Software development for automotive embedded non volatile memories testing

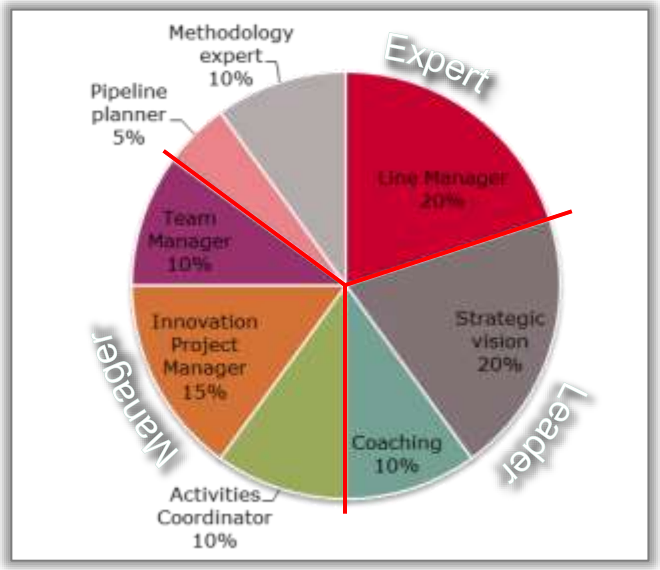
Ing. A. De Poli , Ing. M. Coppetta

Ferrara, December 2017

Self introduction – Angelo De Poli

- **Angelo De Poli** – Rovigo, 1977
  - Master degree in **Electronic Engineering** at **Ferrara University** (2003)
    - Internship in Ericsson AG, Aachen (Germany) with specialization in telecommunication
  - **Master in Business Administration**, **Bologna University** (2009)

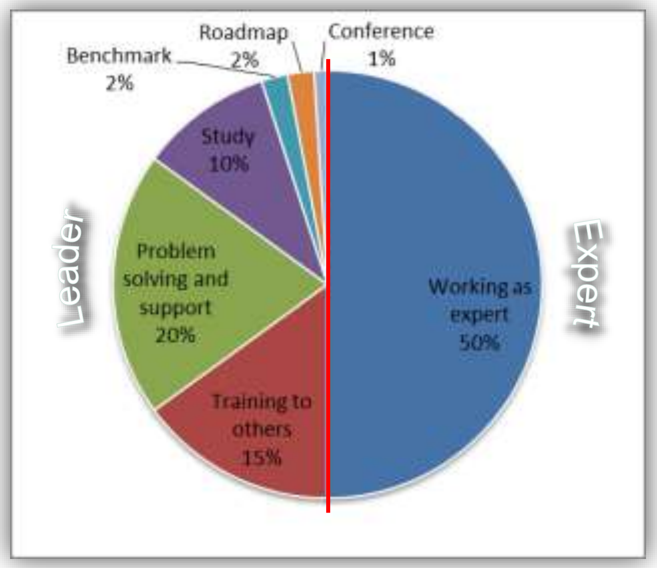
  - 2003: Telecommunication **Researcher** at **Ferrara University**
  - 2004: Application **Product Engineer** at **Brahma**, Legnago
  - 2005: TAV **System Engineer** at **Alstom**, Bologna
  - 2005: **Infineon Technologies Italia**, into Microcontroller team:
    - 2005: eFlash **Product Engineer**
    - 2006: **Methodology Manager** for activities support
    - 2010: eFlash Test Engineering **Team Manager**



Self introduction - Matteo Coppetta

- **Matteo Coppetta** – Rome, 1981
  - Master Degree in **Microelectronics** and **Electronic Engineering** at **Università degli Studi di Roma Sapienza** in 2006.

- Working **@Infineon Technologies Italia** since Jan 2006, into Embedded Flash Microcontroller team.
  - In 2006: Master thesis "Flash Fault modelling"
  - Since Sept 2006 **Product** and **Test Engineer** on **32bits** and **8bits** microcontrollers on eFlash customer validation, test flows and test software
  - Since June 2013 **Technical Leader** responsible for eFlash test flow and test software (**TestWare**)
  - Since Jan 2015 member of **PTE eFLASH Champions Team**
  - Since Jan 2016 **Senior Staff Engineer**
  - In 12/2013 and 12/2014 seminars *@University of Rome "La Sapienza"* about **eFlash testing** with sponsorship of *IEEE Electron Device Society*
  - In 12/2013 seminar *@University of Ferrara* about **Software development and test**
  - In 01/2017 seminar *@Politecnico of Torino* about **eFlash testing**



Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

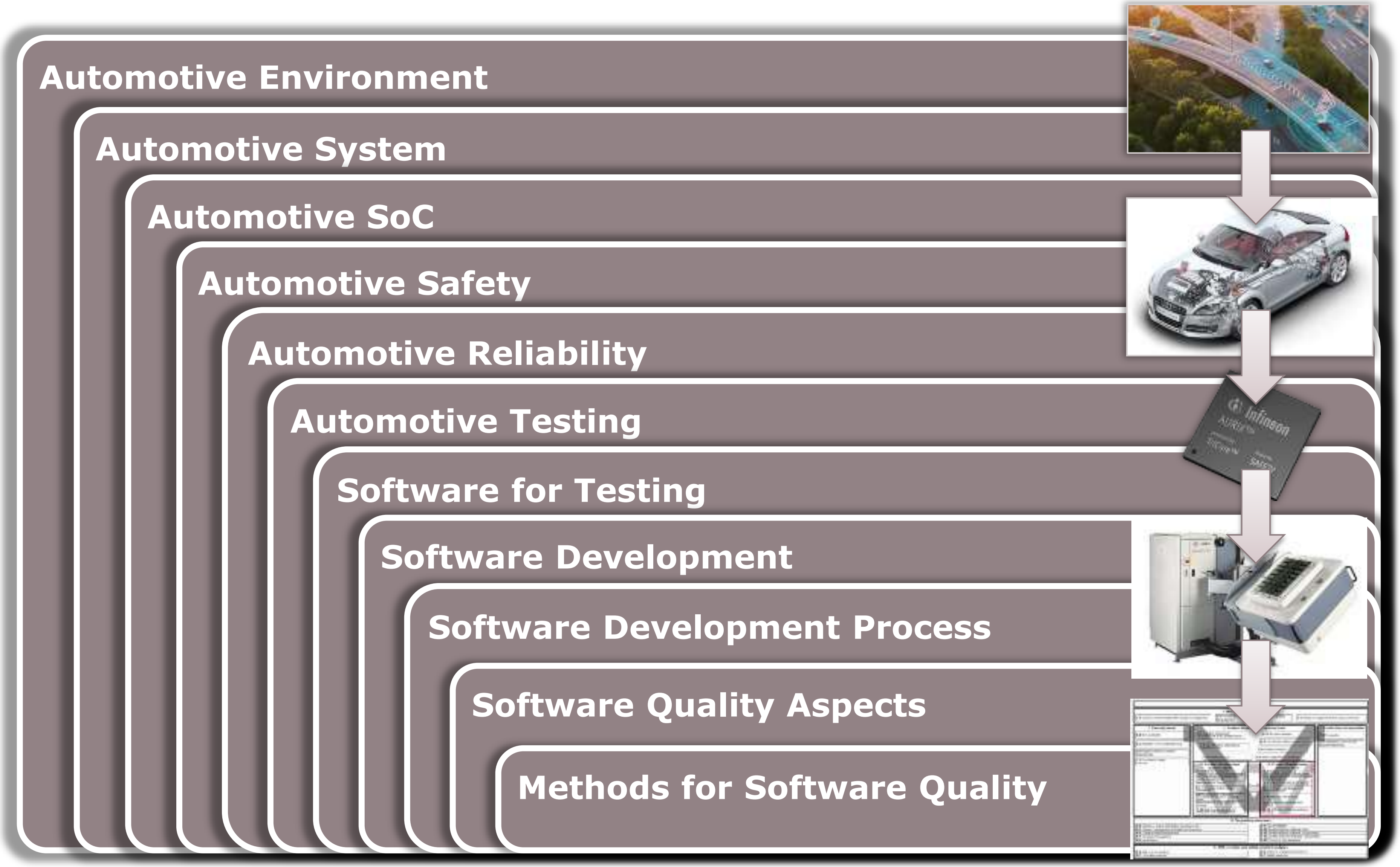
4 Software for Testing Development: why a process

5 Software for Testing: Strategies for Quality assurance

6 Software for Testing Development: phases

7 Conclusions and Questions

Presentation Roadmap



- Automotive Environment
- Automotive System
- Automotive SoC
- Automotive Safety
- Automotive Reliability
- Automotive Testing
- Software for Testing
- Software Development
- Software Development Process
- Software Quality Aspects
- Methods for Software Quality

Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

4 Software for Testing Development: why a process

5 Software for Testing: Strategies for Quality assurance

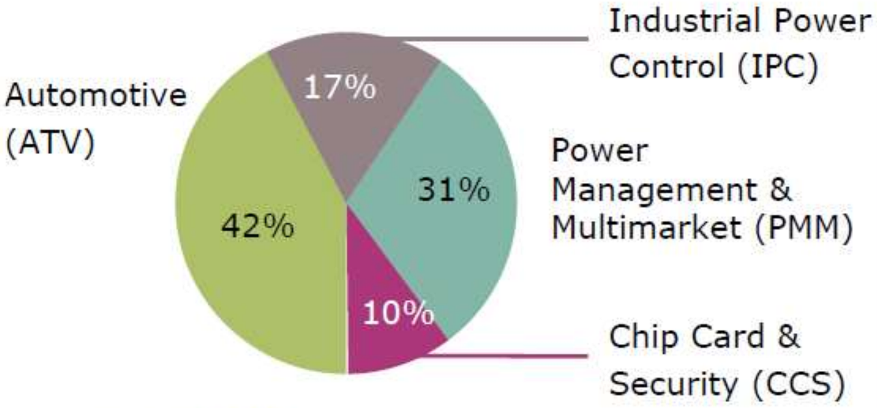
6 Software for Testing Development: phases

7 Conclusions and Questions

* Spin off from **Siemens** in 1999

Infineon* at a glance

Business Segments



Revenue FY 2017

Employees

Around **37,500** employees worldwide (as of Sept. 2017)

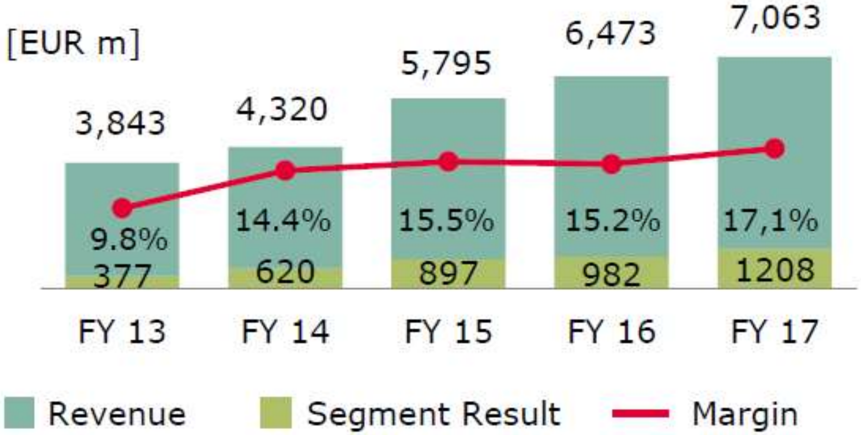
Europe
15,650 employees

Americas
3,850 employees

Asia/Pacific
18,000 employees

36 R&D locations
18 manufacturing locations

Financials



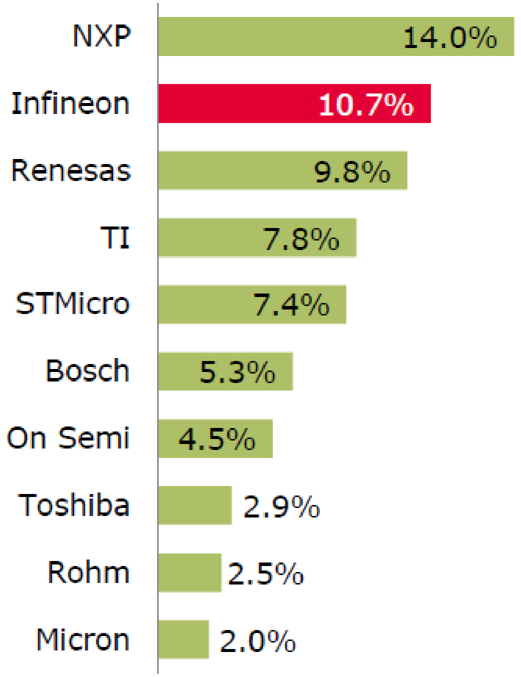
Market Position

Automotive	Power	Smart card ICs
# 2	**# 1**	**# 1**
Strategy Analytics, April 2017	IHS Markit, Technology Group, August 2017	IHS Markit, Technology Group, July 2017

Top positions in all major product categories

Automotive semiconductors

total market in CY 2016:
$30.2bn

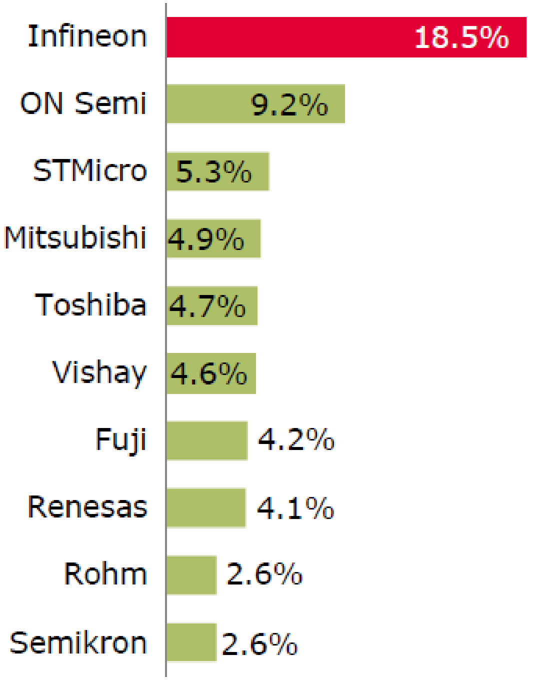


Automotive semiconductors incl. semiconductor sensors

Source: Strategy Analytics, "2016 Automotive Semiconductor Vendor Share", April 2017

Power semiconductors

total market in CY 2016:
$15.9bn

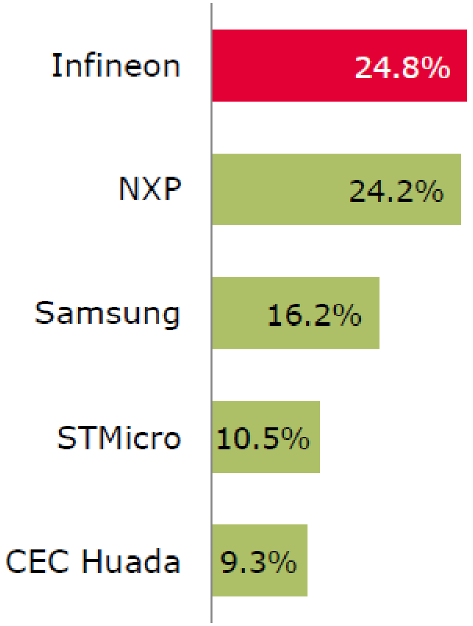


Discrete power semiconductors and power modules

Source: IHS Markit, Technology Group, "Power Semiconductor Annual Market Share Report", August 2017

Smart card ICs

total market in CY 2016:
$2.79bn



Microcontroller-based smart card ICs

Source: IHS Markit, Technology Group, "Smart Cards Semiconductors Report", July 2017

Infineon
Development Centre Padova

- **founded in 2001** by 12 design engineers
- located close to Padova city centre and University
- more than **150 employees**, mostly electronic engineers
- in these 16 years, active on development of
  - automotive **power electronics**
  - automotive **microcontrollers** (eFlash, PRE)
  - **industrial drives**
  - **supply** systems for CPU in desktop and notebook
- more than **199 patents** proposals filed

Automotive Microcontrollers
Padua Team Focus

Product Engineering Embedded Software

› **What?**
  - Developing embedded software for uC embedded flash validation and testing (production).

Product Engineering Lab Analysis

› **What?**
  - Laboratory analysis to validate uC embedded flash

26 Engineers

› **What?**
  - Cross teams support to improve productivity, efficiency and automation.

Product Engineering Methodology

Software development for automotive embedded non volatile memories testing
Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

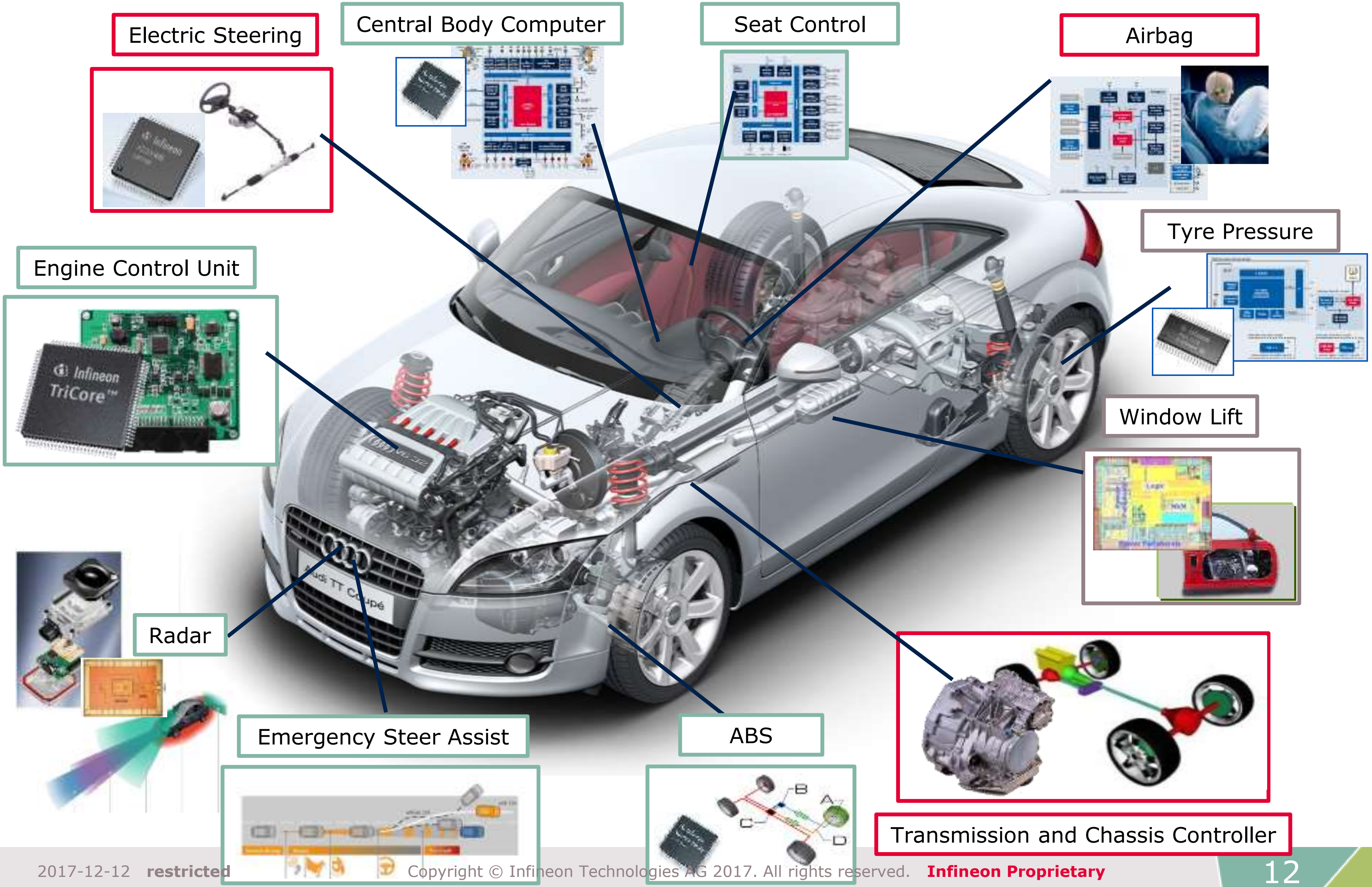
4 Software for Testing Development: why a process

5 Software for Testing: Strategies for Quality assurance

6 Software for Testing Development: phases

7 Conclusions and Questions

Automotive Microcontrollers Application Overview


Electric Steering

Central Body Computer

Seat Control

Airbag

Tyre Pressure

Engine Control Unit

Window Lift

Radar

Emergency Steer Assist

ABS

Transmission and Chassis Controller

Automotive Application Overview
Yesterday and today

Electric power steering

Airbag

ABS

Motor control

Window lift

Radar

Traffic sign detection

Automatic parking

Lane assist

Crash avoidance

Remote parking

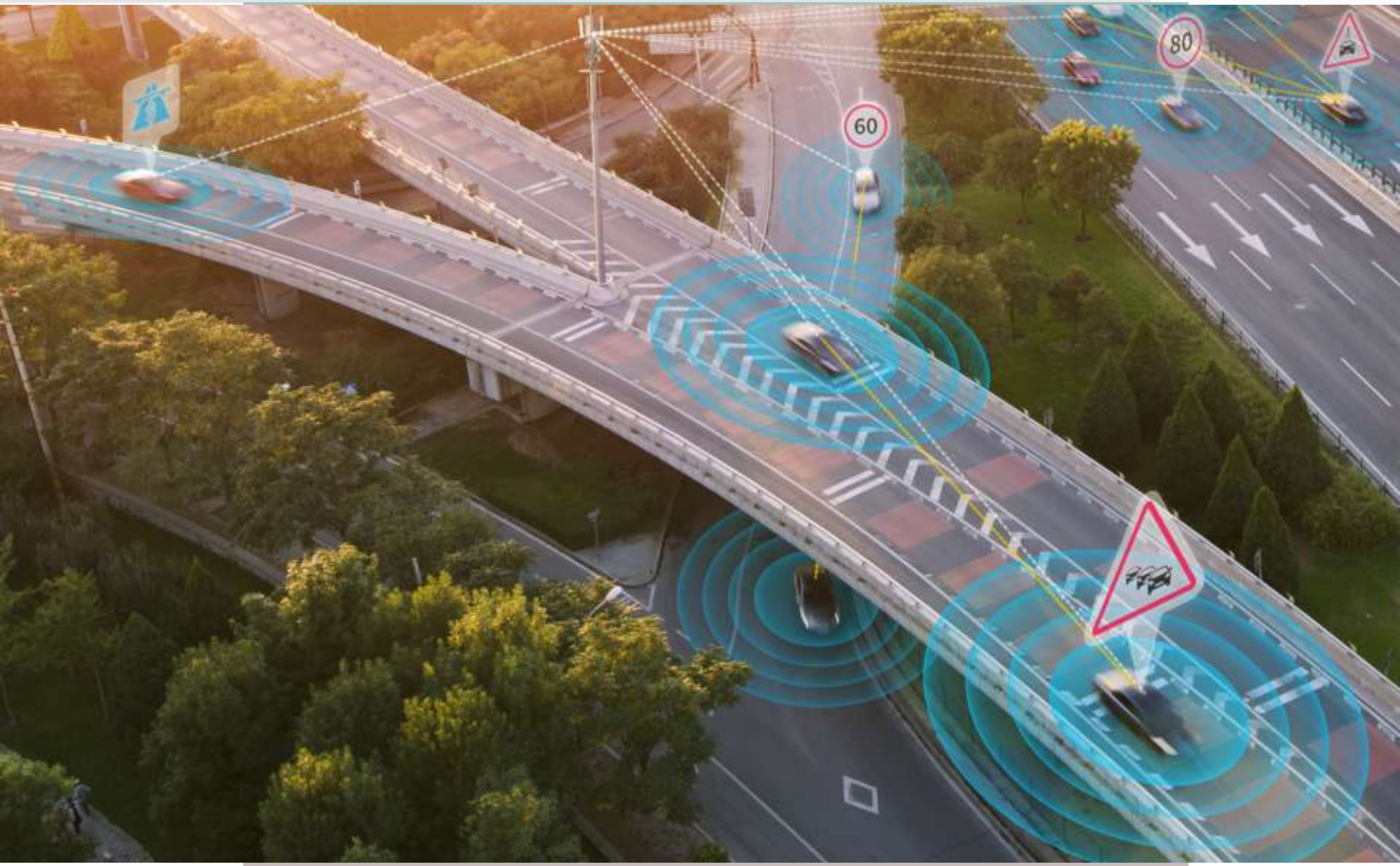
Adaptive head lights

Autonomous driving

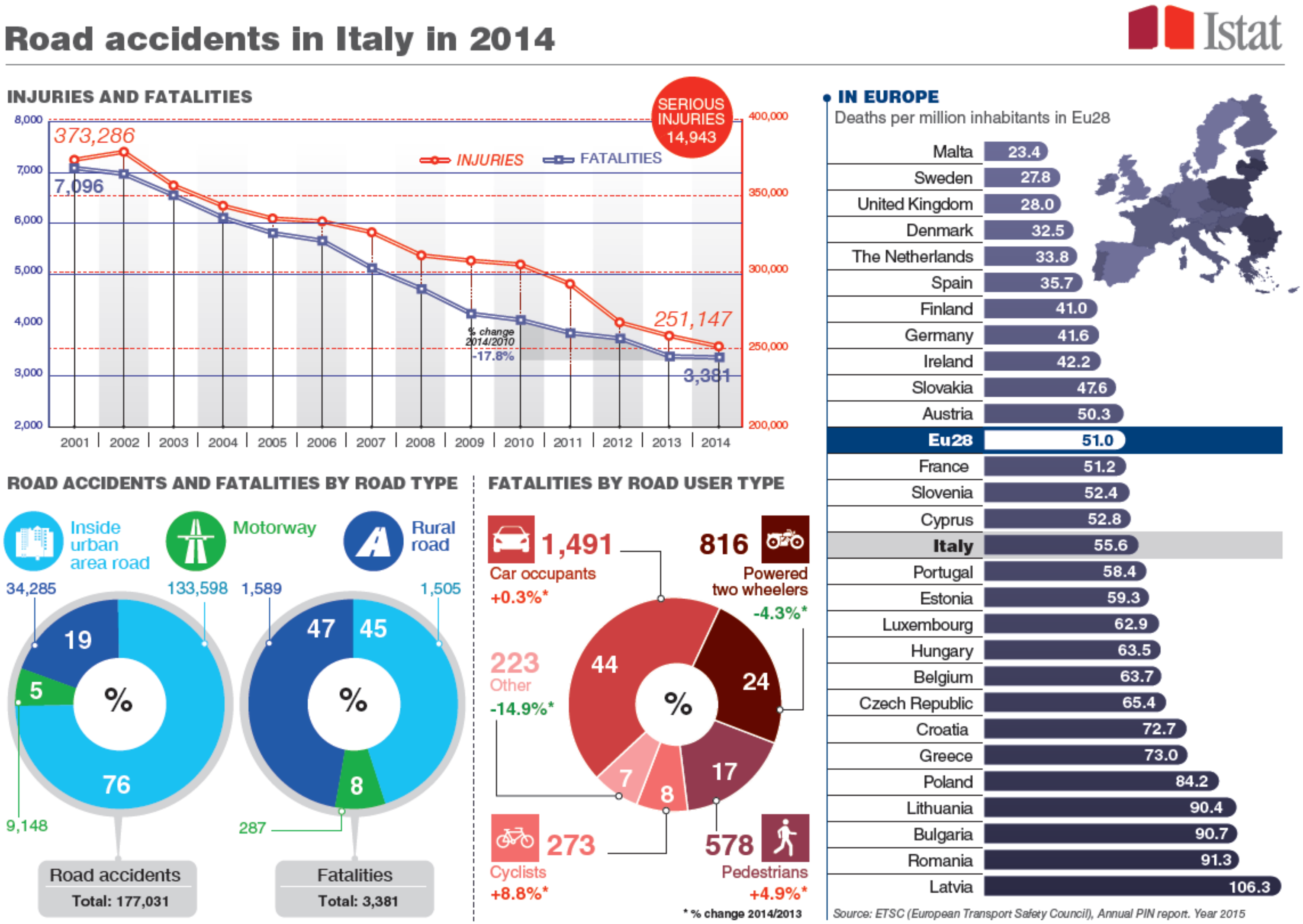
Distance warning

Rear cross traffic allert

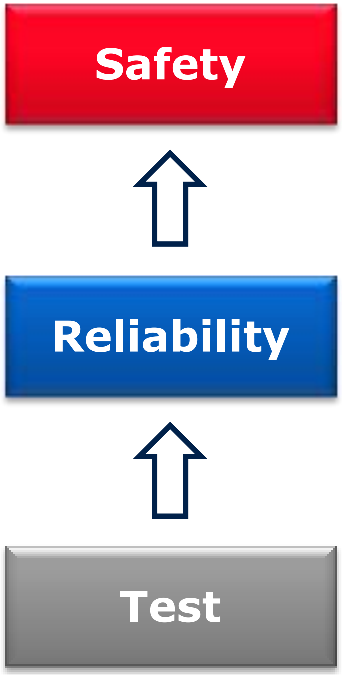
Automotive Application Overview Tomorrow



Why testing SoC



Source: ETSC (European Transport Safety Council), Annual PIN report. Year 2015



Safety

⇧

Reliability

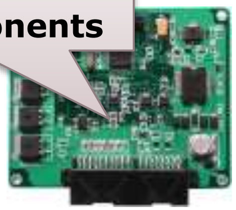
⇧

Test

Quality requirement: Why Zero PPM?

A modern mid-class car has ~**50 Applications** inside

A typical Automotive Application has ~**300 components**



NOT ACCEPTABLE

Car Quality
15.000 ppm = 1,5%

Application Quality
300 ppm

Component Quality
~ 1 ppm

Car Quality
< 500 ppm

Application Quality
< 10 ppm

Component Quality
0 ppm

ZERO DEFECT is MANDATORY

Software development for automotive embedded non volatile memories testing
Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

4 Software for Testing Development: why a process

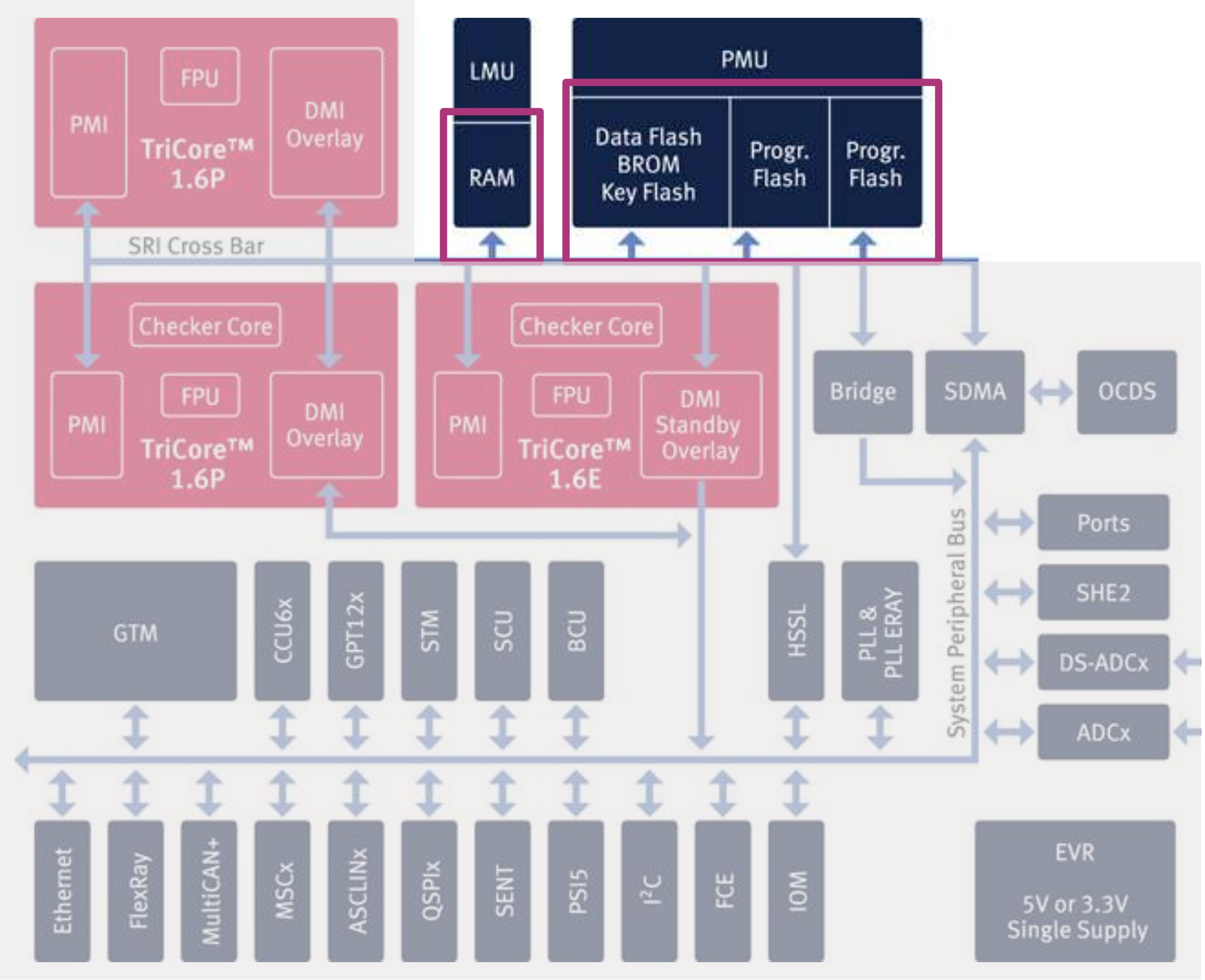
5 Software for Testing: Strategies for Quality assurance

6 Software for Testing Development: phases

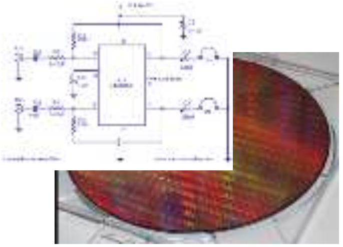
7 Conclusions and Questions

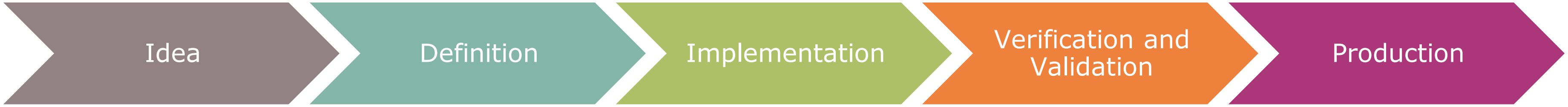
A SoC for automotive market

- Harvard architecture
- TriCore™ CPUs:
  - RISC Load/store core machine
  - Dual MAC (Multiply Accumulate Module)
  - ALU (Arithmetic Logic Unit)
  - Floating point Unit
- Program and Data Flash memory embedded
- Communication buses
- I/O peripherals
- Analogue/digital modules



Product design and production





Idea

- Ideation of the product

Definition

- Requirements collection
- Project release

Implementation

- Concept
- Specification
- Design (including synthesis and layout)

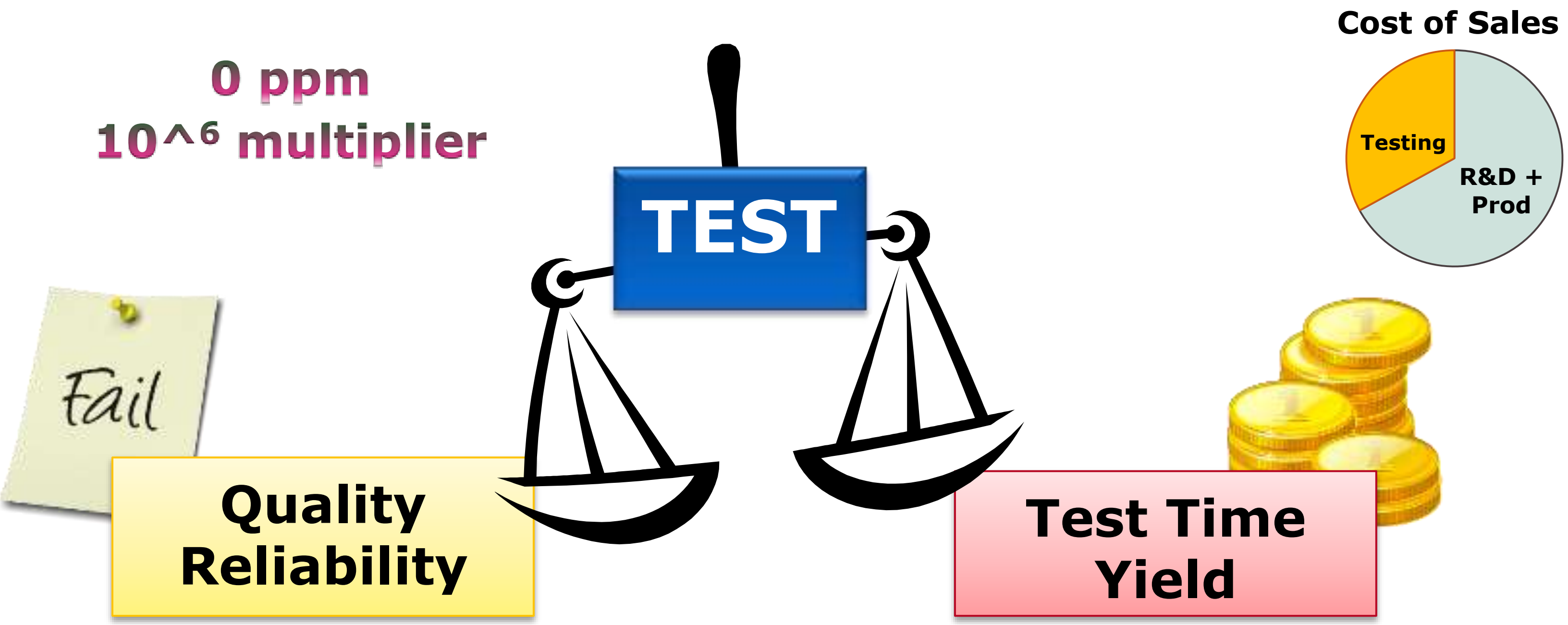
Verification and Validation

- Verification
- First sampling
- Validation test
- Characterization
- Qualification

Production

- Production Testing
- Analysis
- Optimization

Software for Testing Scenario 1/3



0 ppm

10^6 multiplier

TEST

Quality Reliability

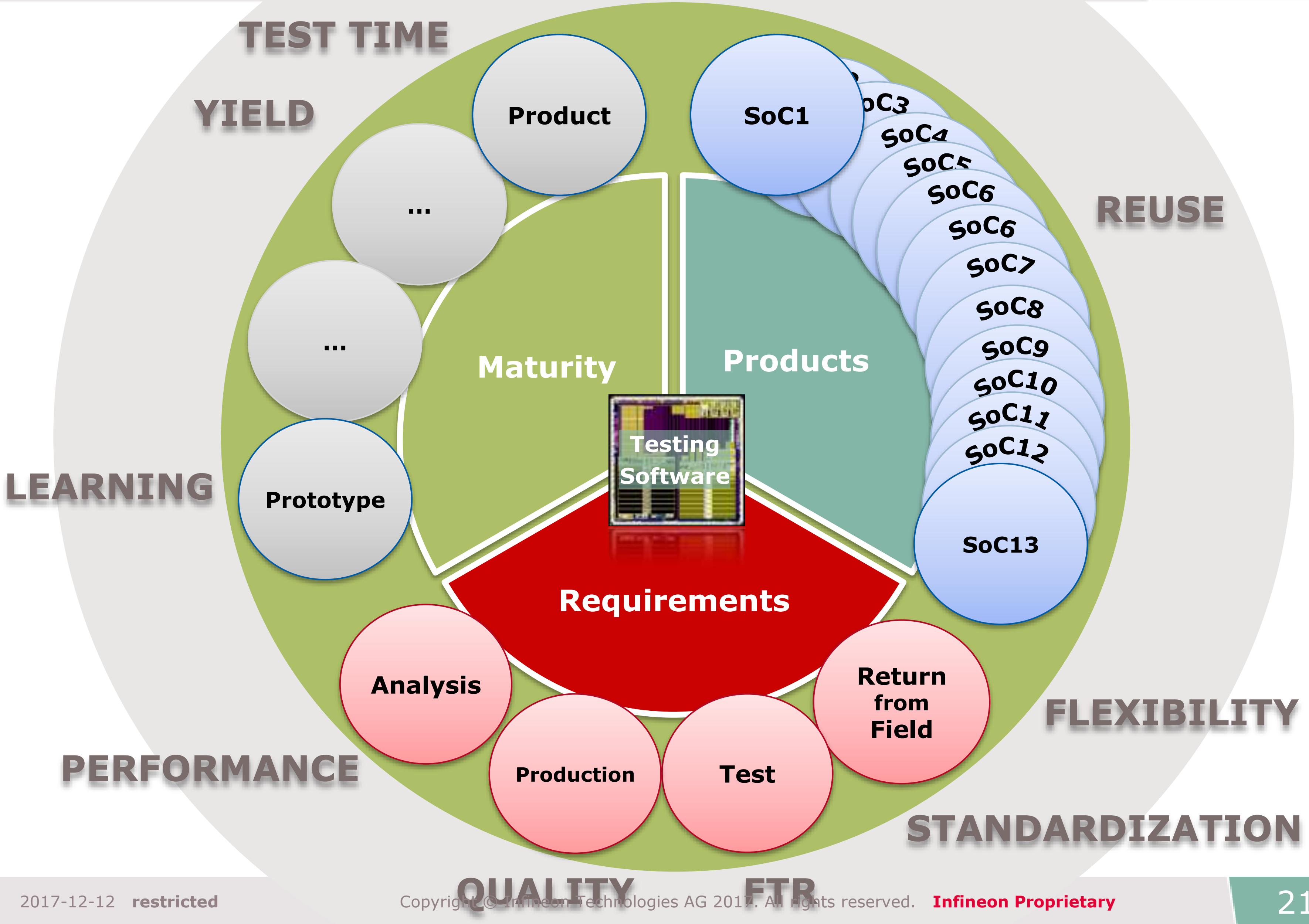
Test Time Yield

Cost of Sales

- eFlash product engineering has the aim to find the better **compromise** between **quality** of delivery and **Test Time** / **Yield** aspects

New test **algorithms** studying eFlash fault models
New test **solution** to optimize costs and test times

Software for Testing Scenario 2/3



Software for Testing Scenario 3/3

- Further constraints:
  - **Embedded software**
    - HW dedicated (based on Data Sheets and schematics)
    - Bit Manipulation (usage of Hex numbers, C language)
    - In-circuit debug (JTAG - no GUI)
  - **Limited SRAM** (10 .. 100KB)
  - **Limited calculation capabilities** (8MHz → 300MHz)
  - **Team work**:
    - 15 engineers working simultaneously on same code
  - **Tight deadline**:
    - few weeks to deliver «*First Time Right*» releases
  - **Traceability:**
    - all activities must be traced so to understand after years what happened in case of troubles at customer (FAR)
    - there must be a bilateral linking between requirements and delivery for validation

Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

4 Software for Testing Development: why a process

5 Software for Testing: Strategies for Quality assurance

6 Software for Testing Development: phases

7 Conclusions and Questions

Software quality

- What does “software quality” mean?
  - The level with which a system, component or process **satisfies user requirements and expectations**
  - **Conformity** with functional and non functional requirements, development standard and internal characteristics of a professional developed software.

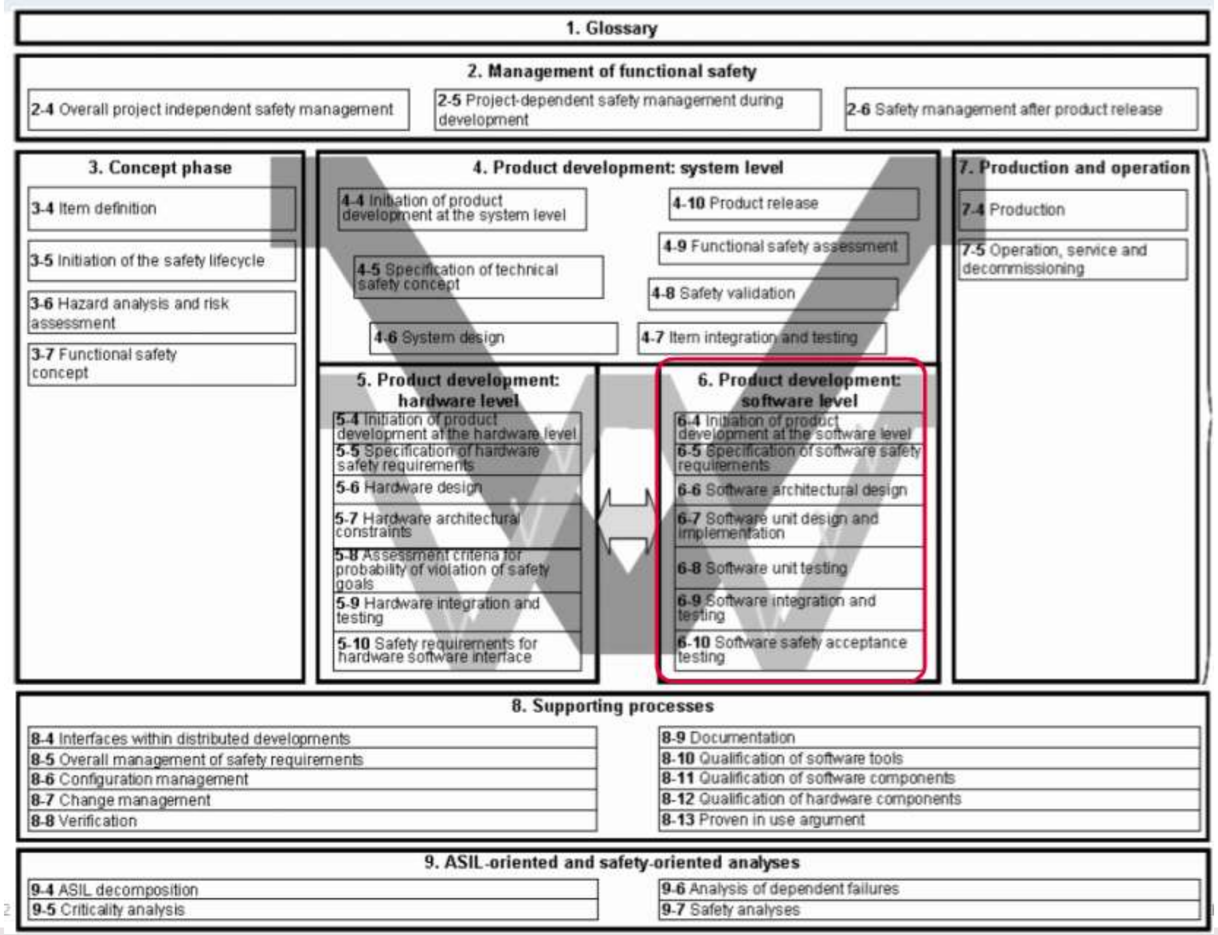
- Definition of software quality based on international standards and models
  - **ISO/IEC 9126**: defines **software product quality** according to a wide range of parameters and it is designed for users, developers, system administrators and customers.
  - **Automotive SPICE** (ASPICE): defines **technical standards** documents for the software development in automotive applications

- Software quality includes **product**, <u>**process**</u> and **producer** quality

ISO 26262

› **ISO 26262** is a standard focused on Automotive Electrical/Electronic Systems for **Functional Safety** (adapted from IEC 61508).

  – A part is dedicated to Product Development at Software Level



1. Glossary

2. Management of functional safety
- 2-4 Overall project independent safety management
- 2-5 Project-dependent safety management during development
- 2-6 Safety management after product release

3. Concept phase
- 3-4 Item definition
- 3-5 Initiation of the safety lifecycle
- 3-6 Hazard analysis and risk assessment
- 3-7 Functional safety concept

4. Product development: system level
- 4-4 Initiation of product development at the system level
- 4-5 Specification of technical safety concept
- 4-6 System design
- 4-7 Item integration and testing
- 4-8 Safety validation
- 4-9 Functional safety assessment
- 4-10 Product release

5. Product development: hardware level
- 5-4 Initiation of product development at the hardware level
- 5-5 Specification of hardware safety requirements
- 5-6 Hardware design
- 5-7 Hardware architectural constraints
- 5-8 Assessment criteria for probability of violation of safety goals
- 5-9 Hardware integration and testing
- 5-10 Safety requirements for hardware software interface

6. Product development: software level
- 6-4 Initiation of product development at the software level
- 6-5 Specification of software safety requirements
- 6-6 Software architectural design
- 6-7 Software unit design and implementation
- 6-8 Software unit testing
- 6-9 Software integration and testing
- 6-10 Software safety acceptance testing

7. Production and operation
- 7-4 Production
- 7-5 Operation, service and decommissioning

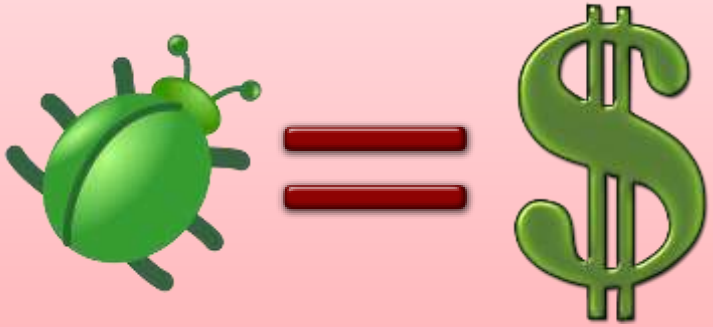
8. Supporting processes
- 8-4 Interfaces within distributed developments
- 8-5 Overall management of safety requirements
- 8-6 Configuration management
- 8-7 Change management
- 8-8 Verification
- 8-9 Documentation
- 8-10 Qualification of software tools
- 8-11 Qualification of software components
- 8-12 Qualification of hardware components
- 8-13 Proven in use argument

9. ASIL-oriented and safety-oriented analyses
- 9-4 ASIL decomposition
- 9-5 Criticality analysis
- 9-6 Analysis of dependent failures
- 9-7 Safety analyses

Why a process: cost of not using a process

WITHOUT Professional SW Dev. Model

Cost of SW Bugs:

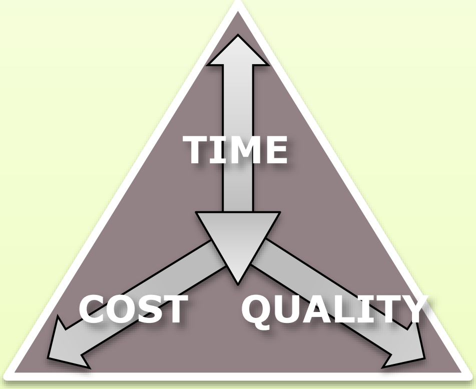


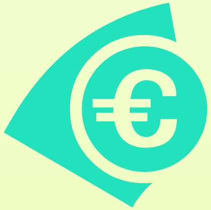
- **Spills** (huge cost “x100k€”)
- **FARs** (sizable cost “x10k€”)
- **SARs** (image loss at customer side)
- Increased test **costs** (can be sizeable for high volume products)
- Increased cost of **yield** (can be big for huge volume products)

Why a process: Benefits on projects

WITH Professional SW Dev. Model

- Better **estimates**
  - Identify **pitfalls** earlier
    - **Backup** developer approach



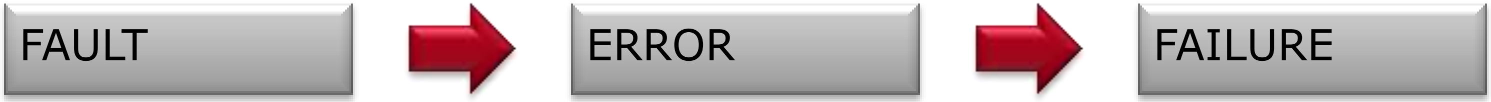


- "**Cooking** recipe" approach
  - Identify **pitfalls** earlier
    - Easy and reliable **pipeline**

- **Traceability**
  - Continuous **Learning**
    - **Stable** systems
      - Customer **informed**
- Focus less on human mistakes, more on **process** weakness

Definitions: Software fault, error and failure

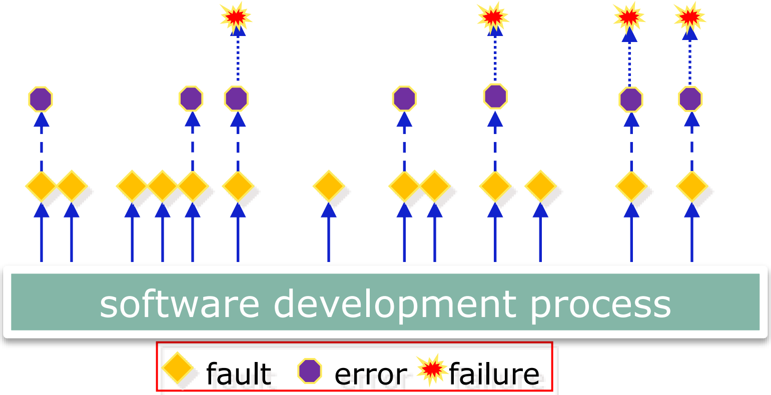
- Definitions:
  - **FAULT:** physical difference between the "good" system and the current one
  - **ERROR:** an error is the state of the system differs from the state in which it should be.
  - **FAILURE:** a Failure is a deviation of a system from its specified behaviour. It occurs when the system fails to do what it should do.
- Human errors lead the faults (encoding errors) which cause the processing (executing) errors and result in software failures
- Failure are detectable but it’s the fault that must be removed (fault is the cause and failure is the result).



Why a Process: Software error sources

- Where errors come from?
  - **Bad requirements** (with mistakes or incomplete)
  - **Design error**
  - **Encoding error**
  - **Fast testing process**
  - **Documentation error**

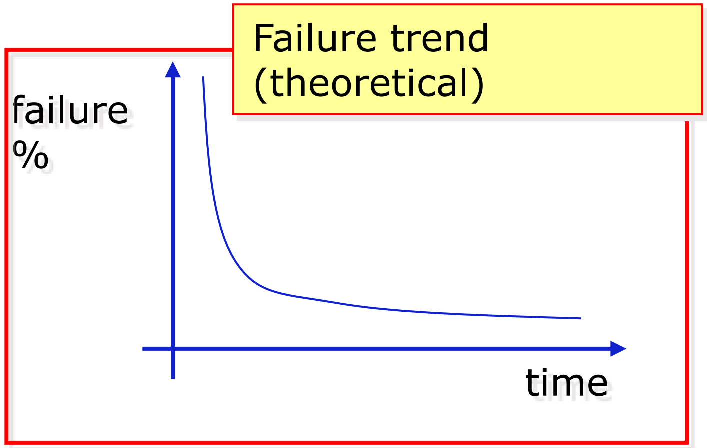
- Not all faults become errors and not all errors become failures

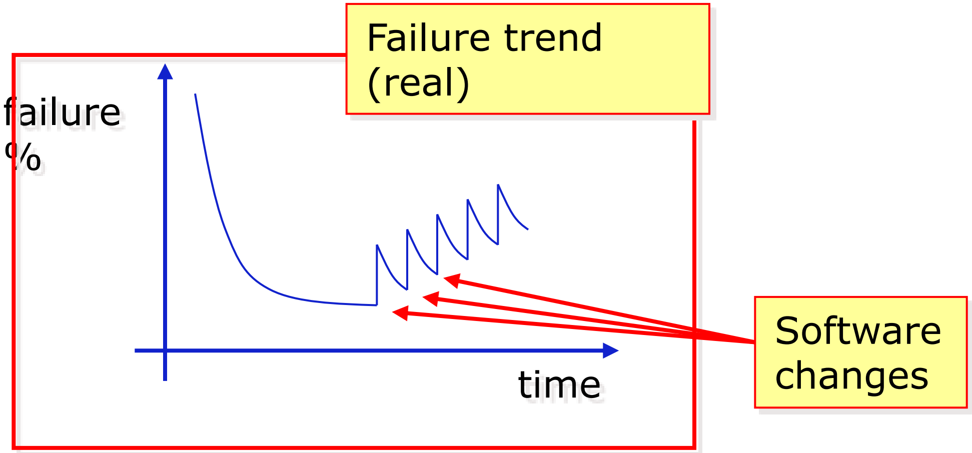


Note: A failure can appear after year of software use (e.g. Y2K bug).

Why a Process: Software failure reasons

1. **Software failure trend**
   - The theoretical behaviour follows a hyperbolic shape
   - The actual behaviour includes new failures due to **software changes** and update

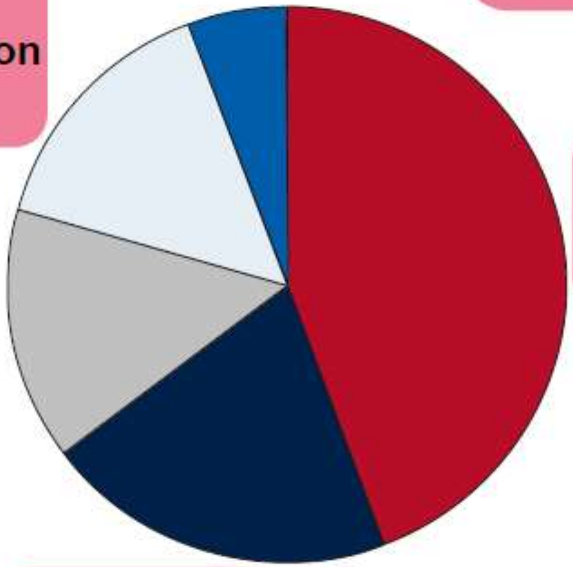




2. **Conflicts** of sharing same code/functionalities
3. **Redundant code functionalities**: doubled solutions not aligned
4. **Cross functionality** bugs (based on same code library)
5. **Cross applications** bugs (based on same code library)

Primary causes of failure in industry

Over 80% off all failures are caused before any user touches installations



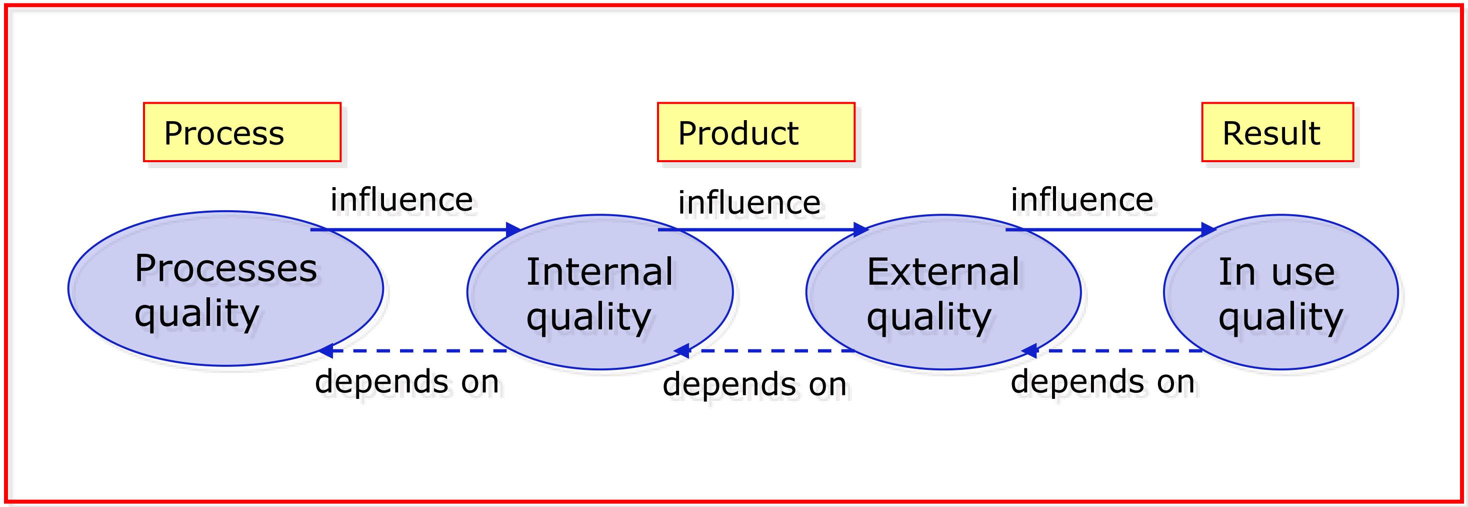
Only 15% of failures happen during operation and maintenance

Source: HSE UK report 1999, based on industrial accidents based on 34 incidents

ISO/IEC 9126 quality relationship

- The ISO/IEC 9126 defines 4 typologies of software quality:
  1. **process:** focus on the development process
  2. **internal**: focus on software internal attributes and it's independent from user and operative environment (static measurements)
  3. **external**: focus on software performance and functionality (running)
  4. **in use**: focus on efficiency and efficacy with which software satisfies user requirements (execution in real conditions)

ISO/IEC 9126 quality relationship



Software design: life cycle model

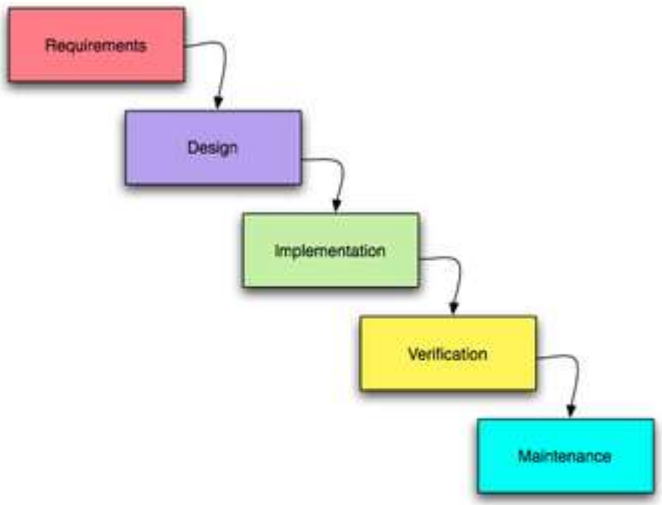
- A software development process may follow different life cycle models. The quality evaluation could also depend on that.

- There're many software design processes suitable for different contest
  - **Waterfall** model
  - **V-Model**
  - **Prototyping** model
  - **Spiral** model
  - **RAD** (Rapid Application Development)
  - **Agile** (Incremental) model
  - **XP** (Extreme Programming)
  - …

Waterfall model
1960s - 1970s

Non-iterative development approach:

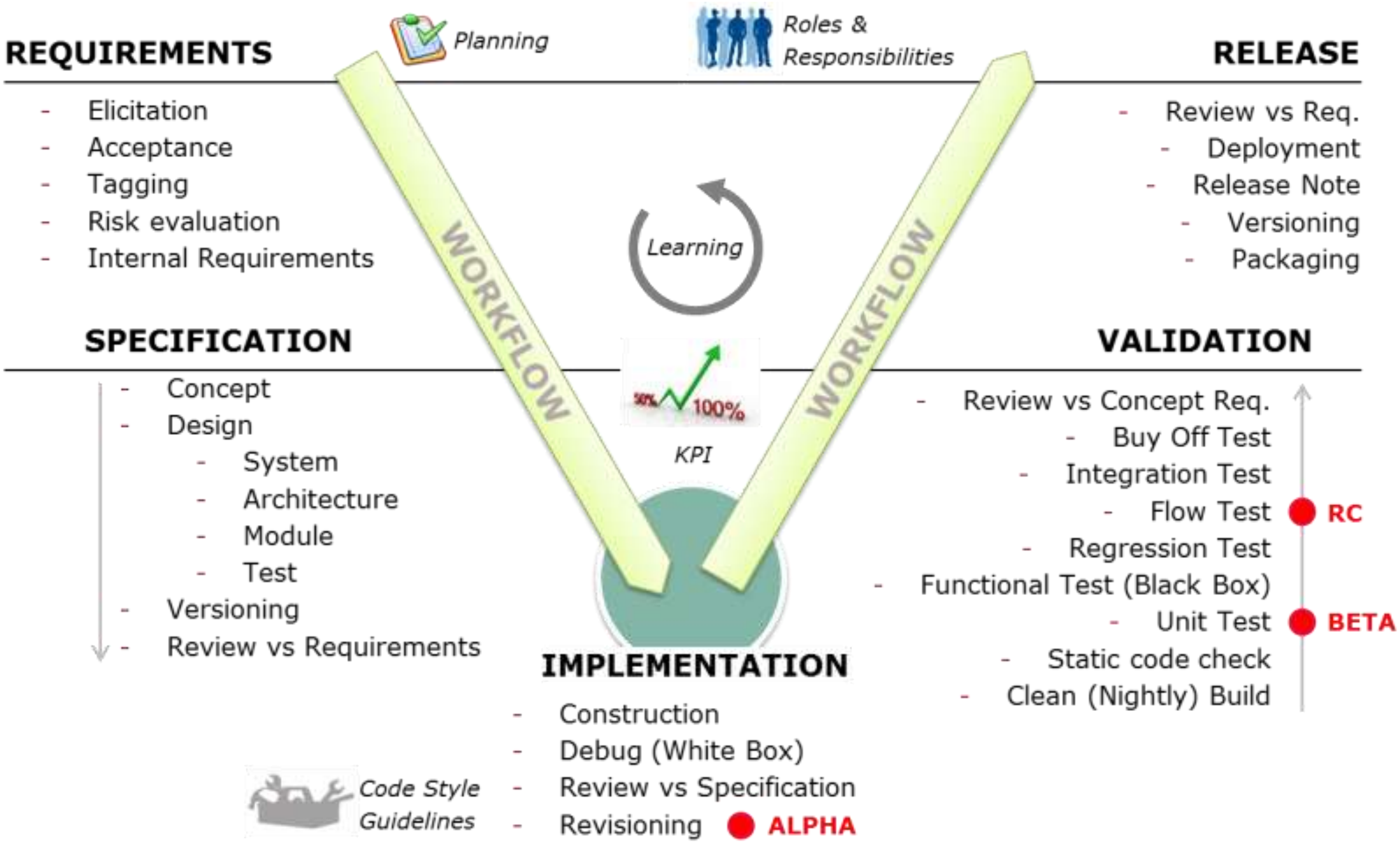
1. Requirements specification
2. Design
3. Construction (aka: implementation or coding)
4. Integration
5. Testing and debugging (aka: verification)
6. Installation
7. Maintenance



- **PROs**:
  - time spent early on making sure that requirements and design are absolutely correct is very useful in economic terms
  - it places emphasis on documentation (such as requirements documents and design documents) as well as source code
- **CONs**:
  - a bad idea in practice, mainly because it is impossible to get one phase of a software product's lifecycle "perfected" before moving on to the next phases and learning from them (“time spent in reconnaissance is seldom wasted”).

V-Model

- **V-Model** is a software development process extension of Waterfall model.
  - each phase of the development life cycle is associated to a related phase of testing.



REQUIREMENTS

- Elicitation
- Acceptance
- Tagging
- Risk evaluation
- Internal Requirements

SPECIFICATION

- Concept
- Design
  - System
  - Architecture
  - Module
  - Test
- Versioning
- Review vs Requirements

IMPLEMENTATION

- Construction
- Debug (White Box)
- Review vs Specification
- Revisioning ● ALPHA

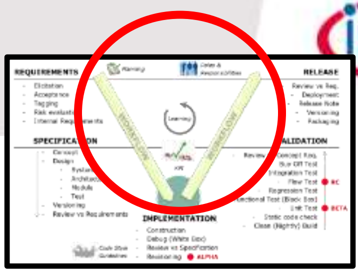
VALIDATION

- Review vs Concept Req.
- Buy Off Test
- Integration Test
- Flow Test ● RC
- Regression Test
- Functional Test (Black Box)
- Unit Test ● BETA
- Static code check
- Clean (Nightly) Build

RELEASE

- Review vs Req.
- Deployment
- Release Note
- Versioning
- Packaging

Overall Process



- **Description**: (glossary link)
  - *It's a set of **interrelated** means and activities that interact to achieve a result.*
- **ASPICE reference**:
  - Aspice Standard
- **Benefits**:
  - Reduce development-caused quality problems
  - Improve efficiency
  - Improve know-how transfer between staff members
  - Versatile staffing of people
  - Performance less dependent from each individual
  - No improvisation
  - Clear insight on project status
  - No/Less “firefighting” → Time for improvement

Process Identification	Process name
ENG.1	Requirements elicitation
ENG.2	System requirements analysis
ENG.3	System architectural design
ENG.4	Software requirements analysis
ENG.5	Software design
ENG.6	Software construction
ENG.7	Software integration test
ENG.8	Software testing
ENG.9	System integration test
ENG.10	System testing

Test SW Development Means 1/3

Documentation (*Wer Schreibt Der Bleibet – Scripta Manent*)

- Each Process phase is described in details into dedicated handbooks or wiki
- Benefits:
  - Team work support (reference & training)
  - AUDIT
  - Avoid process exceptions

FTOS

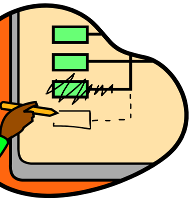
32-Bit Microcontroller

Development Handbook

Microcontrollers

Organization

- Resource Manager, Technical Leader, Engineers, …



Workflow manager

- All change requests are traced
- Benefits:
  - Team work support (documents sharing)
  - Traceability (for history reference and activities monitoring)

JIRA

Test SW Development Means 2/3

Versioning System

- Source code is stored and handled into Versioning System
- Benefits:
  - To keep trace of versioning (revisions)
  - To manage collaboration conflicts (Team work)
  - To permit prototype development (branching)

Development tools

- IDE (Eclipse)
- Compiler (Tasking)
- Debugger (UDE/PLS)
- Debug Platform (board + SoC)

Source code documentation and analysis

- Doxygen
- Static analysis (MisraC compliance, Sonar)

Means 3/3

- **Verification tools**
  - Automatic **Nightly build** (Jenkins)
  - (Automatic nightly) **uTest** execution
  - (Automatic nightly) **Regression** execution
  - Atlassian Crucible

- **Release Tool**
  - Each release is done automatically with a dedicated tool
  - Benefits:
    - **Package build** from clean code (downloaded from versioning)
    - **Configurable** to keep consistent contents
    - Aligned with last **working** and **verified build**
    - Automatic **version tagging**
    - Automatic **release note** definition

Software development for automotive embedded non volatile memories testing
Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

4 Software for Testing Development: why a process

5 **Software for Testing: Strategies for Quality assurance**

6 Software for Testing Development: phases

7 Conclusions and Questions

Implementation: how to guarantee Quality Software

- Quality Software implementation methodology can focus on Robustness or delay to Verification avoiding that **BUGS** will go outside within releases.

Robustness

Verification

Trade-off

- Clean Code
- MisraC rules
- Implementation Specification
- Developers training
- Documentation
- Syntax check

*Carbon Monoxide Detectors
Personal Emergency
Motion Detectors
*Smoke & Heat Sensors
Door & Window Protection
Keychain Touchpad
*Freeze Sensors
*Optional Upgrades
*Water Sensors

- Map file checker
- Debug on SoC
- Unit Test
- Regression Test

Robustness: The "Clean Code" philosophy

- "**Clean Code**" is a coding style that consists in a set of standard rules and best practices which drives the developer through a more reliable, maintainable and readable software coding

Grady Booch: *"Clean code is simple and direct. It reads like a well-written prose. Clean code never obscures the designer's intent."*

Bjarne Stroustrup (inventor of C++): *"I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide… Clean code does one thing well"*

Ward Cunningham: (inventor of Wiki): *"You know you are working when each routine you read turns out to be pretty much what you expect"*

The “Clean Code” basic principle

“Use meaningful names”

Intention revealing names
Pronounceable names
Searchable names

“Functions”

Small!!
Do one thing
Reading code from top to bottom
Function arguments
Have no side effects

“Comments”

Explain yourself in code
Good Vs. Bad comments

“Formatting”

Vertical formatting rules
Horizontal formatting rules
Team rules

““Clean Code: A Handbook of Agile Software Craftsmanship (R. Martin)” → http://books.google.it/books?id=_i6bDeoCQzsC&printsec=frontcover&dq=clean+code+online+book&hl=it&sa=X&ei=-WICUYfDBcbzsgb7sIDIBw&ved=0CDgQ6AEwAA#v=onepage&q&f=false

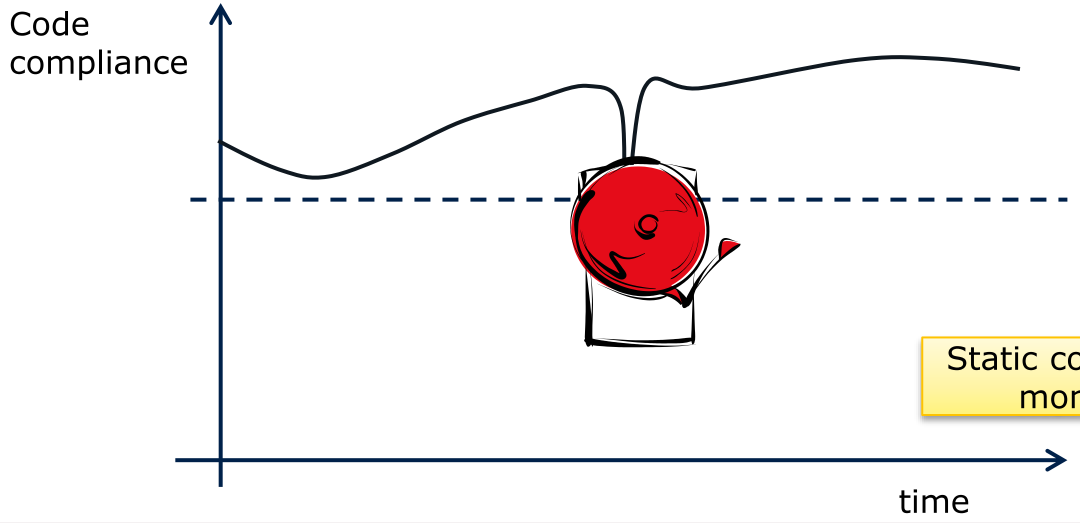
The “Clean Code”

- What is good for?
  - Write a easily **readable** source code
    - without clean code the ratio between code understanding and code writing is 10:1
  - Get a really **maintainable** and debuggable source code.
  - Produce performing and **reliable** source code.
  - Obtain **modular** and scalable functions.

- Why to use it?
  - Produce long-term **supportable** and **expandable** software.
  - Reduce the working time needed to **refactoring**, maintain and debug software source code.
  - Easily **reuse** functions and algorithm in place.
  - Easily introduce **new people** to code already in place (e.g. Libraries, OS, etc…).

MISRA-C

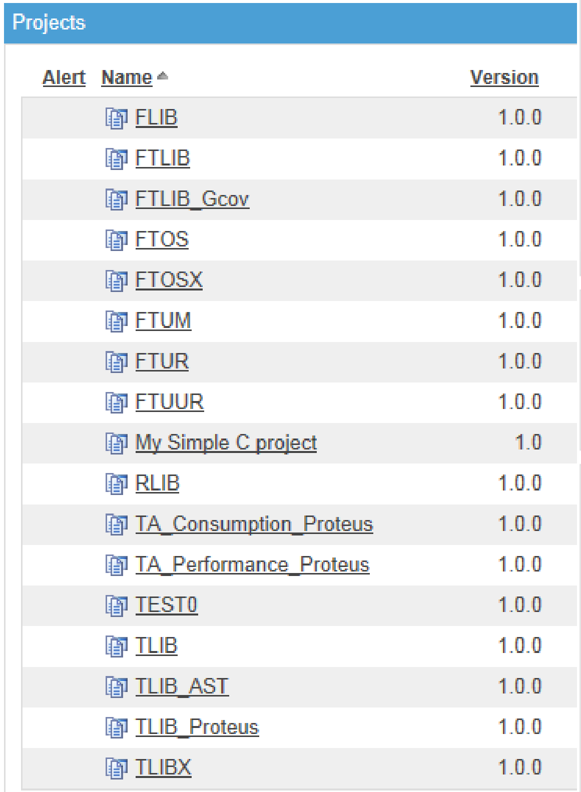
- **MISRA-C** is a software development standard for the C programming language developed by MISRA (*Motor Industry Software Reliability Association*). Its aims are to facilitate code safety, portability and reliability in the context of embedded systems.
- MISRA has evolved as a widely accepted model for best practices by leading developers in sectors including aerospace, telecom, medical devices, defence, railway, and others



Static code analysis monitoring

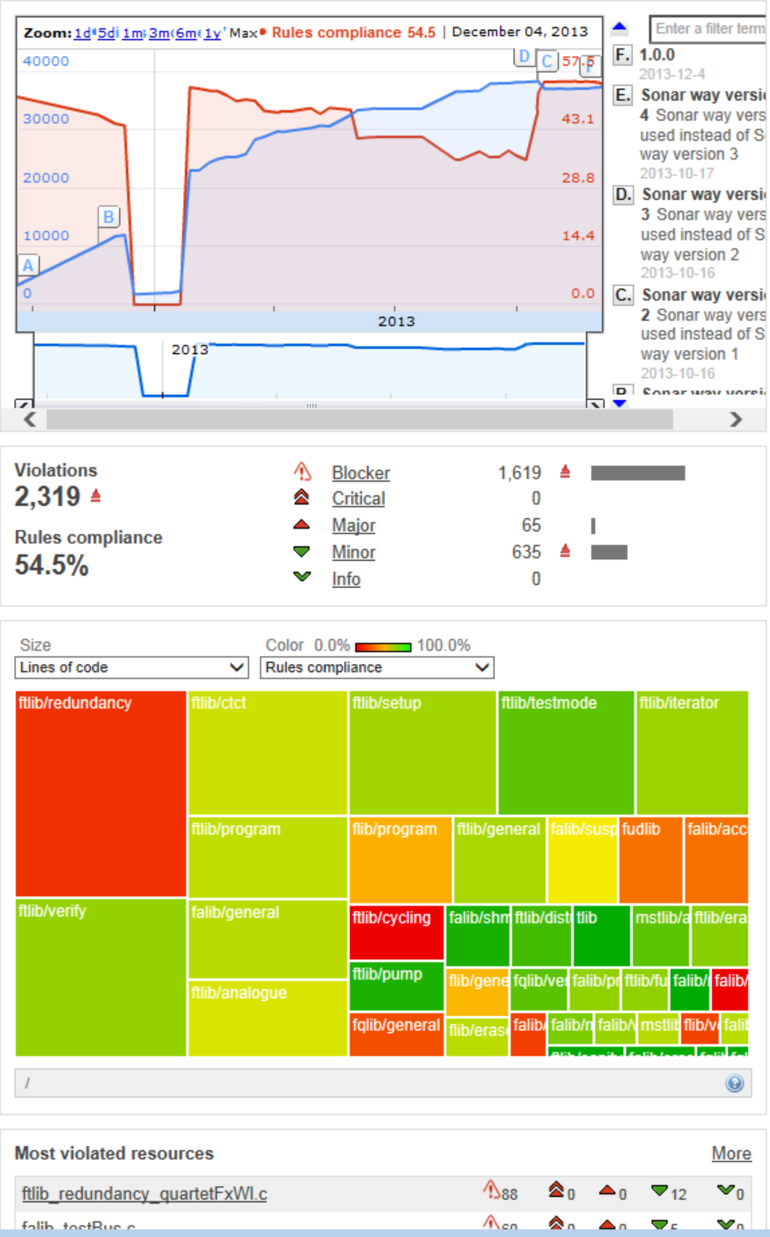
Sonar: rules compliance monitoring

- Projects
- Treemap
- Reviews
- Dependencies



Projects

Alert	Name	Version
	FLIB	1.0.0
	FTLIB	1.0.0
	FTLIB_Gcov	1.0.0
	FTOS	1.0.0
	FTOSX	1.0.0
	FTUM	1.0.0
	FTUR	1.0.0
	FTUUR	1.0.0
	My Simple C project	1.0
	RLIB	1.0.0
	TA_Consumption_Proteus	1.0.0
	TA_Performance_Proteus	1.0.0
	TEST0	1.0.0
	TLIB	1.0.0
	TLIB_AST	1.0.0
	TLIB_Proteus	1.0.0
	TLIBX	1.0.0



Zoom: 1d 5d 1m 3m 6m 1y Max
Rules compliance 54.5 | December 04, 2013
40000
30000
20000
10000
0
57.5
43.1
28.8
14.4
0.0
A
B
C
D
F
2013
2013

Enter a filter term

F. 1.0.0
2013-12-4

E. Sonar way versi
4 Sonar way vers
used instead of S
way version 3
2013-10-17

D. Sonar way versi
3 Sonar way vers
used instead of S
way version 2
2013-10-16

C. Sonar way versi
2 Sonar way vers
used instead of S
way version 1
2013-10-16

Violations
2,319

Rules compliance
54.5%

Blocker	1,619	▲
Critical	0	
Major	65	
Minor	635	▲
Info	0	

Size: Lines of code
Color 0.0% 100.0%: Rules compliance

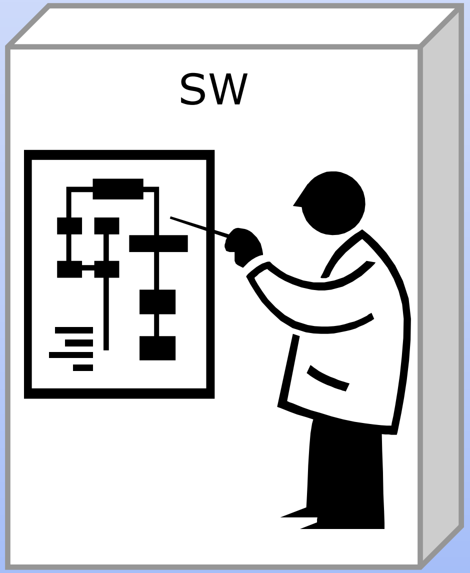
Most violated resources — More

ftlib_redundancy_quartetFxWI.c	88	0	0	12	0
falib_testBus.c	60	0	0	5	0

Verification:
White Box Vs. Black box testing (1/2)

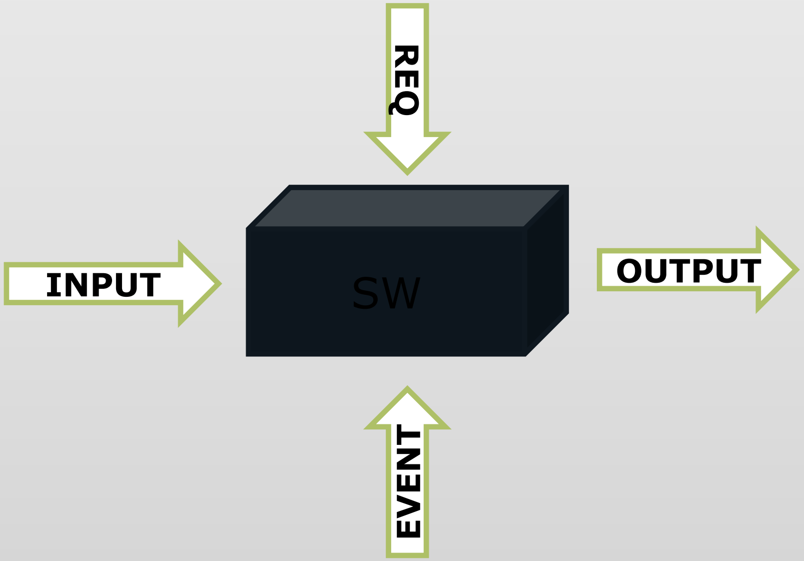
White box test

White-box testing (also known as **clear box testing**, **glass box testing**, **transparent box testing** and **structural testing**) tests internal structures or workings of a program



Back box test

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it



White box test: dual targeting

- Embedded Code is designed to run at least two platforms:
  - the **final target hardware**
  - the **development system**
- Executing some tests already before real hardware integration:
  - It’s a practical way to completely isolate the software under test to avoid debugging hardware and software simultaneously.
  - It allows to test code before hardware is ready, designs with hardware independence.

Unit Test **Final target debug**

White Box testing:
Unit test theory

- **Unit Testing** is a method by which individual units of source code are tested to determine if they are fit for use
- A **Unit** is the smallest testable part of an application
  - In procedural programming a unit may be an individual **Function** or **Procedure**
- Unit tests are created by **Programmers** or occasionally by **White Box Testers** during the development process

- Ideally, each **Test Case** is independent from the others
- Substitutes can be used to assist testing a module in isolation
  - **Method Stubs**
  - **Mock Objects**
  - **Fakes**
  - **Test Harnesses**

Unit test: Pro and Cons

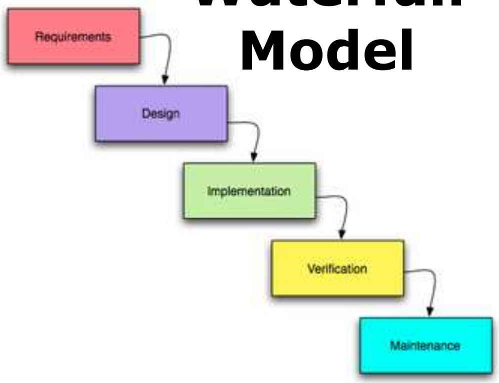
- **Benefits**
  - Allows the programmer to **Refactor** code at a later date, and make sure the module still works correctly
  - May reduce uncertainty in the units themselves and can be used in a **Bottom-Up testing** style approach. By testing the parts of a program first and then testing the sum of its parts, **Integration** testing becomes much easier
  - Provides a sort of living **Documentation** of the system
  - May take the place of **Formal Design**

- **Constrains**
  - Not catch **Integration** errors or **Broader System-Level** errors
    - functions performed across multiple units
    - non-functional test areas such as performance

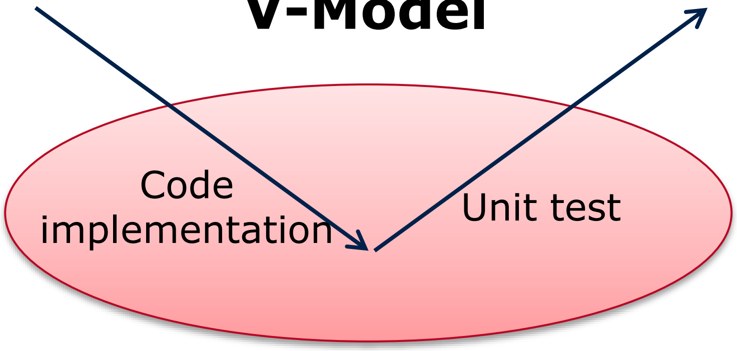
Unit test integration

- Depending on the development process, unit test can be developed before or after the code implementation

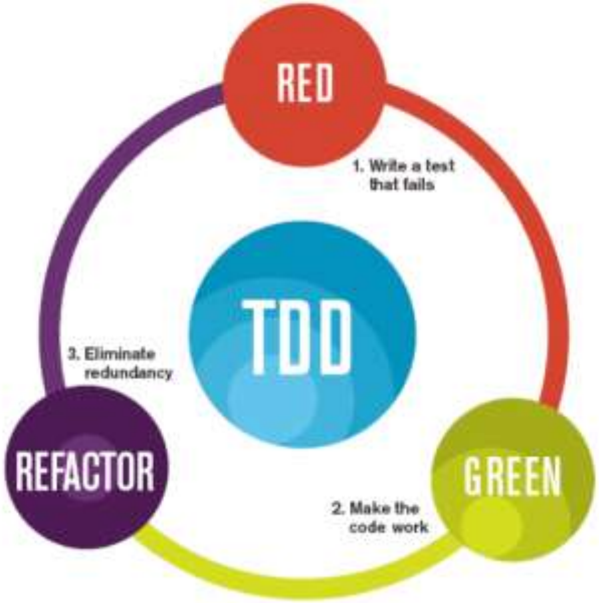
Waterfall Model



V-Model



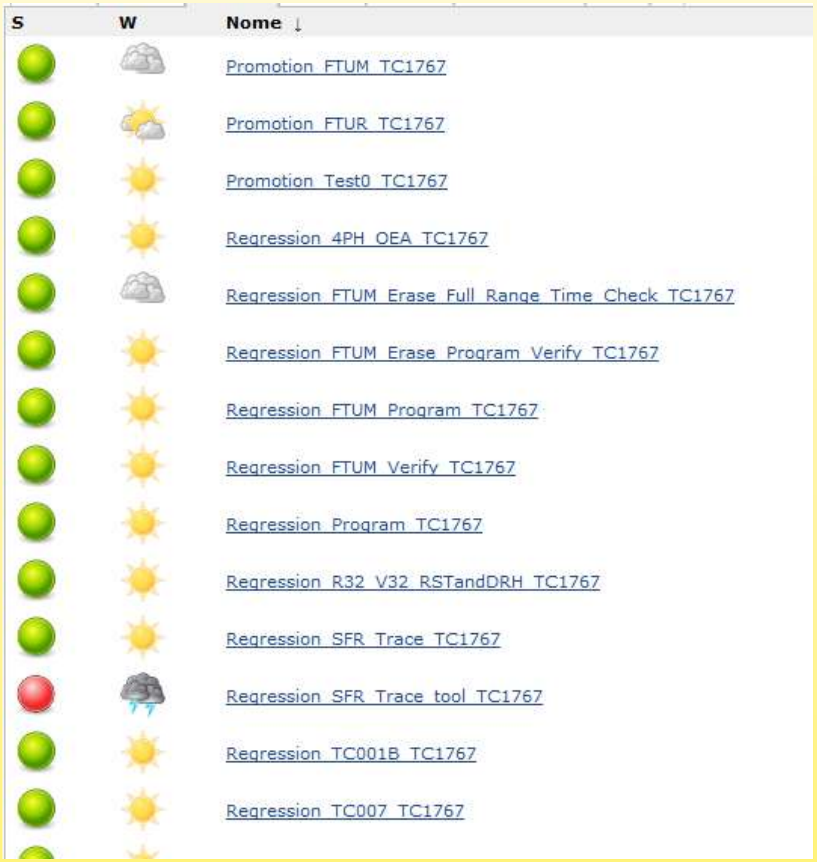
TDD (Test driven development test)



Unit test monitoring

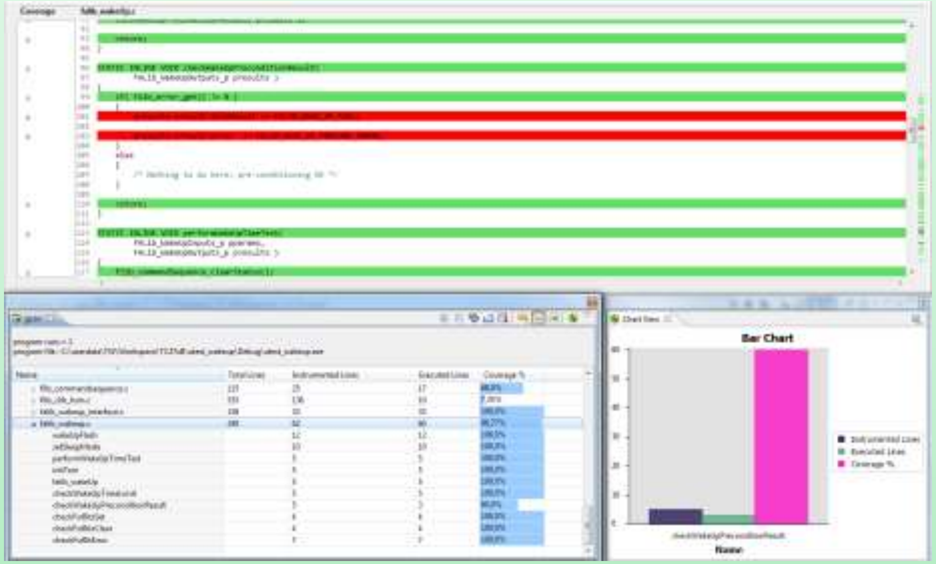
› In order to benefit from unit test power, it is needed to constantly monitor unit test outputs

Test execution results



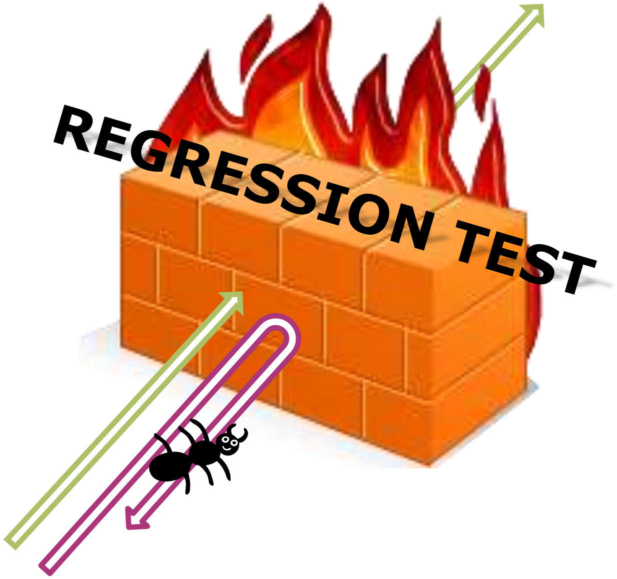
S	W	Nome ↓
🟢	☁	Promotion_FTUM_TC1767
🟢	⛅	Promotion_FTUR_TC1767
🟢	☀	Promotion_Test0_TC1767
🟢	☀	Regression_4PH_OEA_TC1767
🟢	☁	Regression_FTUM_Erase_Full_Range_Time_Check_TC1767
🟢	☀	Regression_FTUM_Erase_Program_Verify_TC1767
🟢	☀	Regression_FTUM_Program_TC1767
🟢	☀	Regression_FTUM_Verify_TC1767
🟢	☀	Regression_Program_TC1767
🟢	☀	Regression_R32_V32_RSTandDRH_TC1767
🟢	☀	Regression_SFR_Trace_TC1767
🔴	⛈	Regression_SFR_Trace_tool_TC1767
🟢	☀	Regression_TC001B_TC1767
🟢	☀	Regression_TC007_TC1767

Test coverage



Black box test: Regression

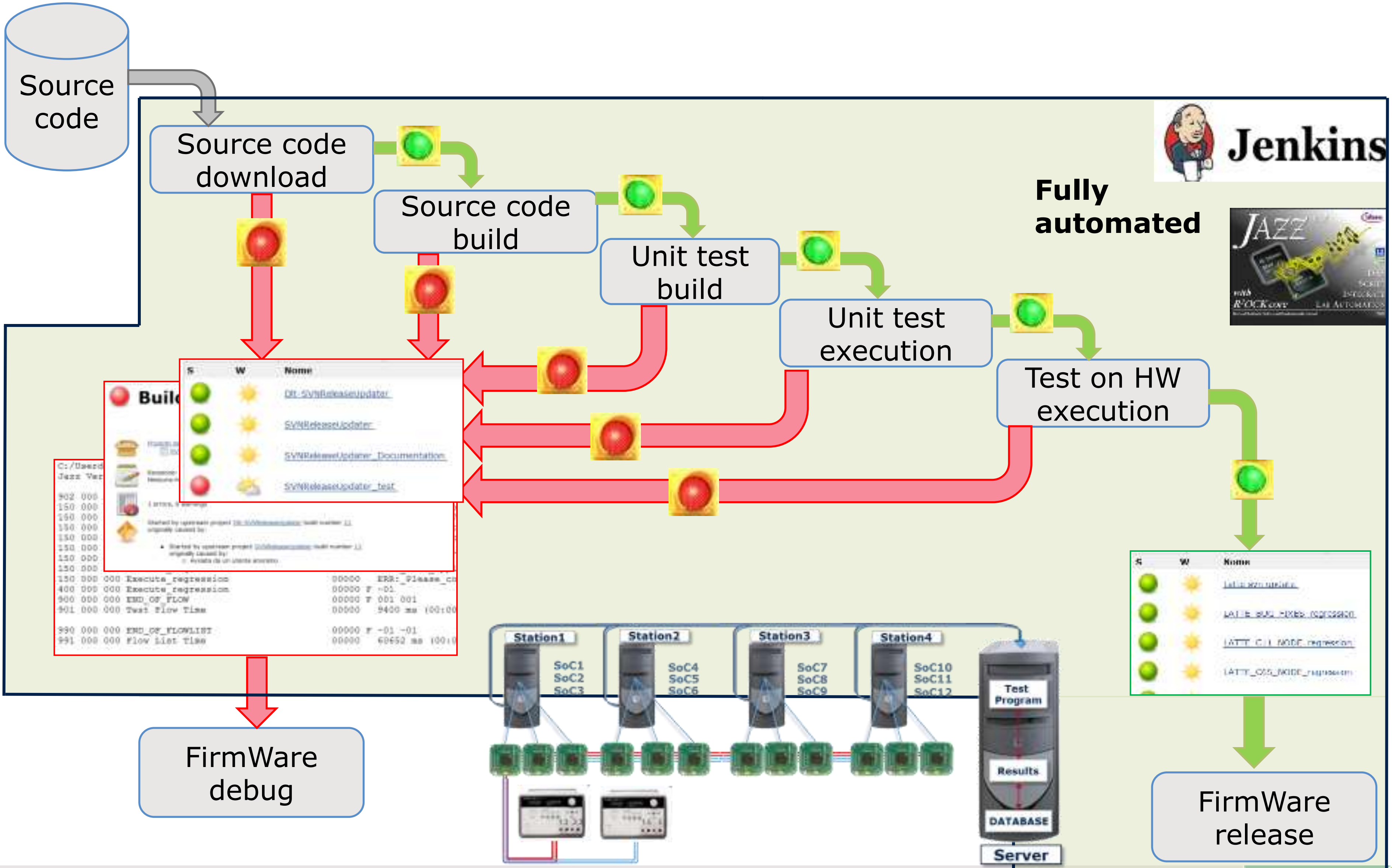
- **Regression** testing is any type of software testing that seeks to uncover new software bugs, or *regressions*, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them
- The intent of regression testing is to **ensure** that a change such as those mentioned above has not introduced new faults



How to build a good regression?

- A good regression test
  - … has to look at **PASS** conditions
  - … has to look at **FAIL** conditions
  - … can be **automatically** executed
  - … is **independent** from previous executed test
  - … should return **pass or fail**
  - … should monitor the software **execution time**

Testing flow environment



Source code

Source code download

Source code build

Unit test build

Unit test execution

Test on HW execution

Fully automated

FirmWare debug

FirmWare release

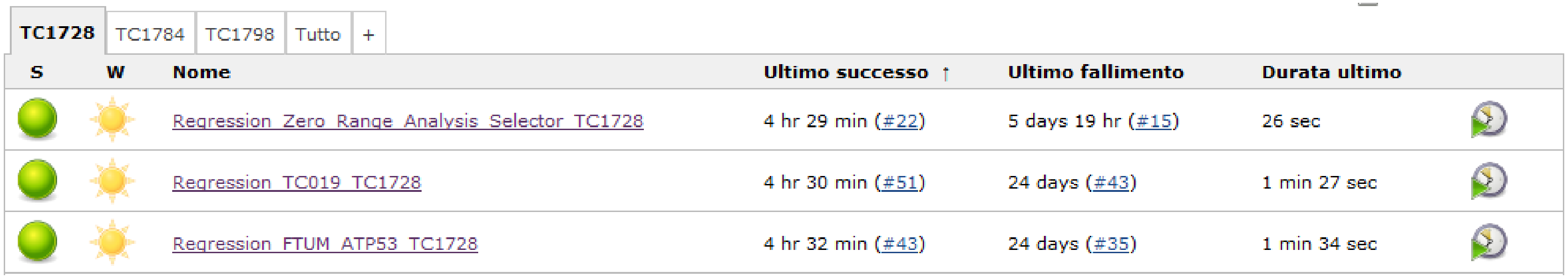
S	W	Nome
●	☀	Dt-SVNReleaseUpdater
●	☀	SVNReleaseUpdater
●	☀	SVNReleaseUpdater_Documentation
●	☀	SVNReleaseUpdater_test

```
C:/Userd
Jazz Ver

902 000
150 000
150 000
150 000
150 000
150 000
150 000
150 000
150 000 000 Execute_regression                00000   ERR:_Please_cr
400 000 000 Execute_regression                00000 F -01
900 000 000 END_OF_FLOW                       00000 F 001 001
901 000 000 Test Flow Time                    00000   9400 ms (00:00

990 000 000 END_OF_FLOWLIST                   00000 F -01 -01
991 000 000 Flow List Time                    00000   60652 ms (00:0
```

Jenkins regression output

- Jenkins provides a graphical interface to monitor the status of the regression environment divided by product in a set of regression flow lists.

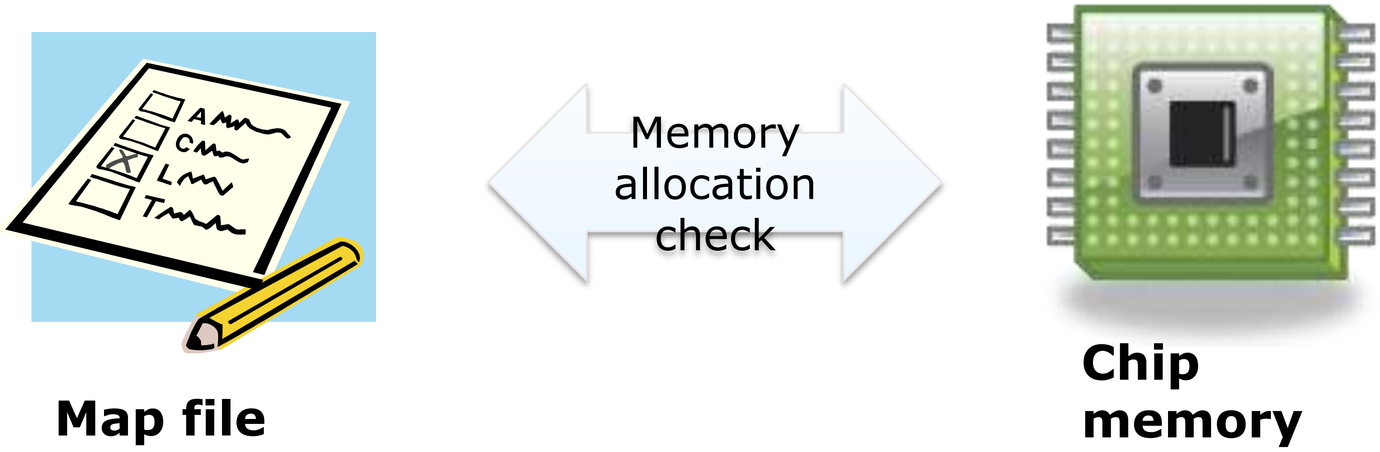
- For each flow list there are:
  - a green/red ball depending on last execution is PASS or FAIL
  - a "weather" situation that monitors the last 5 executions
  - time of last PASS and last FAIL
  - duration of execution
  - a button to execute manually the build



S	W	Nome	Ultimo successo ↑	Ultimo fallimento	Durata ultimo	
●	☀	Regression_Zero_Range_Analysis_Selector_TC1728	4 hr 29 min (#22)	5 days 19 hr (#15)	26 sec	▶
●	☀	Regression_TC019_TC1728	4 hr 30 min (#51)	24 days (#43)	1 min 27 sec	▶
●	☀	Regression_FTUM_ATP53_TC1728	4 hr 32 min (#43)	24 days (#35)	1 min 34 sec	▶

Functional check: Linker output file checker

- During embedded software development it is important to check that code is allocated in the proper area to guarantee:
  - Code execution performance
  - Resource conflicts
  - (Proper interface with production test machine)



Map file

Memory allocation check

Chip memory

Software development for automotive embedded non volatile memories testing
Agenda

1 Infineon MicroController: a SoC for automotive

2 SoC Quality and Testing

3 Software for Testing: Complexity

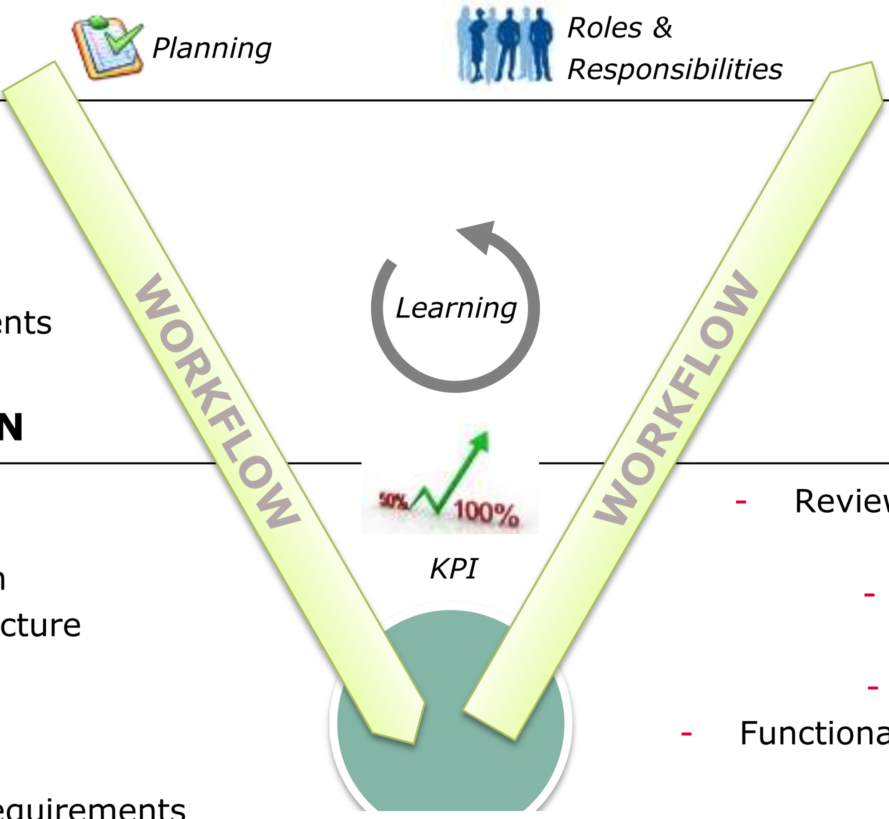
4 Software for Testing Development: why a process

5 Software for Testing: Strategies for Quality assurance

6 **Software for Testing Development: phases**

7 Conclusions and Questions

SW Development Model



REQUIREMENTS

- Elicitation
- Acceptance
- Tagging
- Risk evaluation
- Internal Requirements

SPECIFICATION

- Concept
- Design
  - System
  - Architecture
  - Module
  - Test
- Versioning
- Review vs Requirements

IMPLEMENTATION

- Construction
- Debug (White Box)
- Review vs Specification
- Revisioning ● **ALPHA**

Code Style Guidelines

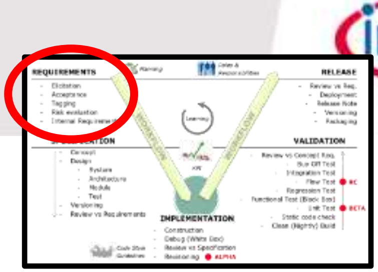
RELEASE

- Review vs Req.
- Deployment
- Release Note
- Versioning
- Packaging

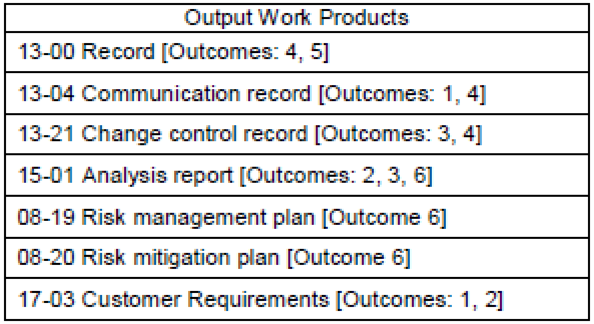
VALIDATION

- Review vs Concept Req.
- Buy Off Test
- Integration Test
- Flow Test ● **RC**
- Regression Test
- Functional Test (Black Box)
- Unit Test ● **BETA**
- Static code check
- Clean (Nightly) Build

Phase: Requirements



- **Description**: (glossary link)
  - *Establishing the **needs** of stakeholders that are **to be solved** by software.*
- **ASPICE reference**:
  - ENG.1 Requirements elicitation: The purpose of the Requirements elicitation process is to **gather**, **process**, and **track** evolving **customer needs** and requirements throughout the life of the product and/or service so as to establish a requirements baseline that serves as the basis for defining the needed work products.
  - ENG.2 System requirements analysis
  - ENG.4 Software requirements analysis

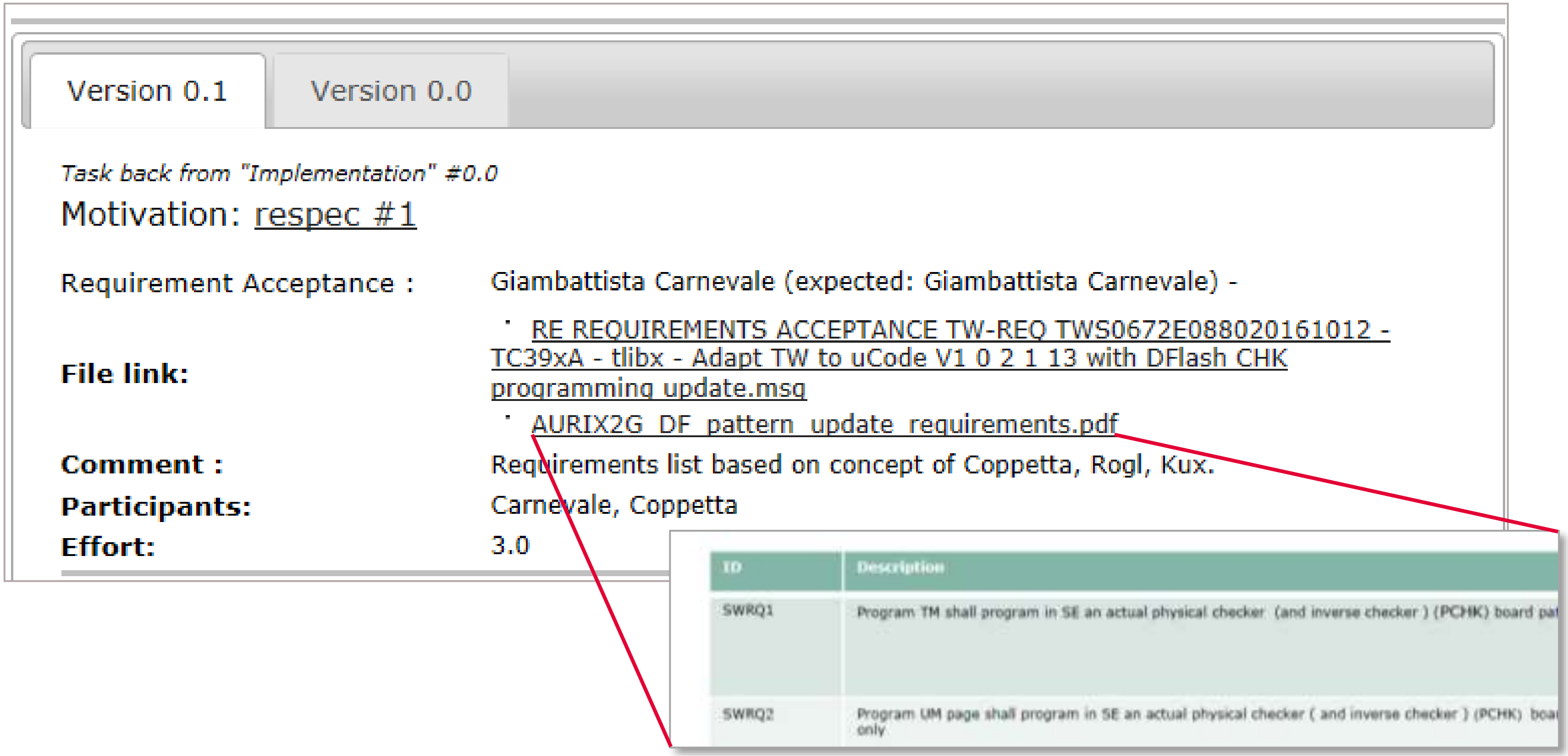


Output Work Products
13-00 Record [Outcomes: 4, 5]
13-04 Communication record [Outcomes: 1, 4]
13-21 Change control record [Outcomes: 3, 4]
15-01 Analysis report [Outcomes: 2, 3, 6]
08-19 Risk management plan [Outcome 6]
08-20 Risk mitigation plan [Outcome 6]
17-03 Customer Requirements [Outcomes: 1, 2]

- **Benefits**:
  - Continuing **communication** with the customer;
  - Continuous **monitoring** of customer needs;
  - Customers can easily determine the **status** and disposition of their requests;
  - Associated **risks** assessed and their **impact** managed.

Phase: Requirements – Example

- **Requirements are:**
  - Taken from **JIRA** tickets or collected from **Stakeholders**
  - **Frozen** and Stored
  - **Accepted** formally by engineer
  - **Tagged**



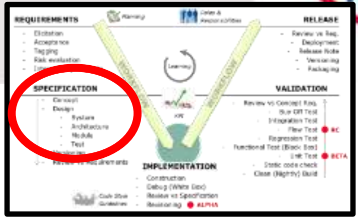
Task back from "Implementation" #0.0

Motivation: respec #1

Requirement Acceptance :	Giambattista Carnevale (expected: Giambattista Carnevale) -
File link:	- RE REQUIREMENTS ACCEPTANCE TW-REQ TWS0672E088020161012 - TC39xA - tlibx - Adapt TW to uCode V1 0 2 1 13 with DFlash CHK programming update.msg - AURIX2G DF pattern update requirements.pdf
Comment :	Requirements list based on concept of Coppetta, Rogl, Kux.
Participants:	Carnevale, Coppetta
Effort:	3.0

ID	Description
SWRQ1	Program TM shall program in SE an actual physical checker (and inverse checker) (PCHK) board pat
SWRQ2	Program UM page shall program in SE an actual physical checker (and inverse checker) (PCHK) boa only

Phase: Specification

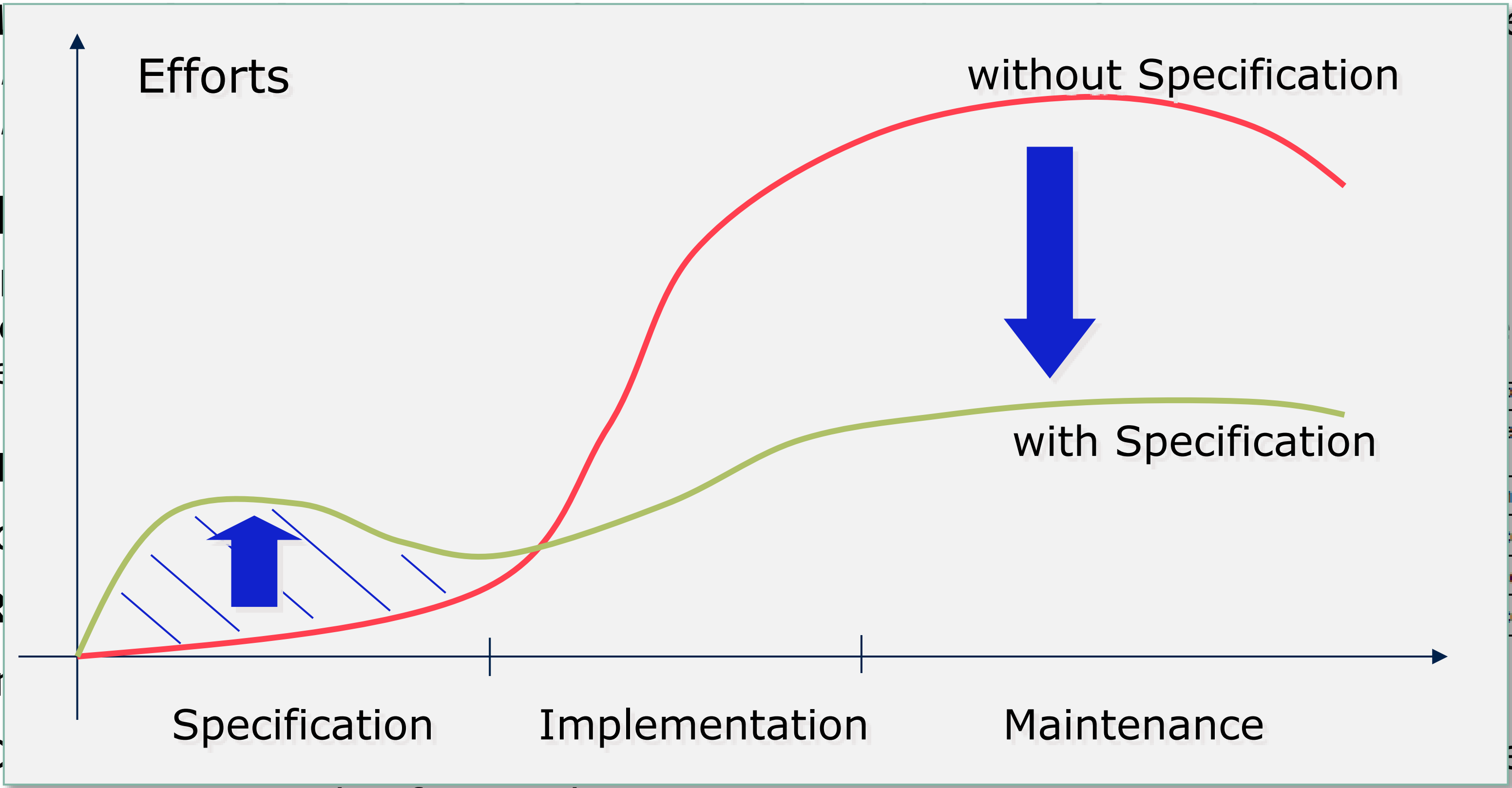


- **Description**: (glossary link)
  - *W … es in*
    o
    a

- **ASI**
  - El … a
    d
    re

Products
esign [Outcome 1
[Outcome 2, 3, 4
ome 2, 6, 7]
ome 4, 5]
ome 5]

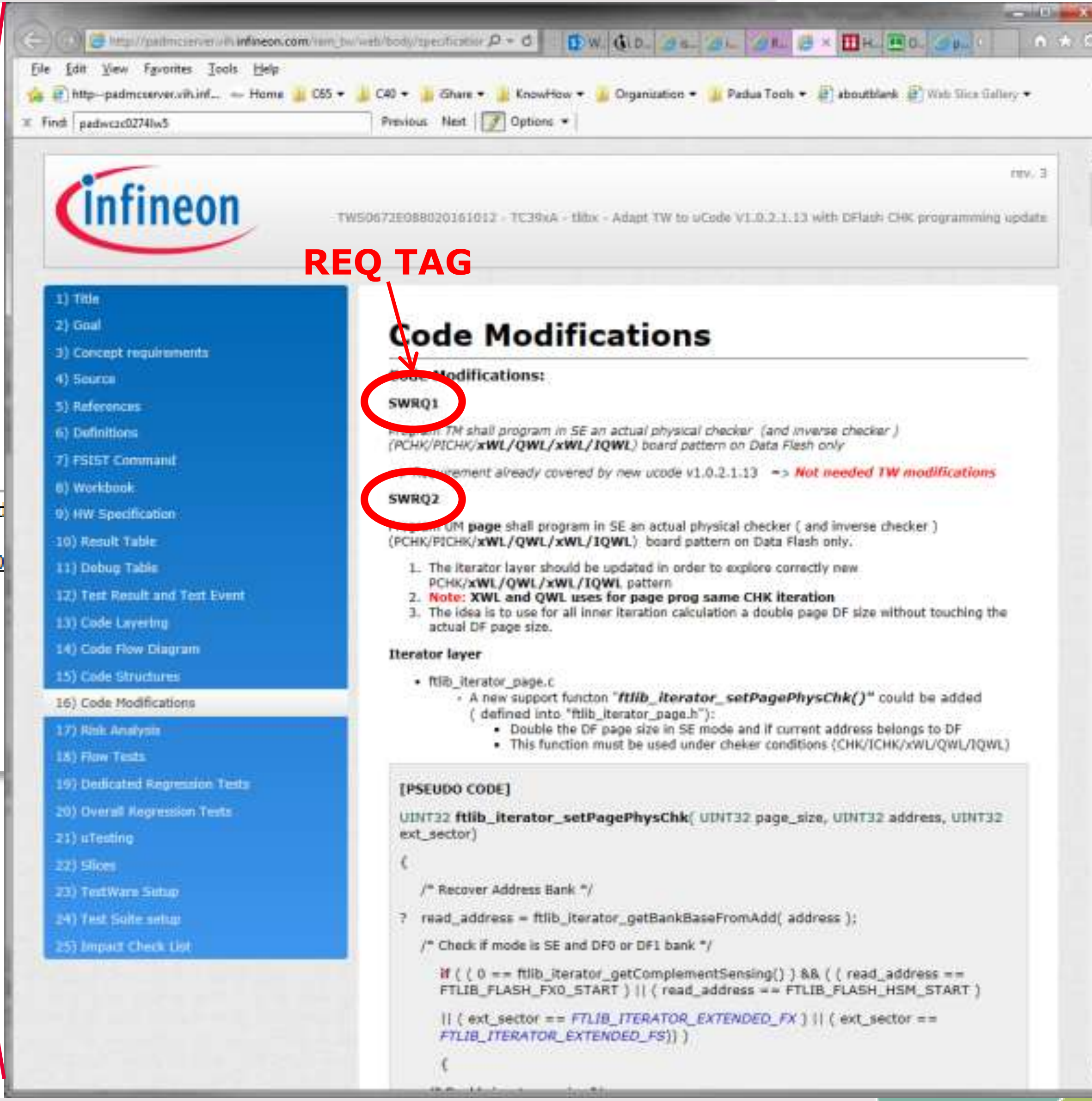
- **Ben**
  - Id
  - **R**
  - In
  - C … e
    requirements and software design
  - Design of each SW **unit** and related **test** is defined



Phase: Specification – Example

- › **Specification focuses on:**
  - › **Product Req**. and **SW Req**.
  - › **Technical** references
  - › **Architectural, Module, Unit** changes
  - › **Test** design
  - › **Risk** assessment

Specification :	Giambattista Carnevale (expected 18:31:45
Specification link:	./specification/public/specs/TWS0
Specification link Final:	Def Specification Rev. 3
Specification link Editor:	...
Command:	0x22/0x21
Sub Command:	ALL
Slice:	-
SVN start:	13043


http://padmcserver.vih.infineon.com/rem_tw/web/body/specification
File Edit View Favorites Tools Help
http--padmcserver.vih.inf... Home C65 C40 iShare KnowHow Organization Padua Tools aboutblank Web Slice Gallery
Find padwczc0274lw5 Previous Next Options
infineon
rev. 3

TWS0672E088020161012 - TC39xA - tlibx - Adapt TW to uCode V1.0.2.1.13 with DFlash CHK programming update

REQ TAG

Code Modifications

Code Modifications:

SWRQ1

*Program TM shall program in SE an actual physical checker (and inverse checker) (PCHK/PICHK/**xWL/QWL/xWL/IQWL**) board pattern on Data Flash only*

Requirement already covered by new ucode v1.0.2.1.13 -> ***Not needed TW modifications***

SWRQ2

Program UM **page** shall program in SE an actual physical checker (and inverse checker) (PCHK/PICHK/**xWL/QWL/xWL/IQWL**) board pattern on Data Flash only.

1. The iterator layer should be updated in order to explore correctly new PCHK/**xWL/QWL/xWL/IQWL** pattern
2. Note: **XWL and QWL uses for page prog same CHK iteration**
3. The idea is to use for all inner iteration calculation a double page DF size without touching the actual DF page size.

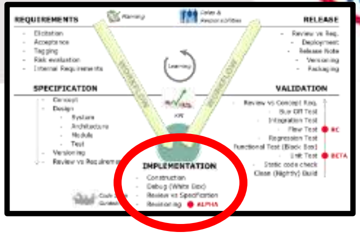
Iterator layer

- ftlib_iterator_page.c
  - A new support functon "***ftlib_iterator_setPagePhysChk()***" could be added (defined into "ftlib_iterator_page.h"):
    - Double the DF page size in SE mode and if current address belongs to DF
    - This function must be used under cheker conditions (CHK/ICHK/xWL/QWL/IQWL)

[PSEUDO CODE]

```
UINT32 ftlib_iterator_setPagePhysChk( UINT32 page_size, UINT32 address, UINT32
ext_sector)
{
    /* Recover Address Bank */
?   read_address = ftlib_iterator_getBankBaseFromAdd( address );
    /* Check if mode is SE and DF0 or DF1 bank */
        if ( ( 0 == ftlib_iterator_getComplementSensing() ) && ( ( read_address ==
        FTLIB_FLASH_FX0_START ) || ( read_address == FTLIB_FLASH_HSM_START )
        || ( ext_sector == FTLIB_ITERATOR_EXTENDED_FX ) || ( ext_sector ==
        FTLIB_ITERATOR_EXTENDED_FS)) )
        {
```

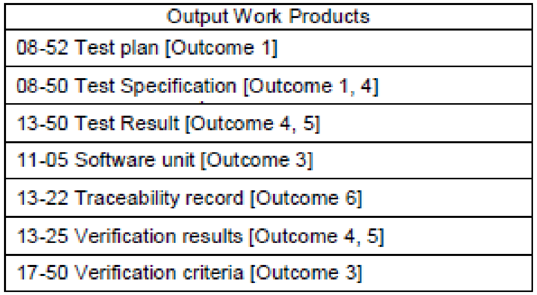
Phase: Implementation



- ## Description: (glossary link)
  - *It is the detailed **creation** of working meaningful software through a combination of **coding**, **verification**, **unit testing**, **integration testing**, and **debugging**.*

- ## ASPICE reference:
  - ENG.6 Software construction: The purpose of the Software construction process is to **produce verified software** units that properly reflect the software design.

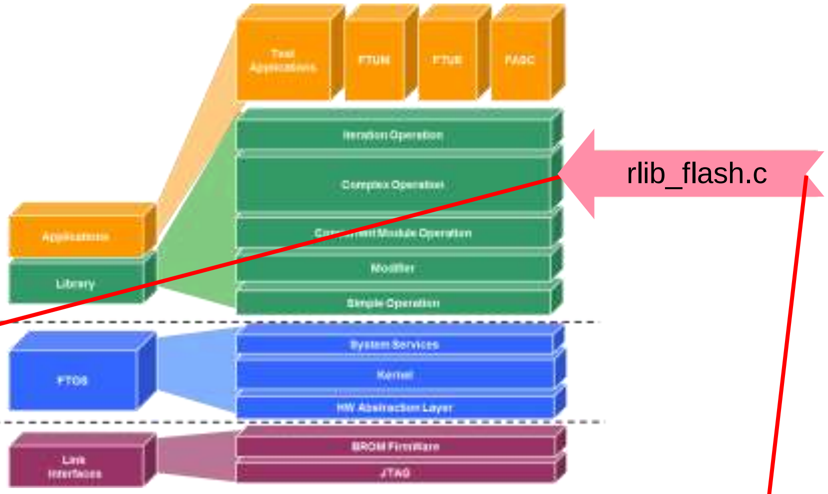
- ## Benefits:
  - SW units **analyzed** for correctness and testability
  - SW units are **verified** according to verification strategy
  - **Results** of unit verification are recorded
  - Consistency and **bilateral** traceability are established between software detailed design and software units

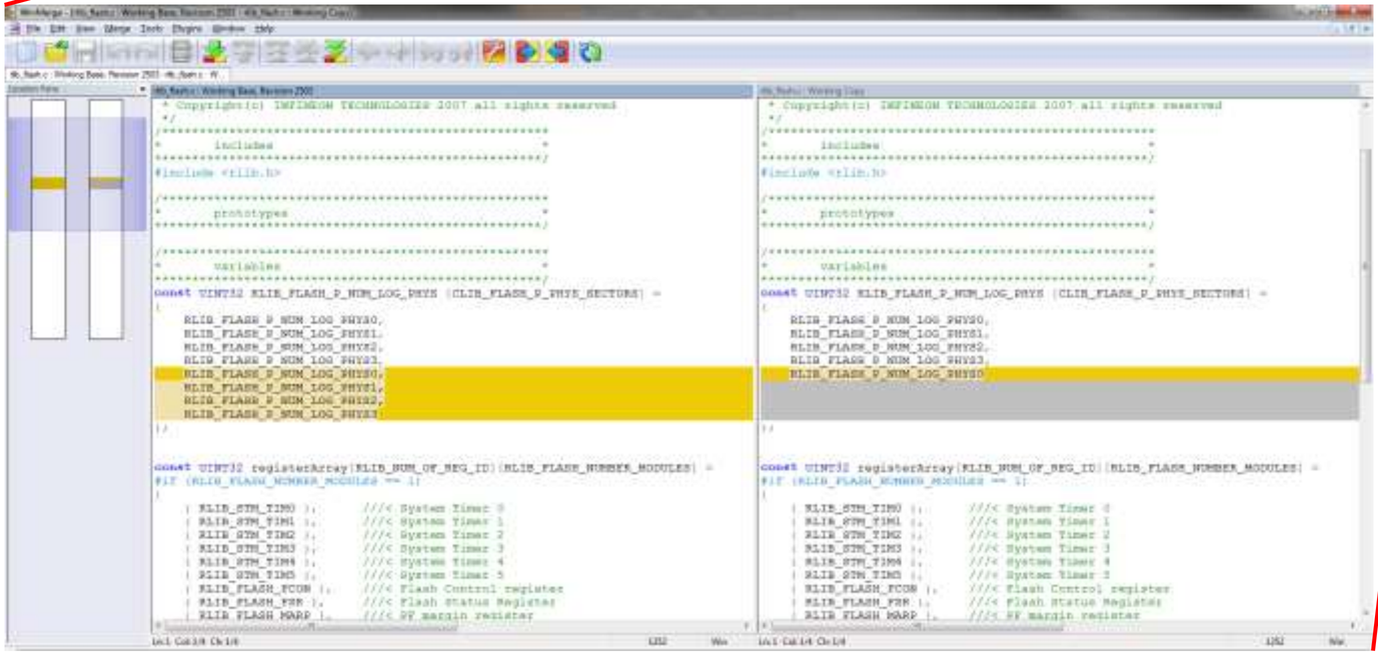


Output Work Products
08-52 Test plan [Outcome 1]
08-50 Test Specification [Outcome 1, 4]
13-50 Test Result [Outcome 4, 5]
11-05 Software unit [Outcome 3]
13-22 Traceability record [Outcome 6]
13-25 Verification results [Outcome 4, 5]
17-50 Verification criteria [Outcome 3]

Phase: Implementation – Example

- **Implementation focuses on:**
  - Code **writing**
  - Code **building** and **debug**
  - **uTest** Code development
  - **4 eyes** Code review



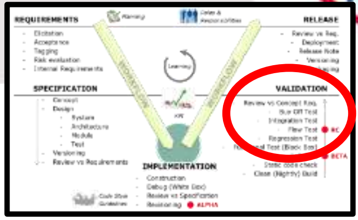


WinMerge - [rlib_flash.c - Working Base, Revision 2503 - rlib_flash.c - Working Copy]
File Edit View Merge Tools Plugins Window Help
rlib_flash.c - Working Base, Revision 2503
rlib_flash.c - Working Copy
Ln: 1 Col: 1/4 Ch: 1/4
1252
Win

```
 * Copyright(c) INFINEON TECHNOLOGIES 2007 all rights reserved
 */
/**********************************************************
*        includes                                         *
**********************************************************/
#include <rlib.h>

/**********************************************************
*        prototypes                                       *
**********************************************************/

/**********************************************************
*        variables                                        *
**********************************************************/
const UINT32 RLIB_FLASH_P_NUM_LOG_PHYS [CLIB_FLASH_P_PHYS_SECTORS] =
{
    RLIB_FLASH_P_NUM_LOG_PHYS0,
    RLIB_FLASH_P_NUM_LOG_PHYS1,
    RLIB_FLASH_P_NUM_LOG_PHYS2,
    RLIB_FLASH_P_NUM_LOG_PHYS3,
    RLIB_FLASH_P_NUM_LOG_PHYS0,
    RLIB_FLASH_P_NUM_LOG_PHYS1,
    RLIB_FLASH_P_NUM_LOG_PHYS2,
    RLIB_FLASH_P_NUM_LOG_PHYS3
};


const UINT32 registerArray[RLIB_NUM_OF_REG_ID][RLIB_FLASH_NUMBER_MODULES] =
#if (RLIB_FLASH_NUMBER_MODULES == 1)
{
    { RLIB_STM_TIM0 },        ///< System Timer 0
    { RLIB_STM_TIM1 },        ///< System Timer 1
    { RLIB_STM_TIM2 },        ///< System Timer 2
    { RLIB_STM_TIM3 },        ///< System Timer 3
    { RLIB_STM_TIM4 },        ///< System Timer 4
    { RLIB_STM_TIM5 },        ///< System Timer 5
    { RLIB_FLASH_FCON },      ///< Flash Control register
    { RLIB_FLASH_FSR },       ///< Flash Status Register
    { RLIB_FLASH_MARP },      ///< PF margin register
```

```
 * Copyright(c) INFINEON TECHNOLOGIES 2007 all rights reserved
 */
/**********************************************************
*        includes                                         *
**********************************************************/
#include <rlib.h>

/**********************************************************
*        prototypes                                       *
**********************************************************/

/**********************************************************
*        variables                                        *
**********************************************************/
const UINT32 RLIB_FLASH_P_NUM_LOG_PHYS [CLIB_FLASH_P_PHYS_SECTORS] =
{
    RLIB_FLASH_P_NUM_LOG_PHYS0,
    RLIB_FLASH_P_NUM_LOG_PHYS1,
    RLIB_FLASH_P_NUM_LOG_PHYS2,
    RLIB_FLASH_P_NUM_LOG_PHYS3,
    RLIB_FLASH_P_NUM_LOG_PHYS0
};


const UINT32 registerArray[RLIB_NUM_OF_REG_ID][RLIB_FLASH_NUMBER_MODULES] =
#if (RLIB_FLASH_NUMBER_MODULES == 1)
{
    { RLIB_STM_TIM0 },        ///< System Timer 0
    { RLIB_STM_TIM1 },        ///< System Timer 1
    { RLIB_STM_TIM2 },        ///< System Timer 2
    { RLIB_STM_TIM3 },        ///< System Timer 3
    { RLIB_STM_TIM4 },        ///< System Timer 4
    { RLIB_STM_TIM5 },        ///< System Timer 5
    { RLIB_FLASH_FCON },      ///< Flash Control register
    { RLIB_FLASH_FSR },       ///< Flash Status Register
    { RLIB_FLASH_MARP },      ///< PF margin register
```

Phase: Validation

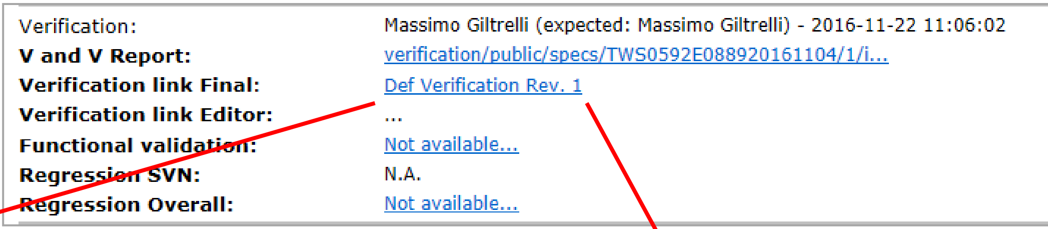


- **Description**: (glossary link)
  - *The process of **evaluating** software during or at the end of the development process to determine whether it **satisfies** specified **requirements**. [IEEE-STD-610].*
- **ASPICE reference:**
  - ENG.8 Software testing: The purpose of the Software testing process is to **confirm** that the integrated software **meets** the defined software **requirements**.
- **Benefits:**
  - Tests are based on a **strategy** and on **specifications**
  - **Results** of software testing are recorded
  - Consistency and **bilateral traceability** are established between software requirements and software test specification including test cases
  - A **regression** test strategy is applied for re-testing the integrated software when a change in software items occur

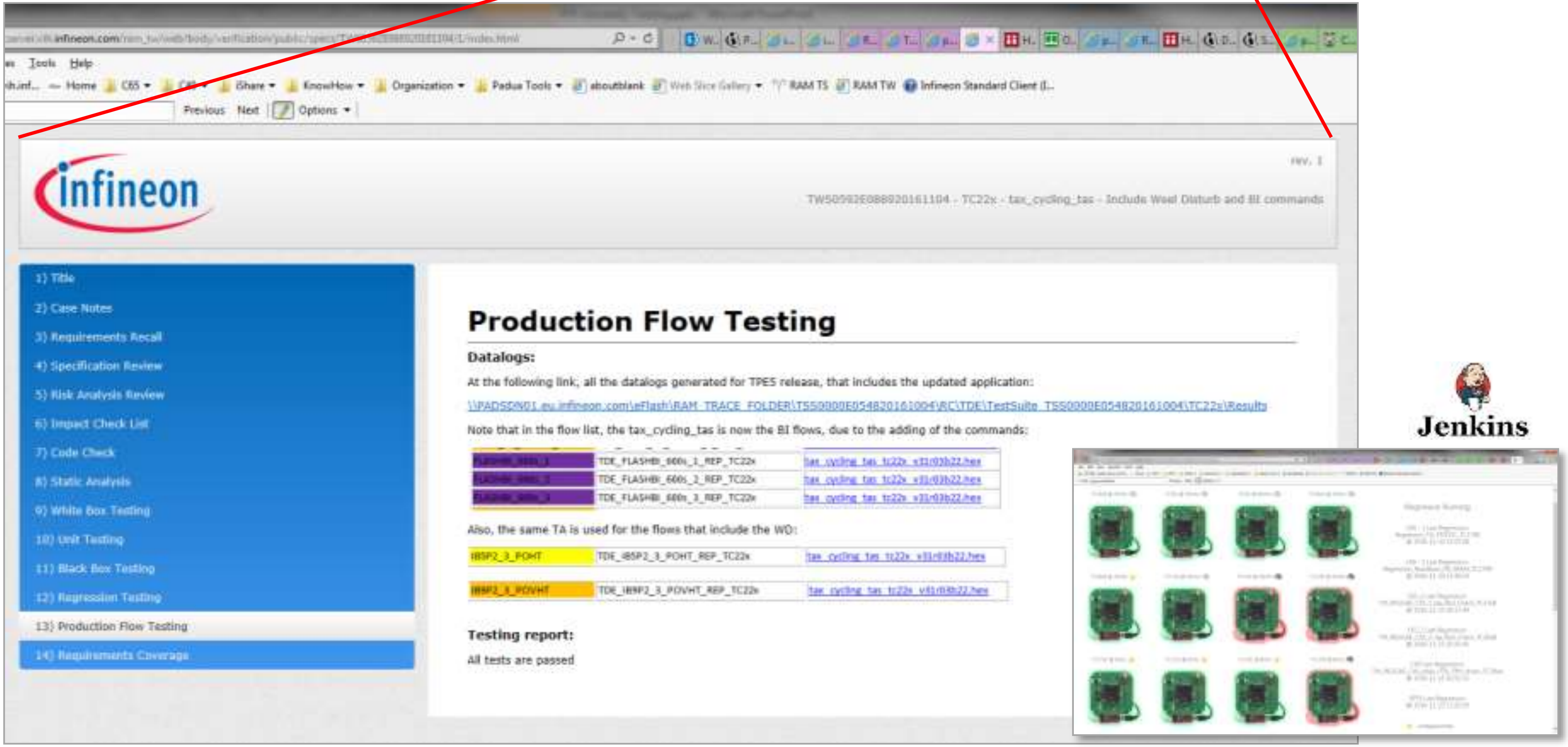
Output Work Products
08-52 Test plan [Outcome 1, 2, 6]
08-50 Test specification [Outcome 2]
13-50 Test result [Outcome 3, 4]
13-22 Traceability record [Outcome 5]

Phase: Validation – Example

- **Validation takes care about:**
  - **Unit** Test
  - **Nightly** build and **Static** check
  - **Functional, Flow** tests
  - **Regression** tests



Verification:	Massimo Giltrelli (expected: Massimo Giltrelli) - 2016-11-22 11:06:02
V and V Report:	verification/public/specs/TWS0592E088920161104/1/i...
Verification link Final:	Def Verification Rev. 1
Verification link Editor:	...
Functional validation:	Not available...
Regression SVN:	N.A.
Regression Overall:	Not available...



TWS0592E088920161104 - TC22x - tax_cycling_tas - Include Weed Disturb and BI commands

1) Title
2) Case Notes
3) Requirements Recall
4) Specification Review
5) Risk Analysis Review
6) Impact Check List
7) Code Check
8) Static Analysis
9) White Box Testing
10) Unit Testing
11) Black Box Testing
12) Regression Testing
13) Production Flow Testing
14) Requirements Coverage

Production Flow Testing

Datalogs:

At the following link, all the datalogs generated for TPES release, that includes the updated application:

\\PADSDN01.eu.infineon.com\eFlash\RAM_TRACE_FOLDER\TS50000E054820161004\RC\TDE\TestSuite_TS50000E054820161004\TC22x\Results

Note that in the flow list, the tax_cycling_tas is now the BI flows, due to the adding of the commands:

[illegible]	TDE_FLASHBI_600s_1_REP_TC22x	tax_cycling_tas_tc22x_v31r03b22.hex
[illegible]	TDE_FLASHBI_600s_2_REP_TC22x	tax_cycling_tas_tc22x_v31r03b22.hex
[illegible]	TDE_FLASHBI_600s_3_REP_TC22x	tax_cycling_tas_tc22x_v31r03b22.hex

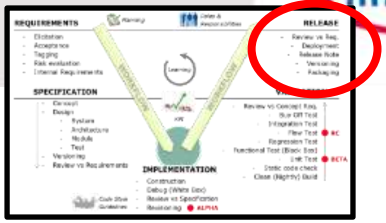
Also, the same TA is used for the flows that include the WD:

IB5P2_3_POHT	TDE_IB5P2_3_POHT_REP_TC22x	tax_cycling_tas_tc22x_v31r03b22.hex
IB9P2_3_POVHT	TDE_IB9P2_3_POVHT_REP_TC22x	tax_cycling_tas_tc22x_v31r03b22.hex

Testing report:

All tests are passed

Phase: Release



- **Description**: (glossary link)
  - *All of the activities that make a software system **available** for use such as Release, Install, Activation, Update, Version tracking, Retire.*

- **ASPICE reference:**
  - SPL.2 Product release: The purpose of Product release process is to **control** the release of a product to the intended **customer**.

Output Work Products
06-01 Customer manual [Outcome 3]
08-16 Release plan [Outcome 1, 3]
11-03 Product release information [Outcome 1, 3, 4, 6]
11-04 Product release package [Outcome 2, 3, 6]
11-07 Temporary solution [Outcome 6]
13-06 Delivery record [Outcome 6,7]
13-13 Product release approval record [Outcome 5]
15-03 Configuration status report [Outcome 2]
18-06 Product release criteria [Outcome 5, 7]

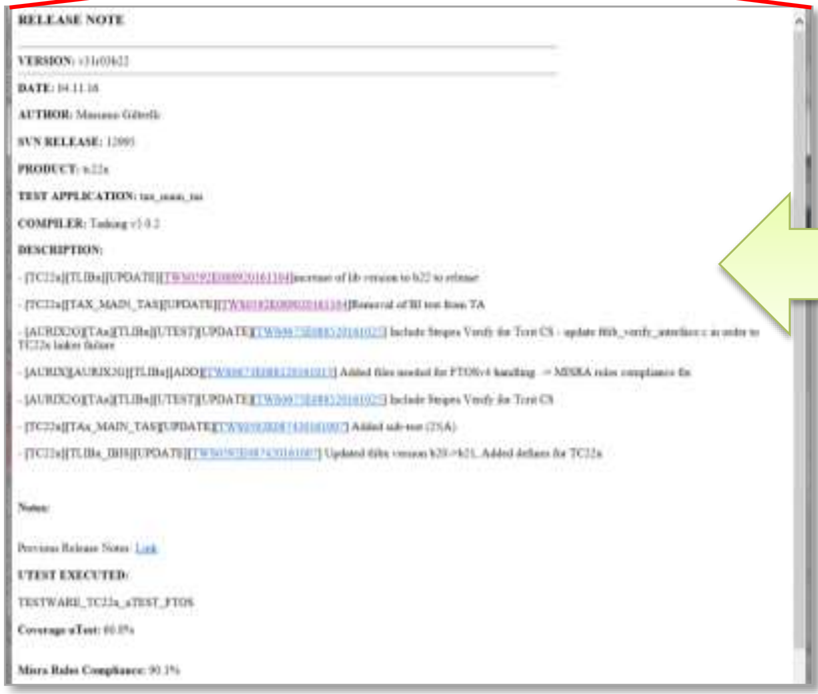
- **Benefits:**
  - Determined **contents** of the product release
  - Easy release **documentation** production
  - Release can be **approved** by process
  - **Approval** by customer is obtained

Phase: Release – Example

- **Release provides:**
  - TW **application** binary
  - **Documentation**
  - Release **notes**
  - **References** (e.g. uCode, DEVCFG)
  - **Validation** reports

Release: Massimo Giltrelli (expected: Massimo Giltrelli) - 2016-11

Version: v31r03b22



RELEASE NOTE

VERSION: v31r03b22

DATE: 04.11.16

AUTHOR: Massimo Giltrelli

SVN RELEASE: 12095

PRODUCT: tc22x

TEST APPLICATION: tax_main_tas

COMPILER: Tasking v5.0.2

DESCRIPTION:

- [TC22x][TLIBs][UPDATE][[illegible]]increase of lib version to b22 to release
- [TC22x][TAX_MAIN_TAS][UPDATE][[illegible]]Removal of BI test from TA
- [AURIX2G][TAs][TLIBs][UTEST][UPDATE][[illegible]] Include Stepper Verify for Test CS - update tlib_verify_interface.c in order to TC22x linker failure
- [AURIX][AURIX2G][TLIBs][ADD][[illegible]] Added files needed for FTOSv4 handling -> MISRA rules compliance fix
- [AURIX2G][TAs][TLIBs][UTEST][UPDATE][[illegible]] Include Stepper Verify for Test CS
- [TC22x][TAx_MAIN_TAS][UPDATE][[illegible]] Added sub-test (25A)
- [TC22x][TLIBs_IBIS][UPDATE][[illegible]] Updated tlibs version b20->b21. Added defines for TC22x

Notes:

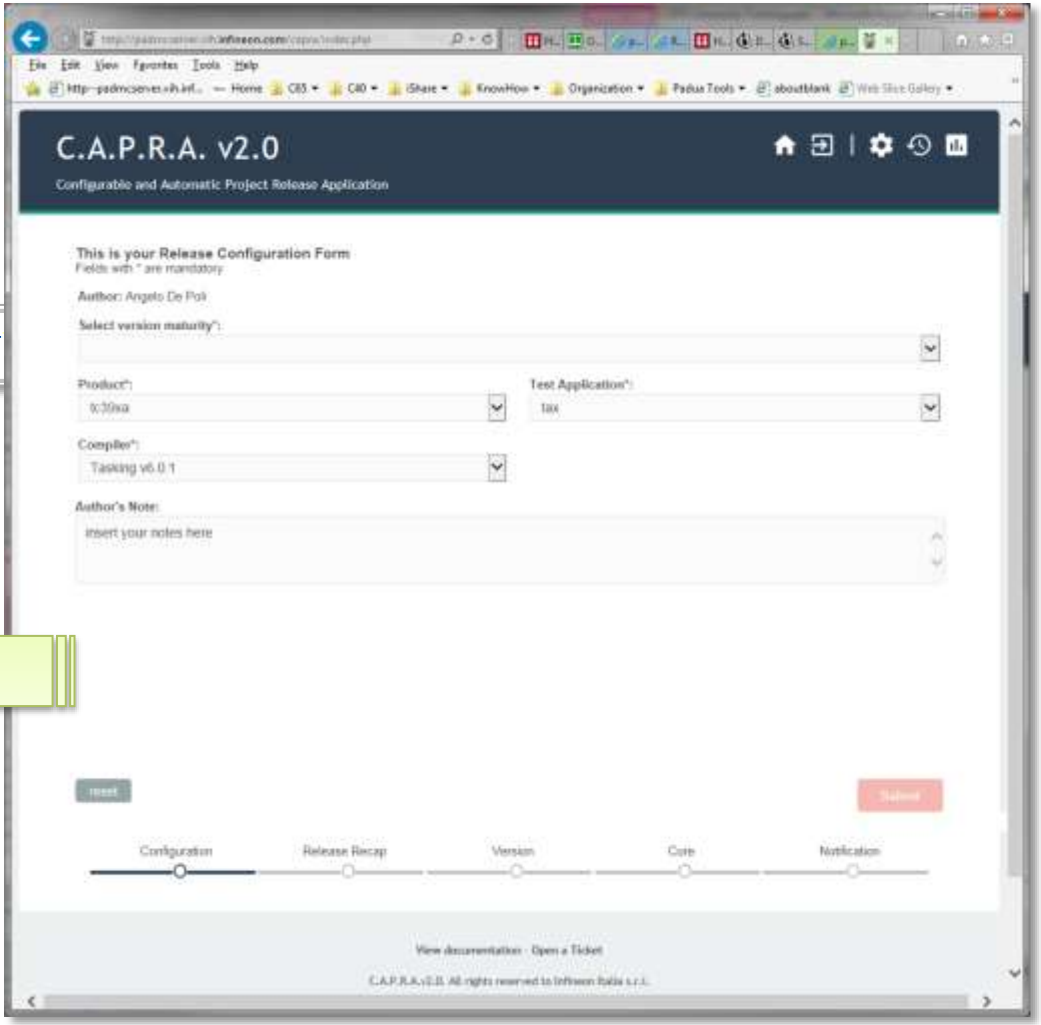
Previous Release Notes: Link

UTEST EXECUTED:

TESTWARE_TC22x_uTEST_FTOS

Coverage uTest: 60.0%

Misra Rules Compliance: 90.1%



C.A.P.R.A. v2.0

Configurable and Automatic Project Release Application

This is your Release Configuration Form
Fields with * are mandatory

Author: Angelo De Poli

Select version maturity*:

Product*: tc39xa

Test Application*: tax

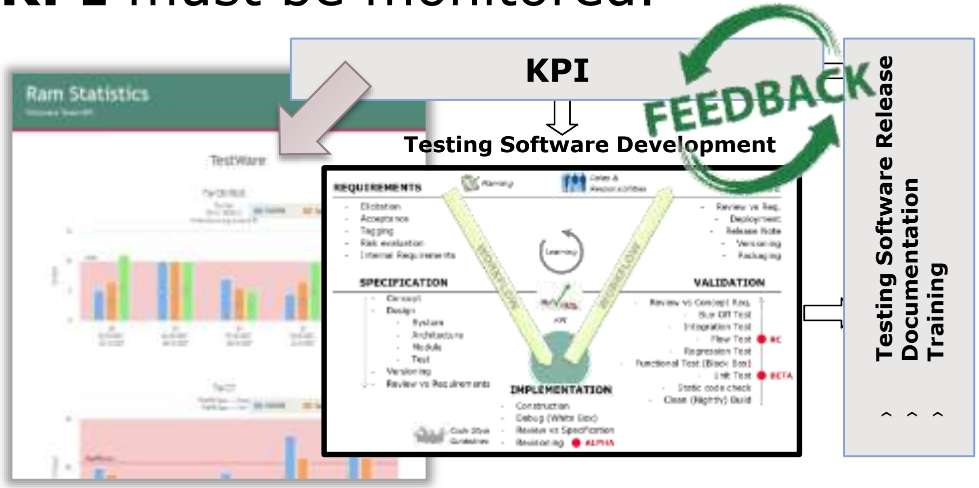
Compiler*: Tasking v6.0.1

Author's Note: insert your notes here

reset
Submit
Configuration
Release Recap
Version
Core
Notification
View documentation · Open a Ticket
C.A.P.R.A.v2.0. All rights reserved to Infineon Italia s.r.l.

Close the loop: KPI and Feedback

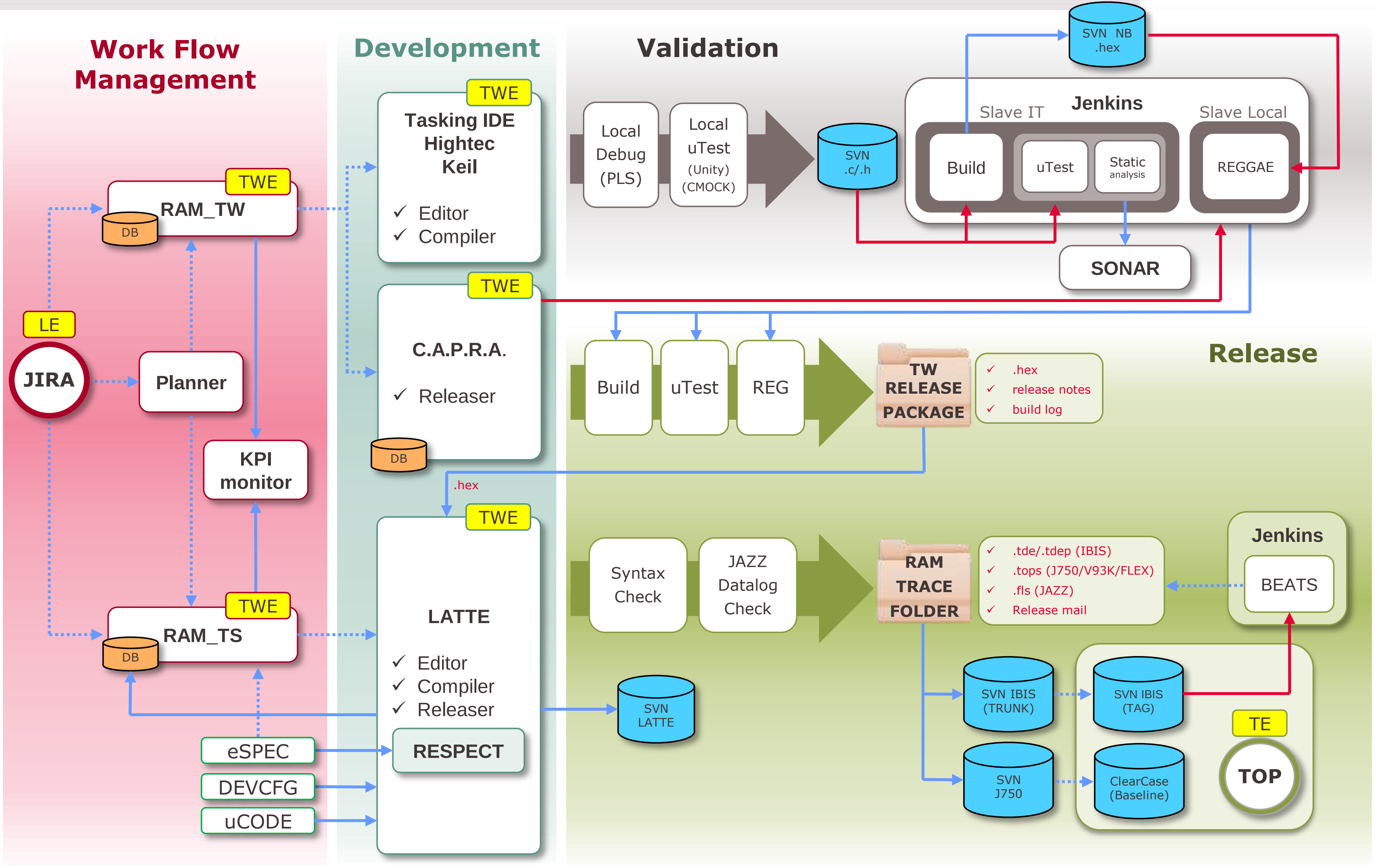
- To keep quality in focus some **KPI** must be monitored.
  - For Example:
    - BUG over Change Requests
    - Total Cycle Time
    - Cycle Time by Phase
    - Rework rate
    - …
  - **Action Items** must be triggered when trends are going over specified limits



- Further feedbacks loops can come with activities like:
  - **5 Why**
  - **Lesson Learned**

eFlash TW and TP Development WorkFlow – Interfaces

Automatic
Manual
Trigger
Flow

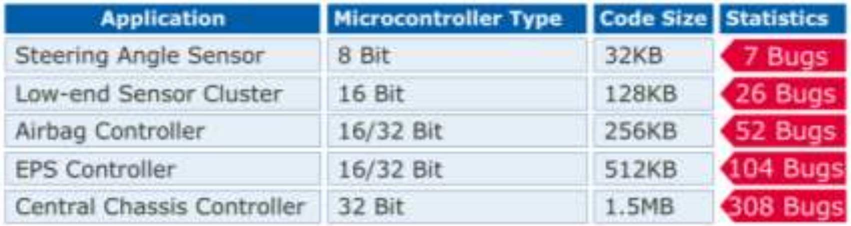


Software development for automotive embedded non volatile memories testing
Agenda

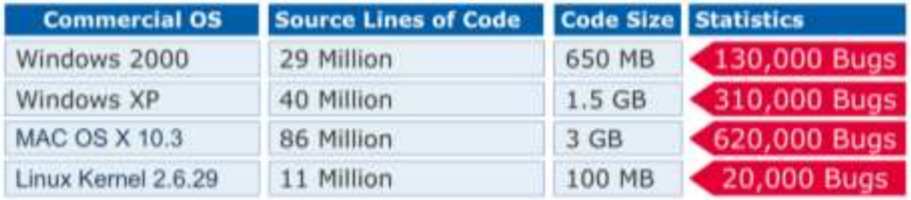
1. Infineon MicroController: a SoC for automotive
2. SoC Quality and Testing
3. Software for Testing: Complexity
4. Software for Testing Development: why a process
5. Software for Testing: Strategies for Quality assurance
6. Software for Testing Development: phases
7. Conclusions and Questions

Conclusions

- From statistics:
  - Code Quality without uTest: 1 bug every 20 LoC
  - Code Quality with uTest: 1 bug every 1000 LoC / 5 KB



Application	Microcontroller Type	Code Size	Statistics
Steering Angle Sensor	8 Bit	32KB	7 Bugs
Low-end Sensor Cluster	16 Bit	128KB	26 Bugs
Airbag Controller	16/32 Bit	256KB	52 Bugs
EPS Controller	16/32 Bit	512KB	104 Bugs
Central Chassis Controller	32 Bit	1.5MB	308 Bugs



Commercial OS	Source Lines of Code	Code Size	Statistics
Windows 2000	29 Million	650 MB	130,000 Bugs
Windows XP	40 Million	1.5 GB	310,000 Bugs
MAC OS X 10.3	86 Million	3 GB	620,000 Bugs
Linux Kernel 2.6.29	11 Million	100 MB	20,000 Bugs

- Our experience :
  - **Quality** (bugged Change Requests):
    - **Without Clean Code/uTest/Regression**: **60%**
    - **With Regression**: **3%**
    - **With Clean Code & uTest**: **0,4%**
    - *Expected With Clean Code + uTest + Regression: 0,2%*
  - **Effort**:
    - **With Regression**: **+30%**
    - **With Clean Code + uTest**: **+20%**

But…

* PADUA experience

- **How long** does it take?
  - **~7 years*** to **define** a **process**
  - **~5 years*** to **refine** to an audit compliant level
    - 3 years for establishing engineers mindset
- What **competences** are needed?
  - NOT pure HW. Not pure SW. But **HW+SW** competence mix required at each engineer.
- Is it for **everyone**?
  - Right people and **ecosystem** are mandatory: ad-hoc **HW+SW** competences mix combined to **integrated toolchain** (RDDF, workflow manager, Continuous Delivery and Integration and in-system nightly regression.

Part of your life. Part of tomorrow.

Software Development Methodology
Benefits for Engineers

- **FAQ**:

  - *Is the workflow wasting my time with bureaucracy?*

- No, Workflow in the **mid/long term saves time**:
  - **information** is always **available** for researches
  - it's easier to **share information** with engineers, even newbies
  - tasks can be **shared within team** quickly (cooperative activity)

  - *What happen if I don't follow the workflow?*

- By **not following the workflow** engineers lose:
  - KPI indication to improve working **performance** and **quality**
  - **Traceability** of what has been done
  - **Information sharing** with newbies and colleagues

  - *Which information should I put?*

- By using a "**Poka Yoke**" concept information set is predefined and proposed/requested to engineers. It should be not possible to miss some information.

  - *I don't see any advantage, why should I follow the process?*

- **Many advantages** for engineers are coming with a process:
  - They can **go to vacation** without worries since tasks can **handover** to team members
  - They are **not guilty**, is the process that's weak
  - They **don't** have to **improvise**
  - Their work is **predictable**: less surprises, less worries

Software Development Methodology
Benefits for Managers

› **FAQ**:

- *Is the workflow wasting my team's time? How much does it cost?*
- *How can I trace who's not following the workflow?*
- *Which information could I get from workflow?*
- *How can I convince the team to use the workflow? How long do I need?*

› No, Workflow in the **mid/long term saves time**:

- In case of trouble, **data** are easily **available** for learning loops
- Easier to **introduce newbies**
- Project can be supported with faster **handover**, more flexibility

› When **engineers don't follow** the workflow:

- Dashboards are showing **empty** tasks **queues**
- **No KPI** updates
- **Missing** cross **links** between CR and Deliverables

› Several **KPI** can be **monitored**:

- CT
- FTR (respins)
- Quality rate (Bugs over CR)

› Team must be introduced **step by step** to process:

- Use a **tailoring** approach
- Provide **automation** to avoid "no brain" activities
- **Monitor** execution together