Statische Analyse von Programmen - Erfahrungen aus einem Process Improvement Experiment bei ABB Netzleittechnik

Horst Lichter
ABB Corporate Research
Postfach 101332
D-69003 Heidelberg
lichter@decrc.abb.de

Peter Piwecki
ABB Corporate Research
Postfach 101332
D-69003 Heidelberg
piwecki@decrc.abb.de

Gerhard Riedinger
ABB Netzleittechnik GmbH
Postfach 1140
D-68526 Ladenburg

Zusammenfassung

In diesem Beitrag präsentieren wir die Ergebnisse eines Projektes der ABB Netzleittechnik im Rahmen ihrer "Software Process Improvement" Aktivitäten.[1] Dieses Projekt sollte Erfahrungen bei der Anwendung der statischen Programmanalyse sammeln und auswerten. Wir stellen dazu zuerst vor, welche Art von Software dort entwickelt wird und beschreiben anschließend die Ziele des Projektes. Nachdem wir einen kurzen Überblick über die statische Programmanalyse gegeben haben, beschreiben wir im Hauptteil dieses Beitrags die Erfahrungen mit dem Einsatz der statischen Programmanalyse und präsentieren die daraus von uns gezogenen Schlußfolgerungen. Wir tun dies, indem wir neun Kernaussagen formulieren und diskutieren.

Abstract

In this paper we present the findings and results obtained in a project of ABB Netzleittechnik, Germany. The project was executed in the context of ABB Netzleittechnik's software process improvement activities. It was launched in order to collect experiences in the usage of systematic static programme analysis. We first give a brief overview on software development at ABB Netzleittechnik. Then we list the objectives of the project. After having given a short overview on the technique of static programme analysis, we present in the main part of the paper the experiences and findings. We do this by formulating and discussing nine major statements.

Key Words

Software Prozessverbesserung, Statische Analyse, Fallstudie, Logiscope, Validierung.

1. Einleitung

Die ABB Netzleittechnik entwickelt und vermarktet ein Netzwerk-Control-System. Dieses integriert verschiedenste Funktionen wie "Energy Management", Funktionen zur Überwachung und zur Steuerung von Hoch- und Mittelspannungsnetzen, oder Funktionen zur Eingabe von Netztopologien und Netzdaten. Das dazu entwickelte Basissystem S.P.I.D.E.R. wird in vier Entwicklungszentren in Deutschland, der Schweiz, Schweden und den USA gepflegt und weiterentwickelt. Damit die Entwicklungsarbeiten systematisch durchgeführt werden können, hat die Netzleittechnik ein Prozeßmodell entwickelt, das alle wesentlichen Aspekte der Softwareentwicklung abdeckt. Das Prozeßmodell der Netzleittechnik basiert auf dem V-Modell (Bröhl, 1995) und definiert die folgenden wesentlichen Phasen:

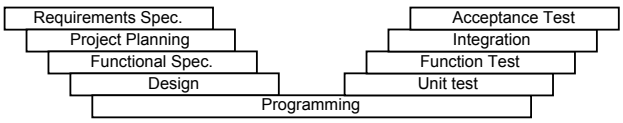

Bild 1: Software-Prozeßmodell

Dieses Prozeßmodell wird bei allen Entwicklungsprojekten verwendet und kann nach den Erfordernissen der einzelnen Projekte angepaßt werden.

1 Diese Arbeit wurde gefördert durch das ESSI Projekt 21612 SMUIT

Die ABB Netzleittechnik hat ihren Entwicklungsprozeß bewerten lassen, um eine gute Ausgangsbasis für Verbesserungen zu haben. Dazu wurde ein Assessment nach dem Capability Maturity Model (CMM) durchgeführt, das am Software Engineering Institute der Carnegie Mellon University entwickelt wurde (Paulk et al., 1991). Dieses Assessment zeigte u.a., daß der existierende Entwicklungsprozeß Schwächen im Bereich Testen und Validierung hat.

Basierend auf den Ergebnissen des Assessments wurde ein Projekt gestartet, das zu Ziel hat, im Bereich Software-Validierung eine Verbesserung herbeizuführen. Zu diesem Zweck hat sich die ABB Netzleittechnik am Förderprogramm der Europäischen Gemeinschaft ESSI (European System and Software Initiative) beteiligt. Im Rahmen des ESSI Förderprogramms können sogenannte Process Improvement Experiments (PIE) durchgeführt werden. Ein solches PIE zum Thema Systematisches Testen wurde von der EU genehmigt und im Zeitrahmen von Mai 96 bis April 97 bei der ABB Netzleittechnik durchgeführt.

In diesem Beitrag werden die Ergebnisse dieses Projekts vorgestellt. Dabei werden vor allem die Erfahrungen berichtet, die bei der Anwendung der statischen Analyse von S.P.I.D.E.R. Modulen erzielt wurden.

2. Ziele des Projekts

Mit dem erwähnten Projekt sollten die folgenden Ziele erreicht werden:

- Definition eines verbesserten Prozesses für das Testen.
- Schulung der Mitarbeiter im Bereich Validierung und Testen.
- Auswahl eines Werkzeugs zur Unterstützung der Test- und Validierungsaktivitäten.
- Erprobung der Prozesse und des gewählten Werkzeugs auf ihre Praxistauglichkeit.

S.P.I.D.E.R. ist ein System, dessen erste Teile etwa vor 20 Jahren entwickelt wurden. Seitdem wurden viele neue Komponenten dazu entwickelt und existierende Komponenten auf neue Hardwareplattformen, Betriebssysteme und Programmiersprachen portiert. Aus diesen Gründen kann man S.P.I.D.E.R. sicherlich als ein “legacy system” bezeichnen, das sehr heterogen und sehr umfangreich ist, und aus Komponenten besteht, die in verschiedenen Programmiersprachen realisiert sind (FORTRAN, C, C++).

Bei der Auswahl eines testunterstützenden Werkzeugs mußte dieses berücksichtigt werden. Vor allem viele moderne Werkzeuge zum Testen der Benutzerschnittstelle konnten in dieser heterogenen Umgebung nicht eingesetzt werden, da sie mit den unterschiedlichen Komponenten und Technologien, die in S.P.I.D.E.R. vorgefunden werden, nicht zurecht kommen.

Aus diesem Grund wurden Werkzeuge untersucht, deren Schwerpunkt die statische Analyse von Programmcode ist, die aber u.U. auch andere Funktionalitäten anbieten, beispielsweise das Messen der erreichten Testüberdeckung oder das automatische Starten von Testsequenzen.

Aufgrund der Randbedingungen des S.P.I.D.E.R. Systems wurde das Werkzeug Logiscope der Firma Verilog ausgewählt. Dieses Werkzeug hat u.a. die Fähigkeit, Systeme, die in verschiedenen Programmiersprachen entwickelt sind, zu analysieren, und stellt eine Menge weiterer testunterstützender Funktionen zur Verfügung. Im Projekt wurden Logiscope-Lizenzen für die Sprachen FORTRAN, C und C++ verwendet.

3. Statische Analyse

Zur statischen Analyse von Programmen werden immer Werkzeuge verwendet. Diese analysieren der Code nach verschiedensten Aspekten und präsentieren die Ergebnisse in graphischer und textueller Form für den Benutzer. Statische Analysewerkzeuge verfügen häufig über eines oder mehrere Qualitätsmodelle, die sie verwenden, um den analysierten Code bzgl. verschiedener Qualitätskriterien zu bewerten. Ein solches Qualitätsmodell definiert der ISO Standard 9126 (ISO, 1991) u.a. für das Qualitätskriterium Wartbarkeit. Dieses wird basierend auf dem Merkmalen Lesbarkeit, Änderbarkeit und Testbarkeit definiert.

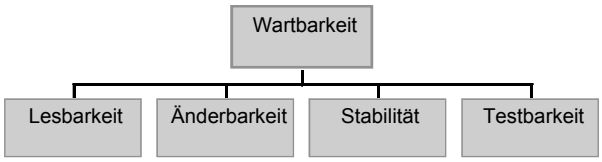

Bild 2: Teil des ISO 9126 Qualitätsmodells

Diese grundlegenden Merkmale werden aus Metriken berechnet, die das Analysewerkzeug liefern kann (z.B. die zyklomatische Komplexität oder die Länge von Funktionen).
Als Ergebnis der statischen Analyse können alle analysierten Komponenten gemäß dem Qualitätsmodell klassifiziert werden. Diese Klassifikation liefert einen Hinweis darauf, welche Komponenten näheren Untersuchungen unterzogen werden sollten.

Mit Hilfe der statischen Analyse von Programmen können verschiedenste Aspekte untersucht werden. Dazu zählen:

- *Die Analyse der Programmstruktur auf Modulebene*. Dazu stellen die Werkzeuge sogenannte Kontrollflußgraphen zur Verfügung.
- *Überprüfung von Programmierrichtlinien*. Dazu verfügen die Werkzeuge über eine Menge vordefinierter und anerkannter Programmierrichtlinien. Diese können jedoch - wenn auch nicht immer einfach - an firmenspezifische Standards angepaßt werden.
- *Berechnen von Metriken* (z.B. zyklomatische Komplexität oder Anzahl der Verschachtelungstiefe)

Das nachfolgende Bild zeigt schematisch, wie die statische Analyse von Programmen durchgeführt wird.

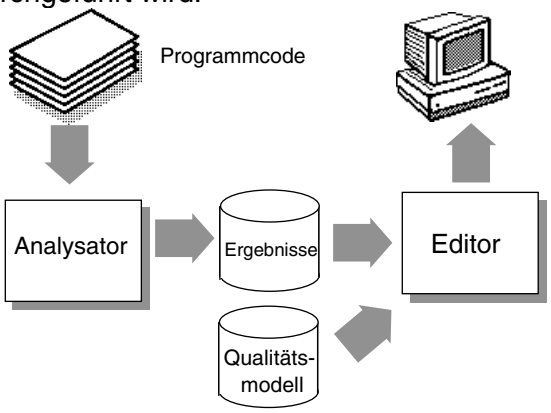

Bild 3: Ablauf bei der statischen Programmanalyse

Sprachspezifische Analysatoren untersuchen den Programmcode und legen die dabei erzielten Ergebnisse in einer oder mehreren Dateien ab. Mithilfe von speziellen Editoren können die Ergebnisse betrachten werden. So können z.B. Aufrufbeziehungen zwischen Modulen oder Funktionen visualisiert werden oder der Kontrollflußgraph einer Funktion betrachtet werden. Auf der Basis der bei der Analyse erzielten Ergebnisse können weiterhin Bewertungen vorgenommen werden. Dazu wird auf gespeicherte Qualitätsmodelle zurückgegriffen.

4. Erfahrungen

In diesem Projekt haben wir versucht, die statische Analyse von Programmen systematisch anzuwenden. Dazu haben wir, nachdem das entsprechende Werkzeug ausgewählt war, eine Vorgehensweise formuliert, die es erlaubt, ein Programm auf der Basis seines Makefiles systematisch zu analysieren.

Des weiteren haben wir einen Leitfaden entwickelt, der herangezogen werden kann, um die Ergebnisse der Analyse zu interpretieren und weiter zu verarbeiten. Dadurch wird gewährleistet, daß vergleichbare Ergebnisse und Bewertungen entstehen. Dieser Leitfaden führt auch dazu, daß die am Projekt beteiligten Entwickler eine gemeinsame Sicht auf Problembereiche entwickeln konnten, die aufgrund der systematischen statischen Programmanalyse weiter betrachtet werden sollen. Zu diesem Problembereichen zählen beispielsweise

- ähnliche, fast identische Codeteile,
- schlecht strukturierte Codeteile,
- extrem triviale Codeteile oder
- isolierte Codeteile.

Im Rahmen des Projektes haben wir eine Vielzahl von S.P.I.D.E.R. Modulen systematisch statisch analysiert. Dabei wurden Module, die in C, C++ und FORTRAN geschrieben wurden, untersucht. Insgesamt wurden etwa 5.000 Funktionen analysiert, die zusammen etwa 1.300.000 Zeilen Code ergeben.

Nachfolgend werden die Erfahrungen vorgestellt, die wir dabei gewonnen haben. Wir gliedern diese in Erfahrungen mit dem

eingesetzten Werkzeug, Erfahrungen mit den am Projekt beteiligten Mitarbeitern und in Management Erfahrungen.

4.1 Werkzeug

Wie bereit erwähnt, haben wir das Werkzeug Logiscope der Firma Verilog eingesetzt. Genauer gesagt ist Logiscope ein Werkzeugkasten, der für die verschiedenen unterstützen Programmiersprachen einzelne Analyse und Editorwerkzeuge anbietet. Logiscope ist, das hat unsere Erfahrung deutlich gemacht, kein einfaches und erst recht kein “plug-and-play” Werkzeug. Bedingt durch die schwierigen Randbedingungen der S.P.I.D.E.R. Umgebung mit hochkomplexen Makefiles, verschiedenen Präprozessoren, einer Mixtur aus C, C++ und FORTRAN Modulen, war es nicht einfach, das Werkzeug so zu kalibrieren, daß es S.P.I.D.E.R. Module analysieren konnte. Es muß jedoch berücksichtigt werden, daß Logiscope spezielle Probleme auf der von uns verwendeten DEC-Unix Umgebung hat Auf anderen Hardware-Plattformen sollten die Probleme nicht in diesem Umfang auftreten.

In vielen Fällen mußten work-arounds entwickelt werden, um zum Ziel zu gelangen. So konnten wir beispielsweise nicht die vorhandenen makefiles verwenden und mußten neue speziell für Logiscope geeignete makefiles schreiben. Trotzdem gelang es nicht immer, alle Module zu analysieren. In einigen Fällen kollabierte Logiscope bei der Analyse von Modulen. Die anschließend gezeigten Fehlermeldungen waren in einigen Fällen nicht ausreichend, um die Fehlerquelle zu identifizieren.

Wir haben insbesondere ein Werkzeug vermißt, mit dem wir die Einzelergebnisse der Analysen aggregierend betrachten und interpretieren konnten. Zu diesem Zweck wurde eine Excel-Anwendung erstellt, die dieses erlaubt. Sie erhält als Eingabe die Analyseergebnisse, die Logiscope berechnet, und stellt diese zusammengefaßt und übersichtlich dar.
Zusammengefaßt muß gesagt werden, daß Logiscope - ein Werkzeug der oberen Preisklasse - an vielen Stellen problematisch im Einsatz war und nur von Spezialisten bedient werden konnte. Auch die Unterstützung von seiten des Lieferanten hätte besser sein können.

Positiv hingegen muß gesagt werden, daß die Qualitätsmodelle, die Logiscope mitliefert, praxisnah und geeignet für die Bewertung sind (darunter fallt auch das Qualitätsmodell nach ISO 9126).

4.2 Mitarbeiter

Wie bei jedem Projekt, bei dem neue unbekannte Techniken erprobt und eingesetzt werden, findet man Mitarbeiter, die dem Projekt positiv gegenüberstehen. Auf der anderen Seite gibt es auch Mitarbeiter, die dies direkt oder indirekt ablehnen. Dieses scheint insbesondere bei Maßnahmen der Fall zu sein, die dazu verwendet werden, Qualität oder Produktivität zu messen.

In unserem Projekt, bei dem es u.a. auch um die qualitative Bewertung der S.P.I.D.E.R. Module ging, war dies ebenfalls so. So fanden wir Mitarbeiter, die die Hinweise und die Ergebnisse der statischen Programmanalyse positiv aufnahmen und als Verbesserungsansatz bewerteten. Diese Mitarbeiter waren überwiegend der Meinung, daß sie selbst die Fehlerquellen, die durch den Einsatz des Werkzeugs gezeigt wurden (z.B. isolierte Funktionen, oder komplexe teils indirekt rekursive Funktionen), gar nicht oder erst nach erheblichem Aufwand gefunden hätten. Diese Mitarbeiter begrüßten durchweg den Einsatz der systematischen statischen Programmanalyse, weil sie darin neben den bereits eingesetzten Mitteln wie Reviews, Code-Inspektionen oder Tests eine weitere Hilfsmittel für die Verbesserung der Module sehen.

Wichtig war es, die Mitarbeiter dahingegen zu informieren und motivieren, daß sie die Ergebnisse der statischen Analyse als Hilfsmittel für Verbesserungen ansehen und nicht als ein Meßwerkzeug, das ihre persönliche Leistungsfähigkeit bewertet. Dies ist uns nicht in allen Fällen gelungen. So mußten wir im Laufe des Projektes häufig Diskussionen darüber führen, inwieweit es überhaupt sinnvoll ist, Messungen an Programmcode durchzuführen. Meinungen wie „Komplexität ist immer relativ zur Problemstellung und zur Fähigkeit des Programmierers zu sehen“ oder „Es macht überhaupt keinen Sinn, Grenzen für die Länge von Funktionen anzugeben oder GOTOs zu verbieten“ waren dementsprechend öfters anzutreffen.

Im nachhinein muß festgestellt werden, daß wir der Information und der Motivation der Mitarbeiter zuwenig Beachtung geschenkt haben. Bevor Maßnahmen wie die systematische statische Analyse von Programmen eingeführt werden können, müssen allen davon Betroffenen die Ziele und die Grenzen der Maßnahme deutlich gemacht werden. Die Akzeptanz bei den Entwicklern ist eine wesentliche Voraussetzung, damit die Potentiale, die mit einer Verbesserungsmaßnahme erzielt werden können, auch erzielt werden.

4.3 Management

Im Laufe des Projektes konnten wir zeigen, daß die Ergebnisse der systematischen statischen Analyse geeignet sind, um als eine Grundlage bei bestimmten Entscheidungen verwendet zu werden. Hierzu zählen wir die z. B. die Entscheidung,

- ob ein Modul weiterentwickelt werden kann oder ob eine Neuentwicklung durchgeführt werden muß
- ob Module portiert werden sollten.

Voraussetzung dazu ist, daß die systematische statische Programmanalyse über mehrere Release-Zyklen durchgeführt wird. Die dabei gemessenen Ergebnisse und die darauf basierenden Bewertungen geben einen Anhaltspunkt dafür, in welche Richtung sich die Qualität einer Release oder der einzelnen Module bewegt. Diese Informationen können bei der zukünftigen Releaseplanung verwendet werden, um Korrektur- oder Redesign-Aufwände zu argumentieren.

Auf seiten des Managements sind die Ergebnisse der systematischen Programmanalyse in diesem Sinn als wertvoll eingestuft worden.

5. Ergebnisse des Projekts

Das in diesem Beitrag geschilderte Projekt wurde Ende April 97 abgeschlossen. Neben den technischen Erfahrungen und Ergebnissen haben wir vieles für kommende Projekte gelernt, die andere neue Verbesserungsmaßnahmen erproben und einführen sollen.
Eines der Ergebnisse, die wir erzielt haben ist, daß sich das Qualitätsmodell nach ISO 9126 im großen und ganzen eignet, um qualitative Bewertungen von Programmcode durchzuführen. Am Anfang des Projektes waren wir skeptisch und glaubten, eine Modellanpassung durchführen zu müssen. Dies ließen wir jedoch fallen, nachdem wir viele statistische Auswertungen und Gegenüberstellungen der von uns analysierten Module durchgeführt hatten.

Weiterhin erwähnenswert scheint uns der folgende Zusammenhang, den wir häufig beobachten konnten: Die Menge der Funktionen, die vom Qualitätsmodell als schwach bewertet wurde, macht relativ zur Gesamtheit aller analysierten Funktionen eine geringen Anteil aus (> 10%). Jedoch muß festgestellt werden, daß diese Funktionen einen großen Teil des Codes ausmachen, wenn man bei dieser Messung die Anzahl der unkommentierten Zeilen Programmcode verwendet (>35%).

Nachfolgend fassen wir unsere Ergebnisse mit der statischen Programmanalyse (SPA) in Form von neun Aussagen zusammen.

> ***A1**: Mithilfe der SPA können Schwachstellen und „Tumore“ in der Software entdeckt werden.*

Dadurch daß wir viele Module systematisch analysiert und “vermessen” haben, haben wir recht bald Stellen oder Funktionen erkannt, die besonders kritisch im Sinne des verwendeten Qualitätsmodells sind. Diese Funktionen wurden speziell untersucht. Damit wurde u.a. auch validiert, inwieweit das Qualitätsmodell mit den Qualitätsvorstellungen der Entwickler übereinstimmte.
Wird die SPA über Release-Zyklen hinweg eingesetzt, so kann erkannt werden, in welchen Bereichen die Qualität zu- bzw. abzunehmen scheint. Auch hier kann dieses mit dedizierten Untersuchungen validiert werden. Entsprechend den Ergebnissen dieser Untersuchungen können Gegenmaßnahmen (z.B. Redesign und Neu-Implementierung von Komponenten) eingeleitet werden.

***A2**: Mithilfe der SPA kann der Aufwand für die Einarbeitung in neuen unbekannten Code vermindert werden.*

Insbesondere bei sehr großen über die Jahre gewachsenen Softwaresystemen stehen die Entwickler häufig vor der Situation, daß sie Modifikationen in Komponenten oder Modulen vornehmen müssen, die nicht von ihnen entwickelt wurden. Bei der Einarbeitung in solche unbekannte Komponenten hilft die SPA erheblich, da sie in geeigneter und übersichtlicher Art und Weise die Strukturen der Komponente visualisiert. Dies trägt dazu bei, daß die Einarbeitungszeit kürzer wird.

***A3**: Die Ergebnisse der SPA müssen in jedem Fall interpretiert werden.*

Unsere Erfahrungen zeigen, daß die Bewertungen auf der Basis der Qualitätsmodelle immer interpretiert und in Frage gestellt werden sollten.
Diese Bewertungen sind keine JA/NEIN Aussagen, die automatisch dazu führen, daß schlecht bewertete Codeteile auch wirklich immer schlecht sind. Es kann oft gute und nachvollziehbare Gründe dafür geben, daß die Implementierung und die Struktur einer Komponente genau so zu wählen war. Die Bewertungen dienen lediglich als Hinweise für potentielle Schwachstellen in der Software, und müssen im einzelnen überprüft werden.

***A4**: Ein Werkzeug zur SPA gehört in jede industriell eingesetzte Software-Entwikklung-umgebung*

Allzu häufig findet man auch heute noch die Situation vor, daß die Entwicklungsumgebung lediglich aus Editor, Compiler, Linker und Debugger besteht. Genauso wie ein Werkzeug zur Versions- und Konfigurationsverwaltung ein Muß ist, sind wir der Meinung, daß jeder Entwickler in seiner Entwicklungsumgebung auch ein Werkzeug für die SPA vorfinden sollte.

***A5**: Das für die SPA notwendige theoretische und technische Know-how muß vorhanden sein, damit die SPA systematisch durchgeführt werden kann.*

Es muß genügend Wissen über die zugrundeliegenden Qualitätsmodelle und über die darin verwendeten Metriken vorhanden sein, damit die Ergebnisse der SPA sinnvoll interpretiert werden können. Wie unsere Erfahrungen zeigen, sind auch die Werkzeuge für die SPA nicht intuitiv und einfach zu bedienen. Da es aber nicht ökonomisch ist, daß jeder Mitarbeiter das werkzeugspezifische Know-how haben muß, müssen dafür organisatorische Lösungen gefunden werden.

***A6**: Die SPA muß in den Software-Entwicklungsprozeß eingebaut sein, damit alle Potentiale ausgeschöpft werden können.*

Auch wenn es durchaus sinnvoll ist, nur ab und zu statische Programmanalysen durchzuführen, so denken wir, daß eine softwareentwickelnde Organisation die Potentiale der SPA nur dann voll ausschöpfen kann, wenn diese systematisch im Entwicklungsprozeß eingesetzt wird.
Dazu gehört auch, daß es einen Leitfaden oder entsprechende Richtlinien geben muß, die beschreiben, wie die Ergebnisse der SPA zu interpretieren sind, und was als Konsequenz daraus durchzuführen ist.

***A7**: Die SPA darf nicht dazu genutzt werden, um die Leistungsfähigkeit der Entwickler zu messen.*

Wenn dieses getan wird, dann werden die Entwickler bald Wege finden, wie sie die SPA entweder umgehen können oder wie sie das Werkzeug zu für sie akzeptablen Bewertungen einstellen können. Dies ist aber kontraproduktiv. Die Mitarbeiter sollen nicht im Sinne des Werkzeugs entwickeln, sondern sie sollen die Ergebnisse der SPA verwenden, um dediziert Verbesserungen vorzunehmen.

***A8**: Die Ergebnisse der SPA können vom Management entscheidungsunterstützend verwendet werden.*

Wie bereits im vorigen Abschnitt erläutert, können die Ergebnisse der SPA bei Entscheidungen herangezogen werden, wenn es darum geht, ob Module weiterentwickelt oder neu implementiert werden müssen. Die Bewertungen der SPA sind in der Regel eine bessere - da quantifizierbare - Grundlage für diese Entscheidungen, als wenn man sich dabei ausschließlich auf das „Gefühl“ der Entwickler verlassen muß.

***A9**: Die SPA wird nur dann von den Entwicklern akzeptiert und angewendet, wenn diese die SPA als Hilfsmittel ansehen.*

Die Versuchung ist sehr groß, die SPA als Meßverfahren für die Leistungsfähigkeit der Entwickler zu verwenden. Unsere Erfahrungen haben gezeigt, daß dieses von Entwicklern häufig als Hauptgrund genannt wurde, wenn dieses Verfahren eingesetzt werden sollte. Es gilt deshalb bereits im Vorfeld der Einführung unmißverständlich klarzumachen, welche Ziele mit der SPA verfolgt werden und wozu sie nicht verwendet wird.

Wir glauben, daß es schwierig, wenn nicht sogar unmöglich ist, ein Verfahren wie die SPA von „oben herab“ einzuführen. Auf der anderen Seite braucht man natürlich die Rückendeckung des Managements, da der Wissensaufbau, der für die systematische SPA notwendig ist, eine nicht unerhebliche Investition benötigt.

6. Ausblick

Die im geschilderten Projekt gemachten Erfahrungen sind durchweg positiv. Wir konnten zeigen, daß die statische Programmanalyse ein - neben den bereits verwendeten Maßnahmen wie Review und Test - probates Mittel ist, um die Qualität von Software-Systemen zu verbessern. Da unser Projekt im wesentlichen ein Feldversuch war, können wir zur Zeit keine Bewertungen dieses Verfahrens im Sinne der “Return on Investment” machen.

Wir haben als eines der Ergebnisse dieses Projektes ein Konzept erarbeitet, wie die statische Programmanalyse in den Software-Entwicklungsprozeß der ABB Netzleittechnik integriert werden kann. Dazu zählt auch der Vorschlag, im Rahmen der Qualitätsorganisation ein sogenanntes “Test-Office” zu etablieren. Dort könnte das für die Programmanalyse benötigte Wissen zentral aufgebaut und den entwickelnden Einheiten zur Verfügung gestellt werden. So wäre es denkbar, daß Mitarbeiter des Test-Office bei der statischen Programmanalyse unterstützend mitarbeiten oder diese im Auftrag der entwickelnden Einheiten durchführen. Dieses könnte wesentlich dazu betragen, daß die Potentiale der statischen Programmanalyse auch tatsächlich ausgeschöpft werden.

Literatur

Bröhl, A.-P., W. Dröschel (1995): Das V-Modell- Der Standard in der Software-Entwicklung mit Praxisleitfaden, Oldenbourg Verlag.

Paulk, M.C., B. Curtis, M. Chrissis (1991): Capability maturity model for software. *SEI Tech. Rep*. CMU/SEI-91-TR-24.

ISO (1991): ISO 9126, “Information technology - Software product evaluation - Quality characteristics and guidelines for their use”.

7. Kurzbiographie

Horst Lichter
Studium der Informatik an der Universität Kaiserslautern. Anschließend wissenschaftlicher Mitarbeiter in der Abteilung Software Engineering der ETH Zürich und der Universität Stuttgart. Promotion mit einer Arbeit zum Thema Prototyping und objektorientierte Software-Architekturen. Von 1993 bis 1995 Mitarbeiter der Schweizerischen Bankgesellschaft Zürich. Seitdem Mitarbeiter in der Abteilung Information Technology des ABB Forschungszentrums Heidelberg.

Peter Piwecki
Studium der Nachrichtentechnik mit Schwerpunkt Hoch- und Höchstfrequenztechnik an der FHT-Mannheim. Anschließend Mitarbeiter der ABB-Kraftwerksleittechnik in der Entwicklung von Koppelsystemen zwischen Leittechnik und MMI-Systemen. Von 1994 bis 1996 zuständig für den Bereich SW Systemintegrationstest. Seit 1997 Mitarbeiter Heidelberg in der Abteilung Information Technology des ABB Forschungszentrums Heidelberg.

Gerhard Riedinger
Studium der Elektrotechnik - Studienmodell Informa-tionsverarbeitung- an der Universität Karlsruhe. Seit 1980 Software-Ingenieur bei ABB Netzleittechnik in Ladenburg. Mitarbeit in verschiedenen Software-Projekten im Bereich Netzleittechnik in Pascal, Fortran, C und C++ unter VMS, UNIX und VxWorks. Langjährige Erfahrungen im Bereich Fehlerver-folgung und qualitätssichernde Maßnahmen.