Gegenseitige Beeinflussungen von Testautomatisierung, Testmanagement und Entwicklung

Jan Düttmann\textsuperscript{1}, Stephan Kleuker\textsuperscript{2}

\textsuperscript{1}Archimedon Software GmbH
Marienstr. 66
D-32427 Minden
Jan.Duettmann@archimedon.de

\textsuperscript{2}Hochschule Osnabrück
Barbarastr. 16
D-49076 Osnabrück
S.Kleuker@HS-Osnabrueck.de


\section*{1 Einleitung}


Mit den ersten Markterfolgen werden Schritt für Schritt neue Ideen und individuelle Kundenwünsche in die Software eingebaut. Dabei muss ein oftmals kleines, eng zusammentarbeitendes Team von Entwicklern früher oder später personell ergänzt werden.
Schließlich muss ein reibungsloser Übergang von der „individuellen Heldenprogrammierung“, bei der alle Beteiligten alle Details des Programm-Codes kennen, zum systematischen Software Engineering stattfinden. Zwar gibt es hierzu eine Vielzahl hilfreicher Literatur, allerdings ist die Frage, welcher Teilprozess als erstes verbessert werden sollte, nicht einfach zu beantworten. Falsche Entscheidungen können hierbei leicht die Existenz eines KMU gefährden, schließlich muss das tägliche Geschäft kontinuierlich weiterlaufen.


Die Arbeiten wurden teilweise mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF) im Projekt KoverJa (Korrekte verteilte Java-Programme) durchgeführt.

2 Fallstudie


2.1 Produkt und Unternehmen


Die modular gestaltete Softwarefamilie admileo® ist eine professionelle Lösung für zukunftsorientiertes Multiprojektmanagement (MPM), Organisations- und Personalmanagement (OPM), Geschäftsentwicklung (GSM), Produktlebenszyklusmanagement (PLM) sowie für eine intelligente, abteilungsübergreifende Aufgabensteuerung. admileo® lässt sich nahtlos an bestehende ERP- und HRM-Systeme ankoppeln, wobei ein Datenaustausch bis auf tiefste Ebenen ermöglicht wird.
Die Project and Business Software admileo basiert ursprünglich auf einer internen Soft-
warelösung für das Management von Großprojekten im Kraftwerksbau. Das System
wurde seit 2006 mit einer kompletten neuen technischen Basis unter konsequenter Nutzung
von Java neu entwickelt. Es wird seitdem kontinuierlich erweitert. Etwas detaillierter
handelt es sich bei admileo um ein modular aufgebautes Client-Server-System mit Da-
tenbankanbindung, welches fast ausnahmslos in Java implementiert ist. Der Server bildet
die Schnittstelle zur Datenbank und übernimmt die Benutzerverwaltung und die Steue-
rung der Kommunikation mit den Clients. Er kann auf diese Weise zum Beispiel die Ak-
tualisierung der Clients veranlassen, wodurch das System eine kollaborierende Arbeits-
weise der Nutzer in Echtzeit gewährleistet. Die Benutzeroberfläche von admileo basiert
auf dem Swing-Framework und verwendet zahlreiche modifizierte und erweiterte
Swing-Elemente für die Darstellung und die Funktionalität der GUI-Elemente. Anwen-
dungsbeispiele hierfür sind die grafische Aufwertung von Navigationsbäumen und die
Darstellung und Editierung von Graphen und Diagrammen.

**Geschäftsmanagement**
- Computer-Telefonie-Interface
- Dokumenten-Manager
- Kunden- und Lieferanten-Manager
- Kontakt-Manager

**Produktlebenszyklusmanagement**
- Anlagen-Manager
- Aufgabensteuerungs-Manager
- Produktdaten-Manager

**Organisations- und Personallangenagement**
- Außendienst-Manager
- Bewerbungs-Manager
- Fremdleistungs-Manager
- Organisations-Manager
- Persönlicher Meldungs-Manager
- Personalqualifikations-Manager
- Personal-Manager
- Resümee-Manager
- Ressourcen-Manager
- Zeiterfassungs-Manager

**Multiprojektmanagement**
- Anfrage- und Änderungs-Manager
- Auftrags- und Projektstruktur-Manager
- Multiprojekt-Manager
- Objekt Meldungs-Manager
- Projekt-Controller
- Projekt-Planer
- Zukunftsprojekt-Manager

Abbildung 1: Anwendermodule von admileo

Die vier Hauptbereiche von admileo bestehen derzeit aus 24 Anwendermodulen, siehe
Abbildung 1, und 25 Konfigurationsmodulen für eine individuelle Prozessanpassung in
der Kundenumgebung. Das System besitzt einen Umfang von weit über 1,5 Millionen
Lines of Code (LoC).

Die Software admileo ist das Premiumprodukt der Archimedon Software + Consulting
GmbH & Co. KG [ASC], einem KMU mit derzeit 18 Mitarbeitern, das auf die Entwick-
lung von Java Software spezialisiert ist. Die Firma mit Hauptsitz in Minden (NRW) un-
terhält seit 2010 eine Zweigstelle in Osnabrück (NDS). Archimedon arbeitet eng mit der
Hochschule Osnabrück zusammen. Die Firma legt sowohl bei der Softwareentwicklung

348

Bereits beim Start der Neuentwicklung lag ein wichtiges Augenmerk auf der Erweiterbarkeit der Software. Das anfänglich noch kleine Team von Entwicklern folgt einem definierten Software-Entwicklungsprozess. Technisch wurde sich sofort für eine Modul- bzw. Komponentenstruktur entschieden, die eine möglichst unabhängige Entwicklung weiterer Module ermöglicht. Grundlage dafür ist eine Postgres-Datenbank, auf die eine mit JPA vergleichbare Persistenzschicht zur Vereinheitlichung der Datenbanknutzung gelegt wurde.

Für neue Module werden zunächst Spezifikationen in Form von Lastenheften geschrie-ben. Diese Prototypen dienen auch zur Analyse und zur Auswahl von Lösungsvarianten. Das zentrale Datenmodell ist so dokumentiert, dass Abhängigkeiten zwischen Modulen schnell erkennbar sind.

Von Beginn an spielte die Qualitätssicherung des Produktes eine wichtige Rolle. Es wur-de darüber hinaus von Beginn an das Ziel verfolgt, sämtliche Schritte im Software-Ent-wicklungsprozess von der Anforderungsanalyse bis hin zur Qualitätssicherung möglichst vollständig zu automatisieren. So wurde aus der Anforderungsdefinition heraus jede Än-derung und Neuentwicklung des Systems über ein workflowbasiertes Aufgabensteuerungsmanagement durchgeführt, was eine sehr detaillierte Versionsplanung sowie ein ef-fizientes Projekt-Controlling ermöglichte. Die Automatisierung von Oberflächentests wurde zwar zu Beginn der Produktentwicklung evaluiert, jedoch auf Grund der mangelnden Werkzeugunterstützung zunächst wieder hinten angestellt. Aus diesem Grund akzeptierten die Entwickler die hohe Produktverantwortung, Fehler durch individuelle Ent-wicklertests zu finden.


Im Laufe der Entwicklung benötigte dieser manuelle Testprozess immer mehr Zeit, da konsequent nicht nur neue Module, sondern auch alte Module mitgetestet wurden, um das Einsickern von Inkonsistenzen in die Software zu vermeiden. Als zweite Optimie-
runsmöglichkeit wurde der relativ starre Release-Zyklus gesehen, der die schrittweise Einführung neuer Funktionalität erschwerte.


2.2 Neuer QS-Prozess

![Diagramm des Build-Prozesses mit integrierter Qualitätssicherung](Diagramm)

Das in Abbildung 2 gezeigte Schaubild gibt einen Überblick über den automatisierten Continuous-Integration-Prozess von der Integration einer Softwareänderung bis hin zur automatisierten Qualitätssicherung. Die wesentlichen Schritte des Prozesses sind:

**Build:** Mit Hilfe des Werkzeugs Jenkins wird hier auf ein Ereignis gewartet, welches einen kompletten Integrationslauf startet (üblicherweise das Einchecken von Quellcode oder die manuelle Ausführung). Ebenfalls übernimmt Jenkins die Steuerung des Build-Vorgangs, indem es über ein Auschecken des Quellcodes aus Subversion die für die lauffähige Software benötigten Artefakte durch Ant-Skripte erzeugt.

**Unit-Test:** Die ausführbare Software wird nun mittels JUnit den Unit- und Integrations-Tests unterzogen. Ant übernimmt dabei auch hier wieder die Ausführung. Mittels eingebundenem JaCoCo wird zeitgleich die Testüberdeckung protokolliert. Als Ergebnisse dieses Schrittes werden JUnit-Testergebnisse und die Messdaten der Testüberdeckung abgespeichert.
GUI-Tests: Im nächsten Schritt wird das Werkzeug QF-Test eingesetzt, um die Benutzeroberflächen-Tests durchzuführen. Um sicherzustellen, dass bei sämtlichen Tests die gleiche Ausgangssituation vorliegt, werden diese Tests auf virtualisierten Rechnern ausgeführt, auf denen eine Jenkins-Instanz im Slave-Betrieb läuft. Auch hier werden wieder die Testergebnisse und die Messdaten der Testüberdeckung protokolliert.


Integrationssysteme: Sind sämtliche Schritte erfolgreich abgelaufen, werden die sogenannten Integrationssysteme erzeugt, welche mit lauffähigen Demo-Daten gefüllt werden, um ständig die aktuelle Software präsentieren und den momentanen Entwicklungsstand nachvollziehen zu können.

Für jeden einzelnen Schritt im Build-Vorgang gilt: Scheitert der entsprechende Vorgang, bricht der Prozess an dieser Stelle mit einer Benachrichtigung an die verantwortliche(n) Person(en) ab. Damit wird gewährleistet, dass auf die Integration einer Software-Änderung schnellstmöglich eine Rückmeldung aus dem Continuous-Integration-System darüber erfolgt, welche negativen wie positiven Auswirkungen die soeben getätigte Änderung auf das Produkt hat.

2.3 Fazit der Prozessanpassung

Zur rein technischen Umsetzung des Integrations- und Testprozesses lässt ist festzustellen, dass diese sehr reibungslos implementiert werden konnte und sich nahtlos in den bisherigen Entwicklungsprozess einfügt. Sämtliche hier verwendeten Tools sind so ausgereift und stabil, dass sie in der alltäglichen Verwendung keine nennenswerten Probleme bereiten. Der Betrieb eines solchen Continuous Integration- und Testing-Systems bringt einen Wartungsaufwand von wenigen Mannstunden pro Monat mit sich, dem stehen viele Vorteile gegenüber, die eine solche Umgebung vollautomatisch mit sich bringt. Insbesondere sind hier zu nennen:

- Ständige Verfügbarkeit von ausführbaren Testsystemen für manuelle Tests sowie Produktpresentationen
- Ausführung aller automatisierten Tests ohne Benutzereingriff
- Risikominimierung für den Releasewechsel beim Kunden (dieser wird mit jedem Build-Schritt mitgetestet)

Die wesentliche Herausforderung bei der Umsetzung des Prozesses liegt daher weniger in der technischen Implementierung der Werkzeuge und Verfahren, sondern vielmehr in der Integration der Testautomatisierung in den Entwicklungsprozess. Damit ein kontinuierlicher Testprozess seine Vorzüge voll entfalten kann, ist es unabdingbar, die Umstellung von manuellen auf automatische Tests möglichst weitreichend durchzuführen. Hier-

3 Lessons Learned


3.1 Testarchitektur

Der Hauptkritikpunkt an einer Testautomatisierung ist, dass die entstandenen Tests sehr gut zum aktuellen getesteten System passen, allerdings mit der Weiterentwicklung des Systems immer wertloser werden. Die Ursachen sind meist, dass die Anpassung der Tests als zu aufwändig angesehen wird und dass neue Tester die Einarbeitung in die komplexe Testentwicklung scheuen.


Eine Testarchitektur ist damit ein zentrales Mittel, flexibel auf Änderungen im zu testenden System oder auf neue Möglichkeiten zu reagieren. Die Testarchitektur spiegelt oft in


Für Komponenten, die nicht eng in der Architektur vernetzt sind, wie z. B. Oberflächen, bietet sich dann eine Schicht mit WebServices an, über die die Oberflächen die benötigten Daten abrufen können. Dieser Ansatz verlangt eventuell die Umsetzung bei der Nutzung spezieller Frameworks, wobei dann individuell zu klären ist, ob dies Auswir-

Abbildung 3: Basisarchitektur für wart- und erweiterbare Systeme

Aus Sicht der Qualitätssicherung haben die Überarbeitungen der betrachteten Systeme auch deutlich gemacht, dass es wichtig ist, dass Tests, insbesondere Abnahmetests klar spezifiziert sind, da sie die Grundlage der Tests für das neue System bilden. In den konkreten Fällen kommen klassische Testspezifikationen in Text-Form zum Einsatz.

3.2 Testorganisation und Testmanagement


passenden Tests auswählen. Ein freies System, das sich leicht in Build Managementsysteme einbauen lässt, fehlt aber.

3.3 Neue Technologien

Die kontinuierliche Weiterentwicklung im Bereich Software Engineering hat unmittelbare Auswirkungen auf die Testentwicklung. Ein konkretes Beispiel sind die Möglichkeiten, die in Java durch CDI (Contexts and Dependency Injection [@CDI]) relativ neu für die Enterprise Edition (JEE), aber auch nutzbar in der Standard Edition (Java SE) hinzugekommen sind. CDI umfasst mehrere Möglichkeiten, die Umsetzung von Funktionalität alternativ zu programmieren. Ein wichtiger Ansatz ist, für Objektvariablen nur Schnittstellen zu nutzen, so dass durch verschiedene Implementierungen der Schnittstellen diese leicht gegeneinander ausgetauscht werden können. Im Fall von JEE kann der umgebende Server dafür verantwortlich sein, die passende Implementierung zur Verfügung zu stellen. Dies kann im konkreten Fall bedeuten, dass der Programmiere nur noch spezifiziert, dass er eine transaktionale Datenbankverbindung haben möchte.

```java
@Inject
private PersistenzService db;

public void persist(Object object) {
    db.persist(object);
}
```

Im weiteren Programm wird dann die Variable db z. B. in der Methode persist() einfach genutzt, ohne dass ihr explizit einen Wert zugewiesen wurde.


@InterceptorQualifier // Methoden dieser Klasse beobachten
public class EventConsumer {
  ...
}

@InterceptorQualifier
@Interceptor // so annotierte Klasse beobachtet @InterceptorQualifier
public class MeinInterceptor {
  @AroundInvoke // Methode wird vor jeder Methode von EventConsumer
  // aufgerufen, Details über Parameter abfragbar
  public Object logCall(InvocationContext context)
  throws Exception {
    ...
    return context.proceed(); // ursprüngliche Methode fortsetzen
  }
}

<table>
<thead>
<tr>
<th>Test</th>
<th>zu testende Software</th>
<th>Interceptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Before</td>
<td>public class SUT{</td>
<td>@InterceptorQualifier</td>
</tr>
<tr>
<td></td>
<td>@InterceptorQualifier</td>
<td>@Interceptor</td>
</tr>
<tr>
<td>@Test</td>
<td>public void testx(){</td>
<td>public class MeinInterceptor {</td>
</tr>
<tr>
<td></td>
<td>sut.methode():</td>
<td>@AroundInvoke</td>
</tr>
<tr>
<td></td>
<td>// pruefe Inceptor-Ergebnis</td>
<td>public Object logCall(</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>InvocationContext context)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Abbildung 4: Aufruf von Tests mit Nutzung von Aspektorientierung


CDI ist damit ein weiterer Baustein, der bei der Testplanung, was soll wann wie getestet werden, zu beachten ist. Angemerkt sei, dass das Thema Aspektorientierung schon relativ lange bekannt ist [Lad03] und es auch mehrere gut nutzbare Frameworks dazu gibt, die Verbreitung des Ansatzes aber immer gering war. Da durch CDI die Aspektorientierung ein weiterer direkt zu nutzender Baustein der Entwicklung wird, steigt dadurch die Popularität des Ansatzes deutlich an.

Die freie Mock-Bibliothek Mockito [@Moc] bietet neben der Mock-Erstellung für noch nicht realisierte Klassen die Möglichkeit, existierende Klassen oder einzelne Methoden davon zu mocken. Dies bedeutet, dass in der laufenden Software ein Methodenauftrag abgefangen und durch eigene Funktionalität ersetzt werden kann. Da die nicht gemockten
Methoden erhalten bleiben, kann so eine konkrete Implementierung einfach in einen Mock umgewandelt werden. Dies ist z. B. dann sehr nützlich, wenn man sicherstellen will, dass eine bestimmte Methode für konkrete Parameterwerte garantiert eine Exception wirft. Das nachfolgende Beispiel zeigt, dass der konkrete Aufruf get(0) für eine Liste statt dem eingetragenen Wert 1 den Wert 42 liefert.

```java
public static void main(String[] args) {
    List<Integer> list = new LinkedList<Integer>();
    list.add(1);
    List<Integer> spy = Mockito.spy(list);
    Mockito.doReturn(42).when(spy).get(0);
    System.out.println(spy.get(0) + " :: " + spy.size());
}
```


Direkt auf den Testbereich bezogen sieht man die neuen Testmöglichkeiten z. B. an der kontinuierlichen Weiterentwicklung der integrierten oder leicht zu installierenden Erweiterungen in der Entwicklungsumgebung Visual Studio von Microsoft, was insbesondere die sogenannten Ultimate Editions betrifft. Relativ neu ist die Möglichkeit, unmittelbar ein ausführbares Programm zu beeinflussen, indem festgelegt wird, dass bei bestimmten Methodenaufriufener anderer Code ausgeführt wird, was verwandt mit dem vorgestellten Mockito-Ansatz ist. Das Microsoft Fakes Framework hilft dabei, zu testenden Code zu isolieren. Dabei werden Teile des Codes durch sogenannte Stubs oder Shims ersetzt. Dies sind kleine Code-Fragmente, die völlig unter Kontrolle der Tests stehen [@Mic]. Der Ansatz erlaubt es, fertige Software zu analysieren und zu beeinflussen, was auf konzeptioneller Ebene den vorgestellten Möglichkeiten der Aspektorientierung und damit CDI entspricht.


3.4 Metriken und Überdeckungen

In der vorgestellten Fallstudie wurden Werkzeuge zur Messung einiger Metriken und zur Analyse der erreichten Überdeckungsgrade eingebaut. Beide Ansätze zeigen sich als hilfreich. Metriken eignen sich insbesondere zur Analyse der Komplexität einzelner Methoden und Klassen sowie zur Analyse der Anzahl der Verknüpfungen und damit Abhängigkeiten von Klassen. Mit Überdeckungen wird insbesondere festgestellt, welche
Bereiche des Programms nicht oder nur in geringem Maße getestet werden. Die genann-
ten Maße sind aber mit Vorsicht zu betrachten, da sie immer nur Indikatoren liefern, aber
weder eine Aussage erlauben, dass die zu testende Software und die Tests in einem gu-
ten Zustand sind, noch dass Verstöße auf Probleme hinweisen müssen.

Dass es sich nur um Indikatoren handelt, ist unbedingt bei der Testerstellung zu berück-
sichtigen. Generell fokussieren Tests zunächst auf die typischen, vom Endnutzer später
am meisten genutzten Abläufe, was natürlich für Systemtests aber oft auch für Unit-
Tests gilt. Im nächsten Schritt werden die Randfälle und erwarteten Ausnahmefälle be-
rücksichtigt. Erst dann ist es sinnvoll, auf Überdeckung zu achten und z. B. die Frage-
stellung aufzuwerfen, warum gewisse Methoden in den erstellten Testfällen nicht be-
rücksichtigt werden. Erst wenn sichergestellt ist, dass die Methode benötigt wird, sollte
man für sie weitere Tests ergänzen. Sehr wichtig bleibt dabei trotzdem, dass die Überde-
cckung ein sehr trügerisches Maß sein kann, wie das Beispiel in Abbildung 5 zeigt, das

Abbildung 5: falsche Software mit perfekter Überdeckung

Die Abbildung 5 zeigt links eine einfache Methode zum Berechnen des Maximums drei-
er übergebener Werte. Die rechte Seite zeigt den zugehörigen Kontrollflussgraphen. Mit
den links unten spezifizierten Testfällen werden alle Kanten des Graphen genutzt, man
spricht von einer vollständigen Zweigabdeckung (C1-Überdeckung) oder auch der Ga-
rantie, dass jeder mögliche Ablaufschritt einmal ausgeführt wurde. Trotz der vollständi-
gen Testabdeckung, wobei in jedem Test das erwartete Ergebnis vorliegt, werden die
zwei enthaltenen gravierenden Fehler nicht gefunden. Die Methode liefert zunächst im-
mer den Wert null, wenn alle Parameter den gleichen Wert haben. Weiterhin liefert die
Methode das falsche Ergebnis, wenn x der größte Wert und z größer als y ist.
3.5 Qualifikation von Mitarbeitern

Die vorherigen Kapitel machen sehr deutlich, dass die Anforderungen an die Qualifikation von Mitarbeitern der QS ständig steigen. Neben den zentralen Grundbegriffen und elementaren Prozessen der QS [SW01], müssen zumindest einige Mitarbeiter Erfahrungen in der Software-Architektur besitzen und sich kontinuierlich in neuen Entwicklungs- und Testtechnologien schulen.


4 Zusammenfassung und Ausblick

Der Weg zu langfristig wart- und erweiterbarer Software besteht aus der kontinuierlichen Anpassung der Entwicklungsprozesse an neue Projektgegebenheiten, wie die Projektgröße und Projektvarianten. Ein zentraler Prozess des LotSE ist dabei die Qualitätssicherung. Die Automatisierung wesentlicher Schritte der Qualitätssicherung garantiert dabei die Testbarkeit des Systems, ist aber nur ein Teil des Ansatzes. Weitere, hier nicht im Detail betrachtete Bestandteile, umfassen den komponentenbasierten Software-Aufbau,
die hohe Qualifikation der Mitarbeiter und die kontinuierliche Suche nach Prozessverbesserungsmöglichkeiten.

Die in der Fallstudie beschriebene Prozessoptimierung zeigt ein erfolgreiches Beispiel dafür, wie durch Verbesserungen eines aktuell gelebten Entwicklungsprozesses eine Effizienzsteigerung in einem Software-herstellenden KMU durch Automatisierung von (GUI-)Tests erreicht wird. Beachtet werden muss, dass der Weg von der Idee zur Automatisierung bis hin zur Umsetzung nicht trivial ist, genau geplant werden muss und es durchaus Faktoren geben kann, die zur Nichtumsetzung führen können. Der beschriebene Automatisierungsansatz wird weiterentwickelt.

Bei den QS-Prozessen und QS-Werkzeugen, aber auch in anderen Bereichen, muss man über Innovationen und ihre mögliche Integration in den bisherigen Entwicklungsweg nachdenken. Zu beachten ist, dass ein einfacher Kriterienkatalog bzgl. der geforderten Funktionalität zu restriktiv sein kann. Man muss zunächst die Möglichkeiten analysieren, welche Werkzeuge anbieten, die aber im aktuellen Entwicklungsprozess noch keine Rolle gespielt haben. Diese Innovationsmöglichkeit ist in der Abbildung 6 zusammengefasst, die von der Ausgangssituation auf der linken Seite nach der Vorauswahl geeigneter Werkzeuge (W) den Prozess zur Evaluation der Werkzeugmöglichkeiten und die resultierende neue Entwicklungsumgebung auf der rechten Seite zeigt.

Abbildung 6: Evaluation innovativer QS-Werkzeuge

Generell ist die QS ein Prozess, der durch neue Technologien auch im Bereich des Software-Engineering sich in ständiger Entwicklung befindet und nur durch hochqualifizierter Mitarbeiter langfristig bei einer sich weiterentwickelnden Software effizient nutzbar bleiben kann.
Literaturverzeichnis

[Bur14] B. Burke, RESTful Java with JAX-RS 2.0, O’Reilly, Sebastopol (CA), USA, 2014
[HF10] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1, Addison-Wesley Longman, Amsterdam, 2010