Diplomarbeit

Automatisierung von Regressionstests eines Programms zur Halbleiter-Strukturanalyse

20.11.2002

Johann Letzel

Jens Hanisch

Agenda

I. **Überblick**
II. Das Testsystem
III. Skriptsprachen
IV. Bewertung
V. Ausblick
VI. Beispiele

Problemstellung

- Wunsch nach Automatisierung der Durchführung von Tests des gesamten XCTL-Systems

- Warum ?
  - Im Zuge des Reengineerings:
    - Restructuring = Quellcodeveränderungen
    - Syntaktische Veränderungen (Bsp. Detektor-Klassenbezeichner)
    - Semantische Veränderungen (Bsp. Trennung GUI / Programmfunktionen)
    - Neue Funktionalitäten (Bsp. „Automatische Justage“)
  - Bevorstehende Portierung auf MSVC++ (32-Bit)

→ Fehleranfällig – welche Version im CVS war noch korrekt ?

Problemstellung

- Regelmäßige Überprüfung aller essentiellen Programm-funktionalitäten notwendig ! → *Regressionstest*
- Dafür notwendig: Definition von Testfällen, die bei jedem *Regressionstest* durchgeführt werden müssen
- *Regressionstestpaket* mit Testfällen zur Überprüfung von Funktionen, die sich als „anwendbar“ erwiesen haben
- Methode der Automatisierung war nicht vorgeschrieben
- Testobjekt (XCTL-System):
  - Oberflächenbasierte Windowsanwendung
  - Implementierung in Borland C++

Verwandte Arbeiten

- Diplomarbeit von Stefan Lützkendorf
  - Attributierte Klassifikationsbäume (später)
  - Kein autarker Testtreiber (später)
- Dissertation über Testen einer GUI-Anwendung
  - GUI-Repräsentation als Graphen mit GUI-Objekten (+ Properties) als Zustände (Knoten) und GUI-Operatoren als Übergänge (Kanten)
  - Problem: Abbildung der komplexen Programmfkt. auf GUI-Operatoren
- *WinRunner* von *Mercury Interactive*
  - Bietet analogen und kontext-sensitiven Modus (später)
  - Kommerzielle „Killerapplikation“ zur Automatisierung von Tests GUI-basierter Windows- und UNIX-Anwendungen
- Weitere Arbeiten zur Automatisierung von Tests mittels proprietäre Skriptsprachen

Vorgehen / Strategie

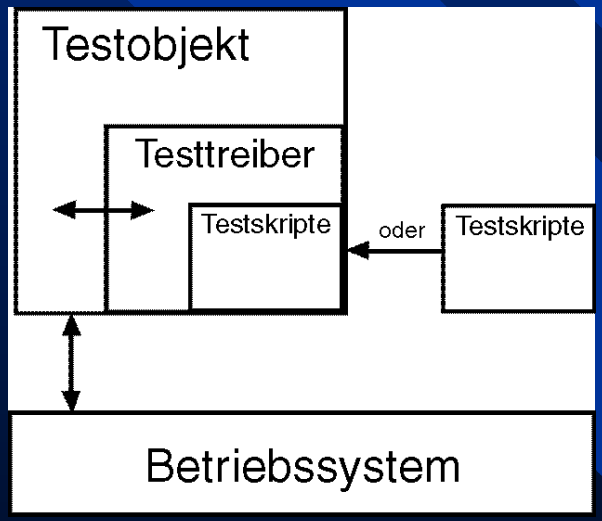
- Entscheidung für Entwicklung eines eigenen Testsystems
  - Kostenlos !
  - Auf das Problem bzw. die Aufgabenstellung anpassbar
  - Thema für Diplomarbeit ☺
- Testfallentwurf aus den Spezifikations-Dokumenten der Anwendungsfälle (UseCases) zur Erfassung der essentiellen Programmfunktionalitäten
- Testfälle aus der Sicht eines Benutzers → *Blackbox-Test*
- Konfigurationsdateien (`HARDWARE.INI, DEVELOP.INI`) sind Eingabedaten für Testobjekt
  - Basiskonfigurationen abgeleitet aus eingesetzten Messplatzdateien
  - Können für jeden Testfall den Zustand des XCTL-Systems festlegen → Anpassung / Modifikation der INI-Dateien (Bsp. `0TOPO_HARDWARE.INI, 1DIFF_HARDWARE.INI`)
  - Austausch der originalen INI-Dateien vor dem Testlauf (automatisch)

Vorgehen / Strategie

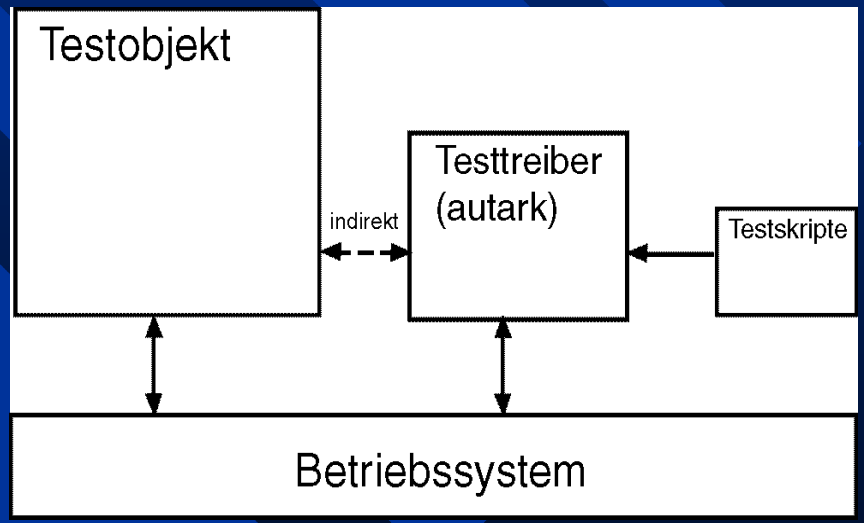
- Testfälle entsprechen einem *Integrationstest*: Komponenten (UseCases) werden möglichst separiert im Gesamtsystem getestet
  - Verwaltung im Web-Repository (übersichtlich)
  - Vermeidung von Redundanz getesteter Funktionen
  - Insgesamt Abdeckung der wesentlichsten Programmfunktionen
- Testfälle in Umgebungssimulation
  - Regressionstests schon auf Entwickler-PCs
    → frühzeitige Fehlererkennung
  - Keine Gefährdung empfindlicher Hardware
  - Alle wesentlichen Programmfunktionen sind in Umgebungssimulation durchführbar
  - Realitätstreue der Simulation wurde erheblich verbessert:
    - Motorsimulation von Stefan Lützkendorf
    - 0-dimensionaler Detektor von Kay Schützler

Vorgehen / Strategie

Varianten für die Automatisierung



1. Testtreiber im Testobjekt (Lützkendorf)



2. Autarker Testtreiber (unsere Wahl)

Vorgehen / Strategie

- Autarker Testtreiber
  - Keine Codefragmente für ein Testsystem im Testobjekt
  - Für jeden Test einer akt. Version muss nicht das gesamte Testsystem neu einkompiliert werden
  - Schwieriger: Kommunikation zwischen Testtreiber und Testobjekt
- Analog vs. Kontext-sensitiv
  - Analog: Aufzeichnen aller Eingaben (Mausklicks) über physische Koordinaten (geeignet zum Test von Zeichenprogrammen)
  - Kontext-sensitiv: unabh. von physischer Lage der GUI-Objekte (geeignet zum Test von maskenorientierten Anwendungen)
- Testfallbegleitende Auswertung vs. Auswertung am Ende
  - GUI-Test erfordert testfallbegleitende Auswertung, weil Testschritte voneinander abhängig
  - Nicht automatisierbare Testschritte erfordern Eingriff des Testers zur Laufzeit

Vorgehen / Strategie

- Kein Capture&Replay-Verfahren (Makrorekorder)
  - Im Rahmen einer Diplomarbeit zu aufwendig
- Skripte werden manuell erstellt
  - Dazu Hilfe von der Testsuite:
    - Unterstützung bei der Konstruktion gültiger Skriptkommandos
    - Generierung von Testskripten aus attributierten CTE-Diagrammen
  - Forderung nach einer benutzerfreundlichen Skriptsprache
- Automatisierung im Rahmen der Diplomarbeit ist begrenzt
  - Mit genügend Zeit- und Arbeitsaufwand ist alles automatisierbar
  - Nicht automatisierbare Testschritte sollen vom Tester zur Laufzeit manuell durchgeführt werden können → interaktive Kommandos `QUESTION, MESSAGE`
    - Bsp. Datenerhebung mit der Maus auf einer AreaScan-Grafik
    - Bsp. Visueller Vergleich von Messkurven auf dem Bildschirm

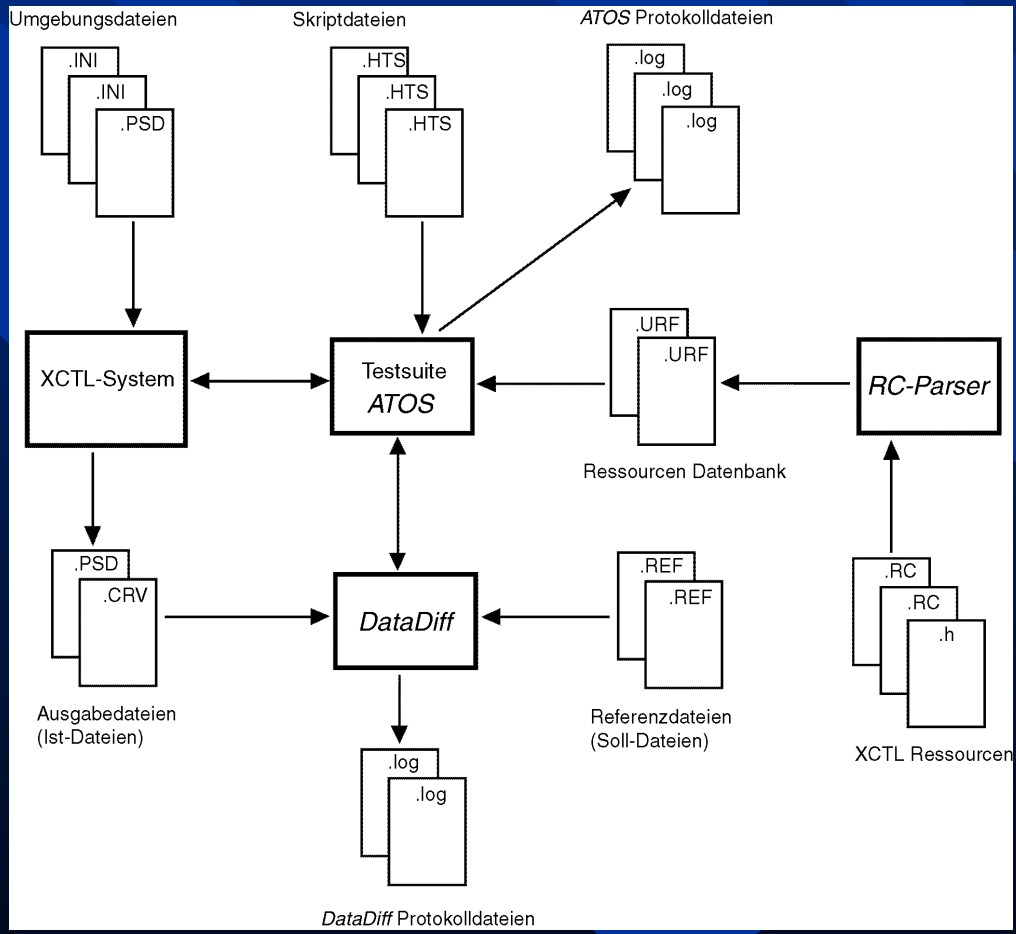
Vorgehen / Strategie

- Technologie für einen autarken Testtreiber
  - Kommunikation über das Nachrichtenkonzept von Windows
  - Bsp. `SendMessage(GetDlgItem(hDlg, IDC_LIST), LB_GETSELCOUNT,0,0)`
  - Jedem Control ist eine numerische Konstante zugeordnet
  - Skriptkommandos werden auf Windows-Nachrichten abgebildet
  - Zuordnung zwischen numerischen Konstanten und ID-Bezeichnern findet in Headerdateien statt
  - Zuordnung von ID-Bezeichnern und Controls findet hauptsächlich in Ressourcen-Dateien (RC-Dateien) statt
  - Verwaltung der Zuordnungen wird mit dem *RCParser* vorgenommen (später)
- Zusätzlich Verfahren zur Automatisierung eines Dateivergleichs
  - Ausgabedateien werden mit Referenzdateien verglichen
  - Automatisierung kein Problem → realisiert mit *DataDiff* (später)

Agenda

- ✓ Überblick
- **II. Das Testsystem**
- III. Skriptsprachen
- IV. Bewertung
- V. Ausblick
- VI. Beispiele

Arbeitsweise des Testsystems



Testdurchführung

1. `% cvs co XC010702`
2. `% cvs tag Test<Datum> XC010702`
3. Kompilieren der aktuellen XCTL-Version
4. Installation der Testsuite *ATOS* und des ATOS-Projektes
   1. `% cvs co ATOS`
   2. `% cvs co ATOS_XCTLProjekt`
5. Regressionstest:
   1. *ATOS* starten
   2. Projekt laden
   3. Testskripte auswählen und durchführen lassen
6. Auswertung der Logdateien
7. Nach gescheitertem Regressionstest:
   `% cvs tag –d Test<Datum> XC010702`

   sonst alles Ok ! ☺

Testdurchführung

Verzeichnisstruktur eines ATOS-Projektes

`.\bin\`	Testbegleitende Programme (*DataDiff*, *IrfanView*)
`.\ref\`	Referenzdateien und Kurvengrafiken
`.\env\`	Präparierte Umgebungsdateien für die Testfälle (INI-Dateien)
`.\seq\`	Testskripte (Endung `.HTS`)
`.\log\`	Logdateien für jedes ausgeführte Skript bzw. Testpaket
`.\urf\`	„Uniform Resource File“ Datenbank(en) über alle Steuerelemente, Fenster und Menüs des Testobjektes
`.\cte\`	Attributierte CTE-Diagramme zur automatisierten Gewinnung von Testskripten (später)

RCParser

- Aufbau
- Zuständigkeit
- Features
- URF-Dateien

ATOS

- Aufbau
- Zuständigkeit
- Features (Projektverwaltung, Portabilität)

DataDiff

- Vergleich von Ausgabedateien (Istdateien) des Testobjektes mit Referenzdateien (Solldateien)
- Referenzdateien sind Ausgabedateien früherer Versionen des Testobjekts mit Metainformationen zur Beachtung von Toleranzen und Positionierungen einer Zeichenkette innerhalb der Textdatei
- Referenzdateien werden manuell für jeden Testfall erstellt
- Ausgabedateien des XCTL-Systems:
  - `.ini` Konfigurationsdateien
  - `.log` Logdateien „Automatische Justage“, „Makroausführung“, „Detektor“
  - `.bk` Vergleichsscan bei der „Diffraktometrie/Reflektometrie“
  - `.crv` Scandaten beim „LineScan“
  - `.psd` Scandaten beim „AreaScan“
  - `.rep` Reportdateien beim „AreaScan“ für zusätzliche Informationen
  - `.dtn` Datenerhebungsdateien aus der Grafik eines „AreaScans“

DataDiff

Beispiel 1 TEST0000.CRV.REF

```
// Referenzdatei für den Testfall ARS.2
// Anwendungsfall Diffraktometrie/Reflektometrie -> AreaScan
// 07.02.2002


[Header]
Zerlegung=SLD
Point_Number= $eq$ 5
FileType=Standard
ArgumentMin= $eq$ -0.5000
ArgumentWidth= $eq$ 0.20000
ArgumentMax= $eq$ 0.3000
Scanaxis=Theta
TimePerScan= $eq$ 1.00

[Data]
$col 0 6 0$
-0.50000  774.0000  -1.0000
-0.30000  774.0000  -1.0000
-0.10000  774.0000  -1.0000
0.10000   774.0000  -1.0000
0.30000   774.0000  -1.0000
```

DataDiff

Beispiel 2 DEVELOP.INI.REF

```
// Referenzdatei für den Testfall AE.1
// Anwendungsfall Allgemeine Einstellungen
// 06.02.2002


[Steuerprogramm]
$back$ User= Testnutzer
$back$ Target= Testname
$back$ Current= 20
$back$ Voltage= 50
$back$ WaveLength= 1.123
$back$ Reflection= [456]
$back$ Orientation= [123]
$back$ Comment= Test!!
$back$ Substrat= TestSub
```

- *ATOS* und *DataDiff* sind nicht voneinander abhängig
- Dateivergleich könnte mit anderen Werkzeugen vorgenommen werden
  Grund: Skriptkommando LAUNCH ruft *DataDiff* im Testskript auf

Design des Testsystems

- Stand der Implementierung
- TODO
- Umfang (LOC)
- Klassenhierarchie (UML)

Agenda

- ✓ Überblick
- ✓ Das Testsystem
- **III. Skriptsprachen**
- IV. Bewertung
- V. Ausblick
- VI. Beispiele

Zweischichtenmodell

- Lowlevel-Skriptsprache ATS
- Highlevel-Skriptsprache HTS
- Zusammenspiel (Abbildung über Regeldateien → Vorteil)

Unterstützung - Skriptentwurf

Ansprüche

1. Skriptsprache HTS ist benutzerfreundlich, aber manueller Entwurf von Skripten ist fehleranfällig
   - Kommandosyntax beachten
   - Bezeichner der Oberflächenelemente kennen
   - Konsistenz zwischen Skriptkommandos und der Oberfläche (Benutzerschnittstelle) einer aktuellen Version sicherstellen

2. Wunsch nach systematischer und automatisierter Generierung von Testskripten
   - Mengen von Testskripten mit geringen Abweichungen

Unterstützung - Skriptentwurf

Realisierung

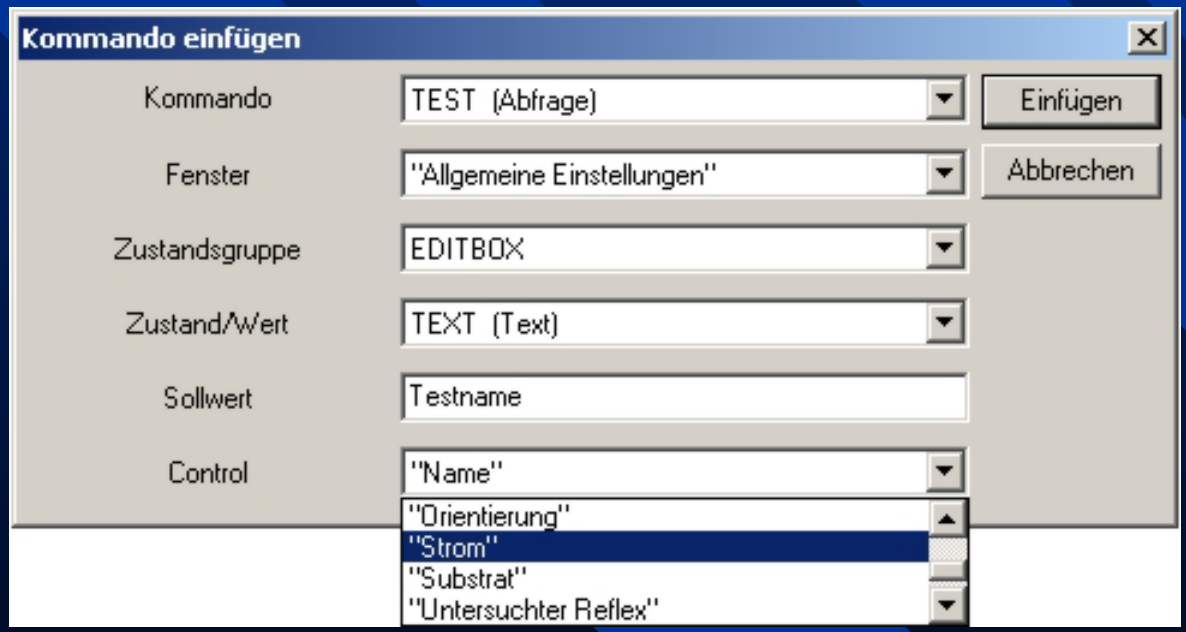
1. ATOS-Komponente
   - Hilft bei der Konstruktion eines Skriptkommandos
   - Kommandos und Parameter werden kommentiert
   - Bietet Auswahl an Parametern in Abhängigkeit vorangegangener
   - Stellt Konsistenz der Benutzerschnittstelle über Ressourcen-Datenbank (URF-Datei) sicher

2. Parser für Diagramme des *Classification Tree Editor*
   - Klassifikationsbäume werden mit CTE entworfen und attributiert
   - ATOS-Funktion (Parser) kann Testskripte aus den Attributen und der Kombinationstabelle automatisch generieren

Unterstützung - Skriptentwurf

1. ATOS-Komponente zur Kommando-Konstruktion

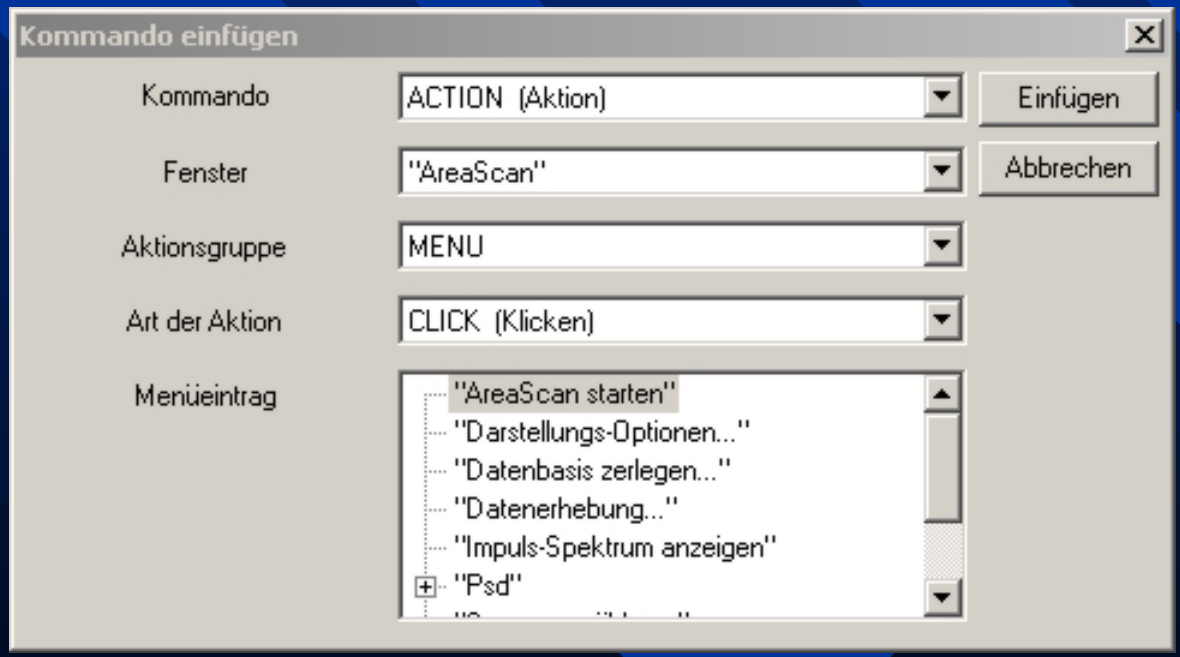
Beispiel 1: Konstruktion eines TEST-Kommandos



Unterstützung - Skriptentwurf

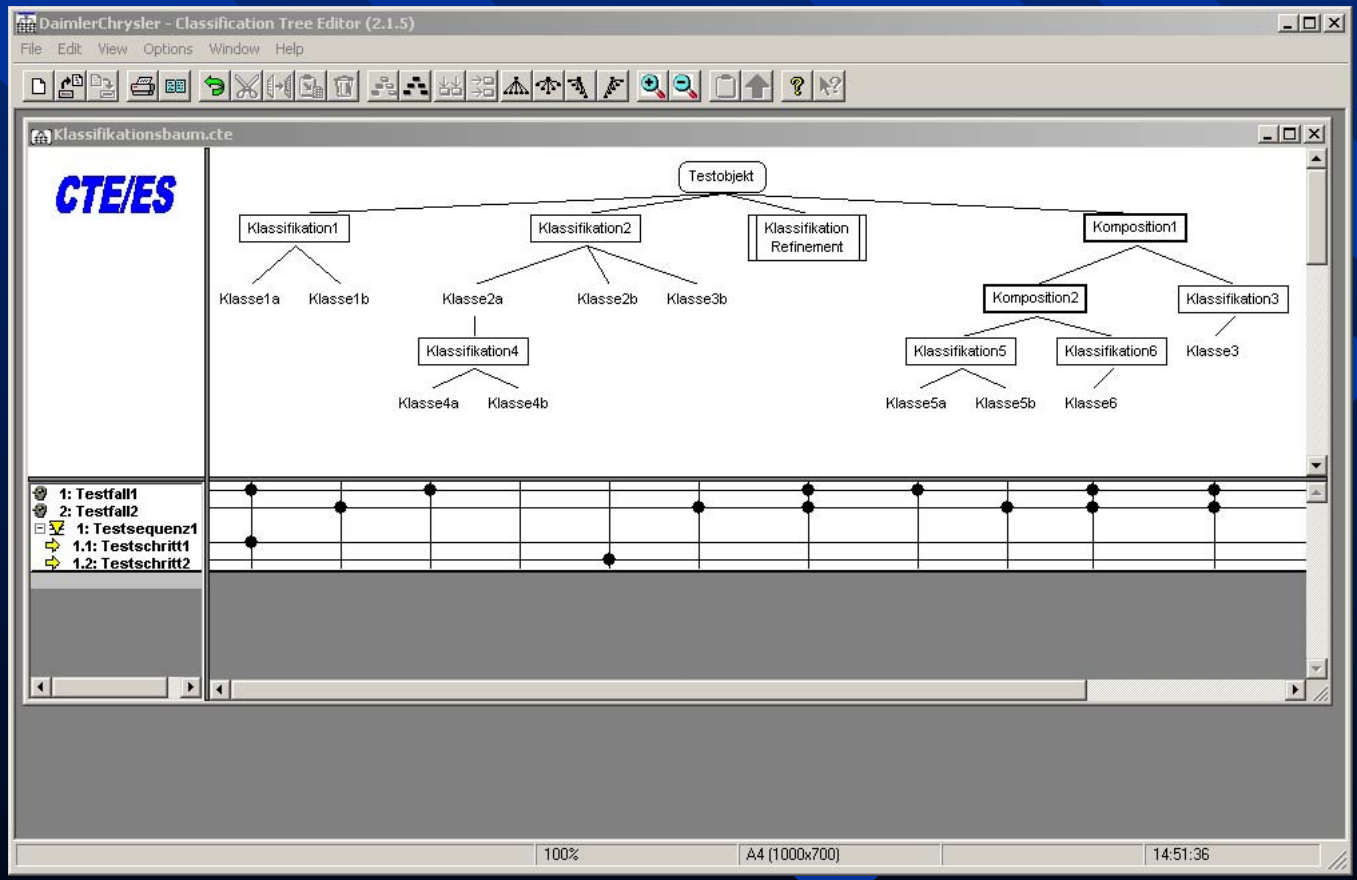
1. ATOS-Komponente zur Kommando-Konstruktion

Beispiel 2: Konstruktion eines `ACTION`-Kommandos



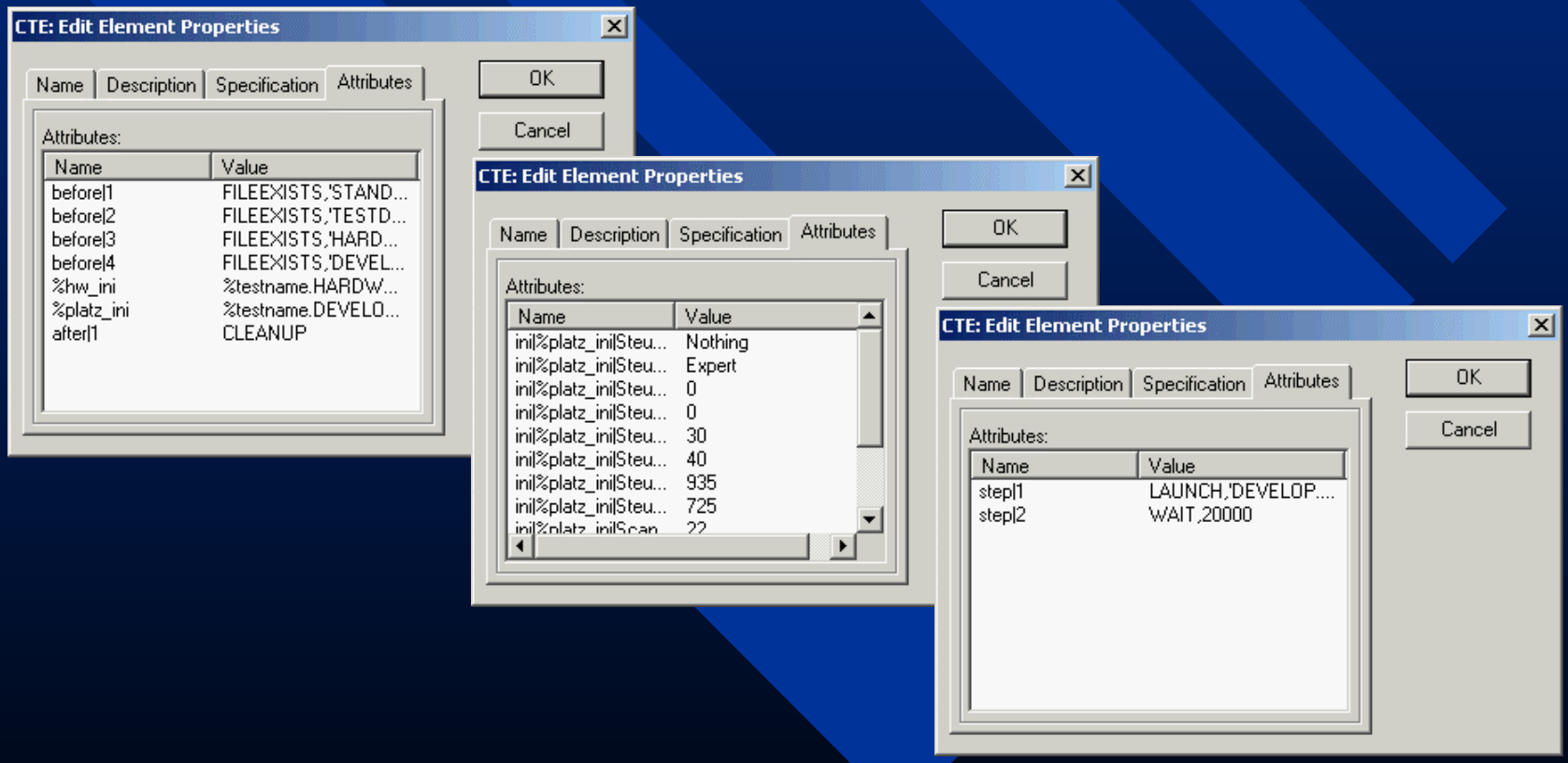
Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume



Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume



Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume

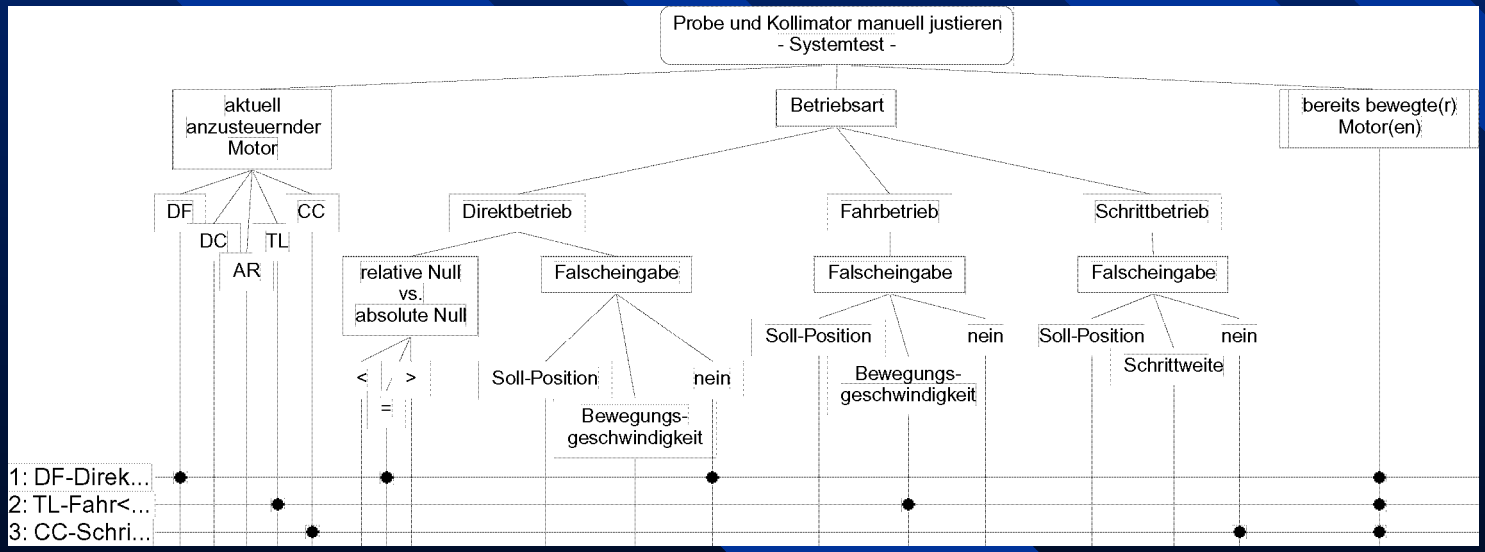
- Zustandsorientierte Klassifikationsbäume
  - Klassifikationen nach Zuständen (z.B. Bewegungszustand eines Motors)
  - Kombinationstabelle ist Menge von CTE-Testfällen

- Aktionsorientierte Klassifikationsbäume
  - Klassifikationen nach bestimmten Aktionen auf der Benutzerschnittstelle (z.B. mögliche Aktionen auf der Dialogbox „Referenzpunktlauf“)
  - Kombinationstabelle ist Menge von CTE-Testsequenzen
  - Attributierung einfach, da Klassen-Elemente direkt in Skriptkommandos übersetzt werden können

Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume

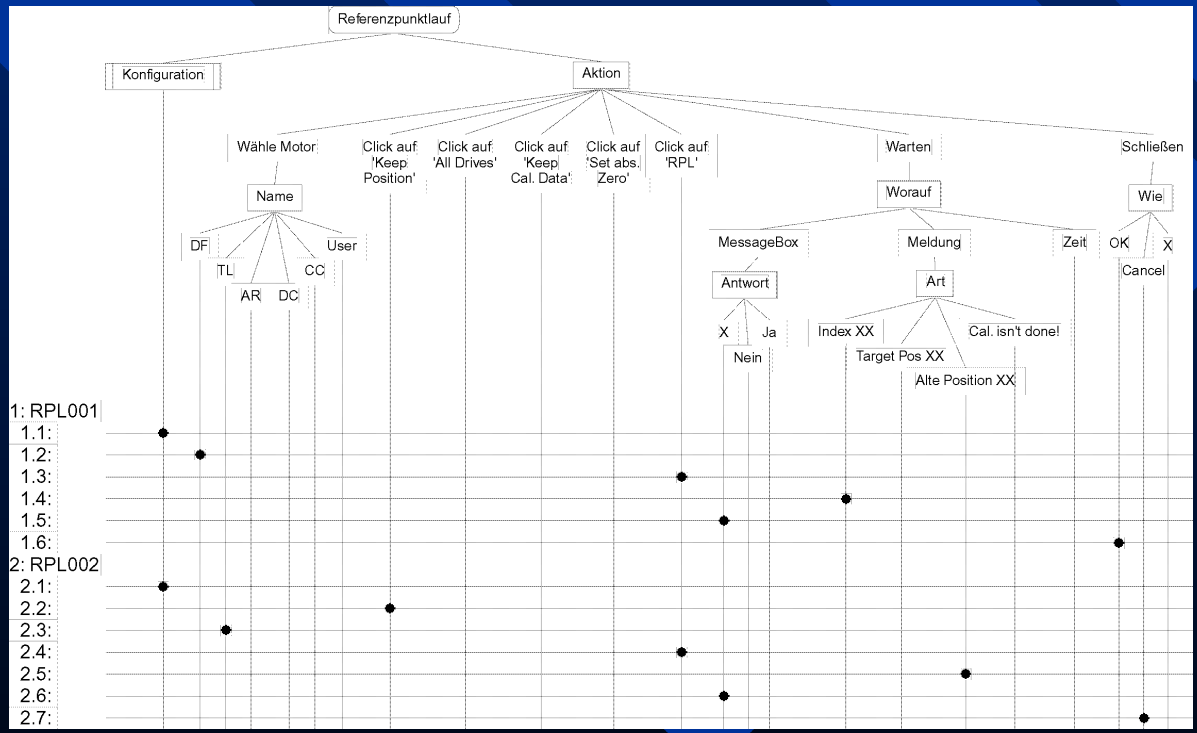
Beispiel 1: Zustandsorientierter Klassifikationsbaum



Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume

Beispiel 2: Aktionsorientierter Klassifikationsbaum



Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume

- Zustandsorientierte Klassifikationsbäume - Schwierigkeiten
  - Komplexe Diagramme schwer zu attributieren (mitunter gar nicht möglich) → fehleranfällig !
  - Positionierung der Auswertungsschritte mittels `step`-Anweisungen in einem CTE-Testschritt ist extrem schwierig

- Aktionsorientierte Klassifikationsbäume – Schwierigkeiten
  - Zeitverbrauch bestimmter Testschritte abhängig von vielen komplexen Zusammenhängen → Zeitverhalten müsste beim ersten Durchlauf ermittelt werden (Referenzdateien dito)
  - Konstruktion sinnloser Testsequenzen leicht möglich
    Grund: Abhängigkeit der GUI-Events
    (Bsp. „Ok“-Button nur im geöffneter Dialogbox anklickbar)

Unterstützung - Skriptentwurf

2. Attributierte Klassifikationsbäume

Fazit

- Viel Arbeit für mäßigen Erfolg ! ☹
- Strukturierte Gewinnung von Testsskripten mit Klassifikationsbaum-Methode sehr aufwändig und fehleranfällig
  - Schon bei Konstruktion der Klassifikationen müssen widersprüchliche Testfälle durch best. Kombinationsmöglichkeiten vermieden werden
- Geeignet für schnelle Konstruktion ähnlicher Testskripte mit geringen Abweichungen durch Copy&Paste einer durchführbaren Testsequenz
- Geeignet für Testfallverwaltung in einer einzigen Datei
  → bei Bedarf können Testskripte für *ATOS* extrahiert werden

Agenda

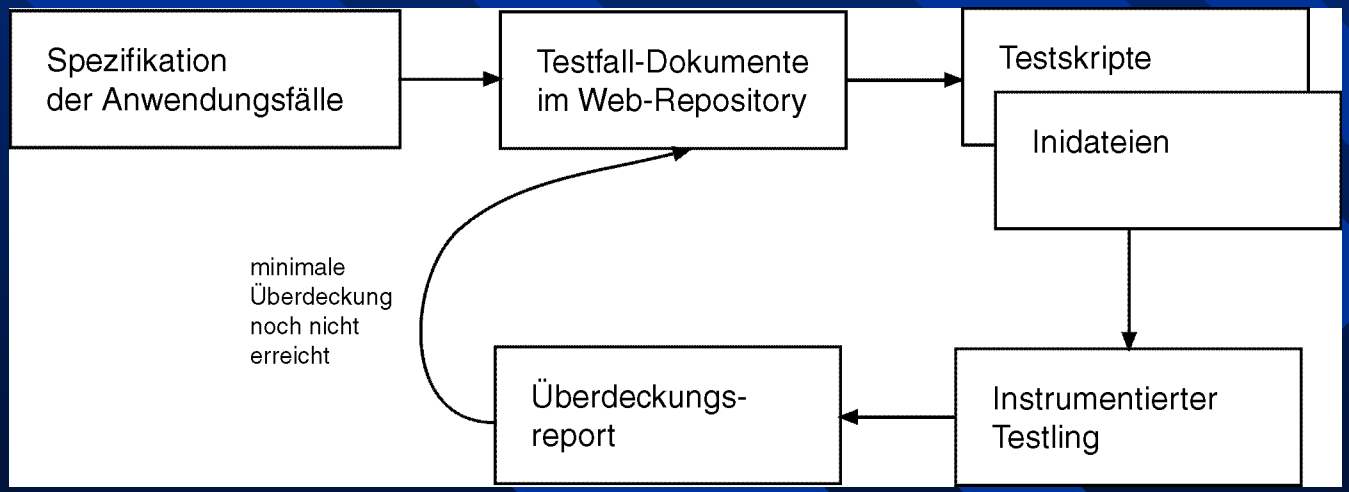
- ✓ Überblick
- ✓ Das Testsystem
- ✓ Skriptsprachen
- **IV. Bewertung**
- V. Ausblick
- VI. Beispiele

Vollständigkeit der Testfälle

- Testfälle wurden aus den Spezifikationsdokumenten der Anwendungsfälle entworfen
- Ziel: möglichst viel Quellcode (Programmfunktionen) erfassen → Wichtig für Regressionstest !
- Nachweis mittels Metriken → dynamische Codeüberdeckung
  - C0 – Anweisungsüberdeckung
  - C1 – Zweigüberdeckung
  - C2/C3 – einfache/mehrfache Bedingungsüberdeckung
  - C4 – Pfadüberdeckung
- Wir analysieren: *funktionale Überdeckung*
  - Funktionen lassen sich Anwendungsfällen zuordnen
  - Umgekehrt lassen sich Testfälle für Anwendungsfälle erweitern, um Testlücken zu schließen
- Benötigen Werkzeug zur *Instrumentierung* des Quellcodes

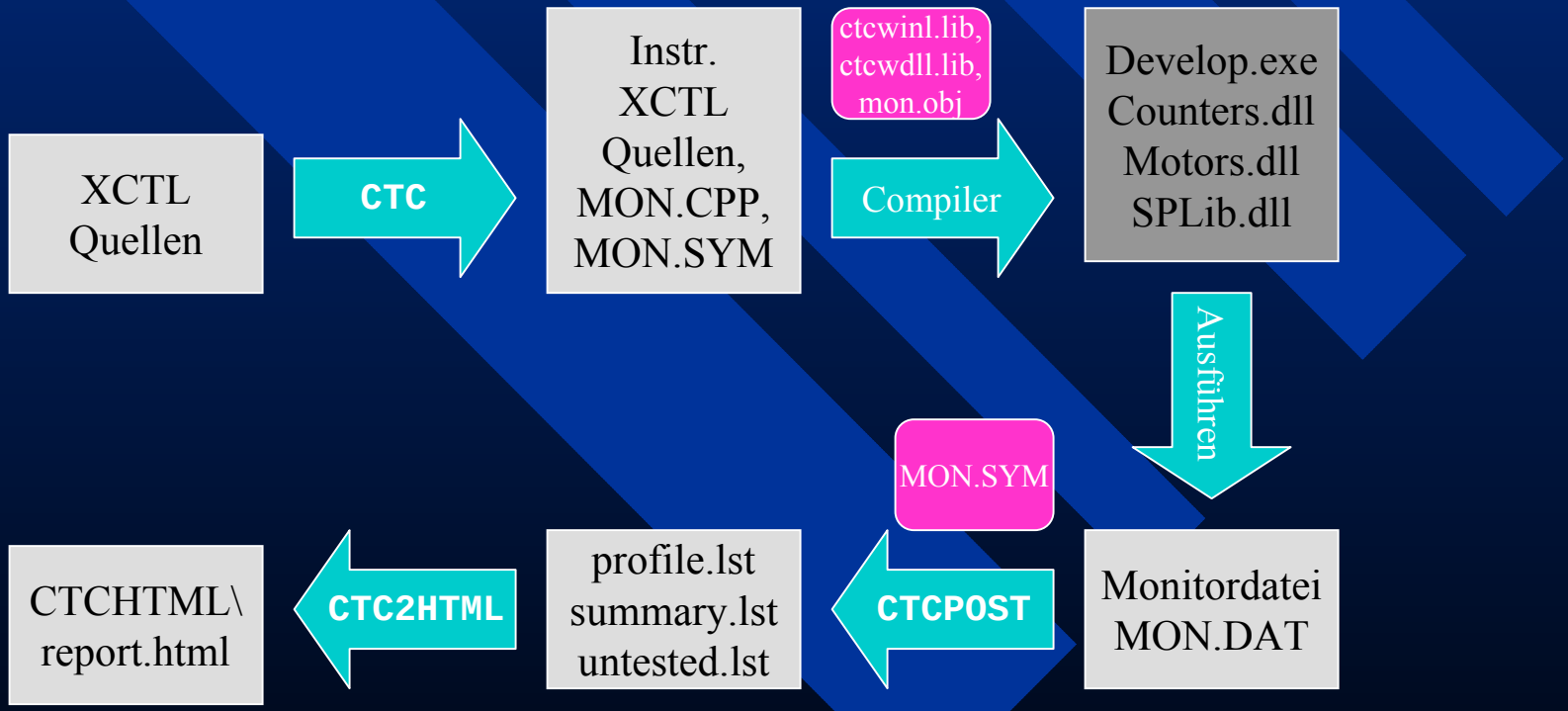
Vollständigkeit der Testfälle

Maximierung des Überdeckungsgrades



Vollständigkeit der Testfälle

- Werkzeug zur Codeüberdeckung: *CTC++ (Testwell Oy)*
- Unterstützt in alter MS-DOS Version 4.3 Borland C/C++ Code



Vollständigkeit der Testfälle

- Anwendung von *CTC++* auf XCTL-Projekt sehr aufwendig
  - Makefile mit über 2000 Zeilen !
  - Lizenz am 31.08.2002 abgelaufen
- Schwierigkeiten bei der Instrumentierung:
  - „Kernighan & Ritchie 1“ Funktionen
  - Fehlende Semikolons hinter Assembleranweisungen
  - Instrumentierung vor oder nach dem C++ Precompiler
  - Mehrfache Registrierung von Funktionen bei `#include`
- Ergebnisse waren zufriedenstellend ☺
  - Neue Testfälle wurden definiert zur Abdeckung unaufgerufener Funktionen bzw. bestehende Testfälle wurden erweitert
  - Viele tote/unbenutzte Funktionen im Quellcode aufgespürt
  - Sogar Fehler gefunden:
    - Fehlende Destruktoraufrufe (fehlende `delete`‘s)
    - Korrekturpolynom zur Umrechnung interner Motorschritte wird ignoriert

Vollständigkeit der Testfälle

CTC++ Test Coverage Report

Generated from

Monitor name : MON
Monitor generated at : Wed Mar 13 10:06:09 2002
Listing produced at : Wed Mar 13 11:47:07 2002
Threshold percent : 100 %

Overall / Files Section

Maximize

TER	%	hits/	all	File
53	% -	8/	15	AUTOJUST\M_JUSTAG.CPP
68	% -	25/	37	AUTOJUST\MATRIX.CPP
82	% -	18/	22	AUTOJUST\TRANSFRM.CPP
70	% -	19/	27	DATAVISA\M_CURVE.CPP
87	% -	54/	62	DATAVISA\M_DATA.CPP
0	% -	0/	23	DETECUSE\AM9513.CPP
0	% -	0/	18	DETECUSE\BRAUNPSD.CPP
53	% -	8/	15	DETECUSE\C_LAYER.CPP
53	% -	50/	95	DETECUSE\COUNTERS.CPP
0	% -	0/	5	DETECUSE\KISL1.C
0	% -	0/	7	DETECUSE\KMPT1.C
100	%	5/	5	DETECUSE\TESTDEV.CPP
93	% -	39/	42	DIFRKMTY\M_ARSCAN.CPP
0	% -	0/	14	DIFRKMTY\M_CCDSCN.CPP
94	% -	31/	33	DIFRKMTY\M_SCAN.CPP
75	% -	30/	40	DIFRKMTY\M_dlgdif.CPP
62	% -	32/	52	INTERNLS\L_LAYER.CPP
85	% -	34/	40	INTERNLS\M_MAIN.CPP
85	% -	46/	54	MOTRSTRG\M_LAYER.CPP
76	% -	119/	156	MOTRSTRG\MOTORS.CPP
86	% -	6/	7	MOTRSTRG\MSIMSTAT.CPP
79	% -	11/	14	SWINTRAC\DLG_TPL.CPP
87	% -	13/	15	SWINTRAC\M_DEVICE.CPP
71	% -	17/	24	SWINTRAC\M_DLG.CPP
100	%	12/	12	TOPOGRFY\M_TOPO.CPP
56	% -	84/	150	WORKFLOW\M_STEERG.CPP
67	%	661/	984	OVERALL

MONITORED SOURCE FILE : AUTOJUST\M_JUSTAG.CPP

```
#line 1 ".\INCLUDE\autojust\matrix.h"
  0   0   60 FUNCTION TMatrix::ist_homogen()
  0   0  118 FUNCTION TVektor::TVektor()
  0   0  125 FUNCTION TVektor::TVektor()
  0   0  130 FUNCTION TVektor::TVektor()
  2   2  193 FUNCTION TMatrizenListe::~TMatrizenListe()
  0   0  201 FUNCTION TMatrizenListe::ist_leer()
  0   0  207 FUNCTION TMatrizenListe::elementanzahl()
#line 1 ".\INCLUDE\autojust\transfrm.h"
1E3   0  111 FUNCTION TransformationClass::GetOrigPosBorders()
  0   0  123 FUNCTION TransformationClass::get_koordinatentrafos()
#line 23 "autojust\m_justag.cpp"
2E3 2E3   36 FUNCTION WriteToJustageLog()
  1   1   62 FUNCTION TAutomaticAngleControl::TAutomaticAngleControl()
  1   1   67 try
  1   0   82 FUNCTION TAutomaticAngleControl::Dlg_OnInit()
127 127  173 FUNCTION TAutomaticAngleControl::Dlg_OnCommand()
  1   0  844 FUNCTION TAutomaticAngleControl::LeaveDialog()
*** TER  53%  (  8/ 15) of SOURCE FILE AUTOJUST\M_JUSTAG.CPP
```

MONITORED SOURCE FILE : AUTOJUST\MATRIX.CPP

```
#line 1 ".\INCLUDE\autojust\matrix.h"
1E3   0   60 FUNCTION TMatrix::ist_homogen()
  0   0  118 FUNCTION TVektor::TVektor()
  0   0  125 FUNCTION TVektor::TVektor()
  0   0  130 FUNCTION TVektor::TVektor()
  0   0  193 FUNCTION TMatrizenListe::~TMatrizenListe()
  0   0  201 FUNCTION TMatrizenListe::ist_leer()
  0   0  207 FUNCTION TMatrizenListe::elementanzahl()
#line 5 "autojust\matrix.cpp"
6E3 6E3   13 FUNCTION TMatrix::TMatrix()
4E3 4E3   21 FUNCTION TMatrix::TMatrix()
2E4 2E4   30 FUNCTION TMatrix::~TMatrix()
1E4 1E4   36 FUNCTION TMatrix::TMatrix()
6E3   0   47 FUNCTION TMatrix::operator =()
  0   0   61 FUNCTION TMatrix::operator +()
  0   0   78 FUNCTION TMatrix::operator -()
  0   0   96 FUNCTION TMatrix::operator *()
  0   0  104 FUNCTION operator *()
3E3   0  114 FUNCTION TMatrix::operator *()
112   0  135 FUNCTION TMatrix::einheitsmatrix()
  0   0  157 FUNCTION TMatrix::invers()
 28   0  198 FUNCTION TMatrix::verschiebematrix()
 28   0  223 FUNCTION TMatrix::rotationsmatrix_x()
 28   0  238 FUNCTION TMatrix::rotationsmatrix_y()
 28   0  253 FUNCTION TMatrix::rotationsmatrix_z()
 28   0  269 FUNCTION TMatrix::transformiere()
2E3 2E3  321 FUNCTION TVektor::TVektor()
 14   0  340 FUNCTION operator *()
500   0  349 FUNCTION TVektor::mache_homogen()
500   0  381 FUNCTION TVektor::mache_kartesisch()
 28   0  411 FUNCTION TVektor::vektor_betrag()
 14   0  435 FUNCTION TVektor::skalarprodukt()
 14   0  469 FUNCTION TVektor::winkel()
874   0  504 FUNCTION TVektor::set_XYZ()
```

Einsatz in der Praxis

Beispiel 1 Neuer Fehler beim Restructuring

Skript	Test_TP.1.HTS
Datum	28.06.2002
Fehler	Im Dialog „Einstellungen Topographie“ ist keine Auswahl von Antrieb „DF“ möglich

Ursache	Fehler beim Zugriff auf Index 0 in Schleife
	`while (++i<mlGetAxisNumber())`
	in der Methode
	`TTopographySetParam::Dlg_OnInit()`

Einsatz in der Praxis

Beispiel 2 Abhängigkeiten nicht beachtet

Skript	Test_AS.1.HTS
Datum	28.06.2002
Fehler	Einlesen der Datei SCAN.MAK schlägt fehl
Ursache	Makrokommando `ChooseDevice` heißt jetzt `ChooseDetector` und wurde in `SCAN.MAK` nicht geändert

Einsatz in der Praxis

Beispiel 3 Veränderung am Programmverhalten

Skript	Test_ARS.1.HTS
Datum	23.09.2002
Fehler	Die Messwerte des 1-dimensionalen Testdetektors `PSD` werden in umgekehrter Reihenfolge in die Ausgabedateien (`*.psd`) ausgeschrieben, weshalb der Vergleich mit *DataDiff* zwischen den Istdateien (`*.psd`) und der Solldatei (`*.psd.ref`) scheitert, obwohl der Unterschied optisch kaum wahrnehmbar ist.
Ursache	In alten Programmversionen des XCTL-Systems wurde die im Dialog „Einstellungen für den PSD“ festgelegte Auslesereihenfolge für simulierte Intensitätswerte ignoriert.

Agenda

- ✓ Überblick
- ✓ Das Testsystem
- ✓ Skriptsprachen
- ✓ Bewertung
- **V. Ausblick**
- VI. Beispiele

Weiterführende Arbeiten

- Komponente zum approximativen Vergleich von Bildschirmgrafiken
- Evt. Verbesserte Komponente zum Dateivergleich
  - *DataDiff* durch beliebiges Programm ersetzbar
  - Trennung der Metainformationen (Toleranz, Struktur) von Solldaten
- XCTL-Statuszeile an Windows-Standards anpassen
  → dann Auslesen mit Skriptkommandos möglich
- Testfälle für neue XCTL-Funktionen entwerfen und als HTS-Skripte dem Regressionstestpaket hinzufügen
  - Protokollierungsfunktion von Jens Klier
  - Neue „Manuelle Justage“ von Günther Reinecker und Thomas Kullmann
- Basis-Skriptsprache ATS könnte „mächtiger“ gemacht werden
  - Zustandsinformationen über GUI-Elemente können teilweise nur über komplexe Speicherstrukturen abgefragt werden → ATS dafür auslegen
  - Austauschbarkeit von ATS-Sprache und -Interpreter untersuchen

Agenda

- ✓ Überblick
- ✓ Das Testsystem
- ✓ Skriptsprachen
- ✓ Bewertung
- ✓ Ausblick
- **VI. Beispiele**