Smart FAT – Zur Sicherheit Automatisiert

Smart FAT – Automated execution for an increased level of safety

Dr.-Ing. Jürgen Greifeneder, Dipl.-Ing. (BA) Katharina Gohr
ABB Corporate Research, Ladenburg

Kurzfassung

Die Funktionsprüfung des Steuerungscodes eines Leitsystems ist integraler Bestandteil der Werksabnahme einer Automatisierungslösung und wird heute weitestgehend manuell durchgeführt. Dies führt dazu, dass die Testdurchführung nicht nur zeitaufwändig ist, sondern auch unsystematisch durchgeführt wird. Erkannter Änderungsbedarf wird zwar – ebenfalls manuell – dokumentiert und entsprechend ausgeführt, ein erneuter Testdurchlauf stellt in der Regel jedoch keine Rückwirkungsfreiheit auf andere Teile des Systems sicher. Dieser Artikel zeigt daher die Möglichkeit einer automatisierten Testdurchführung. Diskutiert werden unterschiedliche Testarten und Voraussetzungen für deren Anwendung. Letztlich wird das Konzept „Smart FAT" vorgestellt, welches die Definition von wiederholbaren Testreihen ermöglicht und damit die Qualität der Tests sicherstellt.

Abstract

The functional test of control code is an integral part of the factory acceptance test (FAT); today, this is typically executed manually. This manual execution is not only time-consuming, but it is commonly done in an unsystematic fashion. Changes are often documented manually during the test and then later corrected. There is no guarantee that correcting such an error does not introduce additional problems in other parts of the system. This article shows different kinds of tests and requirements for their usage. Finally, the concept of "Smart FAT" is presented. This concept allows the definition of repeatable test series and thus increases the quality of testing.

1 MOTIVATION

Bevor eine Prozessanlage in Betrieb genommen werden kann, gilt es verschiedene Aufgaben zu erledigen. Was das Leitsystem betrifft, muss dieses nicht nur konfiguriert, sondern vor allem auch in internen Tests (engl. Internal Acceptance Test, IAT) und Werksabnahmen (engl. Factory Acceptance Test, FAT) umfassend verifiziert und validiert werden. Da die Projektlaufzeiten tendenziell kürzer werden, wird auch die Zeit zum Testen – wie in [1] beschrieben – geringer. Folglich sollten nicht nur die Konfigurationswerkzeuge für den Steuerungscode oder die Datenübernahme aus vorangegangen Projektphasen effizienter werden, sondern auch die Ausführung der Tests.

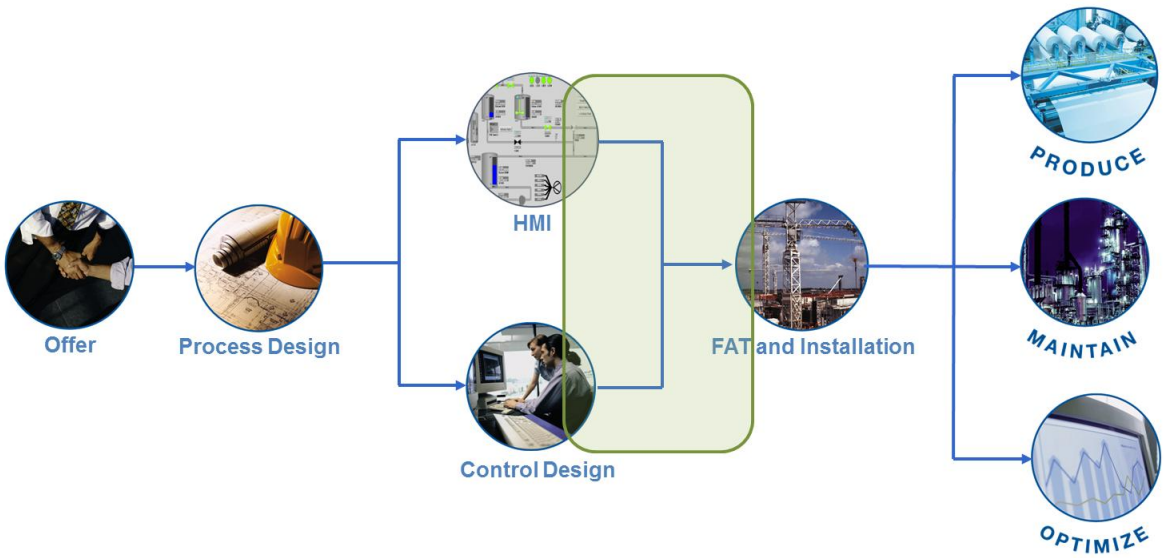

Bild 1: Testphase während des Engineering Prozesses

Die Testphase erstreckt sich, wie in Bild 1 gezeigt, auf den letzten Teil der Implementierung sowie auf FAT und Installation und kann wiederum in drei Phasen unterteilt werden [2]: Vorbereitung, Durchführung und Abbau. In der Vorbereitung muss die Hardware oder entsprechendes Ersatzequipment zum Testlabor transportiert, dort aufgebaut und verdrahtet werden, die Komponenten konfiguriert, sowie die notwendigen Testsequenzen erstellt und eingespielt werden. In der Durchführungsphase werden die einzelnen Tests schrittweise manuell durchgeführt und dokumentiert. In der Abbauphase werden die Testsequenzen gelöscht oder deaktiviert, sowie die Hardware abgebaut, verpackt und zur Baustelle versandt.

Das Konzept „Smart FAT“ setzt auf eine automatisierte Testdurchführung unter zu Hilfenahme emulierter Steuerungskomponenten: Um kostenintensive Hardwaretestumgebungen zu vermeiden, wird im vorliegenden Konzept eine Emulation der Steuerung und der Bussysteme verwendet. Dadurch können bei der Vorbereitung und dem Abbau der Tests Zeit und Kosten gespart werden.

Um auch bei der Durchführung der Tests eine Einsparung zu erzielen, wird hierbei auf eine automatisierte Ausführung größtenteils vordefinierter Testsequenzen und eine automatische Protokollierung der Ergebnisse gesetzt.
Unter der Annahme, dass die Projektlaufzeiten sich auch in Zukunft weiter verkürzen, ist auch eine Zeitersparnis bei der Werksabnahme erforderlich. Zudem lassen sich bei automatisierten Tests ohne signifikanten Mehraufwand im Anschluss an Fehlerkorrekturen Regressionstests durchführen, die bei manueller Testdurchführung weniger umfangreich ausfallen würden. Dadurch lässt sich die Qualität des zu testenden Systems verbessern und Fehler können schneller erkannt werden.

2 TECHNISCHER HINTERGRUND

Idealerweise deckt ein Testtool mehrere Testszenarien, die während des FAT durchgeführt werden, ab. Diese umfassen beispielsweise Tests der Verriegelungslogik, der Grenzwerte, der Ablaufsteuerungen, der Abschaltlogik, des Alarmmanagement, der Kommunikation – zwischen Applikationen und zwischen Controllern – oder Test der Prozessgrafiken. Um ein Konzept zum Test der erwähnten Leitsystemfunktionen zu ermöglichen, werden zum einen die Anforderungen und zum anderem der Stand der Technik in diesem Kapitel näher betrachtet. Beides führt letztlich zum Konzept „Smart FAT“ welche die Anforderungen erfüllen sollte und teilweise auf dem Stand der Technik aufbaut.

2.1 Anforderungen an ein Testtool für Smart FAT

Zur automatischen Durchführung der Tests sollten die Anforderungen, wie in Tabelle 1 dargestellt, von einem Testwerkzeug erfüllt werden:

Tabelle 1: Anforderungen

Anforderung	Motivation
Es sollte die Möglichkeit zur Erstellung und Manipulation einer Testbibliothek bereitgestellt werden	Bibliotheken erleichtern die Bedienung und führen zur Aufwandseinsparung, da Testszenarien vorgegeben und wiederverwendet werden können.
Es sollte die Möglichkeit zur Parametrierung von vorgegebenen Testszenarien vorhanden sein	Eine einfache, projektspezifische Anpassung vorgefertigter Testszenarien erleichtert die Wiederverwendbarkeit und erhöht die Wiederverwendungsrate.
Es sollte die Möglichkeit zur Kombination von Tests zu Testreihen geben	Kombination verringert den Testaufwand, Testreihen können z.B. zeitsparend über Nacht ausgeführt werden, da keine Bedienung mehr nötig ist
Einfache Erstellung von projektspezifischen Tests	Spezielle, projektspezifische Tests sollten ebenso möglich sein, wie vorgefertigte Tests, da davon auszugehen ist, dass viele Projekte sehr spezielle und spezifische Teile beinhalten.
Möglichkeit zum Unterbrechen und späterem Fortfahren der Tests und Testreihen.	Die flexible Anwendung der Tests führt zu einem effizienterem Testablauf

Anforderung	Motivation
Testszenarien sollten sowohl für einzelne Objekte, als auch für komplexere Applikationen und Strukturen anwendbar sein	Die flexible Skalierbarkeit der Testszenarien führt zu einer Zeitersparnis während der Tests.
Unterstützung einer virtuellen Testumgebung (Emulation der Steuerung, Simulation des Prozesses)	Die reale Steuerungshardware und Feldgeräte sind während des FAT meist nur teilweise beim Automatisierungstechniker vorhanden. Der Prozess ist dem Automatisierungstechniker oft nicht direkt zugänglich.
Bei Abbruch einer Testreihe sollten die bis dahin erzielten Ergebnisse trotzdem verwendbar sein	Einmal ausgeführte, rückwirkungsfreie Tests sollten direkt dokumentiert werden, um eine erneute (vielleicht zeitintensive) Ausführung zu verhindern.
Ergebnisse sollten in einem elektronisch auswertbaren Format (z.B. XML), sowie einem menschenlesbaren Format (z.B. HTML) dokumentiert werden	Die automatische Auswertung eines elektronisch lesbaren Formats ist sehr viel einfacher als die eines „prosa“ Formats. Funktionalität wie: filtern, vergleichen, etc. kann einfach realisiert werden. Jedoch sollte das Format ebenso menschenlesbar sein, um die Ergebnisse später an den Kunden zu übergeben. Zusätzlich verlangen Prüfstellen (wie z.B. der TÜV) eine Dokumentation und Archivierung der Ergebnisse.
Aufzeichnung und Wiedergabe einer Testreihe: Schrittweises Ansehen einer kompletten Testsequenz sollte möglich sein	Testingenieur oder Kunde könnten zu einem späteren Zeitpunkt den Test (noch einmal) ansehen wollen. Das Finden eines Fehlers, der z.B. nur durch bestimmte Prozesszustände hervorgerufen wird, wird erheblich vereinfacht.
Regressionstests sollten möglich sein. Zeitbasierte automatische Ausführung der Test sollte möglich sein	Die automatische, zeitlich geplante Ausführung von Regressionstest (bekannt aus der agilen Softwareentwicklung) führt zu einer erheblich höheren Testrate und sichert die Rückwirkungsfreiheit von Korrekturen nachhaltig.
Die automatische Dokumentation sollte Datum, Uhrzeit, Softwareversionen und ausführende Testpersonen beinhalten	Die Testergebnisse sollten nachvollziehbar dokumentiert werden. Da die Tests oft auf einer bestimmten Version der AT-Konfiguration ausgeführt werden, sollte diese in der Auswertung der Tests berücksichtigt werden.
Eine automatische Aufzeichnen von Signalverläufen und mit Zeitstempeln versehenen Alarmen und Warnungen sollte standardmäßig vorgesehen werden	Dies erhöht die Nachvollziehbarkeit der Testergebnisse. Fehler können schneller gefunden und korrigiert werden.
Die Reihenfolge der Tests, sowie der getestete Zustand (Ein- und Ausgangsbelegung von relevanten AT-Teilen), sollten aufgezeichnet werden	Um einen Fehler in der AT nachvollziehen zu können, sollten die relevanten Testparameter aufgezeichnet werden. Oft hängen Fehler mit den zuvor gesetzten Zuständen zusammen, oder ein spezieller Zustand kann nur erreicht werden, wenn die Eingangsparameter in einer bestimmten Reihenfolge gesetzt werden.

2.2 Stand der Technik

Steuerungslogik nach IEC61131-3 [3], sowie Bedienelemente und Grafikbausteine werden im heutigen Ablauf sowohl projektunabhängig, als auch projektbezogen getestet. Die projektunabhängige Testphase betrifft den Test von Bibliothekselementen (Typen oder Klassen, bzw. Typicals oder Templates). Hierbei handelt es sich um vorgefertigte Einheiten, welche neben

Programmcode meist auch zusätzliche Funktionen mit sich bringen, wie zum Beispiel Bedienelemente oder Dokumente. Da diese Elemente nach Erstellung umfassend – bei sicherheitsrelevanten Bibliothekselementen sogar alle möglichen Beschaltungen – getestet werden, können diese im späteren System verwendet werden, ohne einen erneuten Test der Teilfunktionalität. Dies gilt jedoch nicht für die Parametrierung und die Verknüpfung dieser Elemente untereinander oder mit anderen Teilen des Leitsystems. Daraus ergibt sich ein anderer Fokus beim Testen der Bibliothekselemente und des projektspezifischen Steuerungscodes.

Während der Test der Bibliothekselemente auf Vollständigkeit und möglichst effiziente Wiederholung zielt, kommt es bei den projektspezifischen Tests auf eine automatische Testgenerierung und die Verwendbarkeit von vordefinierten Tests an. Dies resultiert daraus, dass der projektspezifische Code zu umfassend ist, um in angemessener Zeit manuell alle nötigen Tests erstellen zu können. Während zum Beispiel für ein sicherheitsrelevantes „UND"-Bibliothekselement (siehe auch IEC61508-3 [4]) mit 8 Eingängen bereits 256, bzw. 2^8 Testfälle entwickelt werden, würde sich ein Testingenieur bei einer vergleichbaren Situation mit projektspezifischen Code auf ausgewählte Testfälle fokussieren – zumindest, wenn er diese Testfälle manuell konfigurieren oder manuell durchführen müsste. Tabelle 2 nennt kommerzielle Werkzeuge welche für die Bibliotheks- und teilweise auch für die Applikationsentwicklung eingesetzt werden.

Tabelle 2: Auswahl an Softwarewerkzeugen für Steuerungscode-Tests.

Testwerkzeug	Beschreibung
CODESYS Test Manager	Durchführung automatisierter Applikationstests im IEC 61131-3-Programmiersystem; Strukturierte Ablage und Verwaltung von Testskripten und Testreports, siehe [5]
SafeFBTest (KW-Software)	Ermöglicht die manuelle Erstellung von Zustandsdiagrammen, aus denen Testsequenzen automatisch erstellt werden und via OPC ausgeführt werden können. Der Funktionsumfang dieses Tools fokussiert sich auf einen einzelnen Funktionsbaustein, siehe [6].
LibraryTestBuilder (ABB)	Für das System 800xA von ABB entwickeltes Testtool, welches den zu testenden Code in die Steuerung lädt und dort testet. Die Tests werden entweder im Werkzeug manuell erstellt oder als Datei geladen. Die Tests können wiederholt werden. Das Tool ist besonders für Bibliothekselemente geeignet, kann aber auch für Applikationen verwendet werden.

Während z.B. der LibraryTestBuilder auf Black-Box-Tests setzt, also das Innere des zu testenden Elements nicht betrachtet wird, ermöglichen White-Box-Tests eine erhöhte Testabdeckung. Hierbei muss die innere Logik eines zu testenden Elements bekannt sein. Dadurch wird unter anderem eine statische Analyse des Codes möglich.

Statische Code Analysen sind heute in der Regel fester Bestandteil von Codeeditoren, jedoch sind die Analysemöglichkeiten für gewöhnlich eingeschränkt. So wird bereits im Editor (z.B. der Control Builder M von ABB) im Bereich von SIL-Code auf nicht erlaubte Konstrukte oder Funktionsblöcke geprüft. Jedoch ließe sich der Prüfungsumfang noch deutlich erweitern, z.B. mit Prüfungen auf nicht erreichbaren Code, wie es teils bei Hochsprachen üblich ist (z.B. „Resharper“ von Jetbrains für .NET Applikationen).
Nachteilig bei der statischen Codeanalyse ist, dass – für eine umfassende Analyse – auch Bibliothekelemente mit einbezogen werden sollten. Dies ist jedoch meist nicht möglich, da diese verschlüsselt (als „Black-Box“) ausgeliefert werden.
Der Einsatz derartiger Codeanalysen macht insbesondere für komplexe Programmiersprachen Sinn, da mit dem Komplexitätsgrad auch die Anzahl der Fehlerquellen ansteigt. Zum Beispiel lässt die die Hochsprache C++ dem Benutzer zwar viele Freiheiten, aber ist besonders im Umgang mit Zeigern sehr fehleranfällig.
Der Ansatz den Funktionsumfang einzuschränken, wird bereits in sicherheitskritischen Bereichen für Steuerungscode nach IEC61131-3 verwendet und soll dazu beitragen, die Fehlerquellen zu reduzieren. Das Einschränken des Funktionsumfanges könnte mit Hilfe von anderen Konfigurationssprachen noch weiter fortgeführt werden. Unter der Berücksichtigung, dass etwa 80% des Steuerungscodes eines Systems nach [7] aus verhältnismäßig einfacher Logik besteht, wäre hier ein anderer Ansatz möglich.
Ein Schritt in diese Richtung stellt die strukturierte Generierung von Codeteilen aufgrund von graphischen Eingabemasken dar, wie es im sicherheitskritischen Bereich mit Hilfe von Cause & Effect Diagrammen schon eingesetzt wird [8]. Aktuell wird ein ähnliches Format auch für die Prozesslogik erprobt und ein entsprechendes Werkzeug dafür umgesetzt [9]. Hierbei werden die Anforderungen an die Prozesslogik in einem matrixähnlichen Format erfasst, welches dem bekannten Cause & Effect-Diagramm ähnelt. Die Verwendung eines bekannten Formates soll die allgemeine Verbreitung begünstigen.

Die in diesem Kapitel erläuterten Methoden stellen eine Möglichkeit zum Test von Bibliothekselementen dar. Für projektbezogenen Tests können diese Konzepte teilweise übernommen werden, müssen jedoch um weitere Funktionalität erweitert werden. Hinzu kommt, dass für den IAT und FAT eine komplexe Testumgebung benötigt wird. Die Steuerungslogik erstreckt sich normalerweise über die Grenzen eines einzelnen Geräts hinaus, wodurch die zu testende Logik auf einer komplexen Testplattform ausgeführt werden sollte, um auch die Abhängigkeiten der verschiedenen Applikation und Geräte adäquat zu verifizieren.

3 DAS KONZEPT „SMART FAT“

Das Konzept „Smart FAT“ umfasst die folgenden Fragestellungen, welche sich durch die Anforderungen und den Stand der Technik ergeben:

- Welche Arten von Tests gibt es und welche davon lassen sich automatisieren?
- Wie lassen sich Testsequenzen mit geringem Aufwand generieren?
  - Hierzu gehören Aspekte wie die Testgenerierung aus kundenspezifischen Engineering-Daten, ebenso wie Testreiheneditoren und Technologien zur Aufzeichnung von Testreihen
- Wie muss eine Test-Ausführungseinheit aufgebaut sein, damit sie komplexe Testabläufe selbständig ausführen kann?
- Wie lassen sich die erzielten Testergebnisse in nutzerfreundlicher und maschinenlesbarer Art und Weise dokumentieren?

3.1 Architektur

Aus den Anforderungen geht hervor, dass es für ein solches Testwerkzeug unerlässlich ist, Schnittstellen zur Leitsystem-Emulation, Prozesssimulation, wie auch zum Leitsystem bereitzustellen. Hierbei wird das zentrale Element als „Test-Engine“ bezeichnet, welches über die genannten Schnittstellen die anderen Elemente steuert, siehe Bild 2. Zudem ist die Test-Engine in der Lage, vordefinierte Testreihen zu Laden und Auszuführen, sowie neue Testreihen zur späteren Wiederverwendung zu speichern. Zur Konfiguration neuer, sowie der Anpassung, Auswahl und Ausführung bereits bestehender Testreihen, muss dem Benutzer eine Bedienoberfläche zur Verfügung gestellt werden. Letztendlich gibt es noch eine Schnittstelle zur Generierung der Dokumentation.

Mit Hilfe der Prozesssimulation werden, wie in Bild 3 gezeigt, zum einen die Messwerte der Sensoren erzeugt, sowie jene Rückmeldesignale, die von den diversen mit dem Prozessablauf interagierenden Geräten rückgemeldet werden, je nachdem, wie sich der Prozesszustand in Abhängigkeit der vom Leitsystem kommenden Steuersignale (und des Eigenverhaltens des Prozesses) verändert. Simulierte Geräte können dabei z.B. Pumpen und Sensoren sein. Eine Prozesssimulation ist insbesondere zur Überprüfung der Anfahrsequenzen notwendig. Für weiterführende Informationen zum Thema Prozesssimulation sei auf [10] sowie [11] verwiesen, in denen ein entsprechendes Tool vorgestellt wird.

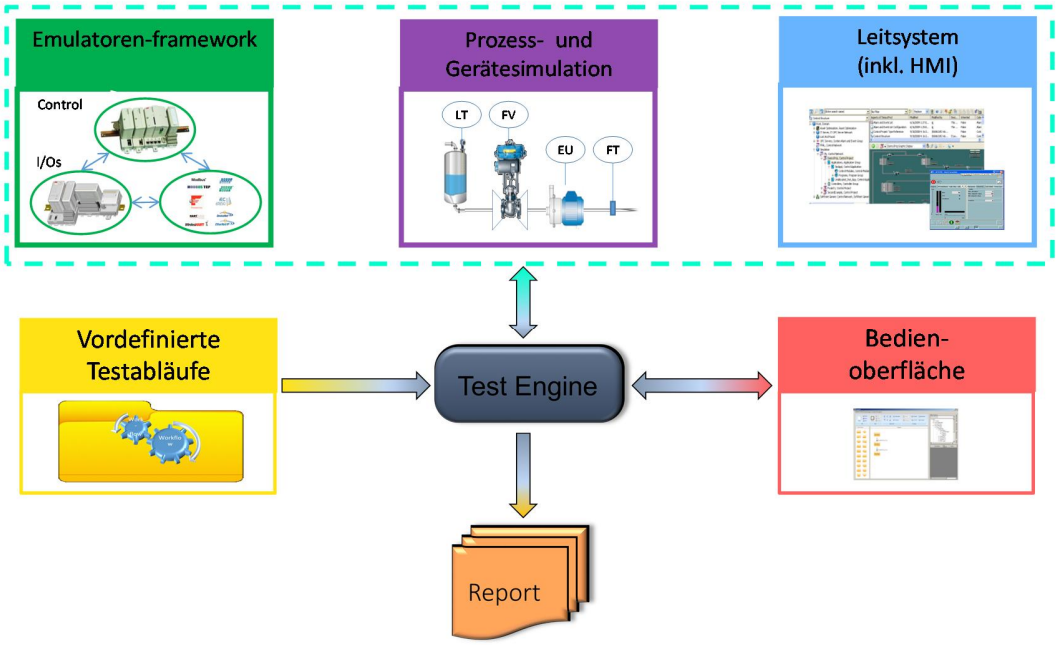

Bild 2: Aspekte des Smart FAT Konzeptes

Eine Emulation bietet sich in diesem Anwendungsfall primär für die Steuerung, die Feldbuskoppler, sowie die Feldbusse an. An dieser Stelle wird auf eine detaillierte Beschreibung von Emulatoren verzichtet und stattdessen auf [12] verwiesen.

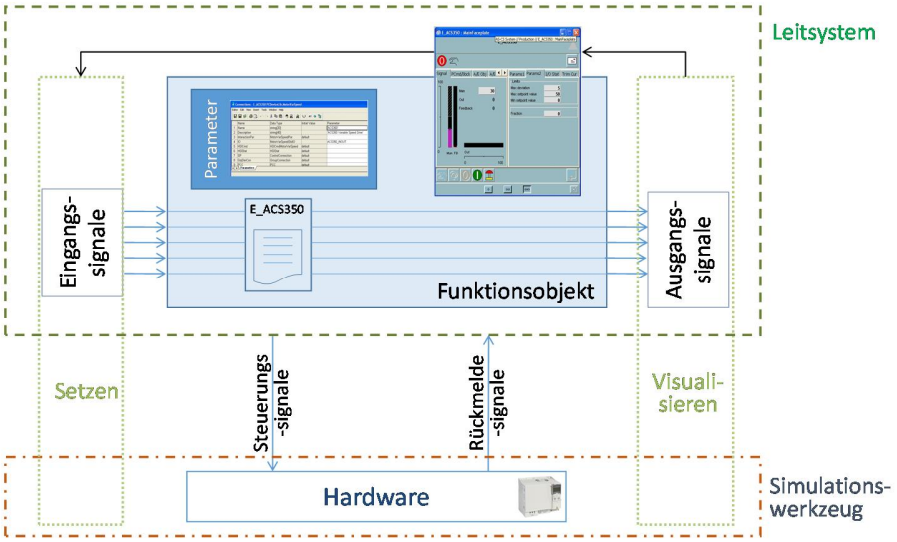

Bild 3: Funktionsüberblick simulationsbasiertes Testen

3.2 Erstellung und Verwaltung von Testsequenzen

Die Erstellung von Testsequenzen kann auf verschiedene Weise erfolgen. Die Praxistauglichkeit der verschiedenen Vorgehensweisen hängt von der jeweiligen Testsequenz ab, wie die im Rahmen der zugrunde liegenden Forschungstätigkeiten durchgeführten Erprobungen gezeigt haben.

Die nächstliegende Möglichkeit, Testsequenzen zu erstellen, besteht a) in der manuellen Erfassung der einzelnen Arbeitsschritte. Des Weiteren kann b) aus zugrunde liegenden Planungsdaten eine automatische Ableitung von Testsequenzen erfolgen. Daneben bietet sich c) die Erstellung von „Metatests“ an, Programmcode also, der ein bestehendes System auf bestimmte Attribute hin untersucht und bei Auftreten die notwendigen Testsequenzen erzeugen kann. Als Beispiel sei die Verriegelungslogik genannt, bei der sich der eigentliche Test erst aus Randbedingungen wie der Anlagentopologie ergibt. Außerdem können Testsequenzen durch d) eine Testaufzeichnung abgebildet werden. Dabei führt der Testingenieur einen Test manuell durch, wobei jede seiner Aktionen von der Aufzeichnung erfasst wird. Je nach Anwendungsfall kann ein so aufgezeichneter Test im Anschluss A) in derselben Art und Weise durchgeführt, oder B) auf andere Objekte oder Strukturen erweitert werden. Schritt B) erfolgt dabei entweder durch manuelle, werkzeugunterstützte Nacharbeit oder durch wiederholtes Aufnehmen ähnlicher Testreihen.

Innerhalb des Forschungsprojekts wurden ca. 60 verschiedene Testarten identifiziert, wovon ein Auszug in Tabelle 3 stichwortartig dargestellt ist.

Tabelle 3: Auswahl möglicher Testarten für ein Testwerkzeug

Testart	Fehlerquelle	Erkennung
Querkommunikationstest zwischen Steuerungen	Kommunikationsvariable muss beidseitig verwendet werden	Regelbasierte Prüfung, statische Regel: „Wird eine Variable einseitig gelesen oder geschrieben, dann ist es vermutlich ein Fehler“
Rückmeldetest der Aktuatoren zum PLS	Verbindungen zwischen den Rückmeldesignalen und der Steuerung	Regelbasierte Prüfung, statische Regel: „Sind weniger als ein Rückmeldesignal für jeden Aktuator zur Steuerung verbunden, dann ist es vermutlich ein Fehler“
Verwendung der Ein- / Ausgabevariablen	Nicht verwendete E/A Variablen	Regelbasierte Prüfung, statisch – Wird jede einem Eingang zugeordnete Variable mindestens 1 mal im Code gelesen und wird jede einem Ausgang zugeordnete Variable genau 1 mal geschrieben
Inkonsistente Namensgebung	Tippfehler während der Objektinstanziierung, Doppelte Tag-Namen	Prüfung gegen eine Namenskonvention und Überprüfung, ob jeder Tag-Name genau einmal vergeben wurde, statisch
Faceplate: Prüfen, ob ein Name und eine Beschreibung existiert	Unvollständiges Engineering	Überprüfung, ob jedes Faceplate einen Namen und eine Beschreibung hat und zusätzlich der Name genau einmal vergeben wurde, statisch
Faceplate: Prüfen, ob Name oder Beschreibung zu lang sind	Textlängen für verschiedene Sprachen variieren	Überprüfung, ob der Name und die Beschriftung jedes Faceplates in das entsprechende Textfeld passt, statisch

Als Beispiel für eine Standardtestsequenz sei der in Bild 4 gezeigte Ablauf zur Überprüfung der Textfeldgröße angeführt. Dies ist notwendig, da beim Design der graphischen Menüelemente der darin anzuzeigende Text oftmals noch nicht bekannt ist, so dass nicht sichergestellt werden kann, dass der anzuzeigende Text auch tatsächlich in das dafür vorgesehene Feld passt. Dies gilt insbesondere dann, wenn die Menüfelder zu Bibliothekselementen gehören, die mit Textblöcken unterschiedlicher Sprachen gefüllt werden können.

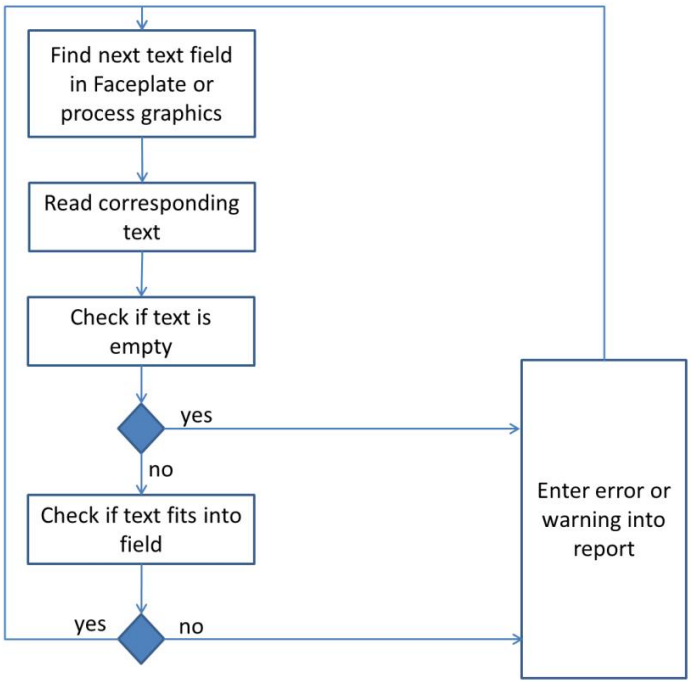

Bild 4: Test Workflow zur Überprüfung von Textfeldern im HMI

Als ein weiteres Beispiel sei hier die Überprüfung der Variablen angeführt. Bei diesem Test soll zum einen überprüft werden, dass sowohl alle E/A-Variablen mit physikalischen E/A-Schnittstellen verknüpft sind, als auch keine anderen Variablen diese Verknüpfung aufweisen. Zum anderen soll überprüft werden, dass auf alle Ausgangsvariablen geschrieben wird, aber auf keine der Eingangsvariablen. Die zugehörige Sequenz ist in Bild 5 dargestellt.

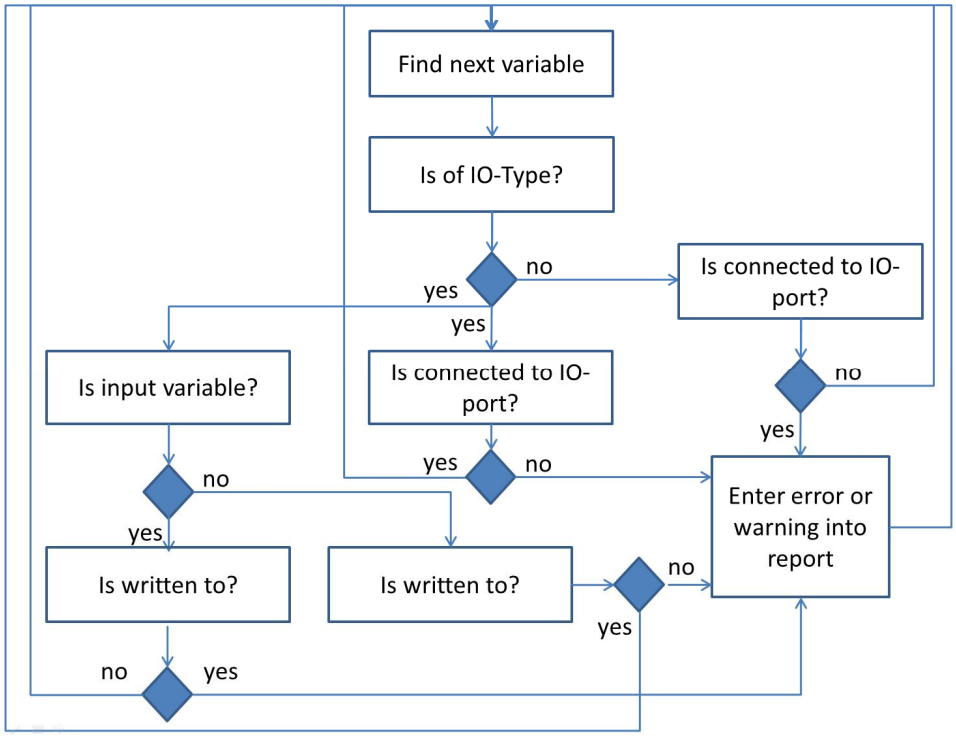

Bild 5: Test Workflow zur Überprüfung von Textfeldern im HMI

Um ein flexibles Werkzeug zu konzipieren, wurden drei Möglichkeiten vorgesehen, Testsequenzen einzubinden:

- Allgemeingültige Sequenzen, die zur Grundausstattung des Testwerkzeugs gehören (hierzu gehören die in Tabelle 3 gezeigten Testarten)
- Branchenspezifische Testsequenzen, die als eigenständige Bibliothek geladen werden (diese können durch die am Anfang dieses Abschnitts beschriebene Vorgehensweise aufgezeichnet werden)
- Projektspezifische Testsequenzen, die wie andere vom Testingenieur selbst generiete Testsequenzen direkt einspielbar sind. Diese können ebenfalls durch die am Anfang dieses Abschnitts beschriebene Vorgehensweise aufgezeichnet werden.

Die eingespielten sowie auch die selbst erzeugten Testsequenzen können zudem an das jeweilige Projekt oder den vorliegenden Testdurchlauf angepasst werden. Die hierfür notwendigen Anpassungen können sowohl die Objektauswahl als auch eine Reihe von spezifischen Parametern betreffen. Um die auf die eine oder andere Weise erzeugten Testsequenzen flexibel einsetzen und auch austauschen zu können, sollten diese in einem standardisierten und offenen Format (beispielsweise basierend auf einem XML-Schema) gespeichert sein.

3.3 Prüfberichterstellung

Eine automatische Testausführung erfordert, dass auch die daraus resultierenden Ergebnisse in einem automatisch generierten Prüfbericht dokumentiert werden. Zum einen ist schon ein

Testdurchlauf mit nur einer Testsequenz meist zu komplex, als dass das Ergebnis durch eine einfache optische Rückmeldung zusammengefasst werden könnte. Zum anderen ist der Testingenieur daran interessiert, seine Ergebnisse für eine spätere Fehlerbehebung oder zu Dokumentationszwecken zu speichern.

Als Format für diesen Report kommt html in Frage. Denkbar wären auch pfd oder Office-Formate, jedoch ist html aufgrund der weiten Verbreitung, weitgehenden Standardisierung, dem minimalen Installationsaufwand für einen Reader und der guten Möglichkeiten der Verlinkung besonders gut geeignet. Bilder können außerdem problemlos eingebettet werden.

Der naheliegende Ansatz einer Prüfbericht-Generierung besteht darin, den Testablaufplan um die erzielten Ergebnisse zu ergänzen. In Bild 6 wurde dies beispielhaft für den Testablauf aus Bild 5, welcher die Größe von Textfeldern inspiziert, dargestellt. Diese Darstellungsform eignet sich insbesondere für die interne Dokumentation, da Fehlerquellen schnell identifiziert werden können. Auch ist diese Darstellung gut geeignet, wenn zwei Durchläufe derselben Testsequenz zu unterschiedlichen Ergebnissen geführt haben. Im Allgemeinen jedoch, vor allem wenn die Tests erfolgreich waren, genügt es vollkommen, die Testsequenz einmalig zu beschreiben (z.B. als Ablaufplan) und auf den folgenden Seiten lediglich alle Objekte und Teilsysteme zu listen, auf welche dieser Test angewandt worden ist; jeweils unter Angabe von erfolgreich oder nicht.

Test: „Textlänge überprüfen“ Datum: 2014-04-01
Objekt: „System11/Area99/11-PT-99403“ Uhrzeit: 14:03

NUMMER	AKTION		ERGEBNIS
1	Überprüfe alle Faceplates auf Textfelder		Name, Titel, Beschreibung
2	Berechne die maximale Textlänge für jedes Textfeld		Name: 25, Titel: 33, Beschreibung: 75
3a	Lese den anzuzeigenden Text für jedes Textfeld in Englisch		Name: 11-PT-99403 Titel: Pressure Transmitter for Tank2 Beschreibung: Pressure Transmitter for Tank2 in System 11
3b, …	Wie 3a, aber für Deutsch, Norwegisch, Schwedisch, Französisch,…		…
4	Berechne die tatsächliche Textlänge und vergleiche sie mit der maximalen Textlänge für jedes der Textfelder		
		Englisch:	Name: 11 < 25 ✓ Title: 32 < 33 ✓ Beschreibung: 45 < 75 ✓
		Deutsch:	Name: 11 < 25 ✓ Title: 28 < 33 ✓ Beschreibung: 82 < 75 x
		Norwegisch, Schwedisch, Französisch,, …	übersprungen

Bild 6: Mögliche Reportseite für den Testworkflow aus Bild 5

Die eigentliche Herausforderung ist jedoch nicht die Generierung eines Prüfberichts per se, sondern die Akzeptanz durch den Kunden, sowie durch die Prüforganisationen. Hier spielt nicht nur der Aufbau des Berichts – er sollte möglichst übersichtlich und intuitiv gestaltet sein – eine Rolle, sondern vor allem dessen Inhalt. Er muss alle nötigen Details enthalten, um den Test zu einem späteren Zeitpunkt exakt nachvollziehen zu können. Hier bieten sich Filtermöglichkeiten an, die sowohl einen schnellen Überblick über die alle ausgeführten, oder auch nur die fehlgeschlagenen, Tests zulassen, aber bei Bedarf auch alle Details einblenden.

Mit der Frage der Akzeptanz eng verknüpft ist die Frage, welche Tests überhaupt automatisch durchgeführt werden dürfen und auch können. In den meisten Projekten verhält es sich so, dass der Kunde oder die Prüforganisation die Tests vorgibt – meistens in Prosa-Papierform. Dies erschwert die automatische Generierung von projektspezifischen Testsequenzen. Vorteilhaft wäre die Übergabe von maschinenlesbaren Testsequenzen (idealerweise schon im richtigen Format). Auch wäre es denkbar, dass der Auftraggeber aus einer Liste mit Standardtests jene auswählt, die er für wichtig hält und somit nur noch die fehlenden Tests spezifizieren muss.

4 ZUSAMMENFASSUNG

Smart FAT stellt einen Ansatz vor, der es ermöglicht, die Hardware des Prozessleitsystems vollständig zu emulieren und den Prozess zu simulieren, wodurch der Test des Prozessleitsystems automatisch ausgeführt und dokumentiert werden kann. Die wichtigsten Teilaspekte sind dabei die Simulation des Prozesses, die Emulation der Hardware, die Erstellung der Testsequenzen, sowie die Dokumentation des Testverlaufs.

Entscheidend für einen umfangreichen Einsatz des Smart FAT sind vor allem die Kundendaten, da viele Tests nicht allgemein gehalten werden können und notwendigerweise auf ihnen basieren. Wenn diese Daten in einem für einen Rechner gut auswertbaren Format vorliegen, können daraus automatisch Tests erstellt werden. Ein standardisiertes Format wäre hier wünschenswert. Auch ist eine gute Softwareunterstützung zur Generalisierung von Tests wichtig.

Für die Praxistauglichkeit von Smart FAT ist es letztendlich nicht nur entscheidend, alle Beteiligten vom sicherheitstechnischen Nutzen dieses Ansatzes zu überzeugen, sondern auch eine Zeitersparnis beim Testen zu erzielen.

5 LITERATUR

[1] Rodies, H.-J.: Planungswerkzeuge aus Sicht des Anlagenbaus. atp – edition 44(1), S. 40-44, 2002

[2] Mario Hoernicke, Jürgen Greifeneder: Next Generation Factory Acceptance Test. ABB Corporate Research Center Germany - Annual Report 2011. 04/2012, 83-89.

[3] IEC61131-3: Programmable Controllers – Part 3: Programming Languages. Edition 2.0, 2003

[4] IEC61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. 2010

[5] CODESYS Professional Developer Edition, http://de.codesys.com/produkte/ codesys-engineering/professional-developer-edition.html (abgerufen am 31.3.2014)

[6] SAFEFBTEST , https://www.kw-software.com/de/iec-61508-safety/test-tool/safefbtest (abgerufen am 31.3.2014)

[7] PACs for Industrial Control, the Future of Control, 18.10.2012, http://www.ni.com/white-paper/3755/en/ (abgerufen am 31.3.2014)

[8] Safeguard 400 Series http://www.abb.co.uk/product/seitp334/ 3e67d74f1f51e2c9c12571e2003305d4.aspx?tabKey=4 (abgerufen am 31.3.2014)

[9] Katharina Gohr, Ralf Jeske: Technik-Kommunikation leicht gemacht: Cause and Effect-Diagramm als Lösungsansatz, atp edition 56(1), S. 20-21, 2014

[10] Greifeneder, Jürgen; Weber, Peter; Barth, Mike; Fay, Alexander: Generierung von Simulationsmodellen auf Basis von PLS-Engineering-Systemen. atp edition 54(4), S. 34-41, 2012

[11] Dirk-Uwe Astrath: VSP zur Unterstützung der Inbetriebnahme von verfahrenstechnischen Anlagen, http://www.et.tu-dresden.de/ifa/fileadmin/ user_upload/www_files/Prozessleittechnik/forschung/VIBN_Workshop/Dirk-Uwe_Astrath.pdf (abgerufen am 31.3.2014)

[12] Mario Hoernicke, Jürgen Greifeneder, Mike Barth: Effizientes Testen heterogener Leitsystemkonfigurationen – Integration gewerkeübergreifender Hardware-Emulatoren. atp Edition 11/2012, 54(11):46-54.