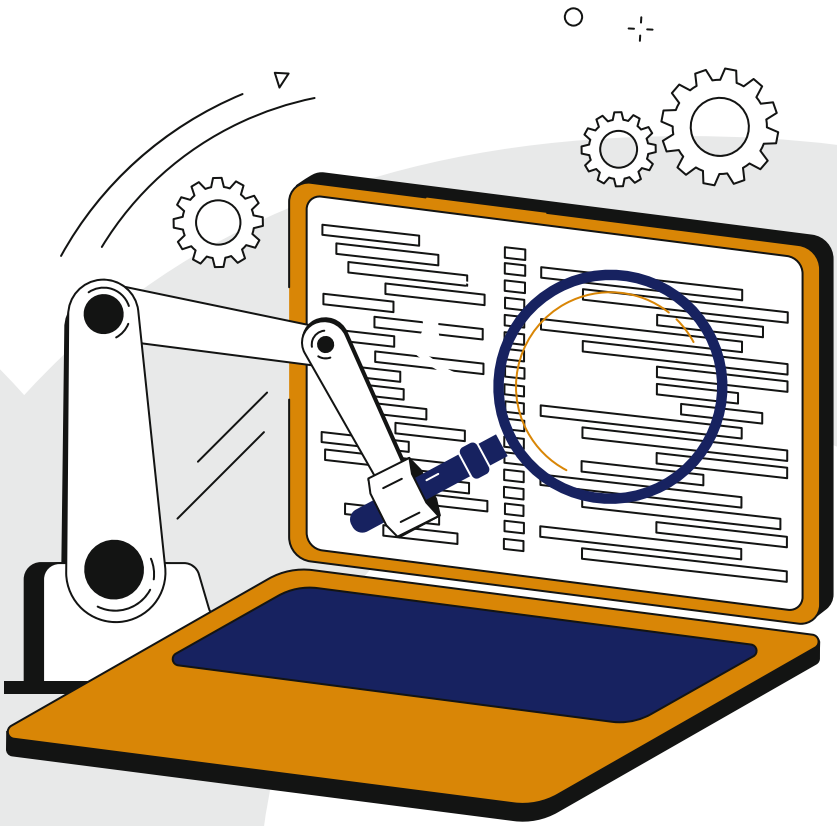
SCCX Structured CodeCoverage Xtension



Automate Your Code

Der einfache Weg zum automatisierten Test
Testabdeckung in IEC61131-3 Structured Text/ ST / SCL

HOOX [SOFTWARE]

Automate your Code - Inhalt

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Einleitung

Bei dem hier beschriebenen Tool handelt es sich um einen Generator für Steuerungssoftware.
Es kann unterschieden werden zwischen

- Öffnen und kompilieren eines bestehenden Projekts.
- Erstellen einer Bibliothek aus einem bestehenden Projekt heraus.
- Ausführen eines bestehenden Projekts auf einem Zielsystem.
- Ausführen eines Testobjekts auf einem Zielsystem.
- Auswerten eines Testobjekts (JUnit, Cobertura Codecoverage).

Installation

Nach erfolgreicher Installation finden sich im Installationspfad folgende Dateien:

Generator	
bin	Generator.exe
doc	dieses Dokument

Lib	
Sigmatek	Bibliotheken für automatisierte Tests
Beckhoff	Bibliotheken für automatisierte Tests

Beckhoff	
Sample	Beispiel für die Anwendung
Framework	Bibliothek
template	Beispiel Konfigurationsdatei

Sigmatek	
Sample	Beispiel für die Anwendung
Testframework	Bibliothek
template	Beispiel Konfigurationsdatei

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Anwendung

Allgemein

Start des Generators über die Kommandozeile und Übergabe des Pfads zur Konfigurationsdatei.

Bsp: C:\Installation*Generator.exe "C:\konfiguration.xml"*

Die Konfiguration enthält die Aufgaben die bearbeitet werden sollen. Das kann ein einziger Build-Befehl sein der nur versucht das angegebene Projekt zu laden und zu kompilieren. Oder ein mehrstufiger Prozess der Bspw. vor dem Build Bibliotheken bereitstellt.

Vorgehensweise

Abhängig von der verwendeten Steuerung ergeben sich leicht unterschiedliche Vorgehensweisen um einen automatischen Testlauf zu realisieren.

Hier kurz die notwendigen Schritte

Sigmatek

1. Testprojekt mit dem Testframework erstellen
   Die Bibliothek wird bei der Installation im Lasal Verzeichnis abgelegt.
   (siehe zugehörige Beschreibung)
2. Konfigurationsdatei erstellen
3. Generator starten

Für die Konfigurationsdatei ist im Installationsverzeichnis unter Lib\Sigmatek\Sample eine Vorlage zu finden.

Beckhoff

1. Bibliotheken importieren
   - optional Testframework.library wenn diese verwendet werden soll.
   - Codecoverage.library zur Erfassung der Testabdeckung.

1. Testprojekt mit Testframework erstellen
   (siehe zugehörige Beschreibung)
   - einbinden der Testframework.library
   - einbinden der Codecoverage.library
2. GVL_Test anlegen (siehe Besonderheiten)
3. Konfigurationsdatei erstellen
4. Generator starten

Für die Konfigurationsdatei ist im Installationsverzeichnis unter Lib\Beckhoff\Sample eine Vorlage zu finden.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Konfigurationsdatei

Die Datei wird im XML-Format erstellt und enthält die auszuführenden Aufgaben als Eintrag. Nachfolgend ein paar Beispiele für eine einfache Build-Konfiguration.

```xml
<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Lasal build="1">
        <Path>C:\\PfadZumProjekt</Path>
    </Lasal>

</Build>
```

Sigmatek Lasal Class 2

```xml
<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <LasalScreen build="1">
        <Path>C:\\PfadZumProjekt</Path>
    </LasalScreen>

</Build>
```

Sigmatek Lasal Screen

```xml
<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Twincat build="1">
        <Path>C:\\PfadZumProjekt</Path>
    </Twincat >

</Build>
```

Beckhoff Twincat

Die erstellte XML Datei wird über das Schema unter https://hoox.software/generator.xsd validiert, und im Fehlerfall der Vorgang abgebrochen.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

XML Format

Unter dem Haupteintrag <Build> wird zunächst die zu verwendende Toolchain eingetragen, und über die Attribute die Aufgabe konfiguriert. Abhängig von den Attributen sind weitere Elemente zur Konfiguration notwendig.

Toolchain	
`<Lasal>`	Sigmatek Steuerung
`<LasalScreen>`	Sigmatek Visualisierung
`<Twincat>`	Beckhoff Toolchain
`<BuR>`	B&R Toolchain
`<Bachmann>`	Bachmann Electronic

Attribute <Toolchain>

Attribut	
`build`	Lädt das, in Path angegebene, Projekt und kompiliert es.
`run`	Lädt die Applikation auf eine Steuerung und führt diese aus.
`generate`	Erstellt im angegebenen Pfad eine Applikation
`testobject`	Erstellt aus dem, in Path angegebenen Projekt, ein Testobjekt
`analyse`	Erstellt Junit und Cobertura Ergebnisse zur externen Weiterverarbeitung
`library`	Erstellt im Exportpfad eine Bibliothek

Elemente der Toolchain

Tag	
`<Path>`	Pfad zum Projekt
`<Export>`	Pfad zum Ablageort für das Testobjekt/die Bibliothek
`<Version>`	Auswahl einer bestimmten Version ??
`<Connection>`	Verbindung zu einer Steuerung IP-Adresse
`<Config>`	Konfiguration des Testobjekts oder der Applikation
`<Plc>`	Auswahl Plc, falls nur ausgewählte einer Solution gebaut werden sollen.

Notwendige Angaben

Attribut	
build	`<Path>` `<Plc>`
run	`<Path>` `<Connection>`

generate	`<Path>`
	`<Export>`
	`<config>`

testobject	`<Path>` `<Export>`
analyse	`<Path>` `<Connection>`
library	`<Path>` `<Export>`

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Umgebungsvariablen

Im Pfad <Path> und <Export> kann die Variable $WORKSPACE verwendet werden.
Wenn diese als Umgebungsvariable definiert wurde, wird deren Inhalt verwendet.

Beispiel

```
<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Lasal build="1">
        <Path>$WORKSPACE\\PfadZumProjekt</Path>
    </Lasal>

</Build>
```

Beispiel Powershell

```
$env:WORKSPACE = 'C:\Hoox\Test'
```

Ergibt als endgültigen Pfad: `'C:\Hoox\Test\PfadZumProjekt'`

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Test automatisieren

Allgemein

Der Generator bietet die Möglichkeit Projekte automatisch zu kompilieren und anschließend auf ein Zielsystem zu laden und zu starten. Dadurch können Testläufe erstellt, ausgeführt und ausgewertet werden.

Konfigurationsdatei

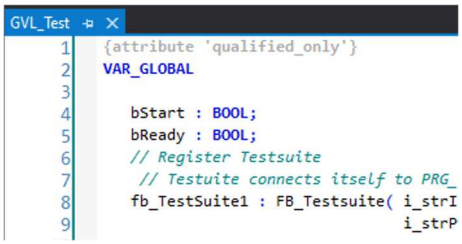
```
<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Lasal run="1" analyse="1">
        <Path>$WORKSPACE\\PfadZumProjekt</Path>
        <Connection>192.168.1.1</Connection>
    </Lasal>

</Build>
Lasal Class


<?xml version="1.0" encoding="utf-8" ?>

<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Twincat run="1" analyse="1">
        <Path>$WORKSPACE\\PfadZumProjekt</Path>
        <Connection>192.168.1.1</Connection>
    </Twincat>

</Build>
Twincat 3
```

Besonderheiten

Bei Twincat wird für den automatischen Teststart und späteren Auswertung eine Globale Variablenliste erwartet über die der Generator den Test starten kann und auf die Rückmeldung "Test beendet" wartet.

`GVL_Test.bStart = true` -> Testbeginn
Wird per ADS auf "true" gesetzt um den Testlauf zu starten. Dieser Wert ist optional.

`GVL_Test.bReady = true` -> Testende
Um die Codecoverage auszuwerten muss dem Generator mitgeteilt werden wann der Test beendet ist.



Folgender Code muss bei Verwendung des Testframeworks noch aktiviert werden:

```
TestFramework.PRG_TestLib.i_bStart := GVL_Test.bStart;
GVL_Test.bReady := (TestFramework.PRG_TestLib.o_eState =TestFramework.EState_t.eStateReady);
```


Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

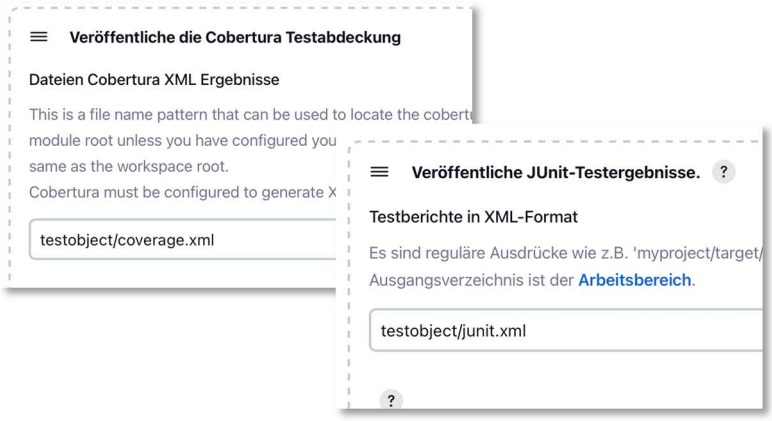
Testabdeckung/Codecoverage ermitteln

Zur Berechnung der Testabdeckung sind die attribute `testobject="1"` und `analyse="1"` notwendig. testobject erzeugt das instrumentierte Testobjekt und analyse sorgt dafür das die Ergebnisse von der Steuerung geladen und berechnet werden.

Danach findet sich im Pfad des Testobjekts die Datei **coverage.xml** im Cobertura Format das im Buildserver dann ausgewertet werden kann.

Bei Verwendung des mitgelieferten Testframeworks wird ausserdem noch die Datei **junit.xml** erstellt.

Beispiel aus Jenkins Buildserver



Ausnahmen

Es wird immer vorkommen das Dateien nicht in der Auswertung vorkommen sollen, z.B die Testbausteine selbst. Wenn in einer Datei innerhalb eines Kommentars **Exclude-SccX** gefunden wird, wird diese nicht instrumentiert und kommt so in der Auswertung nicht vor.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Programm generieren

Allgemein

Wenn das Attribut `generate` verwendet wird, muss eine Konfigurationsdatei im XML Format angegeben werden. Diese enthält eine Beschreibung des zu generierenden Projekts.
Dabei handelt es sich um eine spezielle Softwarearchitektur die keine direkten Verbindungen zwischen den Instanzen benötigt.
Weitere Informationen dazu unter https://www.hoox.software/modulare_maschine.html

Verfügbar für Beckhoff und Sigmatek Steuerungen. Weitere sind auf der Roadmap.

Konfigurationsdatei

```
<?xml version="1.0" encoding="utf-8" ?>

<Framework    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://hoox.software"
              xmlns:Hardware="http://hoox.software/Sigmatek"
              xsi:schemaLocation="http://hoox.software ../xsd/Sigmatek/Framework.xsd">

       <Project Name="TestRun" Connect="Local" Comment="Automatisierter Test" OpcUa="0"/>

       <Libraries>
              <Runtime>C:\\Sigmatek\\Lasal</Runtime>
              <Functions>C:\\ApplikationsBibliothek</Functions>
       </Libraries>

       <Hardware Disabled="false">
              <!-- CPU -->
              <Device Typ="CP112" Bmk="_32A4">
                     <IoElement Typ="DI204" Bmk="_32A5"/>
                     <IoElement Typ="DM161" Bmk="_32A5"/>
                     <IoElement Typ="AM221" Bmk="_32A6"/>
              </Device>
       </Hardware>

       <Software>
              <Network Name="Injection">
                     <Modul Name="Injector1"      Typ="CInjection"   />
                     <Modul Name="Injector2"      Typ="CInjection"   />
                     <Modul Name="Feeder"         Typ="CFeeder"   />
              </Network>
              <Network Name="Temperature">
                     <Modul Name="Zone1"           Typ="CPidRegulator"   />
                     <Modul Name="Zone2"           Typ="CPidRegulator"   />
                     <Modul Name="Zone3"           Typ="CPidRegulator"   />
                     <Modul Name="Zone4"           Typ="CPidRegulator"   />
                     <Modul Name="Zone5"           Typ="CPidRegulator"   />
              </Network>
       </Software>

</Framework>
```

Beispiel für eine Applikationsbeschreibung unter Lasal Class

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Anhang

Beispiel Konfigurationen

Folgende Beispiele sind allgemein gehalten und gelten für alle verfügbare Toolchains (<Lasal> <Twincat> <Bachmann> <BuR> <Codesys>)

Einfacher Build

```
<Toolchain build="1">
    <Path>C:\\PfadZumProjekt</Path>
</Toolchain>
```

Auf Zielsystem laden

```
<Toolchain run="1">
    <Path>C:\\PfadZumProjekt</Path>
    <Connection>192.168.102.104</Connection>
</Toolchain >
```

Testobjekt instrumentieren

```
<Toolchain testobject="1">
    <Path>C:\\PfadZumProjekt</Path>
    <Export>C:\\PfadZumTestobjekt</Export>
</Toolchain >
```

Testobjekt aus Konfigurationsdatei erstellen

```
<Toolchain testobject="1">
   <Path>C:\\PfadZumProjekt</Path>
   <Export>C:\\PfadZumTestobjekt</Export>
   <config>C:\\PfadZurKonfiguration</config>
</Toolchain >
```

Applikation generieren

```
<Toolchain generate="1">
   <Path>C:\\PfadZumProjekt</Path>
   <Export>C:\\PfadZurAblage</Export>
   <config>C:\\PfadZurKonfiguration</config>
</Toolchain >
```

Testlauf auswerten

```
<Toolchain analyse="1">
    <Path>C:\\PfadZumTestobjekt</Path>
    <Connection>192.168.102.104</Connection>
</Toolchain >
```

Bibliothek erstellen

```
<Toolchain library="1">
    <Path>C:\\PfadZumTestobjekt</Path>
    <Export>C:\\PfadZurAblage</Export>
</Toolchain >
```

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Kombination von Attributen

```
<?xml version="1.0" encoding="utf-8" ?>
<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Lasal testobject="1">
        <Path>C:\\Hoox\\Generator\\Sigmatek\\Software\\Sample\\sps</Path>
        <Export>C:\\Hoox\\Test</Export>
    </Lasal>

    <Lasal build="1" run="1">
        <Path>C:\\Hoox\\Test</Path>
        <Connection>192.168.102.104</Connection>
    </Lasal>

    <Lasal analyse="1">
        <Path>C:\\Hoox\\Test</Path>
        <Connection>192.168.102.104</Connection>
    </Lasal>

</Build>
```

1. Testobjekt erstellen
2. Testobjekt übersetzen, auf das Zielsystem laden und starten
3. Testlauf starten und Ergebnis auswerten

Testlauf mit kombinierten Attributen

```
<?xml version="1.0" encoding="utf-8" ?>
<Build xmlns="https://hoox.software"
              xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://hoox.software https://hoox.software/generator.xsd">

    <Lasal testobject="1">
        <Path>C:\\Hoox\\Generator\\Sigmatek\\Software\\Sample\\sps</Path>
        <Export>C:\\Hoox\\Test</Export>
    </Lasal>

    <Lasal build="1" run="1" analyse="1">
        <Path>C:\\Hoox\\Test</Path>
        <Connection>192.168.102.104</Connection>
    </Lasal>

</Build>
```

1. Testobjekt erstellen
2. Testobjekt übersetzen, auf das Zielsystem laden, Testlauf starten und Ergebnis auswerten

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

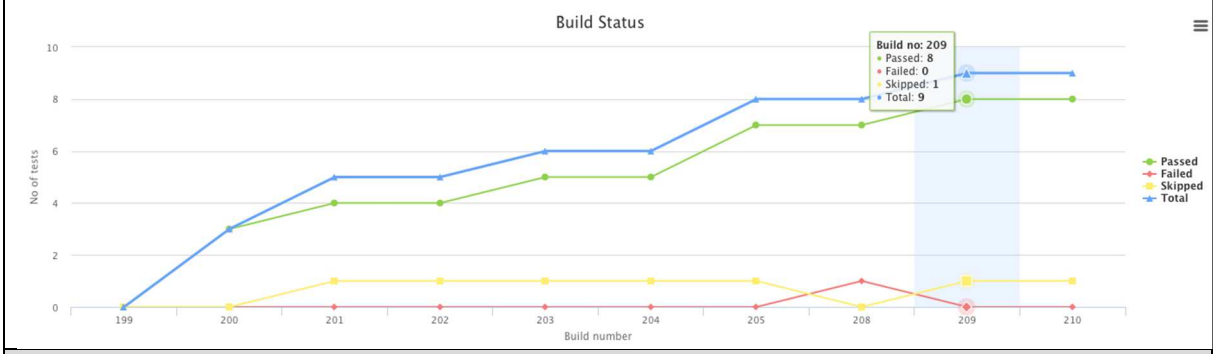
Export als Junit XML

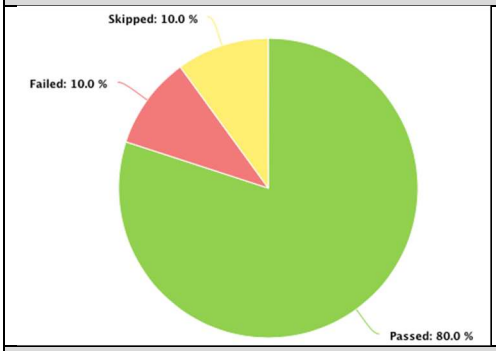
JUnit ist ein allgemein verfügbares Plugin im Jenkins Buildserver das verwendet werden kann um die Ergebnisse der ausgeführten Unittests grafisch darzustellen. Die Unittest Bibliothek erzeugt eine Ausgabedatei die von diesem Plugin gelesen werden kann.

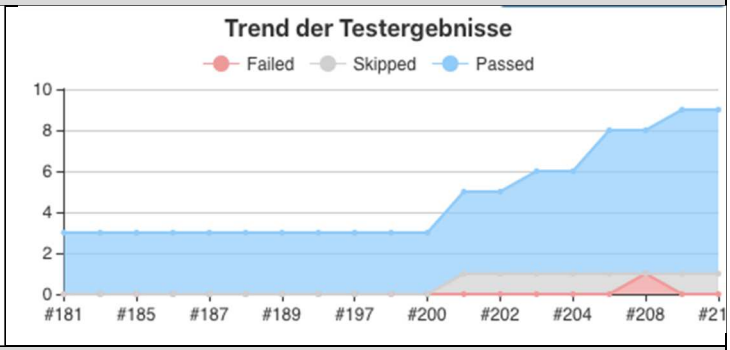
JUnit-Plugin : Tabellarische Darstellung

Chart	Package/Class/Testmethod	Passed	Transitions	210	209	208	205	204	203	202	201	200	199
☐	⊖ POUs	89% (98%)	2	0.700	0.700	0.560	0.560	0.280	0.280	0.140	0.140	0.000	N/A
☐	⊖ Awesome_Function	100% (100%)	0	0.420	0.420	0.420	0.420	0.140	0.140	0.000	0.000	0.000	N/A
☐	TF1.1	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF2.2	100% (100%)	0	0.140	0.140	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A
☐	TF2.3	100% (100%)	0	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A
☐	TF2.4	100% (100%)	0	0.140	0.140	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A
☐	⊖ FB_GreatFunctionblock	100% (100%)	0	0.140	0.140	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF1.3	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF2.5	100% (100%)	0	0.140	0.140	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
☐	⊖ PRG_Main	89% (94%)	2	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.000	N/A
☐	TF1.2	100% (100%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A
☐	TF1.4	0% (0%)	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	N/A	N/A
☐	TF2.1	100% (100%)	0	0.140	0.140	0.140	0.140	0.140	0.140	0.140	0.140	N/A	N/A

JUnit Plugin : Grafische Darstellung







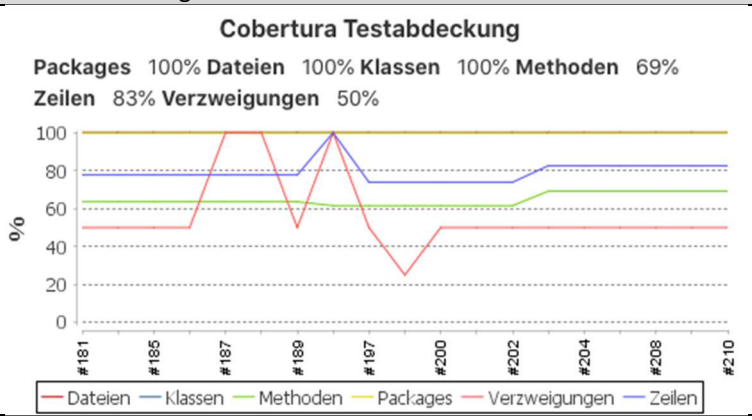
Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Export der Testabdeckung als Cobertura XML

Codecoverage

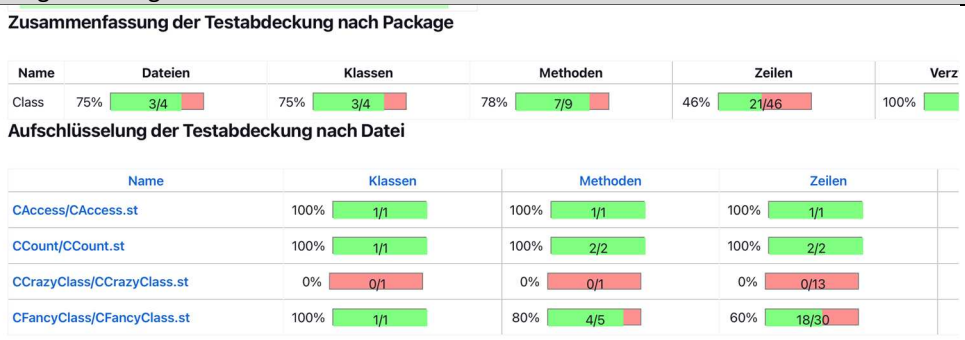
Cobertura ist ein allgemein verfügbares Plugin im Jenkins Buildserver das verwendet werden kann um die Testabdeckung zu berechnen und, noch interessanter, den Quellcode dazu anzuzeigen.
Die Unittest Bibliothek erzeugt eine Ausgabedatei die von diesem Plugin gelesen werden kann.

Cobertura Plugin : Zusammenfassung



Die Testabdeckung kann dann bis in auf die Codeebene verfolgt werden.

Cobertura Plugin : Anzeige des Quellcodes



Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

Wurden Codeteile nicht durchlaufen ist dies einfach ersichtlich (rot markierte Zeile).

Zeilen die vom Testlauf erfasst wurden

```
129     //*********************************************************************
130     //
131     //
132     //*********************************************************************
133     FUNCTION VIRTUAL GLOBAL CFancyClass::CyWork
134             VAR_INPUT
135                     EAX         : UDINT;
136             END_VAR
137             VAR_OUTPUT
138                     state (EAX)         : UDINT;
139             END_VAR
140   2
141        s32Test1 := to_dint(EAX);
142
143             IF s32Test1 = 1 THEN
144   1                 doSomething();
145             END_IF;
146   2
147             IF s32Test1 = 2
148           bTest2    = TRUE THEN
149   0                 doSomething();
150             ELSIF s32Test1 = 3 THEN
151   0                 doSenselessThing();
152             ELSE
153   2                 doNothing();
154             END_IF;
155   2
156             // IF bTest1 = TRUE THEN
157
158             strMessage := "Hallo : Welt";
159
160             strMessage := " IF Hallo THEN ";
161
162        case ClassSvr of
```

Dadurch ist es sehr einfach möglich fehlende Tests zu ermitteln und nachträglich zu implementieren.

Einschränkung

In der aktuellen Version wird nur **Zeilen-** und **Zweigabdeckung** ermittelt.
Bedingungsabdeckung erfolgt in einer späteren Version

Lizenzvereinbarung

Die Nutzung der Codecoverage Messung ist für 5 Dateien kostenlos verfügbar.
Eine Lizenz zur unbegrenzten Nutzung kann kostenpflichtig unter **order@hoox.software** angefordert werden.
Per Email wird dann eine lic-Datei gesendet die ins Verzeichnis der Generator.exe kopiert werden muss.

Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

HOOX Software
Patrick Dressel
Steinbühl 1
95233 Helmbrechts
M: +49 170 5260988
patrick@hoox.software

HOOX

[SOFTWARE]

www.hoox.software