Aus dem Leben für das Leben

Pragmatischer Ansatz modellgetriebener Softwareentwicklung

Hans-Jürgen Kempel, Klaus Masson

Effizient Software zu entwickeln, bedeutet möglichst gut auf die vorgegebenen Rahmenbedingungen der Organisation und der vorhandenen Ressourcen sowie der durchzuführenden Projekte einzugehen. In diesem Artikel möchten wir Kernpunkte eines pragmatischen Ansatzes vorstellen, mit dem in unserem Team seit einigen Jahren mit hoher Motivation Projekte erfolgreich abgewickelt werden.

Die Herausforderungen/Rahmenbedingungen

Der Schwerpunkt der ISO Software Systeme liegt auf der Entwicklung von Unternehmensanwendungen innerhalb der komplexen Domäne Touristik. Dabei handelt es sich sowohl um Produktentwicklungen in Zusammenarbeit mit firmeninternen Domänenexperten als auch um kundengetriebene Projekte, mit Laufzeiten von wenigen Wochen bis zu mehreren Jahren. Es sind bis zu vierzig Entwickler bei der Projektentwicklung beteiligt, die auf vier regional getrennte Entwicklungsstandorte verteilt sind.

Technische Rahmenbedingungen sind unter anderem die Integration der Produkte in die jeweilige kundenspezifische Infrastruktur mit unterschiedlichen Plattformen, Applikationsservern und verschiedenen schon bestehenden Systemen, wie beispielsweise einer SAP-basierten Buchhaltung oder ein bestehendes Reservierungssystem. Die Anwenderinteraktionen werden üblicherweise über browserbasierte Oberflächen in einem firmeninternen Intranet oder als B2C/B2B-Internet-Anwendungen realisiert. Je nach Kunde kann die Anwenderanzahl von zehn bis weit über tausend schwanken. Zeitgleiche Zugriffsraten von mehreren Hundert Anwendern sind bei Touristikanwendungen, welche über das Web angeboten werden, nicht ungewöhnlich.

Aus vertrieblicher Sicht war die Neuentwicklung einer Produktlinie für ein Bausteinsystem, die „DynamicTravelComponents“ (DTC), gefordert, mit dem kundenindividuelle Lösungen durch die Kombination von Anwendungskomponenten erstellt werden können. Neue Vertriebsideen sollen dabei möglichst schnell durch einen neuen Produktbaustein oder einen Prototypen am Markt platziert werden können.

All diese Herausforderungen beeinflussten den für die ISO „maßgeschneiderten“ Entwicklungsprozess entscheidend.

Der Entwicklungsprozess

In Abbildung 1 sind die Phasen des ISO-Entwicklungsprozesses dargestellt. Pro Entwicklungsabschnitt (Inkrement) werden die einzelnen Prozessschritte der Reihe nach durchlaufen. Dabei sind die einzelnen Prozessschritte innerhalb der Iterationen unterschiedlich ausgeprägt oder können auch komplett entfallen (z. B. ist eine Geschäftsprozessanalyse nicht immer erforderlich).

„Golden Rules“ des ISO-Entwicklungsprozesses

- **„Es kann nur ein Projekt-Repository geben…“**
  „The one and only – the project repository“

Aufgrund der vorgestellten Rahmenbedingungen wurde der modellgetriebene Softwareentwicklungsprozess als das für uns am besten geeignete Konzept ausgewählt. Dabei existiert für jedes Projekt genau ein UML-Modell-Repository. In dessen Strukturierung (s. Abb. 2) spiegeln sich die einzelnen Entwicklungsphasen und deren Modellierungsergebnisse wider. Die Repository-Struktur wurde im Zuge der bereits erfolgreich durchgeführten Projekte entwickelt und ihr Anwendungsbereich dabei kontinuierlich verifiziert und weiter verfeinert. Positiver Effekt der gleichartigen Struktur aller Repositorys ist, dass die Mitarbeit in mehreren Projekten oder ein Wechsel in ein anderes Projekt erleichtert wird. Der Aufbau eines neuen Modell-Repositorys ist durch das Importieren von Schablonen, z. B. für die Ordnerstruktur, domänenspezifische Diagrammattribute oder Transformatoren, sehr einfach. In die Schablonen ist das Know-how der bereits ausgearbeiteten Projekte eingeflossen. Sie werden durch eine Export-/Import-Funktionalität

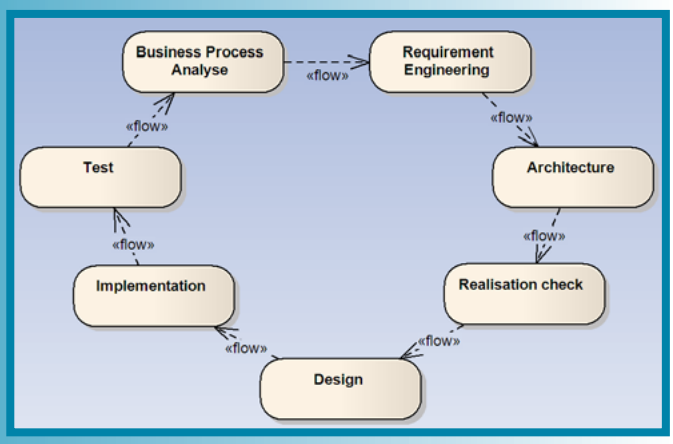

Abb. 1: Phasen des Entwicklungsprozesses

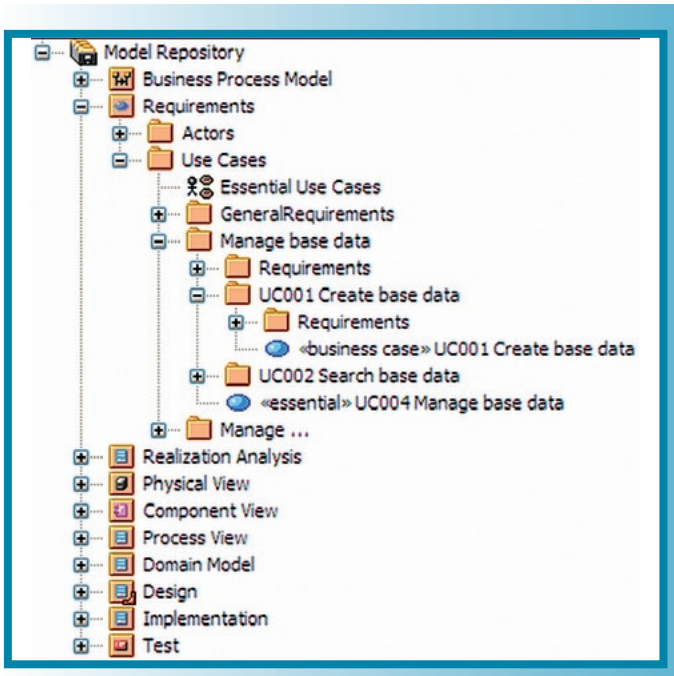

Abb. 2: Struktur des UML-Modell-Repositorys

des eingesetzten UML-Werkzeugs „Enterprise Architect“ (von Sparx Systems) erzeugt bzw. verwendet.

Alle Projektbeteiligten haben Zugriff auf das zentrale Repository und pflegen Informationen aus ihrem jeweiligen Arbeitsbereich in Form von Modellen, Diagrammen, Anwendungsfällen (Use Case) und Anforderungen (Requirement) ein. Synchronisations- und Locking-Konzepte erlauben dabei ein gleichzeitiges Modellieren innerhalb eines Repositorys durch mehrere Bearbeiter.

Ein zentraler Ablageort für sämtliche Projektinformationen der einzelnen Entwicklungsphasen hat sich dabei als sehr vorteilhaft erwiesen. Der Softwarearchitekt hat beispielsweise in der Designphase die Anforderungen aus der Systemspezifikation oder die Ergebnisse der Geschäftsprozessmodellierung unmittelbar im Zugriff. Natürlich können die Objekte der Modellierung auch über die Projektphasen hinweg miteinander verknüpft werden. So ist es möglich, sich in der Testfallmodellierung auf die Anforderungen der Systemspezifikation zu beziehen (s. Abb. 6). Dieses Vorgehen ermöglicht dann eine automatisierte Ermittlung der Testüberdeckung, bezogen auf die Systemspezifikation einzelner Anwendungsfälle, Pakete oder der kompletten Anwendung.

◆ „Modellierung ja, aber mit Herz und Verstand“ „Keep your model simple“

Die UML bietet mit ihren Diagrammen und ihrer Syntax ein gutes Werkzeug für das Modellieren und Dokumentieren in jeder Projektphase. Dennoch heißt es, genau abzuwägen, welche Diagramme und Dokumente erstellt werden sollen. So sind Beschreibungen und Dokumentationen auf Methodenebene erfahrungsgemäß mit einem hohen Aufwand verbunden, nach kurzer Zeit veraltet und durch den Blick in das Designmodell oder den Quellcode unnötig. Hier gilt es, möglichst keine redundanten Informationen zu generieren und abzuwägen, welche Diagramme tatsächlich als Dokumentation oder Diskussionsgrundlage benötigt werden.

Dabei haben wir in unserem Team nicht den Anspruch, jedes UML-Diagramm hundertprozentig UML-konform auszuarbeiten. Im Fokus steht immer das Ziel, eine gute Dokumentation oder Diskussionsgrundlage zu erstellen, die vom jeweiligen Abnehmer (im Normalfall ein weiteres Teammitglied oder der Kunde) verstanden wird. Um hier keine Missverständnisse aufkommen zu lassen: Die UML soll dabei nicht als simples Malwerkzeug missbraucht werden, jedoch versuchen wir, uns Diskussionen über die korrekte Pfeilspitzenmodellierung zu ersparen.

◆ „Und er weiß doch, was er möchte…“ „The river of constant change – the requirements“

In Anlehnung an die Grundsätze der agilen Softwareentwicklung versuchen wir, die Systemspezifikation in enger Zusammenarbeit mit dem Kunden zu realisieren, das heißt, die nicht immer klaren Vorstellungen des Kunden in bedienfreundliche und effiziente Konzepte zu formen. Sehr hilfreich sind für die initiale Klärung Workshops, in denen gemeinsam mit dem Kunden, im Optimalfall auch den späteren Anwendern, Informationen gesammelt werden. Diese können anschließend in Anwendungsfällen und Anforderungsbeschreibungen interpretiert und erneut mit dem Kunden abstimmt werden.

Erfahrungsgemäß sind trotz intensiven Einbezugs des Kunden in die Systemspezifikation Änderungswünsche nach den ersten kundenseitigen Tests der Software nicht ungewöhnlich. Hier hat sich die iterative Entwicklung der Anwendung in mehrwöchigen Iterationsabschnitten mit jeweils frühzeitiger Präsentation der aktuellen Entwicklung auf einem Snapshot-Server als besonders hilfreich erwiesen. Dieses Konzept ermöglicht es, ein frühzeitiges Feedback des Kunden zu erhalten und noch darauf reagieren zu können.

◆ „Die unendliche Geschichte, oder die Systemspezifikation…“ „How to handle the never ending story“

Auf die Systemspezifikation sollte in jedem Projekt ein besonderes Augenmerk geworfen werden. Erfahrungsgemäß wird diese spätestens nach Fertigstellung der initialen Spezifikation vernachlässigt, was in sämtlichen folgenden Phasen zu Problemen führen kann. Wie kann beispielsweise die Qualität einer Applikation durch Tests gesichert werden, wenn die Spezifikation des Verhaltens der Anwendung, und somit der zu testende Zustand, veraltet ist? Der modellgetriebene Ansatz bietet hier eine gute Möglichkeit, die Systemspezifikation über die Entwicklungszeit mit dem Projekt wachsen zu lassen.

Es hat sich in unseren Projekten bewährt, jede Änderung des Systemverhaltens vor deren Design und Implementierung in den Anwendungsfällen und Anforderungen einzuarbeiten. Dieses Vorgehen bietet die Chance, mögliche Konflikte mit schon existierenden Anforderungen bereits in der Spezifikationsphase zu vermeiden und vor der Implementierung im Zusammenhang mit der neuen Anforderung stehende Szenarien zu betrachten.

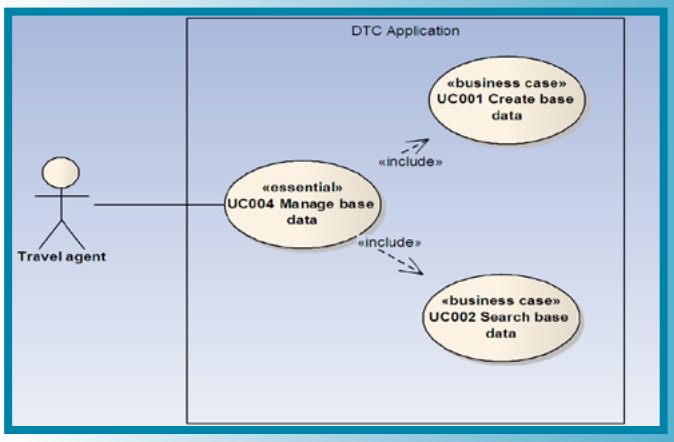

Abb. 3: Use-Case-Modellierung

◆ „Softwarearchitektur – dein Freund und Helfer“ “The invisible guide – the software architecture”

Eine gute Architektur unterstützt den Entwickler in seiner täglichen Arbeit. Dies erreicht man, indem unnötiger Overhead und Ballast durch geeignete Entwurfsmuster gekapselt und vom Entwickler ferngehalten werden.

Betrachten wir eine Serviceschicht, die einer GUI oder einem Webservice den Zugriff auf die Geschäftslogik ermöglicht. Neben dem Servicegedanken verstecken sich weitere Architekturkonzepte in dieser Schicht. So wird darin eine Datenbanktransaktion gestartet und geschlossen oder die Berechtigung eines Servicenutzers überprüft. Durch das Definieren der Objekte dieser Schicht in einem Design-Modell und dem Generieren des dazugehörigen Codes wird sichergestellt, dass projektweit diese Architekturvorgaben ohne Qualitätsverlust verwendet werden. Auf diese Weise wird ein Know-how-Transfer aus den Erfahrungen vergangener in neue Projekte realisiert, unter Einhaltung der Architekturvorgaben.

Jedes noch so ausgefeilte Architekturkonzept sollte auch durch entsprechende Modellierungen einfach und verständlich kommuniziert werden können. Wenn der Architekt in der Lage ist, seine Ideen beispielsweise in Diagrammen dem Team vorzustellen und eine Diskussionsgrundlage zu bieten, können seine Konzepte auch erfolgreich umgesetzt werden. Gibt es vergleichbare Anforderungen in weiteren Projekten, können

die bestehenden Konzepte leicht betrachtet und deren Eignung evaluiert werden. Da Architekturkonzepte üblicherweise mittels UML-Diagrammen beschrieben werden, bietet der modellgetriebene Ansatz optimale Voraussetzungen, diese zu erstellen und an geeigneter Stelle ablegen zu können.

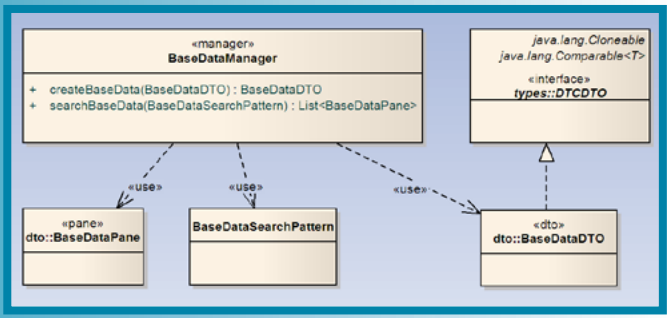

Abb. 4: Klassenmodellierung in der Design-Phase

„Generieren geht über Codieren"
„Generate as much as you can"

Die JEE-Plattform bietet gute Konzepte und Komponenten für die Entwicklung verteilter Unternehmensanwendungen. Hat man die für sich geeignete Architektur und die beteiligten JEE-Komponenten definiert, ist es sehr hilfreich, statischen, sich wiederholenden Code („boilerplate code") zu identifizieren. Verfolgt man einen modellgetriebenen Ansatz, dann liegt es nahe, diesen Code zumindest teilweise aus einem Design-Modell heraus zu generieren. Abbildung 7 zeigt die Architektur mit ihren beteiligten Komponenten. Farblich gekennzeichnet sind die Komponenten, welche durch Transformationen aus dem Modell heraus teilweise oder vollständig generiert werden.

Ziel dieses Vorgehens ist es, das Codieren von Infrastruktur-Code vom Entwickler möglichst fernzuhalten, damit er sein Hauptaugenmerk auf die Erstellung der Geschäftslogik richten kann.

Wurde in der Design-Phase vermieden, Details einer spezifischen Technologie einfließen zu lassen, kann die Einführung einer neuen Technologie durch das Erweitern oder Austauschen der Code-Generatoren erheblich vereinfacht werden.

```
/**
 * Store and return a BaseData object
 * @return the BaseData object
 * *** generated by rule ImplFromDesign.Operation__EJBInterface ***
 * @ejb.interface-method  view-type="both"
 *
 * @param baseDataDTO
 * @param ctx    call context
 * @throws ISOException if call fails throws this exception
 */
public BaseDataDTO createBaseData(BaseDataDTO baseDataDTO, CallContext ctx)
  throws ISOException{
    Session session = HibernateUtils.getNewBeanSession();
    BaseDataManager manager = new BaseDataManager(session);
    Transaction tx = null;
    BaseDataDTO result = null;
    try {
        tx = session.beginTransaction();
        result = manager.createBaseData(baseDataDTO, ctx);
        tx.commit();
    }
    catch (org.hibernate.exception.ConstraintViolationException e) {
        ConstraintViolation violation = new ConstraintViolation(e.getConstraintName());
```

Abb. 5: Beispiel eines generierten Codes – Service-Schicht

Neben dem Generieren von Quellcode ist auch das Erstellen von Dokument-Artefakten aus dem Modell eine große Hilfe. So ist es möglich, projektübergreifende Dokument-Schablonen zu definieren. Diese können von den einzelnen Projekten wiederverwendet werden, um beispielsweise die Systemspezifikation in ein Word-Dokument zu exportieren. Die benötigten fachlichen Informationen werden somit ohne Redundanz an einer zentralen Stelle verwaltet und können bei Bedarf in verschiedene, auf den Abnehmer der Dokumente abgestimmte Dokumente exportiert werden.

„Automatisierte Tests gut finden kann jeder…"
„Don't leave your window broken…"

Die Wichtigkeit von Tests bei der Entwicklung und Qualitätssicherung von Software in einem großen Team ist bekannt. Im Folgenden möchten wir Hinweise geben, wie diese möglichst reibungslos in den täglichen Arbeitsablauf integriert werden können.

Als sehr hilfreich erwiesen hat sich das Etablieren eines projektspezifischen Continuous-Integration-Systems, zeitnah zum Start eines neuen Projekts. So kann die regelmäßige Ausführung der Tests, die optimalerweise sehr früh in der Projektlaufzeit entstehen, sichergestellt werden. Nach einem Check-in kann der Entwickler individuell über die Web-Konsole des Continuous-Integration-Systems den Build des Gesamtsystems auf einem Integrationsserver starten, hierfür verwenden wir Hudson. Insbesondere bei Arbeiten an Basiskomponenten, die von mehreren Projekten verwendet werden, können die nie auszuschließenden Seiteneffekte einer Änderung durch dieses Vorgehen sofort ohne großen Aufwand überprüft werden.

In den lokalen Entwicklungsumgebungen kommen Referenz-Tools, wie beispielsweise der Applikationsserver JBoss, zum Einsatz, da die lokale Entwicklung mit den auf dem Zielsystem eingesetzten Produkten oftmals aus ressourcen- oder lizenztechnischen Gründen nicht möglich ist.

Durch den Einsatz dieser Integrationstestumgebung, die optimalerweise an die Konfiguration der Ziel-Produktivumgebung angepasst ist, erhält der Entwickler schnell Information, ob seine Modifikationen nicht nur in seinem lokalen Entwicklungssystem, sondern auch im Zielsystem funktionstüchtig sind. Werden in der Integrationstestumgebung dann fehlerhafte Tests angezeigt, ist es in der Verantwortung des Entwicklers, schnellstmöglich die Ursache zu ermitteln und zu beseitigen. Dies wird mit einer hohen Disziplin verfolgt, da sich zu einem fehlerhaften Test erfahrungsgemäß schnell weitere fehlschlagende Tests ansammeln, für die sich kein Entwickler verantwortlich fühlt und neue Fehler dann schnell in der Masse verschwinden („broken windows theory").

Damit Änderungen von jedem Entwickler vor einem Check-in umfassend lokal getestet werden können, verfügen diese jeweils über eine eigene unabhängige Entwicklerdatenbank. Jedes Projekt bietet Standardmethoden, um eine Entwicklerdatenbank neu zu generieren und anschließend mithilfe der Anwendung automatisiert Testdaten zu erzeugen.

Die Code-Qualität und Einhaltung von Coding-Conventions wird durch Tools für statische Code-Analyse wie

Checkstyle und Findbugs sichergestellt. Gemäß dem Motto „Vertrauen ist gut, Kontrolle ist besser“ wird der Code vor dem Einchecken in ein Subversion-Repository geprüft und nur bei Einhaltung der definierten Konventionen von diesem akzeptiert.

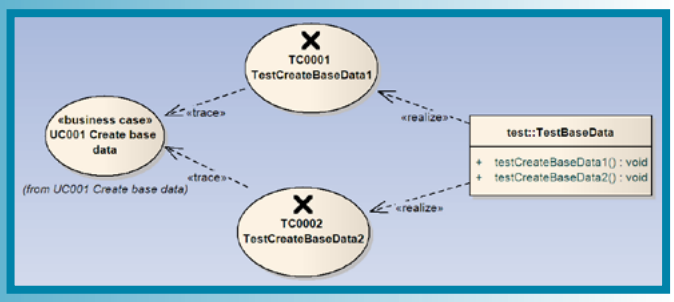

Abb. 6: Modellierung der Beziehung zwischen Use Case und Test Case

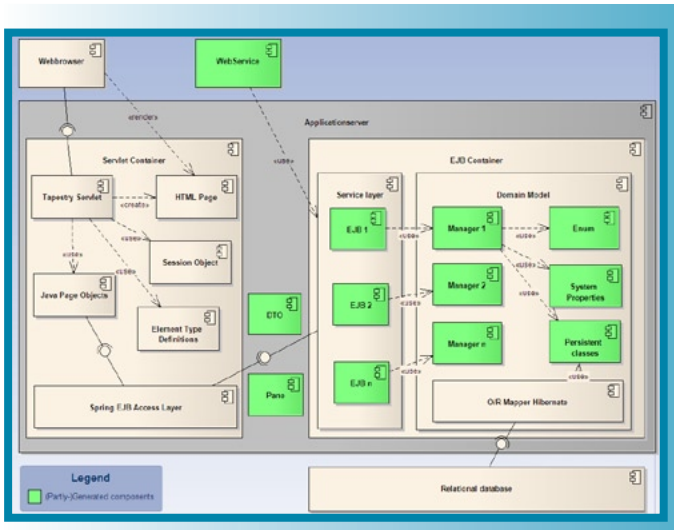

Abb. 7: DTC-Architektur mit Kennzeichnung der (teil-)generierten Komponenten

„Projektinfrastruktur: Ein Geben und Nehmen…“ „Consume, but contribute“

Der Aufbau und das Etablieren eines modellgetriebenen Entwicklungsprozesses kosten Ressourcen und entfalten erst nach einiger Zeit seine volle Wirkung. Erfolgreich damit wird nur der sein, der den Entwicklungsprozess auf Dauer etabliert.

Eine weitere Voraussetzung dafür ist jedoch auch die beständige Verbesserung des Prozesses hinsichtlich der verwendeten Methoden und Werkzeugen. In Zeiten sinkender IT-Budgets trifft dies nicht selten im Management auf Unverständnis, da die Akzeptanz nicht vorhanden ist, diesen Entwicklungsprozess wie einen „großen Garten“ beständig zu pflegen („constant gardening“). Das Vorgehen, den bestehenden Prozess „einzufrieren“ und vom bislang Erreichten zu profitieren, würde sicherlich kurz- bis mittelfristig auch Erfolge bringen, gefährdet aber langfristig das bisher Erreichte und die bisherigen Investitionen in den Prozess. Die Forderung einer Stabsstelle oder Entwicklungsgruppe, die den Prozess wartet und verbessert, wird aber unserer Auffassung nach dem Kern der Sache nicht gerecht. Zu groß erscheint uns die Gefahr, dass die Weiterentwicklung des Prozesses an den Anforderungen der Projekte vorbei geht.

Daher verfolgen wir bei unserem Ansatz einen anderen Weg. Die Prozessweiterentwicklung wird durch die Entwicklungsprojekte vorangetrieben und der dafür notwendige Aufwand von den Projekten selbst getragen.

Dadurch stellen wir sicher, dass sich der Entwicklungsprozess an den Anforderungen der Projekte orientiert und pragmatisch bleibt. Neue Techniken, ein geändertes Vorgehen in der Methodik oder neue Tools werden in einem Projekt exemplarisch erprobt und stehen nach Abschluss und Bewertung allen Entwicklungsprojekten zur Verfügung. Wir vermeiden mit diesem Vorgehen bewusst die Kostendiskussion zur Weiterentwicklung des Prozesses. Jedes Projekt, das vom erprobten Vorgehen im Entwicklungsprozess, den vorhandenen Arbeitsanweisungen und Schablonen profitiert, geht damit auch die Verpflichtung ein, etwas zur Weiterentwicklung des Prozesses beizutragen.

Ein weiterer positiver Effekt ist die hohe Motivation der involvierten Projektmitarbeiter, sich an der Weiterentwicklung des Prozesses zu beteiligen und ihn bei seiner Anwendung kritisch zu hinterfragen.

Ein derartiges Vorgehen ist so sicherlich nicht auf jede Organisation für die Softwareentwicklung anwendbar. Für unsere aktuelle Entwicklungssituation ist dieses Vorgehen sinnvoll und bewirkt die beschriebenen Effekte.

Fazit

Mithilfe des vorgestellten, kontinuierlich verbesserten Entwicklungsprozesses und den dazugehörigen „Golden Rules“ ist es uns bei einer hohen Motivation in unserem Team gelungen, die Effizienz und Qualität unserer Projekte zu steigern. Dennoch stellt er nur eine Momentaufnahme dar. Das Hinterfragen des Prozesses sowie die Suche und Einarbeitung von gezielten Optimierungen wird mit jedem folgenden Projekt weiter gehen. Als Ideengeber dienen dazu unter anderem Innovationen aus den klassischen und agilen Entwicklungsprozessen.

Links

[Hudson] NightlyBuild-Umgebung, `https://hudson.dev.java.net/`
[Selenium] GUI-Testwerkzeug, `http://seleniumhq.org/`
[Sparx] UML-Modellierungswerkzeug, `http://www.sparxsystems.de/`
[Tapestry] Komponentenbasiertes GUI-Framework, `http://tapestry.apache.org/`

Hans-Jürgen Kempel (Dipl.-Inform.) ist Senior-Projekt-Manager bei der ISO Software Systeme. Er ist mitverantwortlich für die bei ISO im Touristikbereich eingesetzten Technologien und Methoden und verfügt über langjährige Erfahrung in der Konzeption, Implementierung und Einführung von komplexen IT-Systemen.
E-Mail: hans-juergen.kempel@isogmbh.de.

Klaus Masson (M. Eng.) ist Software-Ingenieur bei der ISO Software Systeme. Er ist seit zehn Jahren in der Softwareentwicklung tätig und beschäftigt sich aktuell mit der Entwicklung verteilter Java-Anwendungen im Bereich Touristik.
Er ist verantwortlicher Softwarearchitekt der entwickelten JEE-„DynamicTravelComponents“ und der darauf basierenden touristischen Produkte der ISO Software Systeme.
E-Mail: klaus.masson@isogmbh.de.