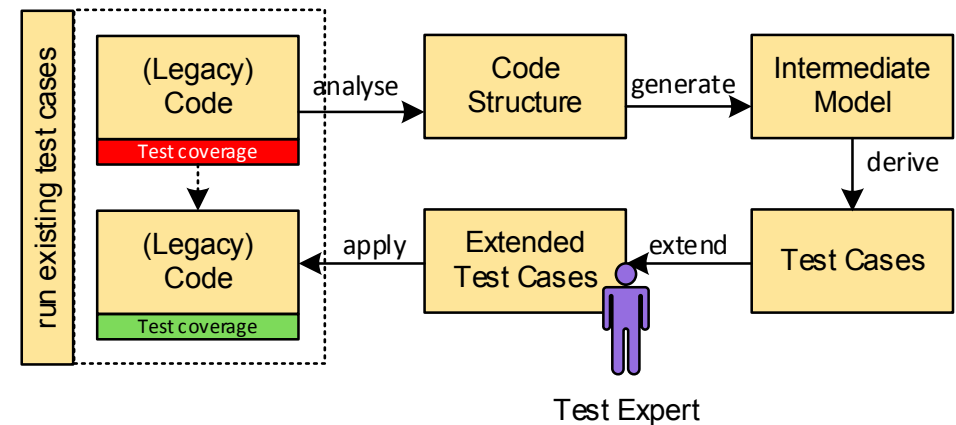


# Semi-automatisierte Testfallgenerierung

- 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.



- **Herausforderungen**

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

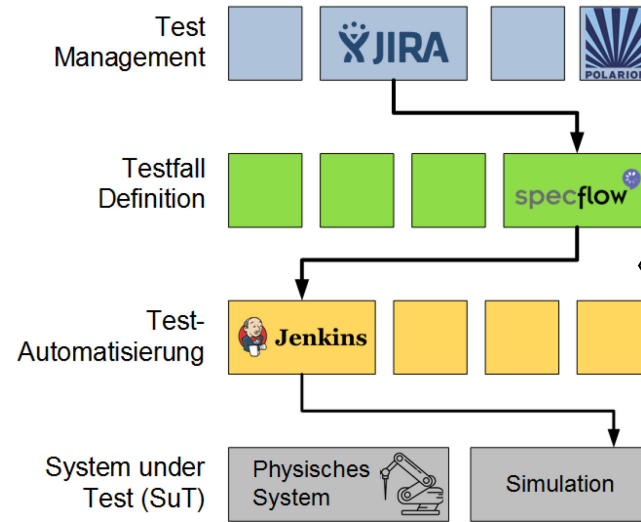
- **Ziel: Semi-automatische Testfallgenerierung**

- 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.

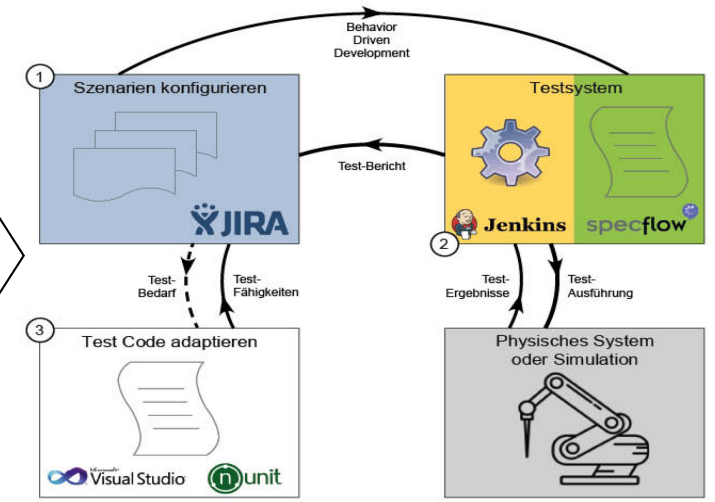


- **Big Picture und Vision**

- A. Test Automation Framework
- B. Anwendungsfall
- C. Verbesserung von Testfällen und Testprozessen



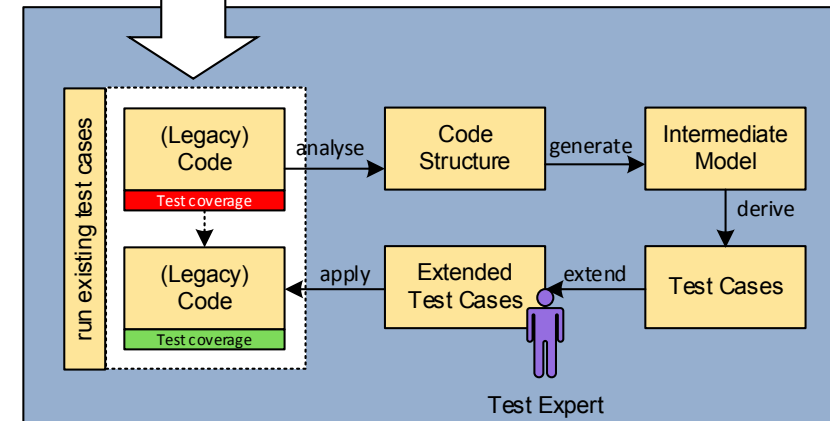
A. Test Automation Framework



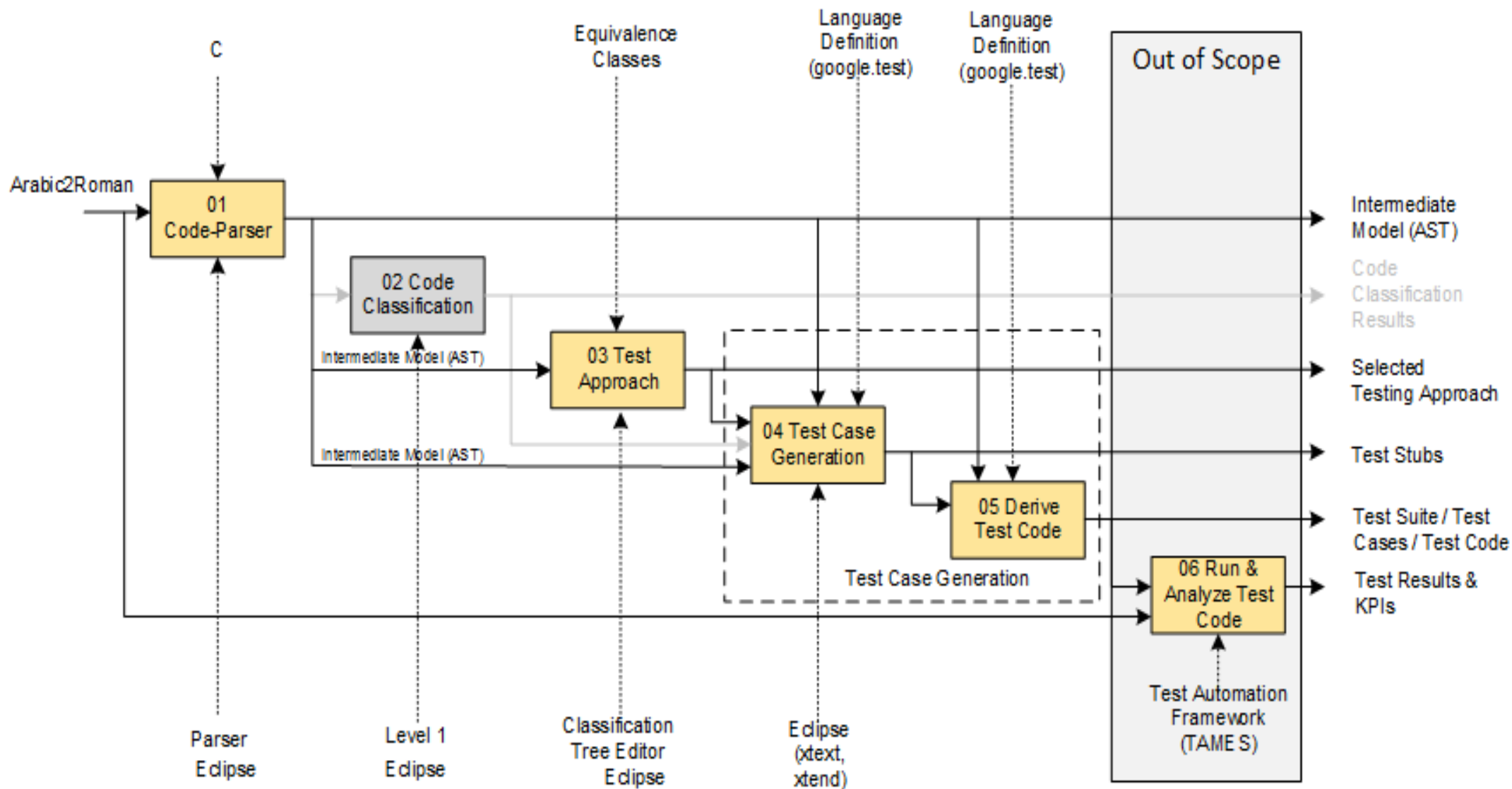
B. Application Use Case

- **Semi-automatische Testfallgenerierung**

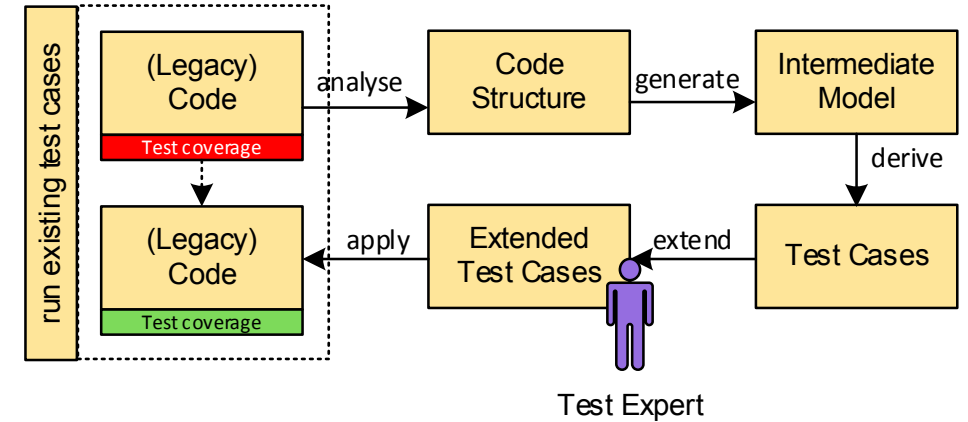
- Offenheit für unterschiedliche Testverfahren.
- Offenheit gegenüber unterschiedlichen Test-Frameworks.
- Verwendung von Best-Practice Ansätzen im Software Testen.
- Sprachunabhängige Modellierung von Software Code.

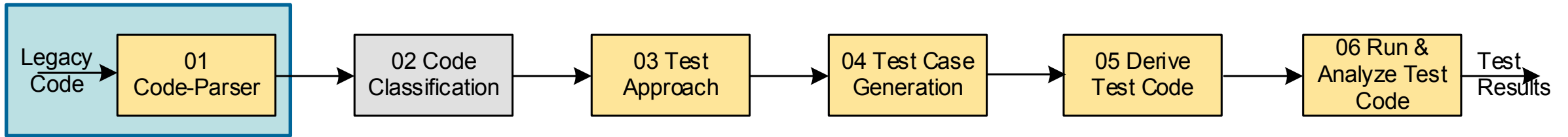


C. Improving Test Cases and Test Processes



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.

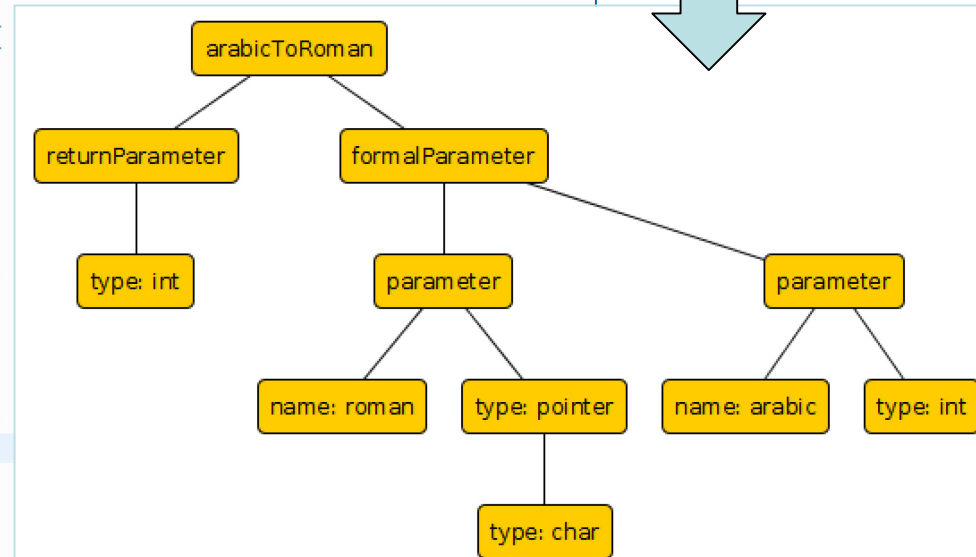




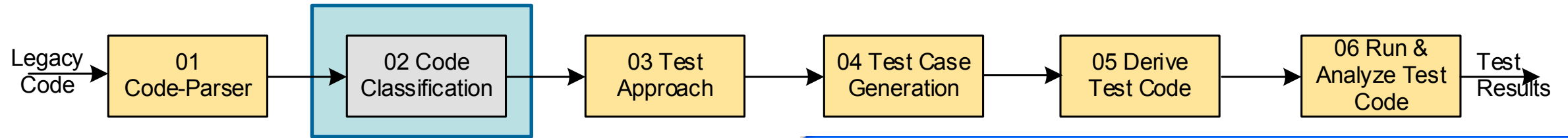
- Automatische **Analyse** von bestehendem Software Code.
- Erstellung einer abstrakten und **sprachunabhängigen Baumstruktur (Abstract Syntax Tree)**
- **Persistierung** als Java Objekt.

```

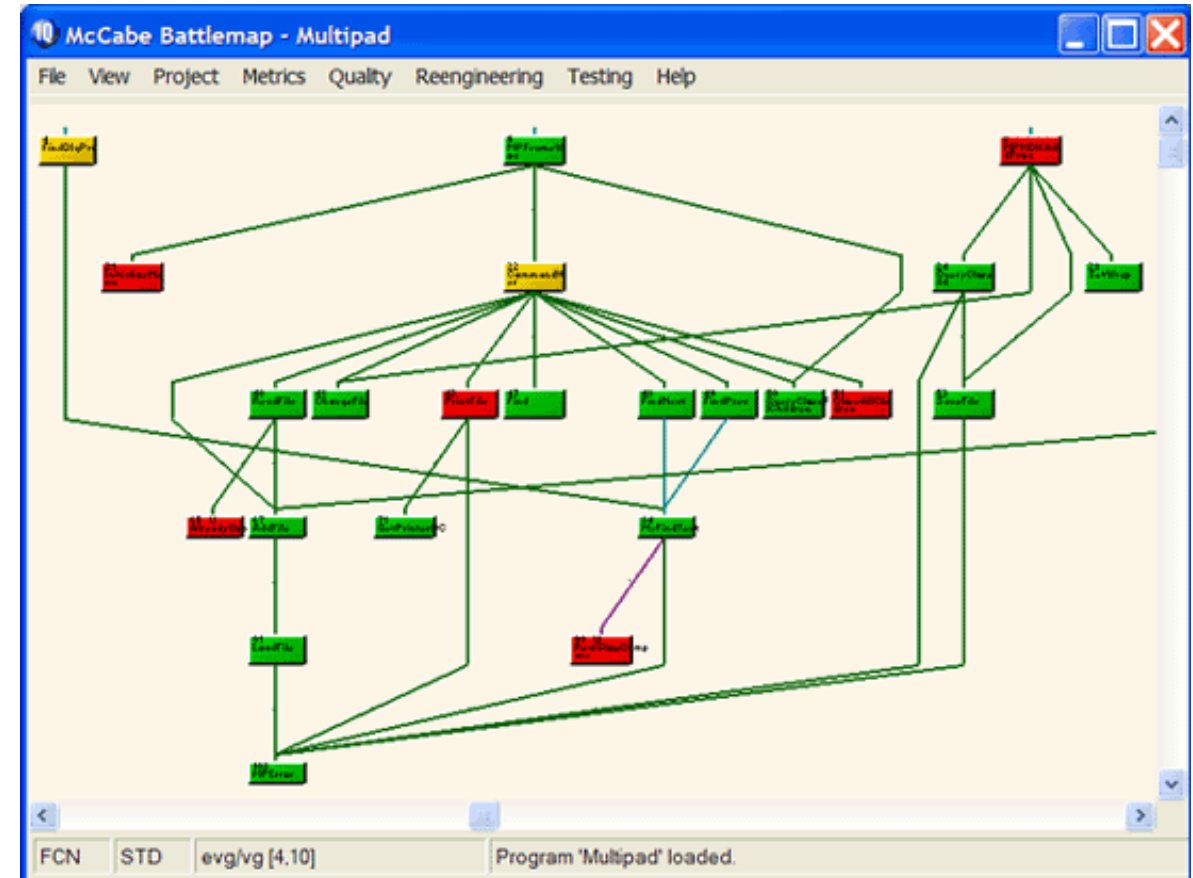
Roman.c
1 #include "Roman.h"
2
3 #define ADD_ROMAN_CHAR(c) if (roman) roman[length] = c; length++
4
5 int arabicToRoman(int arabic, char *roman) {
6     int length = 0;
7
8     if (arabic <= 0){
9         ADD_ROMAN_CHAR('\0');
10        return 0; /* error */
11    }
12
13    while (arabic >= 1000) {
14        arabic -= 1000;
15        ADD_ROMAN_CHAR('M');
16    };
17
18    if (arabic >= 900) {
19        arabic -= 900;
20        ADD_ROMAN_CHAR('C');
21        ADD_ROMAN_CHAR('M');
22    };
23
24    if (arabic >= 500) {
25        arabic -= 500;
26        ADD_ROMAN_CHAR('D');
27    };
28
29
30 #undef ADD_ROMAN_CHAR
31
32 return length;
  
```



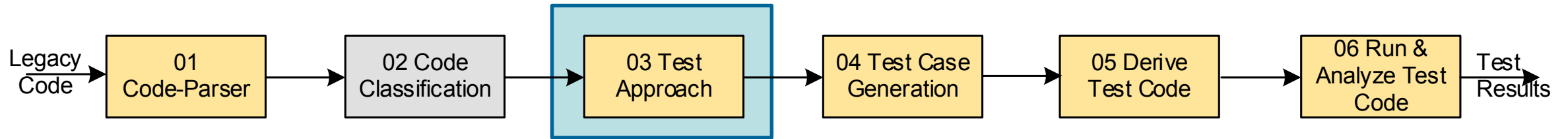




- **Unterschiedliche Code-Komplexität**
  - **Level 1: Bedingungen und Schleifen**
  - Level 2: Globale Variablen
  - Level 3: Externe Abhängigkeiten und Funktionsaufrufe
  
- **Unterschiedliche Vorgehensweisen** erforderlich.  
(Ersetzen von Abhängigkeiten, Mocking).

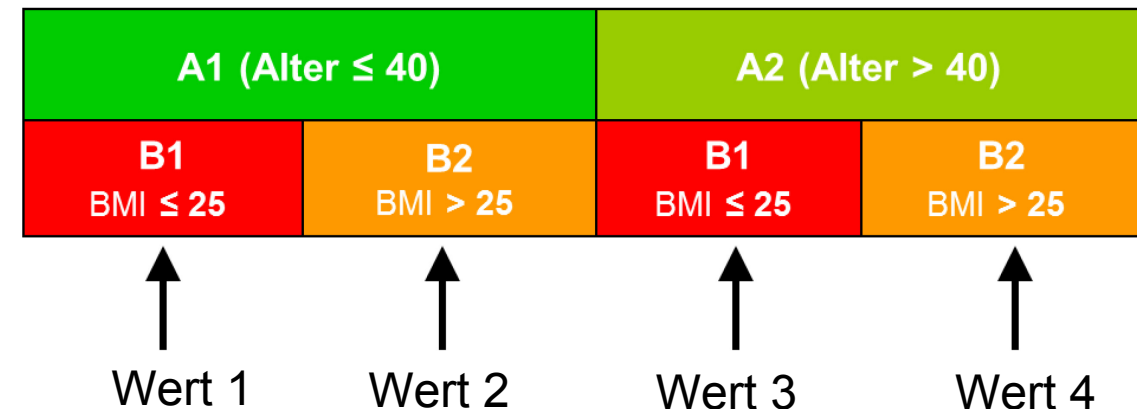


Symbolabbildung: [www.mccabe.com/iq\\_developers.htm](http://www.mccabe.com/iq_developers.htm)

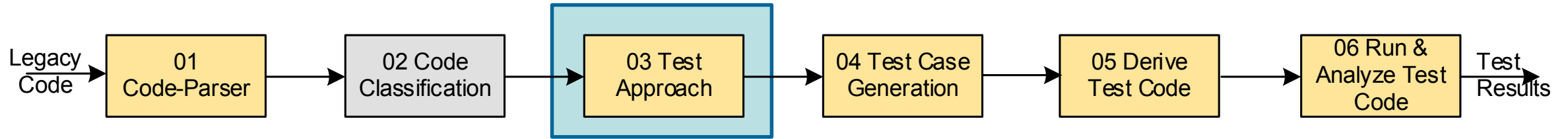


- **Fokus auf Äquivalenzklassen**
  - Aufteilung der Wertebereiche (Input / Output).
  - Auswahl eines repräsentativen Wertes aus jedem Bereich (Äquivalenzklasse).
  - Kombination von Repräsentanten.
  - Erstellung der Testfälle.
- **Offenheit für unterschiedliche Testverfahren** (z.B. Grenzwertanalyse).

Konzeptabbildung für die Äquivalenzklassenerlegung



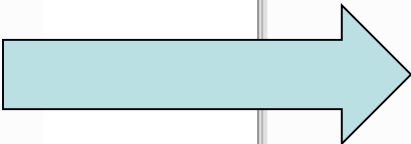




```

1 #include "Roman.h"
2
3 #define ADD_ROMAN_CHAR(c) if (roman) roman[length] = c; length++
4
5 int arabicToRoman(int arabic, char *roman) {
6
7     int length = 0;
8
9     if (arabic <= 0){
10        ADD_ROMAN_CHAR('\0');
11        return 0; /* error */
12    }
13
14    while (arabic >= 1000) {
15        arabic -= 1000;
16        ADD_ROMAN_CHAR('M');
17    };
18
19    if (arabic >= 900) {
20        arabic -= 900;
21        ADD_ROMAN_CHAR('C');
22        ADD_ROMAN_CHAR('M');
23    };
24
25    if (arabic >= 500) {
26        arabic -= 500;
27        ADD_ROMAN_CHAR('D');
28    };
29
30    #undef ADD_ROMAN_CHAR
31    return length;
32 }
33
34
35
  
```

- Undo (Ctrl+Z)
- Revert File
- Save (Ctrl+S)
- Open Declaration (F3)
- Open Type Hierarchy (F4)
- Open Call Hierarchy (Ctrl+Alt+H)
- Quick Outline (Ctrl+O)
- Quick Type Hierarchy (Ctrl+T)
- Explore Macro Expansion (Ctrl+#)
- Toggle Source/Header
- Open With
- Show In (Shift+Alt+W)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Quick Fix (Ctrl+1)
- Source (Shift+Alt+S)
- Test Case Designer**
  - Equivalence Classes
  - Automatically
  - Cause-Effect-Graphing
  - Decision Table
  - Random Testing
- Surround With (Shift+Alt+Z)
- Refactor
- Declarations
- References



Ableitung von Äquivalenzklassen

Package E

- B0-TestDesignTechniques
  - Test-BB

Roman.c EquivalenceClass\_20180616\_141500.ec

Resource Set

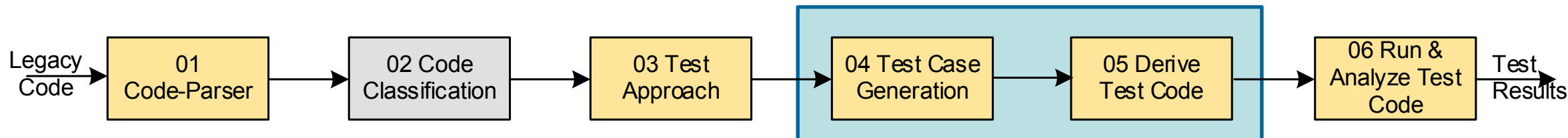
- platform:/resource/B0-TestDesignTechniques\_WhiteBox-Solution
  - Test Object arabicToRoman
    - Input Factor arabic
      - Test Condition Standard
        - Equivalence Class Greater1000
          - Representative 1
        - Equivalence Class Between100And900
          - Representative 0
      - Input Factor roman

Selection Parent List Tree Table Tree with Columns

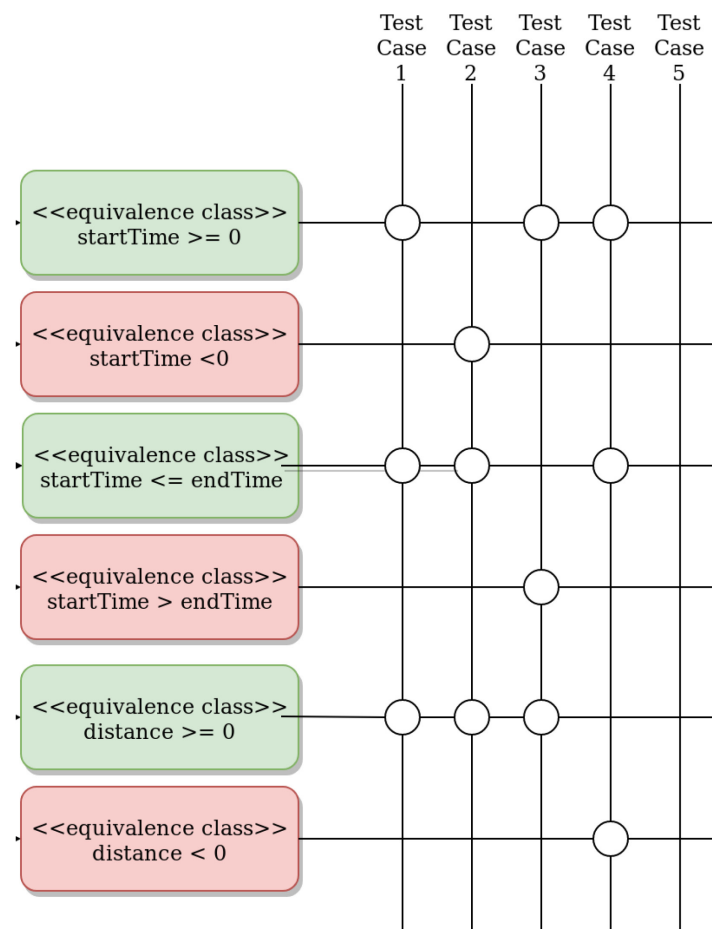
Problems @ Javadoc Declaration C/C++

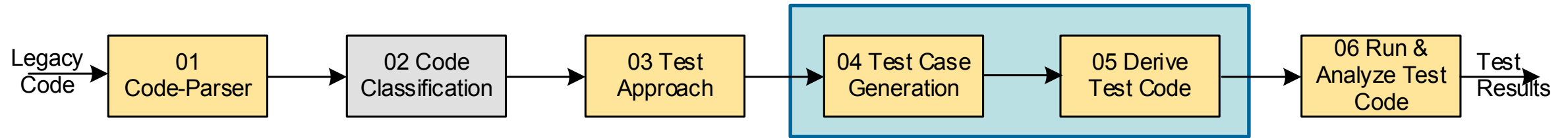
Property	Value
Boundary	NO
Covered	1
Priority	1
Value	1001

Testdaten



- **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.





Resource Set: platform:/resource/B0-TestDesignTechniques\_WhiteBox-Solution/EquivalenceClass\_20180616\_141500.ec

- Test Object arabicToRoman
  - Input Factor arabic
    - Test Condition Standard
      - Equivalence Class Greater1000
        - Representative 1
      - Equivalence Class Between100And900
        - Representative 0
    - Input Factor roman

Context Menu:

- New Child
- Undo Set (Ctrl+Z)
- Redo (Ctrl+Y)
- Cut
- Copy
- Paste
- Delete
- Validate
- Control...
- Resource Configurations
- Coverage As
- Run As
- Debug As
- Profile As
- Validate
- Test Code Generator
  - Generate test code for C and Google Test
  - Generate test code for C++ and Google Test
  - Generate test code for C and CppUnit
  - Generate test code for C++ and CppUnit
- Team
- Compare With
- Replace With
- Load Resource...
- Refresh
- Show Properties View
- Remove from Context

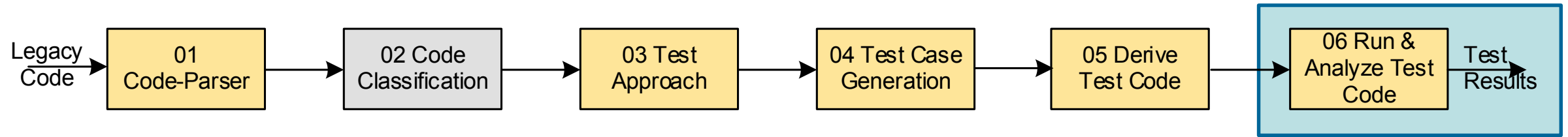
Generierte Testfälle

Package Explorer:

- B0-TestDesignTechniques\_WhiteBox-Solution
  - Default
  - src
    - Roman.c
    - Roman.h
  - test
    - RomanTest.cpp
    - UnitTest\_20180616\_143004.cpp
    - EquivalenceClass\_20180616\_142951.ec
  - Test-BB

```

1 #include <gtest/gtest.h>
2
3 extern "C"{
4     #include "../src/Roman.h"
5 }
6
7 TEST(RomanTest, arabic1001) {
8     char actRoman[16];
9     int expected = 2;
10    int result = arabicToRoman(1001, actRoman);
11    ASSERT_STREQ("MI", actRoman);
12    ASSERT_EQ(expected, result);
13 }
14
15 TEST(RomanTest, arabic950) {
16     char actRoman[16];
17     int expected = 3;
18     int result = arabicToRoman(950, actRoman);
19    ASSERT_STREQ("CML", actRoman);
20    ASSERT_EQ(expected, result);
21 }
22
  
```



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a project structure with folders 'src' and 'test'. The 'test' folder contains 'RomanTest.cpp' and 'UnitTest\_20180616\_143004.cpp'. The main editor shows the source code of 'Roman.c' with two test cases highlighted in blue:

```

7 TEST(RomanTest, arabic1001) {
8     char actRoman[16];
9     int expected = 2;
10    int result = arabicToRoman(1001, actRoman);
11    ASSERT_STREQ("MI", actRoman);
12    ASSERT_EQ(expected, result);
13 }
14
15 TEST(RomanTest, arabic950) {
16     char actRoman[16];
17     int expected = 3;
18     int result = arabicToRoman(950, actRoman);
19     ASSERT_STREQ("CML", actRoman);
20     ASSERT_EQ(expected, result);
21 }
22
  
```

At the bottom, the Console window shows the test execution results:

```

Finished after 0 seconds
Runs: 20 Errors: 0
RomanTest (0.0 s)
  arabic1001 (0.0 s)
  arabic950 (0.0 s)
  
```

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.

