VALIDAS

First
Tool Qualification
Symposium

Dr. Martin Wildmoser / 10.4.2013

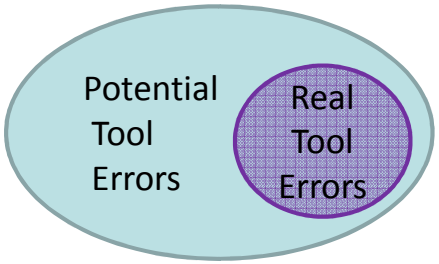
Tool Validation - Tutorial

Tool Validation

1. **Introduction**
2. Validation Process
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. Apply the VS
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

The Tool Confidence Process

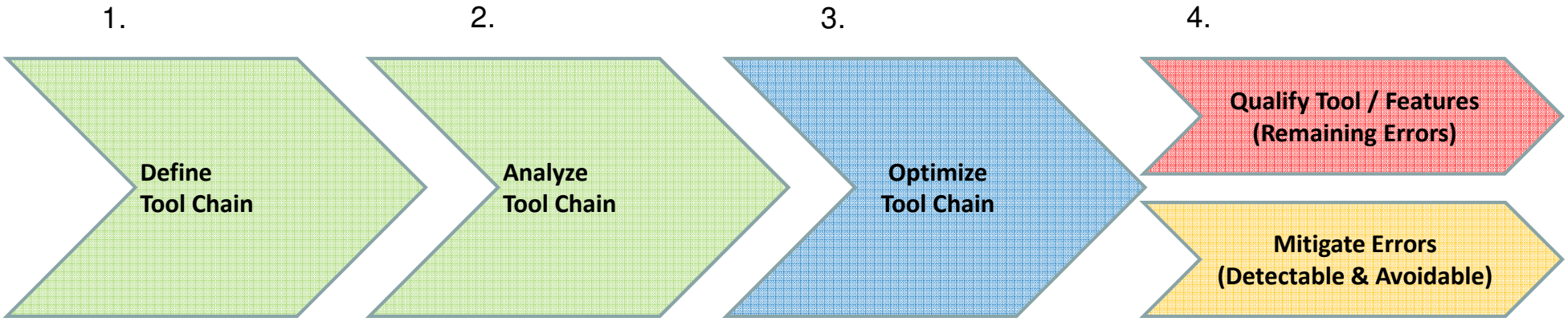
1. Definition: Tools in chain (process) with artifacts
2. Analysis: Determination of
   - Required confidence
   - Potential tool errors
3. (Optional) Optimization: tool chain improvements
4. (Alternative) Qualification: Once for each tool version
4. (Alternative) Mitigation: Every tool application



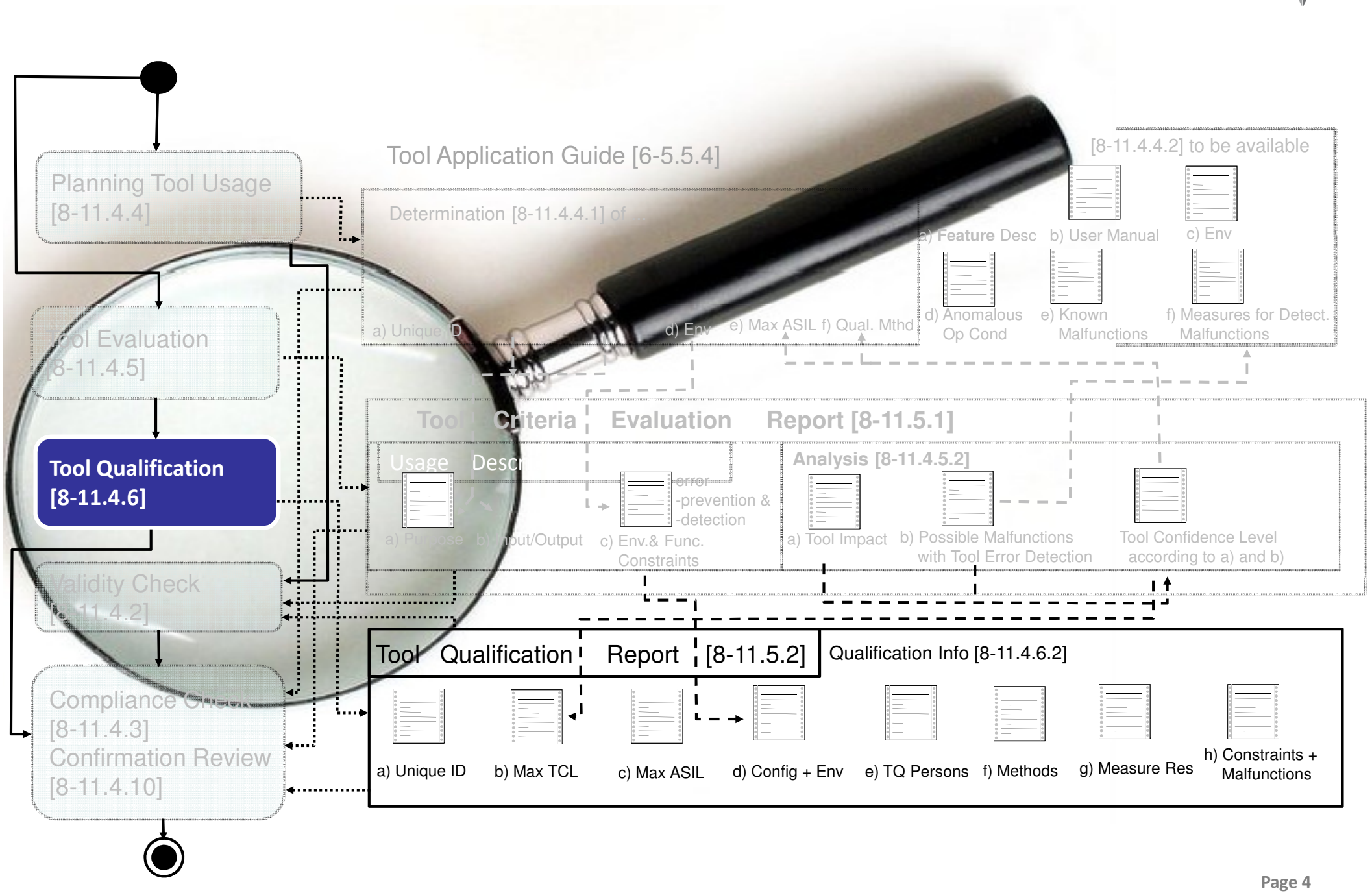
Plan Tool Usage

Classify / Qualify Tool

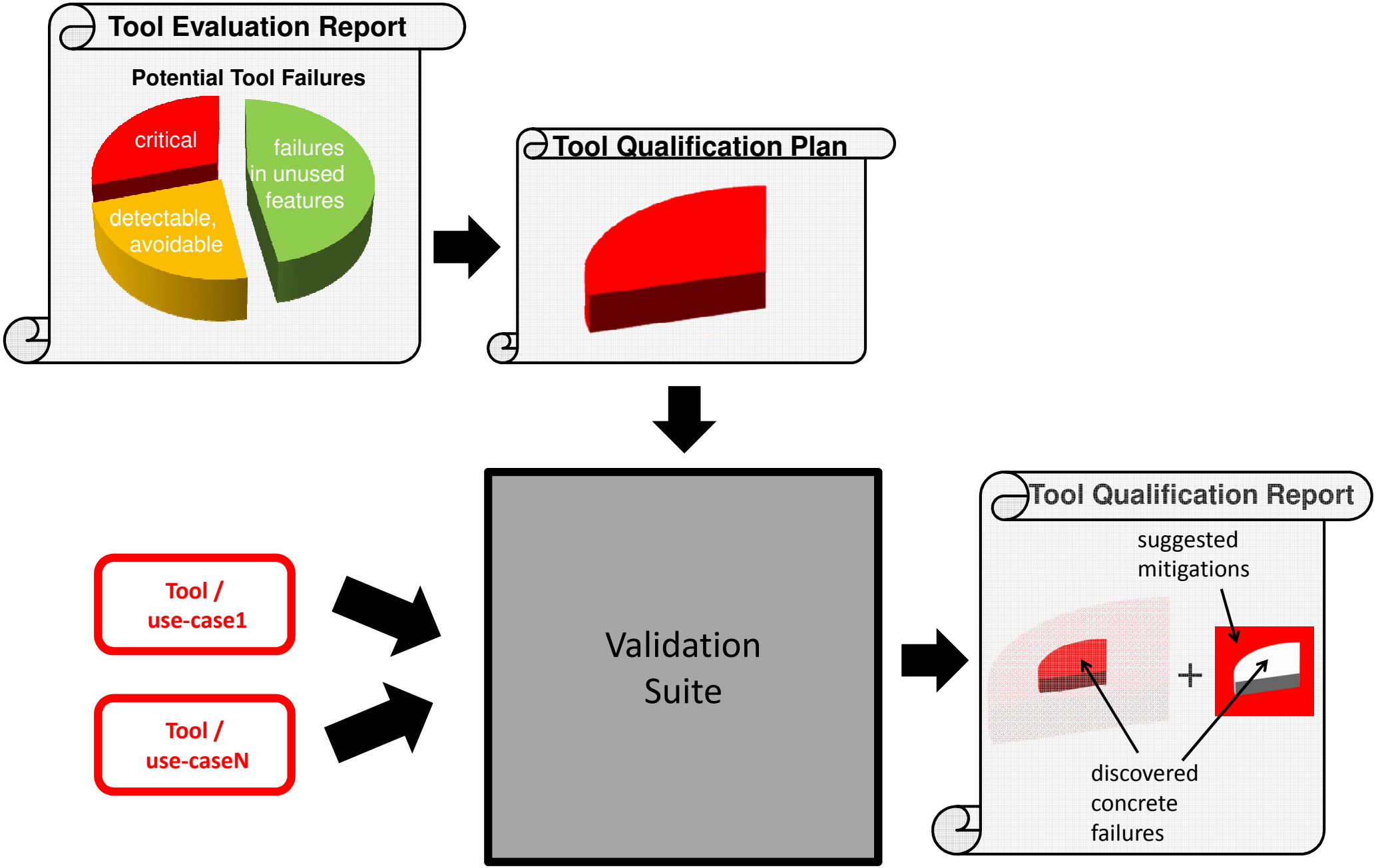
Use Tool



Tool Confidence in ISO26262



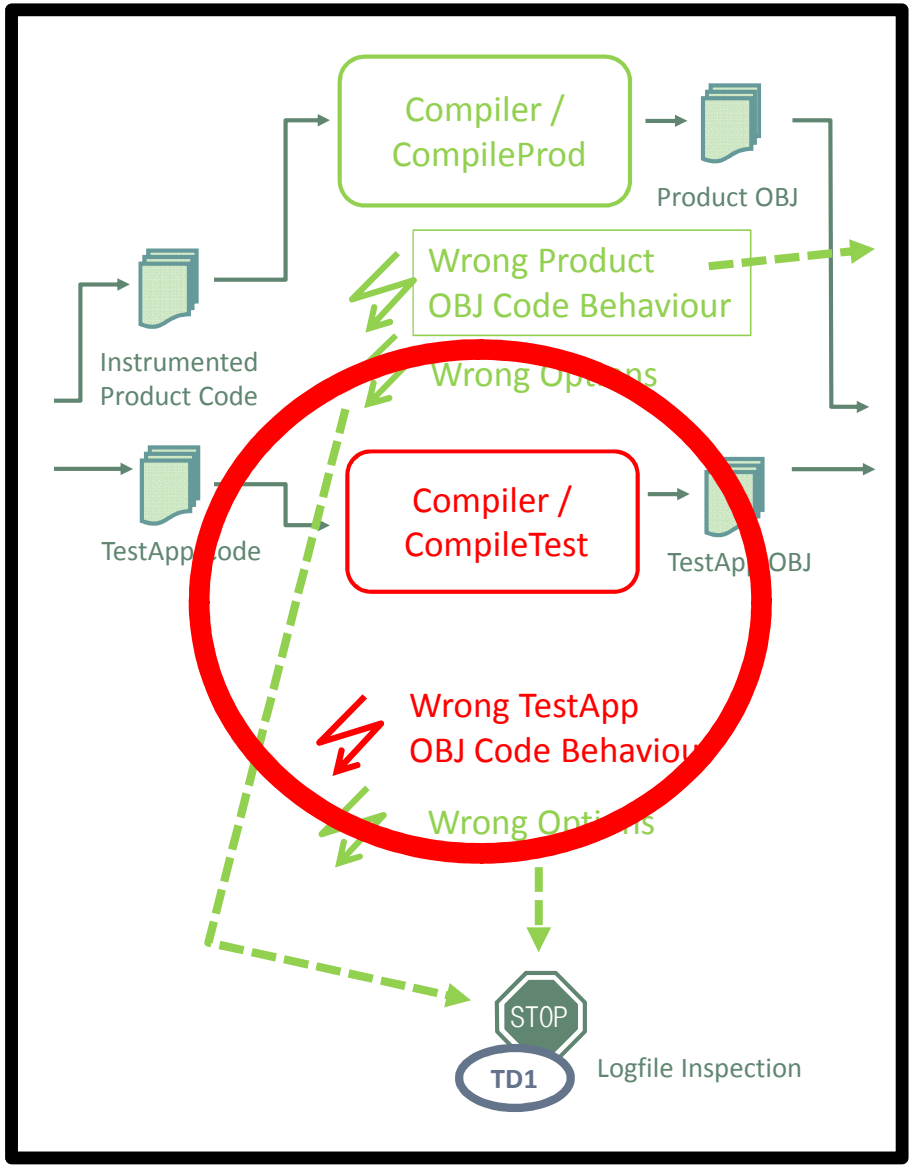
Tool Validation



What is Tool Validation?

- Tool Confidence = Evidence that tool does not inject or fail to detect safety relevant product faults.
- Approaches:
  a. Test every tool output (Translation Validation)
  b. Test the tool (Translator Validation)
- Tool Validation = Test the tool ...
  - as used in the development project
  - with adequate and sufficient tests
  - with adequate process to develop and use the tests

Example: Testing Tool Chain



The goal for tool validation is to show by systematic testing that the potential tool failures without high detection probability do not occur.

Validation Goal:

Provide evidence for non-occurance of the potential failure „Wrong TestApp OBJ Code Behaviour“ in practice.

Return of Investment

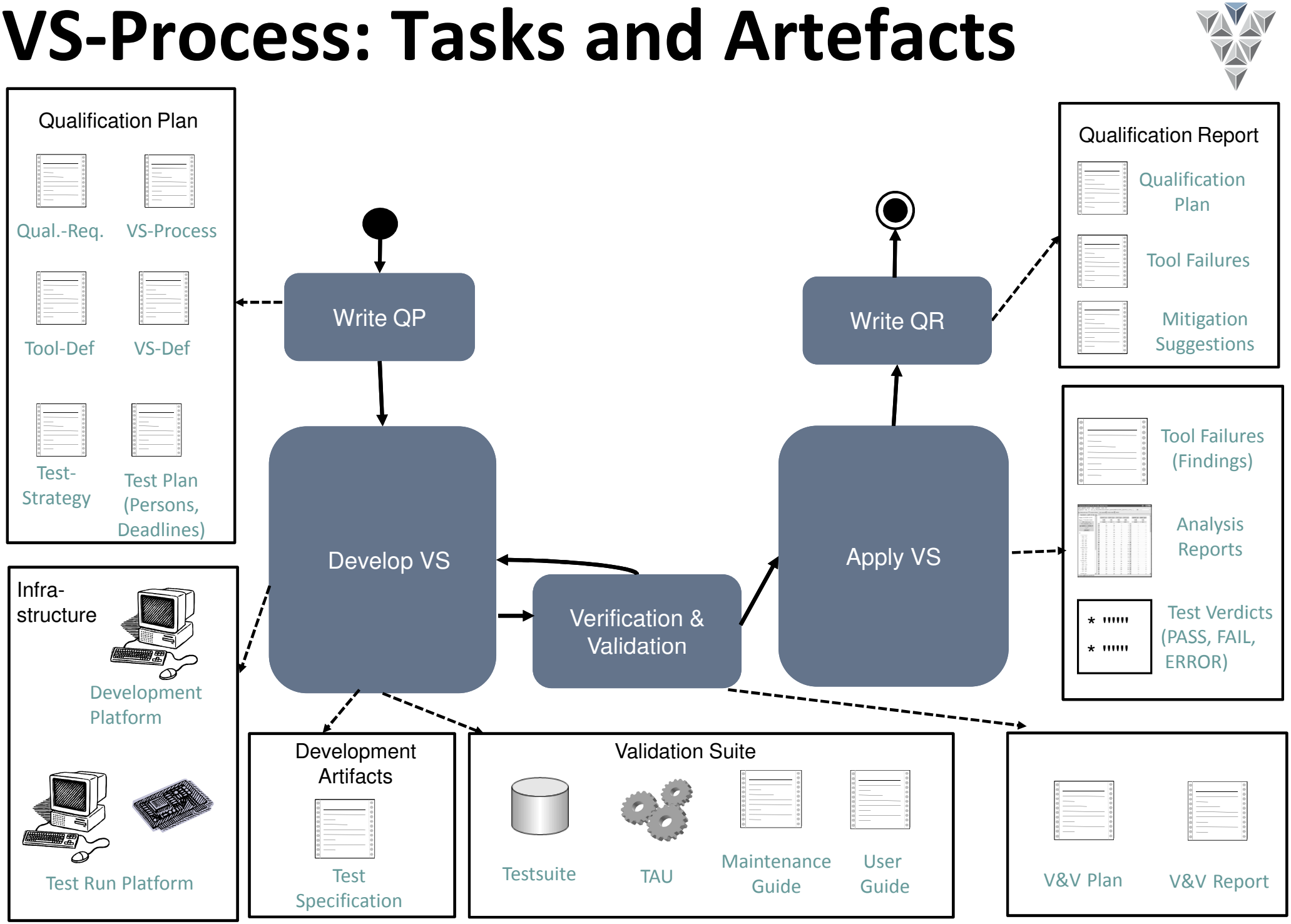
Tool Validation

- Investments
  - Construction of test suite: 1-time
  - Documentation: 1-time
  - Maintenance: N-times
  - Application of VS: N-times
- Returns
  - Evidence for tool confidence
  - Less effort for product tests
  - Less ad-hoc changes in tool chains

Tool Validation

1. Introduction
2. **Validation Process**
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. Apply the VS
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

VS-Process: Tasks and Artefacts



Tool Validation

1. Introduction
2. Validation Process
3. **Write Tool Qualification Plan**
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. Apply the VS
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

Qualification Plan

Tool Definition
-version
-use cases
- environment
- configuration
- usage limitations
- input / output formats
- max ASIL?
- TCL?

Qualification Requirements from **ISO 26262** **§ a,b,…,z**

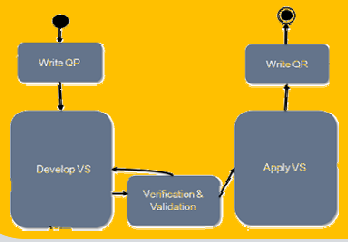
Qualification Plan

Qual.-Req. | VS-Process

Tool-Def | VS-Def

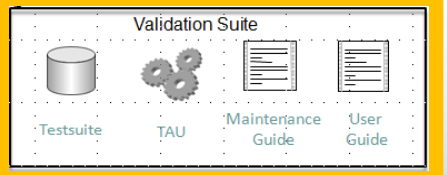
Test-Strategy | Test Plan (Persons, Timeline)

Qualification Method = Tool Validation
Validation Process =



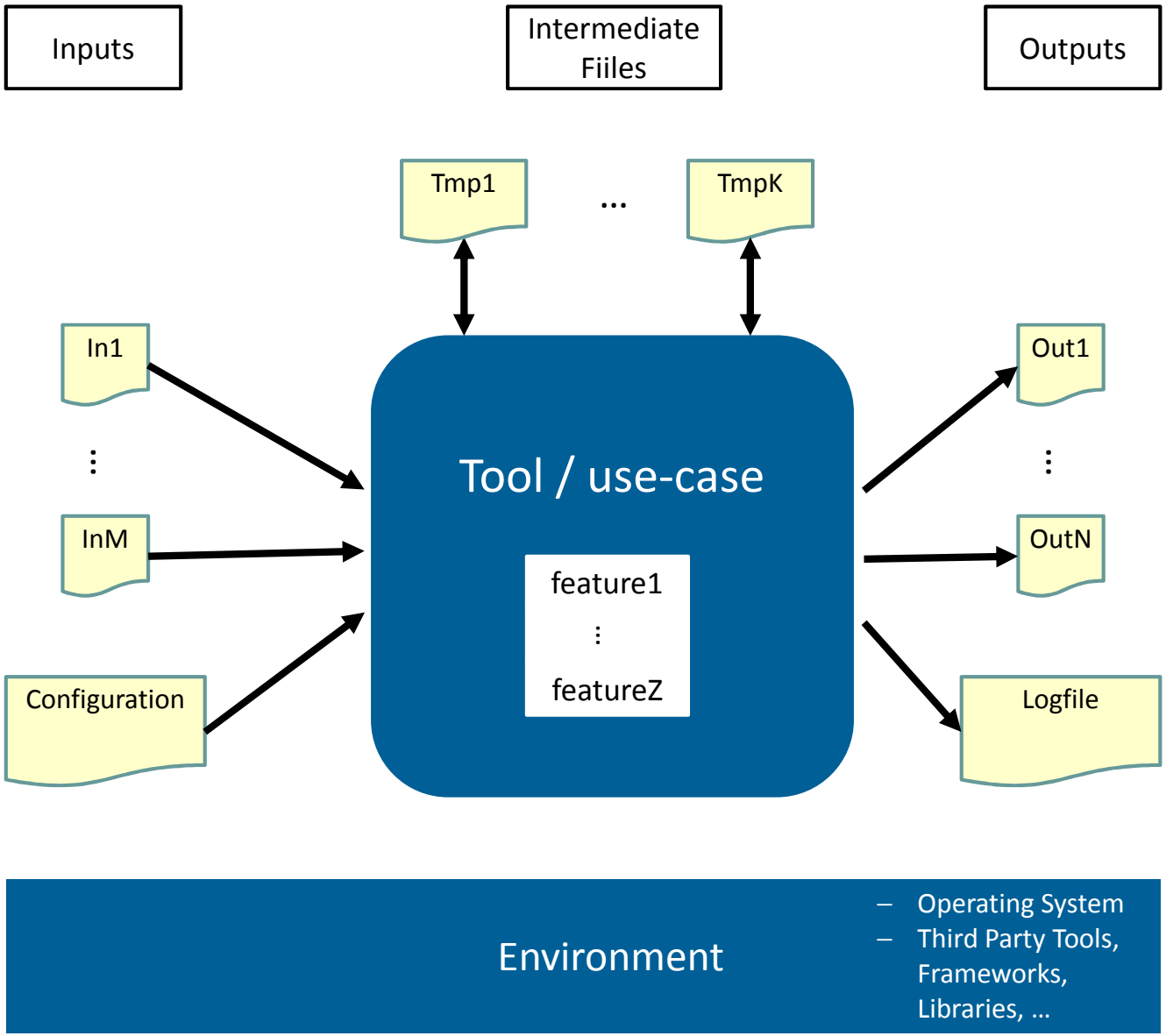
Test Strategy = Mapping from Qualification Requirements to **Test Goals**, **Test Design Techniques** and **Test End Criteria**.

Parts of Product „VS“:



Validations Goals = What properties of the tool shall be shown?

Tool Definition - Input/Output



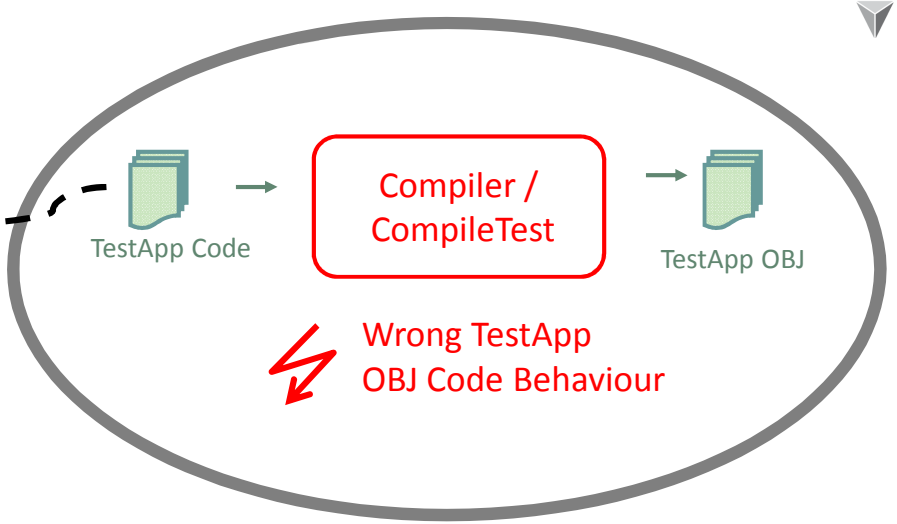
Tool Definition - Tool features

- Tool Feature = Collective term for
  - conceptual functions
  - configuration options
  - language constructs in input/output files
- Tool Evaluation/Classification:
  - high level description of tool features suffices
  - task: find out the tools that need qualification
- Tool Qualification:
  - precise description of tool features needed
  - use official terms from the tool manual
  - task: systematic testing

Example: Test App Constructs

```c
#include <float.h>

/* global product input variables */
extern float in1;
extern float in2;

/* global product output variables */
extern float out1;
extern float out2;

/* product runnable */
extern void run(void);

int main(void) {
  /* temporary variables */
  float tol = 2.4e-7; /* ~ 2*eps = 2^-22 */
  float ref = 0.707106781f; /* sin(pi/4) */
  float tmp1 = 0.0f;
  float tmp2 = 0.0f;

  /* set inputs for runnable */
  in1 = 0.785398163f; /* pi/4 */
  in2 = in1;

  /* call function under test */
  run();

  /* check output */
  tmp1 = out1 - out2;
  tmp1 = (tmp1 >= 0.0f) ? tmp1 : -tmp1;
  tmp2 = out1 - ref;
  tmp2 = (tmp2 >= 0.0f) ? tmp2 : -tmp2;
  /* abs(out1-out2) <= tol and abs(out1-ref) <= tol */
  return ((tmp1 <= tol) && (tmp2 <= tol));
}
```

example



expressions only

call of function under test

expressions only

Observation: Our testapplication only require a tiny subset of constructs of the C-language.

We only need to qualify our C-Compiler for compilation of:

- float variables and expressions (w.o. function call)
- function call of void/void functions

Tool Definition – Input domain

- Tip: Restrict the input domain!
  - only allow tool features essential for the use-case.
  - restricting input domains reduces qualification costs!

- Tip: Create a profile of actually used tool features
  - scan real examples and count occurrences of used constructs.
  - negotiate a precise list of allowed tool features with tool users.

Qualification Requirements

- ISO26262:2011 states at least 15 requirements (directly and indirectly) for qualification method „tool validation“.
- There are 3 kinds of qualification requirements:
  - VS test cases

Table 1 Tool Qualification Requirements from ISO 26262

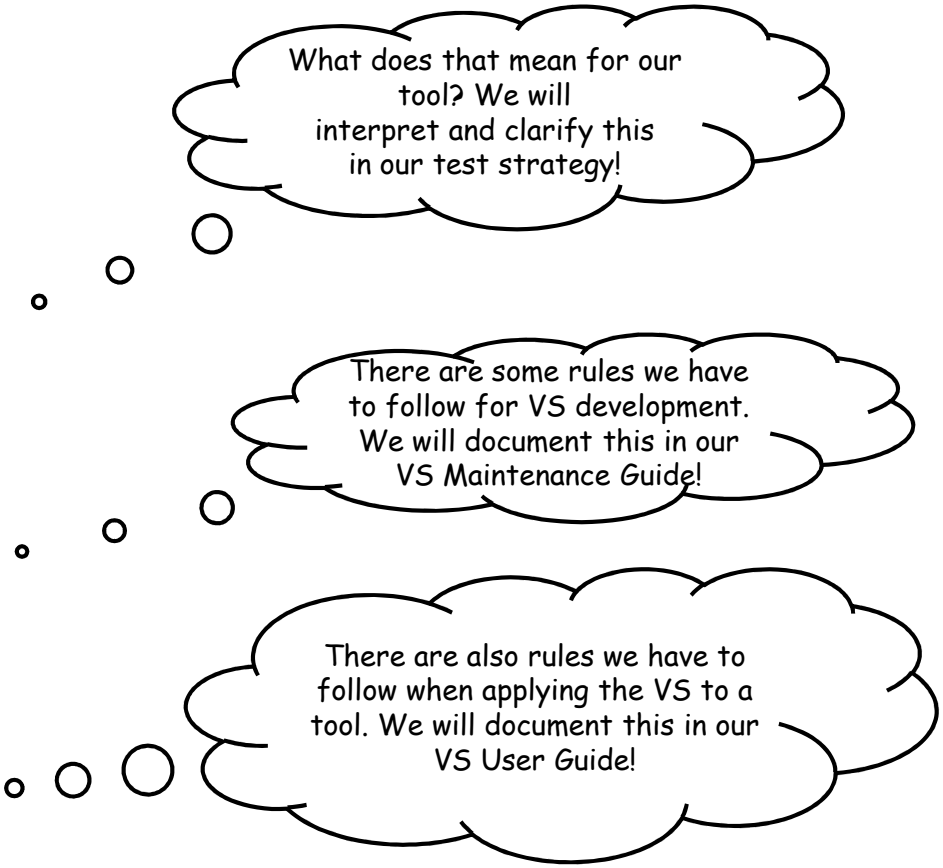
Nr	Req. ID	Source	Derived Requirement Text
10	VS-1	ISO 26262-8:2011 11.4.9.2 a)	The validation suite shall demonstrate that the software tool complies with its specified requirements.
11	VS-2	ISO 26262-8:2011 11.4.9.2 c)	The validation suite shall contain tests that check the software tools reactions under anomalous operating conditions.

  - VS development

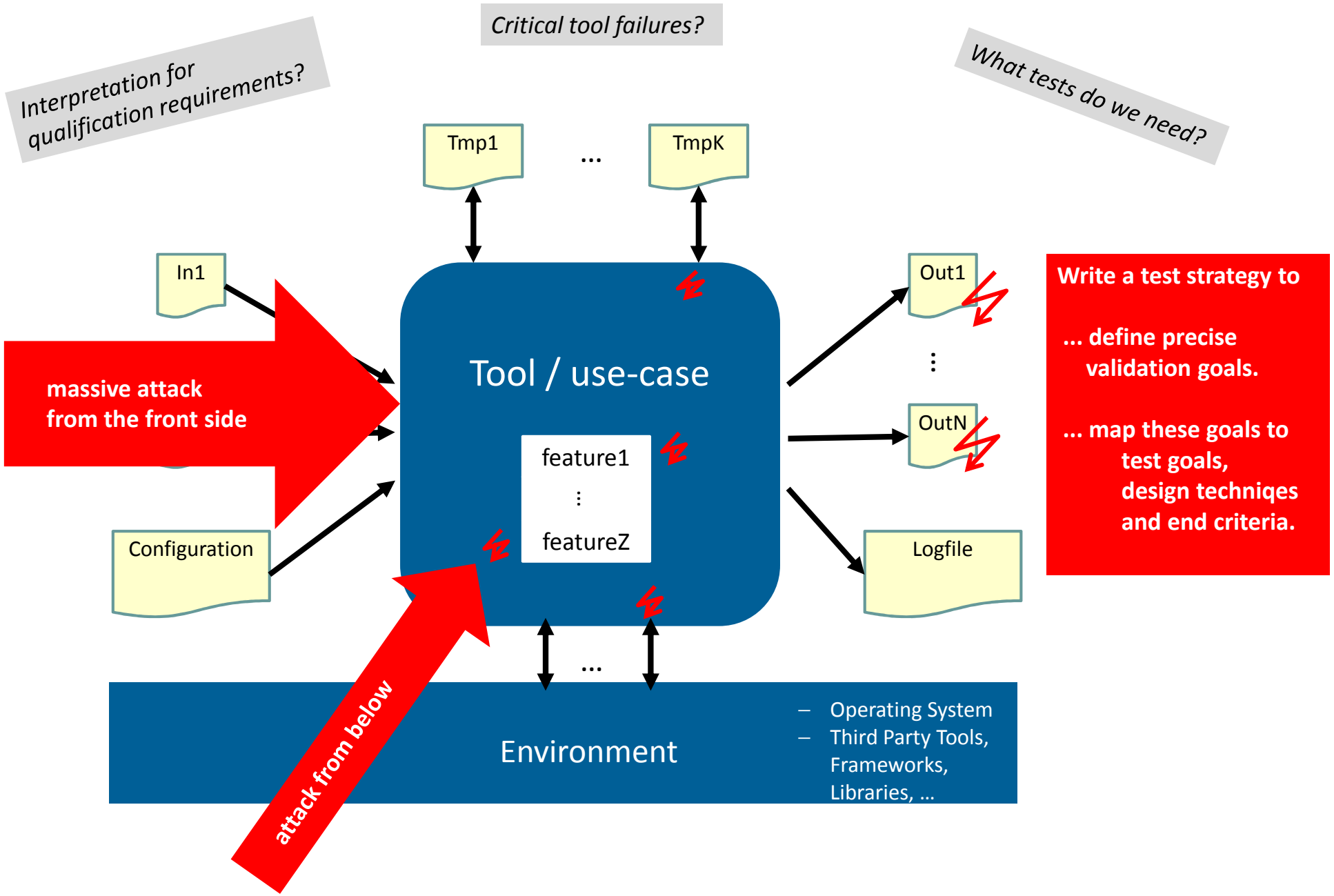
13	Proc-2	ISO 26262-8:2011 7.2	The Tool Qualification Report (work product) and all artefacts required to produce it, i.e. the Validation Suite, shall be put under configuration management.

  - VS application

9	Doc-9	ISO 26262-8:2011 11.4.6.2 h), ISO 26262-8:2011 11.4.9.2 b)	For every malfunction in the software tool identified during qualification, the Tool Qualification Report suggests measures to avoid or detect these errors if possible.



Test Strategy

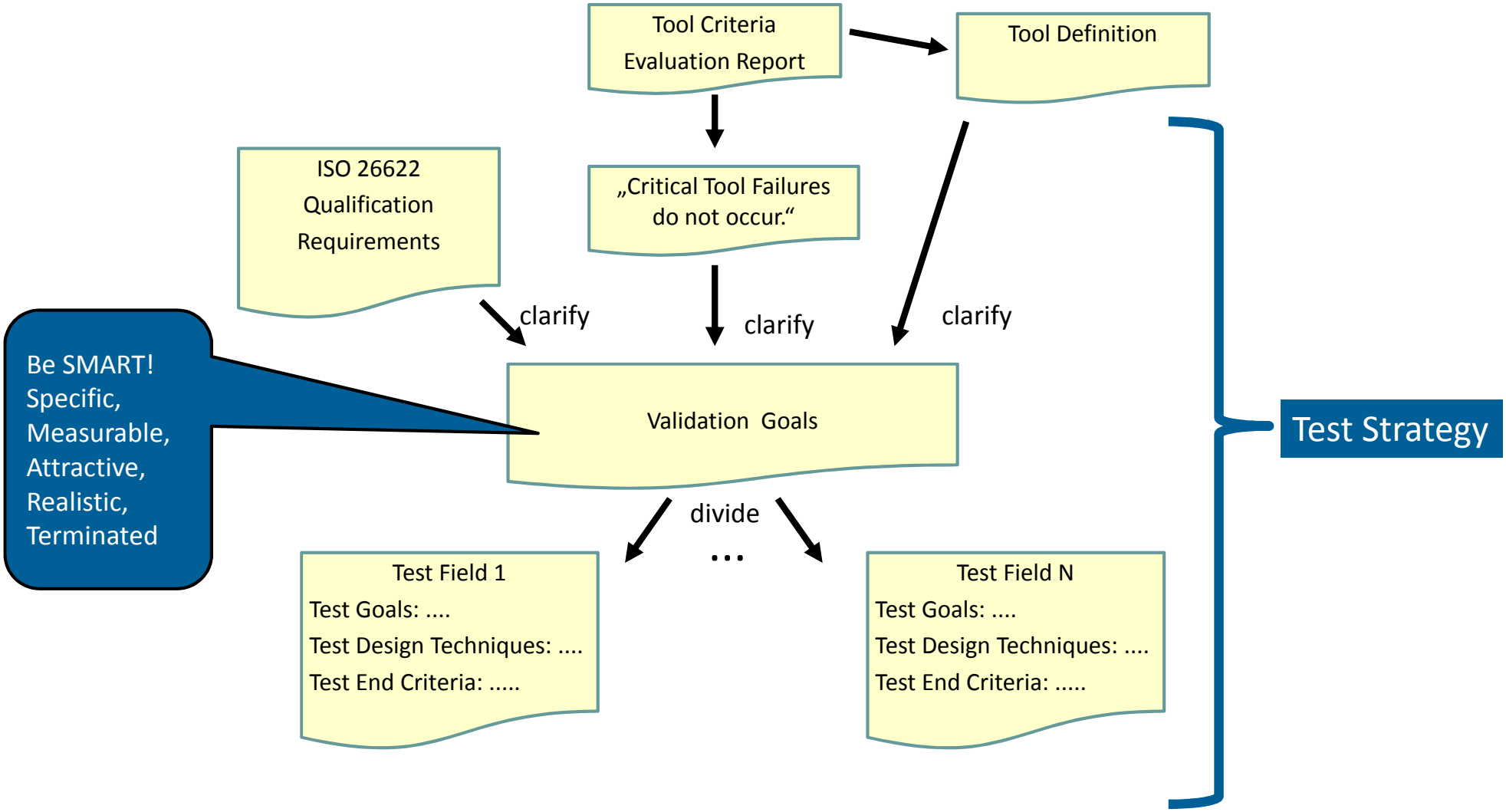


Write a test strategy to

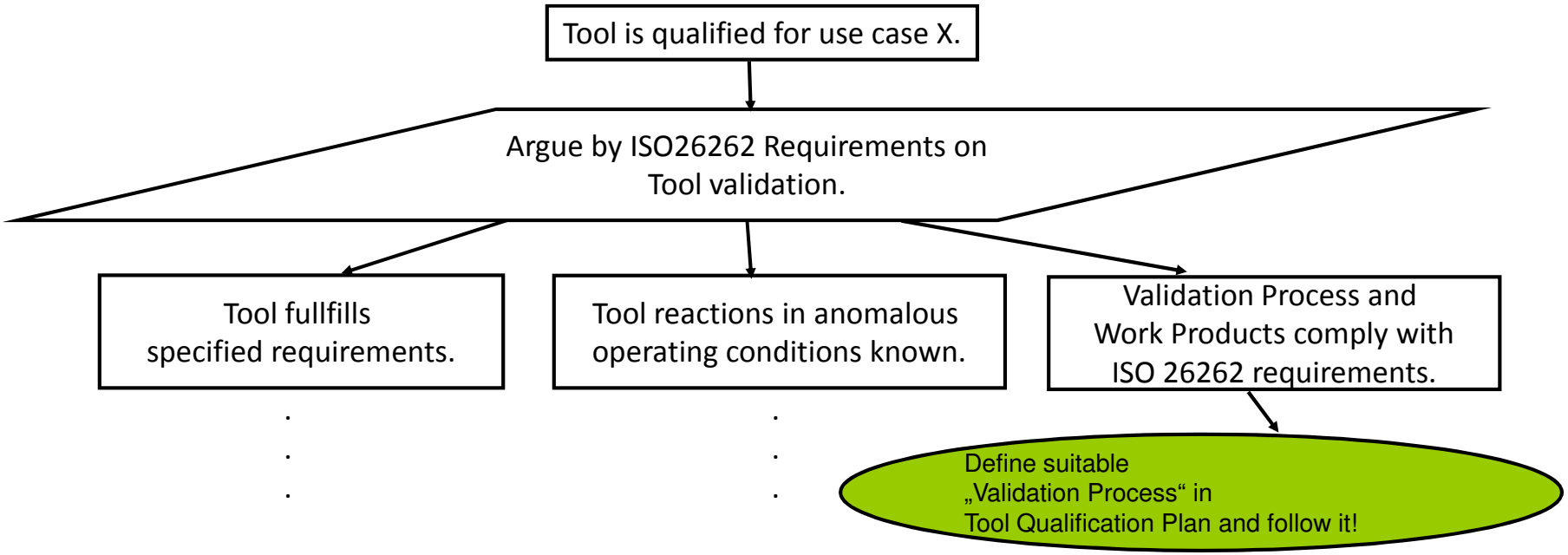
... define precise validation goals.

... map these goals to test goals, design techniqes and end criteria.

Test Strategy - Overview



Test Strategy - Argumentation



Goal Structuring Notation (Only some parts)
(by T. Kelly)

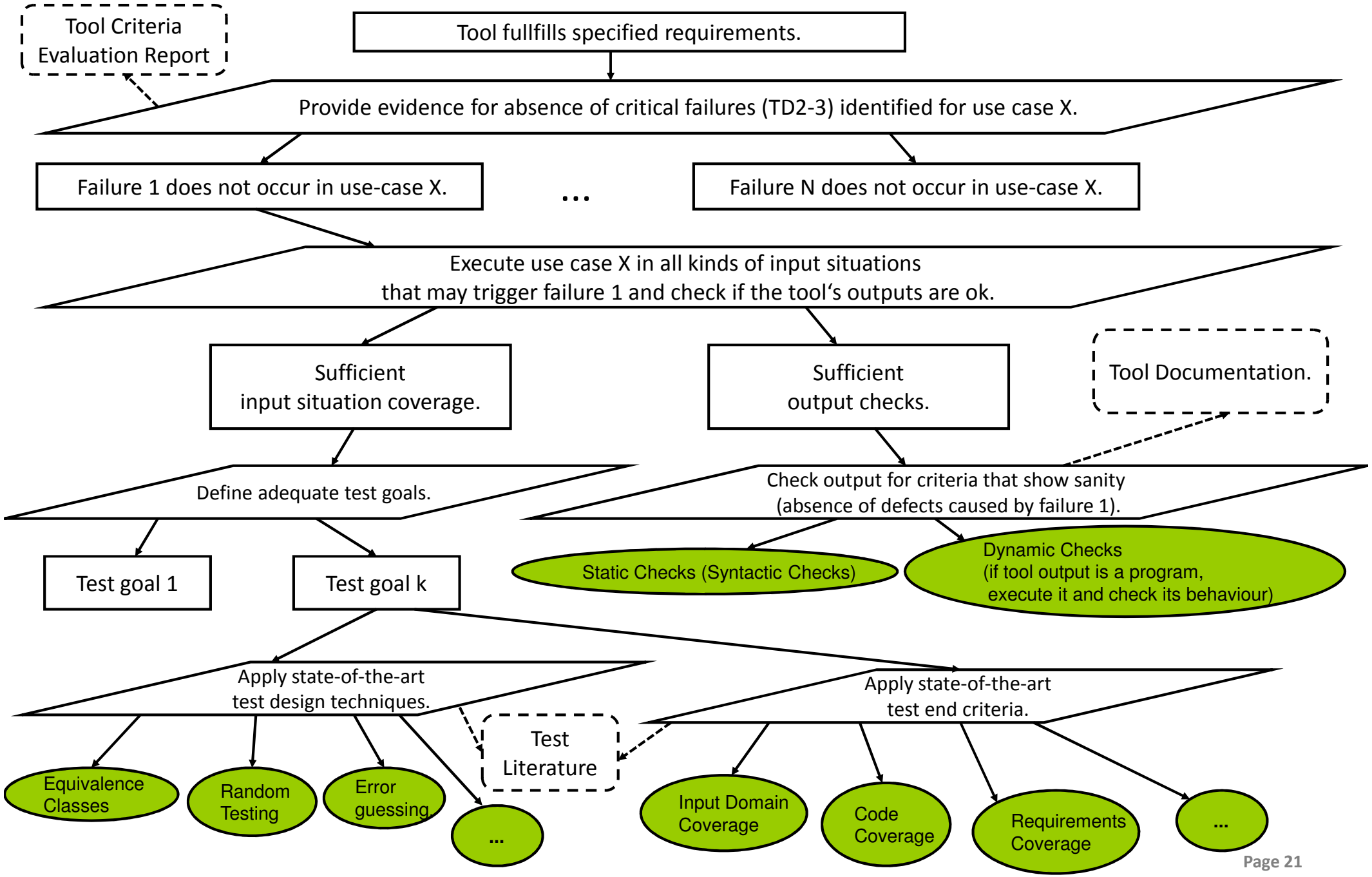
Goal — Hypothesis or Claim that needs to be shown.

Context — Additional relevant information.

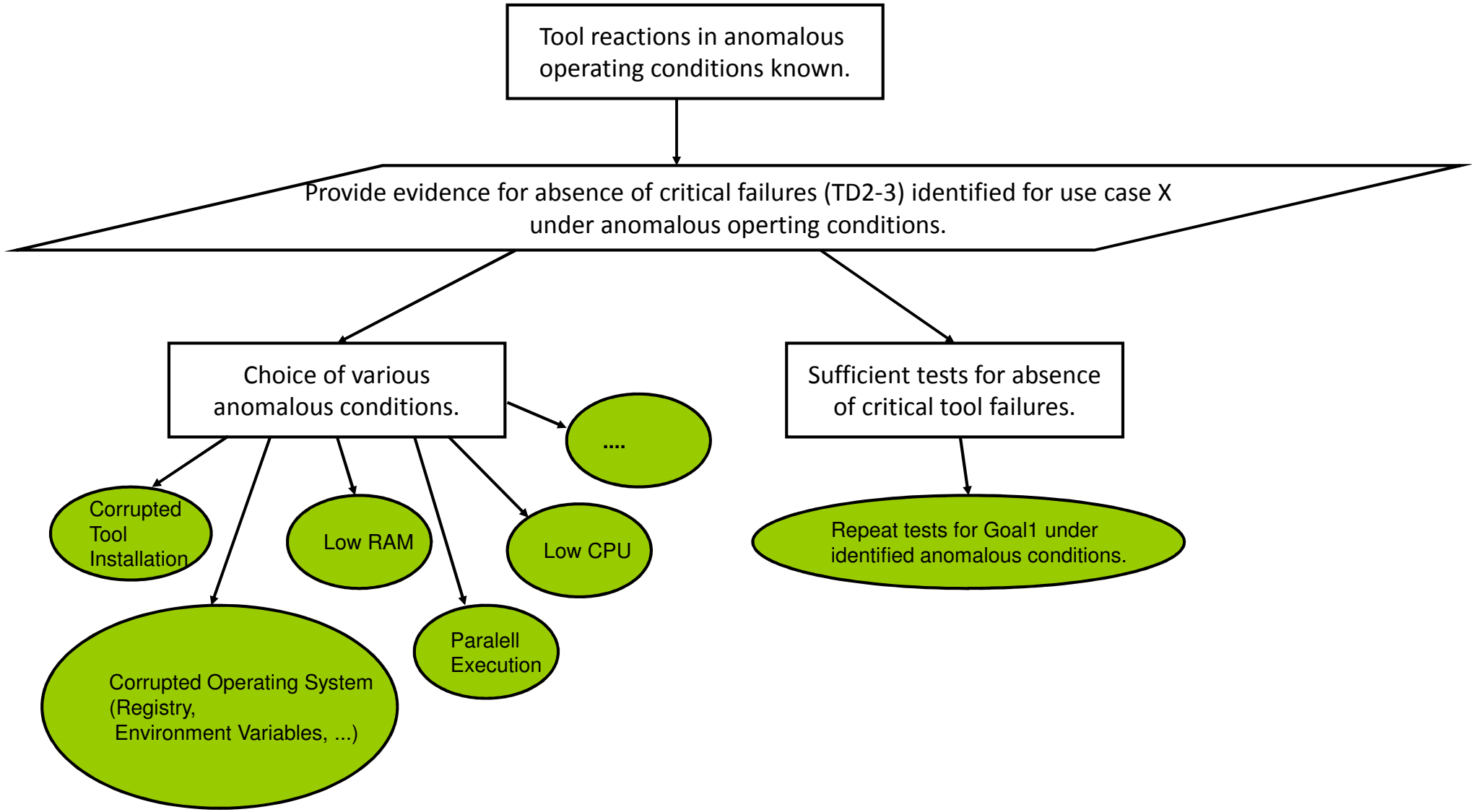
Argument — „Activity“ that reduces a goal to sub goals.

Evidence — Elementary Fact or Claim, End of argumentation

Goal1: Specified Requirements



Goal2: Anomalous Op. Conditions

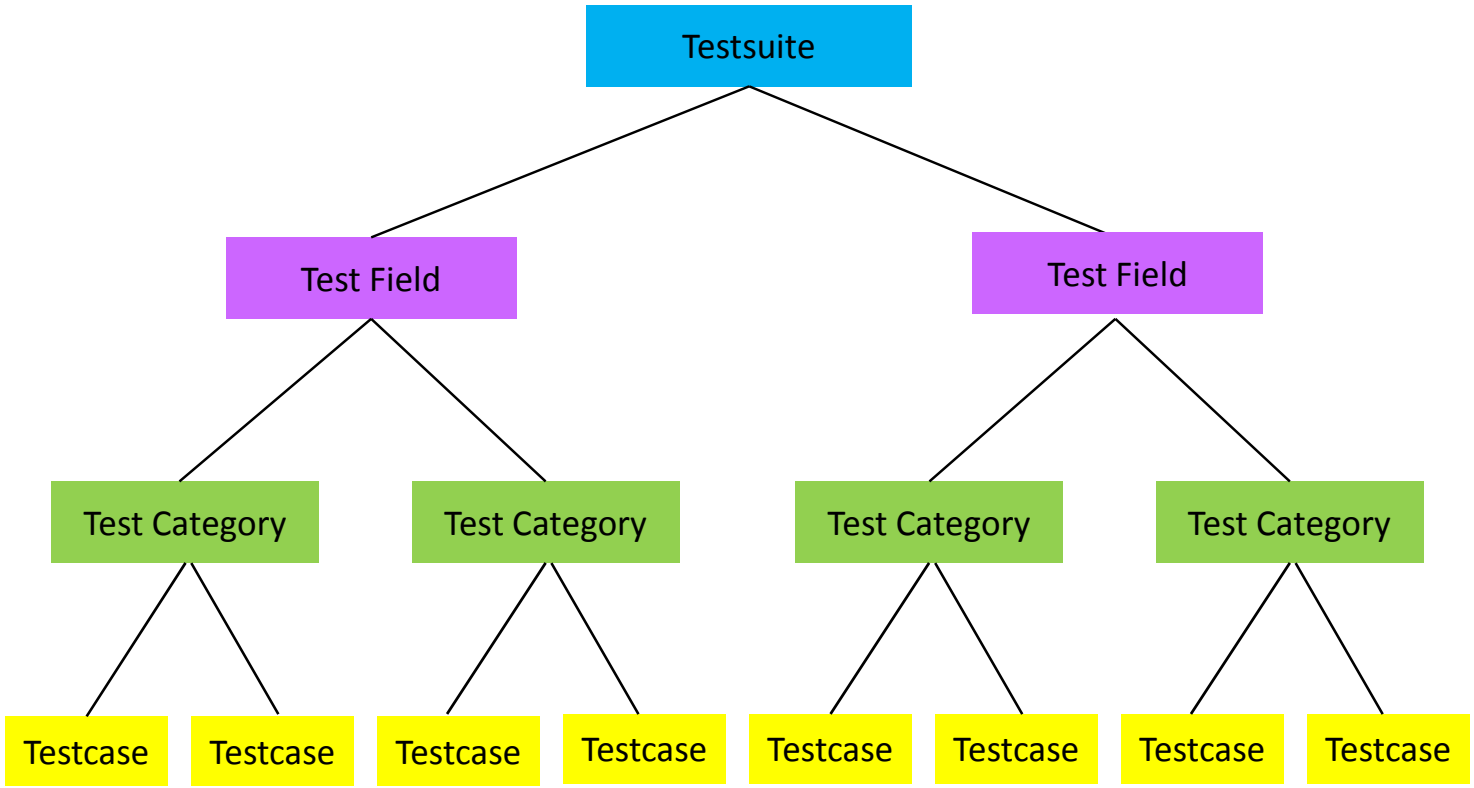


Tool Validation

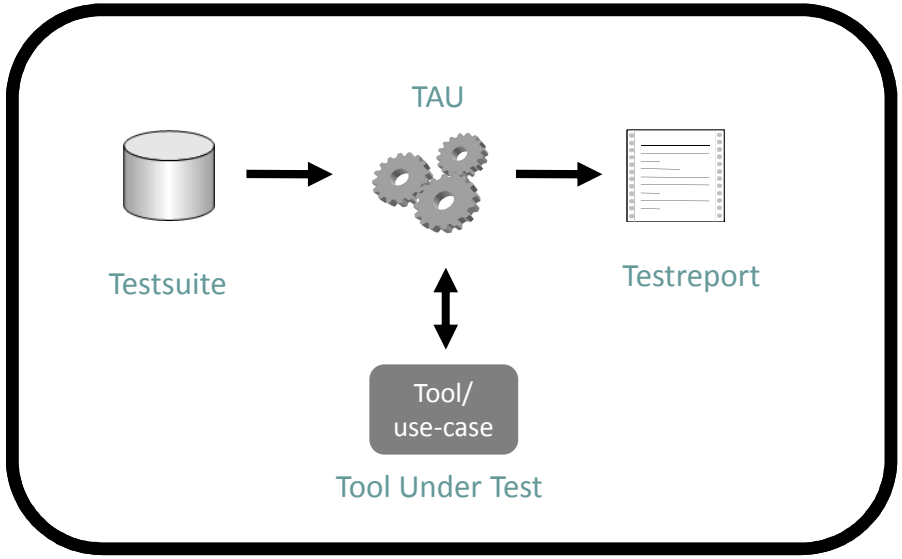
1. Introduction
2. Validation Process
3. Write Tool Qualification Plan
4. **Develop Validation Suite (VS)**
5. Verifiy and Validate VS
6. Apply the VS
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

- Develop Test Automation Unit (TAU)
- Develop Tests
  - Test Design Techniques
  - Measure Test-End Criteria

Test Suite


Testsuite
Test Field
Test Field
Test Category
Test Category
Test Category
Test Category
Testcase
Testcase
Testcase
Testcase
Testcase
Testcase
Testcase
Testcase

Develop Test Automation Unit



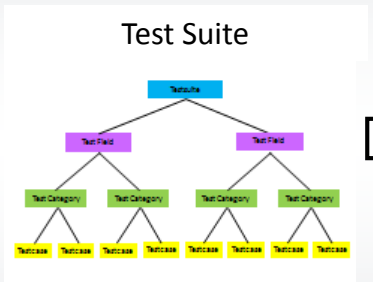
The Test Automation Unit (TAU) ....

... applies the VS tests to the tool-under-test

... automates running collections of tests (inter-test-execution)

... automates the workflow of using the tool on a single test case (intra-test-execution)

TAU: Inter-Test Execution



TAU / Run Tests

Intra Test Execution

Test Run Folder

Test Run Folder

Test Run Folder

Test Run Folder

Test Run Folder

Test Run Folder

Test Run Folder

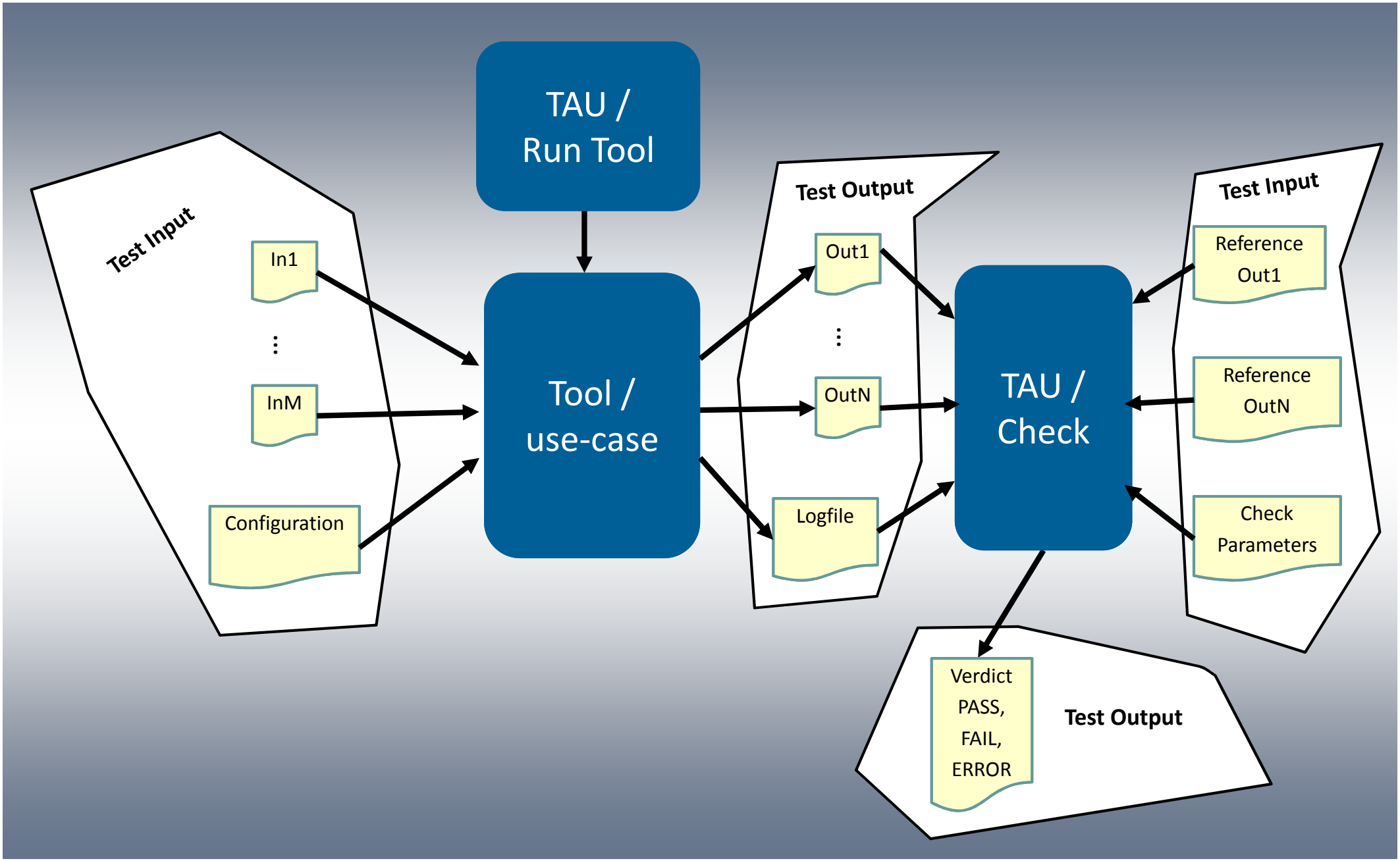
Test Run Folder

TAU / Gen Report

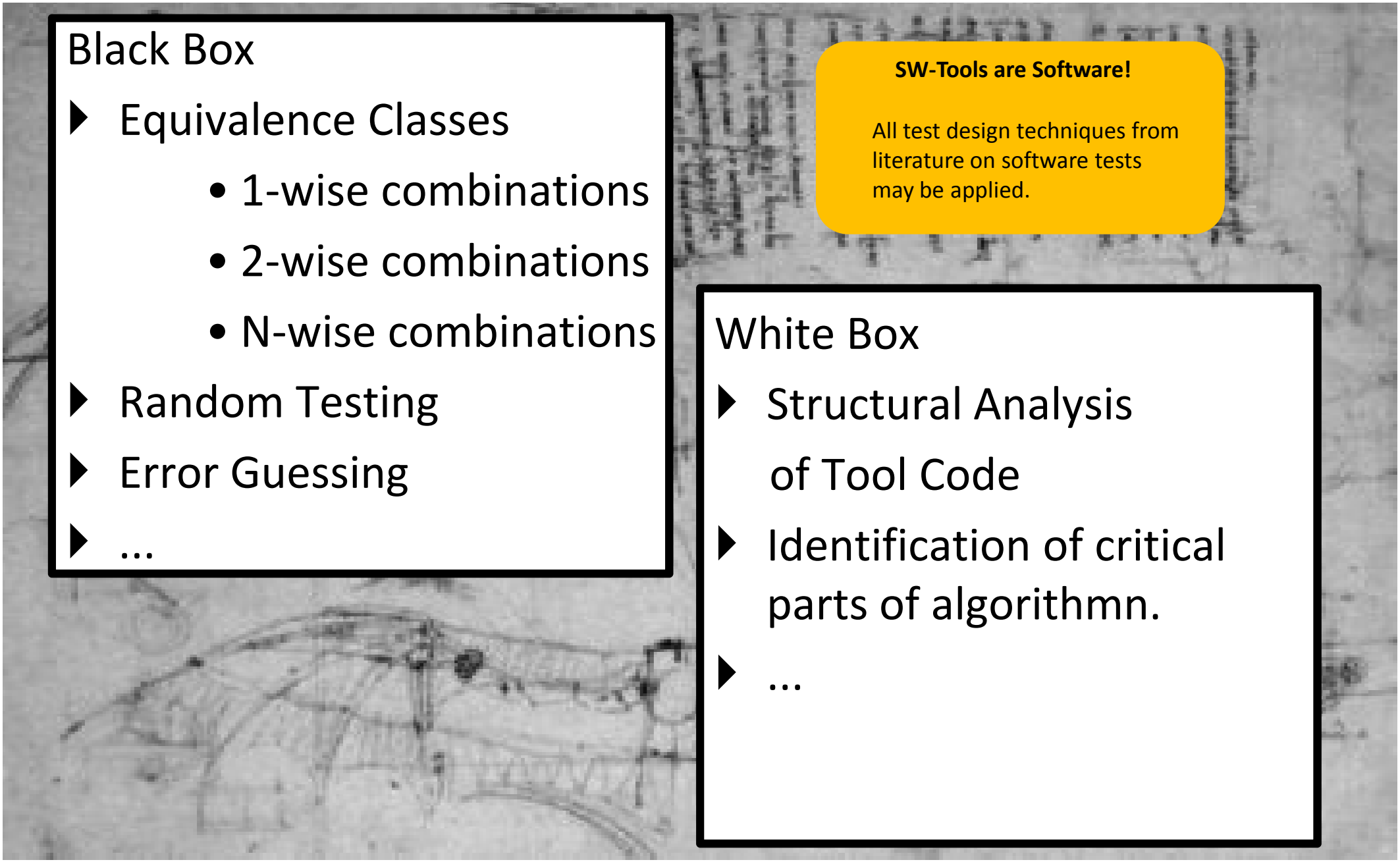
Test Report

Test1: PASS
Test2: PASS
Test3: FAIL
Test4: PASS
Test5: PASS
Test6: FAIL
Test7: FAIL
Test8: ERROR

TAU: Intra-Test Execution



Test Design Techniques



Black Box

- Equivalence Classes
  - 1-wise combinations
  - 2-wise combinations
  - N-wise combinations
- Random Testing
- Error Guessing
- ...

SW-Tools are Software!

All test design techniques from literature on software tests may be applied.

White Box

- Structural Analysis of Tool Code
- Identification of critical parts of algorithmn.
- ...

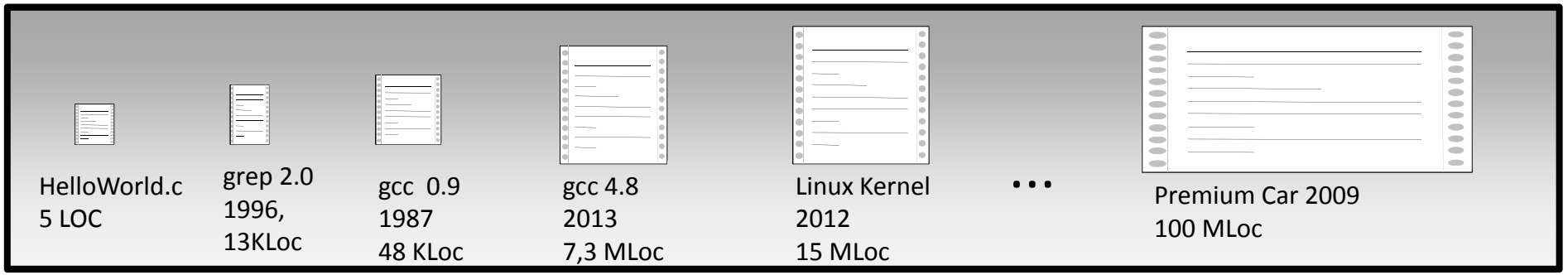
Test Design: Equivalence Classes

- Idea:
  - Partition input domain into finitely many classes.
  - Each class should trigger the same tool failures.
- Simple Example:

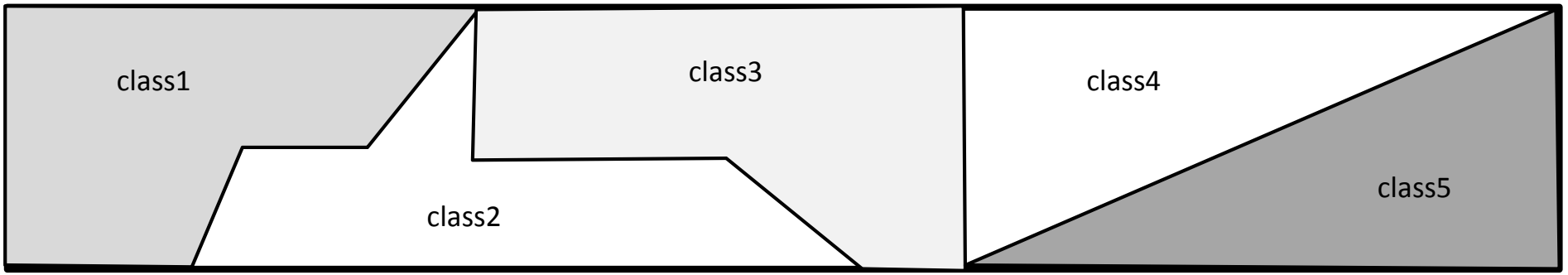
- Challenge with tools:
  - Multi-dimensional and complex input domains.
  - Example: Domain of all C-Programs.

Test Design: Equivalence Classes

- Domain of all C-Programs



- How can we define the borders?

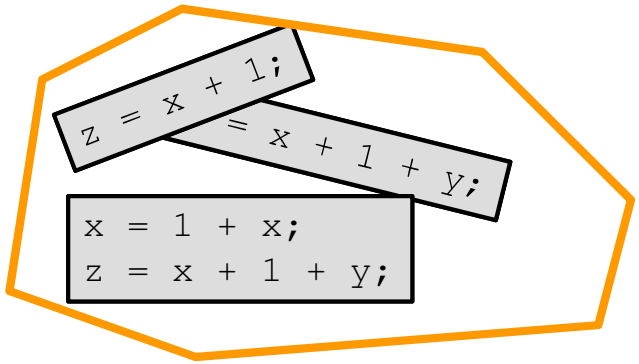


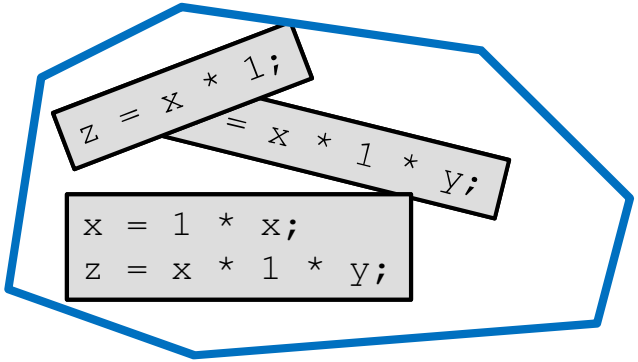
  - Idea: Define properties p: C-Program -> Boolean
  - Examples: contains_pointer_arithmetics(prog) = 0/1

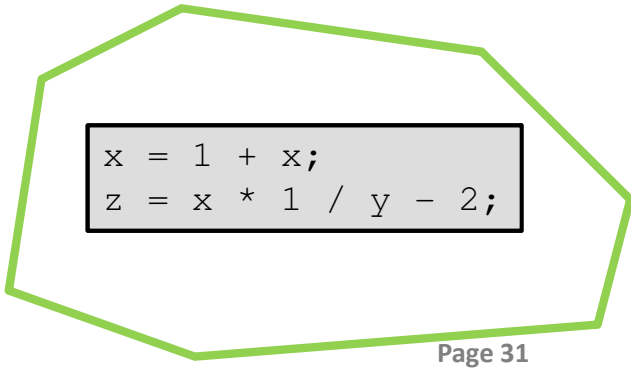
Test Design: Equivalence Classes

- One **property** for each language construct of interest.
- Evaluating properties on a given program yields a **profile**.

profile → / property↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
contains "+" ?	N	Y	N	N	N	Y	Y	Y	N	N	N	Y	Y	Y	N	Y
contains "-" ?	N	N	Y	N	N	Y	N	N	Y	Y	N	Y	Y	N	Y	Y
contains "*" ?	N	N	N	Y	N	N	Y	N	Y	N	Y	Y	N	Y	Y	Y
contains "/" ?	N	N	N	N	Y	N	N	Y	N	Y	Y	N	Y	Y	Y	Y
Eq. class →	op0	add	sub	mul	div	add_ sub	add_ mul	add_ div	sub_ mul	sub_ div	mul_ div	op3				op4
Group →	none	1-wise				N-wise										all
Testfield →	Basic Constructs					Combined Constructs										







Test Field: “Basic Constructs”

- Idea: Test each language construct isolated.

- Test Goals:
  - Check if semantics of each construct is as expected.

- Test Design Techniques:
  - Equivalence Classes
  - 1-wise combinations

- Test End Criteria:
  - At least one test for each construct.
  - Each property-value present in at least one test.

Test Field: “Combined Constructs”

- Idea: Test combinations of language contructs.

- Test Goals:
  - Check for unexpected interactions between constructs.

- Test Design Techniques:
  - Equivalence Classes
  - 2-wise combinations, ..., N-wise combinations

- Test End Criteria:
  - At least one test for each (meaningful) pair of constructs.
  - At least one test with all constructs.

Test Design: Random Testing

- Idea: Create test inputs by using a construction algorithm with random choices.

- Textual Programs: Use grammar of the language.
  - Start with root symbol.
  - At each step: Randomly expand a non-terminal.

- Graphical Models: Use series of actions, e.g. „add block X“ or „connect block A with B“.
  - Start with empty model.
  - At each step: Randomly apply a legal construction action.

Test Field: “Random Constructs”

- Idea:
  Create random tests, e.g. by enumerating the grammar of the language.

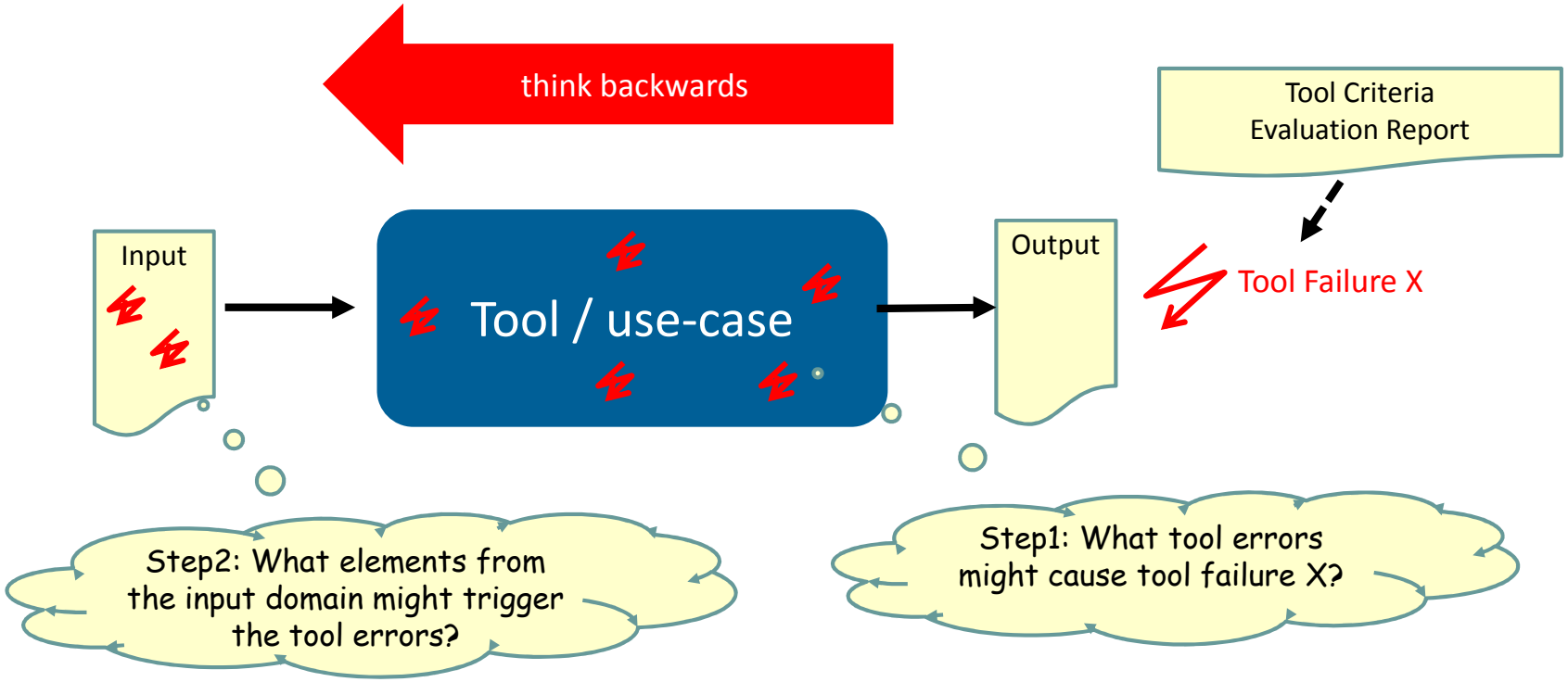
- Test Goals:
  - Trigger tool failures with unusual combinations of constructs.

- Test End Criteria:
  - At least k tests of size N.
  - At least k/10 tests of size N*10.
  - …

Test Design: Error Guessing

- Idea:

  Guess critical situations that likely trigger the tool failures.



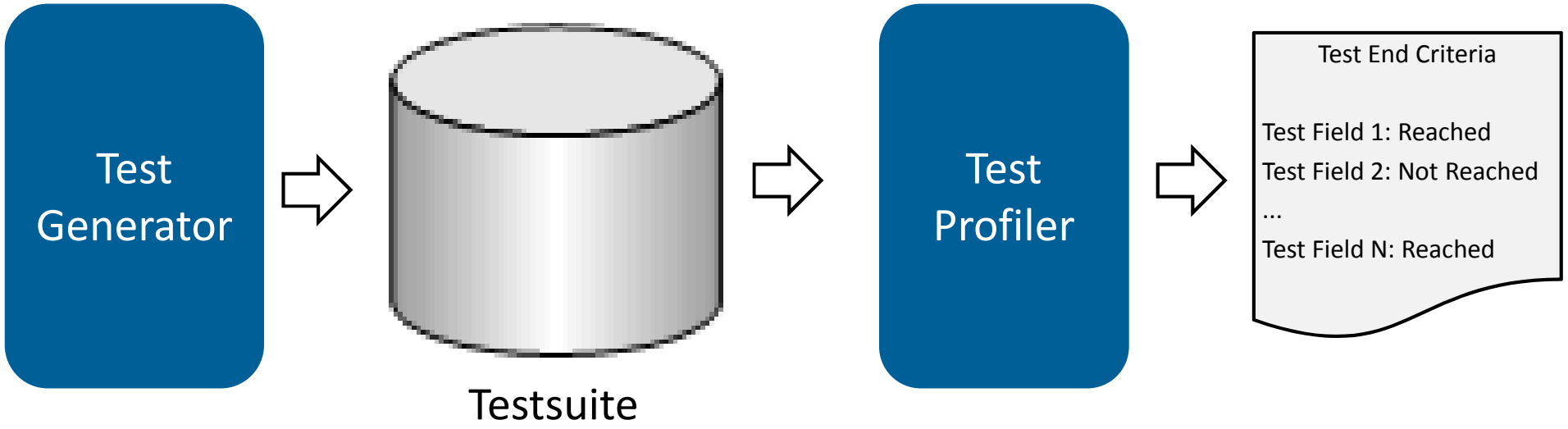
Note: Re-use insights from tool criteria evaluation!

Test Field: “Error Guessing”

- Idea:
  Guess critical situations from general knowledge on a tool‘s features.
- Test Goals:
  Check Tool in situations with
  - extreme inputs: large, small
  - side effects.
  - special values, e.g. Inf, Nan.
  - buffer overflow.
  - …
- Test End Criteria:
  - At least one test per identified critical situation.

Development Tools

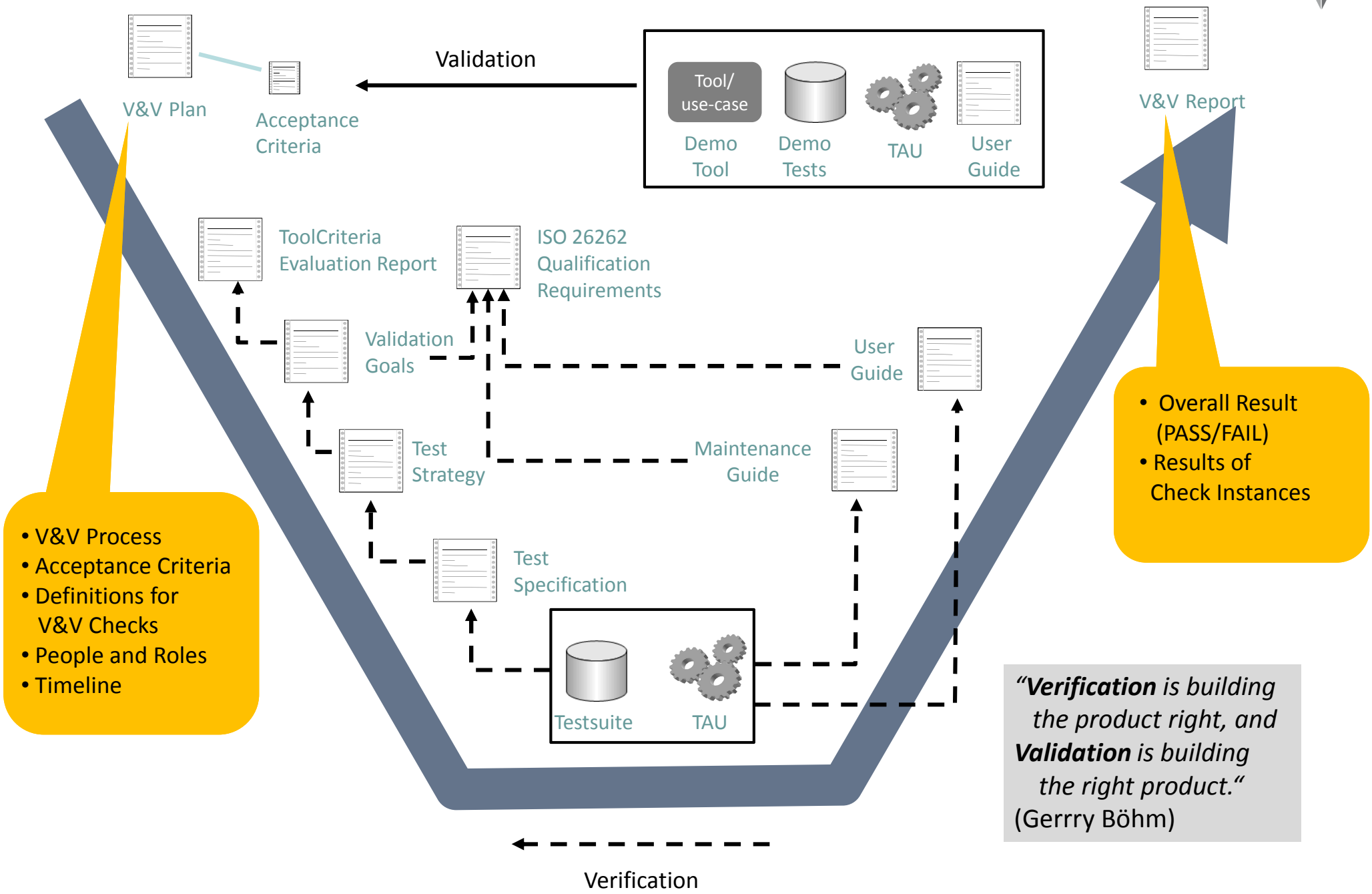
- Additional development tools are typically required.



Tool Validation

1. Introduction
2. Validation Process
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. **Verifiy and Validate VS**
6. Apply the VS
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

VS Verification and Validation

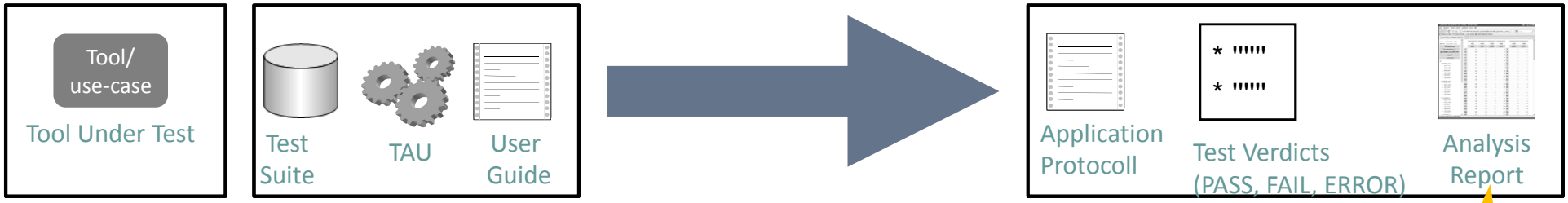


*"**Verification** is building the product right, and **Validation** is building the right product."*
(Gerrry Böhm)

Tool Validation

1. Introduction
2. Validation Process
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. **Apply the VS**
7. Write Tool Qualification Report
8. Vision: Customizable Qualification Kits

VS Application



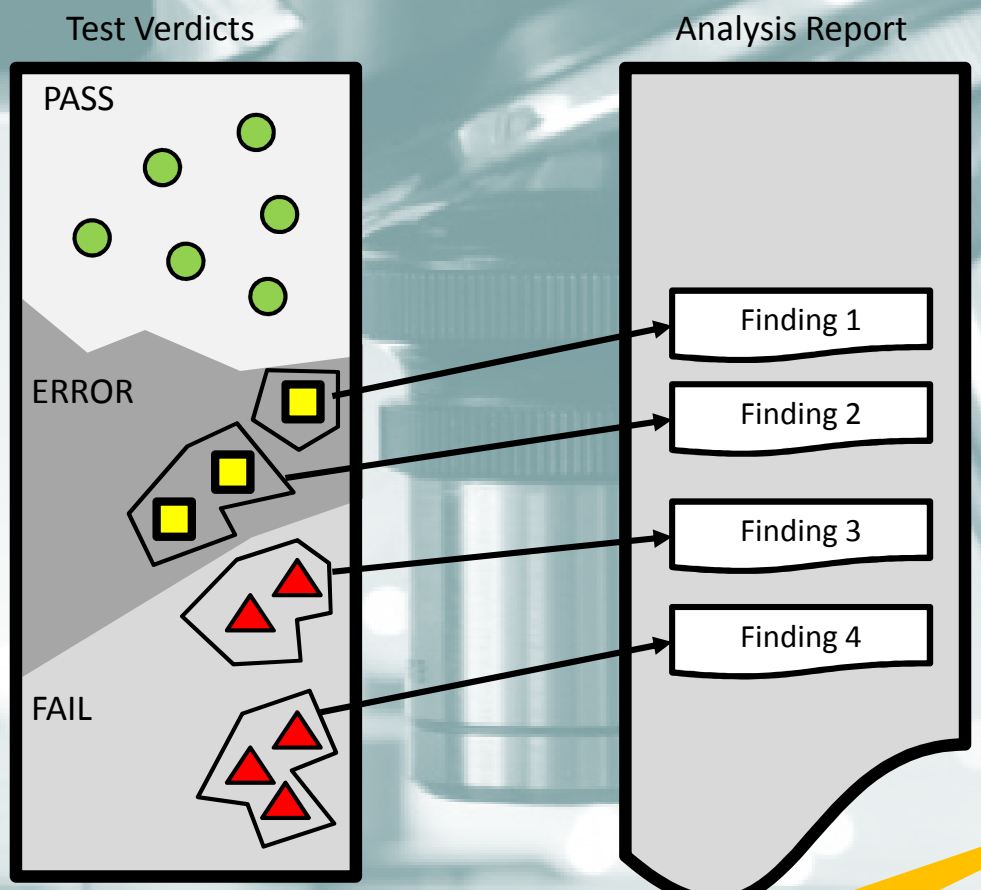
VS-Application Process:

1. Install and Configure VS and Tool.
2. Check „Installation and Configuration“.
3. Initiate Application Protocol.
4. Run the Test Suite on the Tool.
5. Check „Correctness and Completeness“ of Test Run.
6. Analyze Test Results.
7. Check „Correctness and Completeness“ of Analysis.

For every test with verdict FAIL or ERROR:

- Analysis Result: is / is not finding
- Explanation for result.

Analyzing Test Results



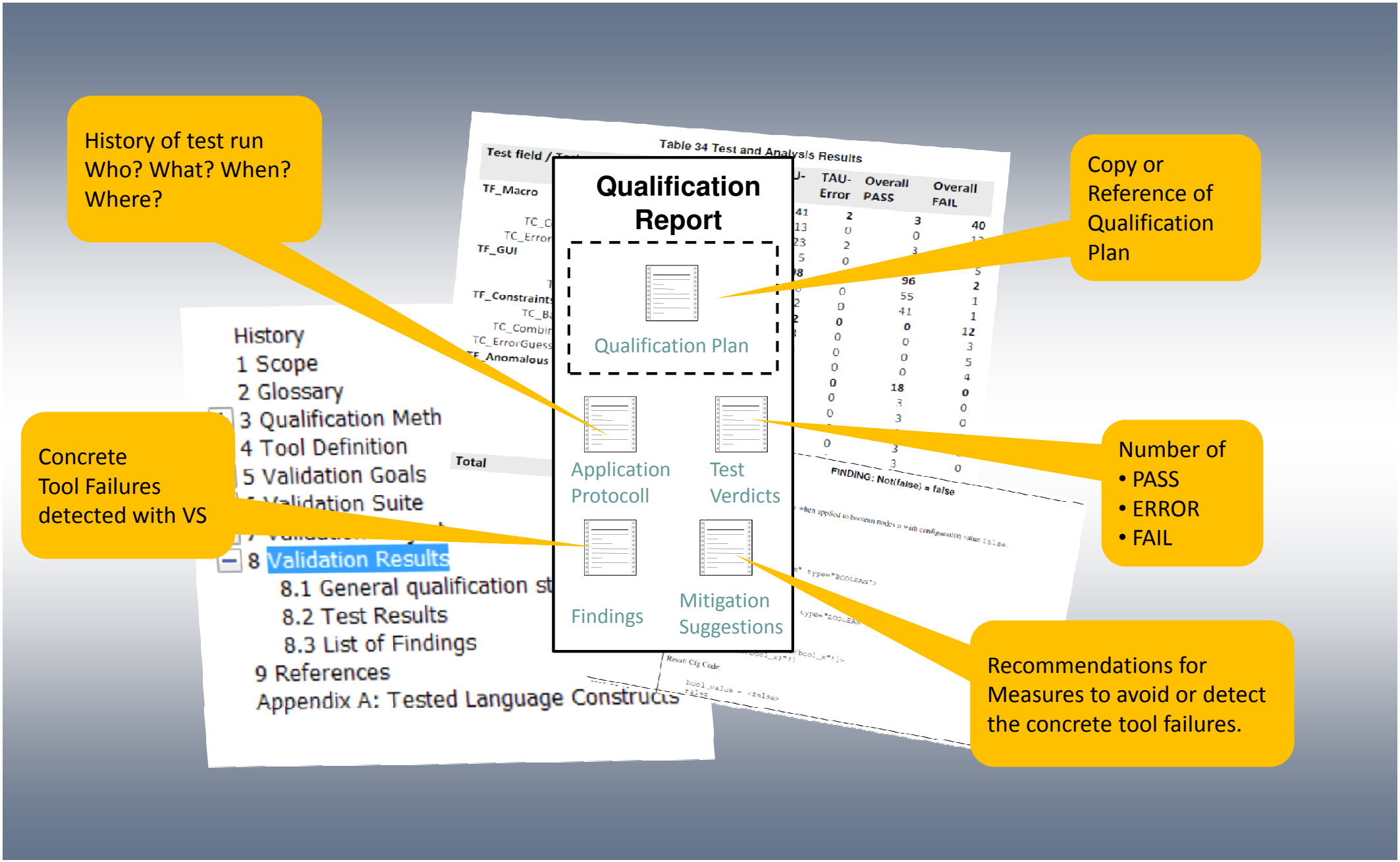
- Task:
  - Find an Explanation for every test with verdict ERROR or FAIL.
- Challenges:
  - diagnosis problem (many reasons)
  - high effort!
  - bug-clones (same bug detected with many tests)

Validas Experience:
Generating 10 million tests is easy,
analyzing 1000 FAILs is not!

Tool Validation

1. Introduction
2. Validation Process
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. Apply the VS
7. **Write Tool Qualification Report**
8. Vision: Customizable Qualification Kits

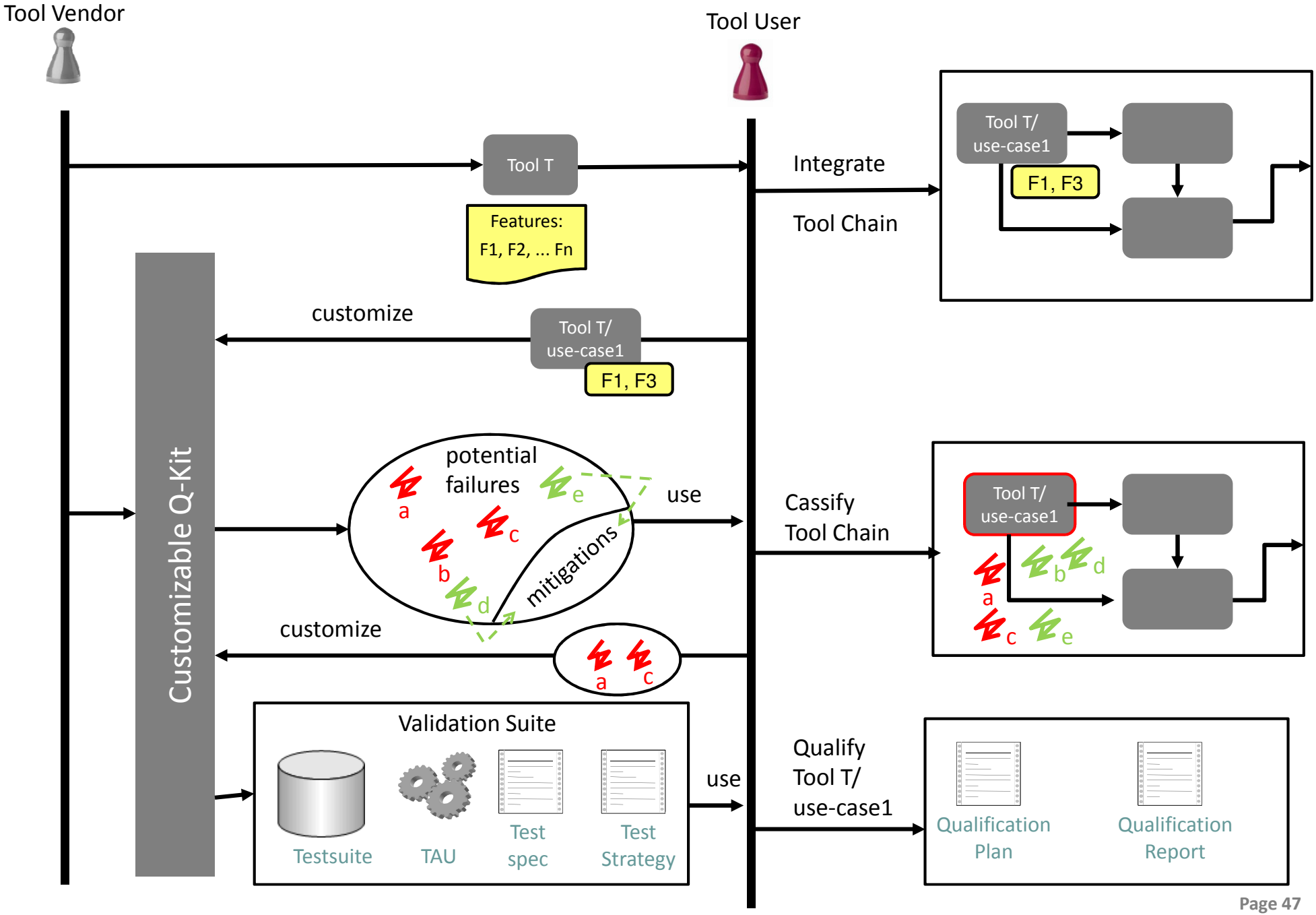
Qualification Report



Tool Validation

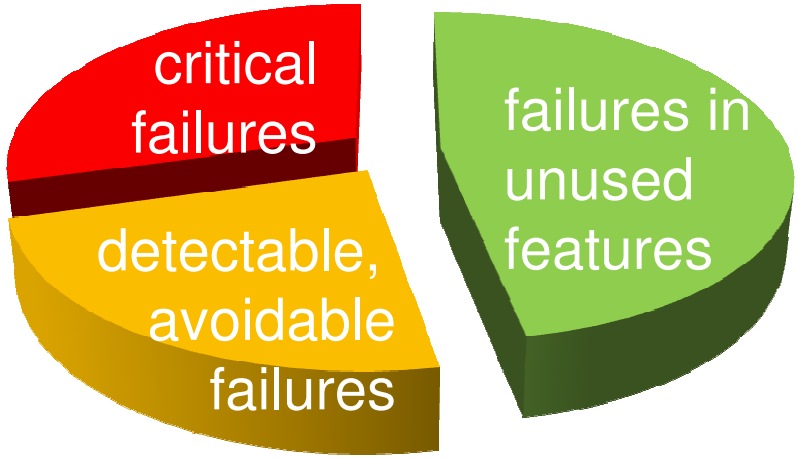
1. Introduction
2. Validation Process
3. Write Tool Qualification Plan
4. Develop Validation Suite (VS)
5. Verifiy and Validate VS
6. Apply the VS
7. Write Tool Qualification Report
8. **Vision: Customizable Qualification Kits**

Scenario for Customizable Q-Kits



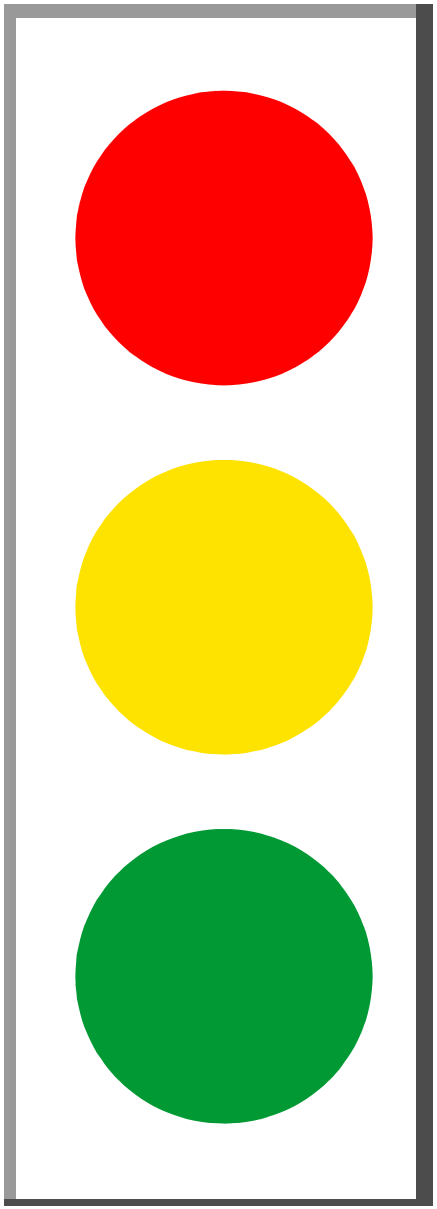
Challenges for Cust. Q-Kits

- Which tests do I require to qualify my use-cases?
  My use-cases have a unique combination of features!

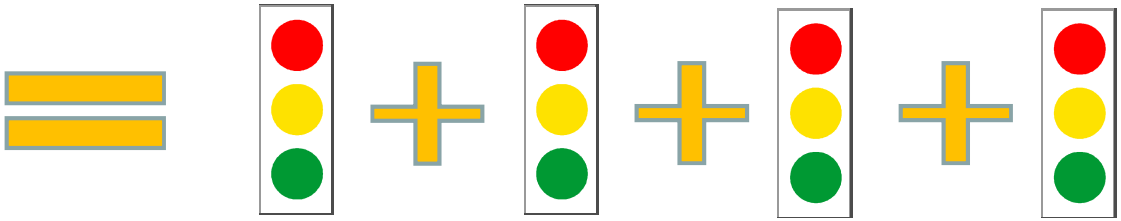


- Q-Kit Tests with flexible “reference values” needed.
- Qualification kit contains many test cases (e.g. Supertest)
  - Do they test the right features for **my** use-cases?
  - Would they detect the critical failures in my tool chain?
- Tracing needed: pot. tool failures <-> tests
- Can I extend the test suite to cover **my** special tool failures?

Qualification Status

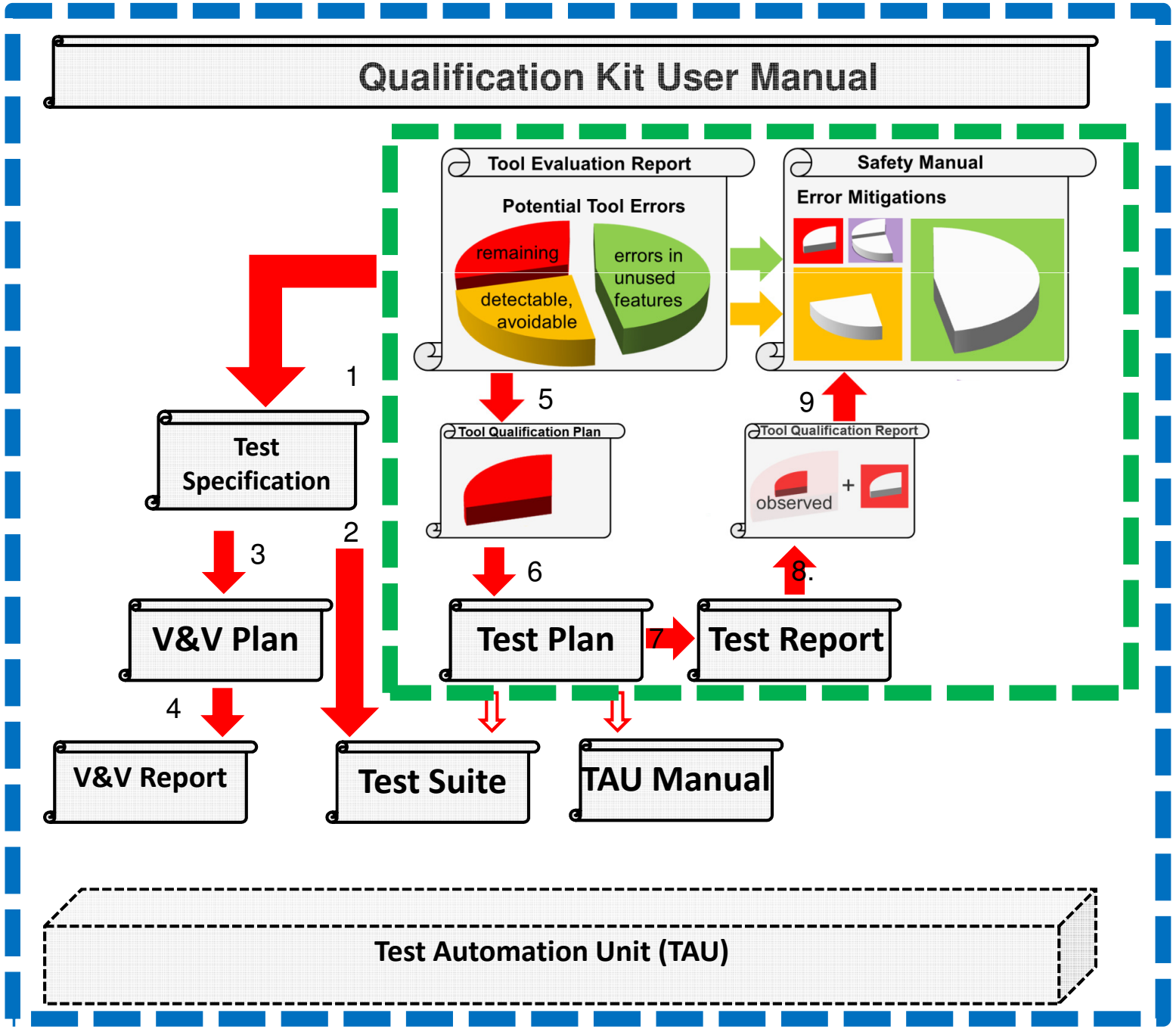


- Tools can be assigned a qualification status:
  - RED: Tool cannot be used safely
  - YELLOW: Tool can be used with constraints
  - GREEN: Tool can be used without constraints
- Qualification status also works for tool features:



- For example
  - Feature 1 (not qualifiable):
  - Feature 2 (not testable, but usable with constraints):
  - Feature 3 (testable):

Using a Qualification Kit



n. Legend:

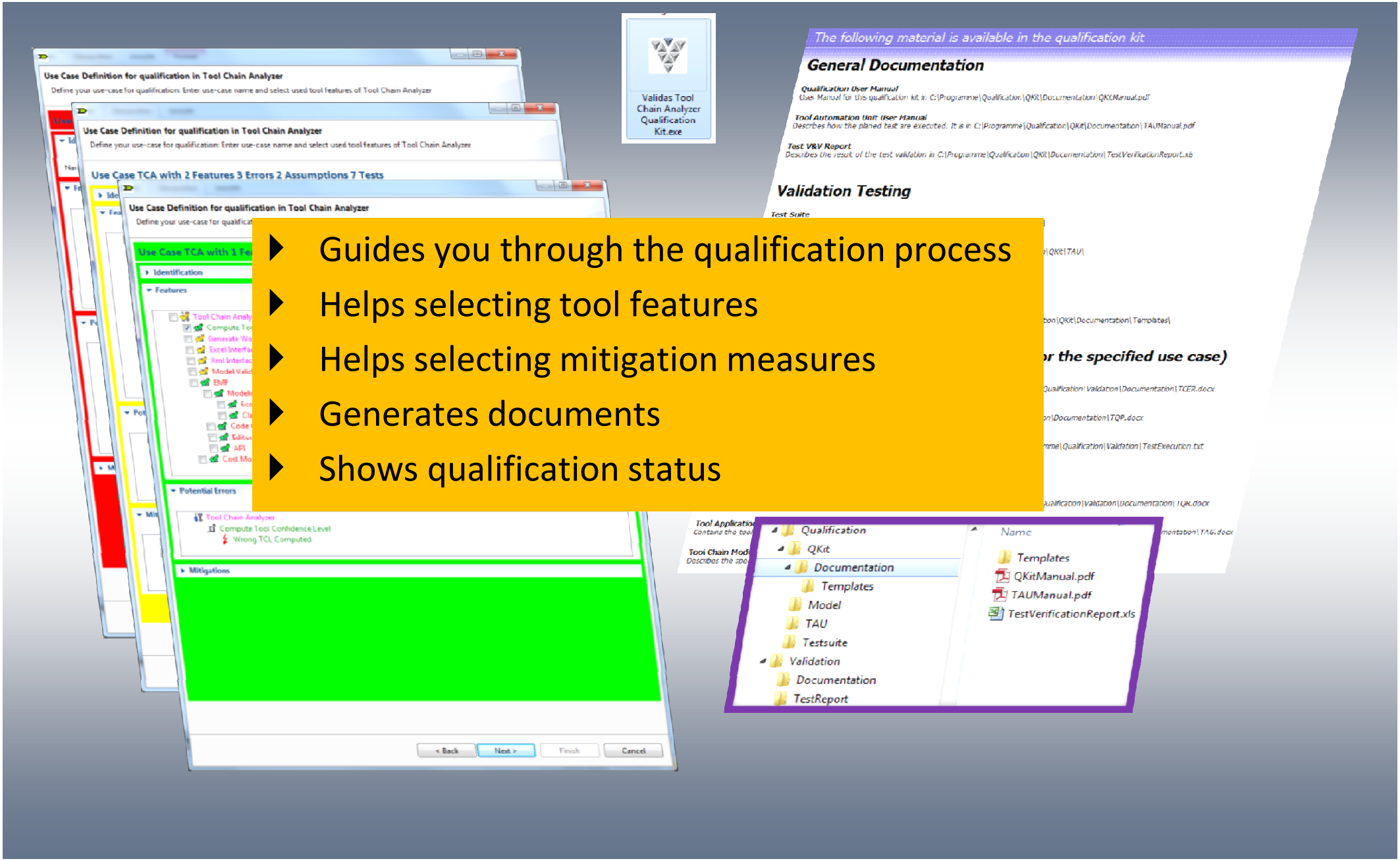
Sequence

Requires

Q-Kit

User Specific

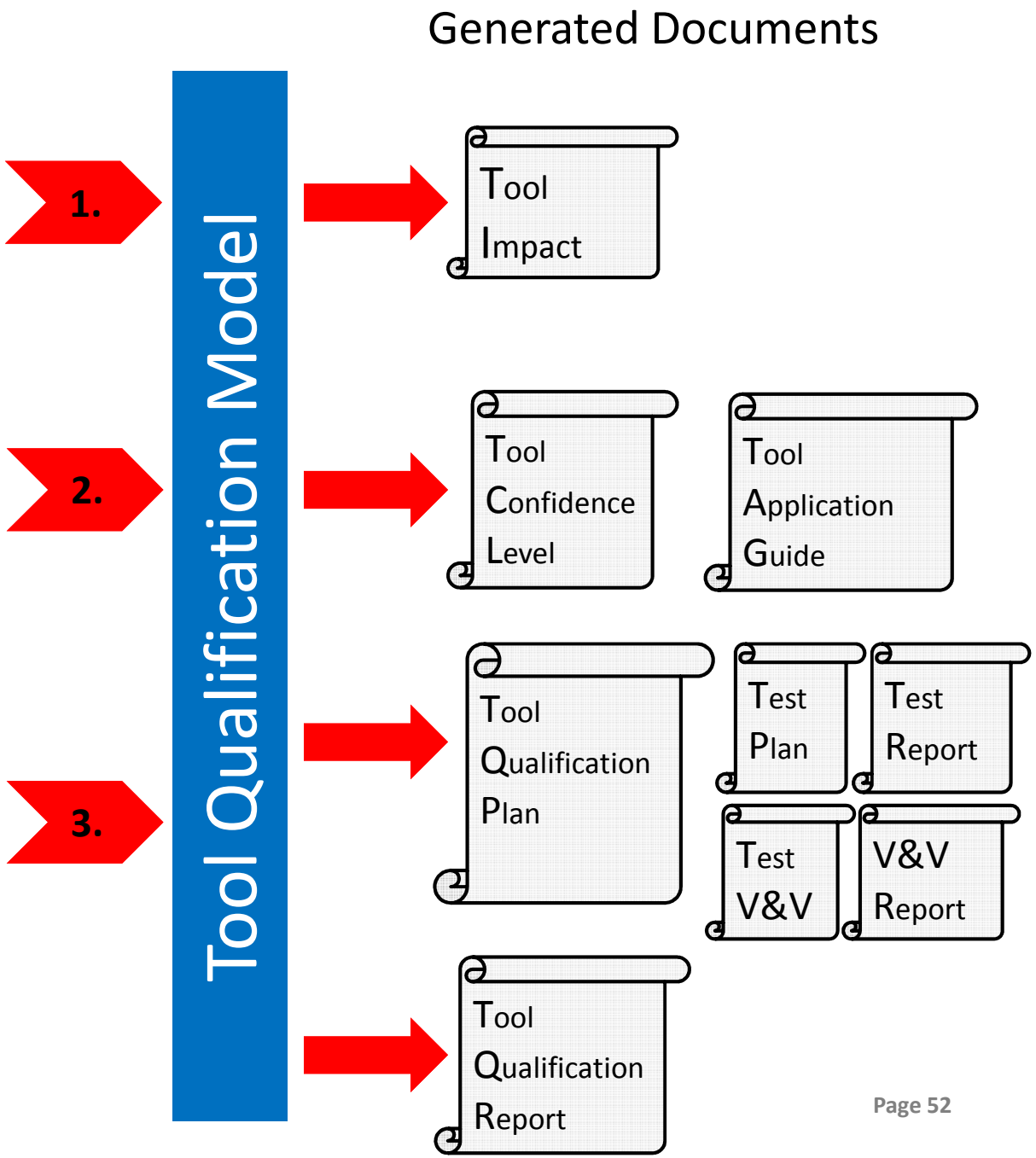
Qualification Tool



- Guides you through the qualification process
- Helps selecting tool features
- Helps selecting mitigation measures
- Generates documents
- Shows qualification status

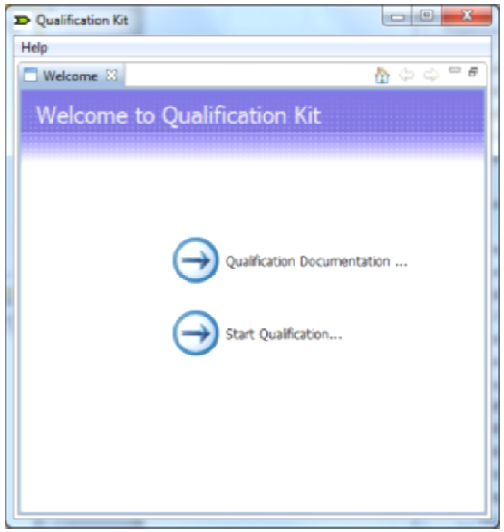
Model-Based Tool Qualification

1. Tool chain model
   - use-Cases,
   - tool features,
   - artifacts,
   - documentation
2. Classification model
   - potential failures
   - checks & restrictions
   - documentation
3. Qualification model
   - cost optimization
   - planning
   - tests:
     - identification,
     - tracing to pot. failures
     - test strategy,
     - test plan
   - documentation



Qualification Kit Overview

- Q-Kit conforms to
  - ISO 26262
  - IEC 61508
  - DO-178 (TQL-4 and TQL-5)
- Q-Kit is based on a model (flexibility)
- Q-Kit is extensible
- Features have modular qualification status
- Process as follows:
  1. Select tool features
  2. Select applicable mitigations
  3. Generate required documents
  4. Execute required tests
  5. Finalize documents
- Qualification process is tool supported



Conclusion and Outlook

- Tool Validation provides confidence, but has high costs!
- Tool Chain Analysis may avoid tool qualification.
- Customizable Q-Kits may
  - reduce costs for tool user.
  - generate profit for tool vendors.
  - be built using a model-based approach.
- Emerging Trends
  - Tools developed according to safety standards, e.g. DO-330.
  - Formally verified tools seem to be within reach: Amazing results: L4.verified, CompCert.