QA Navigation GmbH

Hansjakobstr. 1
D - 81673 München

Telefon: +49 (0)89 / 43 57 36 11
Telefax: +49 (0)89 / 43 57 36 12
Web: www.qa–navigation.com

SOA testen - Der inkrementelle fachliche Integrations-Test (IFIT)

Ronald D. Grindle

1. Zusammenfassung

Komplexe, verteilte Systeme sind eine Herausforderung an die Planung und Durchführung von fachlchen Tests. Mit der Einführung von SOA wachsen die Herausforderungen, da mit SOA das bisherige Schema der synchronen Lieferung von Modulen verlassen wird. Der inkrementelle fachliche Integrations-Test ist eine Strategie, mit der der fachliche Test strukturiert diesen Anforderungen begegnet und dabei die Eigenschaften von SOA-Lösungen zu seinen Gunsten nutzt.

Vorteile

- Frühzeitige Durchführung der ersten Tests, damit frühzeitige Rückmeldung über die Qualität
- Individuelles, risikobasiertes Testen von Einzelkomponenten
- Entschärfung der Testfallexplosion
- Schnellere und bessere Lokalisierung von Fehlern
- Testfortschritt im Takt des Fortschritts der Entwicklung
- Tests auf Schnittstellenebene eröffnen Möglichkeiten zur Testautomation (Regression-Test, daily build etc.)
- Regressions-Tests von Anfang an
- Drastisch verkürzte Nachlaufzeit des Tests und damit Verkürzung der Laufzeit des Gesamtprojekts

Voraussetzungen

- Berücksichtigung der Vorgehensweise des Tests schon beim fachlichen und beim technischen Design (Design for Test)
- Schnitt der fachlichen Spezifikation entspricht dem der Software-Module

- Frühzeitige Offenlegung der Beschreibung der Schnittstellen und der Datenstrukturen durch die Entwicklung
- Zugriff des Tests zu Schnittstellen (im Testsystem)
- Qualifizierung der Tester
- Geeignete Infrastruktur
- Entwicklung von Testtreibern zur Prüfung der einzelnen Module
- Entwicklung von Test-Stubs zur Prüfung der Schnittstellen
- Geeignete Organisation des Tests

2. Der fachliche Test und die Herausforderungen durch SOA

Bei konventionellen, d.h. monolithischen Anwendungen, ist der fachliche Test es gewohnt, die Konformität von Anwendungen mit den fachlichen Anforderungen mit Hilfe der vorhandenen Benutzerschnittstellen zu testen, als End-To-End-Test. Mit SOA löst sich die Einheitlichkeit von Anwendungen zu Gunsten eines Konglomerats an Modulen auf, wobei die Benutzerschnittstelle nur eine weitere Komponente ist.

Werden die Möglichkeiten von SOA genutzt, lösen sich ebenfalls die traditionellen Entwicklungszyklen auf, bei denen Anwendungen als Ganzes entwickelt und abgeliefert werden. Es entsteht ein kontinuierlicher Fluss von Releases einzelner Services.

Die Einführung von SOA stellt den Test aber nicht nur vor neue Herausforderungen, sondern bietet auch Chancen, den Test zu vereinfachen und zu beschleunigen. Auch die fachlichen Tester stehen vor neuen Herausforderungen. Wurden herkömmliche Systeme mittels der Benutzerschnittstelle getestet, so wird bei SOA der fachliche Test an Hand von technischen Schnittstellen durchgeführt. Dies mag für den einen oder anderen Tester ungewohnt sein, ist aber mit dem Paradigmenwechsel, den der Wechsel zu SOA mit sich bringt, unumgänglich.

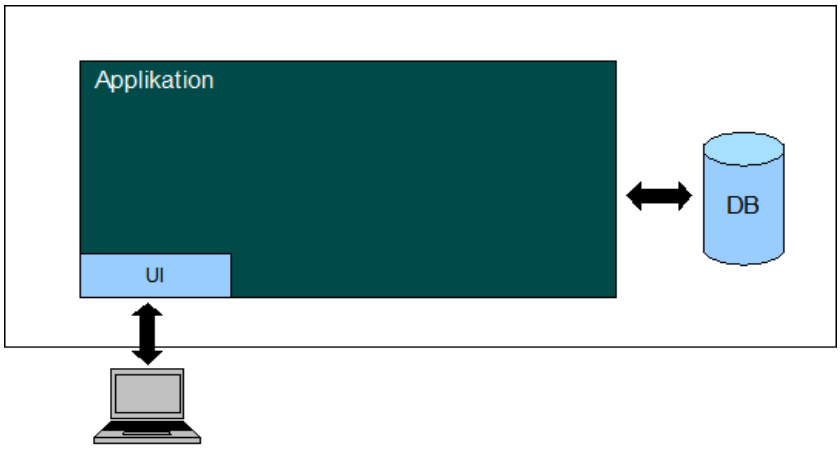

Abbildung 1: Monolithisches System

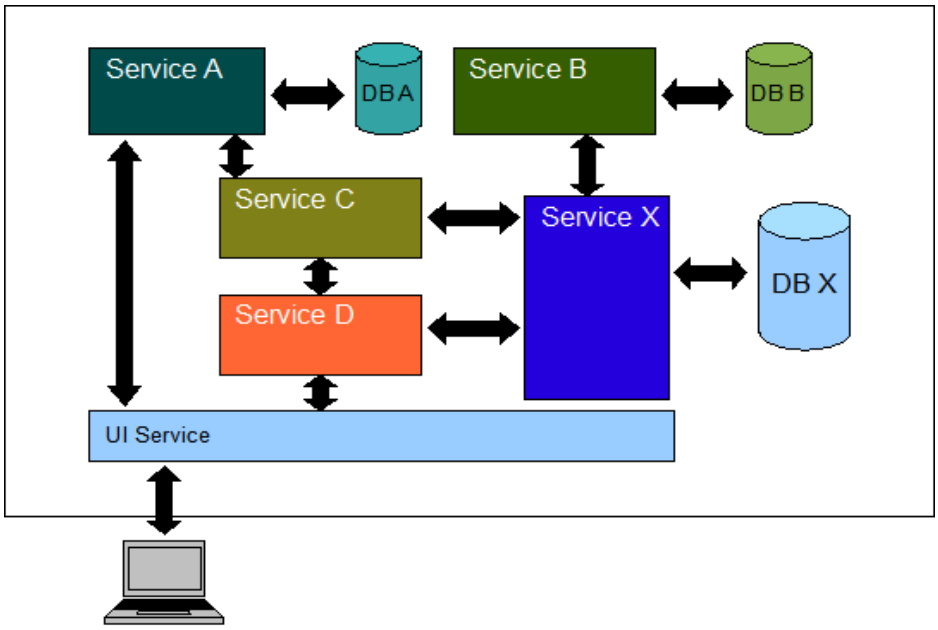

Abbildung 2: SOA System

3. Der inkrementelle fachliche Integrations-Test

Testfallexplosion ist das Phänomen, das sich einstellt, wenn versucht wird, an Hand der Schnittstellen und der Kombinatorik der dazugehörigen Parameter die Zahl der für eine 100%ige Testabdeckung benötigten Testfälle zu ermitteln. Die Zahlen wachsen pro Schnittstelle mit den Parametern exponentiell. Mit den Anforderungen wachsen die Schnittstellen und ihre Parameter und führen so schnell zu astronomischen Zahlen.

Mit der Einführung von SOA verschärft sich das Problem. Mit der Trennung der Services wachsen die Schnittstellen, gleichzeitig ermöglicht die neue Technik neue Kombinationen von Modulen zu neuen Lösungen. Dies lässt die Zahl der notwendigen Tests, will man End-to-End testen, rapide wachsen. Werden auch noch verschiedene Versionen gleichzeitig betrieben, vervielfacht sich die Zahl der notwendigen Tests entsprechend.

Durch die Aufteilung ehemals monolithischer Anwendungen in Services und die damit einhergehende Offenlegung von vormals internen Schnittstellen bietet sich aber auch die Chance, die Situation für den Test zu erleichtern: es wird möglich, den inkrementellen fachlichen Integrations-Test anzuwenden.

4. Was ist der inkrementelle fachliche Integrations-Test?

Der inkrementelle fachliche Integrations-Test (der Einfachheit halber im weiteren mit IFIT abgekürzt) ist eine Strategie zur strukturierten Herangehensweise des Tests an die Herausforderung eines komplexen, verteilten Systems mit dem Ziel, es fachlich so zu testen, dass es zeitnah und in ausreichender Tiefe geprüft werden kann. Der IFIT stellt dabei keine völlig neue Testtechnik dar. Vielmehr werden bekannte und etablierte Testtechniken in einer neuen Weise kombiniert und angewendet. Ein Aspekt dabei ist, die Strategie der iterativen Tests der Entwicklung auf der Ebene der Unit-Tests bei den fachlichen Tests anzuwenden.

Mit der Auflösung des Software-Produkts als einheitliches Ganzes löst sich auch die gleichzeitige Lieferung der Software-Module auf. Im Verlauf der Entwicklung werden die fertigen Module Stück für Stück geliefert, oftmals ohne direkten Zusammenhang. Um solche Lieferungen zeitnah testen zu können, ist es notwendig, die herkömmliche Vorgehensweise, das System in seiner Vollständigkeit zu testen, zu verlassen und statt dessen einen Weg zu finden, im Rhythmus der Einzellieferungen zu testen und dennoch eine belastbare Testaussage zu erhalten.

4.1. Die Vorgehensweise

Fertige Module, die bereits die technischen Tests und die Entwickler-Tests bestanden haben, werden gegen die fachlichen Anforderungen getestet. Die Schnittstellen jedes Moduls werden separat geprüft, wobei technische Kompatibilität nicht ausreicht. Die Schnittstellen müssen vor allem inhaltlich abgestimmt sein.

Die Module werden mit der Hilfe von Test-Stubs geprüft. Diese Programme werden sowohl zur Prüfung der Schnittstellen als auch für die Prüfung auf fachlicher Ebene herangezogen. Die Richtigkeit des Ergebnisses wird an Hand der Rückmeldungen des Moduls festgestellt. Die folgende Abbildung illustriert das Prinzip:

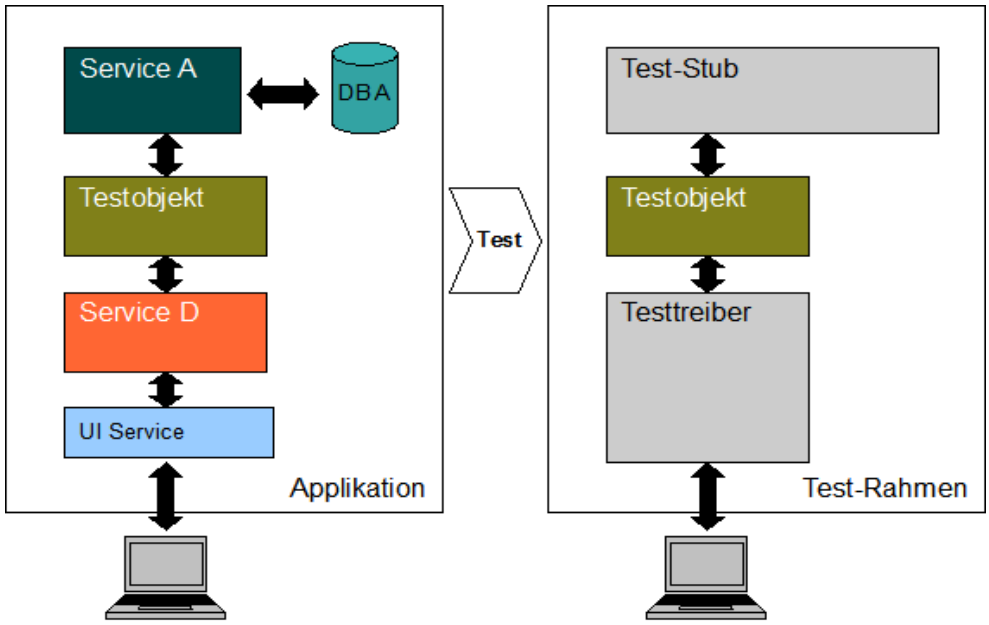

Abbildung 3: Testtreiber und Test-Stub

4.2. Inkrementeller Test

Sind zwei benachbarte Services erstellt und erfolgreich getestet, so wird zunächst die Kompatibilität der Schnittstellen geprüft. Hier genügt es festzustellen, ob die Schnittstellen zwischen den Test-Stubs fachlich wie technisch übereinstimmen.

Wie in Abbildung 4 dargestellt, kann nun mit Hilfe der zwei geprüften Module das erste Inkrement gebildet werden. Da beide Module nicht nur ihre technische, sondern auch ihre fachliche Korrektheit bewiesen haben, können sich die weiteren Tests auf Stichproben und spezielle Tests, die sich nur mit den kombinierten Modulen durchführen lassen, beschränken.

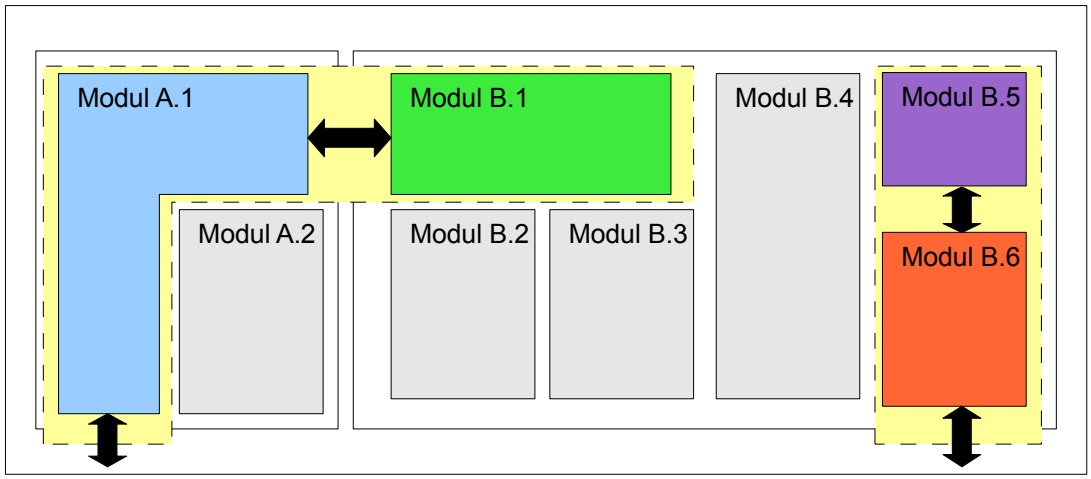

Abbildung 4: Modul-Modell

Mit jedem weiteren Modul wächst das Inkrement und mit jedem erfolgreich getesteten Modul wächst die Gesamtanwendung zu einem Ganzen zusammen. Selbst wenn ein Modul noch nicht alle Funktionalitäten aufweist, können die vorhandenen Funktionalitäten trotzdem getestet werden. Es werden damit nicht nur die Module inkrementell, sondern auch die Funktionen mit zunehmendem Fortschritt getestet.

Bei großen Projekten können sich durchaus Modul-Inseln ausbilden. Es lassen sich aber bereits Fehler erkennen, noch bevor die Anwendung als Ganzes zur Verfügung steht. Solange ein Inkrement sich weder an seinen Schnittstellen noch in seiner Funktionalität ändert, besteht keine Notwendigkeit, es erneut zu testen. Wird ein neues Release eines Moduls geliefert, so durchläuft es den Zyklus von fachlichem Modul-Test, Prüfung der Schnittstellen für die geänderten Teile und Stichproben-Tests des Inkrements erneut.

4.3. Prüfung der Schnittstellen

Inkompatibilität von Schnittstellen sind bei SOA-Anwendungen ein häufig auftretendes und gleichzeitig zeitraubendes Problem. Es empfiehlt sich daher, die Schnittstellen zwischen den Modulen (intern) und den Services frühzeitig zu prüfen. Die Überprüfung muss nicht notwendigerweise am implementierten System stattfinden, sie kann in einer ersten Stufe als Schreibtisch-Test vonstatten gehen. Bei einem solchen Test führen die Vertreter der Module die Prüfung „auf dem Papier“ durch. Ein tatsächlicher operativer Test kann damit aber nicht ersetzt werden. Der Schreibtisch-Test ist dennoch nützlich, da er nicht nur technische Missverständnisse auszuräumen hilft. Man spart sich den Aufwand, vorhandene fachliche Fehlinterpretationen zu implementieren, später im Test darauf zu stoßen und erst dann mit einem Fix zu beheben. Sind die technischen Voraussetzungen erfüllt, lassen sich diese Tests auch automatisiert durchführen.

Vor allem bei Anwendungen, die sich aus Services verschiedener Lieferanten zusammensetzen, empfiehlt es sich, wechselseitig noch vor der Implementierung der fachlichen Anforderungen, die Schnittstellen der Services mit Test-Stubs, die die Verarbeitung des Services simulieren, zu überprüfen, um den korrekten Austausch der Daten sicher zu stellen. Der Erfahrung nach sind inkompatible Schnittstellen eines der größten Hemmnisse im Testfortschritt. Investitionen in den Schnittstellen-Test lohnen sich zweifelsohne.

4.4. Automatisierter Regressions-Test

Unvollständig gelieferte Module zu testen stößt in der Regel bei der Testorganisation auf Widerstand. Den zusätzlichen Aufwand bereits durchgeführter Tests bei vollständiger Lieferung erneut testen zu müssen, um sicherzustellen, dass die erreichte Qualität nicht verloren ging, möchte man möglichst vermeiden.

Da die Testtreiber und die korrekten Ergebnisse bereits vorliegen, können diese Prüfungen leicht automatisiert durchgeführt werden. Damit relativiert sich das Argument des zusätzlichen Aufwands.

5. Vorteile des inkrementellen fachlichen Tests

5.1. Frühzeitige Durchführung der ersten Tests

Da die Module und die gelieferte Funktionalität im Einzelnen getestet werden, ist es möglich, dies direkt bei Lieferung der Module zu tun, ohne auf weitere Modul-Lieferungen zu warten. Dies ermöglicht es, frühzeitig möglichen Fehlerentwicklungen entgegen zu steuern.

Hochkritische Komponenten können bei geeigneter Steuerung der Entwicklung vorgezogen und ihre Qualität vorab geprüft werden. Im schlimmsten Fall erweist sich das gelieferte Werk als völlig untauglich. Dann ist zumindest ein Schutz vor weiteren Fehlinvestitionen erreicht.

5.2. Individuelles, risikobasiertes Testen von Einzelkomponenten

Die Vorgehensweise des IFITs eröffnet die Möglichkeit, jedes Modul basierend auf der individuellen Einschätzung seines Risikos zu testen, also unterschiedlich intensiv gemäß seiner Rolle und dem ihm zugeschriebenen Risiko. Damit wird eine optimale Balance zwischen Testaufwand und seinem Nutzen möglich.

Im Vergleich zum herkömmlichen Test einer kompletten Anwendung an den Benutzerschnittstellen vereinfacht sich die Testplanung und Ausführung. Da die fachlichen Funktionen nunmehr direkt am zuständigen Modul geprüft werden können, fallen komplizierte Testsequenzen zur Vorbereitung eines Testfalls weg. Das Ergebnis wird schneller ermittelt, eine Wiederholung des Testfalls ist weniger aufwändig.

Ebenso entfällt die Notwendigkeit der Prüfung einzelner fachlicher Funktionen in fixer Reihenfolge, bei der der vorangegangene Test die Vorbereitung des anderen sicherstellt. Durch den eigenen Test-Stub verfügt jeder Test über seine eigene Vorbereitung.

5.3. Entschärfung der Testfallexplosion

Da die Software nicht mehr als Monolith über die zentrale Eingangsschnittstelle geprüft werden muss, ist es auch nicht mehr notwendig, die Funktionalität der Gesamtanwendung mit Hilfe von Kombinationen der Eingangs-Parameter zu prüfen. Durch das Zerlegen des Prüfobjekts kann die Zahl der notwendigen Kombinationen von Testparametern reduziert werden. Trotzdem wird eine zufriedenstellende Testabdeckung in Bezug auf die fachlichen Anforderungen erreicht.

5.4. Schnellere und bessere Lokalisierung von Fehlern

Mit der fachlichen Prüfung der Module des Services selbst ergibt sich eine schnellere Lokalisierung von fachlichen Fehlern. Die Analyse und Behebung eines Fehlers werden durch die Möglichkeit, das identifizierte Problem an Hand des Testtreibers nachzuvollziehen, erheblich erleichtert und Ihre Bearbeitung somit beschleunigt. Die Qualität der Fehlerbehebungen steigt, das heißt die Rate der fehlerhaften Behebungen sinkt, da durch den Test-Treiber ein eindeutiges Referenzmodell vorliegt. Auch der Nachtest gestaltet sich einfacher und damit schneller.

5.5. Testfortschritt im Takt des Fortschritts der Entwicklung

Statt das Ende der Entwicklung abwarten zu müssen, kann bereits gelieferte Funktionalität frühestmöglich fachlich geprüft werden. Es wird möglich, den Fortschritt des Tests im Takt der Lieferungen der Entwicklung zu halten.

Da die Module und Services mit Hilfe von Testtreibern und Test-Stubs unabhängig voneinander geprüft werden, halten fehlende oder fehlerhafte Glieder in der Verarbeitungskette den Testfortschritt nicht mehr auf. Der Test kann mit anderen, funktionierenden Modulen fortgesetzt werden.

Beide Faktoren sorgen für eine deutlich beschleunigte Rückmeldung zur Qualität des Produkts und damit für eine schnellere Reifung der Applikation.

5.6. Test auf Schnittstellenebene eröffnet Möglichkeiten zur Testautomation

Da die Schnittstellen zu den fachlichen Funktionen bei SOA in der Regel technische Schnittstellen sind, ergeben sich spürbare Vorteile für die Testautomation. Nachdem die Testtreiber und die korrekten Ergebnisse bereits vorliegen, ist dafür keine Entwicklungsarbeit mehr zu leisten. Damit ist es nur noch ein kleiner Schritt zur automatisierten Durchführung von Tests für Daily Builds.

5.7. Regressions-Tests von Anfang an

Sobald ein Modul die fachliche Prüfung bestanden hat, können mit jeder Lieferung die bereits positiv getesteten Module und ihre Inkremente einem Regressions-Test zugeführt werden. Da man aus den ersten Testläufen bereits Referenzdaten zur Prüfung der Module erhalten hat, bietet es sich an, diese Regressions-Tests automatisiert durchzuführen, und zwar zu 100%. Damit wird die erreichte Qualität gewahrt und eventuelle Rückschritte durch Fehlentwicklung zeitnah erkannt.

5.8. Verkürzte Nachlaufzeit des Tests

Da beim IFIT die fachliche Funktionalität im Einzelnen bereits mit der ersten Lieferung geprüft wird und die Qualität der Gesamtanwendung nur noch in Stichproben geprüft werden muss, verkürzt sich der Nachlauf des Tests wesentlich gegenüber der sequentiellen Vorgehensweise. Dadurch wird wertvolle Zeit gewonnen, die Laufzeit des Gesamtprojekts verkürzt sich, besetzte Ressourcen können freigegeben und die getätigte Investition früher genutzt werden. Die beiden nachfolgenden Abbildungen illustrieren den Zeitgewinn, den das Verfahren ermöglicht.

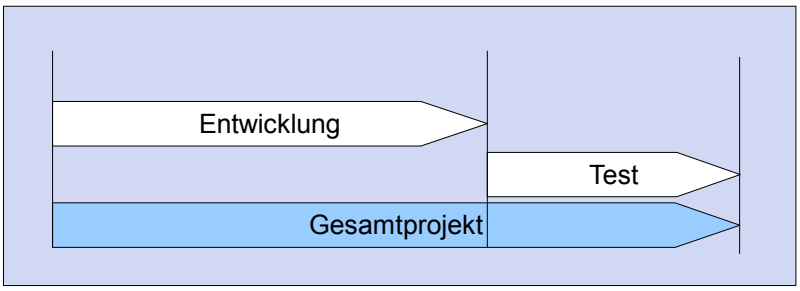

Abbildung 5: Wasserfall-Methode

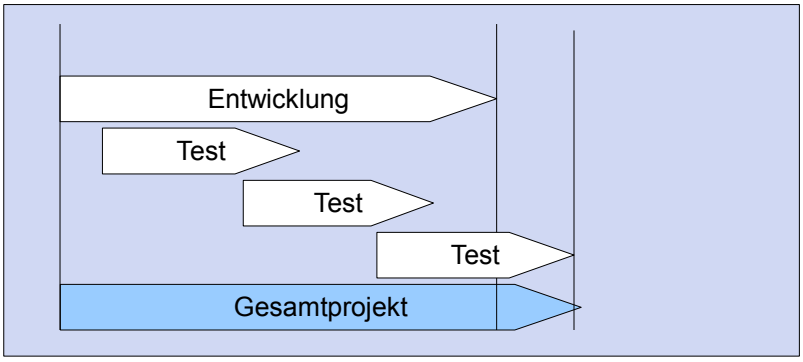

Abbildung 6: Inkrementelle Methode

Ein weiterer Vorteil liegt in der Möglichkeit, Show-Stopper, die am Ende der Verarbeitungskette liegen, frühzeitig zu erkennen. Wird konventionell dem Wasserfall-Modell folgend am Ende der Entwicklung End-to-End getestet, können Probleme in den späten Phasen der Verarbeitung erst erkannt werden, wenn alle auf dem Pfad liegenden Module vorliegen, korrekt arbeiten und der Test bis dahin vorgedrungen ist. Im Extremfall führt dies dazu, dass erst kurz vor Ende des Projekts Probleme erkannt werden, die ein „Go Live" verzögern oder sogar verhindern.

6. Voraussetzungen

6.1. Design for Test

Um die gelieferten Module an ihren Schnittstellen testen zu können, muss die Teilung der Module der fachlichen Struktur folgen. Dies ergibt sich selten von selbst und muss daher bei der Entwicklung eingefordert werden. Die Teilung der Module nach fachlichen Gesichtspunkten ist aber eine entscheidende Voraussetzung, um nach dem IFIT vorgehen zu können und ist gleichzeitig eine der wesentlichen, notwendigen Änderungen gegenüber der Vorgehensweise der bisherigen Software-Entwicklung.

Der Schnitt der Module nach fachlichen Gesichtspunkten ist allerdings keine Forderung, die nur der Testbarkeit dient. Aus Sicht der Wartbarkeit und der Fähigkeit, sich weiter zu entwickeln, ist diese Forderung ebenfalls sinnvoll.

Um die Module einzeln testen zu können, müssen diese eigenständig lauffähig sein und nur in begründeten Ausnahmefällen weitere Module für die Verarbeitung benötigen. Auch bei der

Proportionierung der Module sollte dem Test ein Mitspracherecht eingeräumt werden, um die Entwicklung allzu funktionsmächtiger, aber auch zu detaillierter Module zu vermeiden.

6.2. Dokumentation der Schnittstellen

Jenseits der technischen Hürden dürfen keine fachlichen Hürden zum Verständnis der Schnittstellen aufgebaut werden. Um interpretierbar zu sein, müssen die Parameter der Schnittstellen der fachlichen Funktion folgen und als solche erkennbar und dokumentiert sein. Der Transport von technischen Daten und internen Zwischenergebnissen zusammen mit den Verarbeitungsdaten der Module an den Service-Schnittstellen ist möglichst zu vermeiden bzw. auf anderen Wegen zu transportieren, da sie sonst die Interpretation der Daten durch die fachlichen Tester unnötig erschweren.

Die Dokumentation der Schnittstellen ist zum frühest möglichen Zeitpunkt, also noch vor Beginn der eigentlichen Entwicklung, an den Test weiter zu reichen, damit dort die Tests vorbereitet werden können. Das setzt voraus, dass die Schnittstellen schon vor Beginn der Kodierungsphase festgelegt werden. Auch dies bedeutet eine Änderung gegenüber der Vorgehensweise der bisherigen Software-Entwicklung.

Danach sollten die Schnittstellen nur noch geringen Änderungen unterzogen werden, da sonst dem Test die Basis seiner Entwicklungen entzogen wird.

6.3. Qualifizierung der Tester

Fachliche Tests werden üblicherweise mit Hilfe der Benutzerschnittstelle durchgeführt. Dies hat den Vorteil, dass fachliche Spezialisten und Sachbearbeiter als Tester eingesetzt werden können, wobei diese fachlichen Spezialisten keine besonderen technischen Fähigkeiten mitbringen müssen.

Mit dem Paradigmenwechsel zur Prüfung an den Schnittstellen stellen sich an das Testteam neue Herausforderungen. Technische Schnittstellen sind ihrer Natur nach weniger benutzerfreundlich als Benutzerschnittstellen, ihre Bedienung ist unter Umständen weniger intuitiv. Die Bedienung der Schnittstellen setzt in der Regel einen geübten Umgang mit systemnahen Benutzerschnittstellen (wie z.B. Kommandozeilen-Shells) voraus. Die Schnittstellen sind in der Regel nur per Programm erreichbar, eine gewisse Programmiererfahrung ist also von Nöten.

Trotzdem können rein fachliche Spezialisten und Sachbearbeiter in den Test eingebunden werden. Dafür müssen allerdings die Testtreiber mit ihren Parametern und die Ergebnisse der Tests so aufbereitet werden, dass die fachlichen Spezialisten und Sachbearbeiter in der Lage sind, diese zu bedienen bzw. zu interpretieren. Bei SOA Anwendungen ist diese Hürde nicht allzu hoch, da die Module typischerweise mit Hilfe von XML Datensätzen kommunizieren, die als Text vorliegen und per XSLT leicht in eine lesbare Form gebracht werden können.

6.4. Infrastruktur

Da nicht mehr nur mit Hilfe der Benutzerschnittstelle getestet wird, muss auch die Infrastruktur des Tests den neuen Anforderungen angepasst werden. Um die Schnittstellen der Testsysteme zu erreichen, sind Netzwerkverbindungen mit den entsprechenden Firewall-Freischaltungen, Zugang zu den Betriebssystemen etc. notwendig. Da die Einrichtungen unter Umständen langwierig sein können, ist eine rechtzeitige Planung und Beauftragung notwendig.

Die Zahl und Art der Test-Tools ändert sich. Für den Test von SOA Services gibt es mittlerweile eine Auswahl an Testtools, allerdings in unterschiedlichen Qualitäten. Mit dem Wechsel zu

SOA-Tests steht möglicherweise ein Wechsel des Testtools bzw. eine Heterogenisierung der Testplattform bevor.

6.5. Testtreiber zur Prüfung der einzelnen Module

Wie bereits beschrieben, ist es notwendig, Testtreiber zur Prüfung der Module zu erstellen. Die verfügbaren SOA-Test-Tools unterstützen und erleichtern zwar die Entwicklung von Testtreibern, in einem gewissen Umfang ist aber immer etwas Programmierung gefordert. Wo die Standard-Test-Tools nicht die notwendige Funktionalität bereit stellen, wird es notwendig sein, eigene Testtreiber zu entwickeln.

6.6. Test-Stubs zur Prüfung der Schnittstellen

Zur Simulation von Rückmeldungen von Software-Modulen werden Test-Stubs entwickelt. Neben der Nutzung im Testrahmen dienen die Test-Stubs auch der Prüfung der Schnittstellen.

Für die Planung und Vorbereitung der Tests müssen die Schnittstellen zwischen den Modulen zu Beginn der Entwicklung festgelegt werden und die technische Beschreibung vorab von der Entwicklung offen gelegt werden.

6.7. Organisation des Tests

Mit der Kopplung der Testdurchführung an die Lieferungen der Entwicklung wird es notwendig, die Planung und Entwicklung der einzelnen Tests an der Planung der Entwicklung auszurichten. Da die tatsächliche Ausführung der Entwicklung oftmals vom Plan abweicht, ist eine entsprechende Flexibilität und Reaktionsfähigkeit des Tests gefordert. Der Test muss von Projektbeginn an in das Gesamtprojekt eingebunden werden, um in der Lage zu sein, mit dem Fortschritt der Spezifikationen und dem Design der Anwendung Schritt zu halten, da der Test sich seinerseits mit der Erstellung der Testfälle und dem Aufbau der Testinfrastruktur auf seine Tätigkeit vorbereiten muss.

Die engere Verzahnung des Tests mit der Entwicklung setzt eine engere Kommunikation zwischen Test und Entwicklung voraus. Der Entwicklungsplan muss dem Test frühzeitig übergeben werden, Änderungen des Plans und der Termine sind zeitnah dem Test mitzuteilen. Im Gegenzug muss die Entwicklung die Verarbeitung der Rückmeldungen des Tests, die nun während des Entwicklungszyklus eintreffen, einplanen.

Test-Tool, Fehlermeldungs-Tool, und Liefer-Management-Tool sollten die enge Zusammenarbeit der einzelnen Gruppen unterstützen und nicht behindern. Das Test-Tool, wie auch das Fehler-Reporting sollten die Möglichkeit bieten, die verwendeten Test-Stubs an die Dokumente anzuhängen oder mit ihnen zu verknüpfen, damit die Arbeit der Tester und der Entwickler auf derselben Information bei der Meldung, der Behebung und beim Re-Test eines Problems basiert. Eine enge Verknüpfung von Fehler-, Test- und Lieferinformationen hilft Versions-Wirrwarr zu vermeiden. Die Güte der Vernetzung dieser Tools bestimmt wesentlich die Turn-Around-Zeiten von Fixes.

Das Reporting des Tests wird komplexer. Statt der Meldung eines sequentiellen Forstschritts vermittelt das Reporting das Bild eines Puzzles, das immer mehr Teile hinzugewinnt. Nicht nur das Test-Tool, sondern auch das Fehler-Reporting-Tool und das Liefer-Management-Tool müssen in der Lage sein, die Informationen zu liefern, die notwendig sind, um den Überblick zu wahren.

Wenn auch der schnelle Wechsel von Versionen die Zahl der Tests erhöht, so sind doch viele Tests aus fachlicher Sicht ähnlich oder gleich. Ein Test-Tool, das in der Lage ist, wirkungsvolle Unterstützung bei der Wiederverwendung von Testfällen zu bieten, hilft diese Last zu minimieren.

Die Planung des Tests wird ebenfalls komplexer, da verschiedene Services und deren Releases ihrem Eigenleben entsprechend im Takt ihrer eigenen Lieferplanung zum Test anstehen.

7. Zur Entwicklung von Testtreibern und Test-Stubs

Eine sinnvolle Forderung an die Entwicklung ist es, für jedes Modul vorab Test-Stubs für seine Schnittstellen an die Projekte der Partner-Services zu liefern, um damit vorab Kompatibilität der Services untereinander zu gewährleisten. Diese Stubs werden natürlich auch durch den Test verwendet. Die Akzeptanz des Test-Stubs durch das geprüfte Modul bestätigt die Kompatibilität zur Schnittstelle. Die Test-Stubs stellen damit eine Referenz dar und dies setzt wiederum eine entsprechende Sicherung der Qualität der Test-Stubs voraus.

Idealerweise liefert die Entwicklung aber nicht nur Test-Stubs für die Schnittstellen an der Außenkante des Services, sondern auch für die Schnittstellen zwischen den Modulen innerhalb des Services als Templates an den Test, um mit Hilfe dieser interne, fachliche Prüfungen aufzusetzen. Darüber hinaus ist in der Regel die Unterstützung des Tests durch die Entwicklung bei der Erstellung von Test-Treibern und Test-Stubs in einem gewissen Umfang notwendig, bis die Tester mit der gewählten Technologie und dem technischen Design der Lösung und Ihrer Schnittstellen vertraut sind.

8. Spezielle Risiken des inkrementellen fachlichen Integrations-Tests

8.1. Hohe Volatilität der Schnittstellen

Ändern sich Design und Werte der Schnittstellen zu häufig, müssen die Tests immer wieder überarbeitet und erneut durchgeführt werden. Damit wird der inkrementelle Test immer wieder in seinem Fortschritt zurückgeworfen und verzögert.

Gegenmaßnahmen: Das Schnittstellen-Design frühzeitig festlegen und mit Schnittstellenänderungen diszipliniert umgehen.

8.2. Hohe Komplexität der Schnittstellen

Sind die Schnittstellen zu komplex, stellt sich die Testfallexplosion ein, die vermieden werden sollte. Speziell generische Schnittstellen (Schnittstellen variabler Breite, Smart Keys) erhöhen die Komplexität des Tests bis zur Undurchführbarkeit.

Die Überlegung, aus Gründen der Wirtschaftlichkeit mit wenigen Schnittstellen möglichst viele Anforderungen abzudecken, verkehrt sich zum Nachteil für den Test und für die spätere Wartung und Weiterentwicklung.

Gegenmaßnahmen: Einfachheit beim Schnittstellendesign wahren.

9. Abgrenzung des inkrementellen fachlichen Integrations-Tests vom Unit-Test

Auf den ersten Blick unterscheidet sich der hier beschriebene fachliche Test von Modulen nicht von den Unit-Tests der Entwickler. In beiden Fällen wird die gleiche Technik und Infrastruktur verwendet. Der Unterschied liegt nicht in der Ausführung, sondern im Ansatz.

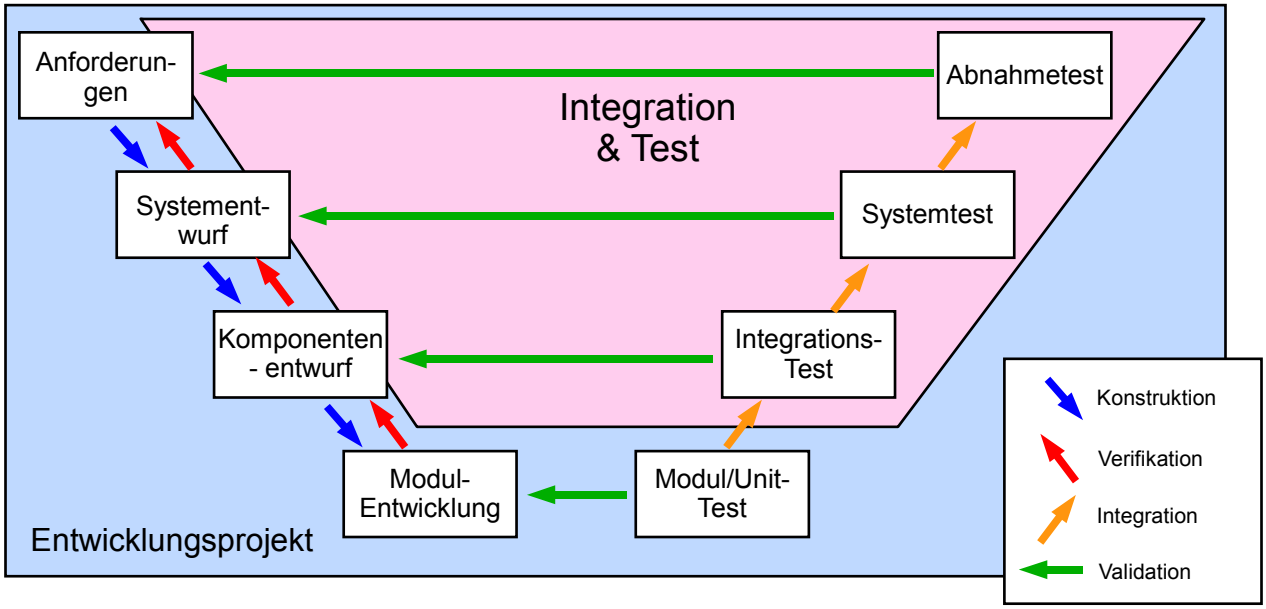

Abbildung 7: V-Modell

Die Unit-Tests der Entwicklung basieren auf den Annahmen und der Interpretation der Anforderungen durch die Entwickler. Bis zur Auslieferung der Module der Applikation durchlaufen die Anforderungen mehrere Stufen der Verarbeitung. In deren Verlauf können sich Abweichungen zu den ursprünglichen Anforderungen einschleichen. Die Aufgabe der fachlichen Tests ist die Identifikation solcher Abweichungen. Die Distanz des Test-Teams zur eigentlichen Umsetzung ermöglicht dem Test einen geschärften Blick für Abweichungen von den Anforderungen. Das hier abgebildete V-Modell soll den Unterschied zwischen den beiden Ansätzen verdeutlichen: die Validation findet auf unterschiedlichen Ebenen statt und referenziert unterschiedliche Quellen. Während der Unit-Test die Vorgaben der Modulentwicklung validiert, vergleicht der fachliche Abnahmetest seine Anforderungen gegen die Vorgaben aus den Anforderungen.

Eine Wiederverwendung der Testtreiber des Unit-Tests der Entwickler scheint sich auf den ersten Blick anzubieten. Es sollte aber darauf verzichtet werden, da man Gefahr läuft, sich auf den gleichen Pfaden zu bewegen, die der Entwickler in seiner Interpretation der Anforderungen eingeschlagen hat. Diese Interpretation könnte schließlich falsch sein. Der fachliche Test muss daher auf einer eigenen Interpretation der Anforderungen basieren, um den ungetrübten Blick zu wahren. Diese Interpretation kann sich technisch durch die Entwicklung beraten lassen, muss aber ihre fachliche Unabhängigkeit wahren. Die Qualität der Tests sollte durch Reviews in Zusammenarbeit mit dem Anforderungsteam gewährleistet werden.

10. Einbeziehung des Deployments

Mit SOA sind Installationen eines Releases gegenüber herkömmlichen Anwendungen häufiger und gleichzeitig komplexer. Um den Überblick zu wahren und das Projekt nicht mit fehlerhaften Installationen zu hemmen, ist es daher wichtig, die Tätigkeiten des Deployments in die Qualitätssicherung mit einzubeziehen. Diese muss gewährleisten, dass einerseits die vorgesehenen Installationen auf den dafür vorgesehenen Systemen installiert wurden und andererseits die Installation vollständig und richtig erfolgte. Dabei ist es von Vorteil, wenn die einzelnen Anwendungen über Health-Check-Routinen verfügen, die über den Zustand der Anwendung, die installierte Version und den Zustand von eventuellen Partnersystemen, von denen es abhängig ist, Auskunft geben.

Die Information über den Status der Deployments auf den verschiedenen Systemen muss dann an die interessierten Parteien weitergeleitet und dokumentiert werden. Hier ist es von Vorteil, wenn die Dokumentation der Deployments integrativ mit den vom Betriebsmanagement, vom Test und von der Entwicklung genutzten Tools geschieht.

Die Qualität des Deployments ist nicht nur für Produktionssysteme wichtig, sondern auch für die Testsysteme; unzureichende Qualität kann zu empfindlichen Störungen des Testbetriebs führen und Testergebnisse nichtig machen.

11. Erfahrungen

Der IFIT hat sich schon in zwei Projekten bestens bewährt. In beiden Fällen stellte sich vor allem die frühzeitige fachliche Prüfung der gelieferten Artefakte als entscheidender Vorteil heraus. Dies verschaffte den zeitkritischen Projekten einen entscheidenden Zeitvorteil.

In beiden Fällen war der neue Ansatz für einen höheren Aufwand bei der Vorbereitung der Tests verantwortlich. Durch den Ansatz war man aber in der Lage, mit einer geringeren Anzahl von Tests ein höheres Vertrauen in die gelieferte Software zu gewinnen. Dies und die kleinen Erleichterungen, die die Vorgehensweise an verschiedenen Stellen bot, kompensierten größtenteils den Mehraufwand der Vorbereitung.

Mit Hilfe der automatisierten Tests konnten die Turn-Around-Zeiten von der Fehlermeldung über die Behebung bis zur Verifizierung des Fix beschleunigt werden. Dem gesamten Entwicklungszyklus verhalf dies zu mehr Agilität.

12. Fazit

Der IFIT nutzt die Möglichkeiten von SOA-basierten Lösungen, um damit komplexe, verteilte Systeme frühzeitig dem Test zu unterziehen. Das Problem der Testfall-Explosion entschärft sich und gefundene Fehler können schneller und genauer geortet werden. Durch den kontinuierlichen Test im Takt der Entwicklung kann zu einem frühen Zeitpunkt eine Aussage zur Qualität der Entwicklung getroffen und möglichen Fehlentwicklungen entgegengewirkt werden.

Kurz nach Abschluss der Entwicklung kann eine Aussage zur Produktionsfreigabe der Anwendung gemacht werden. Damit wird die Länge des Gesamtprojekts signifikant gekürzt.

Das Design der Lösung hat entscheidenden Einfluss auf den Testaufwand und die Testbarkeit der Lösung und somit auf deren Effizienz.

13. Der QA Navigation SLM

Für die Herausforderungen des modernen Software-Lebenszyklus bieten wir eine Lösung an: den Software-Lifecycle Manager, ein Test-, Fehler- und Deployment-Management-Tool, dass sich in Ihre bestehende Infrastruktur integriert. Das in Java und C# verfasste Open-Source-Projekt eröffnet Ihnen alle Möglichkeiten, das Tool ihren individuellen Wünschen anzupassen.

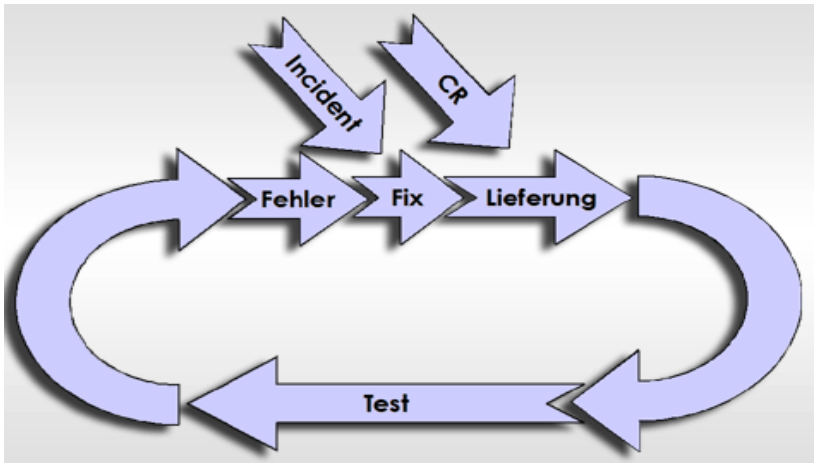

Abbildung 8: Der Software-Lifecycle

Eine ausführliche Beschreibung finden Sie im Internet unter

http://www.qa-navigation/de/Prod.html.

14. Glossar

Fachlicher Modul-Test	Prüfung eines Software-Moduls auf Korrektheit gemäß der fachlichen Anforderungen.
Regressions-Test	Erneute Prüfung bereits getesteter Funktionen, z.B. nach Lieferung eines neuen Releases.
Risiko-basiertes Testen	Planung und Durchführung von Tests, abhängig vom Schadens-Risiko der jeweiligen Funktion bzw. des jeweiligen Moduls.
Schreibtisch-Test	Theoretische Prüfung eines Konzepts durch Simulation des Verfahrens „am Schreibtisch“ ohne Zuhilfenahme der tatsächlichen Umsetzung.
SOA	Service Oriented Architecture
Testfallexplosion	Exponentieller Zuwachs der potentiellen Testfälle durch Kombinatorik der Funktionen mit den Permutationen der jeweiligen Parameter.
Testrahmen	Testinfrastruktur zur Prüfung eines Software-Moduls, bestehend aus Plattform, Testtreiber und Test-Stubs.
Test-Stub	Software zur Emulation von Software-Modulen, zum Test von Software-Modulen.
Testtreiber	Eine Software-Komponente oder ein Testwerkzeug, das ein Programm ersetzt, welches die Kontrolle und/oder den Aufruf einer Komponente oder eines Systems übernimmt.