# TREND GUIDE

## Embedded Test

Danke !

Für die **fachliche Unterstützung** bei der Erstellung dieses Trend Guides bedanken wir uns ganz herzlich bei:

- Helmuth Stahl, Expert Control
- Martin Stockl, I-Logix
- Erol Simsek und Holger Wild, iSYSTEM
- Frank Listing und Dieter Volland, MicroConsult
- Stephan Ahrends und Nikolaus Schedlbauer, National Instruments

Auch den **Sponsoren** dieses Trend Guides danken wir:


Trend Guide Medienpartner:

ELEKTRONIK PRAXIS

www.elektronikpraxis.de

Inhalt

# Embedded Test

# Vorwort

*Liebe Leser,*

***haben Sie Ihre Produkte ausreichend getestet? Sind Sie sicher, dass Sie im Fall der Fälle nachweisen können, dass Sie alle notwendigen Maßnahmen zur Verhinderung eines Schadens ergriffen haben?***

***Wenn Sie jetzt ins Grübeln kommen, lohnt es sich, diesen Trend Guide zu lesen.***

*Sie können sicher sein, dass Sie in bester Gesellschaft sind. Es gibt kaum ein Thema, bei dem schneller ein Gefühl der Unbehaglichkeit in Unternehmen entsteht. Doch das muss nicht sein. Durch kompetente Information, Beratung und Ausbildung oder durch Kooperation mit Spezialisten werden die Risiken schnell greifbar und damit auch beherrschbar.*

*Wir geben Ihnen mit diesem Trend Guide viele wertvolle Anregungen, wie Sie heute systematisch das Thema Testen und Prüfen angehen können. Außerdem erfahren Sie, welche Möglichkeiten moderne Tools bieten und welche Vorteile die konsequente Integration von Test- und Prüfverfahren hat. Neben allen Aspekten von Tools, Methoden und Prozess spielt auch der Mensch eine entscheidende Rolle. Professionelles Testen und Prüfen sind in der Hightech-Welt höchst anspruchsvolle und verantwortungsvolle Aufgaben. Es ist deshalb nicht zu unterschätzen, welche Bedeutung die Anerkennung und Förderung der Menschen hat, die diese wichtige Aufgabe erfüllen.*

*Falls Sie schon jetzt oder spätestens nach der Lektüre des Trend Guides den Entschluss fassen, das Thema Test neu anzupacken, dann denken Sie daran, dass wir Sie dabei gerne tatkräftig mit Training, Coaching, Reviews oder der Übernahme von Testaufgaben unterstützen.*

*Herzlichst, Ihr*

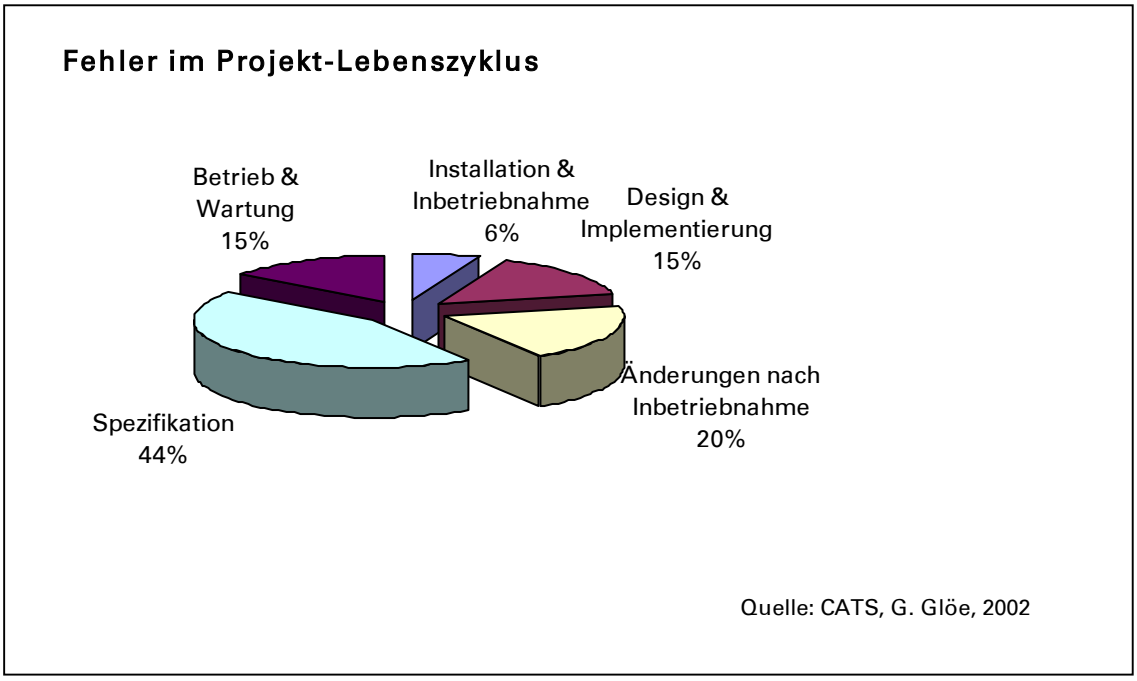
*Peter Siwon*
*MicroConsult*
*p.siwon@microconsult.com*

Trend Spots

## Für Sie kurz zusammengefasst…

**Das wichtigste Kriterium für ein Produkt ist die Time-to-Market. (Studie McKinsey)**

In der Praxis ist das Test-Endekriterium meist vom Management als Auslieferungszeitpunkt des Produkts vorgegeben.




Die Spezifikation ist die Hauptquelle aller Probleme. Diese treten meist spät zutage und sind dann nur mit großem Aufwand zu beseitigen. *(Siehe auch Trend Guide "Embedded Quality", Download unter: www.microconsult.de/jsp/trendGuideQuality.htm)*

Jenseits von Prozessen, Methoden und Tools sind es vor allem die Menschen, die zum Gelingen eines Projekts beitragen. Deshalb müssen Unternehmen das Rollenverständnis ihrer Tester als eine entscheidende Managementaufgabe begreifen.

In der klassischen Software-Entwicklung liegt der Testüberdeckungsgrad einer Testspezifikation gerade einmal bei 50 Prozent. Mit automatisierten Systemen können wesentlich mehr Regressionstests durchgeführt werden. Sie verkürzen die Zeit bis zur Markteinführung von Produkten, steigern Verfügbarkeit, Leistungsfähigkeit sowie Qualität, verbessern die Planungssicherheit und senken die Kosten.

Mit der UML nutzen Tester, Entwickler und Systemingenieure eine gemeinsame Notation, die vor Missverständnissen schützt und Zeit spart.

Fundierte Basis für den Test schaffen

# 1. Von der Anforderungsanalyse zur Testspezifikation

*Bereits in dem frühen Projektstadium der Anforderungsanalyse entscheidet sich, ob ein Produkt durch Tests schnell zur Marktreife gelangen kann. Je genauer dort formuliert ist, was ein System können soll und was nicht, desto klarer umrissen sind damit bereits die späteren Aufgaben der Tester.*

Die Anforderungsanalyse beschreibt ein System mit Ein- und Ausgaben, außerdem seine Umgebung und die Randbedingungen, unter denen es betrieben wird. Aus dieser Analyse resultiert die Anforderungsspezifikation, die den Kontext des Systems darlegt, also was es zu tun hat (funktional) und welche Qualitäten es dabei erfüllen muss (nichtfunktional). Funktionale Anforderungen sollten dabei am besten durch Sequenzen dargestellt sein. Generell werden Anforderungen heute meist noch mündlich mit dem Auftraggeber abgesprochen, obwohl eine schriftliche, im besten Fall sogar modellbasierte Fixierung späteren Problemen vorbeugen könnte. Bei mündlichen Vereinbarungen empfiehlt sich in jedem Fall eine ausführliche Testspezifikation. Außerdem sollte der Tester bei der Formulierung der Anforderungsspezifikation anwesend sein, damit das spätere System testbar ist.

Eine weitere Absicherung ist die Beschreibung von Funktionen, die das System nicht erfüllen soll, was aus Wettbewerbsgründen von Marketing und Vertrieb meist ungern gesehen wird. Doch ist es die Hauptaufgabe der Anforderungsspezifikation, den Rahmen für Entwicklung und Test so eng wie möglich abzustecken.

| Qualitätsmerkmal | Teilmerkmale |
|---|---|
| Funktionalität | Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit |
| Zuverlässigkeit | Reife, Fehlertoleranz, Wiederherstellbarkeit |
| Benutzbarkeit | Verständlichkeit, Erlernbarkeit, Bedienbarkeit |
| Effizienz | Zeitverhalten, Verbrauchsverhalten |
| Änderbarkeit | Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit |
| Übertragbarkeit | Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit |

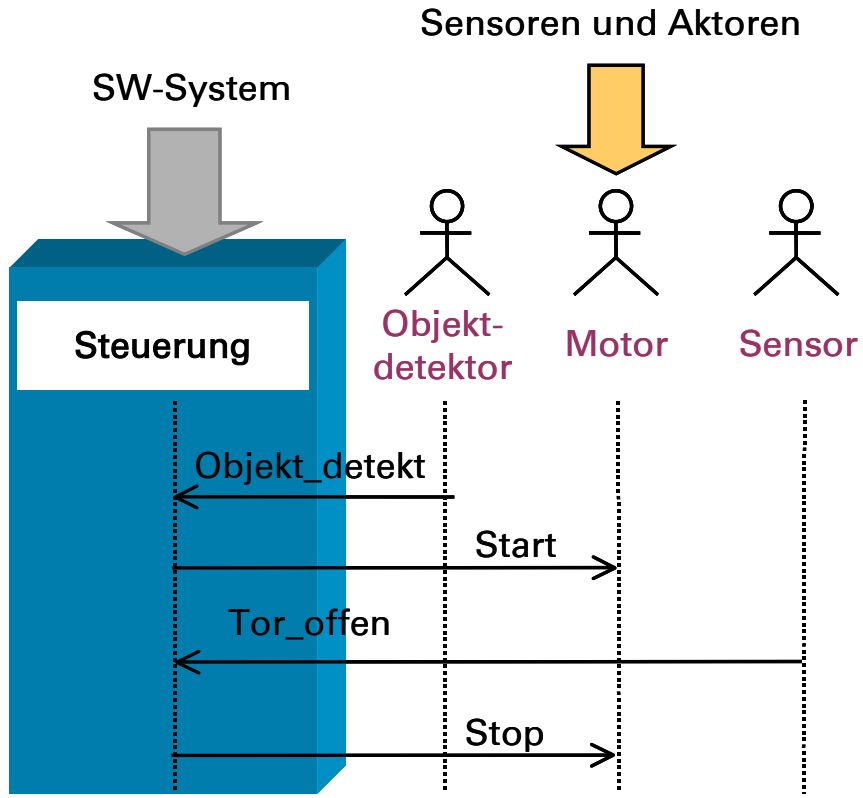
*Die Qualitätsmerkmale der Norm ISO/IEC 9126 sind – je nach Projekt – aus Kosten- und Aufwandsgründen unterschiedlich zu gewichten. Bei Embedded Systemen sollte der Schwerpunkt auf dem Zeitverhalten liegen, da dieses am meisten zur Produktqualität beiträgt.*

Eine professionelle Anforderungsspezifikation nennt auch die so genannten Stakeholder, also die Personen, die ein berechtigtes Interesse an dem System haben, unter anderem Entwickler, Tester, Management und auf Kundenseite den Besteller sowie Nutzer. Bei letzterem wird differenziert zwischen dem Primäranwender, der tagtäglich mit dem Produkt arbeiten muss, und dem Sekundäruser, wie beispielsweise einem Administrator, der nur gelegentlich darauf zugreift.

Außerdem beschreibt die Spezifikation den Zweck des Systems, der sich eng an seinen wirtschaftlichen Zielen orientieren sollte. In einem weiteren Schritt werden die Grenzen des Systems formuliert, das in diesem Stadium als Black Box betrachtet wird: Über welche Schnittstellen gelangen die Daten hinein, wer speist sie ein und wo werden sie von wem wieder abgeholt?

Die Spezifikation lässt sich gut mit der UML erweitern, um primär die Plausibilität der Anforderungen abzusichern. Die UML-Diagramme dienen zur Verfeinerung der Spezifikation und helfen dem Ersteller, tiefer in das System einzudringen. Ein Beispiel dafür ist das Sequenzdiagramm, das dafür prädestiniert ist, Abläufe zwischen dem System und seiner Außenwelt darzustellen.

*Die Sequenzdiagramme der Anforderungsspezifikation betrachten das System als Black Box, das über Schnittstellen mit Akteuren, also Sensoren und Aktoren, kommuniziert. Diese repräsentieren die Außenwelt wie beispielsweise Geräte oder die Umgebung.*




*Im Zentrum steht der Kontext des Produkts und nicht sein Innenleben.*

### Dem Fehler auf der Spur

Nach der Anforderungsspezifikation der Funktionen folgt die Fehleranalyse, die sich mit möglichen Problemen beschäftigt, wie sie Akteure im System verursachen könnten. Daraus werden später Tests oder Testsequenzen abgeleitet, die es auf Robustheit bezüglich äußerer Einflüsse überprüfen. Beispielsweise kann die Fehleranalyse zu dem Ergebnis führen, dass Ein- und Ausgaben redundant ausgelegt sein müssen, weil das System sicherheitskritisch ist. Damit soll gewährleistet werden, dass es im Fehlerfall minimal eingeschränkt oder ohne Funktionseinbußen weiterarbeitet.

Der Fehleranalyse kommt deshalb eine große Bedeutung zu, weil damit bereits in der Anforderungsspezifikation die Robustheit und damit auch Zuverlässigkeit bzw. Sicherheit des späteren Produkts festgelegt wird. Dies geschieht durch die Erkennung, Identifizierung und Klassifizierung der Fehler, aber auch durch die schriftliche Niederlegung von Toleranzen. Leider ist diese detaillierte Art der Beschreibung noch nicht weit verbreitet, obwohl sie die Arbeit des Testers spürbar vereinfacht. Optimal wäre außerdem eine Kopplung der Analyse mit Fehlerdatenbanken: Dann ist die Fehlerhistorie eines Systems jederzeit nachvollziehbar, was die Ableitung von Testfällen erleichtert.

| Softwarefehler | Hardwarefehler |
|---|---|
| Verarbeitungsfehler<br>• Fließkommazahlen<br>• Bereichsüberschreitungen<br>• Algorithmus | Ausfall<br>• Gesamthardware<br>• Peripherie |
| Speicherfehler<br>• Überschreiben von Speicher<br>• Pointer<br>• Speicherüberlauf<br>• Mangelhafte Initialisierung der Variablen | Fehlerhafte Initialisierung der Peripherie |
| Ausführung<br>• Parallelität, z.B. Deadlocks<br>• Missachtung der Zeitbedingungen<br>• Rekursionen | Sporadische Fehler<br>• Einstreuungen<br>• Time-Outs |

*Die Fehleranalyse dient der Robustheit des Systems.*

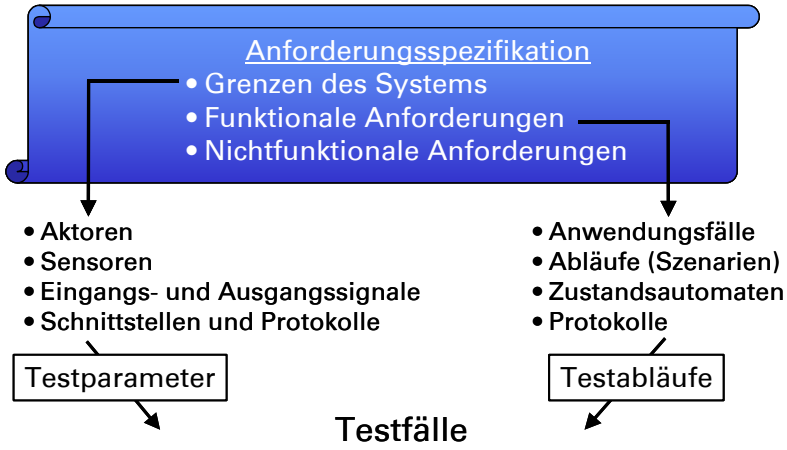
Unter den diversen Fehlerquellen sind im Stadium der Systemspezifikation vor allem die hardwarebedingten von Interesse, da Software-Bugs erst in der späteren Entwicklung produziert werden. Allerdings können in der Anforderungsspezifikation bereits Maßnahmen gefordert werden, die das Erkennen von Verarbeitungs-, Speicher- und Ausführungsfehlern ermöglichen. Fehler sollten in jedem Fall in einer Liste aufgeführt sein, die sich in Fehler, Quelle, Beschreibung, Art (systemkritisch oder -unkritisch) und Reaktion gliedert. Gegenüber einer Darstellung über Sequenzen haben Listen den Vorteil, dass es sich mit ihnen wesentlich schneller und effizienter arbeiten lässt.

| | Fehler | Quelle | Beschreibung | Art | Reaktion |
|---|---|---|---|---|---|
| 1 | Versagen Objektdetektor | Objektdetektor | Die fortlaufende Diagnostik stellt das Versagen des Objektdetektors fest. | systemkritisch | Öffnen des Tors; Abschalten des Systems |
| 2 | Versagen Motor | Motor | | systemkritisch | Warnlicht einschalten |

*Beispiel für eine übersichtliche Fehlerliste*

### Testfälle ableiten

Aus der fertigen Anforderungsspezifikation lassen sich im nächsten Schritt bereits konkret Tests und Prüfungen ableiten. Der Tester sucht dabei nach Testfällen, mit denen er das Verhalten seines Prüflings in bestimmten Situationen untersucht: Verhält er sich gemäß den spezifizierten Vorgaben und Sollwerten? Testfälle setzen sich aus Parametern und Abläufen zusammen. Parameter können gültig oder ungültig gesetzt sein, so dass sich durch geschickte Parametrisierung mehrere Testfälle generieren lassen. Der Prüfer betrachtet dafür die Grenzen des Systems, also welche Ein- und Ausgangssignale die Aktoren und Sensoren liefern. Außerdem bezieht er Schnittstellen und Protokolle mit in die Überlegung ein. Für die Testabläufe betrachtet er Anwendungsfälle, Abläufe, Zustandsautomaten und ebenfalls wieder die Protokolle.




*Die Parameter und Abläufe der Testfälle werden aus der Anforderungsspezifikation abgeleitet.*

Dabei muss im Hinblick auf die nichtfunktionalen Anforderungen eine Unterscheidung getroffen werden: Externe Qualitätsmerkmale wie Zuverlässigkeit lassen sich beispielsweise durch Dauertests untersuchen, bei denen der Prüfling die geforderte Ausfallrate erfüllen muss. Für die Zeitanforderungen muss er eine bestimmte Aufgabe in einer bestimmten Zeit abarbeiten, die der Tester den Sequenzdiagrammen der Anforderungsanalyse als Sollwert entnimmt.

Problematisch gestaltet sich dagegen der Test interner Qualitätsmerkmale wie der Übertragbarkeit. Hier müsste der Tester das System auf unterschiedliche Hardware-Plattformen portieren, was aus Zeit- und Kostengründen praktisch nicht zu realisieren ist. Deshalb beschränkt er sich hier auf Prüfungen, betrachtet beispielsweise, was an der Systemarchitektur geändert werden muss, damit sich ein System von Prozessor A nach Prozessor B portieren lässt. Prüfungen sind immer dann die erste Wahl, wenn ein System nicht ausgeführt werden kann:

> „You can't test what you can't execute!"
> *(Bruce Powel Douglass in seinem Buch "Doing Hard Time")*

Als Prüfmethoden gelten typischerweise Reviews und statische Analysen, wie beispielsweise mit Compiler oder LINT. Prüfungen sind eher allgemein und übergreifend ausgelegt, während Tests exakt ablaufen und konkretere Ergebnisse liefern. So gesehen kann der Test als eine Variante der Prüfung gelten – wenn auch die aussagekräftigste und wichtigste.

| **Testname** | **ZugEinfahrt1** |
|---|---|
| Testziel | Testen des Systemverhaltens bei Einfahrt eines Zuges mit einem Wagen. |
| Testfallbeschreibung | Ein Zug mit einem Wagen fährt von rechts in den Bahnübergang ein. |
| Erwartete Ergebnisse | 1. Ampel blinkt rot (ca. 1s).<br>2. Ampel schaltet um auf Rot.<br>3. Schranke schließt, bis sie sich in waagrechter Position befindet |
| Konkretes Vorgehen | Vorbedingungen:<br>• Schranke geöffnet<br>• Ampel grün<br><br>Vorgehen:<br>1. Drücken des rechten Einfahrschalters (1x)<br>... |

*Beispiel für die tabellarische Darstellung eines Testfalls*

Gute Planung ist die halbe Miete

## 2. Testplan als Managementaufgabe

*Die übergreifende Sicht des Managements spiegelt der Testplan wider, weil er sich als Grobgerüst auf eine Beschreibung der Funktionalitäten und den Testzeitraum beschränkt. Auf seiner Basis kann die spätere Testspezifikation Details festlegen, beispielsweise welches Feature mit welchem Testfall auf den Prüfstand gestellt wird.*

Der Testplan betrachtet das System als Gesamtheit und legt fest, wann es seine Marktreife in Bezug auf Funktionalität und Robustheit erreicht hat. Für seine Verbindlichkeit muss er von allen internen Verantwortlichen unterzeichnet werden. Eine praktikable Gliederung für den Plan liefert die Norm IEEE829, die beispielsweise den Sinn des Tests, den Prüfling samt Komponenten und Funktionalitäten, aber auch nicht zu testende Funktionen berücksichtigt, die Aufwand und Kosten unnötig in die Höhe schrauben würden:

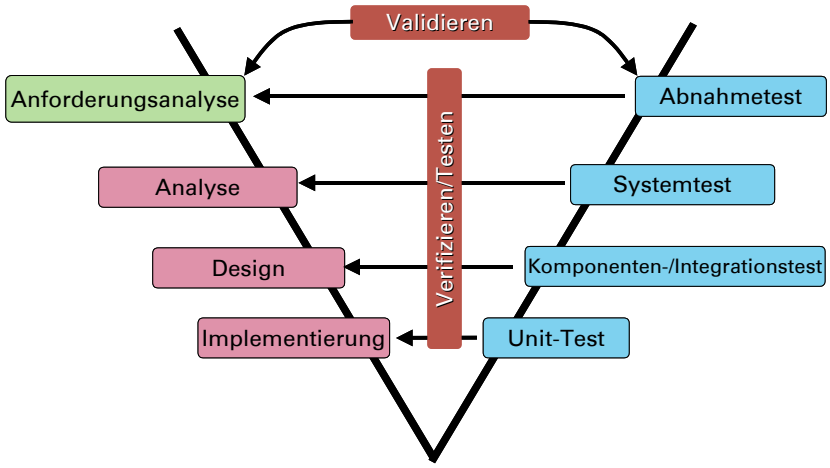
- Einführung
- Zu testende Komponenten
- Zu testende Funktionen
- Nicht zu testende Funktionen
- Vorgehensweise
- Pass/Fail-Kriterien
- Produkte
- Testtätigkeiten
- Testumgebung
- Zuständigkeiten
- Personal
- Zeitplan
- Risiken und Risikomanagement
- Genehmigungen

Der Testplaner entwickelt die Strategie vorwiegend auf Basis von Qualitätsmerkmalen, Komponenten, Produktrisiken und zu erreichenden Meilensteinen. Dabei sollte er den Teststart so früh wie möglich im Entwicklungsprozess ansetzen, da die Problembeseitigung mit fortschreitendem Projektverlauf immer teurer und zeitintensiver wird. Außerdem benötigen Tester ausreichend Vorlaufzeit für die Vorbereitung der Testfälle.

Auch die Testausführungen sind im Plan zu nennen. Die Praxis zeigt, dass Entwickler und Tester häufig ein- und dieselbe Person sind, was aber die Gefahren der Betriebsblindheit und emotionalen Bindung an das Produkt birgt. Aussagekräftiger sind deshalb meist Peer-to-Peer-Tests oder -Reviews, bei denen die Entwickler ihre Module gegenseitig testen. Im Idealfall können Unternehmen auf eine eigene Testabteilung oder spezialisierte Dienstleister wie MicroConsult zurückgreifen, die produktiver, weil fokussierter und objektiver arbeiten.

## Modelle stützen die Strategie

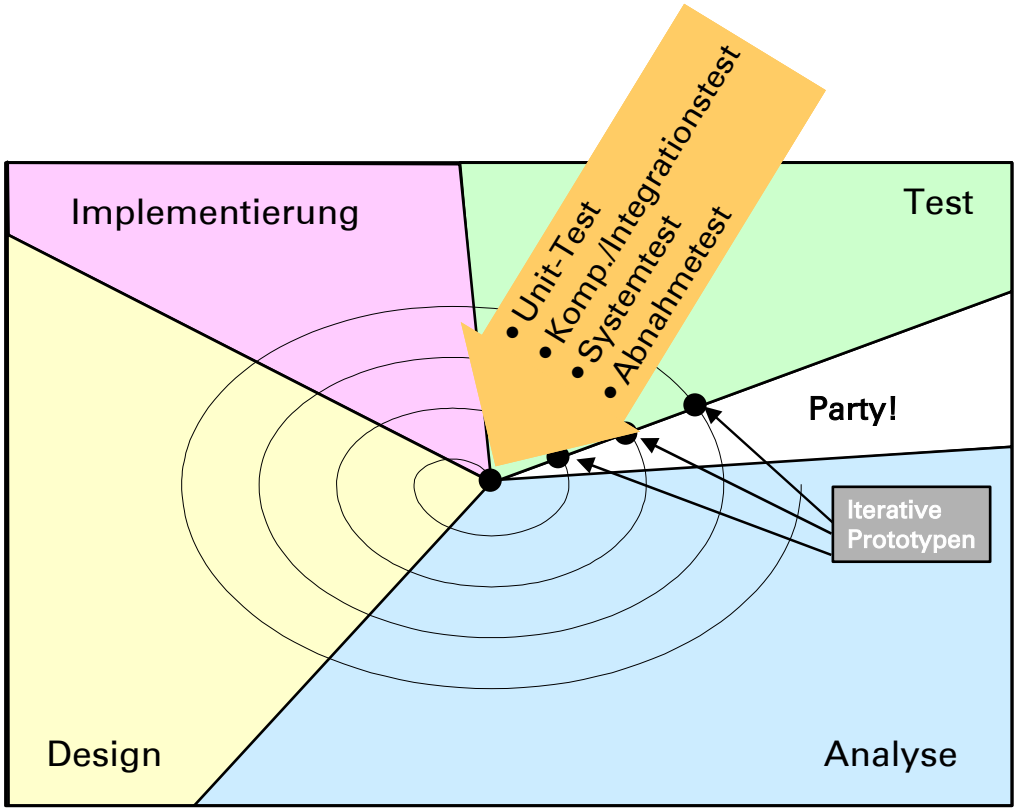
Als Referenz für die Testplanung gelten heute das V- und das Spiral-Modell. Das nacheinander abzuarbeitende V-Modell zeigt auf seinem absteigenden Ast die Workflows Anforderungsanalyse, Analyse, Design und Implementierung. Der aufsteigende Ast widmet sich ausschließlich den Bereichen Testen und Prüfen. Seine vier Workflows korrelieren dabei mit den gegenüberliegenden im absteigenden Ast, also beispielsweise der Abnahmetest mit der Anforderungsanalyse.




*Das V-Modell bildet eine wichtige Grundlage für die Testplanung, berücksichtigt aber nicht die iterative Vorgehensweise.*

Der Unit- bzw. Funktionstest beschäftigt sich mit einem Modul innerhalb beispielsweise eines C-Programms und prüft dessen Funktionsweise. Der Komponenten- bzw. Integrationstest fokussiert sich dagegen auf das Zusammenspiel der Module und Fehler in Schnittstellen, während der Systemtest das gesamte Produkt verifiziert und es gegen die Spezifikation prüft. Dem Kunden oder einem von ihm beauftragten Dritten obliegt schließlich der Abnahmetest. Hier steht normalerweise das Validieren im Vordergrund, wobei der Auftraggeber womöglich auch die Einhaltung der Spezifikation sicherstellen und deshalb verifizieren möchte.

So einleuchtend und hilfreich das V-Modell auch sein mag, spiegelt es doch nur einen Teil der Testrealität wider. Definitiv zu spät angesetzt wäre nämlich ein Systemtest nach Fertigstellung des Produkts. Besser bedient sind Testplaner deshalb mit dem weiter gehenden Spiralenmodell, das einen Systemtest mit beschränkter Funktionalität nach Fertigstellung jedes Prototyps nahe legt. Deshalb dient das V-Modell als gedankliche Grundlage, während gleichzeitig die einzelnen Entwicklungsschritte des Prüflings Berücksichtigung finden müssen.




*Beim iterativen Spiralenmodell wird der Prüfling nach Abschluss von wichtigen Entwicklungsschritten verschiedenen Tests unterzogen.*

### Qualität durch Tests sichern

Auch die Qualitätsmerkmale der Anforderungsspezifikation müssen in die Testplanung einbezogen werden. Dabei kann der Planer zwischen drei Ansätzen wählen und diese bei Bedarf kombinieren: Tests, Komponenten und Produktrisiko. Die ersten beiden Aspekte lassen sich übersichtlich durch Tabellen darstellen.

| | Funktionalität | Zuverlässigkeit | Benutzbarkeit | Effizienz | Übertragbarkeit |
|---|---|---|---|---|---|
| Wichtigkeit | 40% | 30% | 10% | 10% | 10% |
| Source-Code | + | | | ++ | |
| Unit-Test | + | | | + | |
| Komponententest | ++ | + | | | + |
| Systemtest | ++ | ++ | + | | |
| Abnahmetest | ++ | ++ | ++ | | |

*Bei dieser Planungsstrategie liegt der Schwerpunkt auf Qualitätsmerkmalen in den Projektphasen.*

| | Funktionalität | Zuverlässigkeit | Benutzbarkeit | Effizienz | Übertragbarkeit |
|---|---|---|---|---|---|
| Wichtigkeit | 40% | 30% | 10% | 10% | 10% |
| Komponente A | + | + | | ++ | |
| Komponente B | ++ | | ++ | | + |
| Komponente C | + | ++ | | | + |

*Hier steht die Qualität der einzelnen Komponenten im Vordergrund.*

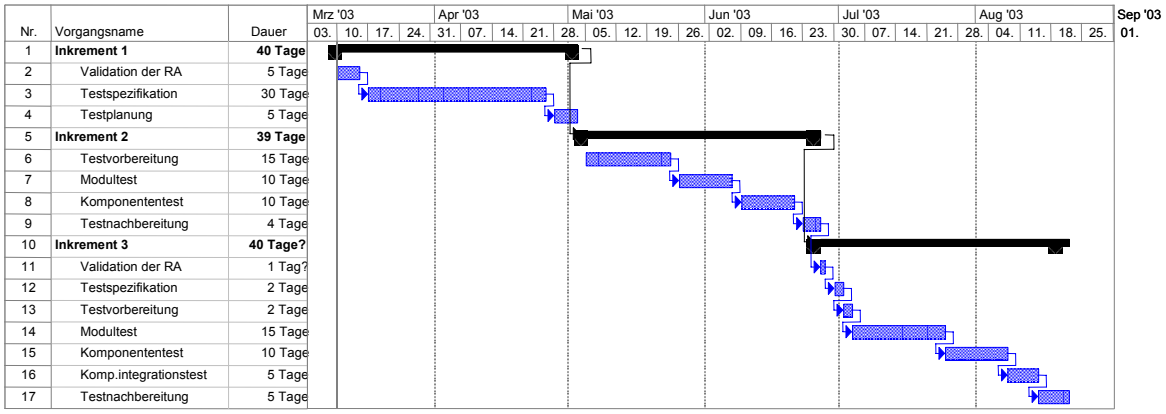
Sinnvoll ist sehr häufig die Testplanung aus Sicht der Produktrisiken als mögliche Ereignisse, die zu einem Schaden führen können:

Risiko = Wahrscheinlichkeit x Schaden

Ein hohes Produktrisiko trägt beispielsweise eine Komponente, an der viele Entwickler arbeiten (hohe Fehlerquote) und die zahlreiche Schnittstellen zu anderen Modulen besitzt (großer Einfluss auf das Gesamtsystem). Das Produkt können aber auch unerfahrene Programmierer gefährden. Die Testplanung muss dies alles berücksichtigen, allerdings ohne konkrete Hinweise darauf, wie sich die Risiken auf den späteren Test auswirken.

## Gute Startposition schaffen

Eine vollständige Testplanung berücksichtigt immer die Testvoraussetzungen, auch wenn die Norm IEEE829 dies nicht zwingend fordert. Wer beispielsweise keinen Komponententest vorschreibt, muss später mit mangelhaft robustem Code rechnen und damit, dass schwerwiegende Modulfehler erst im Systemtest ans Tageslicht kommen. In der Konsequenz geraten Systemtester und Entwickler in eine teure Endlosschleife. Deshalb muss der Testplan beispielsweise vorgeben, dass der Systemtest erst dann erfolgen darf, wenn der Komponententest mit einer bestimmten Tiefe durchgeführt wurde. Damit reduziert sich die Zahl der Schleifen erheblich. Zum Testplan gehört außerdem ein Projektplan inklusive aller Inkremente, also Teilprodukte der Entwickler, aller Tests mit Zeitangabe sowie eine endgültige Deadline für den Prüfling.




| Nr. | Vorgangsname | Dauer |
|---|---|---|
| 1 | **Inkrement 1** | **40 Tage** |
| 2 | Validation der RA | 5 Tage |
| 3 | Testspezifikation | 30 Tage |
| 4 | Testplanung | 5 Tage |
| 5 | **Inkrement 2** | **39 Tage** |
| 6 | Testvorbereitung | 15 Tage |
| 7 | Modultest | 10 Tage |
| 8 | Komponententest | 10 Tage |
| 9 | Testnachbereitung | 4 Tage |
| 10 | **Inkrement 3** | **40 Tage?** |
| 11 | Validation der RA | 1 Tag? |
| 12 | Testspezifikation | 2 Tage |
| 13 | Testvorbereitung | 2 Tage |
| 14 | Modultest | 15 Tage |
| 15 | Komponententest | 10 Tage |
| 16 | Komp.integrationstest | 5 Tage |
| 17 | Testnachbereitung | 5 Tage |

*Der Projektplan ist wesentlicher Bestandteil des Testplans, weil er alle Meilensteine im Überblick darstellt.*

## Schwieriger Schlusspunkt

Wann ein Prüfling Marktreife besitzt, wird über das Test-Endekriterium definiert und stellt den Planer nicht selten vor Probleme, da der Zeitdruck meist gegen Projektende immens wächst. In der Praxis ist das Endekriterium also fast immer die Zeit, da das Management den Auslieferungszeitpunkt vorgibt. Darüber hinaus könnte der Tester beispielsweise Metriken wie die Fehlerentwicklungsrate heranziehen, die bei einer bestimmten Kurvenabflachung signalisiert, dass das System ausreichend getestet wurde. Ein Endekriterium könnte sein, dass das System zehn Tage fehlerfrei unter 110 Prozent Last gefahren wurde. Jedem Testplaner ist natürlich bewusst, dass nie alles hundertprozentig prüfbar sein wird.

Deshalb sollte er nach dem risikobasierten Ansatz vorgehen, sprich: Komponenten mit höherer Priorität werden bevorzugt auf den Prüfstand gestellt.

Als weitere Möglichkeit bietet sich das Formulieren von Funktionsüberdeckungen an. Von n Funktionen der Anforderungsspezifikation sind dabei beispielsweise mindestens 90 Prozent zu testen. Oder der Planer wählt eine bestimmte Ein- und Ausgabeüberdeckung, also einen Prozentsatz der zu testenden Ein- und Ausgabemöglichkeiten – ein eher schwaches Kriterium, weil es nicht verrät, wie das System durchlaufen wird. Aussagekräftiger sind hier sicherlich Robustheitstests. Meist wird man sich für einen Methodenmix entscheiden, der Fokus sollte aber immer auf der Erreichbarkeit liegen. Wer bereits automatisiertes Testen eingeführt hat, kann das Test-Endekriterium genau definieren. Das strategische Abwägen von Aufwand und Nutzen bleibt aber immer eine der Hauptaufgaben des Testplaners.

Konkrete Vorgaben für Tester

## 3. Mit der Testspezifikation ins Detail

*Die Planung kann den Rahmen für alle Arten von Tests und Prüfungen liefern, als konkrete Richtschnur im Arbeitsalltag der Tester dienen letztlich die Testspezifikationen. Dabei ist Detailarbeit gefragt, beispielsweise muss aufgrund der unterschiedlichen Anforderungen für jede der vier Ebenen – Unit-, Komponenten-, System- und Abnahmetest – eine gesonderte Spezifikation erstellt werden.*

Hier kommt erneut die Norm IEEE829 ins Spiel, gibt sie dafür doch die wesentlichen Inhalte vor:

- Zu testende Funktionen
- Testverfahren
- Testskripte und -fälle
- Pass/Fail-Kriterien

Der Bereich „Zu testende Funktionen" beschreibt den Prüfling und listet seine Funktionen inklusive Kombinationen sowie Qualitätsmerkmale auf. Dabei erfolgt auch eine Verlinkung zur Anforderungs- und Design-Spezifikation. Wichtig sind nicht nur Details zu Hardware, Code oder Schnittstellen, sondern auch der Entwicklungsstand des Prüflings zum Testzeitpunkt. In einem frühen Projektstadium werden meist die Funktionen im Mittelpunkt stehen, später die Komponenten und gegen Ende das komplette System. Dies bestimmt auch die jeweiligen Testarten. Ein Systemtest sollte beispielsweise bereits dann erfolgen, wenn die Hardware ausführbar ist und bestimmte Teilfunktionen gegeben sind. Damit lässt sich die Qualität des Prüflings über die Entwicklungsfortschritte hinweg optimal beobachten.

Der Bereich „Testverfahren" beschreibt, wie die Tests ausgeführt werden sollen und welche Testfälle dabei in welchen Umgebungen zum Einsatz kommen. Dazu zählen neben dem Hardware-Target auch Testwerkzeuge: Debugger, Simulatoren, Tools für statische oder dynamische Analyse sowie Programme zur Testverwaltung und -automatisierung. Eine genaue Beschreibung der Umgebung ist deshalb so wichtig, weil damit die Reproduzierbarkeit aller Tests sichergestellt ist. So muss der Tester dem Entwickler jederzeit die Stelle des Versagens und den Auslöser dafür vor Augen führen können.

Wesentliche Bestandteile der Testspezifikation bilden außerdem der tabellarisch dargestellte Testfall und das in Einzelschritte wie Log, Start oder Stop gegliederte Testscript. Als Programmablauf macht es den Fall ausführbar. Und nicht zuletzt geben die Pass/Fail-Kriterien in der Testfalltabelle an, wann ein Test erfolgreich war oder neu durchlaufen werden muss.

### Testfall: weniger ist mehr

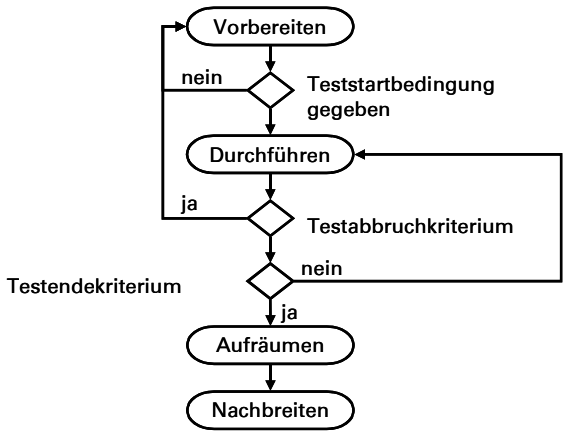
Ein Testfall beschreibt die Prüfung eines Objekts mit Eingaben und erwarteten Reaktionen an den Ausgängen unter Berücksichtigung von Vor- und Randbedingungen, beispielsweise Temperatur, Feuchtigkeit oder das Zusammenspiel mit anderen Systemen.

Unterschieden wird zwischen Muss-, Kann- und Soll-Testfällen, und bereits die Abdeckung der Kann-Testfälle spricht für einen hohen Rationalisierungsgrad des Tests. Zwei Varianten kommen zum Einsatz: Der logische Testfall beschreibt die typische Anwendungssituation für den Prüfling hardwareunabhängig mit unspezifischen Ein- und Ausgaben. Dagegen sind beim konkreten Testfall Ein- und Ausgabe sowie die Hardware festgelegt. So gesehen lassen sich aus einem logischen beliebig viele konkrete Fälle ableiten. Letztere prüfen ab,

- ob sich das Testobjekt nach der Anforderungsspezifikation verhält (Funktion) oder
- ob das Testobjekt falsche Eingaben korrekt zurückweist (Negativfunktion). Dies weist auf seine Robustheit hin.

MicroConsult rät zu möglichst kleinen Testfällen, da eine begrenzte Funktionalität die Reproduktion von Fehlern erleichtert. Kurze Fälle decken außerdem stetig Fehler auf und motivieren so den Tester. Wer große, verknüpfte Programme einsetzt, erhält dagegen häufig kein einheitliches Problembild und setzt sich der Gefahr aus, dass sich Fehler maskieren.

Unabhängig von ihrer Größe sind die Testfälle einheitlich tabellarisch zu planen: Vorbereitung, Beschreibung des Falls inklusive des konkreten Vorgehens, geforderte Testausgaben (Sollwerte), Auswertung inklusive Vorgang und verwendete Werkzeuge sowie Nachbereitung. Letztere fällt in den Bereich des Testmanagements, und hier ist beispielsweise anzugeben, welche Werte für die Metriken bereitstehen müssen.




*Jeder Testfall durchläuft von der Vor- bis zur Nachbereitung dieselben Phasen.*

## Risiken wirksam begegnen

Während der Testplan das Produktrisiko in den Vordergrund stellt, fokussiert sich die Testspezifikation auf das Testrisiko, beispielsweise dass eine Testumgebung zu spät geliefert wird oder die Entwicklung den Prüfling nicht rechtzeitig fertig stellt. Für jede dieser Eventualitäten müssen deshalb die Wahrscheinlichkeit des Eintretens, eine Gegenmaßnahme und der damit verbundene Aufwand tabellarisch festgehalten werden. Im Sinne eines proaktiven Managements sollten Risiken unter ständiger Beobachtung stehen, wobei auch diese Arbeit durch eine tabellarische Aufbereitung wesentlich vereinfacht wird.

Die verkannte Prüfmethode

# 4. Statische Analyse als Kostenkiller

*Ein Schattendasein fristet die statische Analyse heute noch in Testerkreisen. Dabei lassen sich mit ihr Kosten senken, Qualität steigern und Termine sichern. Allerdings: Geprüft wird nicht die Ausführbarkeit der Funktion, sondern der Quellcode auf seine Laufzeitfehler und damit die Robustheit des Gesamtsystems.*

Im Unit-Test müsste theoretisch für jede Funktion ein entsprechender Testtreiber geschrieben werden, der prüft, ob die eingegeben Werte zu korrekten Ausgaben führen. Sind Funktionen nicht fertig programmiert, werden sie durch so genannte Teststubs simuliert, die immer denselben Wert liefern. Da dieses Prozedere für die meisten Projekte zu aufwändig ist, verzichten heute 80 Prozent der Entwicklungsunternehmen auf den Unit-Test. Mit fatalen Folgen, weil sie deshalb keine Aussagen über die Robustheit ihrer Funktionen treffen können. Zahlreiche Fehler sind vorprogrammiert und Tester sowie Entwickler geraten deshalb in die typische Endlosschleife.

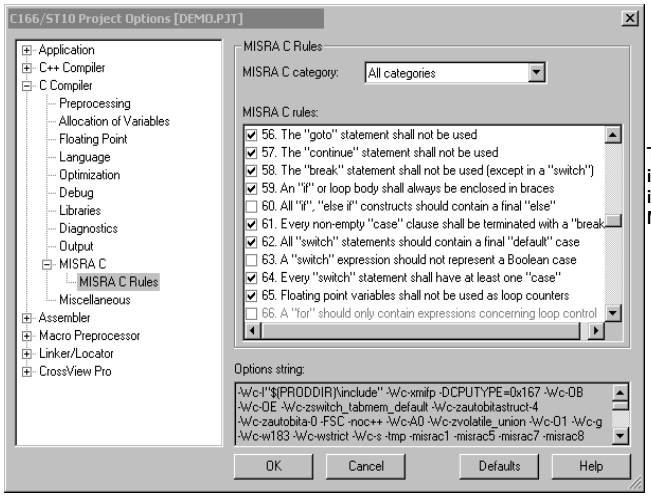
Wer dieses Problem umgehen möchte, setzt vor den eigentlichen Funktionstest die statische Analyse, ein Prüfverfahren, bei dem kein Code ausgeführt wird. Trotzdem lassen sich dadurch Laufzeitfehler erkennen, wie beispielsweise Division durch 0, Speicherüberläufe oder das Schreiben über Array-Grenzen hinweg.

### Vom Team profitieren

Als einfachste Form der statischen Analyse gilt der Compiler. Dabei muss der Entwickler den Code ohne Abschalten der Warnings kompilieren. Eine weitere Methode bilden die Reviews als formal geplante, strukturierte Analysen mit einem Team aus Gutachtern, Moderator und Schriftführer. Technische Reviews dienen dabei der Code-Prüfung, während sich informelle auf Dokumente, beispielsweise die Anforderungsspezifikation oder Architekturbeschreibung, fokussieren. Durch dieses Verfahren erweitert sich das Systemwissen im Team, dessen Verantwortungsbewusstsein steigt.

Durch gut geplante und ausgeführte Reviews lassen sich zwischen 60 und 90 Prozent aller Fehler in Spezifikation und Code aufspüren. Der Nachteil: Maximal 120 Lines of Code (LOC) können pro Stunde effektiv durch diese Methode geprüft werden – im Idealfall sind es nur 90 LOC. Reviews sind also eher bei schriftlichen Dokumenten erste Wahl, die sich nicht testen lassen. Im Code-Bereich geht der Trend dagegen eindeutig in Richtung automatisierter Verfahren.

So sorgt beispielsweise das MISRA-Konsortium mit der Verabschiedung von Standards für Software und andere Engineering-Disziplinen für mehr Sicherheit im Automobilbereich. Unter anderem garantieren die auf MISRA-C basierenden Hochverfügbarkeitssysteme, dass im Fehlerfall weder Mensch noch Umwelt Schaden nehmen. Die automatische Prüfung des Codes auf diesen Standard hin übernehmen Software-Tools wie PC-Lint von Gimpel Software, QA-C MISRA von QA Systems oder der PolySpace MISRA Checker. Auch Compiler wie der C167 von Tasking integrieren bereits eine MISRA-Prüfung.




**The rule numbering in this screen shot is based on the MISRA-C:1998**

*Beispiel-Screenshot des C167-Compilers von Tasking: Die Nummern der Regeln entsprechen der MISRA-C:1998-Norm.*

Mittlerweile wird MISRA-C nicht nur in der Automobilindustrie eingesetzt, sondern auch in anderen sicherheitskritischen Bereichen. Deshalb sollte die Testplanung dort eine Regelprüfung der Software vorsehen, bevor die eigentlichen Tests gestartet werden. Kleiner Wermutstropfen: MISRA ist heute noch auf C beschränkt.

## Mit Mathematik auf Fehlersuche

Als Königsweg im Bereich der statischen Analyse gelten heute Tools wie PolySpace von PolySpace Technologies, das im Gegensatz zu anderen Werkzeugen jede einzelne Zeile auf Basis eines aufwändigen mathematischen Verfahrens prüft. Der Fehler im Code wird dabei direkt farbig markiert, endloses Debuggen gehört damit der Vergangenheit an. Weil PolySpace den Quellcode mathematisch abstrahiert, kann es alle möglichen Konditionen analysieren, die zur Laufzeit auftreten:

- Nicht initialisierte Daten
- Benutzung von Null- oder ungültigen Pointern
- Feldgrenzen
- Ungültige arithmetische Operationen (z.B. Division durch Null)
- Overflow/Underflow von arithmetischen Operationen
- Ungültige Typ-Konversionen
- Zugriffskonflikte bei von Tasks gemeinsam genutzten Daten
- Endlosschleifen
- Nicht erreichbarer Code

Die Ergebnisse von PolySpace lassen sich durch Stubbing weiter optimieren. Das Werkzeug errechnet normalerweise für alle Parameter und Variablen aller Funktionen volle Wertebereiche, was lange Berechnungszeiten nach sich zieht. Zudem werden Fehler dargestellt, die im System aufgrund eingeschränkter Wertebereiche nicht auftreten können, beispielsweise wenn Sensor-Eingänge nicht den vollen Wertebereich abdecken. Mit Stubbing wird dieser bei der Berechnung auf das reale Maß reduziert.

Damit lässt sich allerdings eine hundertprozentige Robustheit des Systems nicht mehr gewährleisten, da beispielsweise später ein neuer Sensor vielleicht den kompletten Wertebereich benötigt. Deshalb muss der Tester hier einen guten Kompromiss finden.

| | Review nach Fagan | PolySpace ohne Stubbing | PolySpace mit Stubbing |
|---|---|---|---|
| NLOC | 3384 | 3384 | 3429 |
| Green Checks | - | 2824 | 3557 |
| Orange Checks | - | 970 | 205 |
| Gray Checks | - | 0 | 47 |
| Red Checks | - | 3 | 3 |
| Analyse Laufzeit | - | 20554 s (5 h 42 min) | 15515 (4 h 20 min) |
| NLOC für Fangan Inspektion | 3384 | 973 | 255 |
| Inspektionszeit nach Fagan (90NLOC/h) | 37,6 h (ca. 5 Tage) | 10,8 h (ca. 1,5 Tage) | 2,8 h (0,5 Tage) |
| Vorbereitungszeit durch MC | - | 1 h (Konfiguration Projekt) | 2 h (Entwicklung Stubs und Konfiguration) |
| **Ersparnis** | - | **27,8 (ca. 3,5 Tage)** | **34,8 (ca. 4,5 Tage)** |

*Dieses anonymisierte MicroConsult-Projekt zeigt, dass ein klassischer Code-Review rund zehnmal so viel Zeit in Anspruch nimmt wie die automatische PolySpace-Analyse mit Teststubs.*

Robustheit und Funktionalität gewährleisten

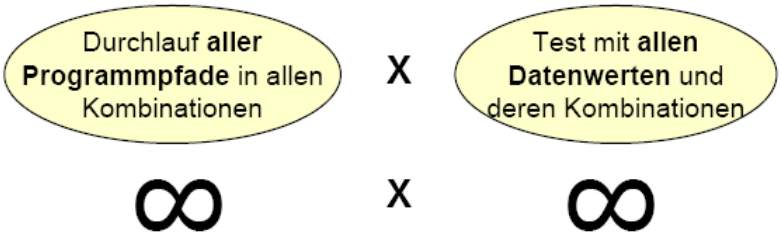
## 5. Dynamischer Test prüft den Code

*Dynamisch testen bedeutet die Ausführung der Software. Dabei wird zwischen White-Box- und Black-Box-Tests unterschieden, die sich gegebenenfalls auch kombinieren lassen. Erstere zielen auf die innere Struktur des Systems und damit auf dessen Robustheit, während zweitere die Funktionalität anhand der Schnittstellen auf den Prüfstand stellen.*

Mit dem White-Box-Verfahren arbeiten im Normalfall Unit- und Integrationstests. Sie lassen sich heute weitgehend durch automatisierte statische Analysen, wie sie z.B. mit dem Tools von PolySpace durchgeführt werden können, ersetzen, denn auch sie suchen Laufzeitfehler im System. White-Box-Tests werden mit Hilfe eines Abdeckungszieles formuliert, beispielsweise mit dem Abdeckungsmaß C0 als einfachste Variante. Dabei wird jede Programmanweisung einmal durchlaufen, wobei die statistische Fehlerfindungsrate bei 18% liegt. Der niedrige Wert liegt darin begründet, dass der C0-Test leere Zweige als hauptsächliche Problemquelle nicht berücksichtigt.

Diese Lücke schließt der C1-Test, der die Funktionen so mit Werten füttert, dass auch leere Else- und Default-Zweige durchlaufen werden. Seine Findungsrate liegt bereits bei fast 34%. Da dieses Verfahren allerdings aufwändig ist, sollten hier hilfreiche Tools wie Cantata++ von IPL zum Einsatz kommen.

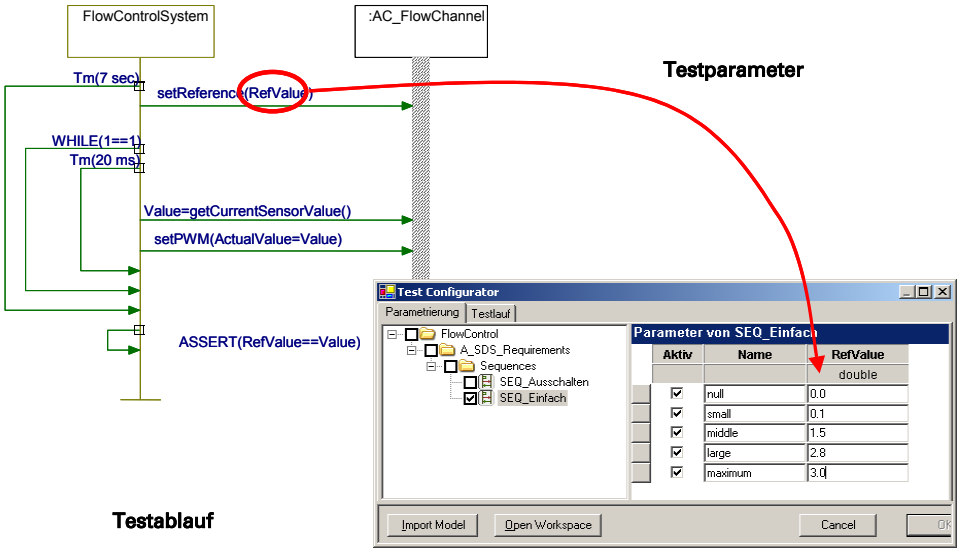
Wie schnell White-Box-Tests an ihre Grenzen kommen, zeigt die ideale Testabdeckung mit einer unendlichen Zahl von Kombinationen:




Statische Analysewerkzeuge wie PolySpace liefern dagegen Algorithmen, welche die Wertemengen und Programmpfade eingrenzen, die zu einem Fehler führen können. Deshalb sind sie dem klassischen White-Box-Test generell vorzuziehen.

### An den Schnittstellen testen

Für den Systemtest kommt normalerweise die Black-Box-Methode zum Einsatz, welche die Funktionalität des Systems, aber auch die sinnvolle Reaktion des Systems beispielsweise bei defekten Sensoren oder Stromausfall sicherstellt.




*Beispiel für einen Black-Box-Test mit MicroConsult ST, ein Framework für die Testautomatisierung: Links ist der Testablauf als Sequenz mit parametrisierbaren Variablen dargestellt. Abläufe und Parameter lassen sich mit MicroConsult ST zu verschiedenen Testfällen kombinieren.*

Die zu den vorgegebenen Eingaben erwarteten Ausgaben des Prüflings berechnet ein so genanntes Testorakel, das als Quelle die Spezifikation nutzt – außer bei Unit- und Integrationstest. Im Idealfall handelt es sich dabei um eine rechnergestützte Komponente, in der Praxis trifft meist der Tester die Voraussagen, da für jede Systemfunktion äußerst aufwändig ein Orakel programmiert werden müsste.

### Mit der richtigen Methode zum Ziel

Bei Black-Box-Tests können unterschiedliche Methoden Anwendung finden, die das Training „Grundlagen des Testens für Embedded Projekte“ bei MicroConsult detailliert vermittelt:

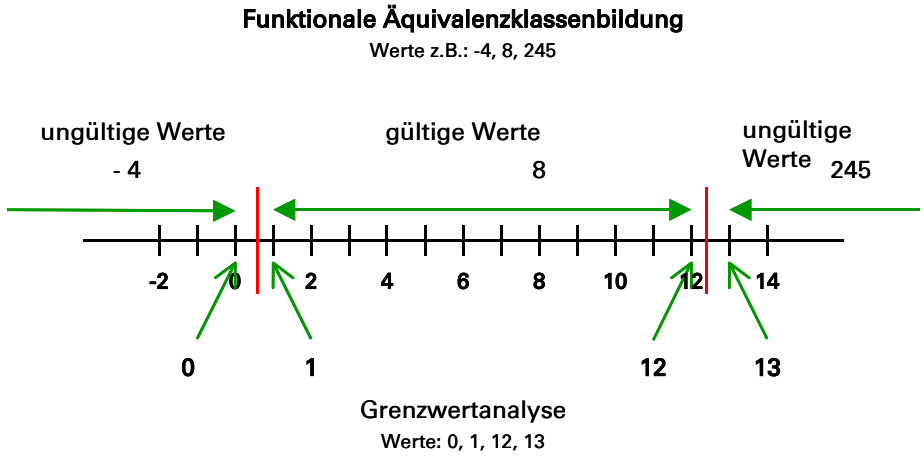
- Äquivalenzklassenbildung
- Grenzwertanalyse
- Wahrheitstabellen
- Time Partitioning
- Zustandsbasiertes Testen
- Ursache-/Wirkungsgraph
- Sequenzbeschreibung
- Error Guessing

Beispielsweise ermittelt die Funktion

int daysOfMonth(int month)

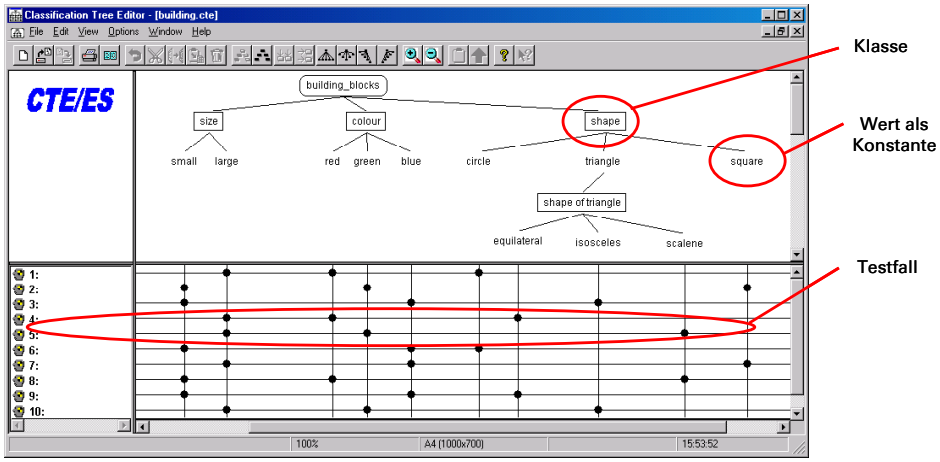
die Tage zu den jeweiligen Monaten und besitzt einen gültigen Definitionsbereich von 1 bis 12. Ein allgemeiner Zufallsgenerator könnte hier Testwerte wie -23, 13 oder 356 liefern, die nicht im Wertebereich der Funktion liegen. Als Alternative nimmt die Äquivalenzklassenbildung eine Eingrenzung vor, weil sie drei Klassen vorgibt: für untere ungültige Werte, für gültige und für obere ungültige.

Zu einem ähnlichen Ergebnis hätte man mit einem C0-Test kommen können, da dieses Verfahren noch nichts über die Grenzen aussagt. Deshalb müssen die drei Testfälle durch vier weitere für Grenzwertanalyse ergänzt werden, nämlich 0 und 1 sowie 12 und 13. Der Vorteil dieses Verfahrens liegt auf der Hand: Es lassen sich schnell gültige und Negativ-Tests generieren.



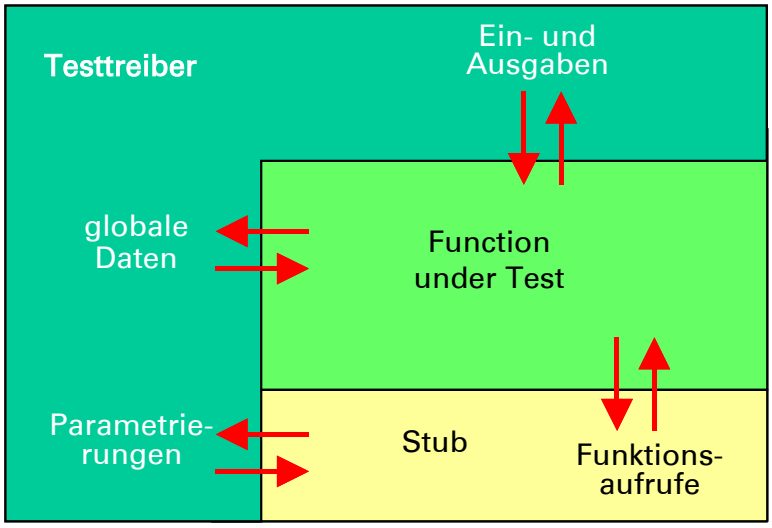

*Äquivalenzklassenbildung und Grenzwertanalyse gehören eng zusammen, denn ohne Äquivalenzklassen lassen sich keine Grenzwerte bilden.*

Problematisch gestalten sich allerdings komplexe, vielfach kombinierte Ein- und Ausgabewerte, denen mit der Klassifikationsbaummethode begegnet wird. Dabei werden die Eingabewerte in Klassen unterteilt, wobei jeder logische Wert in einer gesonderten Zeile mit einer Eingangskombinatorik versehen wird.




*Beispiel für einen logischen Testfall: Für den Klassifikationsbaum typisch wird hier mit sprachlichen Bezeichnern gearbeitet, die in konkrete Werte umgewandelt werden müssen.*

Zu einem Black-Box-Test gehört neben der Wahl der richtigen Methode die Definition eines kompletten Testbetts inklusive Treiber. Damit wird eine Funktion auf die globalen Variablen im System getestet. Stubs simulieren dabei noch nicht implementierte Funktionen.




*Die Abbildung veranschaulicht die Komplexität von Funktionstests: Alles, was die Funktion in irgendeiner Form beeinflussen kann, muss durch Testfälle abgedeckt und reproduzierbar sein.*

Eine große Herausforderung besteht in Änderungen bereits getesteter Software, die damit als ungetestet gilt. Abhilfe könnten hier stark entkoppelte Komponenten schaffen, die keine oder genau kontrollierbare Werte aneinander übergeben. Die Praxis sieht jedoch anders aus, so dass Tester auf Regressionstests zurückgreifen müssen, die allerdings aufgrund des hohen Aufwands priorisiert werden müssen. Dafür bieten sich drei Kategorien an:

Smoke-Tests erfolgen vor der eigentlichen Testdurchführung. So sehen sich Unternehmen mehreren Tausend Regressions-Testfällen gegenüber, die realistisch nicht alle durchlaufen werden können. Deshalb wird das Feld möglicher Fehlerquellen durch Ad-Hoc-Prüfungen eingeengt. Dabei gehen die Tester nach dem Gesetz der Wahrscheinlichkeit, mit der an bestimmten Stellen Probleme auftreten können.
Eine weitere Möglichkeit ist die hierarchische Testauswahl. Dabei werden die wichtigen Funktionen vorrangig getestet.

Auch hier hat der risikobasierten Ansatz Vorteile, da er das Fehlerpotenzial der einzelnen Komponenten beleuchtet. Bewährt haben sich hier Tools wie CCCC von McCabe, das sich kostenlos unter http://cccc.sourceforge.net downloaden lässt und die Komplexität von Software misst. Eine hohe McCabe-Zahl belegt eine hohe Komplexität und damit große Fehlerrate. Neben der Komplexität warten Komponenten allerdings mit weiteren Problemquellen auf:

- Hohe Änderungsfrequenz
- Erstmaliger Einsatz von Technologien, Entwicklungstechniken oder Werkzeugen
- Übertragung anderer Entwicklungen
- Viele Autoren an einer Komponente
- Entwicklung unter hohem Zeitdruck
- Hoher Grad der Optimierung
- Komponenten, die von vielen anderen Komponenten referenziert werden
- Hohe Fehlerrate in früheren Versionen
- Hohe Anzahl von Interfaces

Aus diesen Informationen kann der Tester eine Tabelle mit guter Priorisierung der Testfälle ableiten.

| | **Auswirkungen (1-5)** | **Fehlerwahr-scheinlichkeiten (1-5)** | **Nichtentdeck-barkeit (1-5)** | **Risiko (A*E*F) (1-25)** | **Priorität (H=100-125) (M=75-99) (T=1-74)** |
|---|---|---|---|---|---|
| Test-fall 1 | 5 | 5 | 5 | 125 | Hoch |
| Test-fall 2 | 3 | 5 | 1 | 15 | Mittel |
| Test-fall 3 | 5 | 4 | 4 | 80 | Tief |

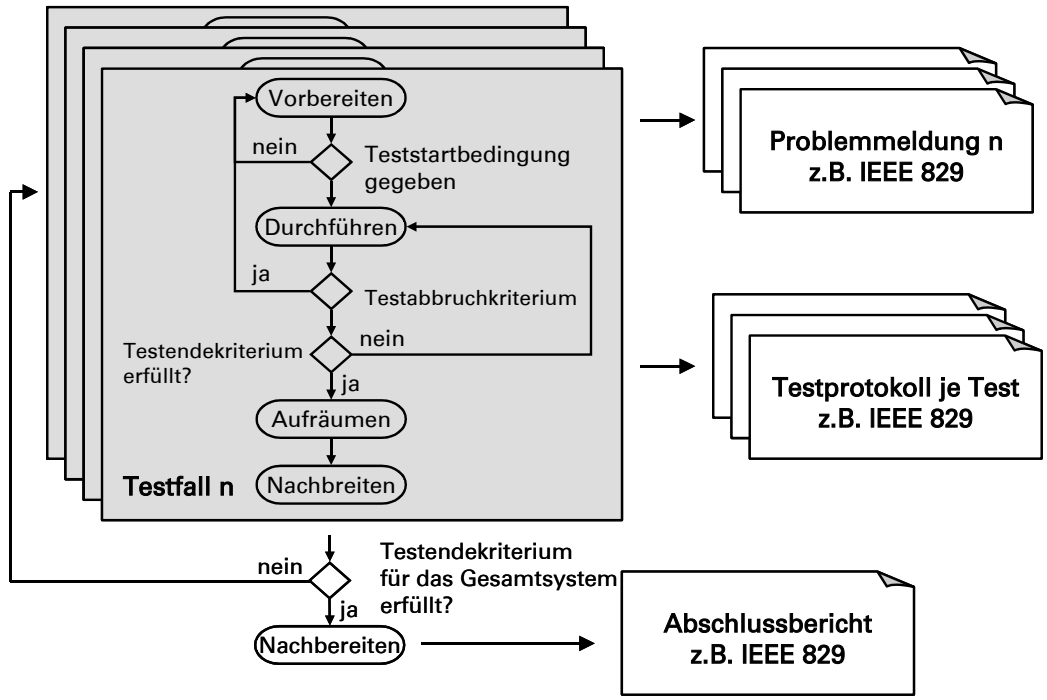
*Die Basis dieser Risikobewertung von Testfällen muss die Testplanung sein, welche die Produktrisiken auflistet.*

Stunde der Wahrheit

# 6. Aussagekräftige Testauswertung

*Eine Testdokumentation beinhaltet das Protokoll jedes Testfalls inklusive Problemmeldungen sowie einen bewertenden Abschlussbericht. Wer dabei nach der Gliederung der IEEE829 vorgeht, ist auch für Gewährleistungsklagen gut gerüstet.*

Die Testplanung gibt bereits Abbruch- und Endekriterien für jeden Testfall, aber auch für das Gesamtsystem vor. Die Testdokumentation beschreibt die Ergebnisse und liefert detaillierte Informationen zu den Zeitpunkten, an denen Fehler aufgetreten sind, damit der Entwickler diese eingrenzen und beheben kann. Auch dafür legt die Testplanung entsprechende Verfahren fest.




*Die Norm IEEE829 liefert die Struktur für Testfälle und die abschließende Testdokumentation.*

Trotz aller Vorgaben ist die ständige, intensive Kommunikation zwischen Testern und Entwicklern unabdingbar. Nicht zuletzt steigen damit die Akzeptanz des Tests und Qualität des Produkts.

### Klare Gliederung der Dokumentationen

Bei jeder Abweichung vom erwarteten Testfallergebnis muss eine Problemmeldung erzeugt werden. An dieser Stelle sei noch einmal auf die Bedeutung einer reproduzierbaren Testumgebung hingewiesen, welche die Informationen zum Fehlerzeitpunkt liefert. Die Dokumentation jedes Testfalls ist identisch gegliedert:

- Zusammenfassung
- Beschreibung
  - Tester
  - Datum und Uhrzeit
  - Umgebung
  - Eingaben
  - Erwartete Ergebnisse
  - Tatsächliche Ergebnisse
  - Fehler
  - Vorgehensschritte
  - Wiederholungsversuche
- Auswirkungen

Zudem muss jeder Testlauf durch ein Protokoll dokumentiert sein, das genaue Angaben über die durchgeführten Testfälle mit deren Ergebnissen beinhaltet:

- Beschreibung
- Aufstellung der durchgeführten Testfälle
- Ausführung
  - Name des Testers
  - Datum, Uhrzeit
  - Kurzbeschreibung
- Ergebnis
- Testumgebung
- Unerwartete Ereignisse

Auf der Grundlage des Abschlussberichtes wird die Entscheidung über die Auslieferung des Produkts getroffen, wobei Entwickler und Tester für die beschriebenen Ergebnisse verantwortlich sind. Der Abschlussbericht korreliert mit der Testplanung sowie der geforderten Funktionalität der Anforderungsspezifikation und gliedert sich in:

- Zusammenfassung
- Abweichungen
- Umfang
- Testergebnis
- Bewertung
- Aktivitäten
- Genehmigungen

Zur Bewertung von Tests bieten sich Metriken an, die allerdings aufgrund fehlender Standards umstritten sind. Wer gerne damit arbeitet, sollte sie demnach individuell erstellen. Dazu zählen beispielsweise fehler-, testfall- oder testobjektbasierte Metriken. Für den Fehlerabgleich können Tester kostenfreie Datenbanken von Bugzilla laden, einem Teilbereich des Mozilla-Browsers: https://bugzilla.mozilla.org/. Einen weiteren Bewertungsmaßstab für das System liefert der Abschlussbericht durch eine tabellarische Übersicht, in die auch die Metriken Eingang finden.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Testfälle** | | | | | | |
| | **Muss** | | **150** | **149** | **99,33%** | |
| | **Soll** | | **100** | **70** | **70,00%** | |
| | **Kann** | | **350** | **125** | **35,71%** | |
| | | | **600** | **344** | **57,33%** | |
| **Testabdeckung** | **Typ** | | **Gefordert** | **Tatsächlich** | | |
| | **C1** | | **100%** | **100%** | | |
| | **C2** | | **80%** | **85%** | | |
| | **Einfache Bedingung** | | **100%** | **92%** | | |
| **Fehler** | | | **Gefunden** | **Beseitigt** | | |
| | **Patch** | | **35** | **35** | | |
| | **Update** | | **63** | **55** | | |
| | **Gelegentlich** | | **23** | **19** | | |
| | **Offen** | | **4** | **0** | | |
| | | | **125** | **109** | **87,20%** | |
| **Fehlertrend** | | | | | | |
| | | **Jan 03** | **Feb 03** | **Mrz 03** | **Apr 03** | **Mai 03** |

*Eine Excel-Tabelle im Abschlussbericht liefert Informationen im schnellen Überblick, aus denen sich gegebenenfalls die Marktreife eines Produkts ableiten lässt.*

Auf der Kostenbremse

## 7. Design for Test

*Von der Planung über die Ausführung bis zur Dokumentation - im Projekt bildet der Test meist einen großen Kostenblock. Nicht zuletzt Testumgebungen und Werkzeuge schlagen teuer zu Buche. Design for Test (DFT) fordert deshalb vom Entwickler ein sauber strukturiertes System, das sich gut und damit kostensensitiv testen lässt.*

Wenn die Fehlerbeseitigung im Projekt aufwändiger als geplant ausfällt, entzündet sich so mancher Streit an der Frage, ob die damit verbundenen Kosten der Test- oder Entwicklungsabteilung aufzubürden sind. Mit durchdachtem DFT wäre es nie so weit gekommen. Es folgt strikt den sechs Prinzipien des Software-Engineerings:

1 *Verständlichkeit:* Der Code muss so einleuchtend sein, dass ein Tester beispielsweise selbst White-Box-Tests durchführen kann.

2 *Modularisierung:* Sie schafft gut verständliche Komponenten.

3 *Lose Kopplung:* Je unabhängiger die Module voneinander sind, desto weniger Regressionstests sind erforderlich.

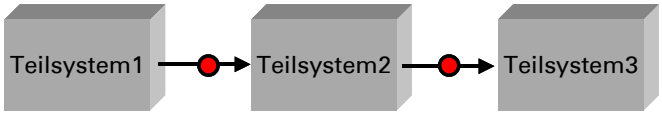
4 *Hohe Kohäsion:* Die Komponenten erbringen nur die geforderte Funktionalität.

5 *Heile Welt:* Die Units sind in sich geschlossen und arbeiten eigenständig.

6 *Magische Sieben:* Das System berücksichtigt die Erkenntnis, dass ein Mensch höchstens sieben Dinge gleichzeitig wahrnehmen und verarbeiten kann.

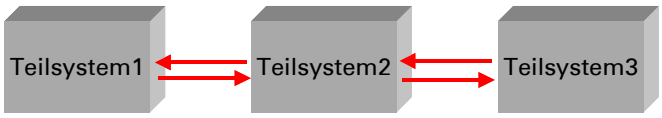
Orientiert sich das Design an diesen Vorgaben, lässt es sich im Normalfall besser kontrollieren. Dafür muss es in einen definierten Zustand gebracht werden können und Schnittstellen für die Eingabe von Testparametern bieten. Ein weiteres wichtiges DFT-Kriterium ist die Beobachtbarkeit. So muss prüfbar sein, ob das System das gewünschte Verhalten zeigt und der Tester muss Ausgaben auslesen können. Außerdem ist zu gewährleisten, dass Teile des Prüflings unabhängig voneinander kontrollier- und beobachtbar sind. Schließlich fordert DFT eine Diagnosemöglichkeit, mit welcher der Entwickler Fehler auf seine Hard- und Software abbilden kann. Dies kann beispielsweise ein Fehlerspeicher in einer Diagnosezelle sein. Der Tester liest diese aus und hinterlegt sie in einem Protokoll mit dem Test.
Die Testbarkeit eines Systems wird vor allem durch seine Architektur bestimmt. Hier einige Beispiele:

### Pipe und Filter



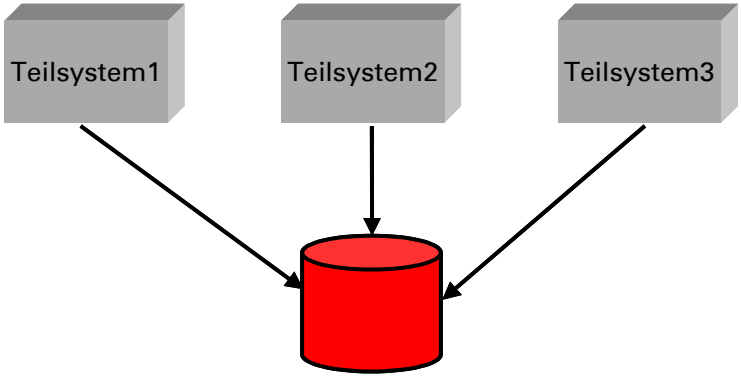

*Sind die Teilsysteme durch Pipes und Filter gekoppelt, können an jeder Stelle Messpunkte eingebracht werden. Deshalb ist die Testbarkeit des Systems gut.*

## Client/Server oder Publisher/Subscriber




*Client/Server- und Publisher/Subscriber-Systeme besitzen genau spezifizierte Schnittstellen. Sie sind gut testbar, weil auf definierte Anfragen definierte Reaktionen der Teilsysteme resultieren. Für den Test eines bestimmten Teilsystems lassen sich Komponenten einfach durch Stubbing ersetzen.*

## Shared Data




*Schwierig zu testen sind Systeme mit gemeinsamen Datenbeständen, da Komponenten und Daten eng gekoppelt und damit stark voneinander abhängig sind. Diese Prüflinge lassen sich schwer in einen definierten Anfangszustand bringen.*

Schneller zur Marktreife

# 8. Automatisches Testen

*Der automatische Test spielt seine Qualitäten vor allem bei Regressionstests aus. Diese werden in regelmäßigen Abständen gefahren , um unerwünschte Wirkungen von Veränderungen auf das Gesamtsystem aufzudecken bzw. auszuschließen. Ziel ist z.B. der Nachweis, dass alle bereits getesteten Funktionen nach einer Code-Änderung noch korrekt ablaufen.*

Investitionen in automatische Testwerkzeuge amortisieren sich im Regelfall schnell, weil damit die Kosten für die Phasen von der Planung über die Durchführung bis zur Dokumentation spürbar sinken. Regressionstests gelten nicht als eigenständige Verfahren, sondern bilden eine Klammer über alle Tests, die vier mögliche Änderungen am existierenden Code erkennen:

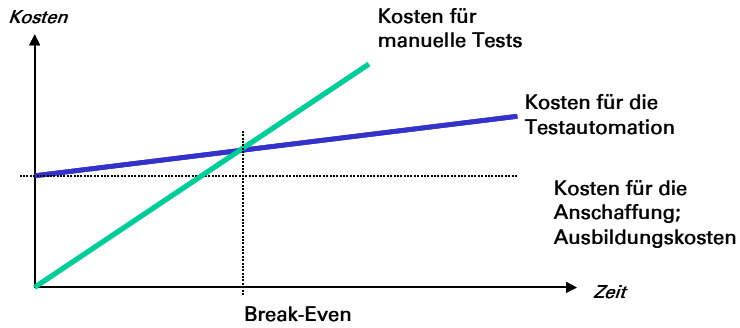
- korrektiv (Spezifikation verletzt)
- inkrementell (Spezifikation erweitert)
- adaptiv (Spezifikation geändert)
- optimierend (Spezifikation unverändert)

Bei jeder Änderung gilt der Prüfling erst einmal als nicht getestet. In der Praxis bedeutet dies meist eine Vielzahl von Regressionstests, die sich mit in einer automatisierten Testumgebung sinnvoll bewältigen lassen.

### Das richtige Tool wählen

Wer sich für eine Automatisierung entscheidet, muss darauf achten, dass sich mit dem Werkzeug Testfälle einfach erstellen und verwalten lassen. Diese sollten so weit ohne manuelle Eingriffe auskommen, dass beispielsweise auch Nachtläufe möglich sind. Außerdem sollte das Tool eine große Testbandbreite unterstützen, beispielsweise neben funktionalen Tests auf Modul- und Systemebene auch Tests zum Systemverhalten (Speicher, Performance, Stress) inklusive der Messung der Code-Coverage.

Die Testdokumentation sollte eine Anpassung von Dokument-Templates ermöglichen. Außerdem muss das Werkzeug alle gängigen Zielsysteme wie Host, HIL und Target unterstützen. Bei letzterem sollte der Tester zwischen Integration auf Betriebssystem- und Hardware-Ebene (beispielsweise durch Emulator) wählen können.

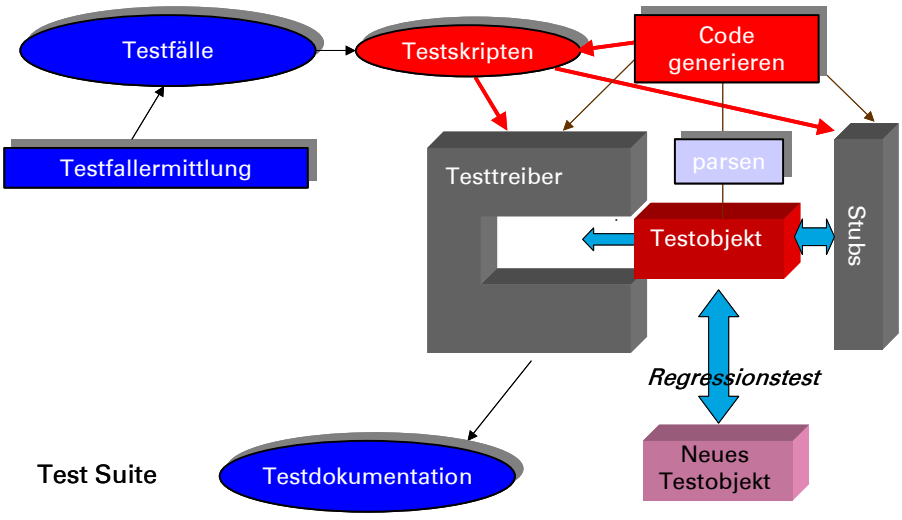



*Investitionen in Werkzeuge zur Testautomatisierung amortisieren sich schnell, da Testfälle einfach verwaltet, Tests ohne manuelle Eingriffe durchgeführt und Dokumente automatisch erstellt werden können.*

Bei diesen hohen Anforderungen ist es meist mit der Auswahl einer einzigen Lösung nicht getan. Tester müssen sich deshalb fragen, wie sie die einzelnen Werkzeuge und deren Ergebnisse unter einen Hut bringen. Deshalb sollte die Entscheidung bereits in der Designphase fallen, nämlich dann, wenn sich die Frage nach dem „Wie" des Tests stellt. Wer diverse Ergebnisse später kombinieren will, sollte darauf achten, ob sie automatisiert überprüfbar sind und ob die Formate zusammen passen. So könnte ein Excel-Makro die Prüfprotokolle parsen und ein Excel-Spreadsheet das Gesamtergebnis darstellen. Außerdem dürfen die Tools den Testprozess nicht beeinflussen, der durchgängig und unabhängig von der eingesetzten Technologie sein muss.

Am meisten profitieren Unternehmen, wenn sie die iterative Testautomatisierung bereits frühzeitig im Projekt einführen, die Tests einfach halten und sich auf wesentliche, beispielsweise stark risikobehaftete Bereiche fokussieren. Die Testspezifikation sollte klare Richtlinien vorgeben, was zu testen ist und was nicht. Der Zeitgewinn durch dieses Vorgehen kann im Sinne eines möglichst hohen Automatisierungsgrades direkt in weitere Regressionstests investiert werden.

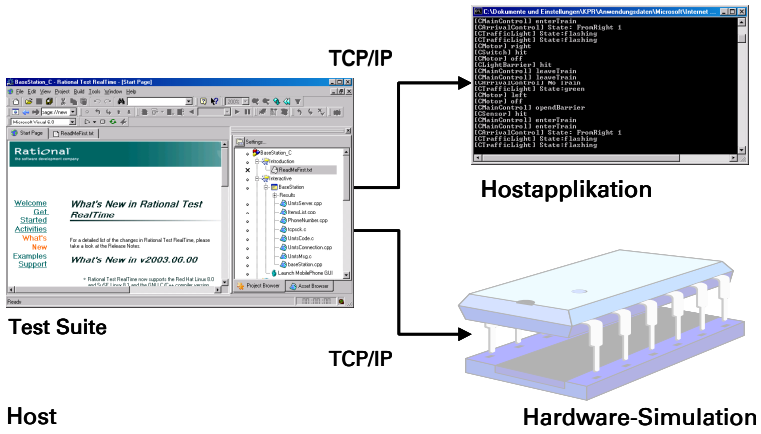
## Automaten für Unit- und Integrationstest




*Suite für einen automatischen Test*

Eine automatische Suite beinhaltet alle Komponenten einer herkömmlichen, manuellen Bench. Durch Parsen werden die Schnittstellen des Prüflings ermittelt – normalerweise die Funktionsprototypen im System. Daraus werden Treiber und Stubs abgeleitet. Durch Parsen generiert die Lösung außerdem den Code für die Testscripten, die durch die Testfallermittlung einfach unterstützt werden.

## Zielsystem Host

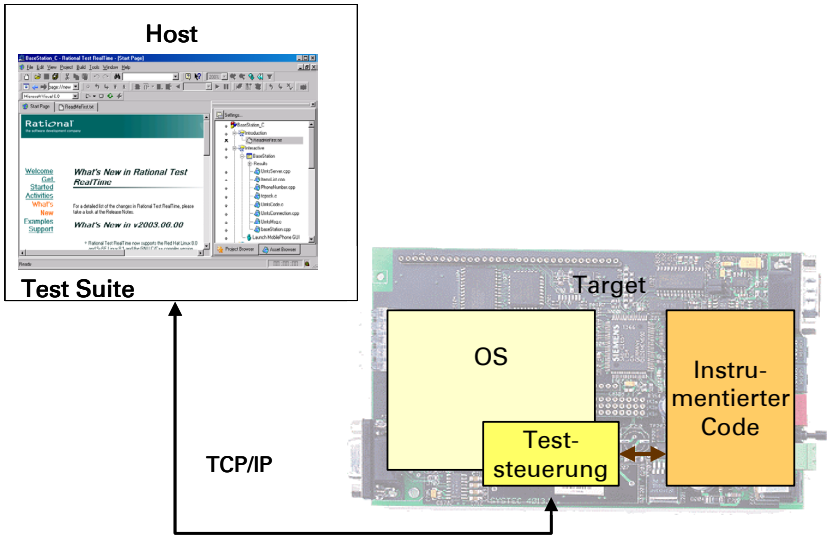



*Der Code kann direkt für den Host oder für eine PC-Simulation generiert werden.*

Ist der Entwicklungs-Host das Zielsystem, generiert der Automat für Unit- und Integrationstest per Compiler den entsprechenden Code und installiert ihn meist auch automatisch. Die kostengünstige Testumgebung steht schnell bereit und bietet offene Schnittstellen, mit denen sich flexible Testkonzepte implementieren lassen. Allerdings müssen für hardwareabhängige Teile eigene Stubs erstellt werden, und das Timing-Modell entspricht nicht der Realität. Ein weiterer Minuspunkt: Die Integration von Hard- und Software lässt sich nicht testen.

Dafür eignet sich eine Hardware-Simulation, die sogar die Mechanik testen kann. Der Code wird hier für das Zielsystem kompiliert und auf dem PC gegen einen simulierten Prozessor ausgeführt. Das Timing-Modell ist in diesem Fall exakt. Das Echtzeitverhalten lässt sich protokollieren, wobei es auch gedehnt werden kann. Beim Keil-Simulator für den C51 ist die Ausführung auf dem Target beispielsweise langsamer als auf dem Host, und die Verhältnisse drehen sich mit höherer Wertigkeit eines Prozesses um. Dann erhöhen sich die Ausführungszeiten, beispielsweise bei einem ARM-Prozessor, um den Faktor 5 bis 10.

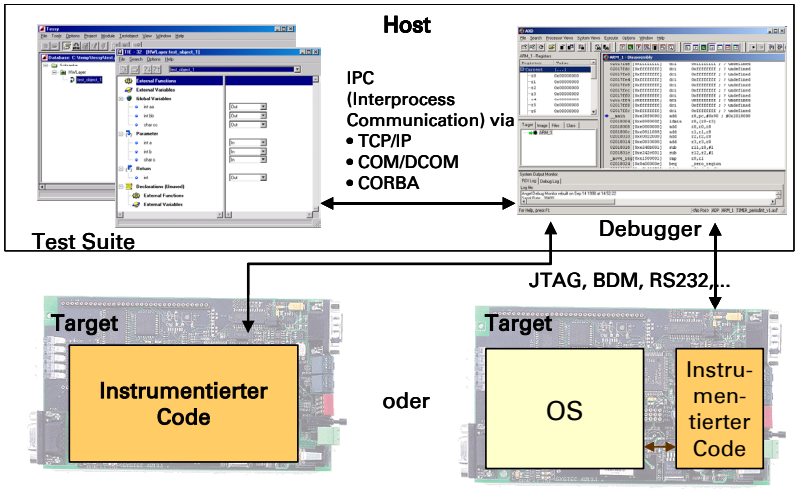
## Zielsystem Target mit Betriebssystem




*Bei der Einbeziehung des Betriebssystems muss eine Teststeuerung integriert werden.*

Bei dieser Variante wird auf dem Target geprüft, das per TCP/IP an die Testsuite angebunden ist. Dies erfordert auf dem Prüfsystem kleine Komponenten für die Teststeuerung. Der Test prüft über die Verbindung bis hinunter zur seriellen Schnittstelle, wobei sich an dieser Stelle das Verhalten des später instrumentierten Codes verändert.

## Zielsystem Target ohne Betriebssystem



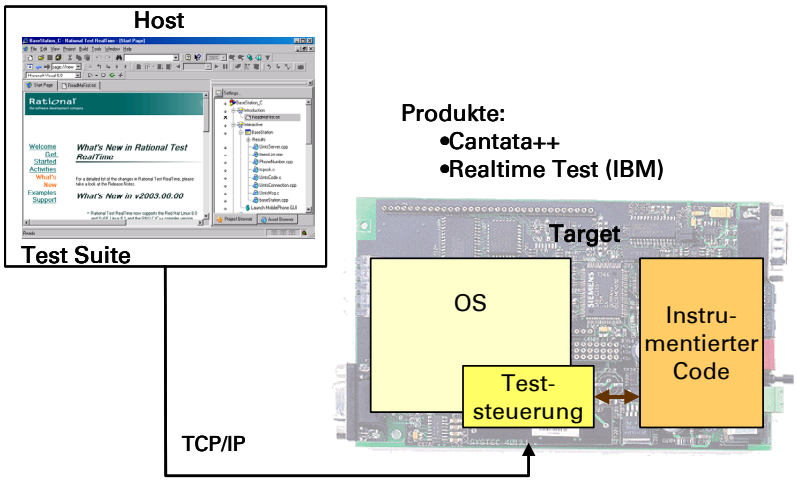

*Die Testsuite erfordert in jedem Fall einen Debugger und Compiler.*

Der Prüfling erhält hier den Code über einen Debugger, beispielsweise über eine JTAG-, BDM- oder RS232-Schnittstelle, womit sich das Target direkt ausführen lässt. Die generierten Testtreiber starten den Code, führen ihn aus, übergeben die Testparameter und initiieren die Testauswertung. Die eigentliche Testsuite ist normalerweise per Interprozesskommunikation an den Debugger angebunden (TCP/IP, COM/DCOM, CORBA).

Beim Target mit oder ohne Betriebssystem profitieren Tester von den offenen Schnittstellen, die flexible Testkonzepte ermöglichen. Außerdem kann die Hardware-Integration einbezogen werden, und auch verteilte Tests sind möglich.

Dabei können verschiedene Anwendungen auf unterschiedlichen Targets gegeneinander laufen. Ein weiterer Vorteil ist die Unabhängigkeit vom Betriebssystem, das man bei Bedarf einbetten kann oder nicht. Das Timing-Modell wird kaum beeinflusst, wobei das Zeitverhalten von der Art des Debuggers abhängt. Die Hardware wird unter realen Bedingungen betrieben, beispielsweise in Bezug auf Memory oder Performance. Allerdings besitzt diese Variante auch Nachteile. So setzt sie einen von der Testsuite unterstützten Compiler und Debugger voraus, was den Herstellern einiges Kopfzerbrechen bereitet, die diese Teile integrieren müssen.

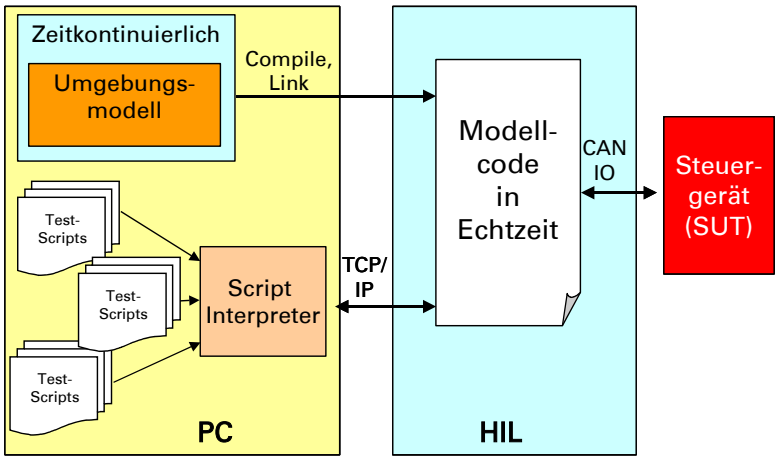
### Testen mit Betriebssystem




*Werkzeuge wie Cantata++ von IPL oder Realtime Test von IBM instrumentieren das Betriebssystem.*

Die Tools in diesem Bereich bieten neben einer guten Testauswertung per Protokoll auch eine ausgezeichnete Messbarkeit der Code-Coverage. Auch die Testverwaltung lässt keine Wünsche offen, was bei der inkrementellen Entwicklung der Testobjekte von großer Bedeutung ist.

### Testen mit Hardware-in-the-Loop (HIL)




*HIL-Teststände lassen sich für Umgebungsmodelle und Prototyping gleichermaßen einsetzen.*

Hardware-in-the-Loop kommt normalerweise erst ab dem Integrationstest infrage. Wer diese Methode bereits für den Funktionstest nutzen möchte, muss bereits auf der Hardware Signale produzieren und aufnehmen können. Auf einem HIL-Simulator mit konfigurierbaren Ein- und Ausgängen läuft ein zeitkontinuierliches Umgebungsmodell, das dem zu testenden Steuergerät die reale Umgebung vorspielt. Der Entwicklungshost ist dabei z.B. per TCP/IP angeschlossen. Der Script-Interpreter läuft nicht unter Echtzeitbedingungen, weil die Scripts nicht auf dem HIL-Simulator, sondern auf dem PC interpretiert werden. Beim klassischen HIL greifen also die Testskripten auf den Code des Umgebungsmodells via Variablen-Mapping über die Ein- und Ausgänge zu.

Ein Nebenaspekt: HIL-Teststände lassen sich auch für Prototyping verwenden, wobei hier der HIL-Simulator als Steuergerät fungiert, das später implementiert werden soll. Mit entsprechenden Prototyping-Werkzeugen lassen sich so schnell Steuergeräte bauen und prüfen. Hersteller in diesem Bereich sind unter anderem dSPACE, National Instruments, ETAS und MicroNova, für Umgebungsmodelle bieten unter anderem Mathworks und National Instruments ausgereifte Tools.

Eine High-End-Lösung für den automatisierten HIL-Test offeriert MicroConsult ST. Im Gegensatz zu herkömmlichen Testsystemen arbeitet es mit einer UML-basierten Architektur, mit der sich die Übersichtlichkeit und Wiederverwendbarkeit von Tests spürbar verbessert. Außerdem bietet MicroConsult ST eine Test-Execution-Engine, womit sich die komplette Testsequenz auf dem HIL laden und in Echtzeit ausführen lässt.

Neue Wege in der Testautomatisierung

# 9. MicroConsult ST: HIL-Systemtest mit UML und LabVIEW

*Wo liegen die ungehobenen Schätze in Projekten, wenn es um das Thema Systemtest geht? Gespräche mit Managern, Testern und Entwicklern ergeben ein eindeutiges Bild:*
*1) in der Kommunikation zwischen Systemingenieuren, Entwicklern und Testern*
*2) in der konsequenten Nutzung von Automatisierung und Standardisierung.*

*Die wachsende Komplexität der Systeme in Verbindung mit hohem Wettbewerbsdruck erhöht die Motivation, diese Schätze nun endlich auch zu heben. Die hier vorgestellte Lösung MicroConsult ST (ST: Systemtest) setzt genau an diesen Punkten an, um das wichtigste Ziel zu erfüllen: hohe Testabdeckung für wenig Geld in kurzer Zeit.*

Die Lösung entstand unter der Federführung von MicroConsult in Zusammenarbeit mit den Firmen ExpertControl, I-Logix, iSYSTEM und National Instruments.

## Prinzip

Beim Systementwurf mit der UML entstehen so genannte Sequenzdiagramme. Diese beschreiben die zeitliche Abfolge von Operationen, die zwischen System und Umgebung ablaufen. Sie stellen damit auch das Verhalten dar, das durch einen Systemtest bestätigt werden muss. Es liegt also nahe, die von den Systemingenieuren erstellten Sequenzdiagramme als Testszenarien für die Gutfall-Tests zu verwenden und weitere notwendige Testszenarien, wie Schlechtfall-Tests, in der gleichen Darstellung zu ergänzen. Der Vorteil: Tester, Entwickler und Systemingenieur benutzen die gleiche Notation. Das schützt vor Missverständnissen und spart Zeit.

Mit dem Test Management Center TMC kann der Tester diese Testsequenzen parametrieren, verwalten und über den Test Executor TXE auf einem Hardware-in-the-Loop Teststand (kurz HIL) automatisch ablaufen lassen. Der HIL bildet dabei das Verhalten der realen Umgebung in Echtzeit nach. Da der Prüfling Steuersignale an diese Hardware sendet und durch Rücksendung der Messsignale die Signalschleife (In the Loop) schließt, spricht man von Hardware-in-the-Loop. Der HIL kann Aufwand und Risiko erheblich senken, wenn die reale Testumgebung nicht verfügbar, zu teuer oder zu gefährlich ist. Die Modellierung der Testumgebung erfolgt dank LabVIEW mit Hilfe gängiger Darstellungsformen aus der Regelungs- und Messtechnik. Über den Test Executor kann auch ein Debugger für den Prüfling gesteuert werden. Das Test Management Center TMC verwaltet alle Testergebnisse, die HIL und Debugger liefern, und erzeugt die Testdokumentation.

## Lösungskomponenten

Mit Hilfe des UML Case Tools Rhapsody von I-Logix werden die Testsequenzen modelliert und verwaltet. Der HIL basiert auf PXI-Baugruppen. Testtreiber und Umgebungsmodell werden mit Hilfe der grafischen Programmiersprache LabVIEW von National Instruments programmiert. Die automatische Generierung der Modell- bzw. Reglerparameter erfolgt mit Hilfe von ecICP von ExpertControl. Die integrierte Debuggerlösung basiert auf Komponenten von iSYSTEM. MicroConsult ST verbindet alle diese Lösungsbausteine zu einer Testautomatisierungslösung. Im Folgenden wird die Lösung im Einzelnen beschrieben.

## UML verbindet Requirements und Testszenarien

Die Sammlung und Strukturierung der Systemanforderungen (Requirements) ist die Grundlage für die Spezifikation eines Systems. Meist handelt es sich dabei um umfangreiche textuelle Beschreibungen. Doch solche Dokumente haben ihre Tücken:

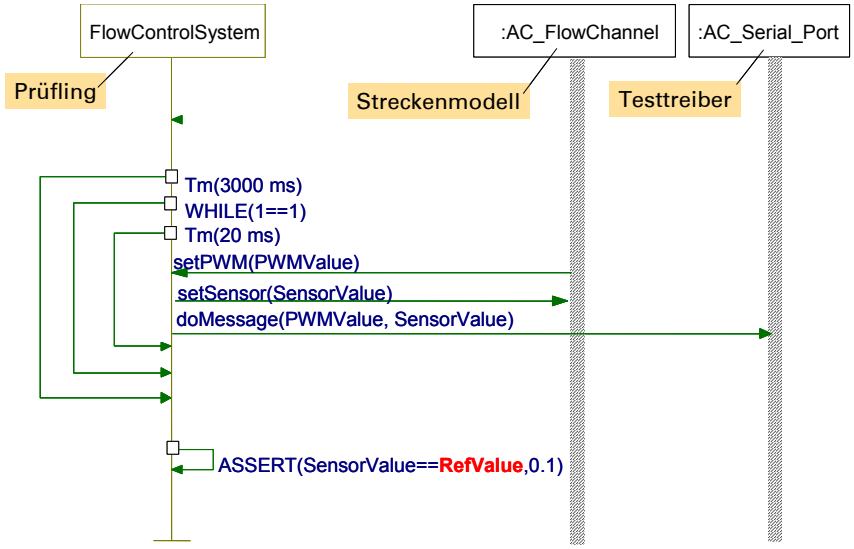
- Inkonsistenz der Informationen über das gesamte Dokument
- Ungenau spezifizierte Abläufe
- Nicht testbare Anforderungen

Schon hier hilft die UML dabei, Defizite zu vermeiden. UML-Diagramme, vor allem Sequenzdiagramme, zeigen anschaulich das Zusammenspiel von System und Systemumgebung und bieten Unterstützung beim Spezifizieren. Andere Diagrammformen aus dem UML-Werkzeugkasten, wie z.B. Use Case Diagramme, sind ebenfalls wertvolle Hilfen beim Systementwurf. Das Case Tool dient nicht nur der Darstellung und Konstruktion von UML-Diagrammen, sondern auch dem Content Management für die Spezifikation. Leistungsfähige Report-Tools, wie der Reporter PLUS von Rhapsody, generieren aus dem UML-Modell und den verlinkten Informationen eine vollständige Dokumentation.

## Testsequenzen

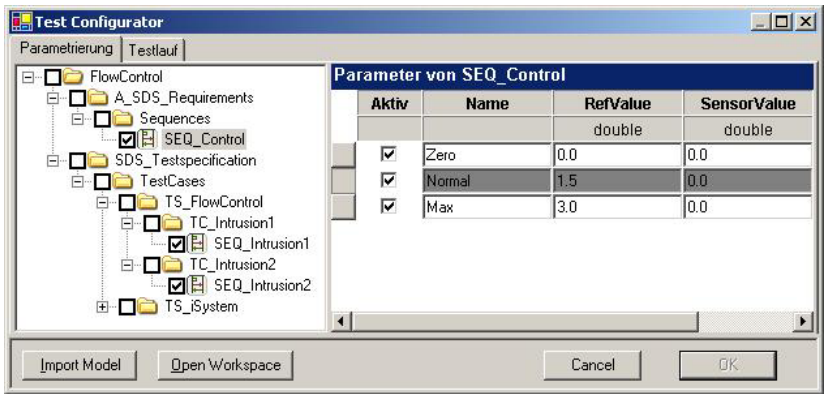
Um die Erstellung von Testsequenzen optimal zu unterstützen, erweiterte MicroConsult die Sequenzdiagramme um folgende Kontrollstrukturen, die jedem Programmierer geläufig sind:

- For- und While-Schleifen für sich wiederholende Sequenzen
- If-Bedingungen für die bedingte Verzweigung von Sequenzen
- Ausdrücke zur Zeitüberwachung (Watchdog) und Zeitsteuerung (zur Herstellung exakter Timings)
- Prüfbedingungen, sogenannte Asserts, zum Vergleich von Soll- und Istwerten




*Testsequenz im UML-Sequenzdiagramm*

Das UML-Sequenzdiagramm zeigt das Zusammenspiel zwischen dem Prüfling (FlowControlSystem) und den Akteuren (ACtors) seiner Umgebung (AC_FlowChannel, AC_Serial_Port). Die Pfeile zwischen den Akteuren und dem Prüfling stehen für Operationen, die in ihrer zeitlichen Abfolge von oben nach unten dargestellt sind. Außerdem sind Zeitbedingungen, Wiederholungen von Teilsequenzen und ein Soll-Istwertvergleich mit ASSERT zu sehen. In den Operationen lassen sich Parameter (z.B. RefValue) angeben.




*Directory mit Testsequenzen und Eingabemaske für die Parametrierung*

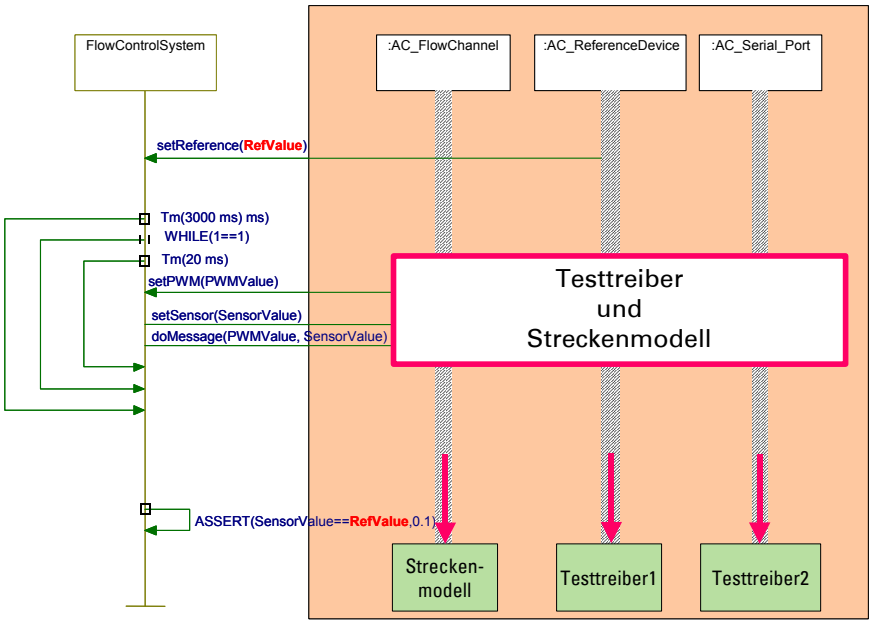
Wird einem Parameter innerhalb der Sequenz kein Wert zugewiesen, so wird er als Variable angesehen. Diese Variablen lassen sich bei der Testkonfiguration auf gewünschte Werte setzen. Damit können Testsequenzen mit verschiedenen Parametersätzen ausgeführt werden.

In Abbildung 2 ist die Ablagestruktur der Testsequenzen innerhalb des Paketbaumes des UML-Modells zu sehen. Wird eine Testsequenz angewählt, öffnet sich eine Tabelle, in der die Parameter für die gewünschten Testfälle bestimmt werden können.

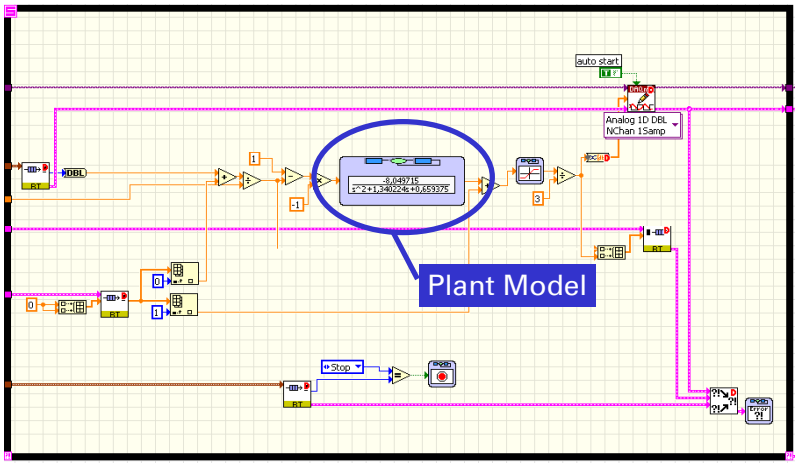
### Testtreiber und Streckenmodell

Aus den Sequenzen ergeben sich die Schnittstellen zu den Testtreibern und dem Streckenmodell (Abbildung unten). Der Tester ist in der Lage, Testtreiber und Streckenmodell unter LabVIEW grafisch zu programmieren und mit dem Test Executor zu einem Umgebungsmodell zu verknüpfen.

Die Testtreiber können frei gestaltet werden. So lassen sich unter anderem Messkarten von National Instruments einbinden sowie beliebige Windows-Komponenten aufrufen (DLLs, .NET-Komponenten, COM/OLE-Module, C-Programme). Die Möglichkeiten sind aufgrund der Vielfalt verfügbarer Messkarten und Windows-Komponenten praktisch unbegrenzt.



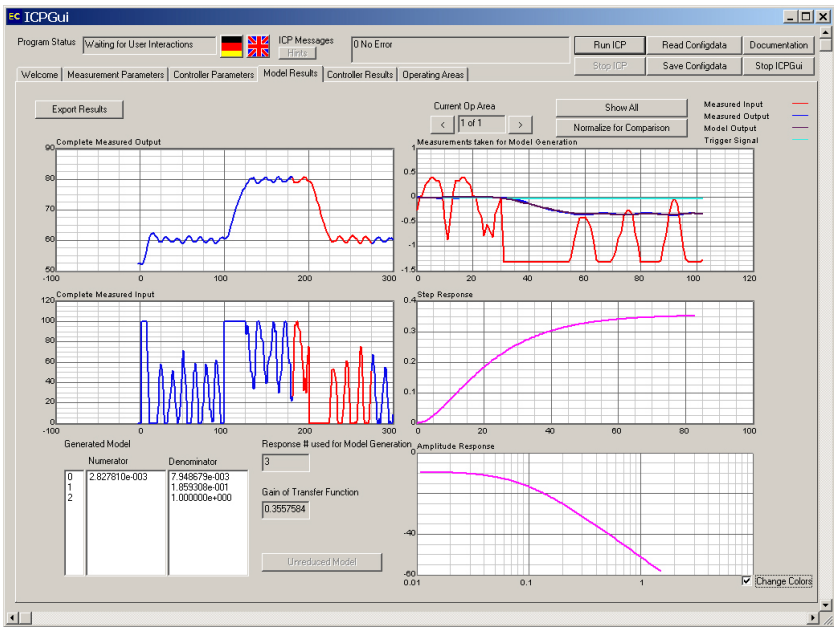

*Streckenmodell und Treiber im Sequenzdiagramm*



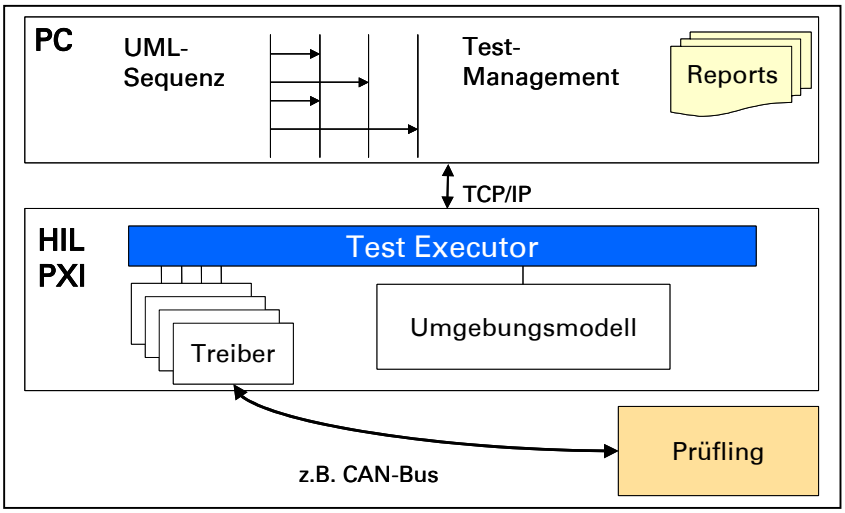

*Simulationsknoten im LabVIEW Modell*

Das Streckenmodell wird in LabVIEW über den Simulationsknoten abgebildet (Abbildung oben). Der Simulationsknoten erlaubt das Einbinden eines Streckenmodells, beispielsweise mittels der Übertragungsfunktion der Regelstrecke. Die Nachbildung des Umgebungsverhaltens durch ein mathematisches Modell in Echtzeit erfordert eine exakte Taktung des Systems. Auch hierfür bietet MicroConsult ST die passende Architekturvorlage, um Testtreiber, Streckenmodell und Test Executor mit exaktem Timing prioritätsgesteuert zu verbinden.

Wer die aufwändige mathematische Bestimmung von Streckenparametern vermeiden will, kann auch diesen Schritt automatisieren. Mit ecICP, einem Produkt der Firma ExpertControl, lassen sich Streckenmodelle aus Messungen oder empirisch ermittelten Daten generieren. In der folgenden Abbildung ist die grafische Oberfläche unter anderem mit den gemessenen Signalen, den daraus errechneten Streckenparametern und dem Sprungverhalten des Modells zu sehen.




*eclCP errechnet aus Messsignalen die Übertragungsfunktion der Strecke*




*Test Executor im Zusammenspiel mit Testtreibern und Streckenmodell*

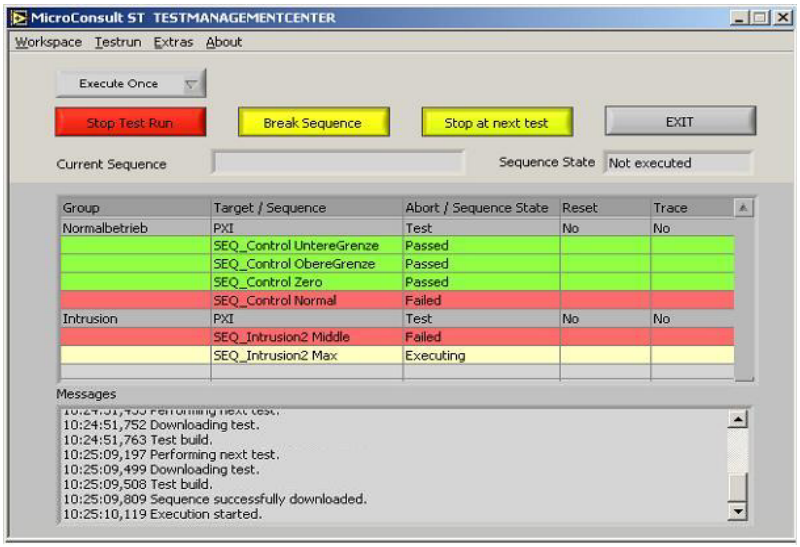
## Ausführung der Testsequenzen

Die parametrierten UML-Sequenzen werden in ausführbare Befehlssequenzen umgewandelt. Diese werden in den Test Executor geladen und gemeinsam mit Testtreibern und dem Streckenmodell in Echtzeit ausgeführt. Die Ausführung wird mittels vollständigem Trace protokolliert. Der Test Executor steht auf verschiedenen Testtargets (PC/Windows, PXI, FPGA-Messkarte) zur Verfügung. Er wird mittels TCP/IP über einen PC gesteuert. Je nach Wahl des Test-Targets wird eine Ausführung in Echtzeit mit einer Schrittweite von bis zu 25 ns erreicht.

## Test Management und Dokumentation

Nachdem die Testsequenzen konstruiert und parametriert sowie die Testtreiber und Streckenmodelle implementiert sind, kann der Test starten. Im Test Management Center werden die einzelnen Testsequenzen in Gruppen organisiert.

Jede Gruppe beinhaltet die Information über den Ausführungsort des Tests (PC, PXI oder FPGA-Messkarte). Alle Testsequenzen zusammen ergeben einen Testlauf. Bei der Ausführung des Testlaufs werden die durchgeführten Testsequenzen abhängig von Status und Ergebnis farblich markiert.




*Test Manager mit Bedienelementen und Statusanzeige für Testsequenzen*

Die Ergebnisse des Testlaufs werden in Verzeichnissen abgelegt, die nach Uhrzeit und Datum des Testlaufs benannt sind. Nach dem Testlauf stehen dem Tester Testprotokolle (beispielsweise nach IEEE 829), Problemreports für jeden gefunden Fehler und das vollständige Trace-Protokoll für die ausgeführten Sequenzen als Debug-Hilfe zur Verfügung. Da das TMC im LabVIEW Source-Code vorliegt, können auch leicht eigene Protokolle erzeugt werden.

## Diagnose

Die Kommunikation zwischen Tester und Entwickler gestaltet sich oft schwierig. Tester denken in Testfällen und Testergebnissen. Entwickler hingegen interessieren sich für die Codezeile, die den Fehler verursacht hat. iSYSTEM bietet mit iTRACE PRO einen Emulator, der über LabVIEW vollständig gesteuert werden kann. Diese Erweiterung zu MicroConsult ST stellt zusätzlich zu den Testprotokollen vollständige Traces bis auf Assembler-Ebene zur Verfügung. Der Entwickler kann dann in winIDEA, dem Debugger von iSYSTEM, den Fehler analysieren und beheben. Dazu bietet MicroConsult ST bereits vorbereitete Testtreiber. Diese erlauben eine Vielzahl an Funktionen: Download, Reset, Start, Stopp, Schreiben/Lesen/Protokollieren von Variablen/Registern, Start/Stopp von Assembler Trace und Protokollen.

## Zusammenfassung

MicroConsult ST ist eine äußerst flexible, in kleinen Schritten skalierbare und leistungsfähige Testautomatisierungslösung. Durch die konsequente Nutzung von Standards bietet sie Unabhängigkeit und ein vorteilhaftes Preis-Leistungs-Verhältnis. Damit lohnt sich der Einstieg in die Testautomatisierung bereits bei Testaufgaben, die bisher aus Kostengründen noch manuell ausgeführt wurden. Die Nutzung der grafischen Programmiersprachen UML und LabVIEW erleichtert die Anwendung und fördert gleichzeitig die Kommunikation zwischen Systemingenieuren, Entwicklern und Testern. Der hohe Automatisierungsgrad entlastet wichtige Ressourcen für Aufgaben, die höheren Nutzen haben als manuelle Tests und Dokumentation.

Info-Pool

## Buch-Tipps

**Design-for-test for digital ICS and Embedded Core** von Alfred L. Crouch.
Prentice Hall 1999, 350 Seiten, ISBN 0130848271

Dies ist ein praxisnaher DFT-Führer eines Industrie-Insiders, der Teststrategien für heutige Geschäftsanforderungen wie Qualität, Verlässlichkeit und Kostenkontrolle vorstellt. Ein wesentlicher Fokus liegt dabei auf der Automatic Test Pattern Generation (ATPG).

**Development Guidelines for Vehicle Based Software**, ISBN 0-95241-560-7, April 1994

**Embedded Systems** – qualitätsorientierte Entwicklung, Qualitätssicherung bei Embedded Software von Klaus Bender. Springer 2005, 385 Seiten, ISBN 3-54022-995-7

Bender zeigt, wie sich Fehler im Entwicklungszyklus frühzeitig erkennen und beseitigen bzw. wie sie sich weitgehend vermeiden lassen. Das Buch stellt Methoden zur Qualitätssicherung vor und integriert sie in ein qualitätsorientiertes Vorgehensmodell.

**IEEE Software Engineering Standards**
IEEE 1997, 1.990 Seiten, ISBN 1-55937-898-0

**MISRA-C: 2004**, Guidelines for the use of the C language in critical systems,
ISBN 0-95241-562-3**,** Oktober 2004

**Peer Reviews in Software**, A Practical Guide von Karl E. Wiegers. Addison Wesley 2001, 256 Seiten, ISBN 0.20173-485-0

Mehr noch als Gilb und Graham geht Wiegers in seinem Werk auf typische Review-Problemfälle ein. Der Leser erfährt beispielsweise, wie er besonders umfangreiche Dokumente per Stichprobenverfahren prüfen kann.

**Software Inspection** von Tom Gilb und Dorothy Graham.
Addison Wesley 1993, 471 Seiten, ISBN 0-20163-181-4

Das erste und umfangreichste Lehrbuch ist noch immer das Standardwerk für alle Reviewer und solche, die es werden wollen.

**Software-Test**, Verifikation und Validation von Georg Erwin Thaller.
Heise 2002, 383 Seiten, ISBN 3882291982

Der Autor beschreibt alle gängigen Testarten, aber auch die Verifikation und Validation, die in vielen Bereichen weit über den Test hinausreichen. In der aktuellen Auflage hat Thaller die Themen Testautomatisierung, Management und Organisation sowie Test und das CMM wesentlich erweitert.

Info-Pool

**Software Test Automation**, Effective Use of Test Execution Tools von Dorothy Graham und Mark Fewster.
Addison Wesley 2000, 600 Seiten, ISBN 0201331403

Wer sich vom Kauf eines automatisierten Test-Tools eine Verbesserung der unternehmensinternen Testprozesse erwartet, liegt falsch – so die zentrale These von Graham und Fewster. Ihr Buch richtet sich an jeden Testmanager oder Ingenieur und zeigt praxisnah, wie sich automatisiertes Testen gewinnbringend einführen lässt.

**Software-Qualität** von Georg Erwin Thaller.
VDE-Verlag 2000, 247 Seiten, ISBN 3-80072-494-4

Der Autor beschreibt die gängigen Methoden wie Verifikation und Validierung, Test, Debugging, Regressionstest, Inspektionen, Walk-Throughs und Prozessmodelle, aber auch fortschrittliche Techniken für das Management wie Tailoring und Clean-Room.

**Testing Embedded Software** von Bart Broekmann und Edwin Notenboom.
Addison Wesley 2003, 320 Seiten, ISBN 0-32115-986-1

Die Autoren adaptieren Projekterfahrungen aus dem strukturierten Testen kommerzieller Software auf die Anforderungen von Embedded Systemen mit Fokus auf die praktische Organisation des Testprozesses.

**Testing Object-Oriented Systems** von Robert V. Binder.
Addison Wesley 2000, 1.191 Seiten, ISBN 0-201-80938-9.

Das Buch ist ein wichtiges Standardwerk für das automatisierte, modellbasierte Testen von objektorientierten Anwendungen mit vielen praktischen Beispielen.

Info-Pool

## Web-Tipps

www.artisansw.com
Website von Artisan Software Tools.

www.isystem.com
Website von iSYSTEM.

www.misra.org.uk
Webseite des MISRA-Konsortiums, dort auch Bestellung oder Download der MISRA-Guidelines.

www.ni.com
Website von National Instruments.

www.opengroup.org/testing/testsuites/embedded.html
Embedded Testsuites der Open Group; die Vereinigung ist ein herstellerneutrales Konsortium für nahtlose Informationsflüsse.

www.pls-mc.com
Website von pls Programmierbare Logik und Systeme.

www.polyspace.de
Deutsche Website von PolySpace Technologies.

www.smart-gmbh.de
Website von Smart Electronic Development.

www.softwaretesting.de/article/archive/7
Webseite der Pix Software GmbH, dort auch interessante Artikel zu den Themen Qualitätssicherung und Testing Tools.

http://cccc.sourceforge.net
Download CCCC von McCabe, mit dem sich die Komplexität von Software messen lässt.

www.willert.de
Website von Willert Software Tools.

MicroConsult hat keinerlei Einfluss auf die Inhalte und Funktion der hier genannten Links . Die Verantwortung liegt ausschließlich bei den Anbietern dieser Seiten.

MicroConsult Leistungen

## Training zum Thema – die optimale Ausrüstung für jede Tiefe

Mit den MicroConsult-Trainings schaffen Sie sich die optimale Wissensbasis, um Projektanforderungen gerecht zu werden und an Sie gestellte Erwartungen zu übertreffen. Erfahrene Spezialisten vermitteln Ihnen dabei auch komplexe Inhalte verständlich und praxisbezogen und sichern Ihnen den effektiven Umgang mit der richtigen Ausrüstung.

Aktuelle Termine und komplettes Trainingsprogramm unter www.microconsult.de.

---

### Grundlagen des Testens für Embedded Projekte

**Trainingsziel:**
Professionelles Testen beherrschen; Tests planen, bewerten und implementieren

**Vorkenntnisse:**
Grundkenntnisse einer höheren Programmiersprache, z.B. C/C++

**Inhalt:**
Was Test im Embedded Markt bedeutet; Software-Qualität für Embedded-Systeme; Requirementanalyse; Testspezifikation und Testplanung; statische Prüfung vom Review zum LINT; Integrationsstrategien; dynamische Prüfung; Testauswertung; kosteneffiziente Tests

**Dauer:** 5 Tage

**Preis:** 1.990 Euro zzgl. Ust.

---

### Requirements Engineering und Management für die Entwicklung in der Industrie

**Trainingsziel:**
Den Requirements Prozess verstehen, bewerten, in der Firma einführen und dort leben; Optimieren der bestehenden Prozesse

**Vorkenntnisse:**
Keine, Projekterfahrungen sind von Vorteil

**Inhalt:**
Entwicklungsprozess, Identifikation von Requirements, Dokumentation anhand von Beispielen und mit grafischen Hilfsmittel der UML, Abnahmetests; Management, praktische Übung anhand eines Beispiels für ein typisches Entwicklungsszenario

**Dauer:** 4 Tage

**Preis:** 1.680 Euro zzgl. Ust.

---

MicroConsult Leistungen

## Software-Qualität für Embedded Systeme

**Trainingsziel:**
Die angestrebte Qualität von Software systematisch vorgeben; die Standards zur Qualität von Software kennen; die erreichte Qualität bestätigen

**Vorkenntnisse:**
Projekterfahrung mit Softwaresystemen

**Inhalt:**
Qualität von Entwicklungsprozessen; Qualität der Produkte Rechner und Software; Bestätigung der Produktqualität; praktische Übungen, z.B. Prüfen von Code oder Bewerten von Prüfergebnissen

**Dauer:** 5 Tage

**Preis:** 1.970 Euro zzgl. Ust.

---

## Reviews erfolgreich durchführen

**Trainingsziel:**
Die Software-Qualität der Projekte erhöhen; die Produktivität der Softwareprojekte steigern; als Moderator ein Review planen und leiten sowie die Ergebnisse quantitativ und qualitativ bewerten

**Vorkenntnisse:**
Kenntnisse einer höheren Programmiersprache (idealerweise C), branchenübliche Englisch-Kenntnisse

**Inhalt:**
Einführung in die Review-Technik; Nutzen von Reviews; Videoszenen aus gut und schlecht durchgeführten Reviews; Übung: Durchführung eines realen Reviews; Einführung in das Lotus Inspection Data System (LIDS); Arbeitshilfsmittel im Internet

**Dauer:** 2 Tage

**Preis:** 1.150 Euro zzgl. Ust.

---

MicroConsult Leistungen

### OOA-Analyse mit der UML

**Trainingsziel:**
Analyse und Entwurfsverfahren sowie die Darstellungsform der Unified Modeling Language (UML) kompetent einsetzen

**Vorkenntnisse:**
Programmiererfahrung, z.B. CHILL, C, Fortran oder C++

**Inhalt:**
Grundbegriffe der OOP; Einführung in die objektorientierten Basiskonzepte; Darstellung der objektorientierten Basiskonzepte mit Hilfe der UML-Klassennotation; dynamisches Verhalten objektorientierter Software; Umsetzung in verschiedene Programmiersprachen

**Dauer:** 5 Tage

**Preis:** 1.850 Euro zzgl. Ust.

---

## Coaching – Entscheidende Impulse im rechten Moment

Als fachkundige Experten stehen wir Ihnen in jeder Phase Ihres Projektes zur Seite und betreuen Ihre Entwickler- und Testteams kompetent und effizient. Mit bewährter Methodik und hohem Praxisbezug machen wir bereits im Vorfeld auf Entwicklungs- und Testrisiken aufmerksam.

Somit können Sie Ihr Projekt transparent, vorhersehbar und wirtschaftlich gestalten. Sie reduzieren die Entwicklungszeit auf das Wesentliche und minimieren die Kosten. Die Entwicklung wird in alle Richtungen abgesichert und zielt auf einen qualitativ hochwertigen Projekterfolg.

---

## Engineering – Entlastung auf Knopfdruck

Wenn Projekte die eigenen Kapazitäten überschreiten, nehmen die Experten von MicroConsult die Last von Ihren Schultern. Wir bieten Ihnen die optimale Unterstützung, damit Sie Ihre eigenen Kräfte dort konzentrieren können, wo sie die größte Wirkung erzielen.

Was nicht zu den Kernkompetenzen Ihres Unternehmens zählt, übertragen Sie unseren Entwicklungsspezialisten: von der Definition und Optimierung von Entwicklungsprozessen über die Entwicklung von Hardware, Software und Systemarchitekturen bis zum Testen des fertigen Produktes.

Service

# Partnerverzeichnis

| Fokus | Logo | Firma / Kontakt |
|---|---|---|
| Case-Tools für komplexe technische Systeme auf Basis von UML 2 und SysML |  | ARTiSAN Software Tools GmbH<br>Eupener Str. 135-137<br>50933 Köln<br>www.artisansw.com<br><br>**Kontakt:**<br>Christiane Kapteina<br>Tel.: +49 221 48522-63<br>cristiane.kapteina@artisansw.com |
| Werkzeuge für die Entwicklung und den Test schneller mechatronischer Regelungssysteme. |  | dSPACE GmbH<br>Technologiepark 25<br>33100 Paderborn<br>www.dspace.com<br><br>**Kontakt:**<br>Tel.: +49 5251 1638-0<br>Fax: +49 5251 66529-<br>info@dspace.de |
| Lösungen für Embedded Systems, Entwicklung und Test-Automatisierung |  | iSYSTEM AG<br>Carl-Zeiss-Str. 1<br>85247 Schwabhausen<br>www.isystem.com<br><br>**Kontakt:**<br>Erol Simsek<br>Tel.: +49 1386971-56<br>erol.simsek@isystem.com |

Service

# Partnerverzeichnis

| Fokus | Logo | Firma / Kontakt |
|---|---|---|
| Training, Coaching, Engineering und Projektbegleitung | MICRO CONSULT | MicroConsult GmbH<br>Rosenheimer Str. 143b<br>81671 München<br>www.microconsult.de<br><br>**Kontakt:**<br>Peter Siwon<br>Tel.: +49 89 450617-44<br>p.siwon@microconsult.com |
| Hard- und Software für computerbasierte Mess- und Automatisierungstechnik | NATIONAL INSTRUMENTS | National Instruments Germany GmbH<br>Konrad-Celtis-Str. 79<br>81369 München<br>www.ni.com<br><br>**Kontakt:**<br>Stephan Ahrends<br>Tel.: +49 89 7413130<br>stephan.ahrends@ni.com |
| • Debugger<br>• On-chip Emulator-Support<br>• In-System FLASH-Programmierung | pls Development Tools | pls Programmierbare Logik & Systeme GmbH<br>Technologiepark<br>02991 Lauta<br>www.pls-mc.com<br><br>**Kontakt:**<br>Heiko Rießland<br>Tel.: 035722 384-0<br>info@pls-mc.com |

Service

# Partnerverzeichnis

| Fokus | Logo | Firma / Kontakt |
|---|---|---|
| • Software-Testing<br>• Entwicklung, Vertrieb, Beratung, Training | PolySpace TECHNOLOGIES | PolySpace Technologies GmbH<br>Argelsrieder Feld 22<br>82234 Wessling-Oberpfaffenhofen<br>www.polyspace.de<br><br>Rudolf Frommknecht<br>Tel.: 08153 9072-10<br>rudolf.frommknecht@polyspace.com |
| Testlösungen für Automotive Electronic<br>• Testengineering<br>• Testsysteme<br>• Testkomponenten<br>• Methoden und Tools für den Funktionstest an HW, SW und Systemen. | SMART | SMART Electronic Development GmbH<br>Hölzelweg 2<br>70191 Stuttgart<br>www.smart-gmbh.de<br><br>**Kontakt:**<br>Wolfgang Neu<br>Tel. 0711 25521-23<br>wolfgang.neu@smart-gmbh.de |
| Embedded UML Solution with optimized code generation for small real-time systems | Willert Software Tools<br>Pioneers in Embedded Software Engineering | Willert Software Tools<br>Hannoversche Str. 21<br>31675 Bückeburg<br>www.willert.de<br><br>**Kontakt:**<br>Andreas Willert<br>Tel.: 05722 9678-60<br>info@willert.de |

Impressum

**Herausgeber:**

MicroConsult GmbH
Rosenheimer Str. 143b
81671 München
Tel. +49 89 450617-0
Fax +49 89 450617-17
www.microconsult.de

**Projektleitung:**

Sabine Pagler, MicroConsult

**Autoren:**

Klaus-Peter Rosenthal, MicroConsult
Peter Siwon, MicroConsult

**Redaktion:**

Eva Schulz, actimedia

MicroConsult Microelectronics Consulting & Training GmbH
Rosenheimer Straße 143b • D-81671 München
Tel. 089 450617-71 • Fax 089 450617-17
E-Mail: info@microconsult.de • www.microconsult.de