Marco Schulz
(banaalo@web.de)
studierte an der HS Merseburg Diplominformatik. Sein persönlicher Schwerpunkt liegt in der Automatisierung von Build-Prozessen und dem Softwarekonfigurationsmanagement. Seit über zehn Jahren entwickelt er auf unterschiedlichen Plattformen Webapplikationen. Derzeit arbeitet er als freier Consultant und ist Autor verschiedener Fachartikel.

Continuous Delivery

Die Einführung der Java-Enterprise-Plattform läutete eine neue Ära in der Softwareentwicklung ein. Der Wechsel von der klassischen Desktopanwendung hin zum Cloud-Service ist der derzeitige Stand der Technik. Mit der Entscheidung für diese neue Technologie eröffnet sich ein breites Spektrum an Möglichkeiten, die aber auch Veränderungen im Softwareentwicklungsprozess nach sich ziehen. Dies spiegelt sich unter anderem in dem neuen Kunstwort *DevOps* wieder, das die bisher getrennt betrachteten Bereiche *Development* und *Operation* zu einer geschlossenen Einheit zusammenführt. Mit *Continuous Delivery* wurde eine weitere Stufe in der Prozessautomatisierung etabliert, die den wachsenden Anforderungen moderner Webanwendungen gerecht wird. Der Artikel gibt eine Einführung in die Thematik und stellt die wichtigsten Konzepte vor.

Permanentes Ausliefern von Cloud-Diensten

Dass komplexe Anwendungen keine reinen Desktoplösungen mehr sein müssen, sondern durchaus als Thin Client im Browser umgesetzt werden können, hat Google mit seinem kostenfreien Dienst Docs unlängst bewiesen. Softwarehersteller werden voraussichtlich ihre Produkte in Zukunft vermehrt als Cloud-Lösung vertreiben. Diese Vision lässt sich hervorragend mit den Cloud-Ebenen IaaS, PaaS sowie SaaS weiterentwickeln, wobei der Fokus vermehrt auf SaaS (Software as a Service) liegt.

SaaS bietet für die verschiedensten Szenarien, wie Business to Business (B2B) und Business to Customer (B2C), hervorragende Ansätze. So ist es denkbar, dass beispielsweise ein bekannter Hersteller für Bildbearbeitung sein Portfolio hin zur Cloud migriert. Damit eröffnen sich diesem Unternehmen verschiedenste Perspektiven, seine Dienste erfolgreich zu vermarkten.

Abb. 1: Allgemeine Phasen des Softwareentwicklungsprozesses

Privatanwender, welche die enormen Anschaffungskosten für die Desktopinstallation nicht aufbringen können, erhalten einen legalen Zugang über ein „Pay per Use"-Lizenzmodell. Selbstständige, professionelle Nutzer und Unternehmen buchen die benötigten Dienste als Flatrate für verschiedene Zeiträume.

Zudem besteht noch die Option, eine Auswahl an Webservices oder ganzen Prozessketten anderen Unternehmen bereitzustellen, die diese Dienste in ihren eigenen Applikationen verwenden. Somit könnte beispielsweise eine Onlinedruckerei ihren Kunden auf ihrem Onlineportal einen zugekauften Service zur digitalen Bildbearbeitung anbieten, um damit das eigene Produkt aufzuwerten, ohne das eigentliche Kerngeschäft durch Nebenkriegsschauplätze zu vernachlässigen.

Der übliche Weg mit einer herkömmlichen Release-Planung ist bei den vorgestellten Szenarien eher eine unbefriedigende Lösung. Beispielsweise kommt dem kontinuierlichen Einspielen von Sicherheitspatches, um den reibungslosen Betrieb aller Services zu gewährleisten, eine große Bedeutung zu. Ähnlich verhält es sich mit neuer Funktionalität, die dem Anwender möglichst schnell bereitgestellt werden soll. Allein durch eine stetige Weiterentwicklung der Dienste können Unternehmen mögliche Wettbewerbsvorteile für sich nutzen.

Das gelingt aber nur dann, wenn innovative Ideen schnell realisiert werden kön-

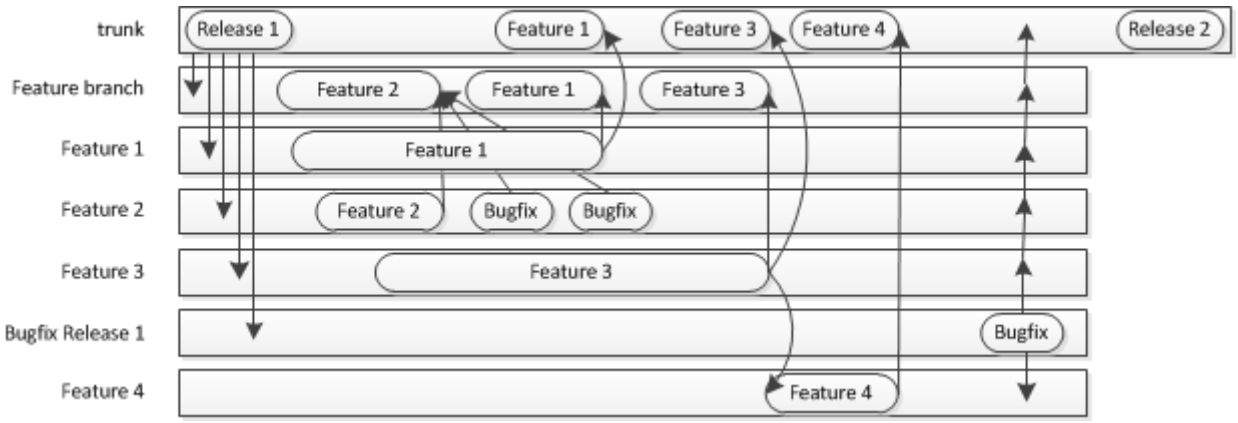

Abb. 2: Branchmodell

nen und unmittelbar nach einer gelungenen Umsetzung im Produktivbetrieb zur Verfügung stehen. Deshalb ist es nicht immer ratsam, Funktionalitäten in einem Release zu sammeln und später in größeren Zyklen zu deployen.

Continuous Delivery [CD] beschreibt die Methodik, wie durch den Einsatz von Automatismen im Softwareentwicklungsprozess permanentes Ausliefern ermöglicht wird. Der Erfolg des Vorhabens hängt in den seltensten Fällen von den ausgewählten Werkzeugen ab, sondern davon, wie diese durch richtigen Gebrauch optimal miteinander kombiniert werden.

CD und DevOps – es wächst zusammen, was zusammengehört

Agile Managementmethoden, wie SCRUM oder Kanban, sind für CD geeignete Planungswerkzeuge. Mit zügigen Iterationen kann gezielt auf kurzfristig vorkommende Adaptionen eines Features eingegangen werden. Im Gegensatz zum klassischen Ansatz für Releases werden die typischen Phasen des Softwareentwicklungsprozesses (vgl. Abbildung 1) für jedes einzelne Feature durchlaufen.

Erst in einem nachgelagerten Prozess zur Strukturierung der Dokumentation wird im Source Control Management (SCM) ein Stand als Release gekennzeichnet. Diese Vorgehensweise erleichtert die Organisation der fertiggestellten Funktionalitäten. Ein zentraler Punkt des CD ist die Organisation des Quellcode-Repositorys. Damit die Applikation jederzeit deployfähig bleibt, haben sich Branchmodelle (vgl. Abbildung 2) als sehr nutzbringend erwiesen.

Die Grundidee ist, im Hauptentwicklungsstrang *(trunk)* des SCM fest definierte Punkte, die Releases, vorzuhalten. Für jede neue Funktionalität wird vom Trunk eine neue Verzweigung *(branch)* erzeugt. Alle fertiggestellten Features werden vorerst in einen separaten Feature-Branch zusammengeführt (merge).

Für jedes Release existiert ein eigener Bugfix-Branch, in dem globale Fehler behoben werden. Global bezieht sich vor allem auf Lösungen, die sich aus dem Zusammenschluss mehrere Features ergeben haben. Fehler, die einem konkreten Feature zugeordnet werden können, sind im entsprechenden Branch des Features zu beheben.

Auf den ersten Blick erzeugt dieses Vorgehen eine nicht unerhebliche Anzahl an Verzweigungen, was aber eine enorme Flexibilität erlaubt. Es können praktisch alle möglichen Kombinationen von umgesetzten Funktionalitäten zu einer Anwendung zusammengebaut werden. Das gestattet es im Falle einer fehlgeplanten Funktionalität, diese aus dem Livesystem herauszunehmen oder auf ein späteres Fertigstellungsdatum zu verschieben.

Für einen effizienten Umgang mit SCM-Systemen sollten im Repository keine Binärdateien, wie beispielsweise zugekaufte Bibliotheken, eingecheckt werden. Dafür gibt es eigene Lösungen [Nexus], die es erlauben, verschiedene Versionen eines Artefaktes mit einem Repositorymanager unter Konfigurationsmanagement zu stellen.

Dadurch ergeben sich mehrere Vorteile. Einerseits bleibt das Quellcode-Repository angenehm klein, was wiederum zu kurzen Checkout-Zeiten führt. Ein weiterer Vorteil ist, dass auch eigene Artefakte bereits als fertiges Kompilat im Repositorymanager vorgehalten werden können. Das wirkt sich positiv auf die Dauer des Builds aus und ermöglicht das Verteilen der eigenen Artefakte unternehmensweit.

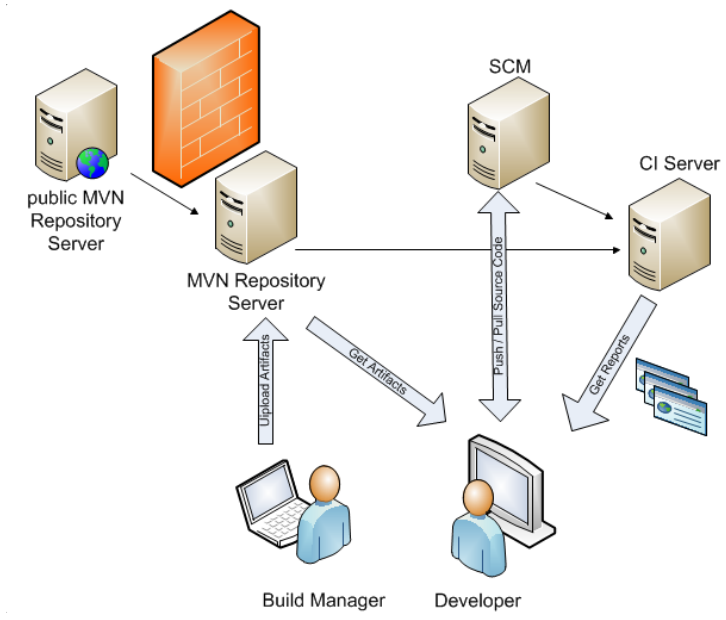

Abb. 3: Build-Infrastruktur

```
[toggle.props]
debugOut.activation=true
debugOut.parameter=debug

[FeatureToggle.java]
public class FeatureToggle {
  private static Properties properties;
  private static final String PROPERTY_FILE = "toggle.props";

  private static void readPropertyFile(final String file)
      throws IOException {
    Properties props = new Properties();
    props.load(file);
    return props;
  }
  // TOGGLES ##############################################################
  public boolean DEBUG_OUT =
    properties.getProperty("debugOut.activation");
  public String DEBUG_OUT_PARAM =
    properties.getProperty("debugOut.parameter");
}

[JSF: template.xhtml]
<c:if test="${featureToggle.DEBUG_OUT eq true}">
  <c:if test="${param.debug eq true}">
    <ui:include src="debugOut.xhtml" />
  </c:if>
</c:if>
```

Listing 1: Toggle

Das Bindeglied zwischen Quellen und Binärartefakten ist die Build-Logik, die den einzelnen Modulen die entsprechenden Artefakte in ihrer korrekten Version zuordnet. Abbildung 3 zeigt den schematischen Aufbau einer Build-Infrastruktur mit Maven als Build-Logik und einem Continuous-Integration(CI)-Server.

Aus Sicht des IT-Betriebs (Operation) ist Monitoring ein wichtiges Instrument, um detaillierte Aussagen über das Verhalten einer Anwendung treffen zu können. Das einfachste Mittel, eine Anwendung zu überwachen, ist Logging. Ein moderner Logger ist logback, der es ermöglicht, während der Laufzeit der Applikation den LogLevel zu ändern, ohne den Server oder die Anwendung erneut zu starten.

Feature Toggle-Pattern

Das stetige Deployen neuer Funktionen erfordert auch einen Mechanismus, um einzelne Funktionalitäten nach Bedarf zu aktivieren oder zu deaktivieren. Ein sehr praktischer Helfer ist das Feature Toggle-Pattern [Fowl10]. Toggels sind Schalter, die entweder über eine Property oder anhand eines festgelegten URL-Parameters gesteuert werden. Die Umsetzung ist sehr einfach bewerkstelligt, indem an der entsprechenden Stelle im Code, die ein Feature einbindet, abgefragt wird, ob der URL-Parameter beziehungsweise die Property gesetzt ist. Wie die Implementierung dazu aussehen kann, ist in Listing 1 kurz skizziert.

Schon das Branchmodell selbst ist sehr mächtig, allerdings erreicht man mit dem Feature Toggle eine höhere Flexibilität. Das Deployment besteht aus mehreren Stages (vgl. Abbildung 4), die für die Qualitätskontrolle unerlässlich sind. Schließlich möchte man um jeden Preis vermeiden, dass ungetesteter Code auf der Liveumgebung eingespielt wird und das System im schlimmsten Fall längere Zeit nicht erreichbar ist.

Die QS-/Testumgebung beinhaltet üblicherweise mindestens zwei verschiedene Versionen: eine Test-Instanz für den Nightly Build und eine QS-Instanz für die ausgelieferten Releases. Es ist sehr komfortabel, per Konfiguration testen zu können, welche Auswirkungen sich ergeben, wenn eine Funktionalität deaktiviert werden soll.

Neben der reinen Bequemlichkeit ist die gewonnene Zeitersparnis kein unbedeutender Faktor. Auch auf der Seite des Livesystems kann es notwendig werden, Funktionen, die sich als problematisch erwiesen haben, mit wenig Aufwand abzuschalten. Dies kann beispielsweise bei Nichtakzeptanz durch den Nutzer erforderlich werden.

Automatisches Deployen der Datenbank mit Hibernate und dem DAO-Pattern

Es gibt verschiedene Möglichkeiten für den Umgang mit Datenbanken. Das Unternehmen Red Gate [Hum14] hat in seinem Portfolio einige Produkte, um Datenbanken zu vergleichen und unter Konfigurationsmanagement zu stellen. Für den IT-Betrieb haben diese Werkzeuge durchaus einen erheblichen Mehrwert, um bestehende Datenbankmanagementsysteme (DBMS) zu warten.

Es ist wichtig, das DBMS möglichst austauschbar zu halten. Ein Schritt in Richtung dieses Ziels ist der Einsatz eines O/R-Mappers, der Objekte auf Datenbanktabellen abbildet. Eines der bekanntesten Tools ist Hibernate, das eine Implementierung des Java Persistence API (JPA) ist. Sehr angenehm ist vor allem die Tatsache, dass kaum noch SQL im Code benötigt wird und die Persistenzobjekte direkt verarbeitet werden können.

Beim Design der Domäne wird kaum noch normalisiert. Über die Konfiguration kann Hibernate angewiesen werden, beim

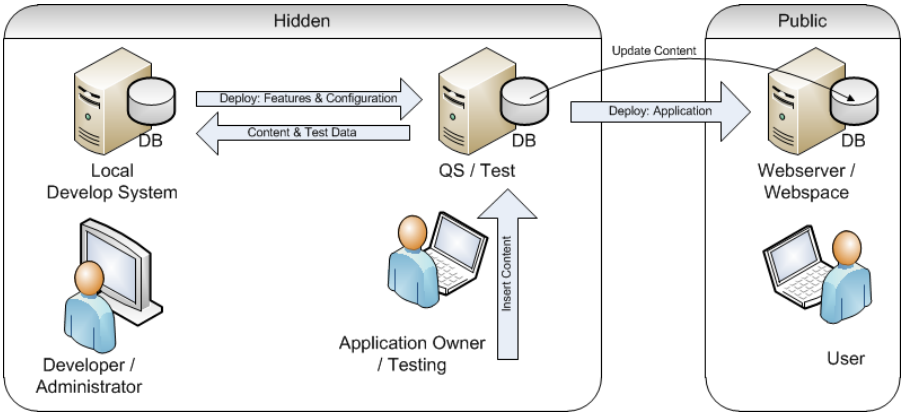

Abb. 4: Deployment-Stages

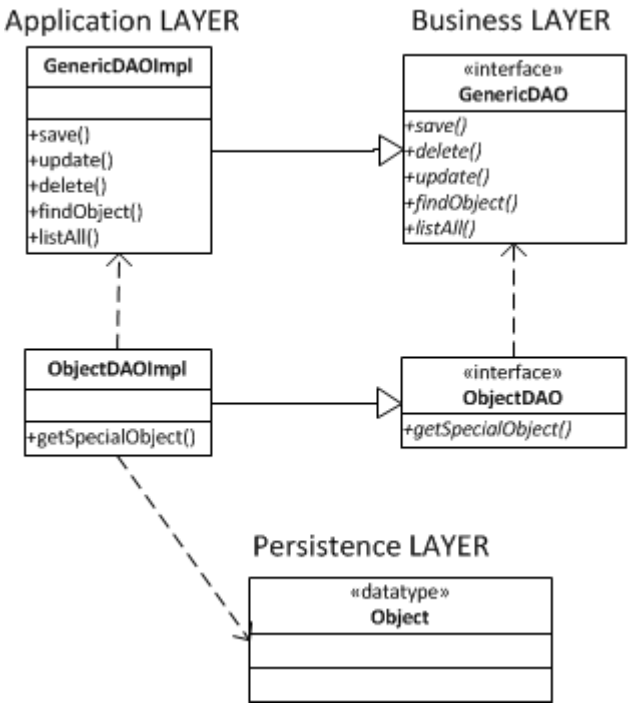

Abb. 5: Datenzugriffsobjekte

Starten der Applikation die Datenbank aus den Entitätsobjekten zu generieren. Mit einem import-Skript wird anschließend die frisch generierte Datenbank befüllt. Da die Entitäten einfache POJOs sind, die durch Annotationen angereichert wurden, steht die Datenbank automatisch unter Konfigurationsmanagement.

Dass eine geschickt gewählte Architektur sich günstig auf ein Projekt auswirkt, ist allseits bekannt. Manche Maßnahmen haben besonders positive Effekte für das permanenten Ausliefern.

Da CD von der Idee Continuous Integration motiviert wurde, ist das Testen auch ein zentraler Punkt. Mit generischen Codefragmenten ist es möglich, die Menge der benötigten Testfälle übersichtlich zu halten. Die Funktionen Speichern, Löschen, Ändern, Finden und Liste alles haben alle Entitäten gemeinsam.

Daher ist es wünschenswert, dies nur einmal implementieren und testen zu müssen. Spezifische Funktionen können dann je nach Bedarf für die einzelnen Entitäten nachgerüstet werden. Abbildung 5 zeigt, wie das DAO (data access object) aufgebaut ist.

Die Umsetzung ist recht leicht be-werkstelligt. Jedes Object-DAO ist vom **`GenericDAO`** abgeleitet, das die generischen Methoden **`save, delete, update, findObject`** und **`listAll`** implementiert. Die konkreten ObjectDAOs greifen direkt auf die Entitäten der Persistenzschicht zu.

```
public class GenericDAOImpl<T> implements GenericDAO<T> {
  private EntityManager entityManager;

  public final List<T> listElements(final Class<T> object) {
    Session session = (Session) entityManager.getDelegate();
    Criteria results = session.createCriteria(object);
    return results.list();
  }
}
public class UserRoleService {
  private RolesDAO roles;
  public UserRoleService() {
    super();
    this.roles = (RolesDAO) context.getBean("rolesDAOImpl");
    this.roles.setEntityManager(entityManager);
  }
  public List<Roles> getAllRoles() {
    transactionHandler();
    List<Roles> roleList = roles.listElements(Roles.class);
    return roleList;
  }
}
@Entity
@Table(name = "ROLES")
public class Roles implements Serializable {
  private static final long serialVersionUID = 1L;
  private String name;
  private String description;

  public Roles() { /* NOT IN USE */ }

  public Roles(final String roleName, final String roleDescription) {
    name = roleName;
    description = roleDescription;
  }
  @Id
  @Column(name = "NAME", length = 50)
  public String getName() {
    return name;
  }
  @Column(name = "DESCRIPTION", length = 255)
  public String getDescription() {
    return description;
  }
  public void setName(final String name) {
    this.name = name;
  }
  public void setDescription(final String description) {
    this.description = description;
  }
}

<ul>
  <c:forEach var="elements" items="#{userRoleService.allRoles}">
    <li>${elements.name}</li>
  </c:forEach>
</ul>
```

Listing 2: Datenzugriffsobjekt

In den Services wird über die Interfaces der Business-Schicht auf die DAO-Objekte zugegriffen. Das Codefragment für die Methode **`listElements()`** in Listing 2 zeigt, wie das Pattern in Verbindung mit Dependency Injection von Spring verwendet werden kann.

Ausblicke

Die Idee des Continuous Delivery ist nicht allein auf die Java-Plattform beschränkt, daher noch ein paar abschließende Worte zur .NET-Plattform. Es fällt unter anderem auf, dass die meisten Dienste, wie Nexus oder der Jenkins-CI-Server, auch .NET-Plug-ins bereitstellen. Für bekannte Artefakte wie JUnit und Hibernate existieren ebenfalls Adaptionen.

Der .NET-Support des Nexus-Repositorymanagers ist allerdings auf die kostenpflichtige Enterprise-Version beschränkt. Dafür können die mit dem in .NET ver-

breiteten Repositorymanager NuGet erzeugten Artefakte unkompliziert verwaltet werden.

Bei den Build-Werkzeugen schaut die Welt leider noch ein wenig trüb aus. Zwar kann Maven auch C#-Projekte kompilieren, aber der Umgang mit Mono, der Linux-Variante des .NET-Frameworks, birgt durchaus einige Stolpersteine, die erst beiseite geschafft werden müssen. Wer mit Ant vertraut ist, wird sich auch in NAnt schnell zurecht finden, wobei schlechte Dokumentation mit teilweise falschen Beispielen mehr als nervenzehrend ist.

Lichtblicke gibt es auch beim Microsoft Team Foundation Server (TSF) zu melden, der mit der Version 2013 nun auch Git unterstützt. So bleibt die Hoffnung, dass in Zukunft auch Subversion einen Platz im TSF bekommt, sodass Unternehmen auch ihre älteren Projekte einbinden können. ■

Literatur und Links

[CD] Wikipedia, http://en.wikipedia.org/wiki/Continuous_delivery

[Fowl10] M. Fowler über Feature Toggle, http://martinfowler.com/bliki/FeatureToggle.html

[Hum10] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, Addison-Wesley, 2010

[Hum14] Blog von Jez Humble zum Thema CD, http://continuousdelivery.com

[Marc13] T. DeMarco, Peopleware: Productive Projects and Teams, 3rd Edition, Addison-Wesley, 2013

[Nexus] Nexus Repository Manager, http://www.sonatype.org/nexus/

[RedGate] Database tools, http://www.red-gate.com/

[Swart12] P. Swartout, Continuous Delivery and DevOps: A Quickstart guide, Packt Publishing, 2012

Der Beitrag wurde ebenfalls in der Printausgabe von JAVAspektrum 03/2014 veröffentlicht.

Literaturliste

▶Das bei Addison-Wesley erschienene Buch [Hum10] von Jez Humble und David Farley beschäftigt sich vorrangig mit den einzelnen Stationen des Softwareentwicklungsprozesses und wie diese gemeistert werden können. Die Autoren beschreiben viele bekannte Kniffe, die zum Alltag eines Deployment-Managers gehören, wobei sie aber keine Anleitung für bekannte Tools zusammengestellt haben.

Das Buch stellt Konzepte vor, die es dem Leser erlauben, für seine individuellen Bedürfnisse eine geeignete Orchestration bekannter Werkzeuge zusammenzustellen. Es wurden die Punkte herausgekehrt, welche für eine erfolgreiche Automatisierung abgearbeitet werden müssen, ohne dabei ein besonderes Tool oder Plattform zu bevorzugen.

Damit gelang den Autoren ein zeitloses Werk, das in der Fachbibliothek eines Entwicklers seinen Platz verdient hat. Auf seinem Blog postet Jez Humble [Hum14] regelmäßig Aktuelles zum Thema.

▶Auf weniger als 150 Seiten gelang Paul Swartout in [Swart12] eine kompakte, aber durchaus umfassende Übersicht der Thematik. Die einzelnen Kapitel sind sehr von Managementmethodik geprägt, worin sich auch die langjährigen Projekterfahrungen Swartouts widerspiegeln.

Immer wieder schlägt er die Brücke vom technischen Ansatz hin zu den durchführenden Personen und deren Motivation. Frei nach dem Motto von Tom DeMarco „Menschen machen Projekte“ [Marc13].

Die Einsicht, dass moderne Softwareentwicklung mehr als das Ausliefern eines Installationspaketes ist, verdeutlicht bereits der Titel des Buches. Das Prädikat „lesenswert“ gilt hier nicht nur für Manager, sondern auch für all jene, die sich für das Warum interessieren.