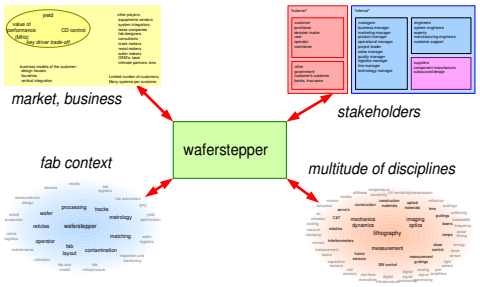

Figure 12: Complexity of Waferstepper Context

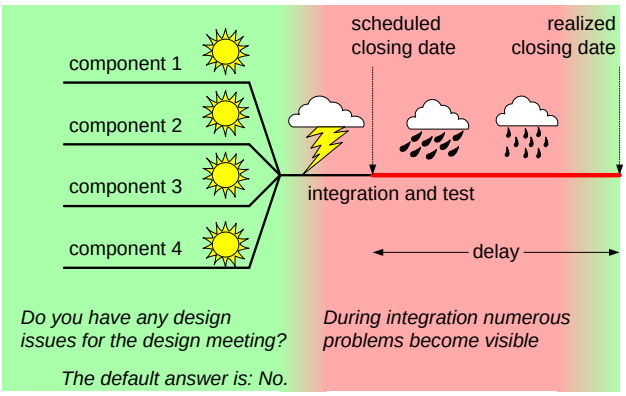

Figure 13: Symptom: Delays appear during Integration

Make a 3 picture description (Application context, Value chain, technologies) of your own system.

3 The Role of Software in General

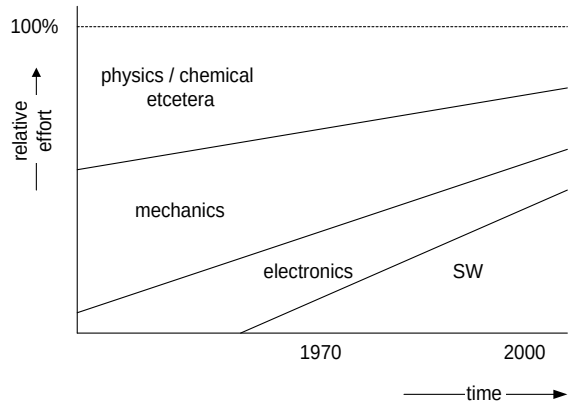

Figure 14: The relative contribution of software effort as function of time

The contribution of different technologies to the total system has changed dramatically during the last century. Figure 14 shows a schematic overview of the relative contribution of the different technologies to the total system.

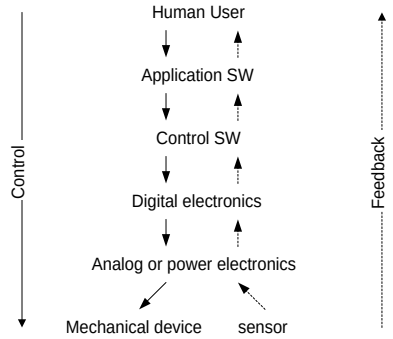

Figure 15: The Control Hierarchy of a system along the Technology dimension

The different technologies contribute in different ways to the total system. The differences between the technologies will provide insight in the specific role of software in an overall system. Figure 15 uses the control viewpoint to look at the different roles of the technologies. Mechanical and Physical devices act as actuators and sensors. The power for mechanical or physical devices is in one way or the other generated by power supplies or amplifiers. Those power supplies or amplifiers are based on analog electronics, sometimes with specific high frequency technology.

The input signal for the analog electronics is often generated in digital electronics.

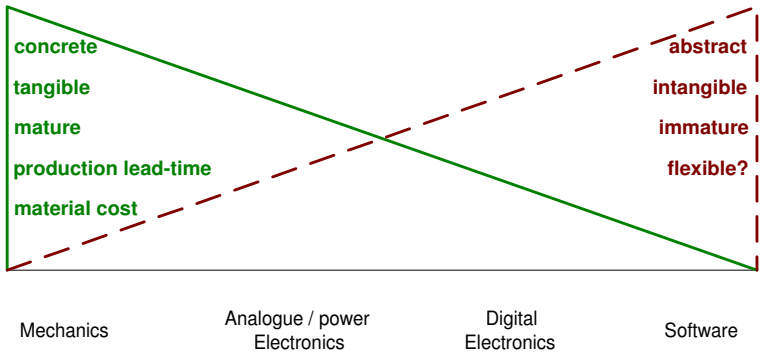

Figure 16: Characterization of disciplines, ordered along the level of abstraction

The digital electronics is then controlled by control software. The set-points for the control are generated by application software. Finally some human being is the operator of the machine and ultimately determines what the system must do.

From control point of view the technologies contribute in an asymmetric way to the system: a clear control hierarchy. Note that for performance or safety reasons many control shortcuts are present, ranging from mechanical interlocks to PID controllers implemented in low level software.

Figure 16 shows the same technologies ordered along the horizontal axis. The vertical axis shows a number of characteristics per technology that appear to be (inverse) proportional with the height in the control hierarchy: concreteness, tangibility, maturity, material cost, and production lead time. For example, mechanical constructs are very concrete and tangible, the engineering discipline is well-known, and the cost and the production lead time are directly related to the materials used and to the construction process. In contrast, software constructs are abstract and intangible, the software engineering discipline is in its early infancy, and the cost and lead time are practically zero.

The translation of system requirements to detailed mono-disciplinary design decisions spans many orders of magnitude. The few statements of performance, cost and size in the system requirements specification ultimately result in millions of details in the technical product description: million(s) of lines of code, connections, and parts. Figure 17 shows this dynamic range as a pyramid with the system at the top and the millions of technical details at the bottom.

In Figure 18 the pyramid is annotated with some examples from the waferstepper case. The key-drivers are clearly at a very high abstraction level. The highly simplified description of the waferstepper, based on Figure 2 is also at this very high abstraction level. The Millions of lines of source code are at the very detailed level.

Figure 19 shows another illustration of the level of abstraction in this pyramid.

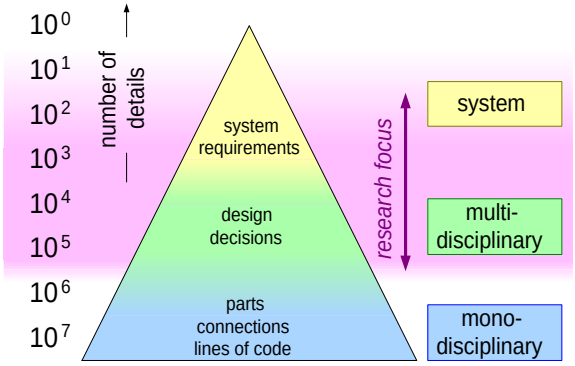

Figure 17: Exponential Pyramid, from requirement to bolts and nuts

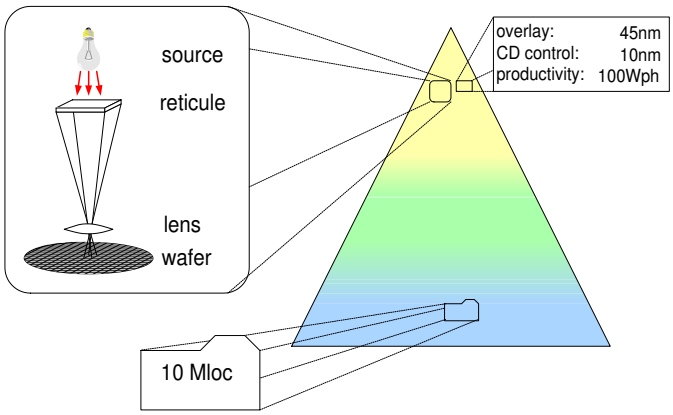

Figure 18: Waferstepper Example

At the lowest abstraction levels the components are shown. Many (atomic) components are mono-disciplinary. Components are aggregated into multi-disciplinary subsystems. These subsystems cooperate to perform system level functions. The cooperating system level functions result in certain system level qualities, such as overlay, critical dimension control, and productivity.

The software implements the functionality and realizes the system level qualities. The software is the integrating technology: it implements the behavior of the cooperating components, and in that way determines the actual system level qualities, see Figure 20

Exercise 3, 10 minutes

Make a toplevel decomposition of the software in your system and estimate the amount of software of the constituting parts

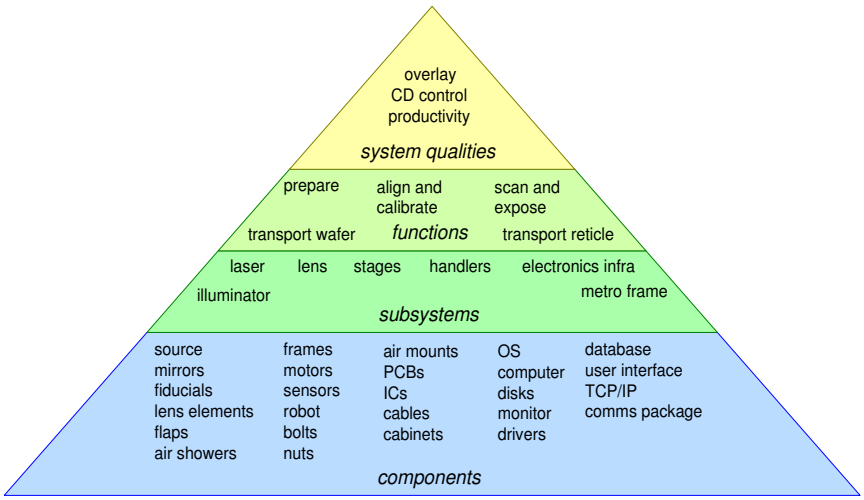

Figure 19: From Components to System Qualities

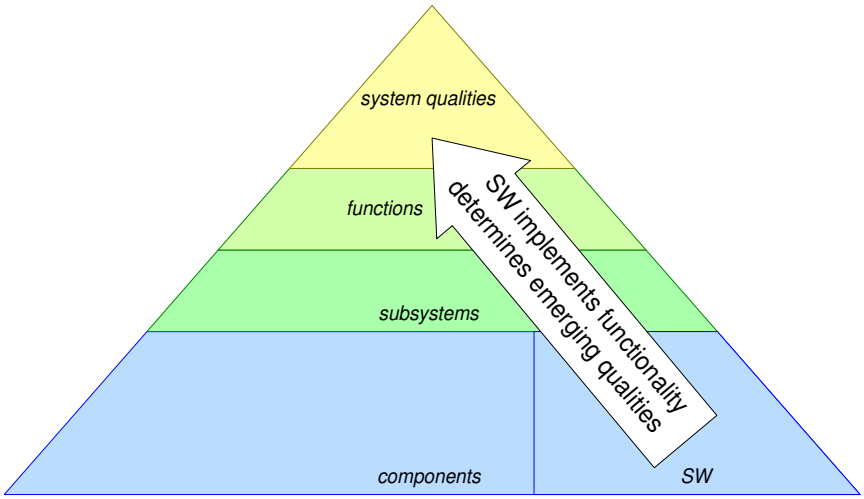

Figure 20: Role of Software

4 Software Requirements

When SW engineers demand "requirements", then they expect *frozen* inputs to be used for the design, implementation and validation of the software. So far, however, we have discussed *system* requirements. System requirements describe the **what** at system level. The system requirement specification can be a limited document, at least if the authors focus on the most important and relevant system functions and characteristics.

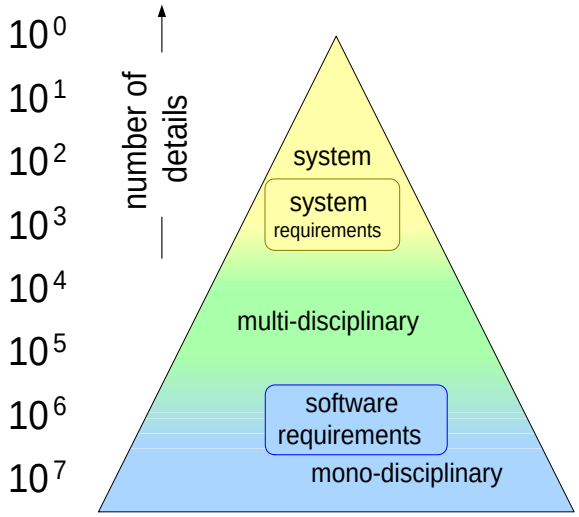

Figure 21: System versus Software Requirements

The translation of system requirements into detailed mono-disciplinary design decisions spans many orders of magnitude. The few statements of performance, cost and size in the system requirements specification ultimately result in millions of details in the technical product description: million(s) of lines of code, connections, and parts. The technical product description is the accumulation of *mono-disciplinary* formalizations. Figure 21 shows this dynamic range as a pyramid with the system at the top and the millions of technical details at the bottom.

The amount of details in a software requirements specification is several orders of magnitude more than the amount of details in the systems requirements specification. Figure 22 shows the software “subsystem” in its context. All the relations of the software subsystem with its context must be reflected in the software requirements specification. The software requirements specification is part of the detailing process of the system design and implementation.

The *user interface* and *system behavior* depends on many design choices. The software in most systems is the technology that implements both *user interface* and *system behavior*. Embedded systems interact with the physical world. The

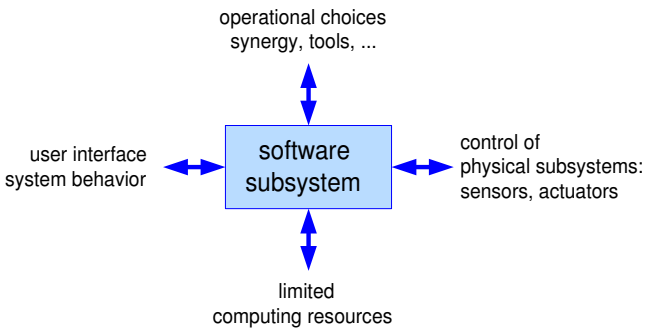

Figure 22: Why is the Software Requirement Specification so Large?

software implements the control of actuators and sensors that perform the interaction with this physical world. The related hardware-software interface (HSI) is a broad interface. The HSI determines many software design choices, and becomes part of the software requirements specification. Software needs a computing infrastructure to be executed upon. The computing infrastructure is always limited, putting constraints on the software. The combination of performance and cost requirements are translated into resource management requirements for the software subsystem. The software development department and environment result in operational requirements for the software subsystems. For instance in terms of tools and languages, and in terms of programming conventions and rules.

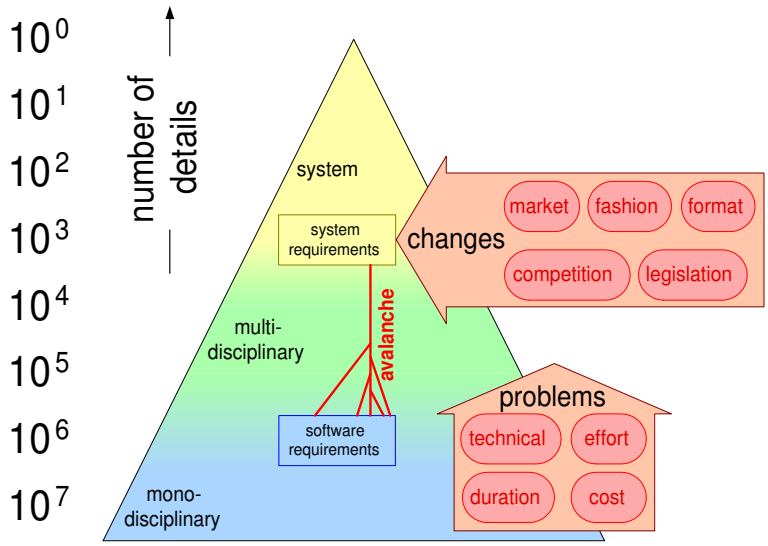

Figure 23: And why is the Software Requirement Specification never up-to-date?

The amount of details in the software requirements specification is huge. One of the consequences is that this specification is never complete nor up-to-date, see Figure 23. The environment of the software requirements specification is in practice highly dynamic. The outside world is changing in many ways (market, competition, legislation, fashion, and format). A small change in the outside world (top-down) may cause many changes in the software requirements. Design and implementation problems (technical, cost, effort, and duration) trigger bottom-up changes that may propagate into changes of the system requirements specification. Bottom-up changes can also trigger an avalanche of changes in the software requirements.

Exercise 4, 2 minutes

How many pages are in your Software Requirements Specification?

5 Evolution and Growth

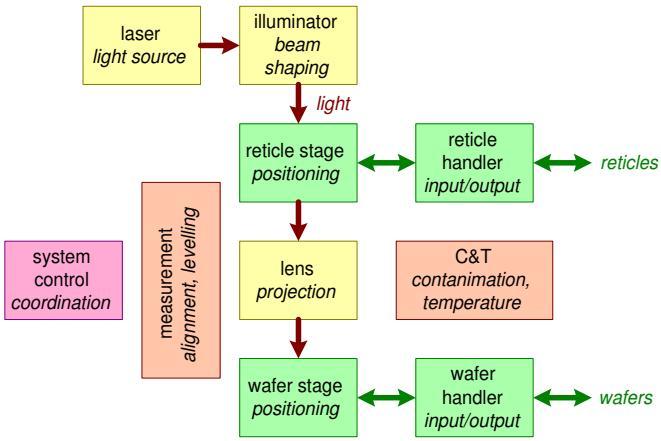

Figure 24: Block Diagram of a Waferstepper

In many domains systems evolve slowly from more or less purely mechanical systems to complex *software-intensive* systems. Many subsystems still have some dominant technology. Figure 24 shows the subsystem decomposition in the waferstepper case. Dominant technologies are optics in the laser, illuminator, and lens subsystems, mechanics in the reticle stage, reticle handler, wafer stage, and wafer handler, and more general physics in measurement and in contamination and temperature control.

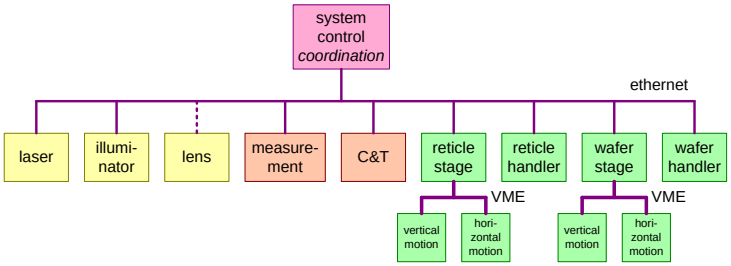

Figure 25: Control Hierarchy of a Waferstepper

The same subsystems can be shown in the electronics communication view, as in Figure 25. The dominant subsystems in Figure 24 have become simple nodes in the communication view.

The historic evolution in these systems is that subsystems start as independent subsystems, connected by a few straightforward synchronization signals. Over time dependencies are developing, when subsystems start to interfere, or when

system performance requires more complex types of interaction.

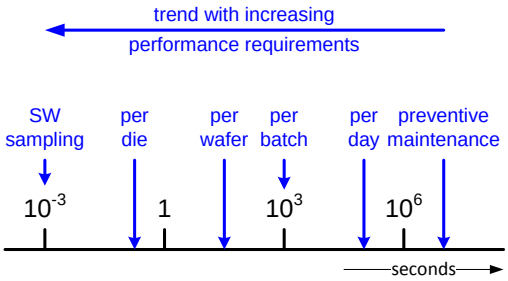

Figure 26: Frequency of Control Actions

One of the discriminating design parameters is the frequency of doing certain operations. For example, if some adjustment has to be done once in the system lifetime, than a factory or installation adjustment can be done manually. If this adjustment is complex or critical it might be supported by (SW) tools. However, when an adjustment is required for every wafer, then the adjustment will be automated for reasons of productivity and operational costs. Figure 26 shows an exponential time axis with different typical frequencies of control actions. In general the more frequent those control actions can be preformed the more stable and predictable the performance will be. However, high frequent control actions are often much more complicated than low frequent actions. Part of the complexity of high frequent actions is caused by the required cooperation between many subsystems (and technologies0 to implement the automated high frequent control action.

During the historic evolution from independent subsystems, with little or no automated adjustments, to continuously cooperating subsystems, full of high frequent automated adjustments, more and more software coupling is growing. Figure 27 shows the SW stacks early in the evolution and late in the evolution.

During this evolution many paradigm shifts are made. Unfortunately the engineered and managers are unaware of the paradigm shifts, and processes and design are not adopted accordingly. Figure 28 shows some of the consequences of such an evolution. Performance and functionality demands cause an increase of complexity; the increase of the complexity threatens the reliability of the system. The original system, with more or less independent subsystems required about 150k lines of code. Such a system can be well understood by a single system designer; the overview fits in a single human mind. The evolved system requires several millions lines of code. In practice that means that the overview is lost, or best case that the complete picture can be reconstructed with a limited set of senior designers. The evolved system suffers significantly from the much higher degree of coupling and the lack of one-to-one relationship between system function and subsystem.
Exercise 5, 10 minutes

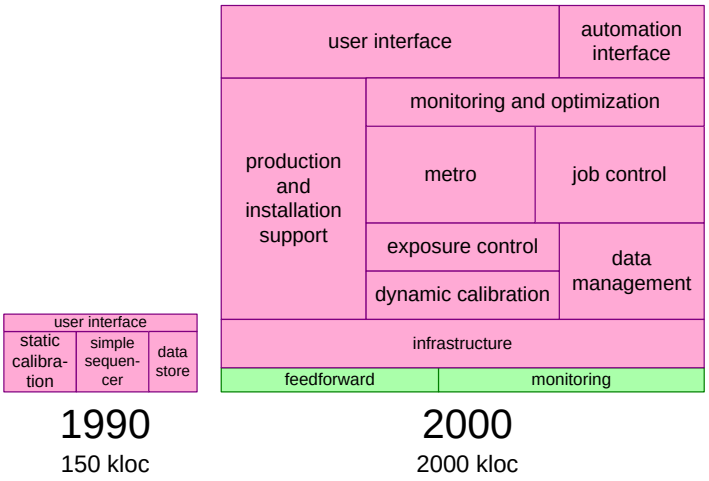

Figure 27: Evolution of System Control

Visualize the (SW) evolution of your system. What is your current phase?

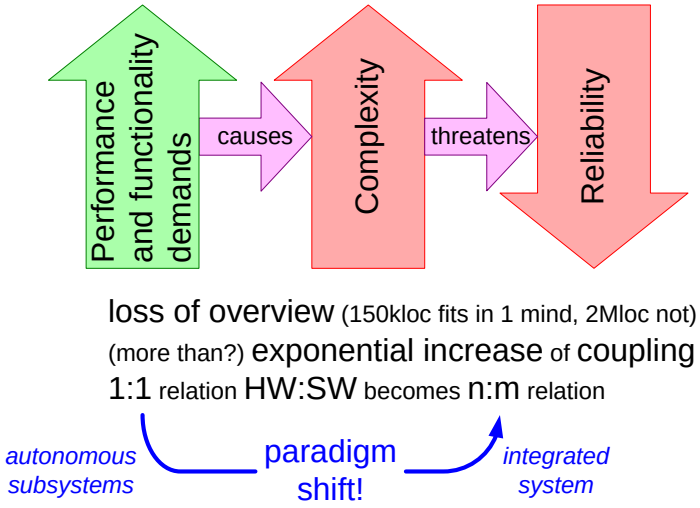

Figure 28: Consequences of Evolution

6 Why do we always have problems with software?

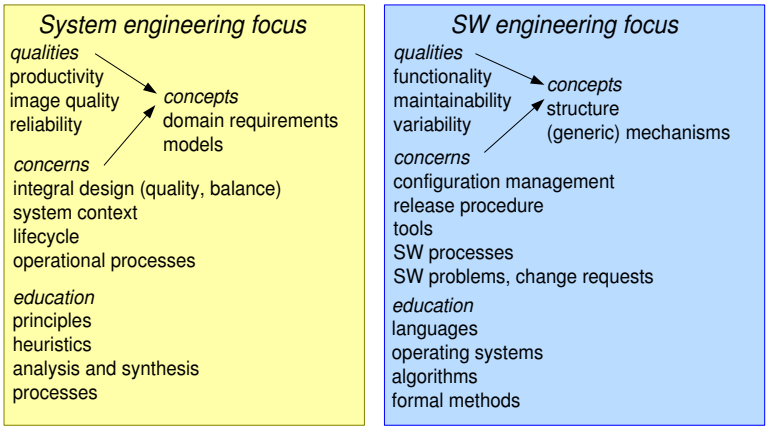

Figure 29: Different Focus of Software and System

In the previous sections we have seen that software is much more abstract and less tangible than the more conventional technologies. We have also seen that the software is the implementation technology of the system behavior, and functionality. The software implementation is the determining factor for the actual system performance. An additional source of problems in product creation is the different focus of the software engineering discipline and the system engineering discipline, see Figure 29.

Observation of SW engineering teams shows that the main concerns are: *configuration management*, *release procedure*, *tools*, *SW processes*, and *SW problem and change requests*. The qualities that get a lot of attention from the software crew are *functionality*, *maintainability*, and *variability*. As a consequence of these concerns and qualities the design focus is on concepts that help to structure and concepts of generic mechanisms.

The System engineering concerns are quite different: *integral design (quality, balance)*, *system context*, *life-cycle*, *operational processes*. The qualities that get a lot of attention by system engineers are *productivity*, *image quality*, and *reliability*. These system level concerns and qualities create a focus on concepts related to domain models and descriptive models of the system design.

The education of software engineers addresses many software technology issues, such as: *languages*, *operating systems*, *algorithms*, and *formal methods*. The system engineering education is less technological and addresses issues such as: *principles*, *heuristics*, *analysis and synthesis*, and *processes*.

The characterization above shows that system and software engineers have a completely different focus and education, while they have a intimate relationship: SW implements behavior, functionality and actual performance!

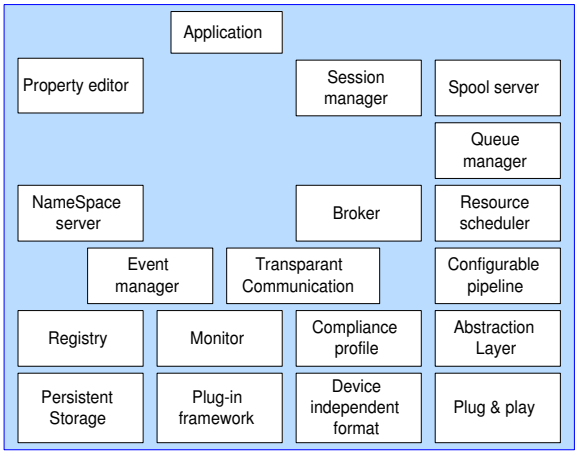

Figure 30: Caricature of a SW Architecture

The differences in focus and education also result in completely different views on the system design itself. Figure 30 shows a caricature of a software architecture. The diagram shows all kinds of generic software mechanisms and their layering. The only reference to the actual system functionality is in the box called application.

In contrast Figure 31 shows a caricatured of the system view of the Physics engineer. This view is dominated by the optical elements and characterizations.

From systems point of view the relation between the optical path and the required control, in software, is important. Figure 32 adds the control aspects to the physics viewpoint. The system engineer is responsible for the multi-disciplinary design. For example the trade-off to solve a image quality requirement in the physics components or to solve it by some software controlled adjustment procedure.

This simple example of the physics and software viewpoint shows the differences in language and system perception by the involved engineers. Many system level problems are caused by the limited understanding of the hardware and physics by the software engineers.

This problem is worsened by the limited awareness by most managers of both this limited understanding as well as the crucial integrating role of the software. The software developments and the system development are intimately related as shown above. Nevertheless the software departments are often quite isolated from the other engineering disciplines. Figure 33 shows a number of the symptoms of this isolation and the potential counter measures.

Exercise 6, 5 minutes

What is the degree of integration or isolation of SW in your organization?

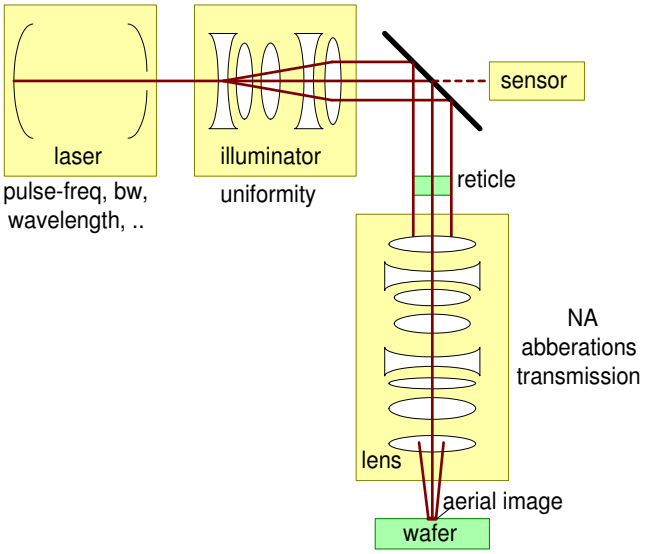

Figure 31: Caricature of Physics Systems View

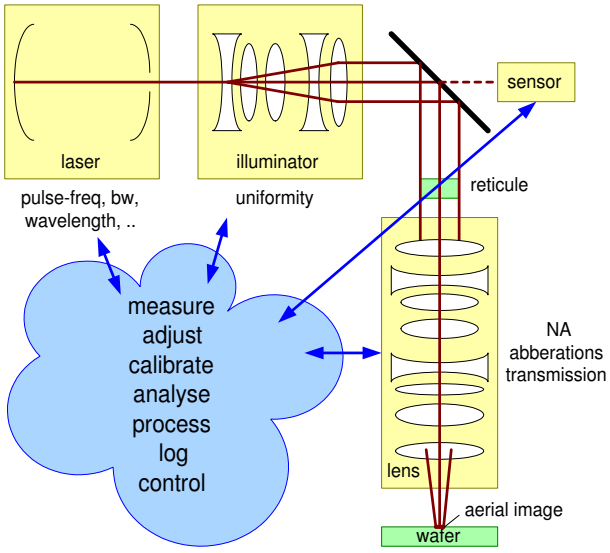

Figure 32: Relation SW and Physics

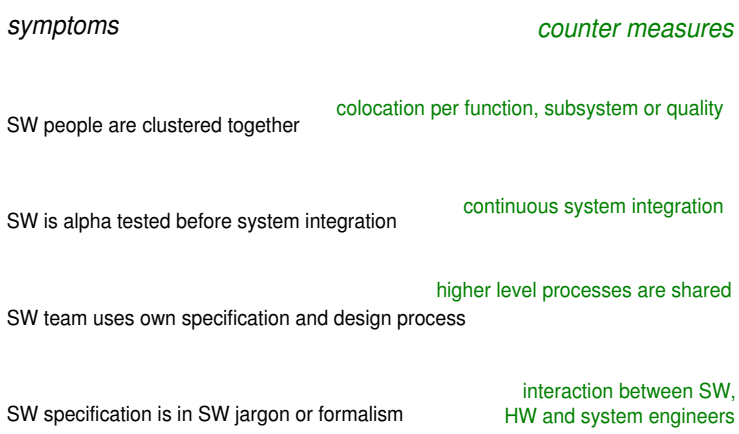

Figure 33: Symptoms of too isolated SW efforts

7 Conclusion

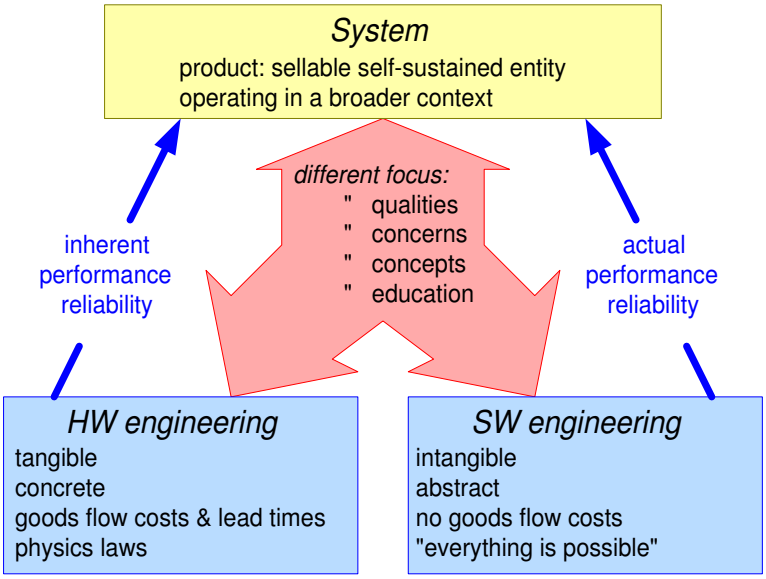

Figure 34: Different Mindsets and Characteristics

The creation of a product is focused on ceating a sellable system that fits well in the customer's context. The system is realized by many different disiciplines. The more conventional hardware oriented disciplines can be characterized by *tangibility*, *concreteness*, *goods flow costs and lead times*, and *physics laws*. The much younger software discipline, and to some degree the also young digital hardware discipline can be characterized by *intangibility*, *abstraction*, *no goods flow costs*, and *the "everything is possible" mindset*.

The hardware technologies determine the intrinsic performance limits, while the controlling software determines the actually realized performance. The system engineers, hardware engineers and software engineers use different languages, have different mindsets, have different concerns and address different qualities. These disciplines are highly complementary. The product creation process will be improved by more interaction and communication between these complementary engineers. Figure 34 shows these considerations in a single diagram.

References

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

History

Version: 0.1, date: 26 April, 2005 changed by: Gerrit Muller

- changed status to concept

Version: 0, date: 13 April, 2005 changed by: Gerrit Muller

- Created, no changelog yet