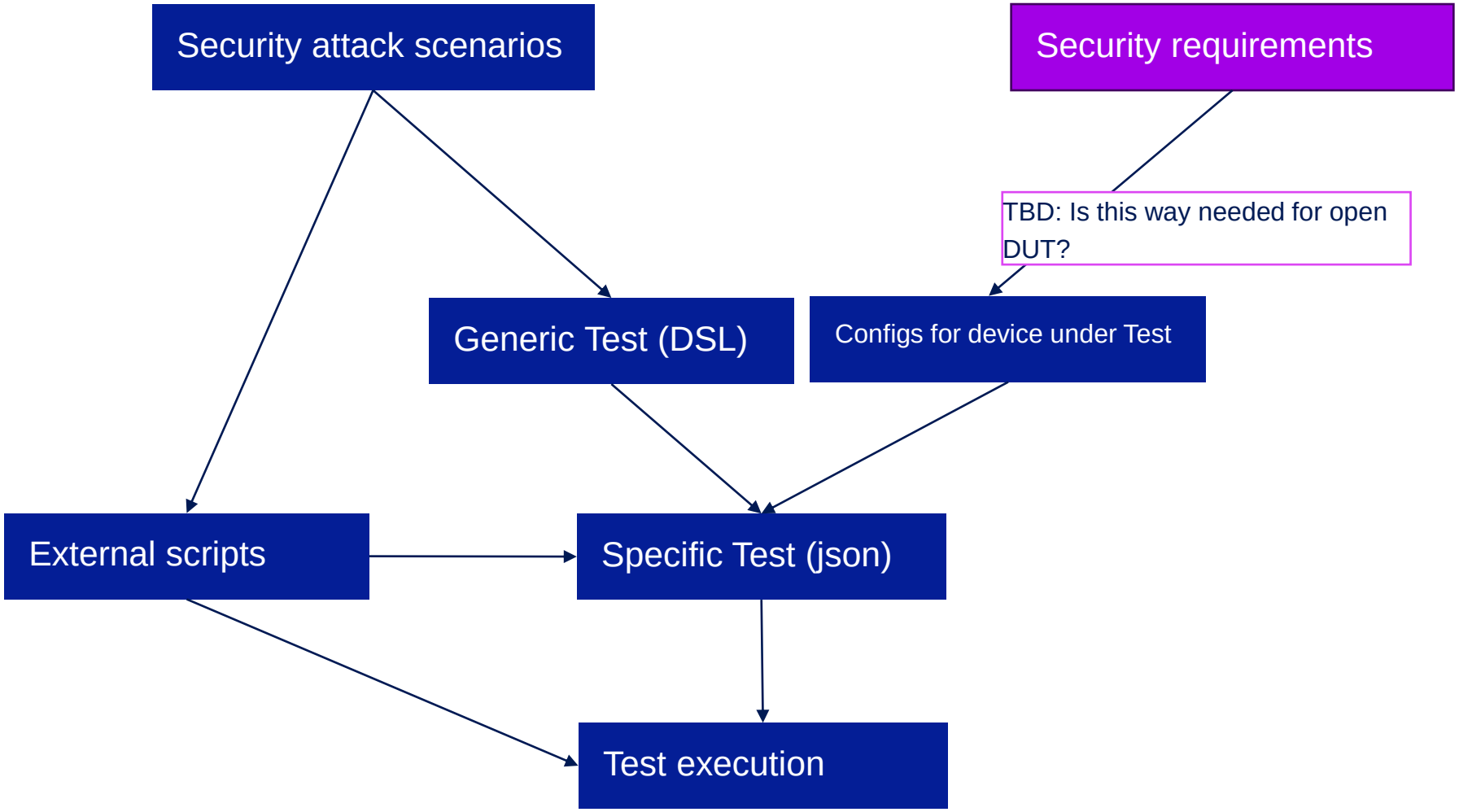
Test-Workflow Open DUT

SKyBT testdesign and test execution workflow

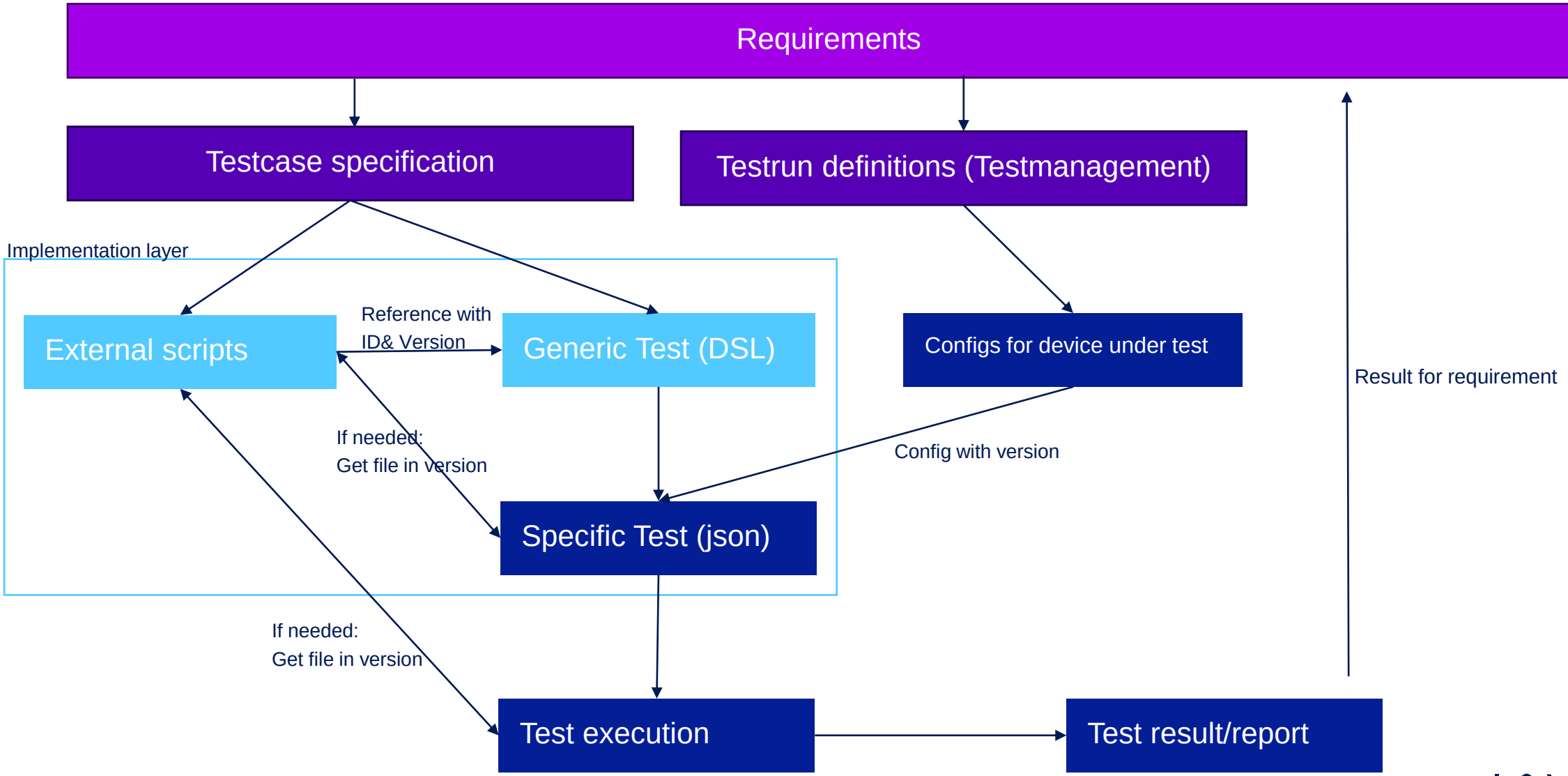
February 2024

Joachim Ehbrecht, Christian Claus, Volker Weck

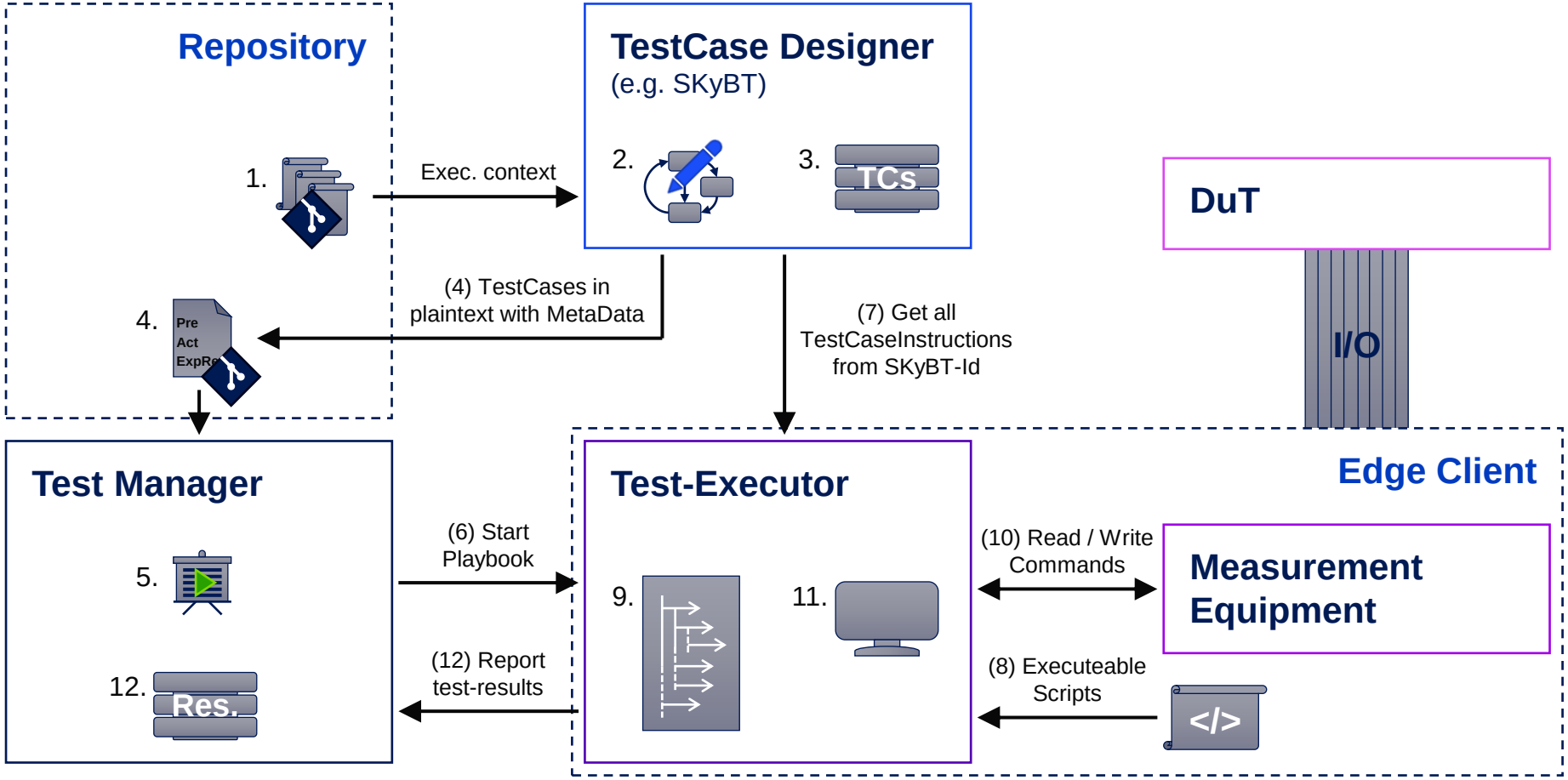
Testcase workflow



Testcase workflow



Components in testing process



Components in testing process

TEST-DESIGN

- The *ExecutionContext* ***(1)*** is the construction input for *generic TestCases* ***(2)*** at TestCase-Designer tool.
  - It might be the vehicle or Platfrom specification, like DBCs, or references to *ExecutableScripts* ***(8)***.
- The *generic TestCases* ***(2)*** uses a well defined *Syntax* and *KeyWords* with unique codes.
  - *Generic TestCases* are parametrizable and can attach different execution context.
- The TestCase-Designer tool generates / updates the *specific TestCases* ***(3)*** *–database* with concrete parameter values, which are ready for execution.
  - Each TestCase contains a version.
- The TestCase-Designer tool exports the *TestCases* ***(4)***, which are not directly executeable with an execution engine, but might be executed manually e.g. at a HiL or in a vehicle.
  - They have attached the TestCases‘s SKyBT-Id, e.g. as meta-data.
  - They needs to be stored in a *Repository*, to get versioned.

Components in testing process

TEST-MANAGEMENT

- The TestManager is able to assemble the executed *TestLists* ***(5)*** as playbooks for a specific test period of the DuT(s), to test specific HW-/SW-versions. This might be a simple list of SKyBT-Ids.
- The *Playbook is started* ***(6)*** from the TestManager), which starts the execution process at the Test-Executor on the Edge-Client.

TEST-EXECUTION

- The Test-Executor asks for *TestCaseInstructions* ***(7)*** from *specific TestCases*-database, with available *SKyBT-Id* (from processed *TestList*) and for *ExecutableScripts* ***(8)*** via reference on certain Instructions.
- The Test-Executor provides the *KeyWord-Code-Interpreter* ***(9)***, which can be seen as a smart switch-case construct, that addresses *Read/Write Commands* ***(10)*** to the MeasurementEquipment.
  - The MeasurementEquipment is a synonym for all hardware- and tools (e.g. Datalogger, Video-Grabber, Vector-HW, CANoe, RemoteControl, etc.), which communicates directly with the DuT, e.g. via I/O-channels.
- The Test-Executor can *Monitore* ***(11)*** the MeasurementEquipment to handle exceptions, e.g. for recovering / healing, if possible, or to abort the test-run.

Components in testing process

REPORT-MANAGEMENT and ERROR-TRACKING

- The Test-Executor reports the ***test-results (12)*** from executed Playbook back to the TestManager, to release the requested HW-/SW-versions, to be able to finally start production.
  - Found issues needs to tracked for retesting-cylce.

How detailed should a TestCase be described?

- A TestCase can be specified as a list of buzzwords (This is more a command list than a TestCase).
  - The behavior of the buzzwords needs to be detailed explained anywhere:
    - The Customer specify what is tested and want‘s the TestCases transparently commited, due to enable Hardware- and Software versions during test-management process → It is very hard to address issue-tickes to DuT‘s hardware components, if the detailed TestCase isn‘t known. → Which part of the Script is responsable for the issue?
    - Error tracking is been made very hard to the AutomationDeveloper → A written Script must work for all changes on Execution-Engine. This will result in version conflicts. → This increases startup-time.
    - These TestCases are not human understandable, so they can‘t be used for manual testing.
  - The TestCase intelligence must be translated to Scripts and the execution engine, which mixes the roles TestDesigner and AutomationDeveloper (Everyone needs to know at least everything):
    - The TestDesigner (role) needs to know the execution engine and must have programming experiences.
    - The AutomationDeveloper needs to know what exactly needs to be implemented for communication with the MeasurementEquipment → Detialed test informations are needed.

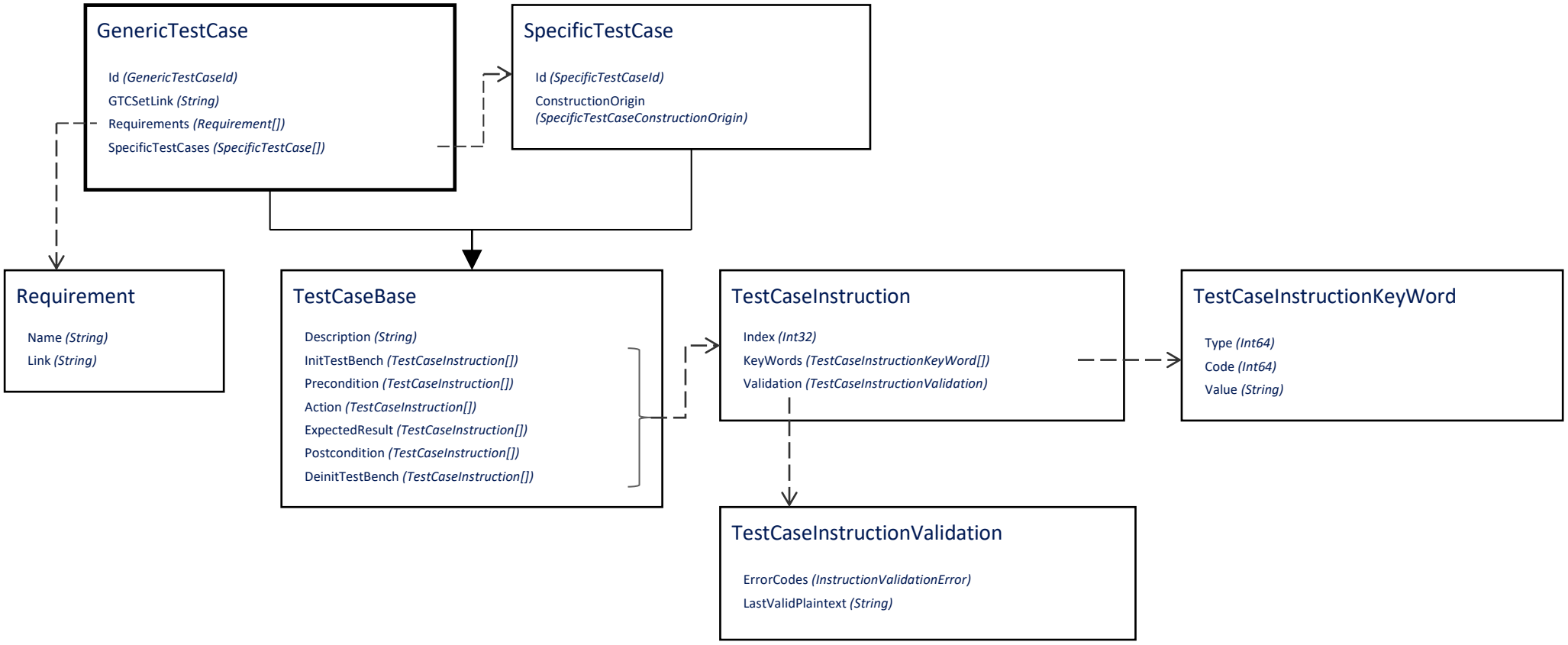
How detailed should a TestCase be described?

- A TestCase can be specified as a list of Instructions, which consit of KeyWords arranged in a transparent Syntax.
  - The KeyWords have unique codes, which allows smart parsing with a KeyWord-Code-Interpreter, that can be prepared for any TestAutomation (Test-Executor) of any programming language.
  - The AutomationDeveloper (role) did only need to know which KeyWords (and their related Codes) are available and how the Syntax is defined → TestCase‘s content is irrelevant.
  - Customer‘s Requirements can be attached to TestCase artefacts, which allows a simpler issue-tracking.
  - It is possible to use paramters instead of concrete KeyWords:
    - Allows generating a set of specific TestCases from a defined- and changeable parameter table.
    - Allows defining a single generic TestCase with different ExecutionContext (Vehicle, Platform, etc.).
  - The Syntax and KeyWords can be designed to start Macros and/or Scripts, if needed.

How detailed should a TestCase be described?

- When should use Scripts with plain-code?
  - If there are small hardware-dependent functionalities needed to be stimulated.
  - When detailed hardware-dependet knowledge is needed, which the AutomationDeveloper doesn‘t need to know, or shouldn‘t know.
  - Be careful of using:
    - The code-dependent execution environment needs to be available on Execution-Engine.
    - Security risks needs to be weight and access to be restricted → We think in most cases the Customer will not allow handling plain-code to be executed (like vehicle attacks) from external execution engines, which are not under Customer‘s control.
- It is recommended, to avoid using Scripts.

TestCase structure (part)



TestCase structure (part)

A `GenericTestCase` is a generated TestCase definition, which can contain parameter. It contains at least one `SpecificTestCase`.

A `SpecificTestCase` is a generated TestCase definition, which contains concrete values. This makes it directly executeable with a Test-Executor (TestAutomation).

A `TestCaseBase` contains the collections of „Instructions“ for the TestBlocks „Precondition“, „Action“ and „ExpectedResult“, etc.

- A `TestCaseInstruction` is a single line of „KeyWords“ within a „TestBlock“, which describes an atomically executeable function for the Test-Executor (TestAutomation).
  - A `TestCaseKeyWord` is a firmly defined word (or word combination with wide-spaces), which relates to a unique identification „Code“. KeyWords are currently defined for three different classes, divided into several types:
    - „*StaticKeyWords*“ – are statically defined in the SKyBT environment. The contained types are e.g. *Condition*, *Operator*, *Value*, etc.
    - „*CustomKeyWords*“ – are dynamically createable with the SKyBT-Editor. The contained types are *Elementary* and *Macro*.
    - „*ExternalKeyWords*“ – from external sources. The contained types are *ECU-Data-Interface* and *Vehicle-Bus-Data-Interface*.

Conclusion

- If the TestDesing-layer is established, you earn more flexibility, which pays off the invested work:
  - Allows Requirements handling, which focuses the TestCase and prevents loosing focus during implementation.
  - Possibility for organized review of TestCases.
  - Reusability of TestCases or TestCase-parts with parametrization and a variable execution-context.
  - Special designe-features, like StateMachine-based design, which allows pictured discussions with the Customer.
  - Etc.
- Error-tracking and bug-fixing needs to be possible at Execution-Engine, without the need of understanding what Tests are running on it.
- Results from Testautomation (Test-Execution) also needs to be processed back in a loop:
  - Results needs to be mapped against Requirements, to enable Hardware- and Software versions of the DuT.
  - Issue-tracking and restart of related TestCases is needed.
  - Handle the large amount of data, which comes back from automation, for a part automated post evaluation.
- Keep focus on Customers perspective: Check use-cases; Check usability of own tool-chain and try to integrate in Customer‘s tool-chain; Note that there exist roles in the test-process with more than one coding-person; etc.

Kontakt

Joachim Ehbrecht
IAV GmbH
Carnotstraße 1, 10587 Berlin
Telefon 0512 29125629
joachim.ehbrecht@iav.de
www.iav.com