The Fab Experience

How I stopped whining and started to appreciate Process

Michael Stahl, Ron Moussafi

michael.stahl@intel.com , ron.moussafi@intel.com

Abstract

Many software teams struggle to implement and comply with quality processes. Adoption and strict adherence to process is seen as stifling, blocking innovation and redundant.

Contrast this with semiconductor fabrication plants (fabs) where adherence to process is the everyday norm and no fab engineer feels compliance with quality processes is optional.

Studying the top reasons why the fab world follows process so diligently reveals some underlying principles that can be applied to the software development world. Applying these ideas, software companies can improve the ability to implement and adhere to quality processes.

Biography

Michael Stahl is a 24-years veteran SW Validation Architect at Intel.

In this role, he defines testing strategies and work methodologies for test teams, and sometimes even gets to test something himself - which he enjoys most.

Before joining R&D, Michael worked for 10 years in Fab 8. In this paper, Michael draws upon his experience in the fab, in search of improving software quality.

Michael routinely conducts training sessions inside Intel, presented papers at a number of international conferences, and teaches a course in SW testing in the Hebrew University.

Ron Moussaffi is a 27-years veteran, currently managing a SW/FW Validation group at Intel. In this role, he leads a cross-site Validation team supporting on-chip embedded Firmware and Software products.

Prior to his current role, Ron worked for 23 years in Semiconductor factories, in a variety of Systems, Manufacturing and Technology leading roles. In this paper, Ron draws upon his experience in Quality management and improvement in Semiconductor Manufacturing, in search of improving software R&D quality.

1 Introduction

Software development processes are nothing new. “The Mythical Man Month” – a classic milestone in software development thinking - was published in 1975 and is still largely relevant today. There are 40 years’ worth of data, research and experience that shows how adoption of processes such as requirement management, reviews and unit testing help produce high quality software.

And yet, most teams ignore at least some of these learnings. Most projects don’t have proper requirements; most teams do not do thorough unit-test; peer reviews are done on best effort basis.

There are many reasons for this situation: developers who think that structured processes are an obstacle to get real work done; managers who don’t feel comfortable enforcing processes that their team objects to; plain old lack of knowledge about processes and the impact of non-compliance; distortion of methods to avoid parts of the process that seem like a drag and many more.

This paper does not delve deeply into these reasons. We believe they are mostly common knowledge.

Instead, we want to take a look at an engineering community who DOES follow a strict process to the letter and try to understand how and why it works for them. Once we know that, we can look at what can be applied to software development.

The engineering community we refer to are the engineers who work in silicon manufacturing facilities (also known as “fabs”).

Everyone who works in a fab conducts all activities in accordance to clearly written specs. The specs are always up-to-date. Nothing is changed without documentation, review and approval. Everyone – from technician level all the way to the Principal Engineers - follow the process in their daily work.

Why is this so? How is it that whole organizations, hundreds of people, follow strict process, something we, in software, can’t seem to get even a small tight team to do?

We believe that the fab culture holds some of the keys to adoption of proper software development processes. This paper first introduces relevant details about silicon manufacturing and some of the quality management principles used in fabs. It then identifies four areas where the principles and behaviors of the fab world differs significantly from what we have in software development teams. The paper then suggests how these principles can be applied to the software development world in an effort to improve process compliance, which in turn will improve the overall quality of the developed software.

2 Silicon Manufacturing Basics

Today’s integrated circuits are nothing short of a miracle. To give you an idea why we think so, let’s look at some numbers:

Intel’s most advanced CPU (Central Processing Unit) contains about 1.4 billion transistors. Each of these transistors is made of structures that are about 20nm in size (1 nm = 1 nanometer = 10^{-9} meter). If you lay 4,500 such transistors in a row, the total width will be approximately that of a single human hair.

Integrated circuits are manufactured on silicon wafers – a very thin and highly pure silicon base. Today’s newest fabs process 12” wafers – the size of a vinyl long-play record. About 500 CPUs can be manufactured on a single 12” wafer. During the manufacturing process, a wafer passes through more than 400 manufacturing steps. Each step is a stage in the process where the wafer is physically changed: a new layer is added, material is etched away or sputtered on, films are grown, impurities implanted into the wafer surface.

To get a commercially viable product, the yield at the end of the process must be in the high 90’s percentages (yield = percent of functional integrated circuits on a wafer at the end of the process).

As an exercise, assume that the yield at any step is 99.9%. That is, only 1 integrated circuit out of 1000 is damaged at each step. With 400 steps, the yield at the end of the line will be: $(0.9999)^{400}$ which is 67%! This is an intolerable low yield that will make the whole operation not profitable. The yield in each step needs to be closer to 99.99% (one damaged CPU in 10,000) in order to achieve a yield of 96%. This means that almost nothing should go wrong in the 400 steps - a very tall order.

Achieving such tight control over the manufacturing process is made possible by continuously being VERY careful about every aspect of the process.

In the fab, everything is monitored: temperature, humidity, air-born dust level, incoming material composition, gas flows, liquid concentration, electric power values, air pressure, etc. Each and every quantity is monitored by control charts, and the minute any monitored parameter starts to deviate from the normal, machines are put on hold and someone gets an alert.

3 Quality Management in Fabs

All activities in the fab are regulated by specifications and well defined processes. Every person in the fab who has anything to do with the manufacturing process must first read the specs and confirm by signature he/she understands the work procedures. When a change is needed to fix a problem, there is a specification how changes are implemented: what experiments must be done; what data must be presented before a change is allowed. As far as possible, nothing is left for chance.

Things do sometimes go wrong. A machine breaks down in an unexpected way; someone makes a mistake. In fab lingo, these situations are called “Excursions” – a severe deviation from the norm. In such cases, retrospectives and structured problem solving techniques are used to find root-cause for problems. Process fixes are designed that will eliminate the problem, preferably in a way that won’t allow it to happen again.

In the fabs, Quality is the factor that guarantees a product at the end of the line and therefore “Quality is #1 priority” is not a cliché; it’s a way of life. Without tight control over every aspect of the manufacturing process, the yield will be too low or the resulting products won’t last long enough in real life use. If you can’t maintain the highest possible quality level you may as well just close the fab.

To achieve the level of quality needed, fabs maintain a comprehensive quality management process. The quality processes govern the following aspects:

- Metrics & Controls (myriad parameters are measured and tracked: e.g. incoming materials quality, inline machine parameters, on-wafer electrical parameters).
- Excursion identification and containment (short time from the occurrence of an excursion to the identification that it occurred; quick lock-down of impacted materials and fab areas)
- Disposition (how to deal with impacted material)
- Root Cause analysis (why an excursion happened)
- Fix design and implementation (avoid the excursion in the future)

Each of these aspects is in itself a long list of activities and processes. There are also supporting processes related to workforce training programs; change control regulations; Manufacturing process methodology (e.g. Lean). Appendix A gives an idea how comprehensive these lists are.

Everyone in the fab sees quality as an integral part of their responsibilities. As a result, the QA department does not OWN quality – it just facilitates everyone else in maintaining quality in their operations.

The quality processes are also used to achieve continuous improvement of the manufacturing process. Engineers can propose process improvement. The ideas are tried out and tested in accordance with change control processes. If the results are positive, the change is accepted. If the results are inconclusive or negative, the change is rejected. This provides a balance between the wish not to change anything ("if it works, don't mess with it") to the benefits that may be gained by a well-designed change.

For someone not familiar with the fabs (such as software developers), the heavy focus on careful change management seems to indicate that the fab environment stifles innovation. The reality is that there is a LOT of innovation in the fab world. New Technologies and Products introduce constant challenge to the factory line. Engineers are continuously improving the manufacturing processes or find ways to reduce costs. Both authors, who worked at fabs during part of their career at Intel, were personally involved in highly innovative projects that streamlined the manufacturing process and saved millions of dollars. Yes, it was done carefully; but it was definitely innovative. We never felt stifled or held back due to the fab processes. Still, you may ask: what has this to do with software development?

4 Software Development Quality Processes

Maybe the reason that processes are less dominant in the software development domain, is that there are not that many well defined and proven processes?

The answer is a clear NO. There are many well defined processes. There is ample data to prove the benefit of implementing a variety of processes at different stages of the development process. (Blake et al., 1995), (Intel, 2010) are just two examples.

Processes like inspections, unit-testing, continuous integration, as well as development models like Agile, Pair-programming, waterfall, V-model, Spiral and CMMI (Capability Maturity Matrix Integration) are familiar terms to many developers. As an example of quality management framework for software development, we can take CMMI. CMMI defines five maturity levels:

1. Initial
2. Managed
3. Defined
4. Quantitatively managed
5. Optimizing

A set of software development processes are associated with each level. A CMMI level is achieved when the processes associated with this level are implemented and used by an organization.

While many software teams do not use CMMI as a framework for quality management, one can survey the actual processes used by a team and draw parallels to CMMI processes. Our personal experience tells us the result of such a survey will show that most teams operate somewhere between CMMI level 2 and 3. This experience is also supported by research (Carleton, Anita, 2009); (Elm, Joseph P. et al., 2007).

While seemingly different, there are many parallels between fab and software development processes (see appendix B). This similarity allows us to assess the fab's "CMMI maturity level" by comparing fab processes to CMMI-defined processes. Doing such comparison will reveal that fabs operate consistently at what would be the equivalent of CMMI Level 5.

How come so few software shops are at CMMI-5? Why do software developers find it SO HARD to do what fab engineers think of as "WAY OF LIFE"?

In the next section we will list a number of differences between the fab world and the software development world that we believe are the reason for the different attitudes towards Process.

5 Software Development vs. Fabs

The experience of the authors in both fab and software development, puts us in a position to suggest some major differences between the fab and the software development worlds. We believe these differences explain the different attitude towards process.

Additionally, we propose how adopting some of the fab culture and management practices may emphasis the value of development processes and thus increase the chances that teams will implement these processes.

5.1 Cost of Error

In the fab, a single mistake can cost millions of dollars. A human error in setting up a machine may cause all the wafers processed by that machine to be defective. Between the time when the error happened and the time it is noticed, a good number of wafer-lots may be processed by the problematic machine, each worth many thousands of dollars. If the error persists for a few days, it may cause enormous losses to the fab. Not only the cost of the raw materials, processing costs and workforce costs are involved, but also the cost of lost sales. The loss is immediate. Within hours or days of making the mistake, the full monetary impact of the mistake is clear.

Everyone in the fab understands the connection between mistakes and the associated costs. People want to be accurate and careful since the impact of not being careful is very tangible.

By comparison, in software development, the cost of error is unknown and hardly felt. What is the cost of a coding bug? What is the cost of a mistake entered at requirements gathering time? We are taught it’s 10x more expensive to find a bug in Test rather than at requirements stage, but how much is X? That is a very hard item to quantify. Another factor at play is that fixing a bug is many times just a few hours’ work. Not a big deal. Hardly any pain involved. Developers do not see the hidden costs of bugs: The added testing effort; the costs of releasing an emergency patch, There is no big chart showing how these costs accumulate. Indeed one bug is not terribly costly. But it is common to have thousands of bugs in a large software project. These costs add up!

Even when slipping the schedule has clear monetary impact such as delay penalties, many mistakes are made months or years before ship time. It is hard for a developer to link skipping a design review with the risk of late shipment and the associated monetary loss.

In short: Fab engineers have a very strong mental connection between making mistakes and losing money, while many developers do not. Trying to promote process adherence as a method to reduce the cost of errors makes a lot of sense for fab people. For software people, it does not resonate well since there is no clear linkage between bugs and high costs.

We believe that knowing the cost of bugs will cause software developers to be more conscious how their work impacts the bottom line. This in turn will provide a strong incentive to adopt processes that may help reduce this cost. When bug costs are known, they may be associated to the bug-injection point. This in turn will provide the ability to calculate projected ROI for processes that reduce errors from each development stage.

Once we know how to calculate bug costs, the next step is to make these costs visible at program level. For example, when a bug is found in test, a cost chart should show the impact. Visibility creates an incentive to reduce costs; no one wants to be the cause of a jump in the cost chart.

The problem is that it is difficult to create a clear measurement of cost-of-bug. Fixing a SW bug is relatively low effort, however the indirect cost can be substantial. Many hard-to-quantify costs are involved: the cost of reporting, triaging and reproducing the bug; the amount of validation effort the bug-fix triggers; the cost of creating and distributing a new release; the indirect cost of context-switch a developer have to do from their current work; the risk of lost sales. These are just a few of the parameters that add to the cost of a bug.

For programs with clearly associated delay costs, such as delay penalties, one approach may be to calculate the cost of an hour's delay in shipment and associate this cost to the time it takes to fix the bug.

Overall, we believe calculation of bug costs is an uncharted area that calls for further studies.

See (Holzmann, Gerard, 2012) for an example of creating a direct link between mistakes and costs (min. 17:45 in the video).

5.2 Accountability and Ownership

As part of the quality management principles of the fab, any misprocess event is investigated for root cause. Whether it was a human error, a spec that was not updated on time or an unexpected machine failure, the location of failure will be isolated. Each area has a clear owner and as the investigation proceeds, the responsible persons will be identified. The involved people know that a large monetary loss, delay in shipments or other negative impacts are associated with their area of responsibility. Not a comfortable position to be in, even if it does not mean punitive action (in most cases there are no serious personal ramifications, unless the problem was caused by gross negligence or violating specs). Since fab culture cultivate the notion of "everyone owns quality", the responsible people feel … responsible.

In the software world, we have many situations where ownership is non-existing or unclear. Documents are written as a one-time-shot and never updated to reflect the final product; architects design a product, move on to the next product before the current one is done, leaving any design problems to be solved by someone else. If there was a major architecture flaw, they may not even hear about it. Quality is owned by the QA. Developers don't feel they own quality – this is why there are testers. Bugs that are found in the field are Test responsibility. In some convoluted way, one may say that developers are almost expected to produce bugs in the code. There is hardly ever an effort to trace bugs to the responsible developers or to initiate root-cause analysis for bugs located in-house.

Ownership and accountability is a major characteristic of the fab work environment and people are proud to be owners of specific processing area. Ownership and accountability in software development is fluid at best, and there is little connection between cause and effect when failures occur.

Process helps to avoid mistakes or provide a proof you were compliant. Process is therefore seen by fab people as a support tool and the fact you can't do whatever you want is seen as a reasonable price to pay. Software engineers are not commonly held responsible for their mistakes, so have less inclination to take on the burden of adhering to specs.

We believe that software teams need to emulate the fabs in that "quality is everyone's job"; that people are expected to own up to their responsibilities. Quality as everyone's job is already part of agile development methodologies, so it seems we are in the right direction. We need to stop treating bugs as "integral part of the process" and start treating even internally found bugs as Excursions - something that should not have happened. Excursions call for root-cause analysis to find and eliminate the problem's source. One of the side effects of such analysis is finding the responsible area and the responsible individuals.

We propose to measure the Development team's efficiency not only by the extent of new features they create, but also by how much it costs to Test and Integrate them properly. Create quality metrics on per-module basis and make the defect-creating modules stand out. Suddenly, each team would have a professional-prestige stake in eliminating bugs. See (Holzmann, Gerard, 2012) for an example how per-module, public quality metrics help achieve better quality (min. 34:25 in the video).

Root cause analysis of bugs reported by important customers is something that does happen often, due to the visibility and impact of these bugs. However, we suggest to change the focus of such analysis from “why test missed the bug” to “why did the bug happen”.

Documentation is another area that will benefit from ownership. While agile process value working software over documentation, it’s also a fact of life that good documentation will help you achieve working software. Define the level of documentation you think is beneficial and hold the owners of these documents accountable for keeping them accurate and current. Un-update documents should be reported as bugs and fixing the documents should be tracked it as part of the project’s deliverables.

5.3 Culture

In the fab, quality and compliance are values. People are committed to quality and follow the process knowing that this is the only way to have a viable product. No one is allowed to be non-compliant, and there is zero-tolerance for those who violate the specs. Control systems are in place to ensure everyone is trained and updated on procedures and as much as possible lock-out engineers from doing their work (such as maintaining machines) if they did not read the latest spec update. Everyone in the fab follows the rules; there are no exceptions for unique people who are allowed to operate “their way”. With that, no one feels a sucker when following the spec. Everyone does it. It’s the norm.

In the software world, flexibility is a value. Processes are seen as stifling innovation and blocking progress and therefore bad for getting things done.

The reality that indeed one CAN produce a software product even when not following a process makes managers less confident when they try to impose a process on the team.

Many teams have one or two amazing developers that are so good the managers just let them do whatever they want as long as they crank out code. The fact that some people are exempt from following process has a very strong impact on the other team members, who get the message that following process is for lesser beings.

Even when processes are adopted, teams tend to take well defined and thought-through processes and do only the parts they like. The result are processes that sometime work and sometime not, “proving” the notion that “processes don’t work for us”.

The strong fab culture causes any new engineer entering the fab to adopt compliance and quality as a main theme in the work place. The opposite is true in software. New college graduates starting a new job see that most of the processes they learned about in university are not implemented or even mocked. The culture calls for flexibility, speed and trying new ideas. Process is seen as adding delay and unnecessary overhead to the overall cost. There is disbelief and ignorance about the positive impact of solid processes. The new engineers adopt this attitude and forget their education.

We believe that the most disruptive behaviors related to process adoption, is exempting some individuals from adhering to the process. If everyone follow the process, no one feels entitled NOT to follow the process; if some people are exempt, everyone treats the process as a “best effort” item. You may be encouraged by this: It’s hard to start mandating processes, but once the culture is changed to be one that embraces process, keeping it up calls for much less energy. It becomes just “this is how we do business here”.

Management should instill a Zero Tolerance to process violations. An example may illustrate this well: A few year ago, Michael led the deployment of Requirements Management tool in his organization. The pushback from the engineering community was strong and anyone who had some kind of half-justified argument allowed himself to avoid the system.

At a certain point, the manager of one team announced that he heard enough about the system's shortcomings and it was time to move on. He mandated that within two weeks each and every team member will load a requirement document to the tool, regardless of their reservations. That team was the only team who really embraced the tool and realized benefits from its use. Other teams kept procrastinating for quite some time; some of them never used the tool.

See (Holzmann, Gerard, 2012) for an example of imposing "everyone has to do it" attitude to agreed-upon processes (min. 23:40 in the video).As for stifling innovation: processes may actually free time for innovation. For example: when documents are all updated and all reside in an agreed upon location, time is saved for anyone who needs them. Skipping process frequently cause inefficiencies and therefore leaves less time for innovation.

5.4 Holistic view of the Process

In the fab, the mission is to run a smooth and efficient production facility. To achieve this, all parts of the line need to work in perfect synchronization. Workers at each step understand how the processing that they do impact the end product. Engineers are aware of the interactions between different stages in the manufacturing line. When something goes wrong, there is a very short feedback cycle to the location where the problem started, so the cause-effect linkage is strengthened. Engineers see the process as a whole and understand their role in this whole.

In a development environment, the mission is R&D. Many products have some technological aspects that are very new and not necessarily fully solved. In such an environment, factory-level efficiency is not expected and tight interaction between the moving pieces is not seen as a goal. It's the other way around: decoupling is a value and abstraction layers are built to hide details not necessary for integration of modules. As a result, some developers – especially in large projects - don't have a holistic view of the whole product. Everyone works on their piece of the code, trusting the defined APIs to make things integrate well. On top of it, some of the feedback cycles are very long. It may be six month before a developer learns a mistake he did in the code resulted in a costly delay or a hot-fix.

Seeing the product as the main mission; seeing it as a whole from start to finish, increases the notion of overall responsibility for outgoing quality and is a common sentiment in fabs. It makes every worker, at every step, willing to abide to the process, which helps reduce the risk of negatively impacting the next steps.

Seeing R&D as the main mission, puts the focus on "making it work". The focus is not on how efficient where you in finding the solution; nor how easy it will be to test or maintain. Many software development processes call for upfront investment to save back-end effort. When downstream efficiency is not a goal, adoption of processes suffer.

There is a view that software development cannot use production-type processes, since every product is unique. Therefore, flexibility is a must and mandating processes make the development team not flexible enough. We think otherwise: Creating software goes through rather generic steps: requirements, architecture, coding… in each project the CONTENT is different, but the PROCESS of creating a product is pretty much the same each time. You want to have reviews, unit test, CI, source control, change control. The process is the level you can standardize without losing the ability to produce a wide variety of software products.

The development process will be streamlined if we start thinking in term of "Software Production" and not just "software development" – simply because the generic supporting processes can be made to conform to production-like controls.

Once you have a stable Software Production framework, you have a stable process. Now you can start measuring and monitoring the impact of one step of the process on the next one, and initiate effective continuous improvement activities – just like it's done in the fabs.

Another aspect that can be adopted from the fab is the notion of overall owner. A senior process engineer is assigned at the chief technologist for the manufacturing process. This is the person who runs the change control board. This person’s role is to understand the full flow of the production process down to the details. This is the go-to person for questions about cross-steps interactions. This person helps design tests to check a proposed process change. A similar approach is software development will increase the cohesiveness of solutions and avoid the risk of different teams going in different directions.

Another approach, from Toyota, is to keep the same people on a project from start to end:

> "Toyota organizes teams around complete projects from start to finish. For example, the design phase of the interior of the vehicle is the responsibility of one team from the design phase through production. Having the responsibility of participating in the project from beginning to end enriches and empowers the employee." (Liker, Jeffrey K. 2004, p. 196)

It is our view that it not only empowers the employee. It also creates an incentive to invest upfront in doing things properly, since the benefit of this investment will be ripped by the same people. This notion works well for Agile teams where the boundaries between developers and testers are not rigid and developers have a concrete stake in bug reduction since the entire team will suffer from last minute critical bugs.

5.5 Engineers Self Image

In the fab, engineers see themselves as scientists. The image impacts the work style: fab engineers are careful, meticulous, methodical, make decisions backed by data. They can’t afford to be reckless: a single experiment may take 10 weeks to run through the manufacturing line and cost many thousands of dollars. Things must be planned, analyzed and thought through before implementation.

Developers, on the other hand, see themselves as creative beings with unlimited potential to experiment. They are willing to try ideas without totally analyzing them first. Implemented in code, any mistake can be fixed quickly: the next version of the code is just a button click away.

In the fab, experiments take a long time by definition (it takes weeks to months to run a single experiment). The extra time added for following process does not change this timeline in a significant way. It also helps avoid silly mistakes which may invalidate their results. For a developer, just coding something seems much quicker than having it first documented and reviewed. The fact that taking the seemingly longer documentation and review path may eventually make a better solution that works the first time is hard to believe when you KNOW you can hack the code in two days. Worst, if it does not work – two more days to fix it.

This difference in attitude is not something we propose to change, but is an important parameter to remember when attempting to increase the adoption of processes in a software development environment.

6 Summary

We described and discussed some fundamental differences between the fab and software development.

While these two worlds are very different, we believe there are aspects of the fab environment that can be adopted by the software development community. If adopted, in full or partially, these aspects will help drive process acceptance and process adherence in the development world, resulting in higher quality of software products, achieved with less thrush and less rework.

References

Blake Ireland, Ed Wojtaszek, Dan Nash, Ray Dion, Tom Haley, 1995, *Raytheon Electronic Systems Experience in Software Process Improvement*

Carleton, Anita, 2009, *CMMI® Impact,* slide 8.
http://resources.sei.cmu.edu/asset_files/presentation/2009_017_001_22751.pdf

Elm, Joseph P. et al., 2007, *A Survey of Systems Engineering Effectiveness – Initial results*, p. 33
http://resources.sei.cmu.edu/asset_files/specialreport/2007_003_001_14801.pdf

Holzmann, Gerard, 2012, *Mars Code*, https://www.usenix.org/conference/hotdep12/workshop-program/presentation/Holzmann . Holzman tells about the quality processes implemented in his team for the Mars Rover. Some notable points are the selection of process rules based on known failures, creating a tight connection between bugs and cost. Also, everyone had to comply with the agreed-upon processes and problematic modules were made publicly visible.

Intel, *Report on process improvement project*, 2010. Implementing requirments management, unit tests, continuous integration, design and code review regime, static analysis, exit criteria for each development step. Some of the results: Only one customer-reported defect (the product used in 100 designs, by more than 70 customers); defects/KLOC: 5x below target; estimated $1.6 million saved by reviews; porting to the new version by customers done in less than 1 week; no need for post released sustaining effort (no hot-fixes needed).

Liker, Jeffrey K, 2004, *The Toyota Way,* McGraw-Hill.

Appendix A

A very partial list of quality processes implemented in fabs

- Incoming material inspections: Any material entering the manufacturing line is tested for purity and cleanliness
- Correlation tests: Process reference wafers through a machine and compare the results to the expected ones. Continue to use the machine only if it passes the correlation test
- Track trends of hundreds of parameters. Use Statistical Process Control (SPC) methods to identify drifts.
- Setting the Control Limits for the SPC is in itself governed by strict rules and procedures.
- Track Quality results indicators such as yield; number of re-processed wafers; machine Mean-time-between-failure
- Training and certification tracking for each individual
- Change Management Process: a list of procedures define how a change is proposed, reviewed, tested and implemented
- Recipe control: a recipe defines the setting of production machines. A host of interlocks ensure the correct recipe is loaded to a machine to process a certain wafer-lot
- Clean Room behavior rules: Dress procedures; allowed materials; allowed behavior in the cleanroom
- Safety procedures: What to do in case of emergency. The fab considers even a simple water spill as a safety event that needs to be addressed according to predefined procedures
- Material disposition rules: How to quarantine and how to release from quarantine any wafers that did not go through the standard process (due to mistake, machine problem etc.).

Appendix B

Drawing parallels between software development processes and fab processes

SW Development processes	**Fab processes**
Requirements: Define the inputs and outputs of the system.	Process targets: Define the exact values of input parameters (e.g as gas pressure, chemicals content) and the output targets (e.g. layer thickness; resistivity) for each process step.
Architecture specs: Define the various pieces of the software that together achieve a Business goal	Process flow: Define the order in which material moves from one processing step to another, to achieve a complete product
Design documents: Define the mechanics of each module that will perform a required data processing	Machine recipe: Define how exactly the machine is set up for processing of wafers
Reviews, Change Requests, Design Change Notification (DCN)	Change Control Board: Review, critic, improve and approve suggested process changes
Unit tests, Static analysis	In-line tests & monitors: Tests done inline on in-process wafers and on machines, before the final product is completed.
Configuration management: A system that ensures developers use the correct version of files.	Recipe control: A system that ensures machines are using the correct settings for processing wafers
Build automation & Continuous Integration (CI): A system that ensures fast testing of new code, to ensure smooth flow when new code is added	Factory automation: A set of automation control programs that ensure material is moving into the right machines at the right time, with minimal time lost in waiting for free machines.
Test and Test documents	End of Line Tests: Electrical tests to validate the end product is performing its intended function
Risk Based Analysis	Test sample plan: Definition of the frequency in which in-line tests are done. Testing too often holds the line and is costly; not enough testing risks losing a lot of material when there is an excursion
Root-cause analysis	Root-cause analysis