VectorCAST Presentation

AdaEurope 2017

Advanced safety strategies for DO178C certification

Massimo Bombino, MSCE

Software Quality Overview

VECTOR software

QUALITY HAZARDS IN AVIONICS INDUSTRY

1. Software is blamed for more major business problems than any other man-made product.

2. Poor software quality has become one of the most expensive topics in human history: > $150 billion per year in U.S.; > $500 billion per year world wide.

3. Projects cancelled due to poor quality >15% more costly than successful projects of the same size and type.

4. Software executives, managers, and technical personnel are regarded by many CEO's as a painful necessity rather than top professionals.

5. Improving software quality is a key topic for all industries, and it's mandatory for *AVIONICS*

TESTING: A REAL SOLUTION?

TESTING REQUIRES A LOT MONEY AND TIME:

- Significant effort into the software lifecycle **(up to 70-80%)**
- DO-178 Low-level testing is much more expensive than developing **(2-3 times bigger!)**
- Code changes daily, "Continuous Integration" is a myth **(up to 100 times than your computational power)**

TRADITIONAL TESTING COULD BE INEFFECTIVE:

- Best organizations have 1-5 bugs per KLOC, 25% Critical (**1M LOC has >1000 bugs, > 250 critical bugs)**
- Could you measure the effectiveness of your testing? **(Coverage is a necessity)**

TESTING REQUIRES GREAT SKILLS:

- Typical testers are highly skilled and expensive, often developers **(average testing costs are up to 2€ x LOC)**
- Testing for engineers is a **boring** activity **(> 50% of testers are unsatisfied and plan new activity or job)**

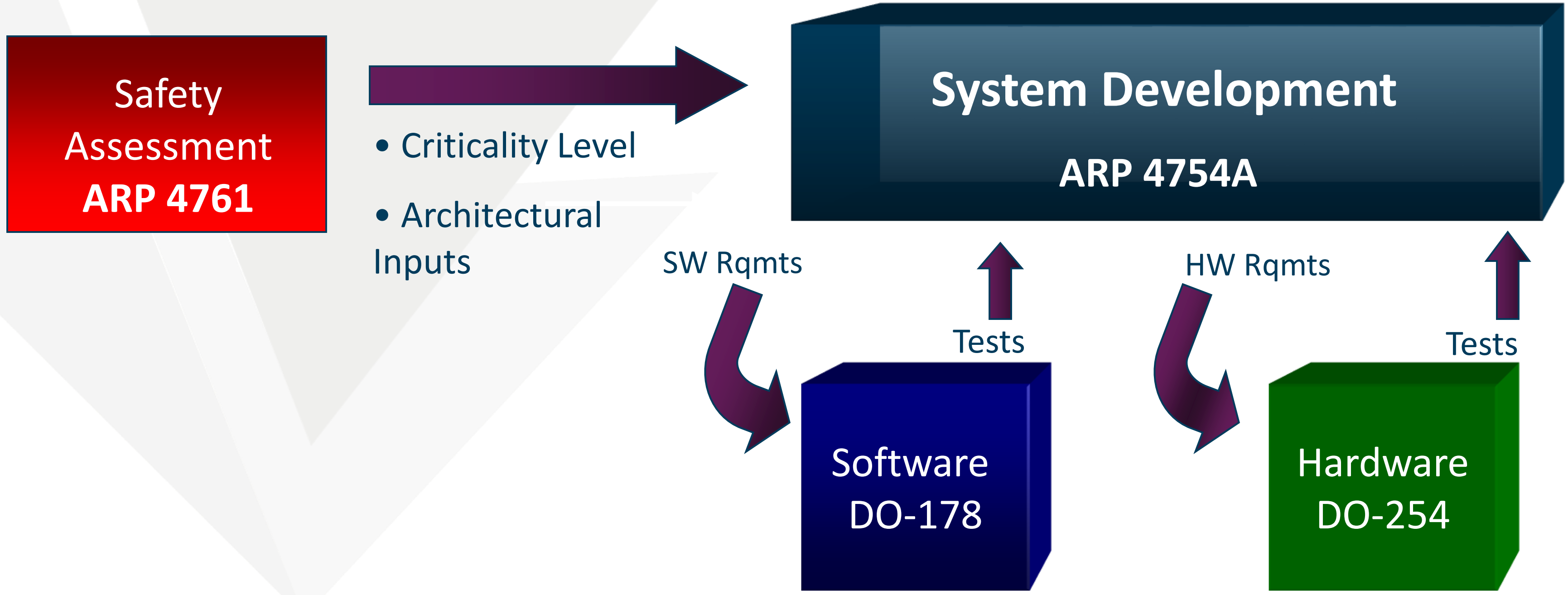
NOT TESTING IS A BIGGER COST:

- Do you quantify your risk of not testing? **(Your Technological Debt "iceberg" is >10 times than expected)**
- Your **final customers** are the most expensive "testers" **(> 1000 bigger than early detection)**

DO-178 Verification Strategy Overview

VECTOR software

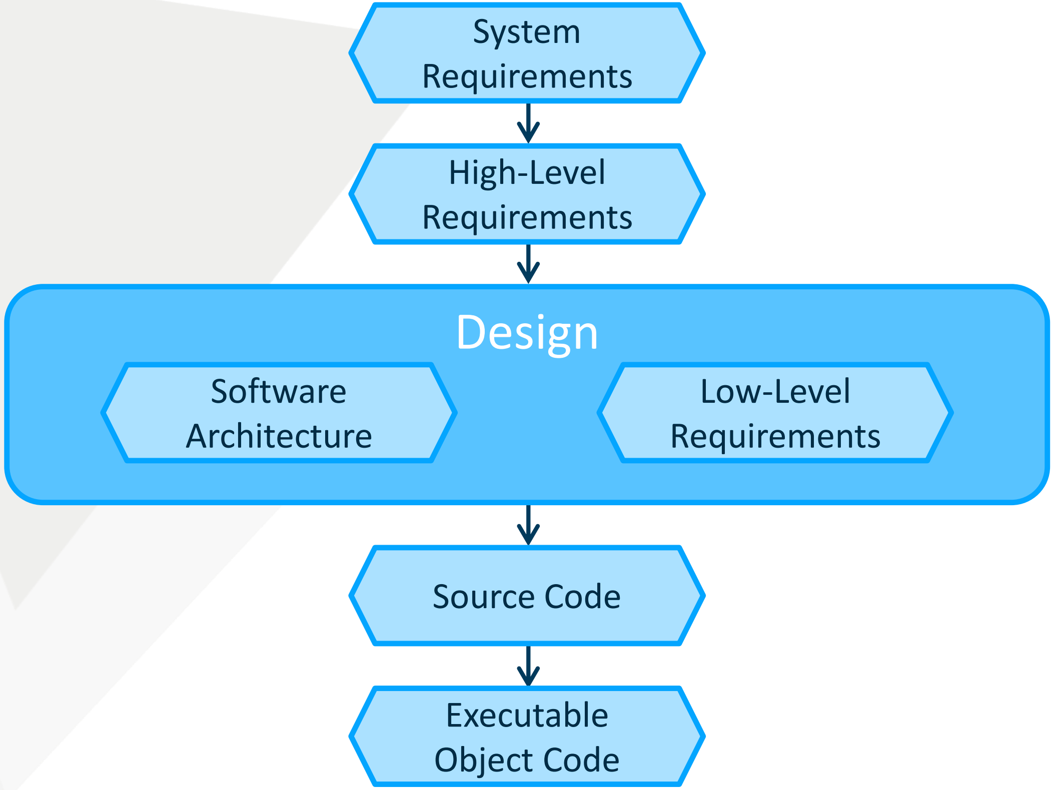
Avionics Development Ecosystem

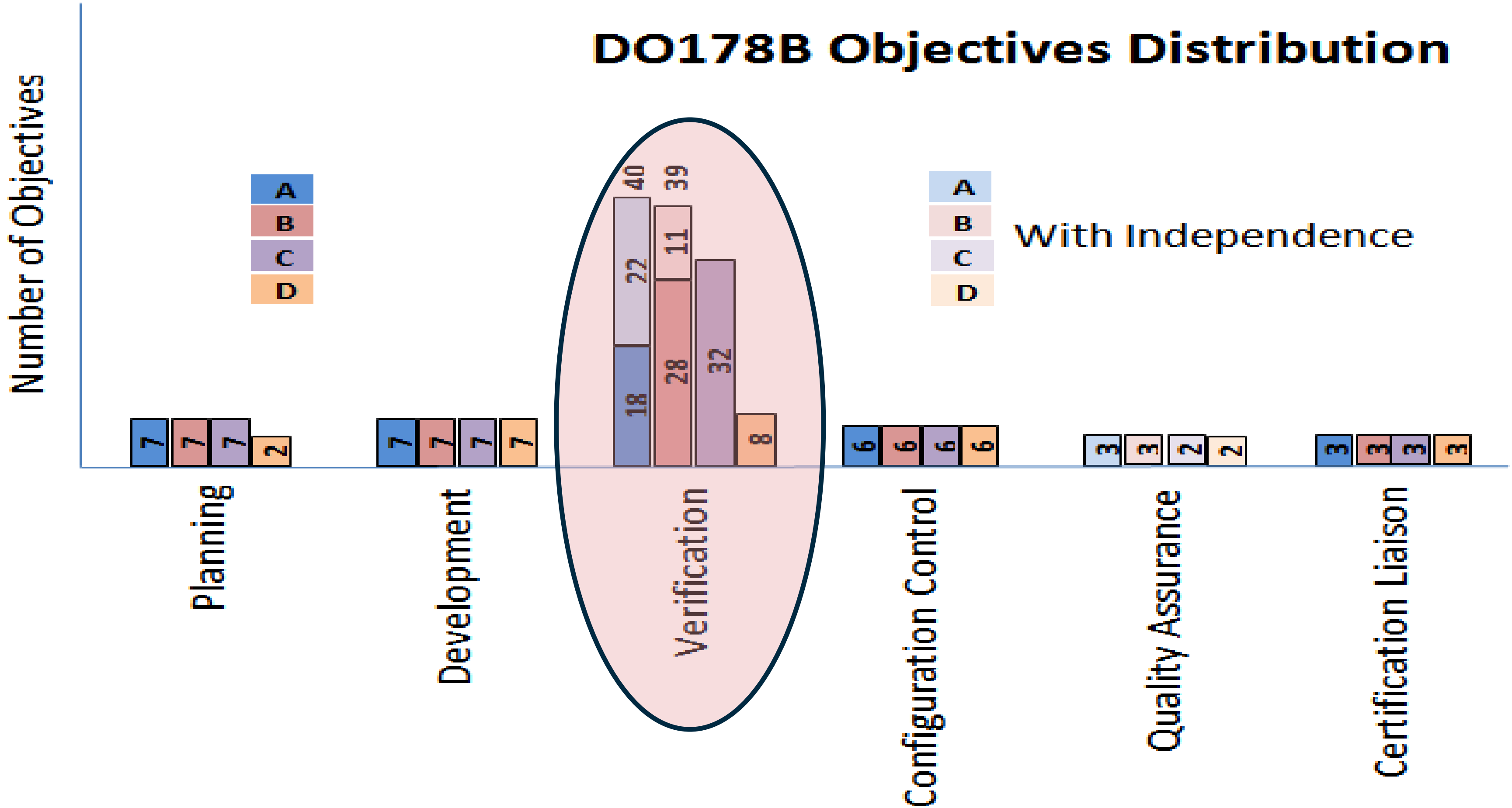


Key Principle: DO-178B Objectives by Level

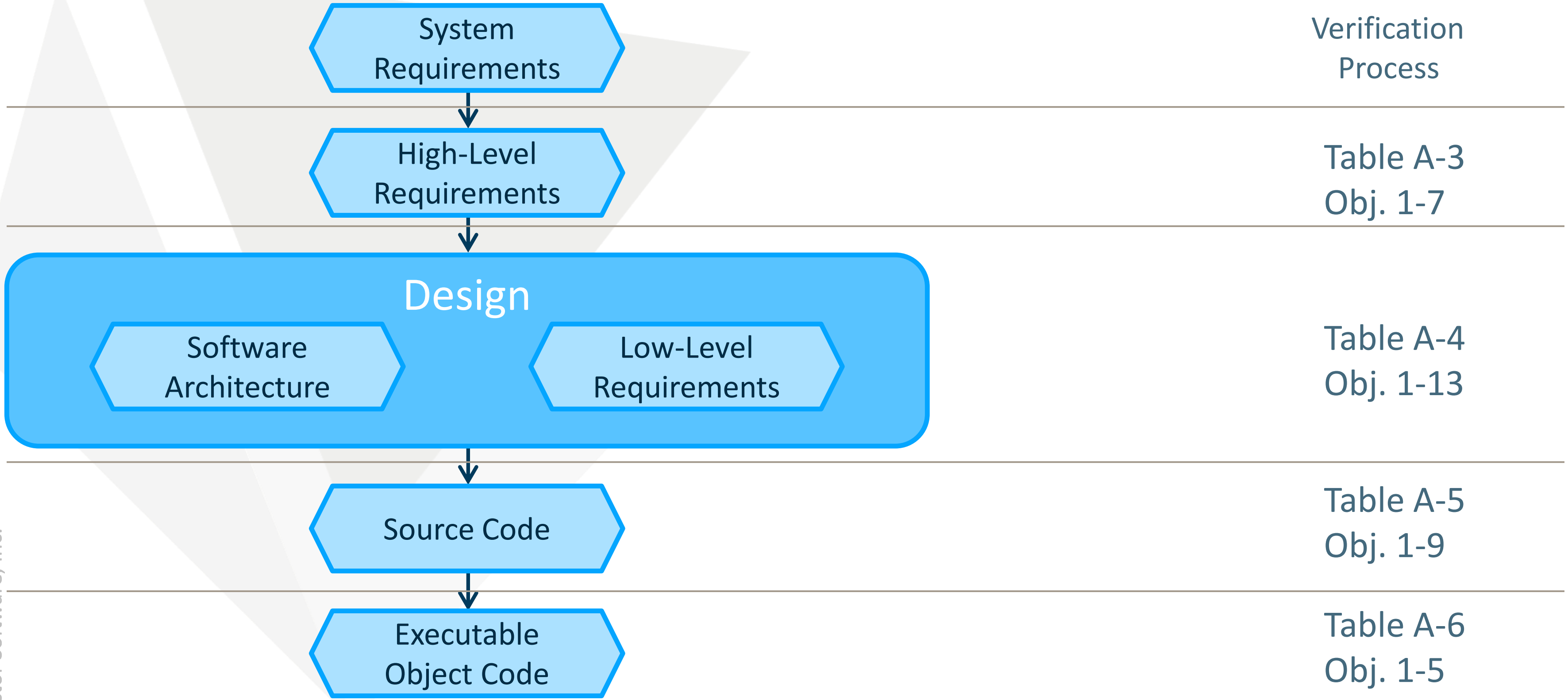
- **Level A: 71 Objectives** *(30with independence)*
- **Level B: 69 Objectives** *(18 with independence)*
- **Level C: 62 Objectives** *(5 with independence)*
- **Level D: 26 Objectives** *(2 with independence)*
- **Level E: No Objectives** ***(just prove you are Level E!)***

DO-178C SW production lifecycle

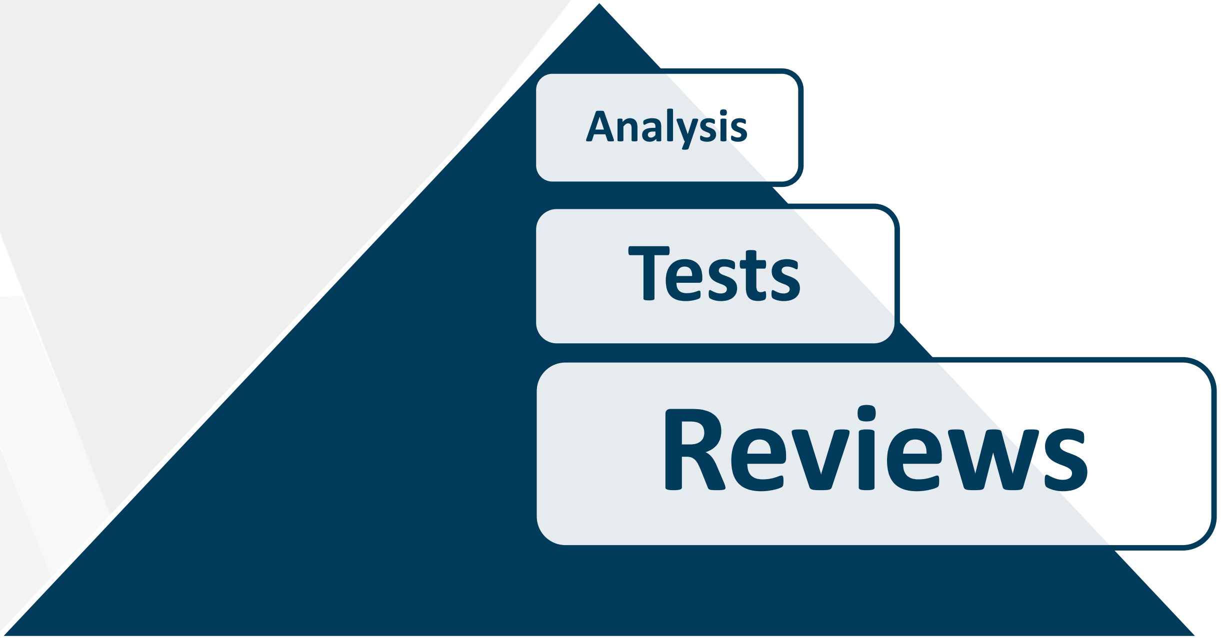



DO178B Objectives Distribution
Number of Objectives
A
B
C
D
With Independence
A
B
C
D
7 7 7 2
7 7 7 7
18 22 40
28 11 39
32
8
6 6 6 6
3 3 2 2
3 3 3 3
Planning
Development
Verification
Configuration Control
Quality Assurance
Certification Liaison

DO-178 Objectives per phase



DO-178C: Verification Pyramid Foundation

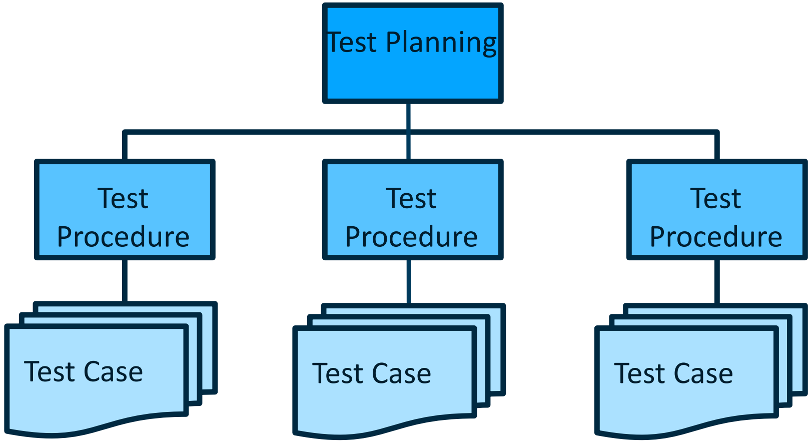


“The Verification Equation”

Verification = Reviews + Tests & Analysis

Test Implementation - Developing the Test

- > Organization
- > Planning
- > Test
  - Procedures
  - Cases
  - Strategies to test or analyze requirements and achieve desired code coverage



Requirement-Based Testing: QUIZ!

```
int F (int P)
{
  if (P > 0)
    return P+1;
  else
    return P-4;
}
```

P = 3

EXPECTED/CORRECT OUTPUT OF FUNCTION F ?

a) 4
b) -1
c) NOT ENOUGH INFO

Test Cases

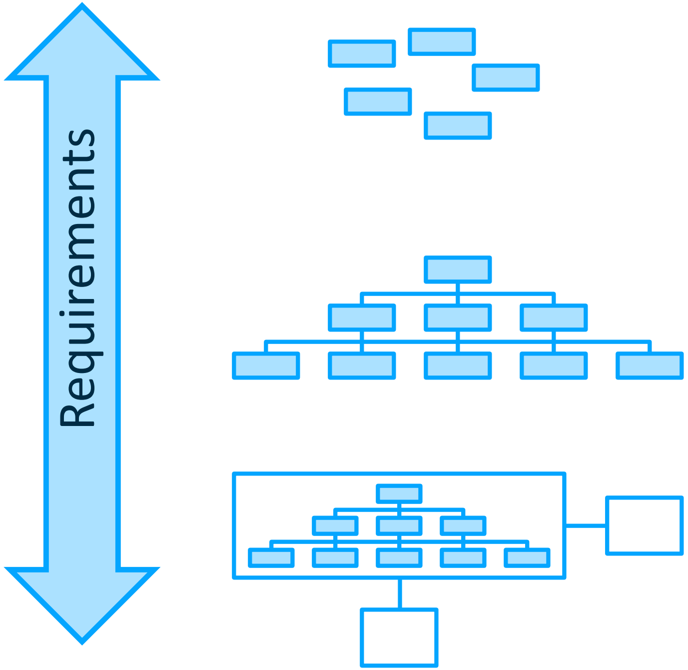
- Normal Range Tests
  - Normal conditions and inputs
    - *In range inputs, normal events interrupts, normal state transitions, normal logic processing*
- Robustness Tests
  - Abnormal conditions and inputs
    - *Out of range inputs, unexpected interrupts and state transitions, exception handling, system initialization*
- Performance Tests/Analyses (can be part of Robustness)

Test Environments

- Software testing activities that may be used to achieve the DO-178 software testing objectives:
  - **Low-level testing**: To verify the implementation of low-level requirements and of the basic functionalities of your system
  - **Software integration testing**: To verify the interrelationships between software requirements and components and to verify the implementation of the software requirements and software components within the software architecture.
  - **Hardware/software integration testing**: To verify correct operation of the software in the target computer environment.

Test Case Levels

- Low
  - Low Level Requirements
  - Unit/Module Testing
- Intermediate
  - Software Integration Tests
  - Test Simulators
- High
  - Hardware Software Integration Tests
  - Target Environment



Code Coverage via Low Level Testing

- Advantages
  - Tests detailed (design) requirements
  - Tests can be done independently in parallel
  - Does not require expensive test equipment
  - Easier to target particular code areas
- Disadvantages
  - More testing required
  - Only tests an isolated part of code
  - Tests can be contrived

Code Coverage via High Level Testing

- Advantages
  - Tests software functional requirements
  - More coverage per test
  - More realistic, useful tests
- Disadvantages
  - Tests harder to setup
  - Some classes of errors harder to target
  - Tests results require more analysis
  - Need tools to determine structural coverage
  - Harder to plan up front what coverage provided

Code Coverage via Analysis

- Advantages
  - May be less expensive to setup
  - Does not require tools or code
  - Instrumentation
- Disadvantages
  - More labor intensive
  - Needs to be repeated each time code changes/tests rerun
  - Can be less rigorous (error-prone and tedious process)
  - Hard to prove, & few people do it right …

Error Detection Objectives

Software Integration Testing

- Incorrect initialization of variables
- Parameter passing errors
- (Global) Data Corruption
- Inadequate Numerical Resolution
- Incorrect Sequencing of Events and Operations

Low Level Testing

- Algorithm Failures
- Incorrect Loop Operations
- Incorrect Logic Decisions
- Failure to Process Correct Input combinations
- Incorrect Response to bad Input data
- Incorrect Exception Handling
- Incorrect Computation Sequence
- Inadequate Algorithm Precision, Accuracy, Performance

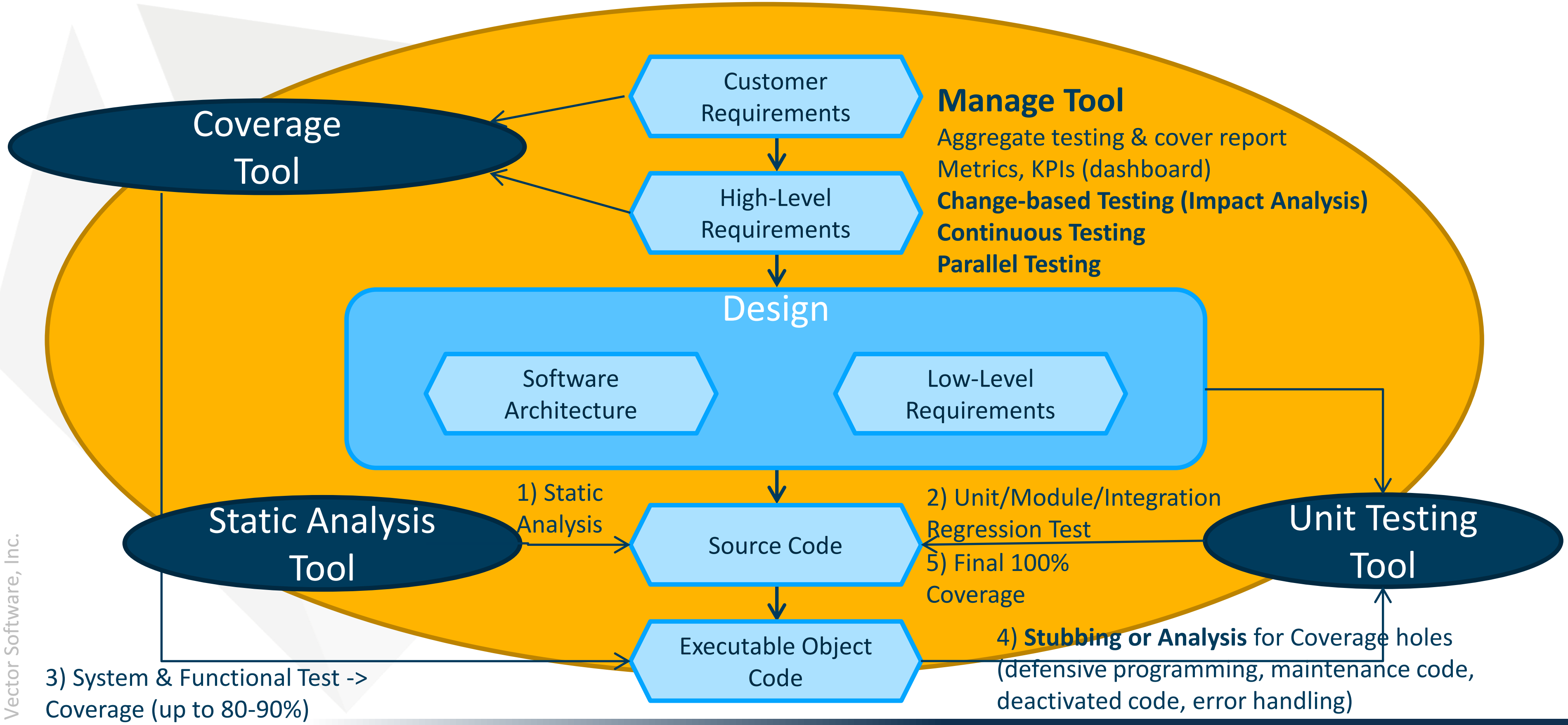
Hardware/Software Integration Tests

- Incorrect Interrupt Handling
- Miss Timing Requirements
- Hardware transient Errors
- Resource Contention
- BIT Detection Errors
- Bad feedback Loops
- Incorrect Device Control
- Stack Overflow
- Incorrect Load Version Verification
- Software Partitioning Violations

DO-178 Advanced Verification & Testing

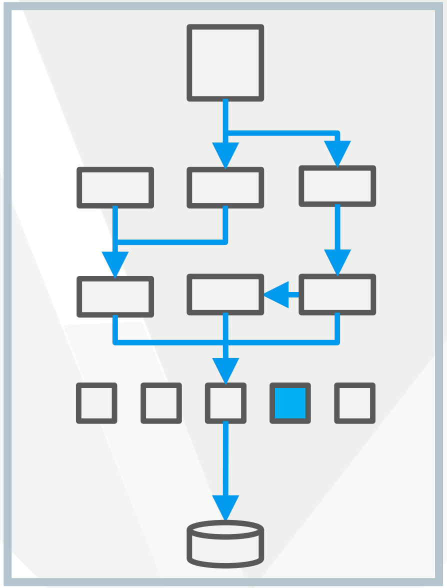
VECTOR software

How to satisfy DO-178 with best testing strategy and tools



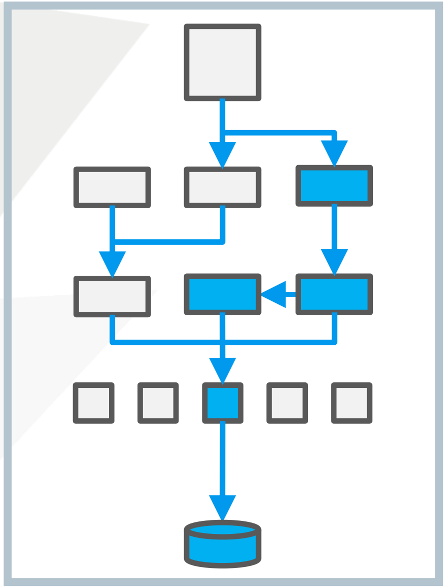
C, C++, Ada Embedded Integration Testing

Unit Testing



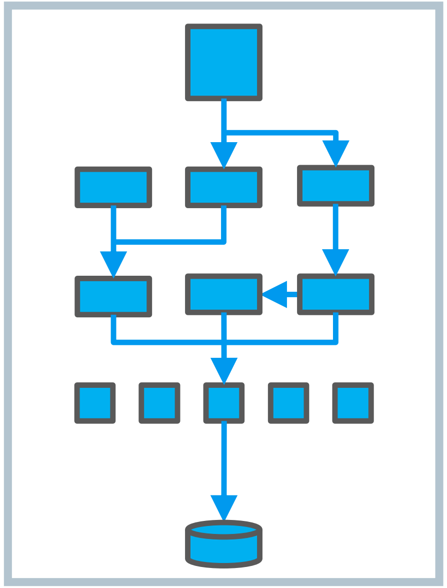
Individual units or modules are tested. It involves testing of source code by developers.

Integration Testing



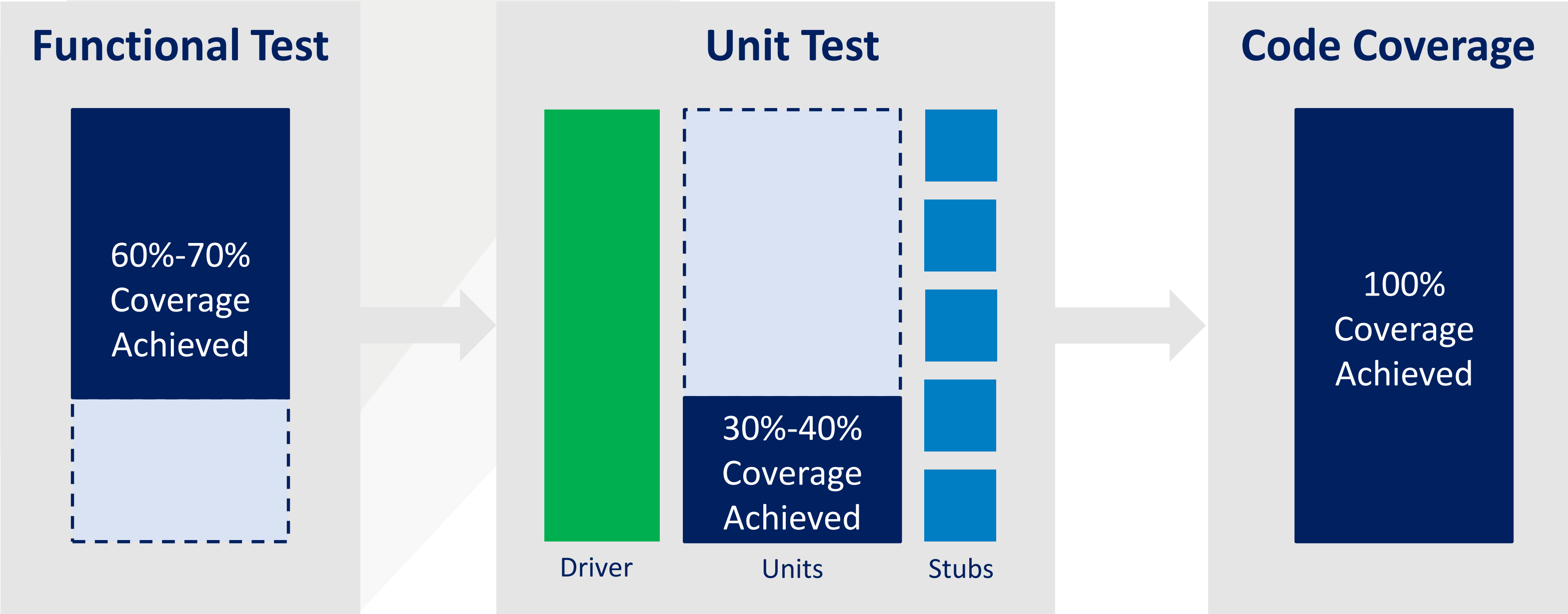
Individual modules are grouped together and tested. The purpose is to determine that modules are working as expected once they are integrated.

System Testing

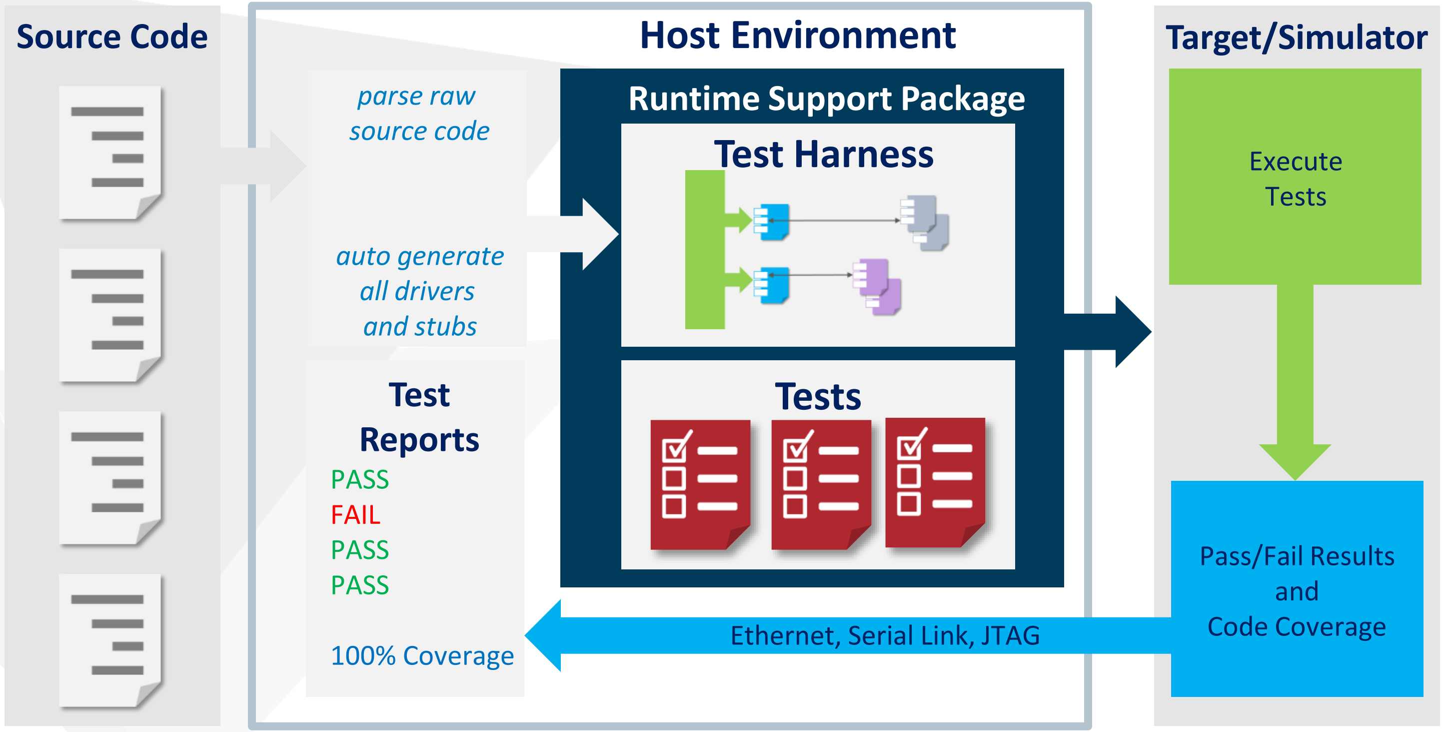


Testing is performed on the whole system by checking whether the system or application meets the requirement specification document.

Code Coverage



On-Target Embedded Unit Testing



Change Based Testing

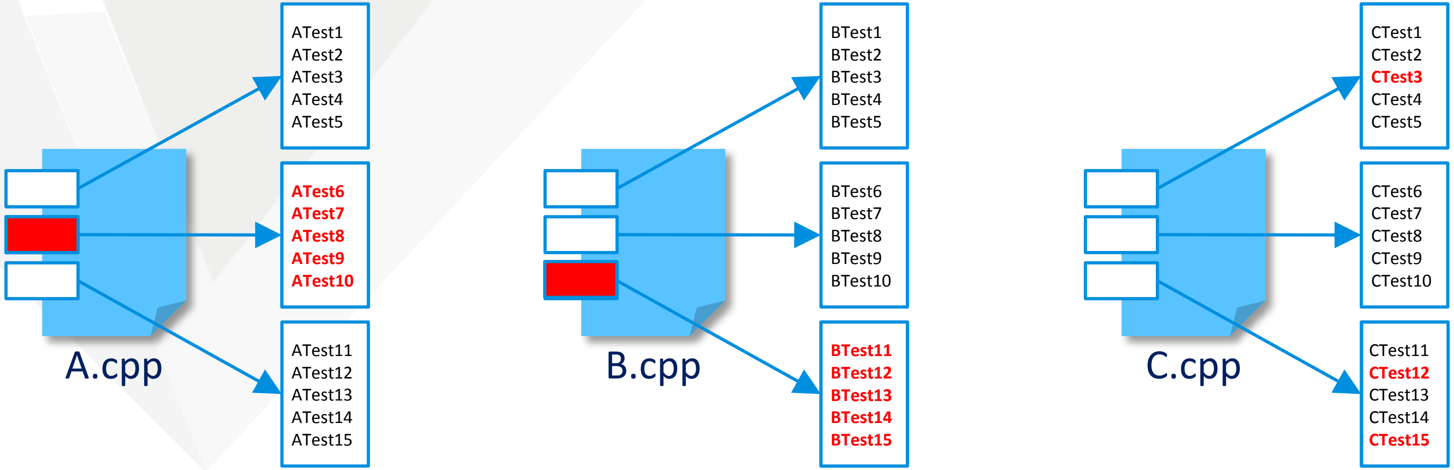
> Comparing changes is key to assessing risk

> Determine if a code change affects other parts of the system

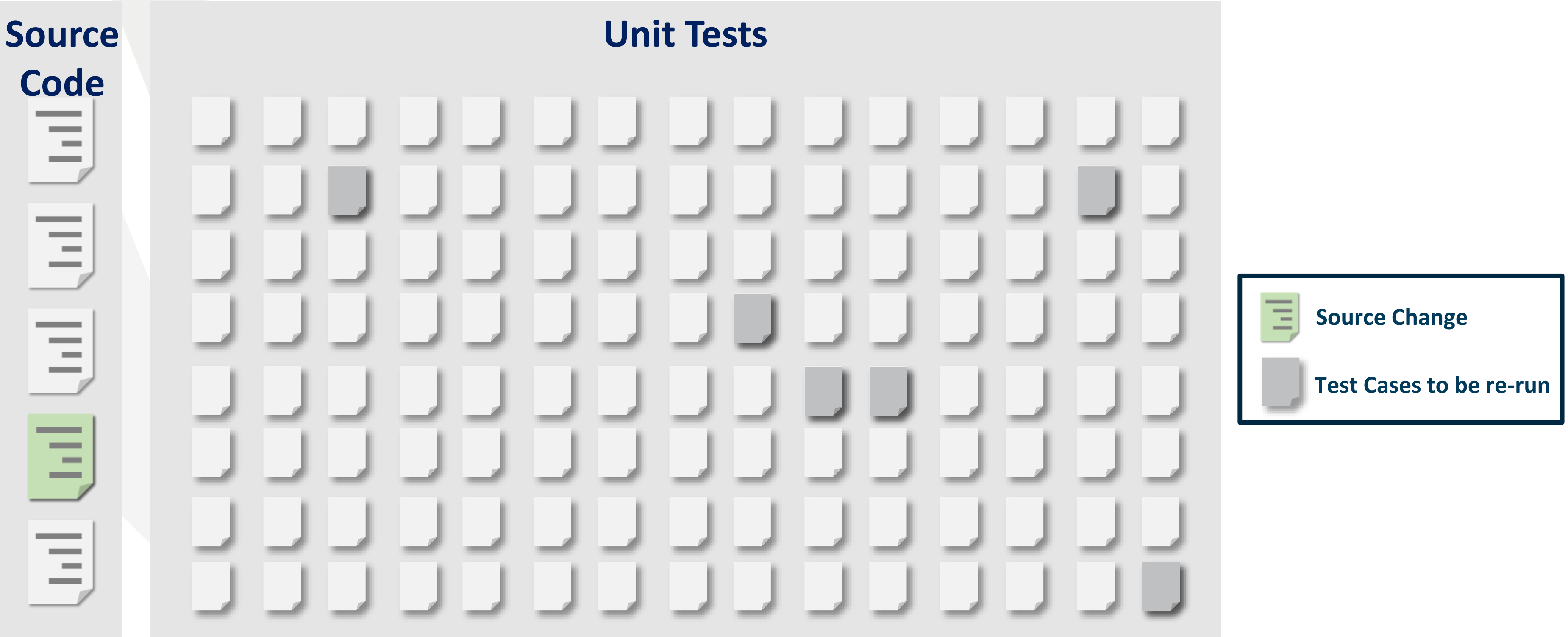
> Prioritize tests based on risk, change, and criticality of modules

  > Change-based testing permits prioritized tests of modified modules

  > Regression testing ensures changes do not introduce new faults

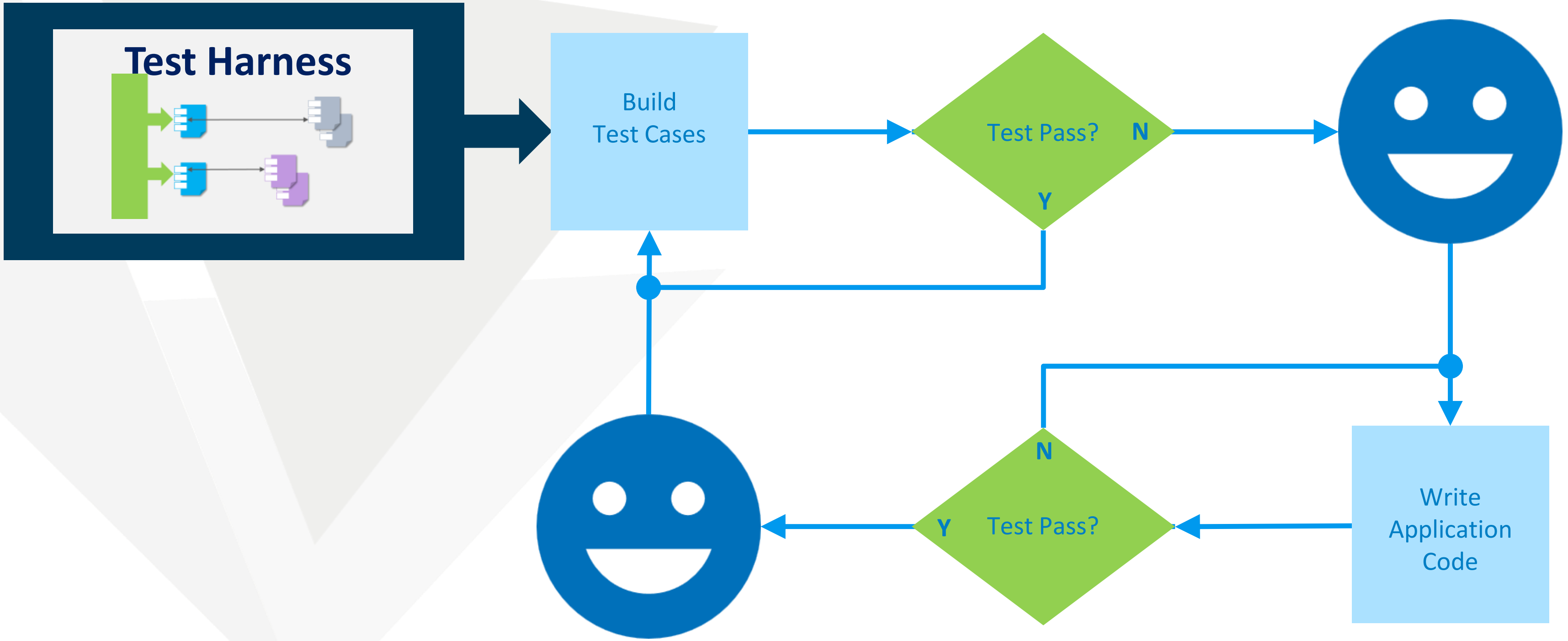


Change Based Testing – “Test Less”, “Fail Faster”



Automatic detection of which test cases have been affected by a source change

Continuous Testing Agile / Test Driven Development

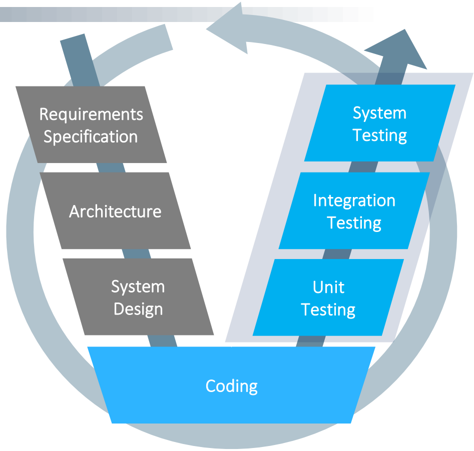


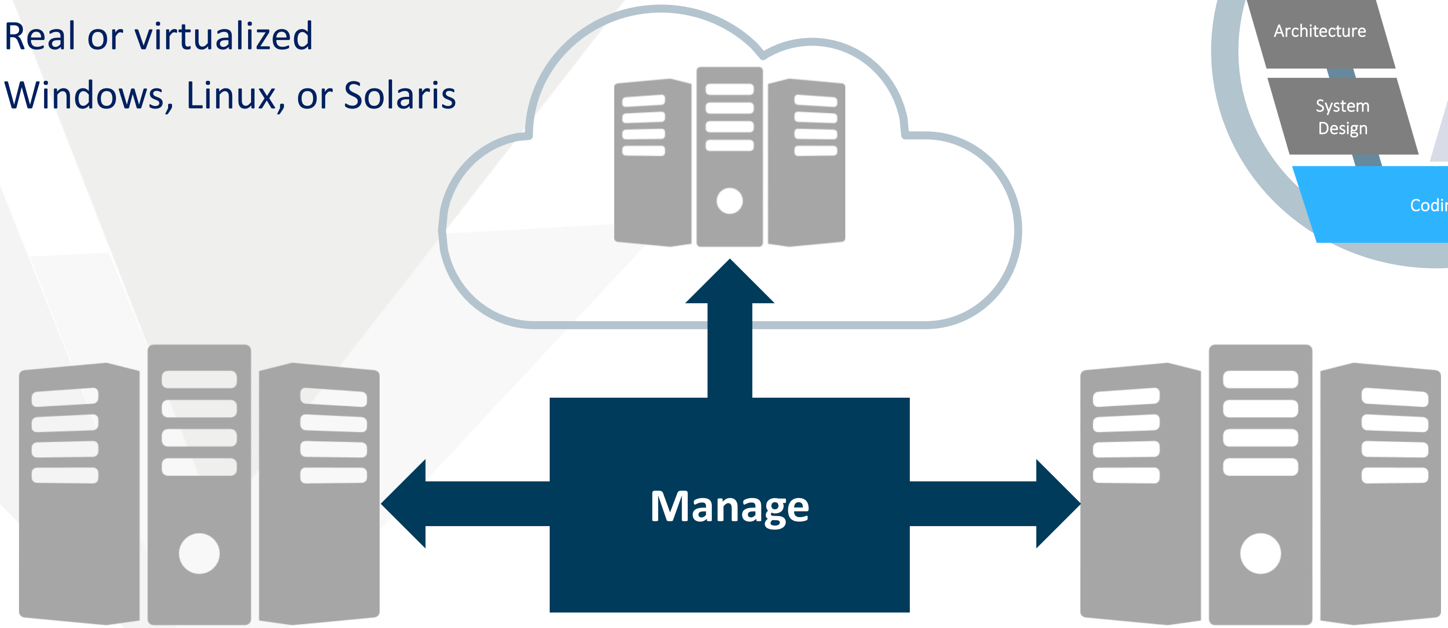
Parallel Testing

- Jenkins
  - Jenkins to allow for continuous integration and test
  - Perfect for large projects with many users and a lot of tests
  - Overnight or complete application test execution can be reduced from days to hours
  - Impact of Change analysis can be performed on the master project, greatly reducing the time it takes to identify regression errors
  - Speeds overall project testing time and reduces late-in-the-project side effects
- Wind River Simics
  - Massively parallel testing
  - Used together with CI and VectorCAST

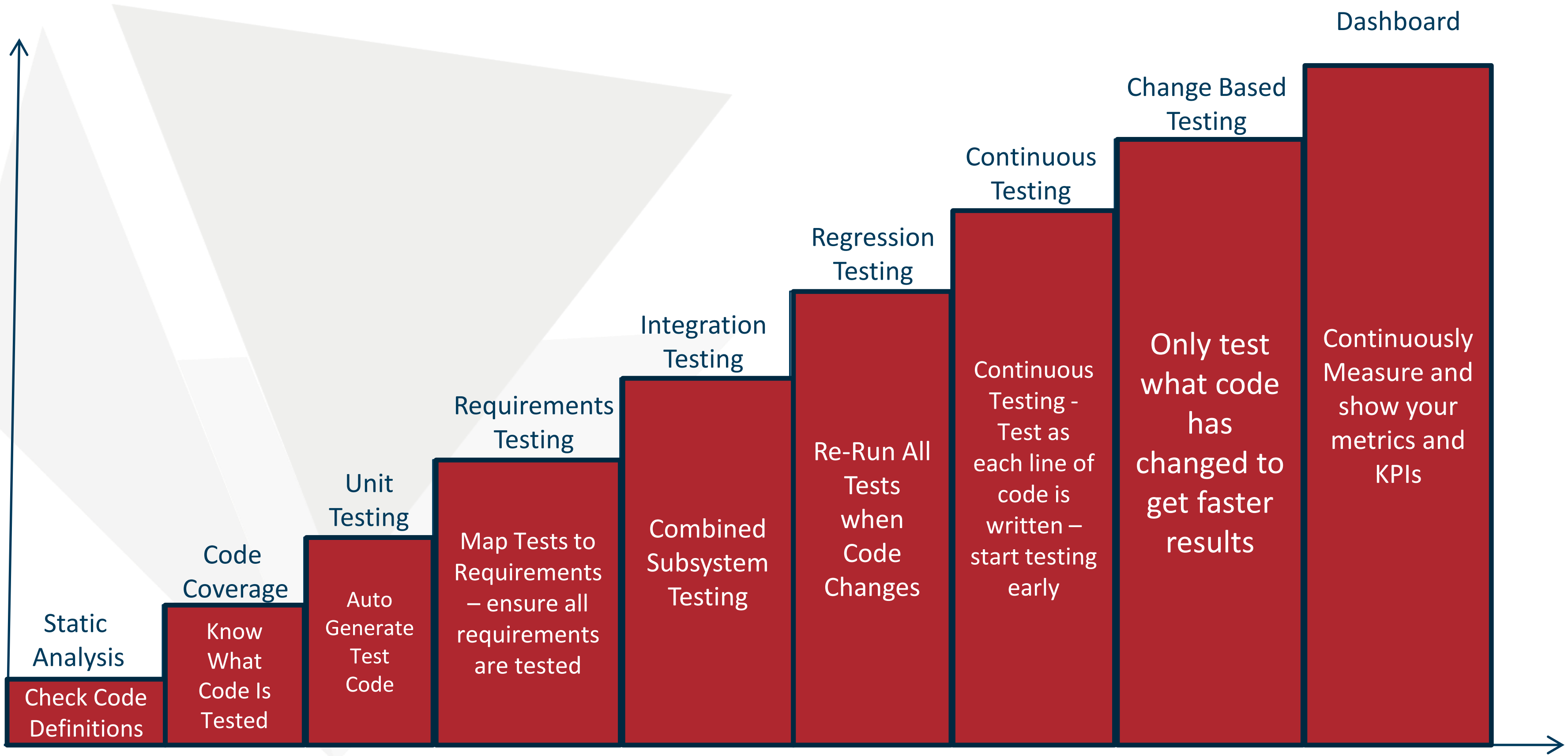
Parallel Testing

- User controlled execution of tests on clusters
  - Local or remote
  - Real or virtualized
  - Windows, Linux, or Solaris



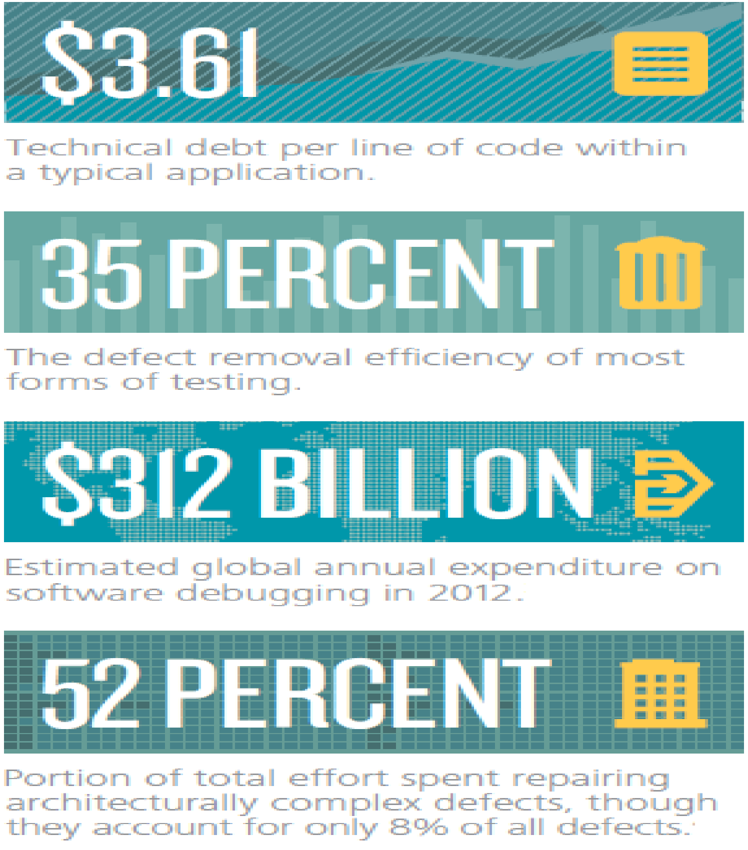


Where are you with your Test Process?



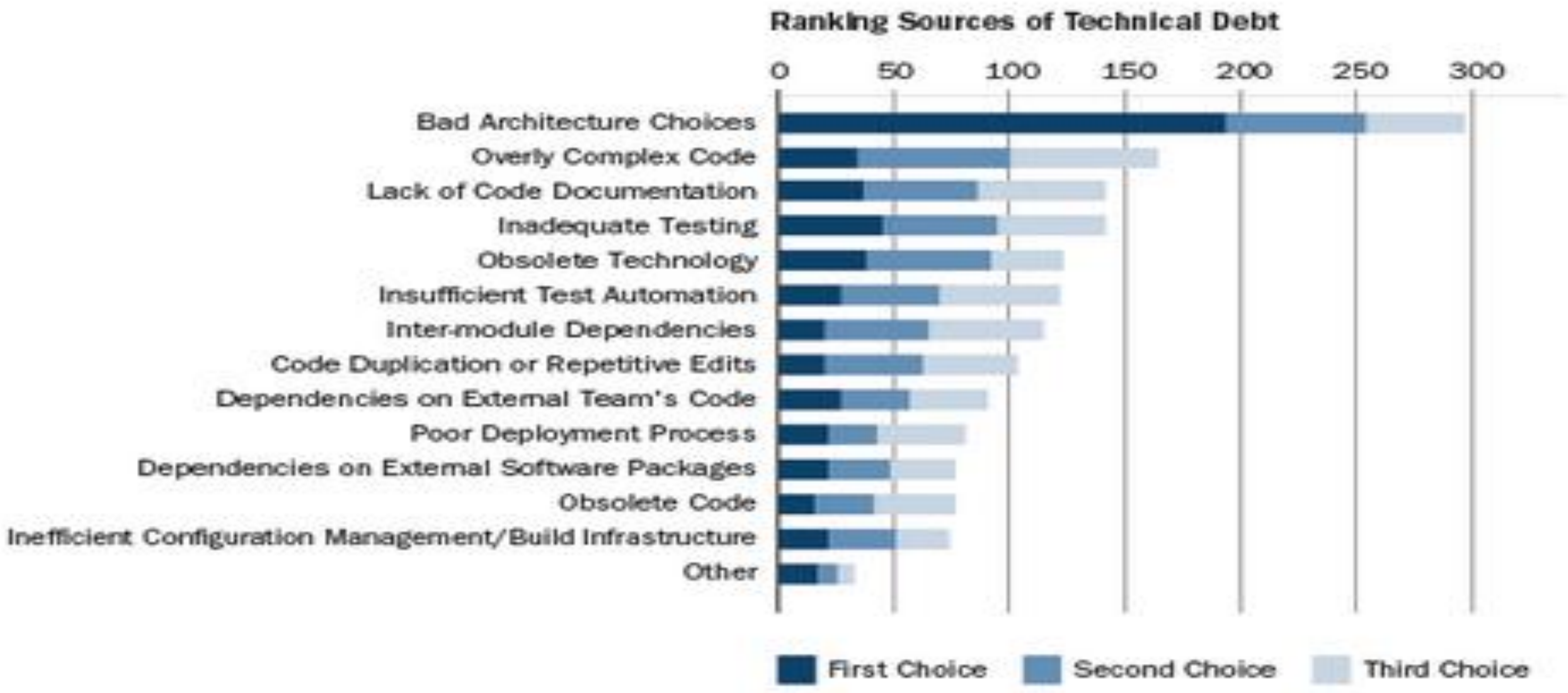
Reduction of the Technological Debt

TECHNOLOGICAL DEBT



Source: Deloitte University Press Technological Trends 2014

TECHNOLOGICAL DEBT



TECHNOLOGICAL DEBT

- Technical Debt: a burden for innovation
  - What is? An euphemism referring to the risk in production and potential rework assumed in software development
  - Due to the rush and other factors, a lack of quality in deployed software developments is allowed
  - It is normal that resources or quality are limited in every product (with infinite time and budget, everything is possible!)
    - *BUT in EVERY business world and in any professional field, the debt MUST be known or predictable, so that it can be managed, avoiding going bankrupt*
    - *So why Technical Debt is apparently IGNORED even in **AVIONICS**?*
  - ***Insist on this debt can also lead to technical bankruptcy***
  - ***Technical debt spends the entire budget for new projects and ends up being a tremendous drag on innovation***

<u>Technical Debt is the greater risk in a DO-178 Project</u>

TECHNOLOGICAL DEBT

- Technological debt and DO-178
  - ***The risk of prototyping & iterating***
  - ***The risk of «compliance»***
  - ***Last minute changes***
  - ***Different configuration/customer/product***

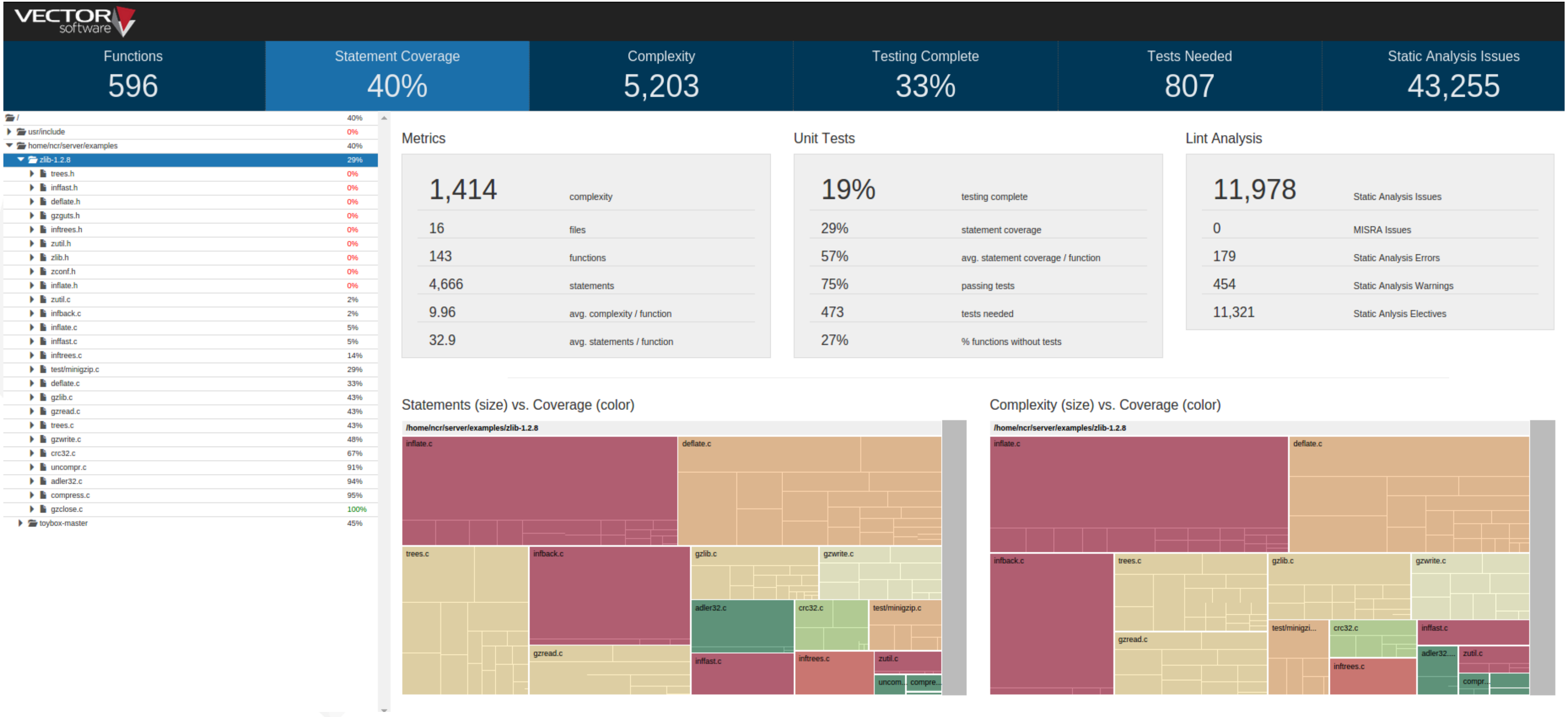
TECHNOLOGICAL DEBT

- RISK REDUCTION TECHNIQUES
  - ***GAP Analysis***
  - ***Assess the status of the code for current projects***
  - ***Project Ranking***
    - ***Higher to lower risk***
    - ***Size***
    - ***Maintenance status***
    - ***Problem Reports (open and total)***
  - ***Availability of talent to support debt remediation***
  - ***Action Plans based on priorities***

- ***Understanding and managing the technical debt hidden in the code means to eliminate the risk it generates.***

TECHNOLOGICAL DEBT: NEED YET ANOTHER DASHBOARD?

Analytics Dashboard for Metrics and KPIs



Solving problems in modern avionics

- Quality:
  - How to measure? *Metrics (Static Analysis, Coverage, McCabe, other KPIs)*
  - How to improve? *Correct verification strategy (Review, Requirement–based Testing)*
  - How to maintain? *Continuous Testing, Change-based Testing*
- Costs:
  - How to predict? *Metrics (Requirements #, automated coverage, other KPIs)*
  - How to reduce? *Early bug detection by Requirement-Based Testing (TDD), Change-Based testing*
- Impacts:
  - How to reduce the impact of first software version?

    Requirement Based Testing (TDD), Low-level requirements, 100% Coverage
  - How to reduce the impact of last minute software changes?

    Baselining, Change-based Testing, Parallel testing