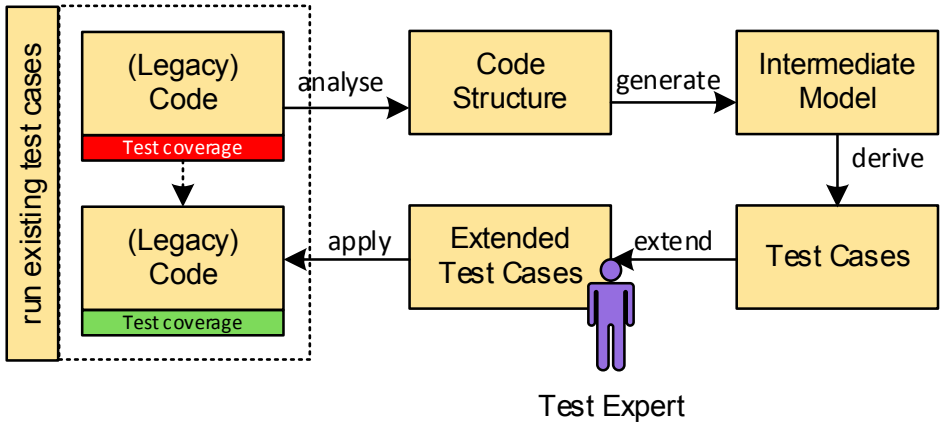
Semi-automatisierte Testfallgenerierung

- Unterstützung bei der systematischen Erstellung von (fehlenden) Software-Tests.
- Qualitätssicherung bei Migrationsprojekten durch wieder verwendbare Testfälle.
- Verwendung von Best-Practice Ansätzen.
- Sprachunabhängige Modellierung von Software Code.



Herausforderung und Ziel

- **Herausforderungen**
  - Unzureichende Tests von Legacy Code.
  - Unzureichende Mehrfachabdeckung von Software Code.
  - Unsystematisches Testvorgehen.
  - Keine oder mangelhafte Dokumentation von Software- und Systemtests.

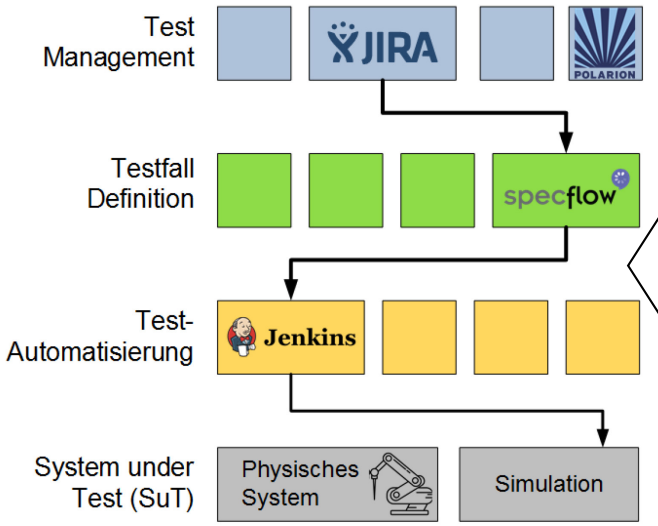
- **Ziel: Semi-automatische Testfallgenerierung**
  - Nachträgliche Ergänzung / Erweiterung von Software Tests.
  - Unterstützung von Migrationsprojekten (von Legacy Systemen zu migrierten Systemen).
  - Verbesserte Absicherung von Software Code durch Wiederverwendung von (generierten) Software Tests.

Simone Giertz

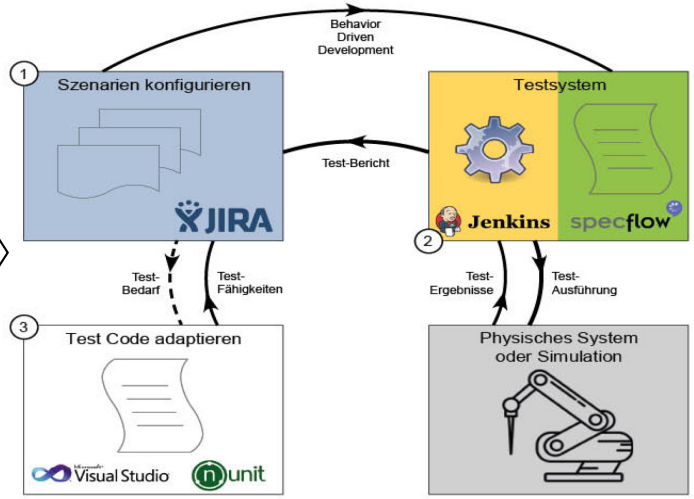
Symolbild

Test Automation in der Zukunft

- **Big Picture und Vision**
  - A. Test Automation Framework
  - B. Anwendungsfall
  - C. Verbesserung von Testfällen und Testprozessen

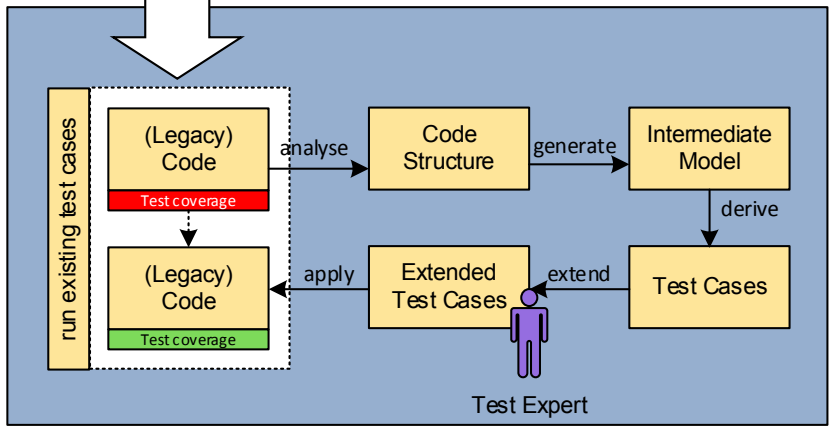


A. Test Automation Framework



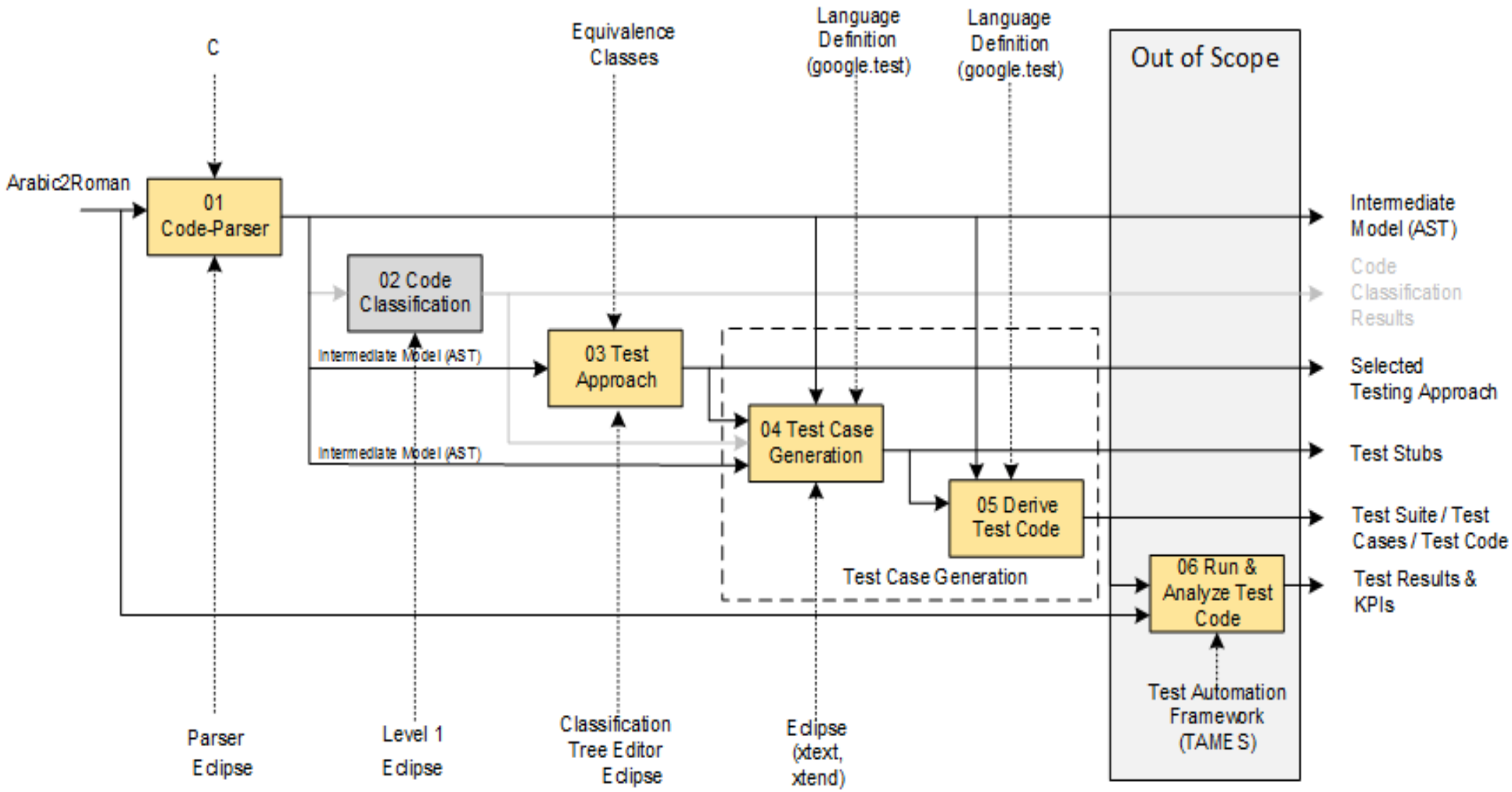
B. Application Use Case

- **Semi-automatische Testfallgenerierung**
  - Offenheit für unterschiedliche Testverfahren.
  - Offenheit gegenüber unterschiedlichen Test-Frameworks.
  - Verwendung von Best-Practice Ansätzen im Software Testen.
  - Sprachunabhängige Modellierung von Software Code.



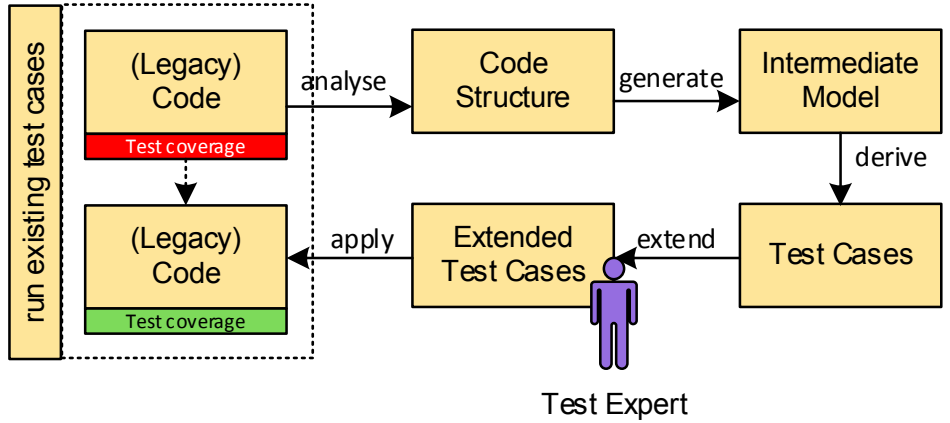
C. Improving Test Cases and Test Processes

Grundlegender Ablauf

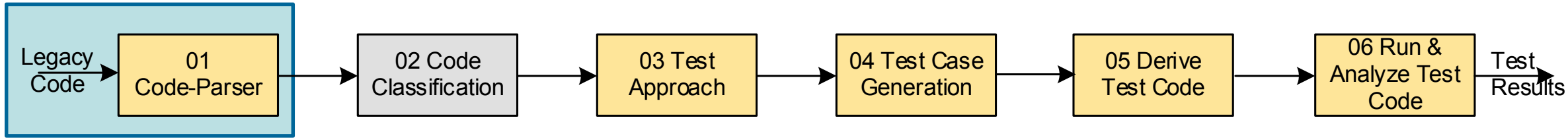


Wesentliche Schritte der Testautomatisierung

1. **Parse Code**: bestehender Code wird analysiert und als abstrakte Baumstruktur als Java Objekt persistiert.
2. **Code Classification**: Die Komplexität des Codes erfordert eine unterschiedliche Vorgehensweise.
3. **Test Approach**: Als Testansatz werden derzeit Äquivalenzklassen mit geeignetem Editor unterstützt.
4. **Test Case Generation**:
    1. die Erstellung von Test-Stubs
    2. die Generierung von Test Code. Im initialen Prototyp werden diese Schritte zusammengefasst.
5. **Derive Test Code**: Erstellung bzw. Generierung von Test Code
6. **Run & Analyze Test Code**: Ausführung der Testfälle ist im Rahmen der Eclipse Entwicklungsumgebung.



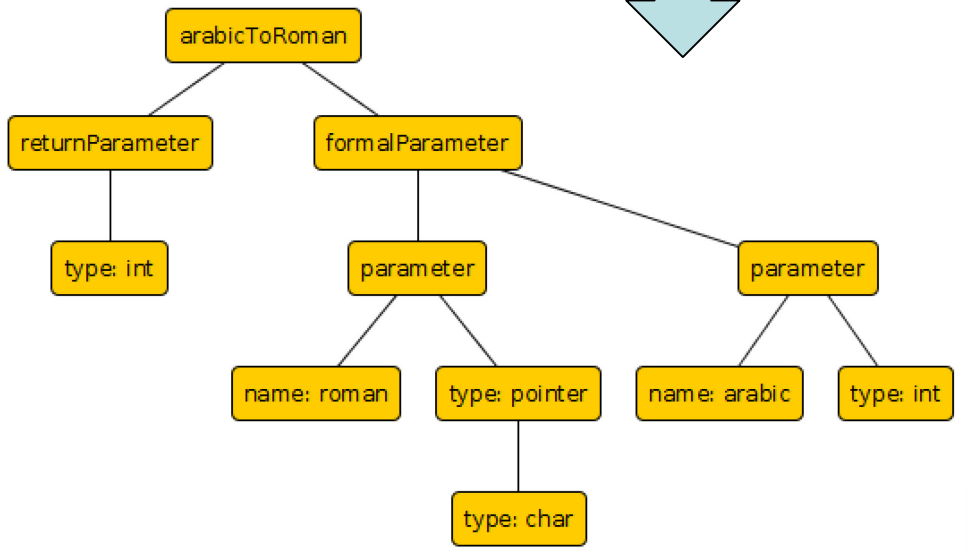
1. Parse Code



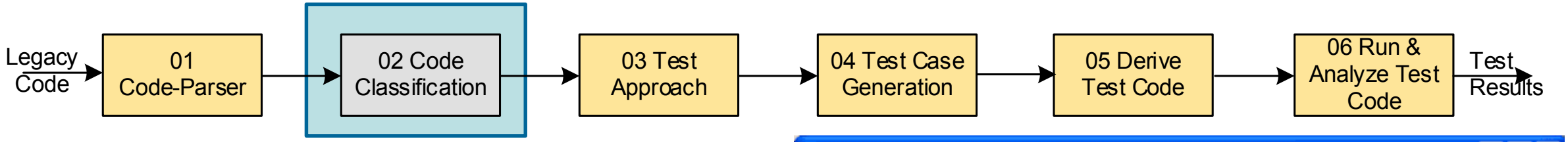
- Automatische **Analyse** von bestehendem Software Code.
- Erstellung einer abstrakten und **sprachunabhängigen Baumstruktur** *(Abstract Syntax Tree)*
- **Persistierung** als Java Objekt.

Roman.c

```c
#include "Roman.h"

#define ADD_ROMAN_CHAR(c) if (roman) roman[length] = c; length++

int arabicToRoman(int arabic, char *roman) {

    int length = 0;

    if (arabic <= 0){
        ADD_ROMAN_CHAR('\0');
        return 0; /* error */
    }

    while (arabic >= 1000) {
        arabic -= 1000;
        ADD_ROMAN_CHAR('M');
    };

    if (arabic >= 900) {
        arabic -= 900;
        ADD_ROMAN_CHAR('C');
        ADD_ROMAN_CHAR('M');
    };

    if (arabic >= 500) {
        arabic -= 500;
        ADD_ROMAN_CHAR('D');
    };

    #undef ADD_ROMAN_CHAR

    return length;
```

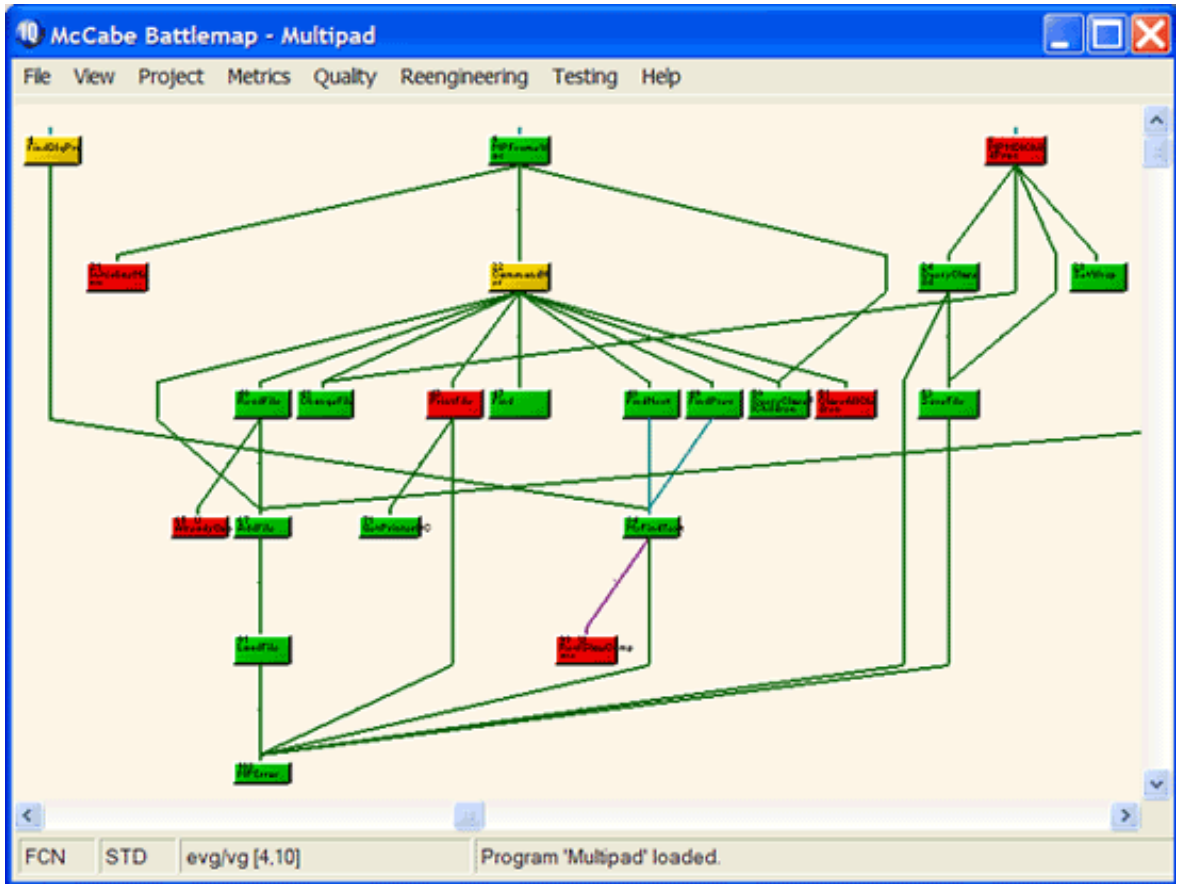


2. Code Klassifikation



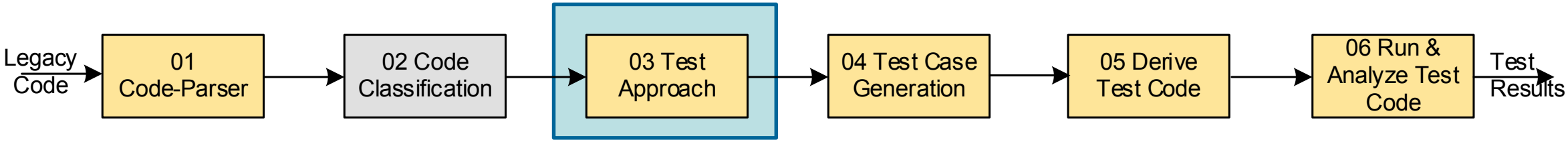
- **Unterschiedliche Code-Komplexität**
  - **Level 1: Bedingungen und Schleifen**
  - Level 2: Globale Variablen
  - Level 3: Externe Abhängigkeiten und Funktionsaufrufe

- **Unterschiedliche Vorgehensweisen** erforderlich.
  (Ersetzen von Abhängigkeiten, Mocking).



Symbolabbildung: www.mccabe.com/iq_developers.htm

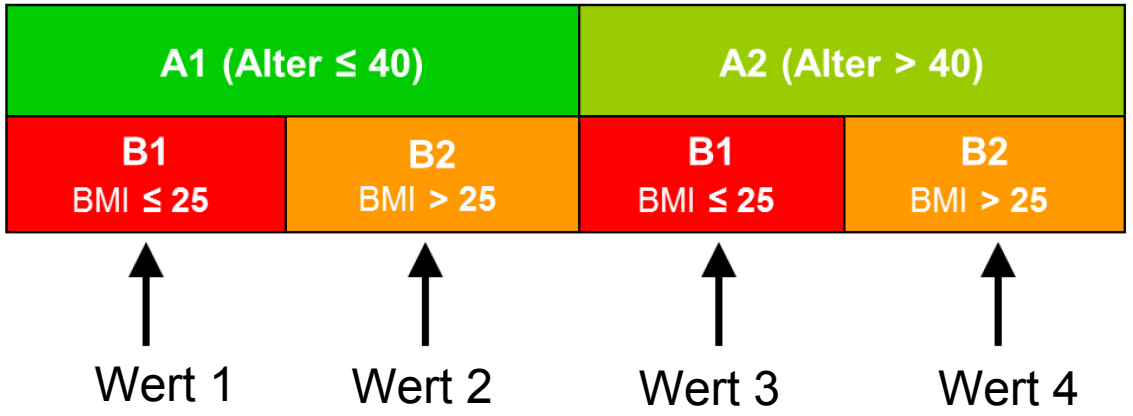
3. Testverfahren



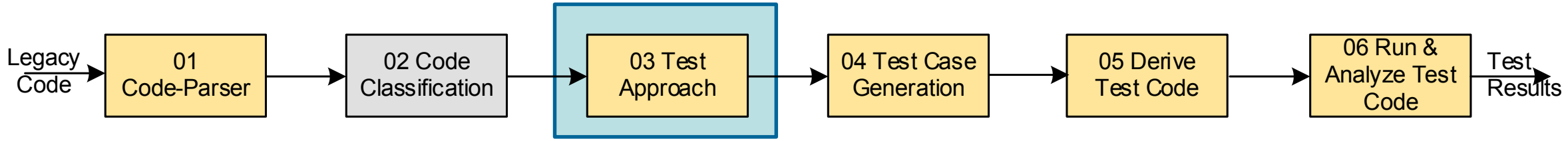
- **Fokus auf Äquivalenzklassen**
  - Aufteilung der Wertebereiche (Input / Output).
  - Auswahl eines repräsentativen Wertes aus jedem Bereich (Äquivalenzklasse).
  - Kombination von Repräsentanten.
  - Erstellung der Testfälle.

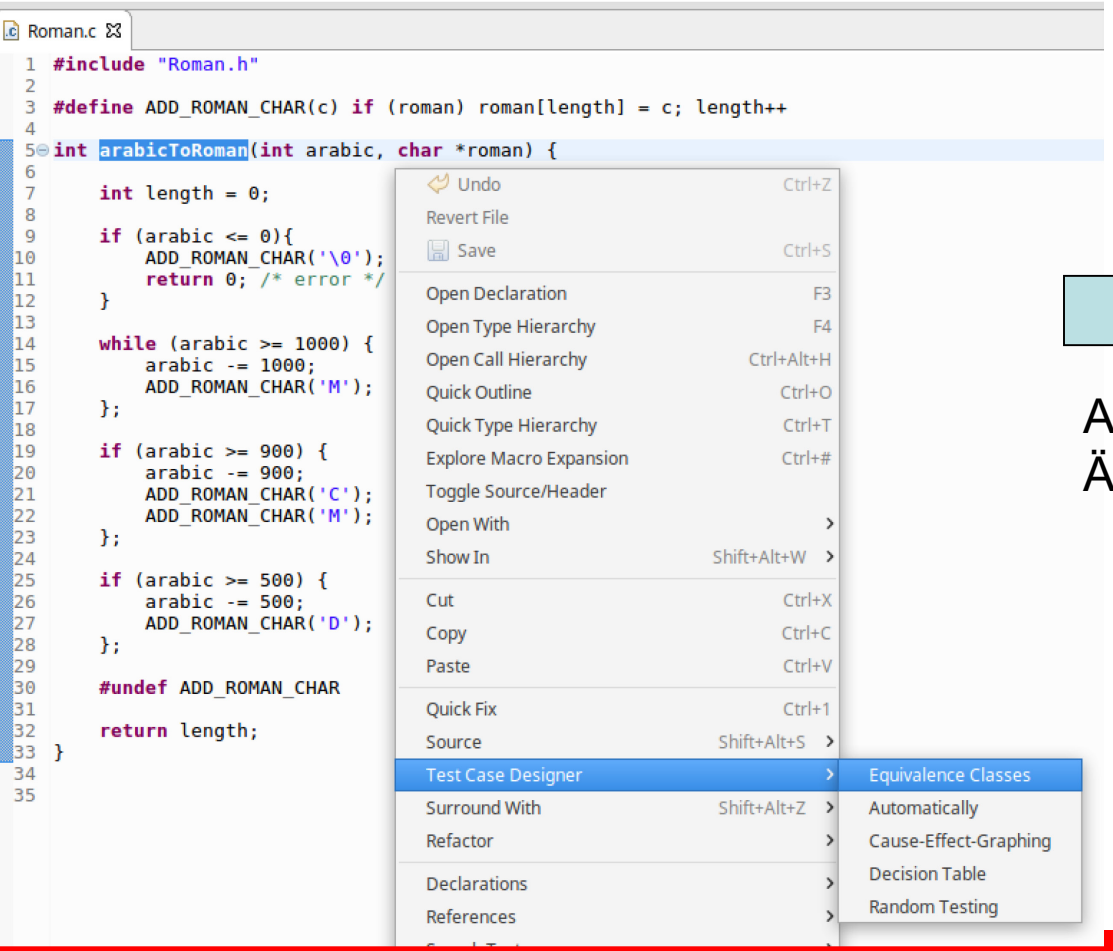
- **Offenheit für unterschiedliche Testverfahren** (z.B. Grenzwertanalyse).

Konzeptabbildung für die Äquivalenzklassenzerlegung

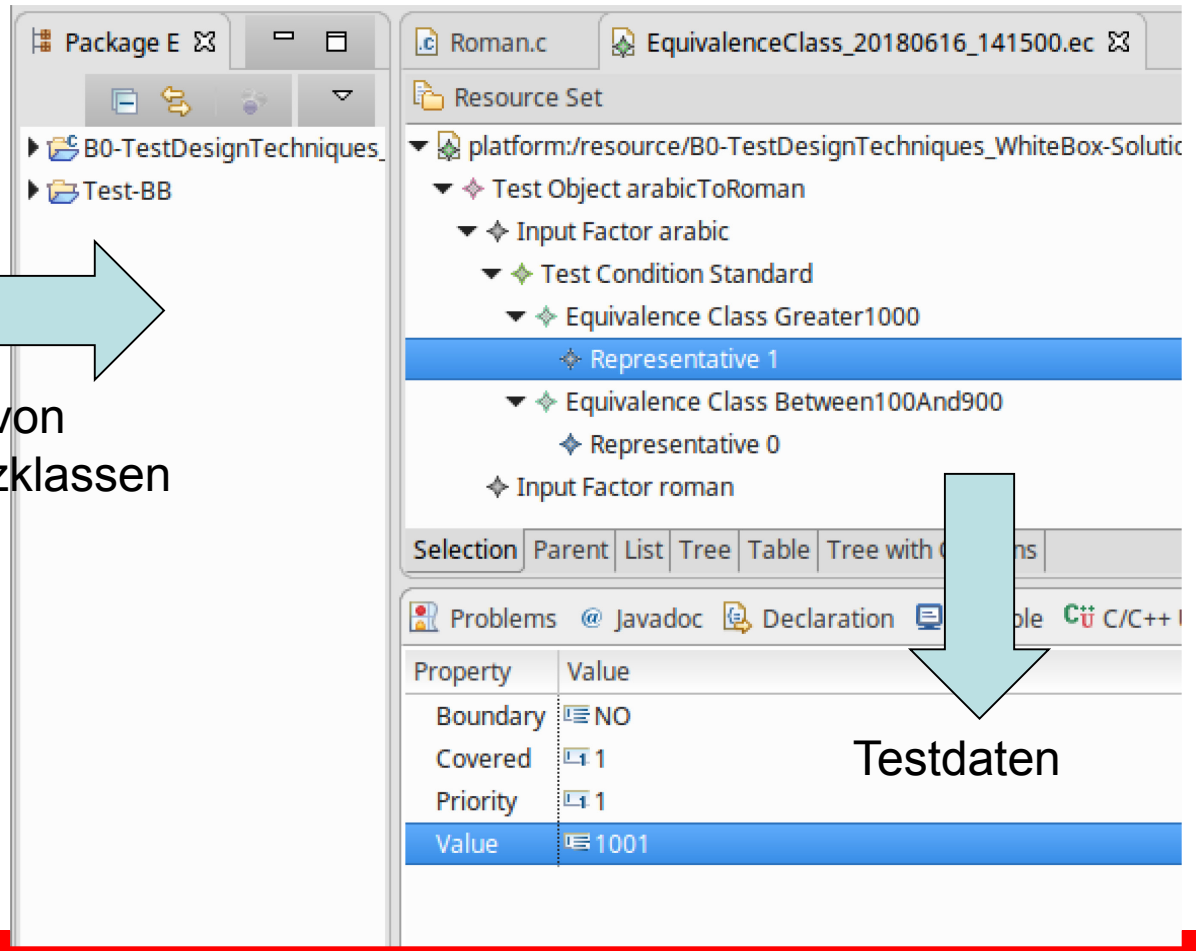


3. Testverfahren – Prototypische Umsetzung



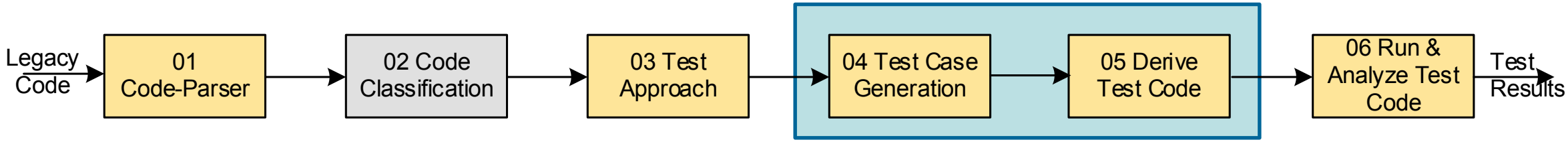


Ableitung von Äquivalenzklassen

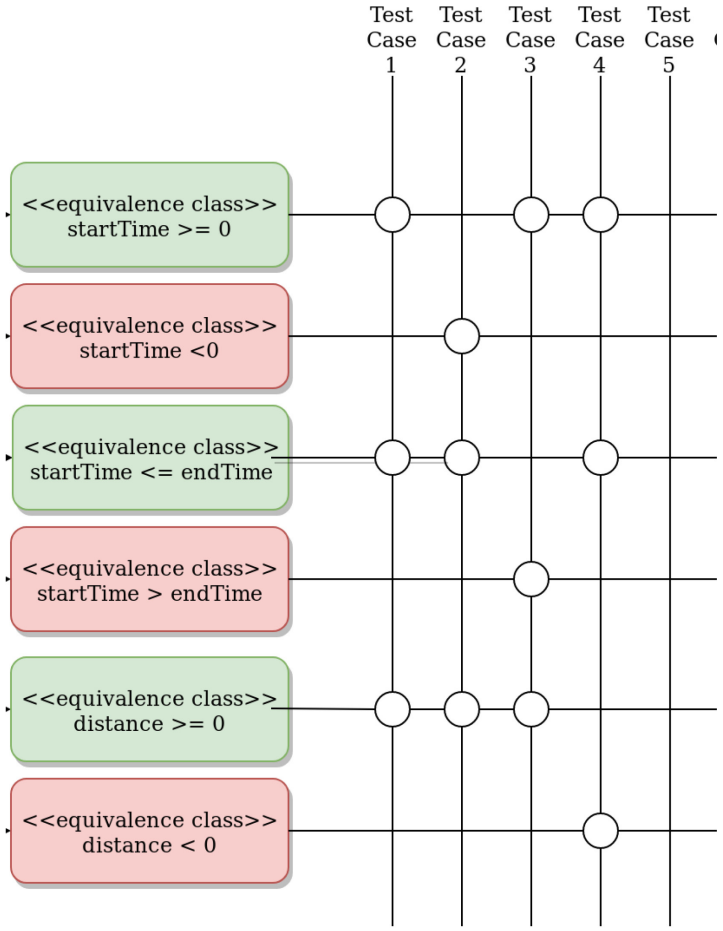


Testdaten

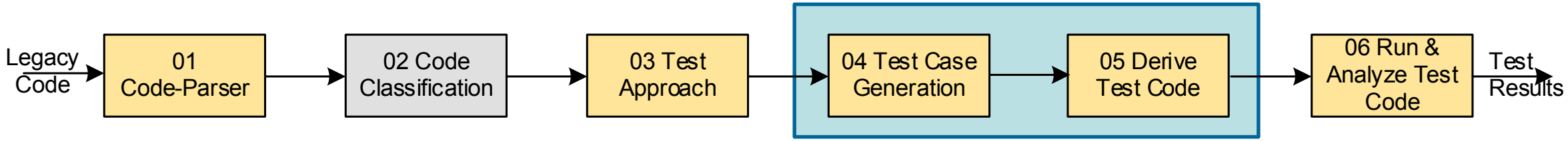
4. Test Case & 5. Code Generation

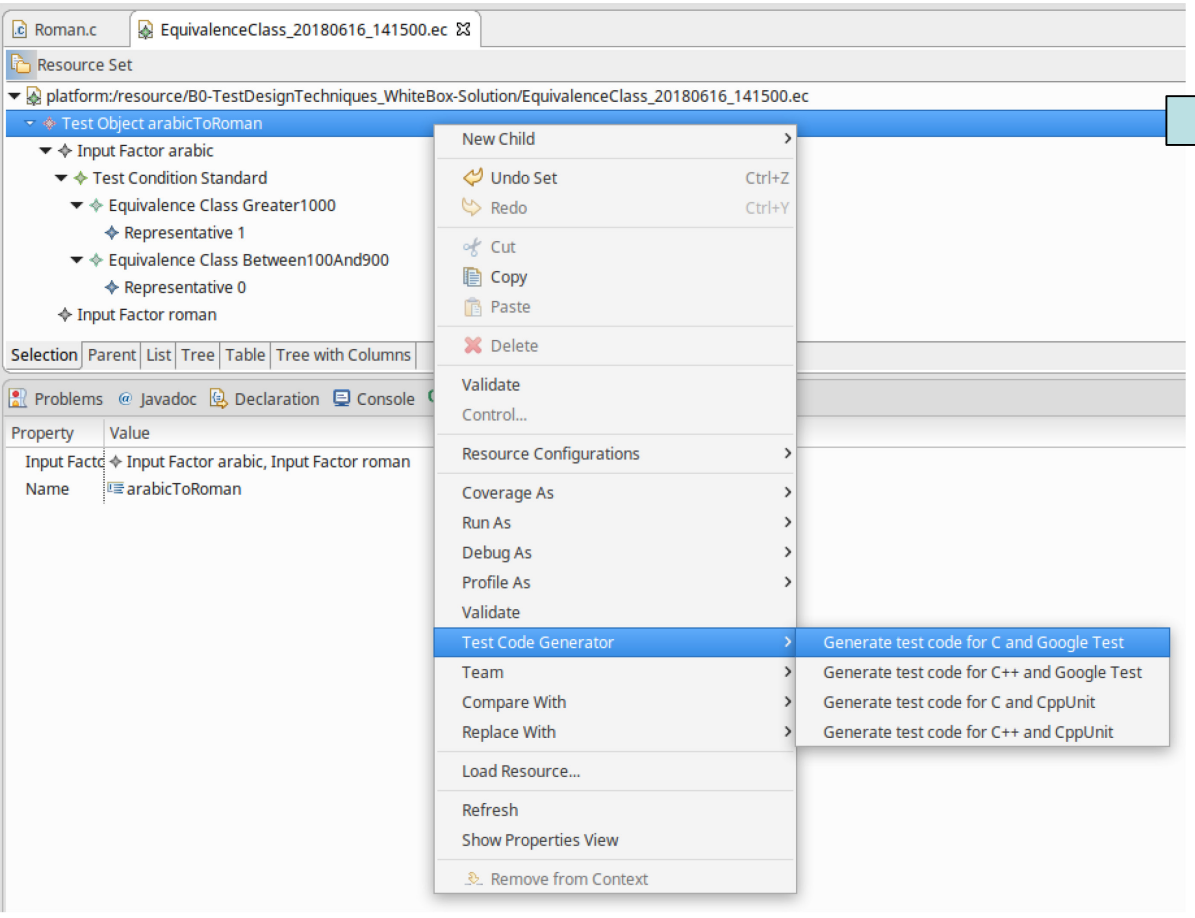


- **Automatische Kombination** von Äquivalenzklassen-Ausprägungen.
- **Erstellung von Testfällen** aus den verbleibenden Äquivalenzklassen.
- **Generierung von Test-Code** anhand definierter Test-Stubs für ein bestimmtes Testframework.
- **Offenheit** gegenüber unterschiedlichen Test-Frameworks.

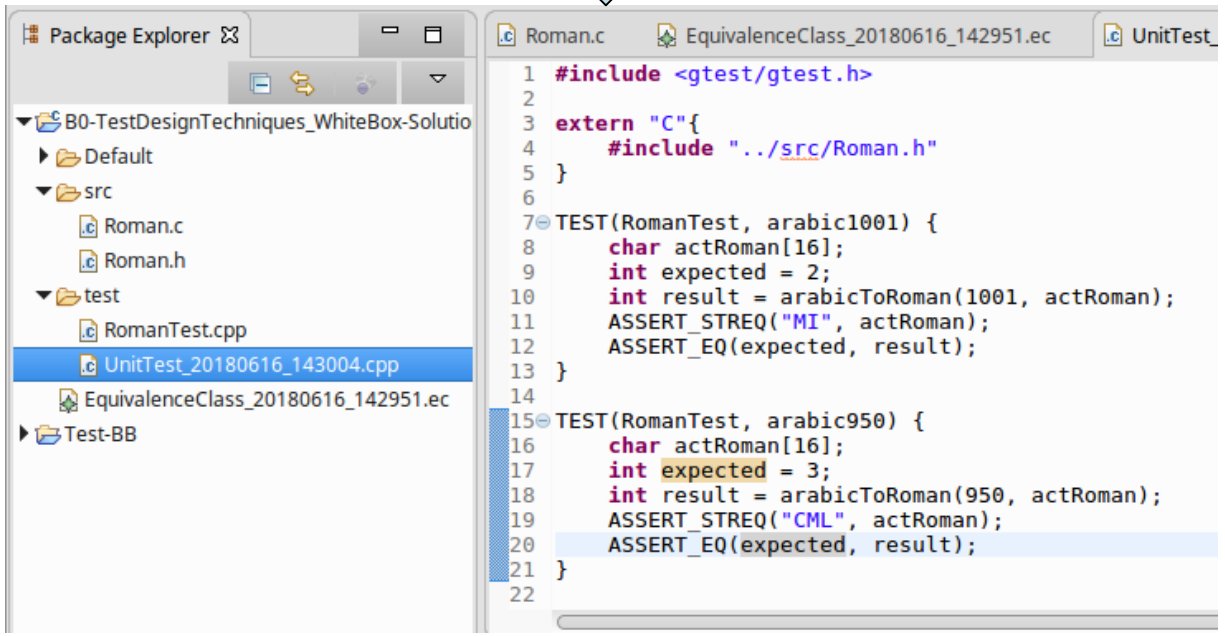


4. Test Case & 5. Code Generation

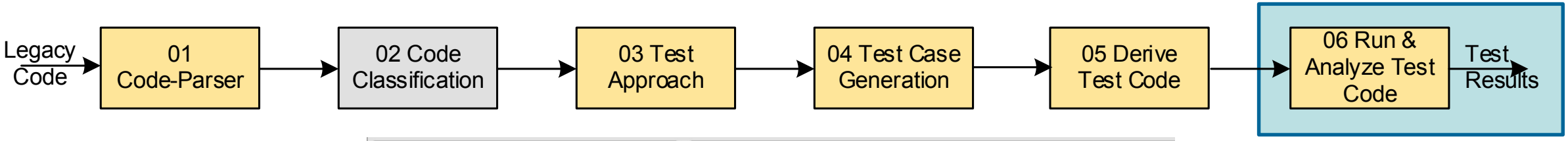


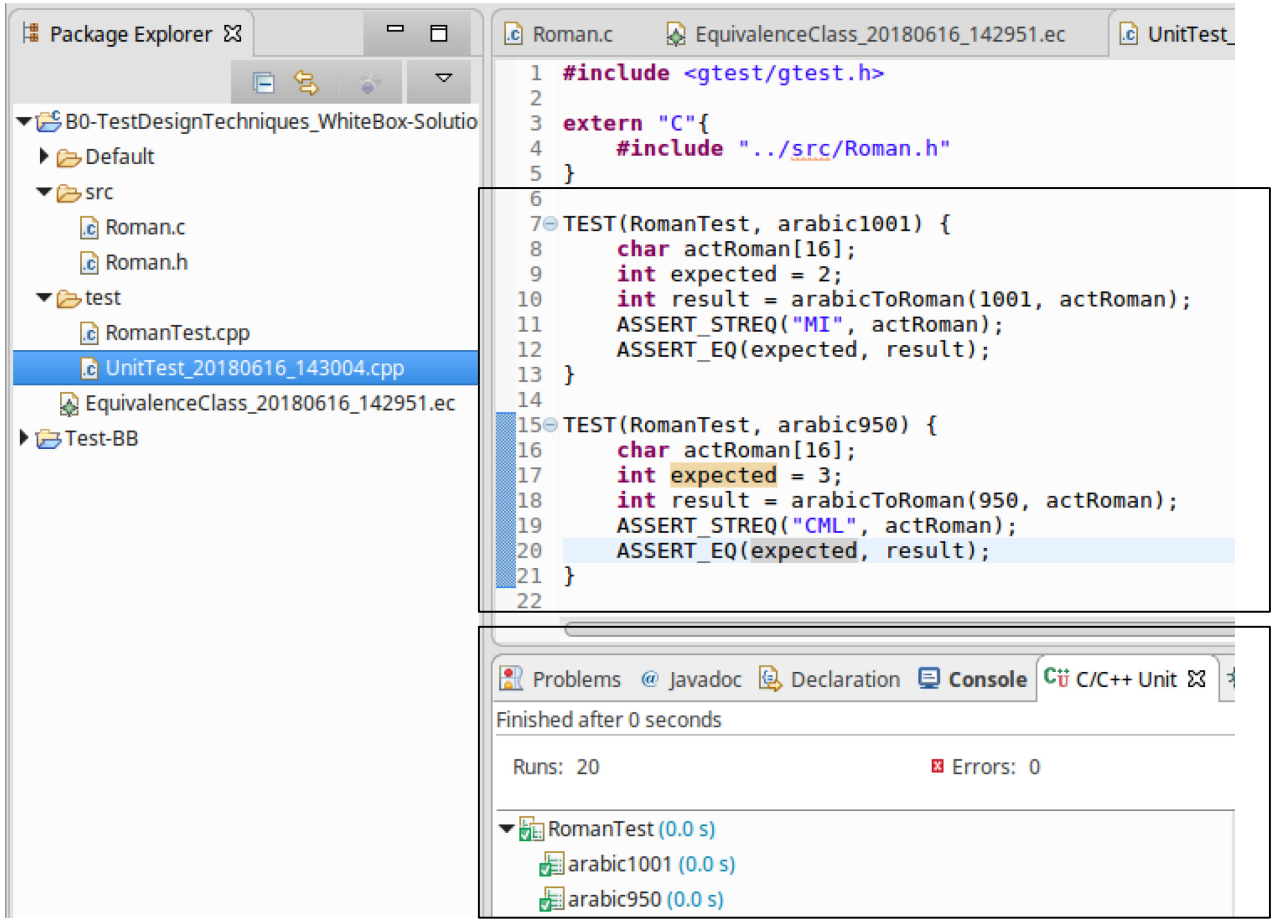


Generierte Testfälle



6. Run & Analyse Test Code





Generierte Testfälle

Testergebnisse

Semi-automatisierte Testfallgenerierung

- Unterstützung bei der systematischen Erstellung von (fehlenden) Software-Tests.
- Qualitätssicherung bei Migrationsprojekten durch wieder verwendbare Testfälle.
- Verwendung von Best-Practice Ansätzen.
- Sprachunabhängige Modellierung von Software Code.