Rüdiger Schilling

(E-Mail: rschilling@d-s-t-g.com)
ist Geschäftsführer der Delta Software Technology GmbH. In dieser Funktion ist er seit mehr als zwanzig Jahren verantwortlich für die Konzeption und den Einsatz generativer Werkzeuge zur Automation von Entwicklungsaufgaben im Bereich der Software-Entwicklung, Wartung, Integration und Modernisierung.

Radikale Änderung der Teststrategien durch 100%-ige Automation von Massenänderungen

Unter welchen Bedingungen lassen sich die Fehlerquoten bei Massenänderungen senken und wie weit? Wie lassen sich die Risiken für die produktiven Systeme reduzieren? Wie sehen sinnvolle Organisationsstrategien aus?

In großen Änderungsprojekten ist der Test einer der größten Kostenfaktoren. Massenänderungen an produktiven Anwendungen, wie sie für Migrationen, zur Erfüllung gesetzlicher Bestimmungen, Architekturkonsolidierung, Modernisierung und für bestimmte Erweiterungen wie Unicode erforderlich sind, stellen andere Anforderungen an Werkzeuge und Vorgehensweisen als „normale" Entwicklungsprojekte. Sowohl Outsourcing als auch die Entwicklung eigener Tools können einen Teil der Änderungskosten einsparen. Aber beide Ansätze helfen nicht, die größte Hürde – den Test – zu nehmen. Durch eine 100%-ige Automation der Änderungen lassen sich die Fehlerquoten jedoch drastisch (bis nahe 0) senken. Die vollständig automatische Durchführung der Änderungen eröffnet neue Wege der Projektorganisation, die den Testaufwand und die Risiken dieser Projekte sehr stark reduzieren.

Besondere Probleme bei massenhaften Änderungen

Die effiziente, kostengünstige und sichere Durchführung von Massenänderungen an der produktiven Anwendungssoftware ist ein Dauerthema. Anforderungen dazu gibt es permanent – für alle Sprachen, Systeme, Architekturen und Implementierungstechniken. Die Auslöser sind vielfältig, mal wirtschaftlicher, mal gesetzlicher, mal unternehmenspolitischer Natur: Plattformwechsel, neue Gesetze, Optimierungen, Konsolidierung, etc.

Einige Beispiele für Massenänderungen, die im Rahmen solcher Projekte notwendig werden, sind Datenstrukturänderungen, wie sie für UNICODE, SEPA, Mehrsprachigkeit etc. benötigt werden, Plattformmigrationen, Sprachkonversionen, aber auch komplexe Architekturtransformationen, wie sie bei der Anwendungsmodernisierung für die Auslagerung von Datenzugriffen und Präsentationsschichten oder der Einführung service-orientierter Architekturen (SOA) gebraucht werden.

Typisch für diese Art der (nicht-funktionalen) Änderungen sind die folgenden Probleme:

- **Massenhafte Änderungen:** Änderungen quer durch alle Module resultieren in extrem hohen Änderungsvolumen und einem damit verbundenen extrem hohen Testbedarf.
- **Abhängigkeiten zwischen Änderungen:** Änderungen müssen aufeinander abgestimmt und synchronisiert werden, damit die resultierenden Module noch zusammenpassen.
- **Abgrenzung der Tests:** Tests können nicht auf einzelne Funktionen und deren Schnittstellen beschränkt werden, Änderungen pflanzen sich durch das gesamte System fort, Clusterbildung von Tests ist äußerst schwierig.
- **Hohes Risiko:** Projekte mit massenhaften Änderungen sind quasi Operationen am lebenden Objekt, weil sie immer die produktiven Anwendungen involvieren.
- **Projektgröße:** Lange Projektlaufzeiten verschärfen die vorgenannten Punkte. Konflikte entstehen, weil parallel neu und weiterentwickelt, d. h. an den gleichen Modulen geändert wird.
- **Deadlines:** Sehr häufig sind diese Änderungen zeitkritisch, z. B. bei gesetzlichen Änderungen oder bei Abhängigkeiten von externen Partnern.

Die Regeln für die erforderlichen Code-Transformationen festzulegen, ist vergleichsweise einfach. Entsprechend der Aufgabenstellung, also bezogen auf klar identifizierbare Aspekte, müssen geeignete Discovery-Regeln alle potenziell betroffenen Stellen auffinden, Analyse-Regeln die Abhängigkeiten und Auswirkungen über die gesamten Anwendungen ermitteln und Transformationsregeln die eigentlichen Änderungen durchgängig und konsistent in allen Anwendungskomponenten durchführen.

Im Unterschied zu Änderungen aufgrund von funktionaler Neu- und Weiterentwick-

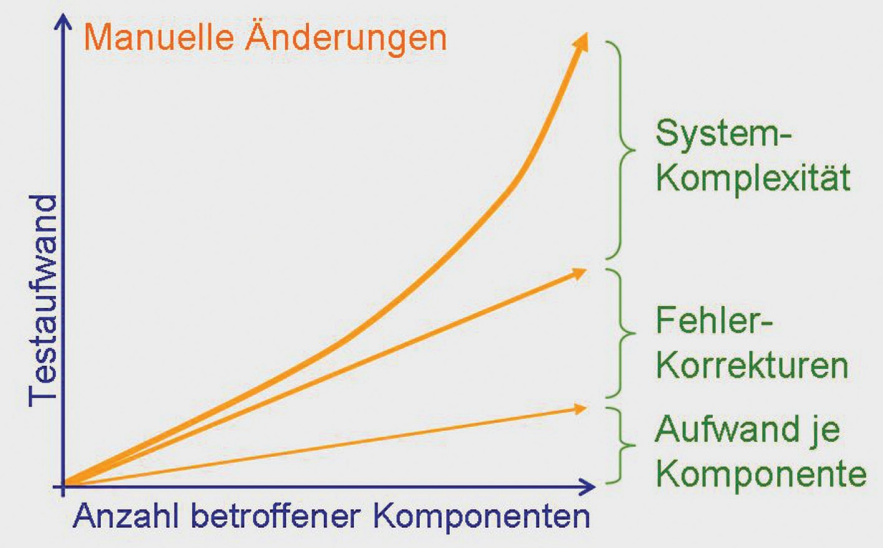

Abb. 1: Überproportional steigender Testaufwand bei manuellen Änderungen

lung sind die Änderungen im Einzelnen selten komplex. Es geht bei den Modifikationen in erster Linie um eine Fleißaufgabe, die in der Regel nicht einmal die genaue Kenntnis der Anwendungen voraussetzt. Die Probleme beim Testen ergeben sich nicht aus der Komplexität der Änderungen an sich, sondern aus der Vernetzung und der Komplexität der Anwendungen sowie der großen Menge des betroffenen Codes. Bei Massenänderungen müssen tausende Module (also Millionen Lines of Code) geändert und getestet werden. Die massenhafte Änderung ganzer Anwendungen gehen weit über das hinaus, was man als Refactoring aus dem Bereich moderner Programmiersprachen kennt (siehe Infoblock: Das Produkt – AMELIO Modernization Platform).

Automation vs. Vollautomation der Änderungen

In großen Änderungsprojekten benutzt natürlich jeder Werkzeuge und jeder automatisiert. Viele Werkzeuge analysieren nur, manche bieten einen 80%-Ansatz. Sie ermöglichen die automatische Durchführung „nahezu aller“ Änderungen. Das heißt in der Regel, dass bestimmte, meist einfache, standardisierbare Änderungen automatisch umgesetzt werden, der verbleibende Rest der schwierigen und komplexen Änderungen dann aber manuell gemacht wird. Auch selbst entwickelte Änderungsscripts arbeiten oftmals so.

Werden Module manuell geändert, ist der Testaufwand je Komponente direkt abhängig vom Umfang der Änderungen; gleiches gilt für Fehlerkorrekturen und Testwiederholungen. Mit steigender Systemkomplexität, d. h. steigender Größe des zu testenden Gesamtsystems, wachsender Vernetzung der Komponenten und Komplexität der Schnittstellen, steigt der Aufwand jedoch überproportional (s. Abbildung 1).

Der mit manueller Programmierung, Scripts oder auch vielen Tools erreichbare Automationsgrad reicht nicht aus, um den Testaufwand wirklich zu reduzieren. Ob 80%, 85% oder auch 90% maschinell und der Rest manuell geändert wird, ob inhouse oder per Outsourcing, das Ergebnis bleibt gleich, weil der manuelle Anteil immer noch relevant ist. Was sich zunächst wie eine gute Lösung anhört, mit der man schon mal ein erhebliches Stück weiterkommt, erweist sich spätestens beim Test als Bumerang. Selbst wenn 90% der Änderungen in 10.000 Modulen automatisch durchgeführt werden würden und 10% manuell, müssten immer noch alle 10.000 Module zu 100% getestet werden.

Erst wenn man einen so hohen Automationsgrad erreicht, dass der Anteil der manuell geänderten Module nicht mehr relevant ist, reduziert sich der Testaufwand signifikant. Der größte Vorteil automatischer Änderungen ergibt sich dort, wo manuelle Änderungen ihre größte Schwäche haben: bei den Tests. Mit vollständiger Automation der Änderungen, d. h. wenn ein Werkzeug die Module zu 100% automatisch ändert, kann der Test von der Ebene der einzelnen Änderungen auf die Ebene der Regeln verlagert werden (Meta-Level-Testing). Und genau in diesem Fall verringert sich der Testaufwand signifikant.

Meta-Level-Testing reduziert Testaufwand drastisch

Bei der Neu- und Weiterentwicklung von Software müssen Tests die Funktionsfähigkeit und Integrität der Anwendungen nachweisen. Im Falle der hier betrachteten Massenänderungen am Source-Code ändert sich an der inhaltlichen Funktionsweise der Anwendungen nichts, d. h. hier müssen Tests die Korrektheit der Code-Transformationen verifizieren.

Die Anzahl der Regeln zur Definition aller Änderungsfälle und Transformationen ist überraschend gering. Sie stehen jedoch einer extrem hohen Anzahl einzelner Code-Änderungen gegenüber. So genügten in einem sehr großen Modernisierungsprojekt ca. 100 Regeln, um alle Änderungen für

a) die Herstellung der Plattformunabhängigkeit der Anwendungen im Vorfeld einer Plattformmigration und
b) die Architekturänderung (SOA) mit Auslagerung der Datenzugriffsschicht sowie
c) die Neutralisierung der Front-Ends zu definieren,

obwohl die Systeme extrem voneinander abweichende Datenstrukturen und Sprachelemente aufwiesen. Insgesamt wurden mit diesen Regeln 14 Millionen Lines of Code analysiert und 1,3 Millionen Änderungen durchgeführt (siehe Infoblock: Referenz – Modernisierung bei RDW).

Für Code-Transformationen nach wohldefinierten Regeln im Rahmen von Massenänderungen ist deshalb folgende Teststrategie sinnvoll: Anstatt jede Änderung individuell zu testen, werden die Regeln als solche getestet. Dieser Test findet auf der übergeordneten Ebene, der Meta-Ebene statt. Diese Meta-Level-Tests verifizieren die Regeln für die Discovery, die Analyse und die Transformation des Codes. Sie sind keine Funktionstests für die Anwendungen.

Offensichtlich reduziert das Meta-Level-Testing den Testaufwand drastisch. Es ist weniger aufwendig, die oben beispielhaft genannten 100 Regeln zu testen als die 1,3 Millionen Änderungen. Der kostengünstigste Test ist jener, der *nicht* gemacht werden muss! Die Voraussetzung ist jedoch, dass keine manuellen Änderungen vorgenommen werden. Alle Änderungen müssen in jedem einzelnen Programm zu 100% automatisch durchgeführt werden! Nur so kann sichergestellt werden, dass die richtige Regel angewandt, der richtige Algorithmus ausgeführt wird und die Änderungen reproduzierbar sind.

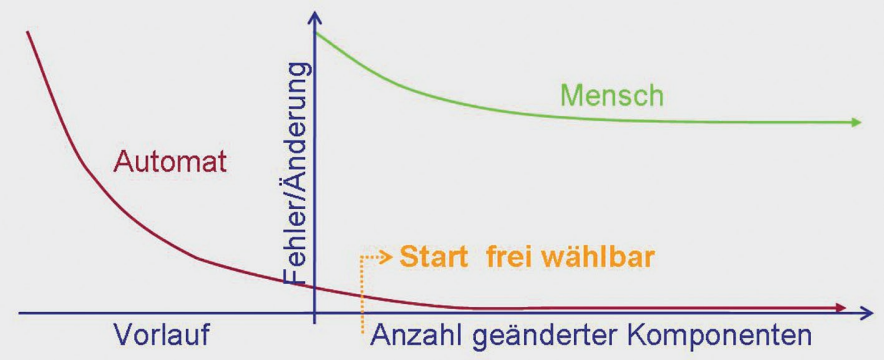

Abb. 2: Vollständige Automation – Transformationszeitpunkt frei wählbar

Qualität durch vollständige Automation: Fehlerquote nahe 0

Zur Entwicklung der Regeln und Algorithmen für die vollständig automatische Durchführung der Änderungen, d. h. zur Einrichtung eines Software-Automaten (Transformation Factory), empfiehlt sich eine agile und iterative Arbeitsweise auf der Meta-Ebene, bei der die Regeln und Algorithmen so lange getestet und optimiert werden bis sie fehlerfrei sind – bezogen auf die Gesamtheit aller betroffenen Anwendungen. Genau hier ist der Punkt, an dem auch manuell eingegriffen werden kann: Nicht alle Entscheidungen können von vornherein automatisch getroffen werden. Bei der Einrichtung der Transformation Factory lernt diese erst, was in welchem Fall anzuwenden ist. Wenn eine Analyse-Regel etwas nicht zweifelsfrei entscheiden kann, dann ist eine manuelle Entscheidung gefragt. Wichtig ist, dass auch die manuellen Entscheidungen festgeschrieben und in die automatischen Prozesse integriert werden. Sie müssen ebenso verifiziert und zurückgenommen werden können, falls sie zu Fehlern oder Widersprüchen führen. Es ist ein iterativer Prozess: Ändern, Testen, Optimieren, Testen, Ändern, … bis sämtliche Änderungen zu 100% vollständig automatisch durchgeführt werden können.

Alle Änderungen sind jederzeit 1:1 reproduzierbar und werden vollständig automatisch dokumentiert. Die Regeln und Algorithmen für die Änderungen können auch am gesamten Source-Code verifiziert werden. Treten Fehler auf, kommen Änderungsregeln hinzu, korrigiert man die Regeln und verifiziert erneut. Das kostet zwar ein wenig Zeit, tut aber nicht weh. Die produktiven Anwendungen sind ebenso wenig in Gefahr wie die parallel laufenden Projekte. Eine Fehlerrate von (nahe) Null ist so erreichbar – bevor die eigentlichen Massenänderungen am produktiven Source-Code beginnen.

Die Risiken für die produktiven Systeme lassen sich durch die 100%-ige Automation der Änderungen vollständig ausschalten. Die eigentliche Transformation aller Sourcen kann auf einmal, insgesamt und vollständig vorgenommen werden. Der Transformationszeitpunkt kann völlig frei gewählt werden. Salopp ausgedrückt: Ein Big-Bang ohne Knall (siehe Abbildung 2).

Organisation: Entkopplung der Projekte

Ein besonders kritisches Problem großer Änderungsprojekte ist die Konkurrenz zu anderen Projekten. Parallele Neuentwicklung, Weiterentwicklung und die normale Maintenance, all das betrifft Module, die auch von den Massenänderungen betroffen sind.

Hat man einen signifikanten Anteil manueller Änderungen und muss jede einzelne Änderung testen, können Änderungen nicht einfach zurückgenommen werden und die Änderungsprozesse aufgrund der Abhängigkeiten und der langen Durchlaufzeiten nicht einfach parallelisiert werden. Skalierung über größere Teams verursacht höhere Kosten und ein höheres Risiko. Entwicklung, Wartung und Massenänderungen blockieren sich gegenseitig (s. Abbildung 3).

Bei vollständiger Automation der Änderungen übernimmt der Software-Automat (Factory) alle Arbeitsschritte: (Teil)Systeme werden vollständig erfasst (Discovery), im Zusammenhang analysiert (Analyse) und gesamthaft automatisch transformiert (Transformation). Dazu braucht man nur wenige Mitarbeiter. Die (Teil)-Prozesse können (zeitlich) getrennt und parallelisiert und deshalb einfach durch den Einsatz weitere Rechner beschleunigt werden.

Die Einrichtung der Factory ist unabhängig von parallel laufenden Projekten. Die Regeln und Algorithmen werden in der Vorbereitungsphase mit einer Kopie der produktiven Anwendungen verifiziert. Deshalb können alle Regeln und Algorithmen für die Änderungen im Voraus getestet werden. Die Tests können beliebig vorgezogen werden - ohne Einfluss auf laufende Entwicklungen oder die Produktion. Werden bereits analysierte Module durch andere Projekte geändert oder ändern sich die Anforderungen, werden sie einfach erneut analysiert bzw. wird die Analyse mit geänderten Regeln wiederholt. Im Extremfall können die Transformationen vorab mit dem gesamten Anwendungsvolumen verifiziert werden (s. Abbildung 4).

Vollautomatische Prozesse sind beliebig oft, mit wenig Aufwand und schnell wiederholbar. Deshalb ist es möglich:

- frühzeitig zu testen
- Änderungen probeweise durchzuführen, um Regeln, Algorithmen und deren Auswirkungen zu testen
- reproduzierbar zu ändern: Wenn Änderungsregeln angepasst werden, neue Bedingungen hinzu kommen, werden die Auswirkungen schnell erkannt, alle Sourcen können erneut analysiert, alle Auswirkungen neu ermittelt werden und alle Änderungen entsprechend den jetzt gültigen Regeln erneut durchgeführt werden. Wohlgemerkt, alles vollautomatisch!
- Module, die parallel in anderen Projekten geändert wurden, erneut durch die Massenänderungen laufen zu lassen

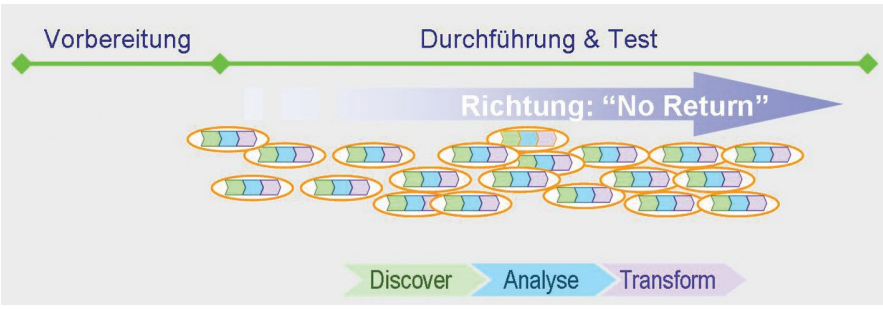

Abb. 3: Abhängigkeiten bei manuellen Änderungen

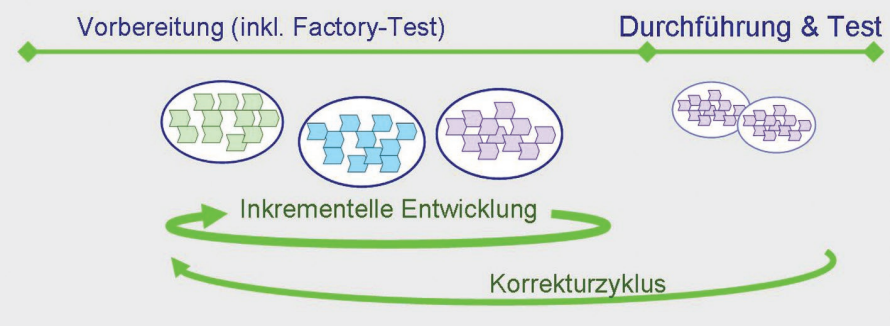

Abb.4: Vollständige Automation – Entkopplung und Unabhängigkeit

- durchgängig mit 100% Abdeckung zu testen, d. h. jede Änderungsregel zu verifizieren, die Richtigkeit jeder Änderung revisionssicher zu belegen
- Änderungen selektiv rückgängig zu machen
- bereits geänderte und noch nicht geänderte Module zu kombinieren.

Die vollständige Automation der Änderungen ist mehr als Fließbandfertigung. Die Projekte können anders organisiert und voneinander entkoppelt werden, wenn man alle Änderungen und deren Auswirkungen vor der tatsächlichen Implementierung testen kann und wenn man 100%-ig sicherstellen kann, dass alle Änderungen dann wirklich korrekt durchgeführt werden. Die „normalen" Risiken eines großen Änderungsprojekts lösen sich in Luft auf.

Fazit: Software ändert Software – 100%

Massenänderungen an großen, produktiven Anwendungssystemen stellen andere Anforderungen an die Vorgehensweise und insbesondere an den Test als die Neu- oder Weiterentwicklung. Sowohl Outsourcing als auch die Entwicklung eigener Tools helfen nicht, die größte Hürde – den Test – zu nehmen. Gebraucht werden Strategien und Werkzeuge, die eine effiziente, kostengünstige und sichere Durchführung von Massenänderungen an der produktiven Anwendungssoftware ermöglichen.

Mit vollständiger Automation der Änderungen, das heißt wirklich zu 100%, ist man in der Lage:

- Fehlerquoten nahe 0 zu erreichen
- die Risiken für die produktiven Systeme zu minimieren
- den Testaufwand radikal zu reduzieren
- die Abhängigkeiten von anderen Projekten aufzulösen.

Referenz: Modernisierung bei RDW

RDW ICT, die selbstständige IT der nationalen Kraftfahrzeugverwaltung der Niederlande, modernisierte ihre gesamten Mainframe-Anwendungen mit zwei Zielen: Reduktion der Plattformkosten und Modernisierung der Anwendungsarchitektur als Vorbereitung für die zukünftige Entwicklung. Realisiert wurden

- Plattformunabhängigkeit aller Anwendungen
- Implementierung einer leichter wartbaren separaten Datenzugriffschicht
- Neutrale Front-Ends als Voraussetzung für die zukünftige Verwendung browserbasierter Technologien

RDW ICT, seit fünf Jahren ausgezeichnet als innovativste Regierungsorganisation, für bestes Management und beste Jahresergebnisse, entschied sich gegen Outsourcing und für die vollständige Automation mit der AMELIO Modernization Platform von Delta Software Technology.

- Mehr als 11.000 Module
- 14 Mio. Lines of Code wurden zu 100% automatisch modernisiert
- 1,3 Millionen einzelner Änderungen
- Fehlerrate 0,0024%, d. h. 30 Fehler auf 11.000 modifizierte Komponenten oder ein Fehler auf 43.000 Änderungen!

- Testaufwand um mindestens 90% reduziert
- Return on Investment (ROI) innerhalb eines Jahres

„Ein entscheidender Grund für AMELIO war für uns das neue Meta-Level-Testkonzept: Wir können damit 90% und mehr an Testaufwand und Kosten einsparen." Gerard Doll, Direktor RDW ICT, Niederlande

Infoblock: Referenz: Modernisierung bei RDW

Das Produkt: AMELIO Modernization Platform

AMELIO Modernization Platform von Delta Software Technology ist ein sicheres und wirtschaftliches Werkzeug, das die Modernisierung und Transformation von Anwendungen zu 100% automatisiert durchführt.

AMELIO

- analysiert und ändert Anwendungen vollautomatisch,
- reduziert Testaufwände drastisch,
- realisiert Änderungen, ohne andere Projekte zu blockieren,
- garantiert zu jeder Zeit die Stabilität und Integrität der Anwendungen,
- lässt die Wahl zwischen schrittweiser Umsetzung und „Big Bang".

Einsatzbereiche:

- Massenänderungen: Änderung von Datenformaten, Kunden-, Konto- oder Versicherungsnummern, Jahr 2000, Euro, Swift, UTF-16 (Unicode) für EU-Harmonisierung, …
- Modernisierung: Plattformwechsel, Framework-Anpassung/Austausch, …
- Architektur-Transformation: Service Enablement, Modularisierung, Plattformneutralisierung
- Migrationen und Transformationen
- Quality Assurance

Anwender bestätigen: Große Änderungsprojekte können auf diese Art in kürzerer Zeit, sicherer und mit weniger Ressourcen durchgeführt werden.

Weitere Informationen:
Delta Software Technology
http://www.software4software.com
E-Mail: info@software4software.com
Tel: 02972-97190

Infoblock: Das Produkt: AMELIO Modernization Platform