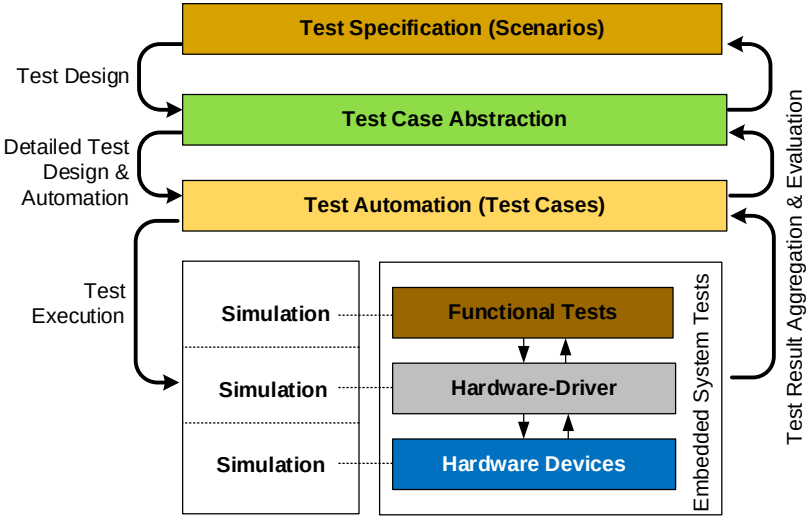
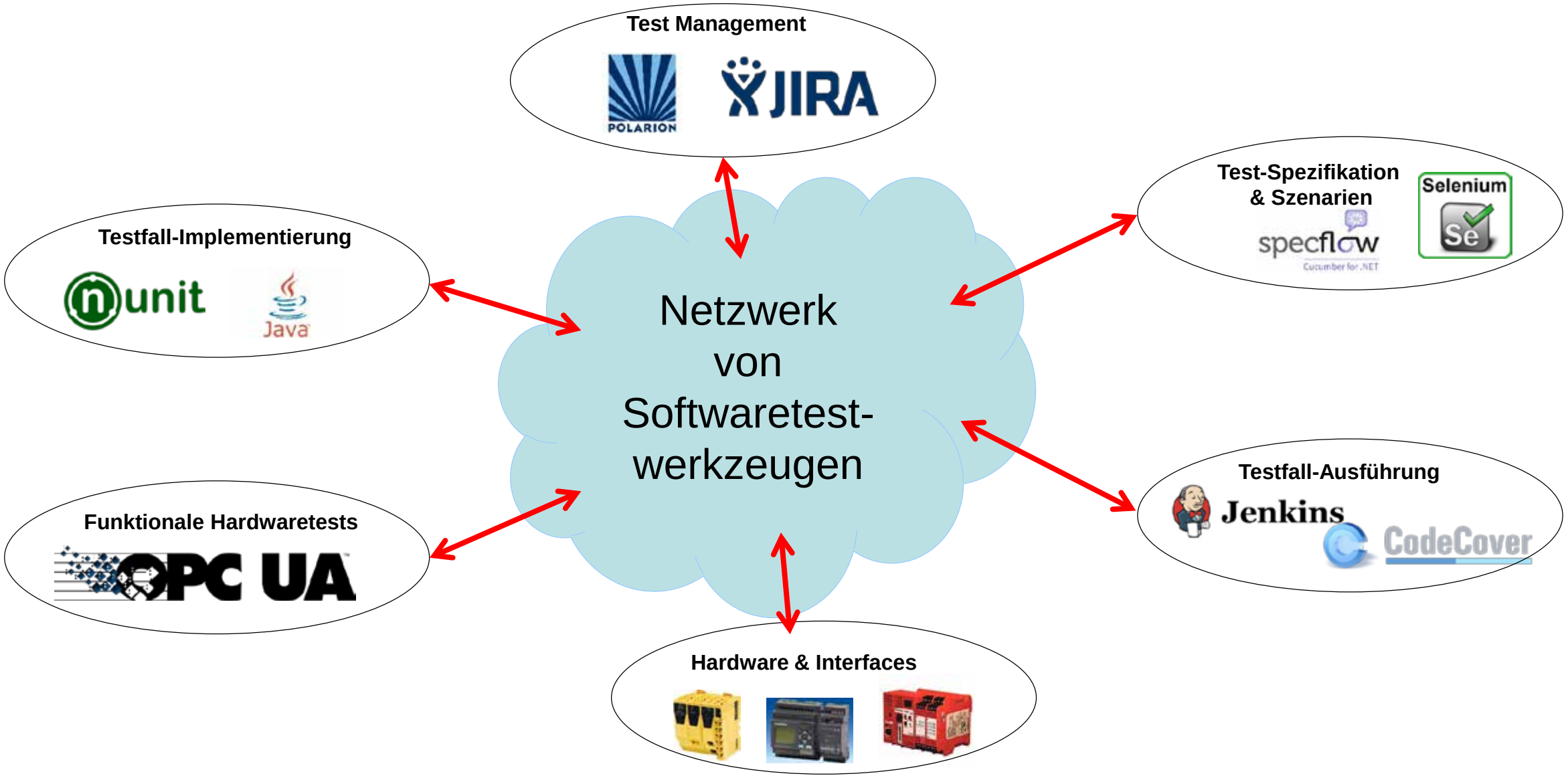
Flexible Testautomatisierung für einen modularen Systems Teststand

- Testfallbeschreibung durch Fachexperten.
- Effizientes Mapping von abstrakten Testszenarien zu ausführbarem Test Code.
- Automatische Testfalldurchführung und Reporting.
- Flexible Werkzeugkette für Testautomatisierung.



Heterogene Werkzeuglandschaft



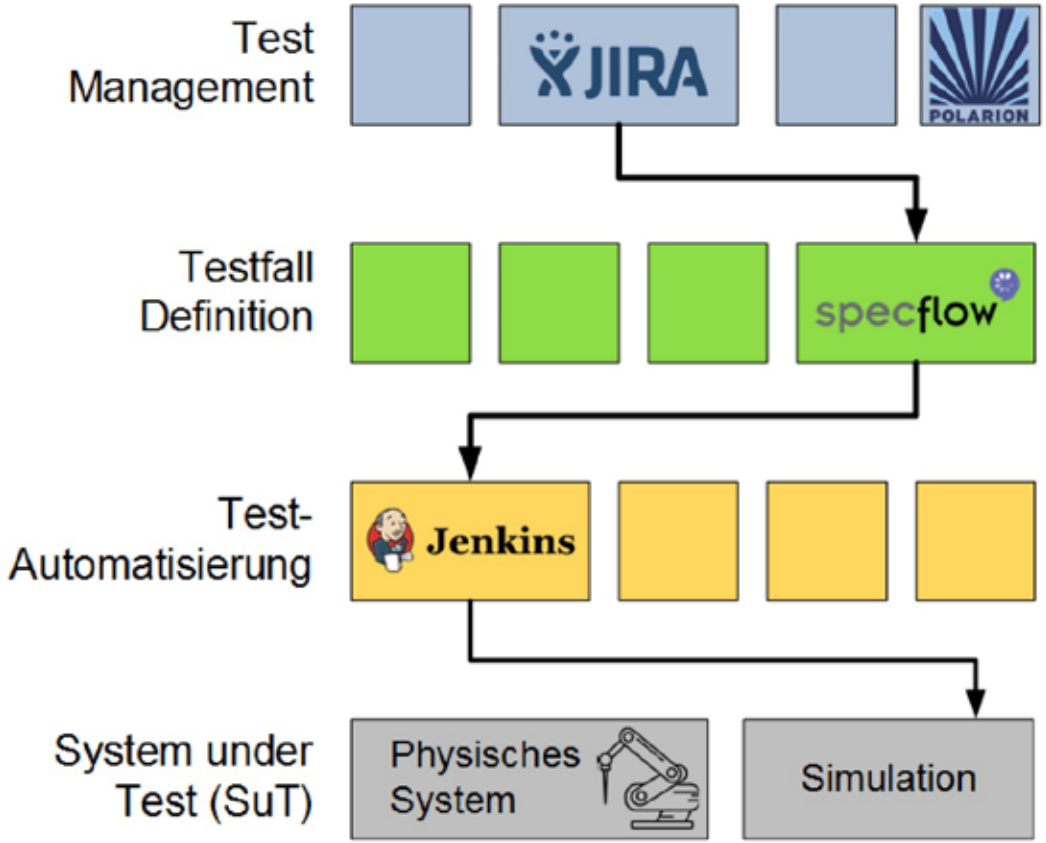
Herausforderung und Ziel

- **Herausforderungen**
  - Heterogene Werkzeuglandschaft beim Kunden (manuelle Integration).
  - Automatisches Testen nur in Teilbereichen möglich.
  - Mangelnde Flexibilität / Erweiterbarkeit etablierter Werkzeugketten.
  - Software-/Testexperten erforderlich.

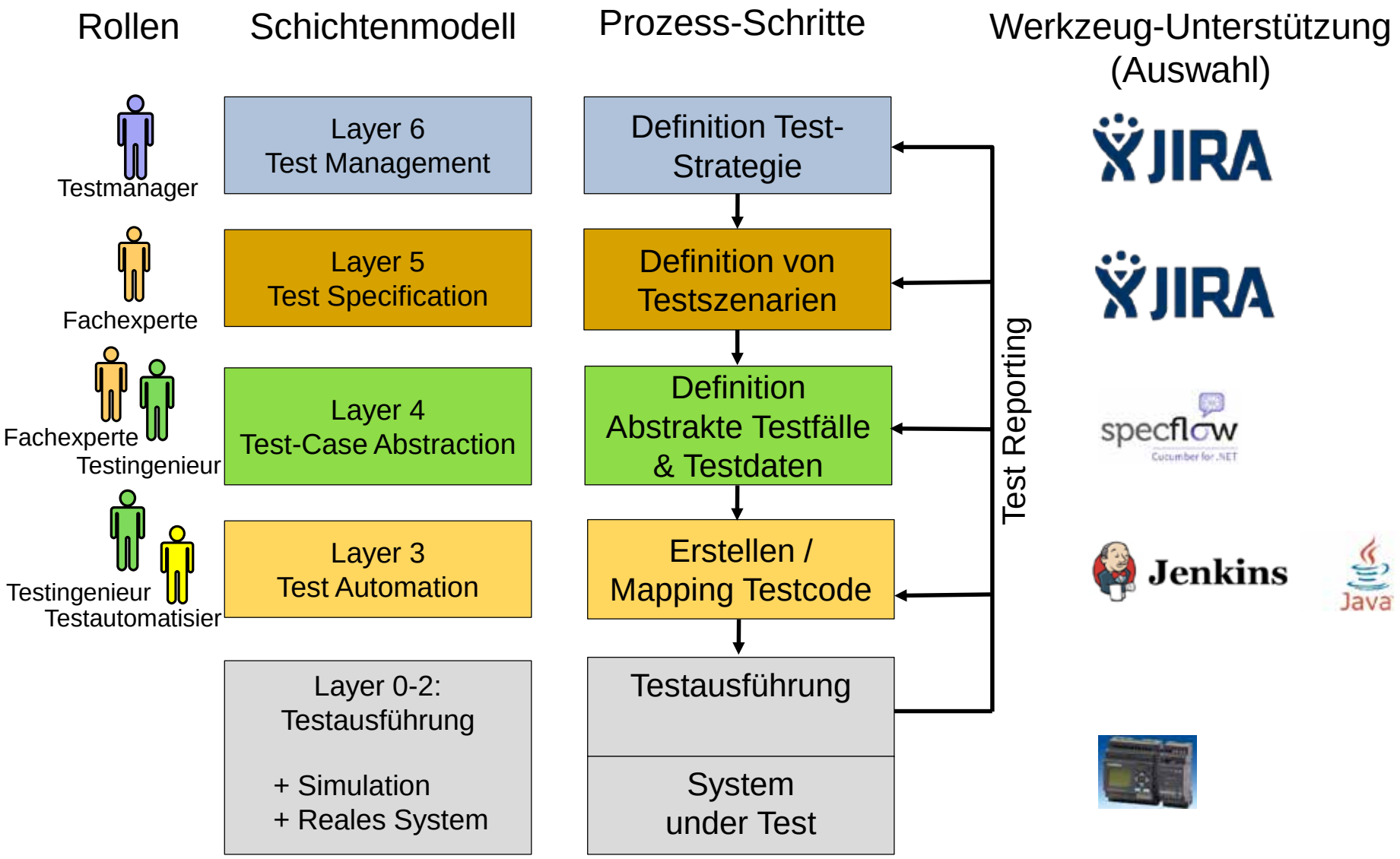
- **Ziel „Modularer Embedded Teststand“**
  - **Automatisches Testen** von Automatisierungsanlagen.
  - Erstellung von Testszenarien durch **Fachexperten**.
  - Flexibles und konfigurierbares **Testautomatisierungsframework**.
  - Unterstützung **unterschiedlicher Werkzeuge**.

Lösungsansatz

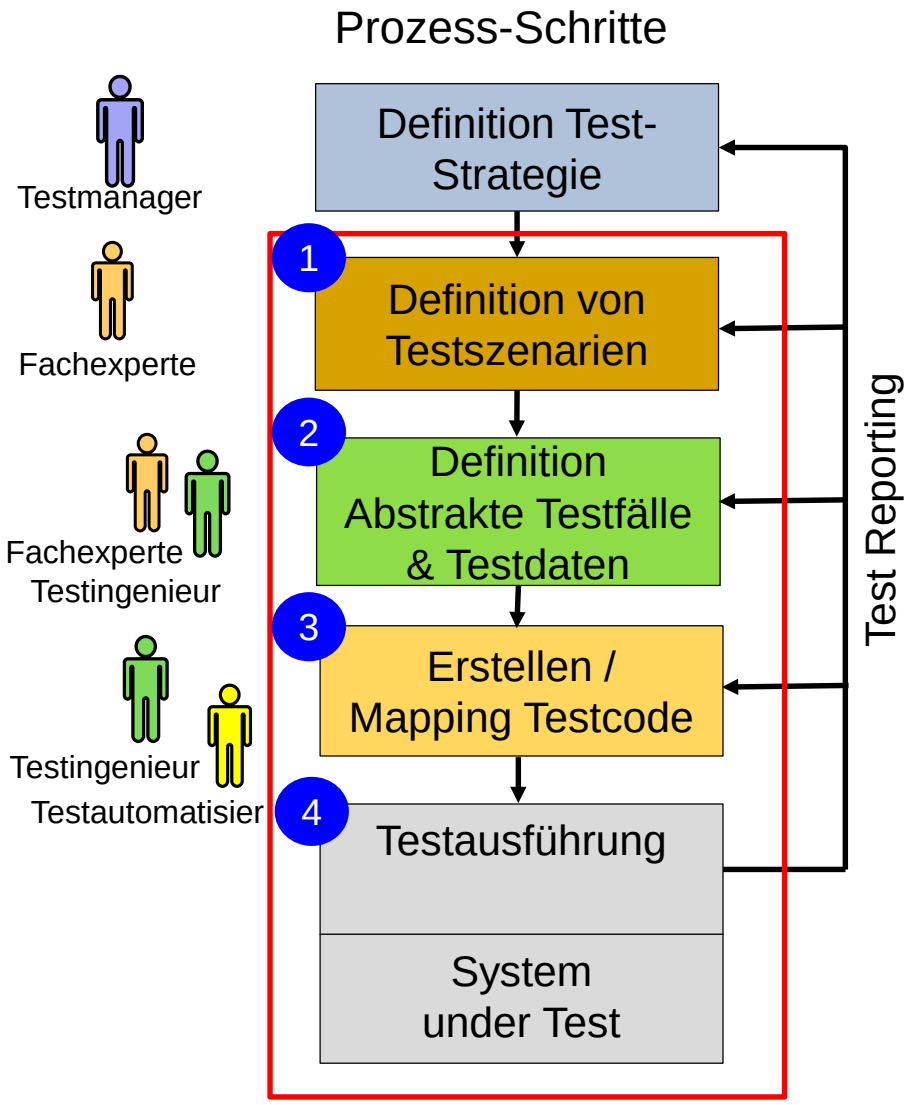
- **Testautomatisierungsframework**: Schichten Modell
- **Werkzeugbausteinkasten**
  - **„Best-of-breed“** auf unterschiedlichen Schichten der Testautomatisierung.
  - **Fachexperten** für spezifische Aufgaben.
- Definierte **Schnittstellen** zwischen einzelnen Schichten / Werkzeugen.
- Flexible Konfiguration von **Werkzeugketten**.
- Effizientes **Füllen von Testautomatisierungslücken**.



Vision: Automatisierter Testprozess



Use Case & Prototyp



1. **Definition von Testszenarien**
   - Use Cases, User Stories
   - Werkzeug: Jira
2. **Definition abstrakter Testfälle & Testdaten**
   - Keyword-Driven Test: *Given*, *When*, *Then*
   - Werkzeug: Jira Plug-In, SpecFlow
3. **Erstellen / Mapping Testcode**
   - Existierender Testcode à Mapping von Keywords – Java-Code
   - Fehlender Testcode à Auftrag an Testingenieur.
   - Werkzeug: Jira Plug-In, Java Repository
4. **Testausführung & Reporting**
   - Jenkins: Simulation, Real-World System
   - Reports für unterschiedliche Schichten, z.B. auf Testszenarien
   - Werkzeug: Jira Plug-In

1. Definition von Testszenarien

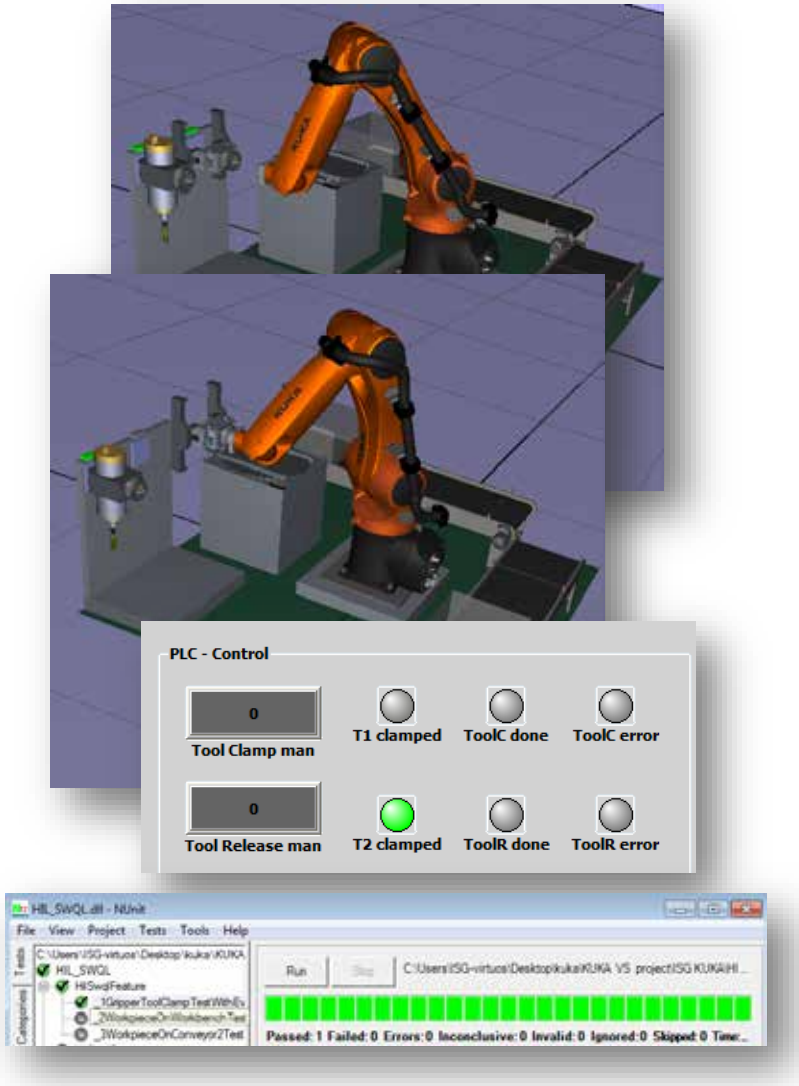
Szenario: Gripper Tool Clamp Test

1. Der Roboterarm fährt zuerst auf die Initialisierungsposition, damit immer die gleiche Ausgangsposition vorhanden ist.

2. Im nächsten Schritt wird das zu bearbeitende Werkstück auf das Förderband 1 gelegt.

3. Nachdem das Werkstück auf dem Förderband vorhanden und für den nächsten Arbeitsschritt bereit ist, wird der Roboterarm das Greifer-Werkzeug klemmen.
   - **Überprüfung des Greifwerkzeugs.**
   - …

4. Der Testfall sollte grün aufleuchten wenn das entsprechende Signal vorhanden ist.

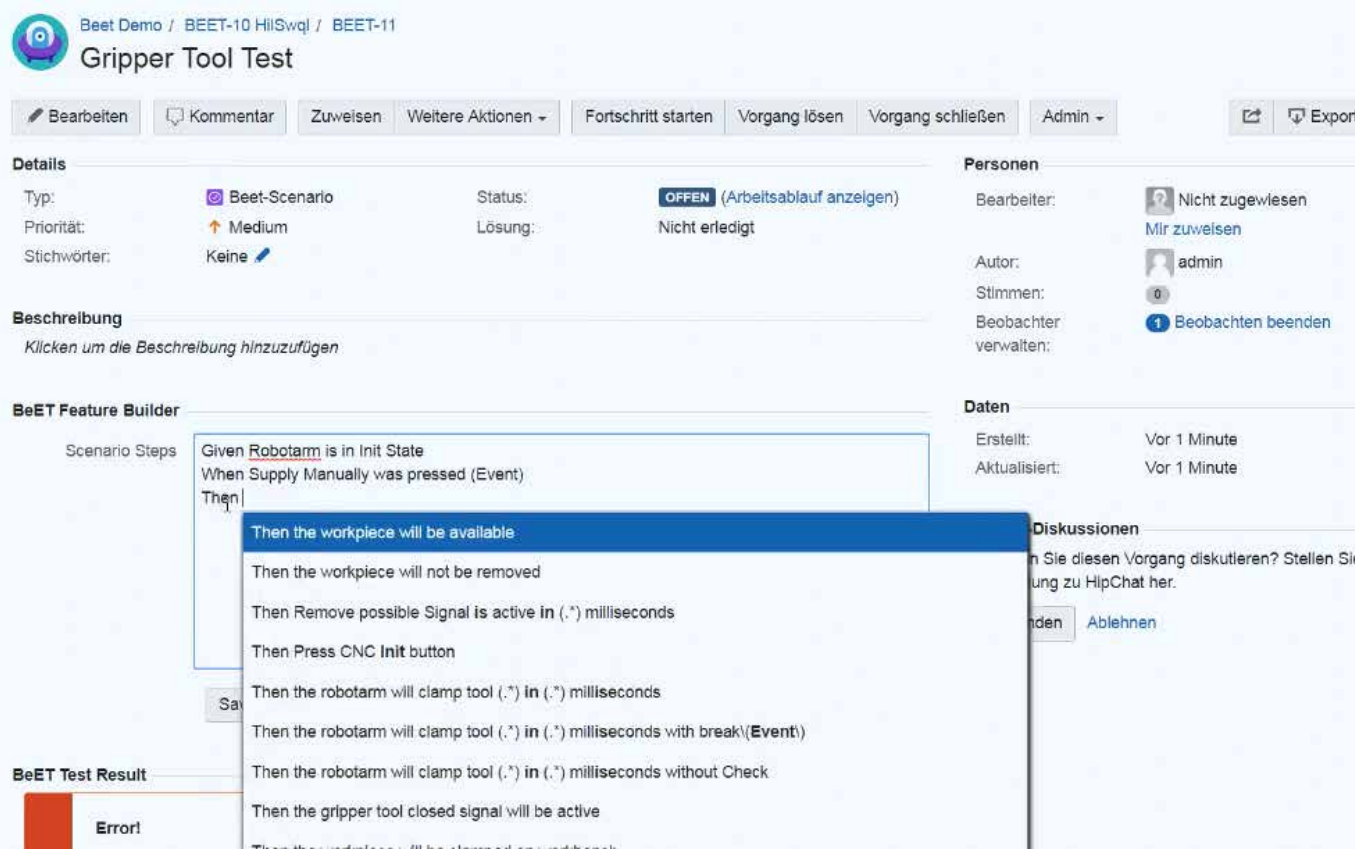


2. Definition Abstrakter Testfälle

Testsequenz mit Keyword-Driven Tests

Scenario: Test Gripper Tool Clamped Signal

- *Given* Robotarm is in Init State
- *When* Supply manually pressed (Event)
- *Then* Remove possible Signal is active in 4000 milliseconds (Event)
- …



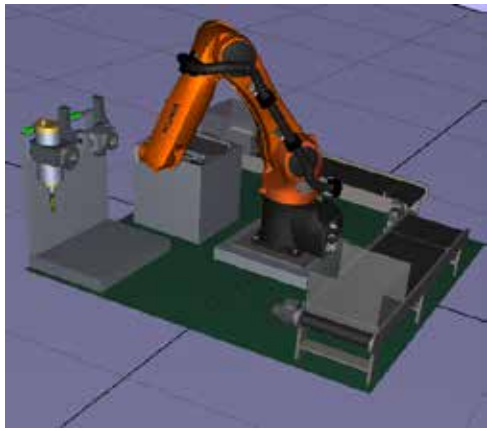
3. Mapping von Testcode

Mapping von Szenario zu Testcode

Beispiel: Test Gripper Tool Clamped Signal

- *Given* Robotarm is in Init State
- *When* Supply manually pressed (Event)
- *Then* Remove possible Signal is active in 4000 milliseconds (Event)

Mapping zu Testcode



Ausführung

```
/// <summary>
/// Function for writing log messages
/// </summary>
public static void Log(String Msg)
{
    File.AppendAllText(@"c:\path\file_hil.txt", Msg + Environme
}

/// <summary>
/// Function that compares two values, Soll and Ist.
/// Returns TRUE, if they are equal, and FALSE otherwise.
/// Asserts the result.
/// </summary>
public static bool AssertIntEqual(double Soll, double Ist, Stri
{
    try
    {
        Assert.AreEqual(Soll, Ist);
        Console.WriteLine("PASSED: " + Msg + " " + Soll.ToStrin
        return true;
    }
    catch
    {
        Console.WriteLine("FAILED: " + Msg + " " + Soll.ToStrin
        return false;
    }
}
```

4. Testausführung und Reporting

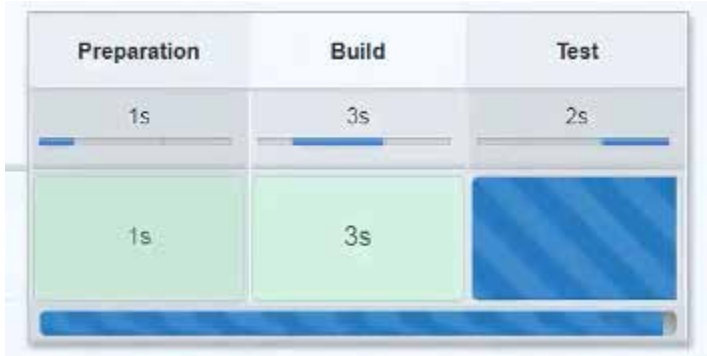
Automatische Ausführung und Reporting der Testfälle

```
/// <summary>
/// Function for writing log messages
/// </summary>
public static void Log(String Msg)
{
    File.AppendAllText(@"c:\path\file_hil.txt", Msg + Environment.NewLine);
}

/// <summary>
/// Function that compares two values, Soll and Ist.
/// Returns TRUE, if they are equal, and FALSE otherwise.
/// Asserts the result.
/// </summary>
public static bool AssertIntEqual(double Soll, double Ist, String Msg)
{
    try
    {
        Assert.AreEqual(Soll, Ist);
        Console.WriteLine("PASSED: " + Msg + " " + Soll.ToString() + " == " + Ist.ToString());
        return true;
    }
    catch
    {
        Console.WriteLine("FAILED: " + Msg + " " + Soll.ToString() + " <> " + Ist.ToString());
        return false;
    }
}
```

Testcode

Testfall-ausführung



nunit

Jenkins

Reporting

specflow
Cucumber for .NET

BeET Test Result

Name	Duration	Result
GripperToolTest	47.124	SUCCESS
Initialize	0.003	SUCCESS

JIRA

Flexible Testautomatisierung für einen modularen Systems Teststand

- Testfallbeschreibung durch Fachexperten.
- Effizientes Mapping von abstrakten Testszenarien zu ausführbarem Test Code.
- Automatische Testfalldurchführung und Reporting.
- Flexible Werkzeugkette für Testautomatisierung.