High-End IT-Management - Wenn Scott Tiger einfach nicht mehr reicht

Sebastian Graf
PROMATIS software GmbH
Pforzheimer Strasse 160, 76275 Ettlingen

Schlüsselworte:

IT-Management, Performance Tuning, High-Availability-Architekturen, Requirements Management, IT-Testing

Einleitung

Failure is not an option. Mit dieser Aussage hat Gene Krenz, ehemaliger Flugdirektor der NASA, seine Mitarbeiter während des missglückten Mondfluges von Apollo 13 daran erinnert, dass bei der Rettung der Crew keine Fehler erlaubt sind. Betrachtet man aktuelle IT-Umgebungen kritischer Anwendungen, so findet man nicht selten veraltete Technologien wie z.B. Host-basierte Architekturen. Auf die Frage, warum derart in die Jahre gekommene Architekturen immer noch im Einsatz sind, bekommt man von den Verantwortlichen nicht selten die Antwort: "Weil wir wissen, dass sie funktionieren". Und betrachtet man die heute zur Anwendung kommenden Software-Entwicklungsprozesse, welche nicht selten auch in hochkritischen Umgebungen zum Einsatz kommen, dann bewegt man sich tatsächlich mitunter im Spannungsfeld zwischen Agilität und explorativem Agieren: Anforderungen und Soll-Prozesse werden nicht verstanden, moderne Technologien und Entwicklungswerkzeuge sind nicht in der Lage, den hohen Anforderungen kritischer Anwendungen mit Blick auf Qualität und Performance gerecht zu werden, und in der Ausbildung junger Softwareingenieure wird weniger Wert auf die Grundzüge des korrekten Vorgehens als auf die Entwicklung von Lines of Code gelegt. Der Beitrag beleuchtet die Herausforderungen bezogen auf kritischen IT-Anwendungen mit speziellem Bezug auf Performance- und Verfügbarkeitsaspekte. Er zeigt an praktischen Beispielen, welche Fehler dabei unbedingt vermieden werden müssen, und geht der Frage nach, ob die wunderbaren neuen Errungenschaften von Java & Co. tatsächlich geeignet sind, wenn es darum geht, IT-Systeme zu entwickeln, deren Anforderungen in der Champions League angesiedelt sind.

Wie steht es um die Erfolgsquote von Softwareprojekten?

In den vergangenen Jahren ist viel über die Qualität in Softwareprojekten gesagt und geschrieben worden, wobei ein einheitliches Bild nicht zu erkennen ist. So gibt es Befürworter der These, dass die Softwarekrise nach wie vor nicht überwunden ist und immer noch sehr viele Projekte scheitern. Ein Vertreter dieser Gruppe ist die Standish Group (http://www.standishgroup.com/), die mit ihrem *CHAOS Report* immer wieder Untersuchungen veröffentlicht, wonach über 30% der Softwareprojekte scheitern, mit jährlich steigender Tendenz. Allerdings wird der Report in der jüngeren Vergangenheit immer skeptischer betrachtet. Nicht zuletzt deshalb, weil die Standish Group das zugrundeliegende Bewertungsverfahren nicht veröffentlicht. Aber auch in weniger kritischen Untersuchungen wird immer noch von einer Fehlerrate von > 15% ausgegangen. Zu diesem Ergebnis kommen z.B. Khaled Eman und A. Koru in Ihrer IEEE Untersuchung *A Replicated Survey of IT Software Project Failures*.

In dieser Studie wird insbesondere darauf hingewiesen, dass es keinen Zusammenhang zwischen dem Umfang eines Projekts und dessen Wahrscheinlichkeit zu scheitern gibt. Bei den Gründen führen Eman und Kru allerdings an, dass sich die Stakeholder in einem größeren Projekt deutlich schwerer tun, dieses vorzeitig als gescheitert zu beenden, weil die bis zu diesem Zeitpunkt getätigten Investitionen nach Möglichkeit geschützt werden sollen. Was aber sind nun die Gründe, die insbesondere große Projekte in Schieflage bringen können? Aus Sicht des Autors sind die folgenden Aspekte in Projekten mit starkem Datenbankbezug für den Projekterfolg von herausragender Bedeutung:

- Die Requirements-Analyse
- Maßnahmen zur Sicherung der Performance der Anwendung
- Maßnahmen zur Sicherung der Verfügbarkeit der Anwendung
- Die eingesetzten Testmethoden

High-End-Analysen

Die meisten gescheiterten Softwareprojekte haben den Grundstein für das Scheitern bereits in der Analysephase gelegt. Auf eine umfassende Analyse der Soll-Prozesse wird verzichtet. Stattdessen wird eine mehr oder weniger vollständige und konsistente Requirements-Sammlung mit unterschiedlichsten, für diesen Zweck in der Regel völlig ungeeigneten Werkzeugen (in der Regel handelt es sich um unstrukturierte Informationen, die in Excel, Word und PowerPoint gesammelt werden) durchgeführt. Selbstverständlich sind diese ganzen „Projektartefakte“ in einem Vorgehensmodell verankert, was den trügerischen Eindruck vermittelt, dass man bei Einhaltung der Vorgaben des Vorgehensmodells eigentlich gar keine Probleme bekommen sollte. Trotzdem werden aber die Soll-Prozesse nicht verstanden, weil Sie nicht vernünftig, vollständig und strukturiert dokumentiert sind und somit auch nicht korrekt abgebildet werden. In der Folge kommt es dann im weiteren Projektverlauf zu zahllosen Rework-Szenarien, die am Budget und an den Nerven der Business User und Stakeholder zerren, und die sich sogar bis in den eigentlichen Betrieb der Anwendung hineinziehen („Dieses Problem lösen wir dann in Rework 1…“).

Aber anstatt dieses „iterativ explorative Annähern“ an eine scheinbar brauchbare Lösung als Problem zu identifizieren und nach Auswegen aus dieser Misere zu suchen, haben viele IT-Verantwortliche beschlossen, diese Vorgehensweise mit pseudowissenschaftlichen Ansätzen zu manifestieren und gegen jegliche Kritik abzusichern. Aktuelle Vertreter dieser Taktik sind XP (Extreme Programming) und Scrum. In einem Beitrag zum Thema Scrum beantwortet Ken Schwaber die Frage, warum in Zukunft auf eine ausgedehnte Requirements-Analyse verzichtet werden sollte, wie folgt: „Scrum akzeptiert, dass der Entwicklungsprozess nicht vorherzusehen ist. Das Produkt ist die bestmögliche Software unter Berücksichtigung der Kosten, der Funktionalität, der Zeit und der Qualität“. Prinzipiell ist nach Meinung des Autors Scrum durchaus geeignet, um die Kommunikation innerhalb eines Teams zu verbessern und allen Teammitgliedern ein breiteres Verständnis für den Scope des Projekts und die aktuellen Aufgaben zu erschließen. Betrachtet man aber, wie Scrum in vielen Fällen in der Praxis gelebt wird, dann stellt man nicht selten fest, dass Scrum als scheinbar wissenschaftlich fundierte Begründung für das Unterlassen einer brauchbaren Requirements-Analyse herhalten muss. Es bleibt nur zu hoffen, dass die Ingenieure von Boeing bei der Entwicklung ihrer Flugzeuge nicht derselben Vorgehensweise gefolgt sind…

Nach Ansicht des Autors spielt eine qualitativ hochwertige Analyse, gerade in komplexen Softwareprojekten, eine große Rolle und ist der Schlüssel zum Erfolg - Agilität hin oder her. Hier sollten gerade in komplexen Projekten Strategien aus dem Bereich BPM (Business Process

Management) zum Einsatz kommen. Die dafür benötigten Werkzeuge sind verfügbar und so ist es möglich, von der Analyse über das Systemdesign bis hin zur Implementierung einen prozessorientierten Ansatz zu verwenden, der es allen am Projekt Beteiligten schon in einer frühen Phase erlaubt zu bewerten, ob sich das Projekt auf einem guten Weg befindet oder nicht. Klar ist aber auch, dass ein solcher Ansatz nicht nur die passenden Entwicklungswerkzeuge verlangt, sondern dass die am Projekt beteiligten Akteure in der Lage sein müssen, einen prozessorientierten Ansatz tatsächlich konsequent zu leben.

Ist dieser konsequente Weg der prozessorientierten Analyse einmal eingeschlagen, und werden zur Unterstützung Werkzeuge verwendet, die eine Weiterverwendung der Analyseartefakte in der Design- und Implementierungsphase erlauben, dann hat das unschätzbar wertvolle Vorteile:

- Feedback und Input von den Fachabteilungen zu bekommen wird deutlich einfacher, da diese schon immer in Prozessen denken und erfahrungsgemäß mit einem solchen Ansatz deutlich besser zurechtkommen.
- Missverständnisse („…Das habe ich aber so nicht verstanden…“) und Unvollständigkeiten können konsequent eliminiert werden, weil über die BPM-Werkzeuge ein formales Beschreibungsverfahren verwendet wird, welches keinen Platz für Zweideutigkeiten lässt.
- Technische Details wie z.B. Services, Schnittstellen, Austauschformate etc. können konsequent aus der fachlichen Analyse herausgehalten werden.
- Die Qualität der Analyseartefakte wird deutlich höher sein, als die oben beschriebene „Loseblattsammlung“, weswegen auch das Management deutlich leichter in das Projekt eingebunden werden kann.

Für ein weiterführendes Studium des Themas Prozessmanagement und prozessorientiertes Requirements Management sei der Leser auf http://www.horus.biz verwiesen.

High-End Performance Tuning

Das Dokumentieren und Bereitstellen der gewünschten Funktionalitäten ist jedoch nicht der einzige Stolperstein, der sich im Projektverlauf als kritischer Faktor erweisen kann. Insbesondere in großen Projekten sind nichtfunktionale Anforderungen mindestens ebenso bedeutend. So ist bei einer hochtransaktionalen Datenbankanwendung neben der reinen funktionalen Korrektheit das Erfüllen der Performance Anforderungen von mindestens ebenso großer Bedeutung.
In der Regel beschränken sich die Aktivitäten rund um das Thema Performance Tuning aber leider auf das Erstellen von Trace- und AWR (Automatic Workload Repository-Auswertungen), um anhand kritischer Ausführungspläne Zugriffe zu identifizieren, die durch eine überarbeitete Indizierung verbessert werden können.

Performance Tuning sollte aber schon viel früher beginnen: So ist es z.B. beim Design des Datenmodells zwingend notwendig, die zukünftigen Zugriffsprofile bereits zu kennen, um so ein optimal auf Performance ausgerichtetes Datenmodell zu schneiden. D.h. bereits in einer frühen Teststufe müssen alle relevanten Zugriffe bekannt sein, um so die entsprechenden Anpassungen am Datenmodell vornehmen zu können. Dies ist insbesondere in Umgebungen, in denen Zugriffe aus einem Persistenzframework (z.B. Hibernate) automatisch generiert werden, ein nicht triviales Unterfangen, da die Softwareentwickler in der Regel die Zugriffe gar nicht kennen, die der OR-Mapper generiert. Hier muss rechtzeitig ein SQL-Experte involviert werden, der via Tracing alle Zugriffe auswerten kann. Ferner ist bei Verwendung eines OR-Mappers zu untersuchen, ob alle Zugriffe auf die Datenbank aus fachlicher Sicht wirklich notwendig und angemessen sind. Sehr oft

kommt es vor, dass der OR-Mapper ganze Objektgeflechte auflöst, was in Zugriffen auf diverse Tabellen resultiert, obwohl eigentlich nur in einer Tabelle der Wert einer Spalte für einen Datensatz geändert werden soll. Dieses Verhalten eines OR-Mappers kann sich recht schnell zum KO-Kriterium entwickeln. Typischerweise treten solche Effekte auf, wenn einmal entwickelte Services mehr oder weniger gedankenlos wiederverwendet werden.

Ein weiterer Fehler, welcher insbesondere in hochtransaktionalen Umfeldern immer wieder gemacht wird, besteht darin, dass alle Performance-Überlegungen auf die dialogorientierten Anteile einer Anwendung ausgelegt werden. Batch-Prozesse, die sich des gleichen Datenmodells bedienen, bleiben in den Untersuchungen meist außen vor und führen dann bei Massenverarbeitungen zu großen Problemen. Ganz besonders brisant wird es dann, wenn zur Implementierung von Batch-Abläufen auf Dialoganwendungen ausgelegte OR-Mapper zum Einsatz gebracht werden oder gar bereits für den Dialog entwickelte Services für die Batch Verarbeitung „wiederverwendet“ werden. Nach Meinung des Autors haben sich OR-Mapper insbesondere in der Batch-Verarbeitung bisher nicht bewährt und sollten für diesen Einsatzfall mit der größten Vorsicht verwendet werden. Dies scheint auch ein Grund dafür zu sein, warum viele Unternehmen bei der Massenverarbeitung an PL/SQL oder etwa Cobol festhalten.

Ebenso problematisch ist der Umstand, dass bei der Entwicklung von hochtransaktionalen Anwendungen mitunter vergessen wird, das zugrundeliegende logische Datenmodell in ein wirklich passendes physisches Modell abzubilden. Nur in den wenigsten Fällen machen sich Entwickler und Datenbankadministratoren Gedanken über Themen wie Clusterbildung, Partitionierung oder Denormalisierung, obwohl durch derartige Maßnahme sehr oft spürbare Performance-Gewinne erzielbar sind.

Schließlich werden Tuningmaßnahmen in den seltensten Fällen im Vorfeld des Einsatzes umfassend bewertet und alle Seiteneffekte beleuchtet. So kann z.B. ein durchaus gut gemeinter Index zu katastrophalen Auswirkungen führen, wenn auf die betroffene Tabelle bei den meisten Zugriffen geschrieben wird.

Generell ist zu sagen, dass Performance Tuning kein Thema ist, welches man gegen Ende des Projekts „mal so nebenbei“ durch den Datenbankadministrator bearbeiten lassen kann. Hier sind alle Beteiligten gefordert, bereits im Systemdesign Tuning-Aspekte zu bedenken.

High-End-Verfügbarkeit

Wenn es um das Thema Hochverfügbarkeit geht, werden immer gerne Prozentzahlen für die Verfügbarkeit eines Systems gegeben, ohne diese Zahlen wirklich zu hinterfragen. Man beachte, dass beispielsweise eine Verfügbarkeitsquote von 99,99 Prozent (eine sehr oft und gerne genannte und geforderte Zahl) für ein IT-System bedeutet, dass es im Schnitt pro Jahr maximal 3154 Sekunden = 53 Minuten nicht verfügbar sein darf, womit dann pro Woche noch ein Wartungsfenster von maximal einer Stunde zur Verfügung steht.

Bei der Planung und dem Betrieb eines hochverfügbaren Systems sind die folgenden Aspekte zu beachten:

Bei der Berechnung der Gesamtverfügbarkeit eines Systems dürfen nicht nur die wichtigen Komponenten wie z.B. der Datenbank-Server und der Application Server betrachtet werden. Vielmehr müssen alle Infrastrukturkomponenten vom Client bis zum DB-Server sowie alle integrierten Fremdsysteme in die Betrachtung einbezogen werden. Bei Fremdsystemen ist zu beachten, dass man

sehr oft gar keinen Hebel hat, auf die Verfügbarkeit dieser Systeme einzuwirken. Dem Abnehmer der Leistung ist es aber relativ egal, aus welchen Gründen das Gesamtsystem nicht verfügbar war.
Ferner ist zu berücksichtigen, dass bei der Berechnung der Verfügbarkeit eines Gesamtsystems nicht nur das vermeintlich schwächste Glied berücksichtigt werden darf, sondern dass eine gesamtheitliche Betrachtung aller Komponenten notwendig ist. Wählt man diesen Ansatz, dann befindet man sich recht schnell im Bereich der Wahrscheinlichkeitsrechnung – ein Ansatz, der ob seiner Komplexität auch so gut wie nie verfolgt wird, obwohl es sich dabei um den einzig vernünftigen Weg handelt, über die Verfügbarkeit eines Systems belastbare Aussagen zu machen.

Ebenso selten wird bedacht, dass der Begriff der Verfügbarkeit kein binärer Begriff ist, sondern eigentlich über einen SLA (Service Level Agreement) klar definiert werden muss, ab welchen Leistungsmerkmalen ein Dienst bzw. System verfügbar ist oder nicht. So ist rein technisch der Dienst einer Online-Kontostandsauskunft verfügbar, wenn die Antwort auf die Anfrage nach 10 Minuten mit dem korrekten Ergebnis beim Client eintrifft. Aber ist dieser Dienst deshalb als verfügbar einzustufen?

Eine weitere wichtige Differenzierung des Begriffs Verfügbarkeit ergibt sich aus der einfachen Tatsache, dass es Dienste innerhalb der Gesamtanwendung gibt, die hochverfügbar sein müssen. Aber sicher auch andere Dienste, für die eine geringere Verfügbarkeit durchaus akzeptabel ist. Bezogen auf das vorige Beispiel könnten man sagen, die Funktion „Geld an einem EC-Automaten abheben“ hat sicher eine höhere Priorität als die Funktion „Kontoauszug drucken“ und sollte auch so in die Betrachtung der Verfügbarkeit einfließen.

Bei der Planung der zugrundeliegenden Architektur werden oftmals unpassende Technologien ausgewählt, die scheinbar zu einer Verbesserung der Gesamtverfügbarkeit des Systems führen, im Zweifelsfall aber eher kontraproduktiv wirken. So ist beispielsweise nach wie vor die Meinung weit verbreitet, dass zwei via Streams-Replikation verbundene Datenbankinstanzen zur Verbesserung der Verfügbarkeit beitragen, weil bei Ausfall einer Instanz ja auf die andere Instanz umgeschaltet werden kann. Dass es sich dabei um einen fatalen Irrtum handelt, wird dann meistens in Produktion festgestellt, wenn die Architektur nicht mehr zu ändern ist. Ferner ist bei der Planung einer hochverfügbaren Architektur zu bedenken, dass jede Komponente zu einem beliebigen Zeitpunkt vom Netz genommen werden können muss, ohne die Gesamtleistung oder die Funktionalität des Systems zu beeinträchtigen.

Die Anforderung nach einem hochverfügbaren System hat aber durchaus auch Auswirkungen in den Bereich der Anwendungsentwicklung hinein. So muss es z.B. möglich sein, bestimmte Services im Notfall zu „black-listen“, d.h. es muss möglich sein, die Ausführung der Services für eine bestimmte Zeitdauer zu unterbinden, ohne damit allerdings die Funktionalität des Gesamtsystems zu beeinträchtigen oder dadurch gar inkonsistente Datenstände zu riskieren. In diesem Zusammenhang spielt das Thema Wiederanlauffähigkeit eine große Rolle.

High-End Testing

Hinreichendes Testen ist die Grundvoraussetzung schlechthin für den performanten und hochverfügbaren Betrieb einer Anwendung. Diese banale Aussage wird niemand wirklich bestreiten. Trotzdem kann man sogar in hochkritischen Umgebungen immer wieder beobachten, dass schwerwiegende Fehler „erst“ in Produktion entdeckt werden und dann unter massivem Aufwand korrigiert werden müssen. Die Probleme mit Bezug auf das Thema Testing sind in der Regel sehr vielschichtig:

Sehr oft kommt es vor, dass Testing in der Projektplanung an das Ende des Projekts gelegt wird, in der Regel vor die Pilotphase, womit dann keine oder zu wenig Zeit vorhanden ist, identifizierte Probleme wirklich nachhaltig zu beseitigen. Viele Betreiber hochkritischer Anwendungen sind daher mittlerweile dazu übergegangen, Testaktivitäten bereist fest in den Entwicklungsprozess zu integrieren. Typischerweise so, dass zu jeder geänderten oder neu zu entwickelnden Funktionalität parallel zur Entwicklung auch ein oder mehrere Test Cases erstellt werden müssen, die dann regelmäßig noch vor Beginn der eigentlichen Tests in sogenannten Continuous Integration Tests abgefahren werden. So besteht die Möglichkeit, Fehler und Probleme frühzeitig zu erkennen.
Ein weiterer Kardinalsfehler besteht darin, die Testdurchführung den Entwicklern zu überlassen. Eine Vorgehensweise, die grundsätzlich zu Problemen führen wird. Hier kann nur sehr eindringlich geraten werden, die Projektressourcen strikt zwischen Entwicklung und Test zu trennen.

Sehr wichtig ist auch der Testumfang, also die Festlegung, was denn genau getestet wird. In der Regel werden die im Rahmen eines Projekts selbst entwickelten Funktionalitäten stichprobenartig getestet - von einem vollumfänglichen Test ist man meistens weit entfernt. Die wenigsten Projekte kommen darüber hinaus auf den Gedanken, die verwendeten Infrastrukturkomponenten, die meist von Drittanbietern hinzugekauft werden, vorab im eigenen Umfeld zu testen. Nicht selten erlebt man dann böse Überraschungen, wenn die neue Version der verwendeten Datenbank-Software mit der eigenen Anwendung deutlich schlechter zurande kommt als die Vorgängerversion. Gerade beim Austausch einer Infrastrukturkomponente ist es aber von größter Bedeutung, die Reife der Komponente in Bezug auf den Betrieb der eigenen Anwendung anhand umfangreicher Tests zu bewerten.

Ein Umstand, der in sehr vielen Projekten zu größten Problemen führt, ist die Ausstattung der Testumfelder: Gemeint ist hier die Ausstattung mit Ressourcen und mit Testdaten. Befragt man einen Entwickler, wie denn das Testsystem seiner Träume dimensioniert werden sollte, dann wird dieser in der Regel antworten: „So wie das Produktionsumfeld“. Leider ist diese Forderung in den meisten Projekten aus finanziellen Gründen nicht darstellbar, normalerweise wird an der Ausstattung der Testumfelder sogar deutlich gespart. Hier kann sich eine auf Virtualisierung basierende Testumgebung als großer Vorteil erweisen. In einer solchen Umgebung ist es möglich, die rein funktionalen Tests mit reduzierter Hardware-Ausstattung durchzuführen, wohingegen Performance- und Lasttests zeitlich begrenzt dann mit produktionsnahen Ressourcen abgefahren werden können, welche der virtuellen Umgebung temporär zugeordnet werden.

Betrachtet man die Ausstattung der Testsysteme mit Testdaten, so begegnet man leider auch bei kritischen Anwendungen sehr häufig den gleichen Problemen wie bei kleinen und mittleren Anwendungen: Das Volumen der Testdaten entspricht nicht einmal annähernd den Datenvolumina in Produktion. Und die Datenverteilung, welche im Hinblick auf die Performance von Datenbank-Zugriffen eine entscheidende Rolle spielt, stimmt ebenfalls nicht mit Produktion überein. Betrachtet man die Verfahren, wie Testumgebungen heute mit Daten bestückt werden, dann findet man in der Regel zwei Verfahren: In manchen Fällen wird in regelmäßigen Abständen ein Abzug aus der Produktion in den Testumfeldern eingespielt. Diesem Verfahren ist aus Sicht des Autors der absolute Vorzug zu geben. Eine Variante besteht darin, dass man nur eine Teilmenge der Produktionsdaten auf das Testsystem abzieht, wodurch die Aussagen die auf Basis eines Tests über Performance und Stabilität gemacht werden aber bereit deutlich verwässert werden. Das zweite Verfahren besteht in der Erstellung von synthetischen Testdaten über Testdatentreiber. Mit diesem Verfahren können zwar innerhalb kurzer Zeit sehr große Mengen an Testdaten erstellt werden; trotzdem ist das Verfahren sehr fragwürdig, da in der Regel keine produktionsnahen Datenverteilungen erstellt werden. Ignoriert man diesen Sachverhalt, dann ist es durchaus wahrscheinlich, dass sich die Anwendung im Testumfeld, insbesondere bzgl. der Performance, komplett abweichend vom Produktionsumfeld verhält. Mit

anderen Worten: Das Erstellen sinnvoller synthetischer Testdatenbestände will sorgsam geplant und durchdacht sein.

In den meisten Fällen, in denen synthetische Testdaten zum Einsatz kommen, liegt dies daran, dass (gesetzliche) Vorgaben die Übernahme von Produktionsdaten in die Testumfelder, zu denen in der Regel alle Entwickler uneingeschränkt Zugang haben, verbieten. Dies gilt insbesondere im Finanzdienstleistungsumfeld. Weithin unbekannt ist in diesen Fällen der Umstand, dass alle namhaften Datenbankhersteller sogenannte Anonymisierungswerkzeuge anbieten, mit denen gesamte Produktionsdatenbestände automatisiert anonymisiert werden können. Der Vorteil dieses Verfahren liegt klar auf der Hand: Es besteht hier die Möglichkeit, große Datenmengen innerhalb kurzer Zeit bereitzustellen und man hat darüber hinaus noch den Vorteil, dass die anonymisierten Datenbestände die gleichen Verteilungen aufweisen wie die Produktionsbestände. Der Nachteil der Anonymisierung besteht darin, dass das Verfahren in der Bereitstellung sehr aufwendig ist, weil für das gesamte zugrunde liegende Datenmodell Anonymisierungsregeln definiert werden müssen und bei einer Anpassung des Datenmodells auch zu pflegen sind. (Der interessierte Leser sei an dieser Stelle auf das Produkt Oracle Data Masking verwiesen, welches ein Bestandteil des Enterprise Managers ist.) Ein weiterer Vorteil dieser Methode besteht darin, dass dieses Verfahren der Testdatenerstellung von verschiedensten Stellen offiziell zertifiziert ist und somit einer Prüfung durch die eigene Revision und / oder externen Prüfstellen standhalten sollte.

Kontaktadresse:

Dipl.-Inform. Sebastian Graf
PROMATIS software GmbH
Pforzheimer Strasse 160
D-76275 Ettlingen

Telefon:	+49 (0) 7243-2179-0
Fax:	+49 (0) 7243-2179-99
E-Mail	sebastian.graf@promatis.de
	sebastian.graf@doag.org
Internet:	http://www.promatis.de
	http://www.horus.biz