Leuchter, S. & Urbas, L. (2004): Integrierter agiler Entwicklungsprozess für softwareintensive Mensch-Maschine-Systeme. In: C. Steffens, M. Thüring & L. Urbas (Eds), *Entwerfen und Gestalten. 5. Berliner Werkstatt Mensch-Maschine-Systeme (pp. 53-68).* Düsseldorf: VDI Verlag (Fortschritt-Berichte, Reihe 22; 16). http://www.safety-critical.de/doc/bwmms03.pdf.

Integrierter agiler Entwicklungsprozess für softwareintensive Mensch-Maschine-Systeme

Sandro Leuchter & Leon Urbas

Software-Engineering, Feature-Driven Development, agile, Objekt-Orientierung, Prozess

Zusammenfassung

Anhand des Prozessbegriffes wird die systemtechnische Entwicklungsmethodik PIE (parallel-iterative Entwicklung) vorgestellt, die den Erstellungsprozess interaktiver Systeme auf einer globalen Ebene in einen humanwissenschaftlichen und einen ingenieurwissenschaftlichen Strang mit definierten Schnittstellen trennt. Wir erweitern die PIE auf einer feingranularen Ebene um ein Prozessmodell aus dem agilen Software-Engineering (Feature-Driven Development) und erreichen so einen strukturierten, interdisziplinär handhabbaren und kommunizierbaren Entwicklungsprozess PIE/FDD, mit dem interaktive Systeme in Iterationsschritten nach ISO 13407 entwickelt werden können. PIE/FDD wird anhand der Erfahrungen aus der Entwicklung in innovativen Anwendungsfeldern diskutiert verglichen.

1 Einleitung

Die Entwicklung von komplexen technischen Systemen wird häufig als Projekt aufgefasst. DIN 69901 definiert den Begriff Projekt als ein „Vorhaben, das im wesentlichen durch Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist ...“. Das bezieht sich z.B. auf Zielvorgabe, Ressourcenbegrenzungen (Projektdauer, Mitarbeiter, Qualität) und die Organisationsform. Projekte sind demnach zeitlich abgeschlossen, es stehen zur Bearbeitung begrenzte Ressourcen zur Verfügung und eine spezielle Organisationsform muss gewählt werden, um ein konkretes Projekt im Rahmen einer oder mehrerer Organisationen umzusetzen. Zumeist besteht jedoch ein Risiko bezüglich der Erreichung aller Projektziele unter den gegebenen Ressourcenbeschränkungen. Um dieses Risiko fortlaufend bewerten zu können und den Einsatz der Ressourcen steuern zu können, sollte ein definierter Prozess, der das Vorgehen regelt, eingesetzt werden. Er dient ebenso zur Koordination innerhalb der Projektbearbeiter, die in unterschiedlichen Organisationen angesiedelt sein können, aber auch mit Externen, wie Auftraggebern oder Zielgruppe. Des weiteren wird dadurch möglich, ein effektives Controlling einzurichten und die Projektplanung an bekannten Projektphasen und Erfahrungswerten dazu auszurichten. Methoden, die auch von Projektplanungs- und -koordinationswerkzeugen angeboten werden, sind Projektstrukturpläne, Netzplantechniken und

Gantt-Charts, mit denen die einzelnen Phasen eines Projektes definiert, in Beziehung gesetzt und deren zeitlicher Verlauf geplant werden können.

In den nächsten Abschnitten werden unterschiedliche Klassen von Softwareentwicklungsprozessmodellen, die als Vorlage für die Struktur solcher Projekte dienen können, in Bezug sowohl auf die verteilte und interdisziplinäre Entwicklung, als auch in Bezug auf die Möglichkeit zur Risikominimierung bei innovativen Entwicklungsprojekten diskutiert.

Softwareentwicklungsprozesse

Für die Erstellung von Software wurden bereits früh spezifische immer wiederkehrende Teilschritte bzw. Projektstrukturen identifiziert (Analyse, Design, Programmieren, Test, Einführung). Die einzelnen Tätigkeiten können aber in unterschiedlichen Abläufen und Wiederholungen ausgeführt werden, wobei anzunehmen ist, dass es je nach vorgegebenen Zielgrößen jeweils ein besonders günstiges Vorgehen gibt, um die Arbeit in besonders großen Projekten zu koordinieren und den Entwicklungsprozess in sinnvolle Bahnen zu lenken. Insofern ist das Ziel des Software-Engineering, die geordnete Anwendung von Prinzipien, Methoden und Werkzeugen für die Produktion von qualitativ hochwertige Software. Die Definition der Blaupause für ein Vorgehensmodell, das in mehreren Softwareentwicklungsprojekten angewendet werden kann, ist ein Prozess. Prozesse werden benutzt, um die Wiederholbarkeit von Projekterfahrungen zu gewährleisten, neue Mitarbeiter einfacher in eine Organisation und in Abläufe einzubinden und allgemein zur Zielorientierung der Arbeitsschritte. Softwareentwicklungsprozesse können formal definiert werden, um einen deskriptiven oder einen normativen Zweck zu erfüllen. Normativ ist die Prozessdefinition eine Anleitung zur Durchführung von Teilschritten und Aufgaben einzelner Mitarbeiter. Aus ihr ergeben sich Vorgaben für nützliche Qualitätsmaße und Vorgehensweisen. Standards leiten die Arbeit. Das Vorgehen und sein Erfolg werden wiederholbar gemacht. Der deskriptive Zweck von Prozessmodellen ist es, ein gutes Vorgehen zu dokumentieren und dadurch innerhalb einer Organisation (oder auch nach außen) darüber zu kommunizieren. Die formale Beschreibung ermöglicht weiterhin automatische Analysen über Effizienz und Effektivität („Business Process Reengineering“) und einen quantitativen Vergleich mit konkurrierenden Methoden.

Zur Formalisierung von Prozessen bieten sich insbesondere zwei Beschreibungen (Prozessarchitekturen) an:

- Textuell: ETVX

Prozessmodelle beschreiben Tätigkeiten und ihre Abfolge anhand von definierten Phasen. Radice et al. [1] schlagen deshalb als textuelle Repräsentation eines Prozesses ein kapitelweises Vorgehen vor. Pro Kapitel wird eine Phase beschrieben. Jedes Kapitel besteht aus den Abschnitten „Entry“,

„Tasks“, „Verfication/Validation“, and „eXit“ (ETVX). Entry beschreibt die Bedingungen, Zwischenergebnisse und Ressourcen, die vorhanden sein müssen, damit der Prozess in die im Kapitel beschriebene Phase gehen kann, Tasks beschreibt die Aufgaben und Tätigkeiten der Phase und ordnet Kompetenzen und organisatorische Einheiten zu. Verification/Validation ist eine Menge von Hilfsmitteln und Methoden, die während der Abarbeitung der tasks helfen, zu überprüfen, ob die Arbeit richtig, bzw. gut genug verläuft. eXit umfasst alle Kriterien, die erfüllt sein müssen, damit die Phase als abgeschlossen betrachtet werden kann.

- Graphisch: IDEF0

Die Abfolge und Zusammenhänge zwischen den Phasen lassen sich insbesondere für zyklische Prozesse in einem Flussdiagramm nach IDEF0 [2] besonders übersichtlich darstellen (s. Abbildung 1). Weiterhin können zusammengehörende Gruppen von Vorgängen bzw. Teilgraphen zu „Makros“ aggregiert werden.

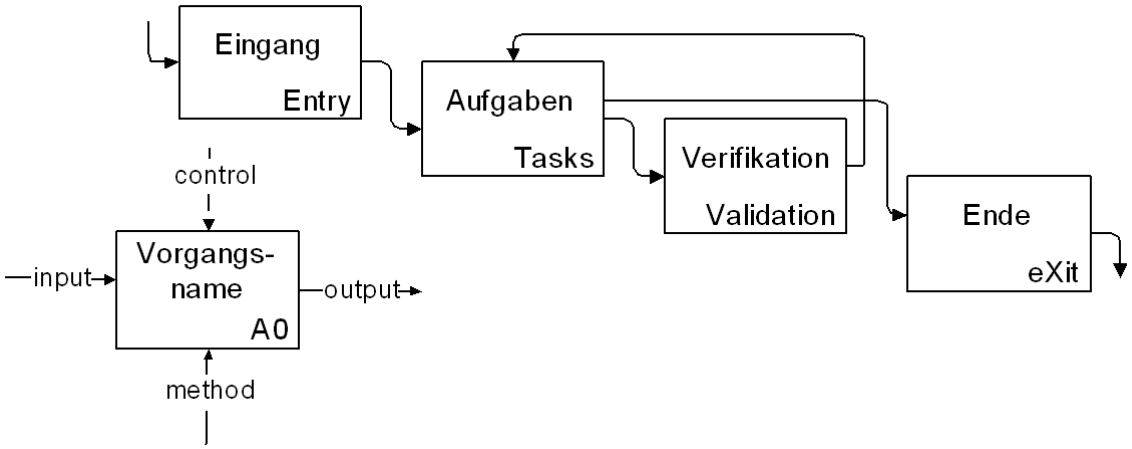

Bild 1: Prozessarchitektur IDEF0 stellt ETVX dar

Es gibt viele eingeführte Prozessmodelle, die jeweils unterschiedliche Zielstellungen abdecken. Ein und derselbe Prozess lässt sich jedoch auch sehr unterschiedlich mit Prozessarchitekturen darstellen. Je nach Detaillierungsgrad der Darstellung unterscheidet man zwischen *universal*, *worldly* und *atomic*. Ein universal dargestelltes Prozessmodell beinhaltet nur eine Aufzählung der Phasen und Aufgaben und ihre Sequenz. Eine worldly Definition beinhaltet zusätzlich noch Informationen über erwartete Resultate, anwendbare Maße und Bedingungen für Checkpoints. Atomic spezifizierte Prozesse sind dagegen noch weiter – nahezu algorithmisch – formalisiert.

Sowohl die Darstellung eines Prozesses, als auch das Vorgehensmodell selbst kann unterschiedlich aufwändig gestaltet sein. Es können folglich schwergewichtige Prozesse, die zumeist auch nahezu atomic spezifiziert werden, und agile, die sich zumeist auch mit einer worldly Darstellung begnügen, unterschieden werden. Beispiele für schwergewichtige Prozesse sind der

Personal Software Process/Team Software Process [3, 4], der Rational Unified Process und das V-Modell [5]. Die agilen Prozesse (z.B. Extreme Programming [6], Feature Driven Development [7] und Chrystal [8]) unterscheiden sich von den schwergewichtigen durch folgende Schwerpunktsetzungen:

- Frühe und fortwährende Lieferung von Prototypen
- bei kurzen Lieferzyklen,
- laufende Software ist das wichtigste Fortschrittsmaß,
- Anforderungsänderungen sind jederzeit erwünscht,
- Einfachheit der technischen Lösung auch bei Verminderung der Wiederverwendbarkeit wird bevorzugt,
- Betonung der Bedeutung des Entwicklungsteams und
- dessen Entwicklung durch regelmäßige Reflektion.

Gerade die Entwicklung innovativer Produkte ist mit einem hohen Erfolgsrisiko verbunden. Hier sind agile Prozesse mit ihren kurzen Feedback-Zyklen besonders geeignet, Fehlentwicklungen zusammen mit dem Auftraggeber bzw. Kunden frühzeitig zu erkennen und das Risiko jederzeit realistisch einschätzbar zu machen. Dadurch und durch die Betonung der dynamisch adaptierbaren Zielvorgabe ist es möglich, realistisch umzuplanen und ggf. bessere Lösungen zu entwickeln, wenn sich Hindernisse bei der ursprünglichen Konzeption herausstellen. Agile Entwicklungsmethoden lassen sich auf der anderen Seite nur in Projekten einsetzen, in denen die intensive und regelmäßige Kommunikation mit dem Auftraggeber auch gegeben ist.

Die Entwicklungsteams können jedoch bei einem agilen Vorgehen nicht beliebig groß werden. Als durchschnittliche Teamgröße werden neun Entwickler genannt, in der Praxis liegt die maximale Gruppengröße bei ca. 50 Mitarbeitern.

Die Fokussierung der Systementwicklungsarbeit auf die Softwareerstellung ist jedoch besonders bei interaktiven „technischeren“ Systemen (verkehrstechnische Systeme, Werkzeugmaschinen, Prozesssteuerungen, andere *embedded systems*) eine Einschränkung, weil offensichtlich viele Teilschritte in der Gesamtentwicklung nicht die Programmierung direkt betreffen. Einen Prozess zu benutzen, der aber genau die Modelle, Konstrukte und Fragestellungen der Programmierung in den Mittelpunkt stellt, degradiert alle anderen beteiligten Disziplinen zu Zuarbeitern. Im folgenden Abschnitt wird deshalb ein *universal* Prozess dargestellt, der die interdisziplinäre Entwicklung von Mensch-Maschine-Systemen zum Gegenstand hat.

Parallel-iterative Entwicklung

Die parallel-iterative Entwicklung (PIE [9]) beschreibt einen Entwicklungsprozess, der besonders für die Erstellung von innovativen industriellen Automationsanwendungen in Mensch-Maschine-Systemen geeignet ist, denn es wird insbesondere die Entwicklung der Funktionsteilung als Strukturierungsmittel benutzt (s. Abbildung 2).

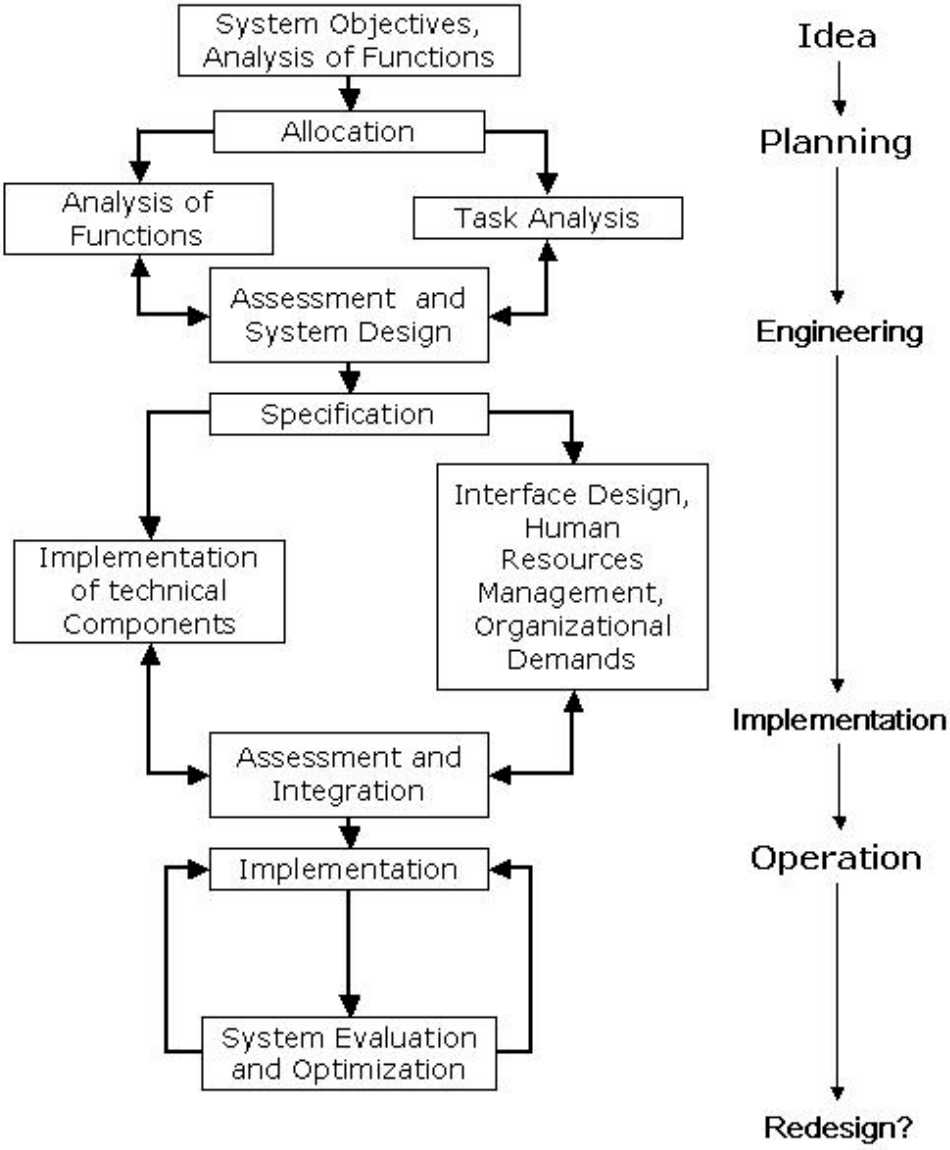

Bild 2: : Parallel-iterative Entwicklung [9]

Die Entwicklung wird in einen human- und einen ingenieurwissenschaftlichen Zweig getrennt. An den Anfang wird eine allgemeine Definition des zu erstellenden Systems gesetzt. Die Grenzen werden festgelegt und die grundlegenden Anforderungen erhoben. Im Herkunftsbereich dieses Prozessmodells ist die Funktionsteilung zwischen Mensch und Maschine, die Automatisierung der Tätigkeit besonders wichtig. Sie trennt die Arbeit in den technischen und den humanwissenschaftli-

chen Teil der Spezifikation, deren Ergebnisse in der nächsten Phase zusammengeführt und bewertet werden. Es folgt die Realisierung, die wiederum in zwei parallelen Strängen erfolgt, in denen zum einen technische Komponenten, zum anderen die Benutzungsschnittstellen und die organisatorische Einbettung entwickelt werden. Auch diese Ergebnisse werden am Ende der Phase zusammengeführt. Anschließend wird das System in mehreren Evaluationsschritten realisiert und installiert. Der Lebenszyklus endet mit Systemeinstellung oder -wechsel.

Die vielfältigen Iterationen und Rückkopplungen, die während der Entwicklung möglich und nötig sind, sind in dem Phasenschema in Abbildung 2 nur angedeutet: Nach jeder Bewertung können Änderungen an der vorigen Phase erforderlich sein. Das ist insbesondere wegen der Integration der teilweise widersprüchlichen Ergebnisse der beiden parallelen Stränge erforderlich.

Der PIE-Prozess zeichnet sich durch klar getrennte und planbare Phasen aus. Durch die disziplinäre Trennung der Phasen ist eine leichtere Koordination zwischen den Beteiligten möglich. In jeder Phase ist Benutzerbeteiligung möglich oder vorgesehen. Technologische und Human-Factors-Aktivitäten laufen parallelisiert ab. Evaluation und Bewertung sind integrierte Bestandteile des gesamten Erstellungsprozesses. Das iterative Vorgehen innerhalb der Phasen ermöglicht eine frühzeitige und realistische Erfolgskontrolle.

Der Prozess ist jedoch nicht innerhalb der Phasen genau spezifiziert. Es ist deshalb erforderlich, die einzelnen Aktivitäten anhand von *best practices* und anderen Modellen zu verfeinern. Im folgenden Abschnitt stellen wir die agile Software Engineering-Methode „Feature-Driven Development“ vor, die wir benutzen, um die PIE zu konkretisieren.

Feature-Driven Development

Die Feature-Driven Development (FDD) Methode von Coad et al. ([7], S. 182 ff) beschreibt einen agilen Softwareentwicklungsprozess, der mit besonders wenig administrativem Aufwand verbunden ist. Die wichtigste Planungsgröße ist das Feature (s. Tabelle 1). Ein Feature beschreibt ein Stück Funktionalität des Gesamtsystems, das „in den Augen des Kunden nützlich ist“ – es geht nicht um Architektureigenschaften. Die Größe der Features sollte so gewählt sein, dass es in weniger als zwei Wochen implementiert werden kann. Mehrere Features bilden ein Feature-Set, die wiederum zu Major Feature-Sets zusammengefasst werden.

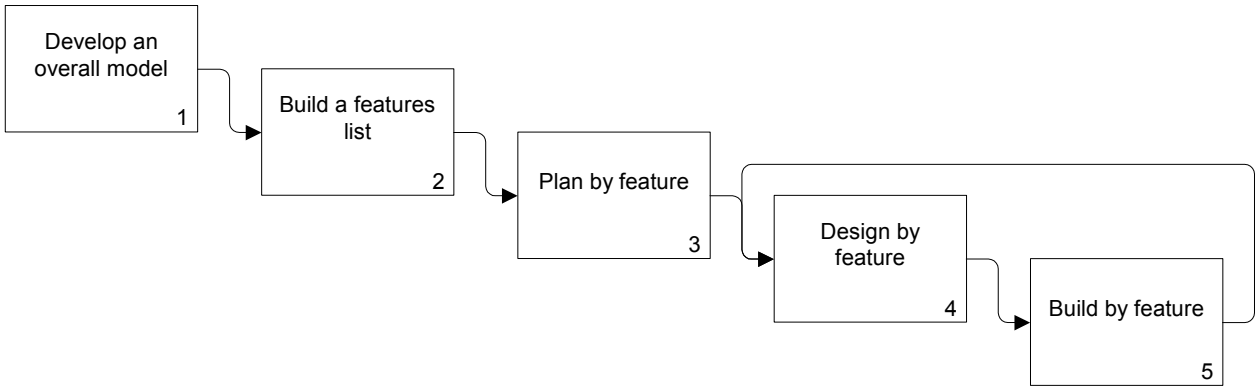

Bild 3: Phasenschema Feature-Driven Development (nach [7])

Die Softwareerstellung orientiert sich an fünf Phasen (zur Abfolge der Phasen s. Abbildung 3 und zur genaueren Phasenbeschreibung Tabelle 2):

1. *Develop an overall model:* Diese Phase nimmt etwa 14% der gesamten Entwicklungszeit in Anspruch. Auf der Basis von zugänglichen Dokumenten wird ein Objektmodell für die Domäne erstellt. Das Ergebnis ist ein UML-Diagramm (statisch) und eine informelle Featureliste. Designalternativen werden protokolliert.
2. *Build a features list:* Features, Feature-Sets und Major Feature-Sets (s. Tabelle 1) werden identifiziert und ggf. zusammen mit dem Auftraggeber priorisiert. Diese Phase nimmt etwa 5% der Entwicklungszeit in Anspruch.
3. *Plan by Feature:* Anhand der Anforderungen (Objektmodell und Featureliste) wird der Programmierablauf in etwa 4% der Gesamtdauer geplant: Zeitdauern werden geschätzt, Termine und Zuständigkeiten werden definiert.
4. *Design by feature (DBF):* In diesem Schritt wird für ein bestimmtes Feature ein Sequenzdiagramm erstellt, aus dem hervorgeht, welche Klassen an der Interaktion beteiligt sind. Es wird darauf aufbauend ein Entwurf für neue Methoden und andere Änderungen an allen beteiligten Klassen erstellt.
5. *Build by feature (BBF):* Das Design für ein Feature wird umgesetzt, Unit-Tests werden durchgeführt, die getesteten Änderungen werden in das Konfigurationsmanagementsystem eingecheckt – das Feature ist abgeschlossen.

Die Phasen *design by feature*/*build by feature* werden iterativ für alle Features durchgeführt. Hier findet die eigentliche Systemerstellung dar. Diese Tätigkeit nimmt etwa 77% der Zeit ein. Eine Iteration dauert etwa zwei Wochen. Alle zwei Wochen steht damit eine aktuelle Systeminstanz mit

allen bis dahin umgesetzten Features zur Verfügung. Zu einem Zeitpunkt wird in mehreren Teams an mehreren Features in DBF/BBF-Zyklen gearbeitet.

Tabelle 1: Charakterisierung der Elemente des FDD

Element	Eigenschaften
Feature	Nützlich für den Kunden/Benutzer Granularität: in weniger als zwei Wochen implementiert ➔ „<action> the <result> <by|for|of|to> a(n) <object>“
Feature-Set	Strukturierend auf der Interaktionsebene ➔ „<action><-ing> a(n) <object>“
Major Feature-Set	Strukturierend auf Modul/Komponenten-Ebene ➔ „<object> management“

Die Entwicklung erfolgt in mehreren Rollen: Neben administrativen Funktionen in den ersten drei Phasen werden in den DBF/BBF-Zyklen Feature Teams aus jeweils einem Chief Programmer und mehreren Class Owners gebildet. Jedes Feature wird während der Planung einem Chief Programmer zugeordnet, dessen Aufgabe es ist, den korrekten Entwurf und Implementierung von Klasseninteraktion für sein Feature zu koordinieren. Die eigentliche Umsetzungsarbeit passiert an den Klassen. Jede Klasse ist genau einem Programmierer zugeordnet. Für die Dauer der Arbeit an einem Feature (DBF/BBF-Zyklus) bilden alle beteiligten Class Owners und der zuständige Chief Programmer das Feature Team. Class Owners können zur selben Zeit zu mehreren Feature Teams gehören. Die Aufgabe der Chief Programmers ist die Koordination untereinander und die Qualitätssicherung des Codes des Feature Teams, sowie dessen Fortbildung.

Tabelle 2: Prozessbeschreibung (ETVX) FDD

Phase	Entry	Tasks	Verification	eXit
1) Develop an overall model	Auftrag, Anforderungen liegen vor	*Domain walkthrough*, informelle Featureliste erstellen, Modell erstellen, Alternativen dokumentieren	Erfolgreiche interne und externe Bewertung des Modells	Klassendiagramme, informelle Featureliste, Dokumentation der Designentscheidungen
2) Build a features list	Modell ist entwickelt	Identifikation von Features anhand Tabelle 1, Priorisierung, Aufteiler zu großer Features in kleinere	Interne und ggf. externe Bewertung durch Domain-Experten	Detaillierte Featureliste (unterteilt in Sets und Major Sets), Review durch Entwicklungsleitung
3) Plan by feature	Featureliste ist erstellt	Sequenzieren von (Major) Feature-Sets, *class owners* & *chief programmers* festlegen	Interne und ggf. externe Bewertung durch Management	Entwicklungsplan: Festlegung der Termine und *class owners*
4) Design by feature	Plan ist erstellt	Ggf. *domain walkthrough,* Sequenzdiagramme, Methodenrümpfe erstellen, Design-Inspetkion, Alternativen dokumentieren	Interne und ggf. externe Bewertung durch Featureteam	Feature + Sequenzdiagramm, Klassendiagrammänderungen, Methodemrümpfe, Dokumentation der Designentscheidungen: Review durch Entwicklungsleitung
5) Build by feature	Design für Umsetzung der Features für diesen Zyklus liegt vor	Implementierung Klassen und Methoden, Code-Inspektion, Unit-Test, Check-In (*class owner*), Build des Gesamtsystems	Featureteam führt Code-Insepektion und Unit-Test durch	Implementierte Methoden und Testmethoden, Ergebnisse der Unit-Tests (jede Methode einzeln + Gesamtsequenz), eingecheckte Klassen

PIE/FDD

Weil PIE einen universalen (Makro-) Prozess definiert, ist es zur Anwendung in der Erstellung interaktiver Software notwendig, die einzelnen Phasen weiter zu konkretisieren.

Innerhalb der Phasen ist dafür eine stärkere Strukturierung und Tätigkeitsleitung nötig. Dazu favorisieren wir die Anwendung eines agilen Prozesses, weil so die Interaktion zwischen Evaluatoren und Softwareentwicklern besonders gut abgebildet werden kann. Die Auswahl von FDD als konkretem agilen Software Engineering Prozess ist in der guten interdisziplinären Kommunizierbarkeit von Features und dem geringen administrativen Aufwand der Methode begründet. Außerdem bietet die Featurenotation eine günstige Richtlinie für die Benennung von Use-Cases.

Im folgenden wird die konkrete Ausgestaltung von PIE/FDD beschrieben. Am Anfang der Arbeit muss die Zielgruppe, der Systemzweck und die Grenzen des zu betrachtenden Systems

festgelegt werden. In der zweiten Phase werden auf der einen Seite die technischen Funktionen (Fetaures) definiert und auf der anderen Seite die Benutzeranforderungen, -erwartungen und -fähigkeiten anhand von Befragungen oder anderen Aufgabenanalysen erhoben. Sie leiten den Entwurf von Funktion und Interaktionsprinzipien. Der Abgleich dieser Ergebnisse resultiert in einem Detailentwurf anhand von Benutzungsszenarien in der Form von Use-Cases, Klassenmodell und Featureliste. Damit kann der FDD-Schritt „plan by feature“ durchgeführt werden. Der nächste parallele Schritt ist die feature-getriebene Implementierung (DBF/BBF) der Architektur und Basisfunktionalität des technischen Systems, bzw. der Entwurf von Interface Designs mit Klickmodellen.

Das Konfigurationsmanagement wird mit cvs umgesetzt: Compilierbarer Code darf eingecheckt werden, wenn keine Seiteneffekte auftreten. Fertige Features (komplette DBF/BBF-Zyklen) werden mit einem Tag im Repository gekennzeichnet.

Sowohl technische Funktionen, als auch die Interaktion werden getestet: Die technische Entwicklung mit Unit-Tests, das Interaktionsdesign mit formativen Benutzertests an Mock-Ups und Prototypen. Nachdem die Interaktion festgelegt ist, werden Oberflächen programmiert und mit den vorhandenen Funktionen verbunden. Je nach zugrundeliegender Softwarearchitektur ist die Frequenz der Implementierungs-/Bewertungszyklen unterschiedlich zu gestalten, denn für eine summative Evaluation eines Prototypen muss der Funktionszuwachs und die Bedienbarkeit von Interaktionskonzepten der Zwischenstände eine kritische Masse erreichen. Die Evaluationsergebnisse fließen als neue Priorisierung und ggf. Neudefinition oder Änderung von Features in den Entwicklungsprozess zurück. Hier übernimmt die „Evaluationsabteilung“ die Funktion der Kunden in der agilen Softwareentwicklung. Durch die Featureänderungen wird eine Neuplanung der Ressourcen und Termine in der FDD-Phase „plan by feature“ erforderlich. Ergebnis ist eine neue Roadmap, aus der Release-Termine für Featuresets hervorgehen.

Für diese Entwicklungs-/Evaluationsphase ist es jedoch von besonderem Vorteil, dass durch die kurzen DBF/BBF-Zyklen aus dem FDD jederzeit ein Produkt zur Verfügung steht, das zudem einen definierten Featurestand hat. Es lassen sich so auf den aktuellen Features und vorhanden Use-Cases aufbauend Benutzungsszenarien definieren, aus denen Benutzungstests abgeleitet werden können. Die Resultate werden im Lichte der erhobenen Benutzereigenschaften interpretiert und gewichtet. Im Sinne der agilen Softwareentwicklung lassen sich jederzeit neue Kundenanforderungen – in diesem Fall Evaluationsergebnisse – einbeziehen.

Damit agile Systementwicklung erfolgreich ist, ist es von großer Bedeutung, dass alle Entwickler einen guten Einblick in die Gestaltung der Teilsysteme haben. Für PIE/FDD haben wir ein mehrstufiges Verfahren umgesetzt, das der verteilten Entwicklung Rechnung trägt: *Lokale*

Entwicklerteams halten sich gegenseitig in wöchentlichen CodeMeetings auf dem Laufenden, in denen jeweils ein Thema aus der aktuellen Entwicklung vorgestellt und diskutiert wird. Das kann ein Standard, das Design eines Teilsystems oder eine Technologie betreffen. Designentscheidungen werden hier getroffen und dokumentiert. Die Protokolle stehen projektweit per HTTP zur Verfügung. Zur *projektweiten Vernetzung* werden zusätzlich nützliche Dokumente wie die Konventionen zur Codierung und Benennungen im Programmtext, aber auch binäre Objekte, wie Datenbank-Dumps oder aktuelle Prototypen bereitgestellt und in einem monatlichen Newsletter der Gesamtstand auf der Basis der Featurelisten kommuniziert.

Anwendung von PIE/FDD

Durch den Innovationsdruck bei der Entwicklung moderner Mensch-Maschine-Systeme werden Funktionen zunehmend als Software realisiert. So werden z.B. auch moderne Maschinensteuerungen inzwischen auf der Basis von MS Windows realisiert und insbesondere auch die MMI-Schnittstellen mit konventionellen softwaretechnischen Werkzeugen erstellt [10]. Es ist deshalb erforderlich, in der Entwicklungsmethodik die Besonderheiten von Softwareerstellung zu berücksichtigen. Die im vorangegangenen Abschnitt beschriebene Methode PIE/FDD ist ein Versuch, genau diesen neuen Anforderungen gerecht zu werden.

Die Methode PIE/FDD wurde bereits für die Entwicklung interaktiver Software eingesetzt (s. Tabelle 3). Im Projekt useworld.net [11] wurde ein Web-Portal entwickelt, das Informations- und Kooperationsangebote für die *scientific community* Mensch-Maschine-Interaktionsforscherinnen und -forscher bereitstellt. Die Entwicklung fand interdisziplinär und an mehreren Standorten verteilt statt: Softwareentwicklung in Aachen und Berlin, Interaktions- und Oberflächendesign in Essen und Evaluation in Kaiserslautern. Um während der zweijährigen Projektlaufzeit alle Einzelschritte zu koordinieren, wurde PIE/FDD eingesetzt.

Die Arbeiten begannen mit Analysen von Arbeitsabläufen, Bedürfnissen und Möglichkeiten der Zielgruppe [12]. Als Ergebnis dieser Phase wurde weiterer Aufwand in die Analyse des *community building*-Prozesses investiert [9]. Diese Ergebnisse haben den Detailentwurf insbesondere der Interaktionsprinzipien, aber auch der technischen Funktionalität, z.B. durch die Einführung von Moderatorenrollen in der Benutzerverwaltung, gesteuert. Die vollparallele Entwicklung von Oberflächendesign und technischen Funktionskomponenten ist i.a. schwierig, da für die Integration und Test der implementierten Funktionen ein GUI erforderlich ist. Durch die Entwicklung auf der Basis des Java 2 Enterprise Edition Frameworks war jedoch eine nahezu vollständige Trennung von Funktion und Design in Servlets (nach dem Controller/Command-Schema) und Java Server Pages (JSP) möglich. Die Realisierung technischer Funktionen konnte dabei über einen längeren Zeitraum

losgelöst von der Interaktionsgestaltung mit einfachen Test-JSPs geschehen. Neben wiederholten begleitenden Design-Reviews gab es eine formative Usability-Untersuchung, aus deren Ergebnissen eine neue Feature-Liste für Interaktions-Design und Funktions-Implementierung resultierte.

Die verteilte Entwicklung wurde durch den Einsatz eines cvs-Repositories möglich. Von dem nützlichen Konzept der Class-Ownership musste wegen der Mehrfachprojektzugehörigkeit (und damit nicht ständigen Verfügbarkeit) einiger Entwickler abgewichen werden. Der Projektfortschritt konnte anhand der Features in monatlichen Newslettern kommuniziert werden. Es hat sich gezeigt, dass die *plan by feature*-Phase etwa alle 3-4 Monate durchlaufen werden muss, um den geänderten Randbedingungen Rechnung zu tragen. Insbesondere während der Evaluationsphase wurde die Webanwendung nach jedem *build by feature* auf einem projektinternen Server *deployed*, damit Design und Funktion auch den nicht cvs-Benutzern zugänglich war.

Tabelle 3: Übersicht über mit PIE/FDD durchgeführte Projekte

	useworld.net	**[my:PAT.org]**	**PLS-Prototyping**
Anwendungs-bereich	webgestütztes kollaboratives Informationssystem für MMI-Forschung	webgestütztes, simulationsbasiertes eLearning-Angebot für Prozess- und Anlagentechnik	Prozessleitsystem-Prototypen für Training und Experimente
Software-Architektur	Java 2 EE, MVC-Modell 2	Java 2 EE	Proper Educt Small System Edition, Java 2 SE
Projekt-Charakteristika	interdisziplinär, verteilt, Zyklen 3 x 3-4 Monate	interdisziplinär, Zyklen: 2 x 3 Monate	kleine disziplinäre Einzelprojekte in einem interdisziplinären Rahmen

In anderen Projekten (s. Tabelle 3) wurde diese Methode erprobt und weitere Erfahrungen gesammelt.

Ausblick

Während der PIE/FDD Prozess im useworld.net-Projekt noch verfeinert wurde, wurde er in anderen teilweise kleineren Softwareentwicklungsprojekten bereits in ähnlicher Form eingesetzt. Es hat sich gezeigt, dass eine konsequente Verwendung von Featurelisten als Ergebnis von Benutzungsszenarien, als Basis für die Implementierungsarbeit und die Bewertung nützlich ist. Das agile Vorgehensmodell macht eine Integration der sich im Prozess ändernden Anforderungen, wie sie in der partizipativen Gestaltung und in der Entwicklung innovativer Systeme auftreten, leicht. Der Prozess PIE/FDD muss jedoch jeweils für unterschiedliche Anwendungsframeworks und Software-Architekturen verfeinert werden. Insbesondere der Umgang mit dem Konfigurationsmanagement und die Iterationsdauer (*build a featurelist*) müssen je nach Software-Typ angepasst werden. PIE/FDD ist so bereits für die Erstellung von web-basierten Anwendungen mit J2EE konfiguriert

worden. Für andere Gegebenheiten müssen erst noch Erfahrungen zur Anpassung des Prozesses gesammelt werden.

Obwohl der Prozess FDD explizit die Entwicklung objekt-orientierter Systeme adressiert, ist eine Anpassung der entsprechenden Aktivitäten in andere softwaretechnische Bereiche denkbar und wünschenswert, weil die Featurenotation die interdisziplinäre Zusammenarbeit erleichtert. In erster Linie müssten Klassen mit den entsprechenden verbundenen Konzepten Klassendiagramm, Interaktionsdiagramm und *Class-Ownership* durch eine andere Form der Kapselung der Entwicklung an zu bearbeitetenden Systemteilen ersetzt werden.

Die Erfahrungen aus den durchgeführten Projekten zeigt, dass Mitarbeiterschulungen zum Einsatz von PIE/FDD bzw. FDD trotz der geringen Anforderungen notwendig sind. Hier sollte noch standardisiertes Material erstellt werden. Um den administrativen Aufwand möglichst gering zu halten, ist eine weitere Tool-Unterstützung ebenfalls wünschenswert: Ein solches Werkzeug sollte eine Feature-Datenbank projektweit zur Verfügung stellen, die Features, Ressourcen-Zuordnung, Terminplanung und Bearbeitungsstand mit optionalen Notizen zu Designentscheidungen und Implementierungsproblemen enthält und aus der heraus ein entsprechendes Reporting möglich ist.

Abschließend lässt sich festhalten, dass die agile Systementwicklung, wie sie mit PIE/FDD ermöglicht wird, zu einer Flexibilisierung des Anforderungsmanagements führt. Dies resultiert nicht in schlechterer oder fehlerhafterer Software, sondern im Gegenteil zu qualitätsgesicherten anforderungsgerechten Systemen. Agile Methoden skalieren jedoch nicht gut. Sehr große Systeme müssen also entsprechend modularisiert werden und es muss dazu ein Management der Schnittstellendefinitionen eingeführt werden, das sie entgegen der Agilitätsidee strikt konstant hält und gleichzeitig noch die Transparenz der Subsystemzusammenhänge in einem (dann sehr großen) mglw. verteilten interdisziplinären Entwicklerteam aufrecht erhält.

Gefördert von der VolkswagenStiftung im Rahmen des Programms „Nachwuchsgruppen an Universitäten“ und vom DFN-Verein mit Mitteln des BMBF im Bereich Einsatz von Netzdiensten im wissenschaftlichen Informationswesen.

Literatur

[1] Radice, R. A.; Roth, N. K.; O'Hara Jr., A. C. & Ciarfella, W. A.: A Programming Process Architecture. IBM Systems Journal 24 (1985) Nr. 2, S. S. 79/90.

[2] NIST: Integration Definition for Function Modelling (IDEF0). FIPSPUB 183. Springfield, VA: NIST 1993. Verfügbar unter http://www.idef.com/Downloads/pdf/idef0.pdf (letzter Zugriff am 7. Okt. 2003).

[3] Humphrey, W. S.: Introduction to the Personal Software Process. Reading, MA: Addison-Wesley 1997.

[4] McAndrews, D. R.: The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices (Technical Report CMU/SEI-2000-TR-015 ESC-TR-200-015). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute 2000. Verfügbar unter: http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr015.pdf (letzter Zugriff am 7. Okt. 2003).

[5] BMI: Planung und Durchführung von IT-Vorhaben – Vorgehensmodell (Allgemeiner Umdruck Nr. 250). Berlin: Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt) 1997. Verfügbar unter: http://www.v-modell.iabg.de/vm97.htm (letzter Zugriff am 7. Okt. 2003).

[6] C3 Team: Chrysler Goes to "Extremes". Distributed Computing 10 (1998), S. 24/28.

[7] Coad, P.; Lefebvre, E. & De Luca, J.: Java Modeling in Color With UML: Enterprise Components and Process. Upper Saddle River, NJ: Prentice Hall International 1999.

[8] Cockburn, A.: Humans and Technology's Manifesto for software development. o. Jahr. Verfügbar unter: http://members.aol.com/acockburn/manifesto.html (letzter Zugriff am 7. Okt. 2003).

[9] Timpe, K.-P. & Kolrep, H.: Das Mensch-Maschine-System als interdisziplinärer Gegenstand. In: K.-P. Timpe; T. Jürgensohn & H. Kolrep (Hrsg.): Mensch-Maschine-Systemtechnik: Konzepte, Modellierung, Gestaltung, Evaluation. Düsseldorf: Symposium 2002. S. 9/40.

[10] Zühlke, D.: Bedienung komplexer Maschinen. Heute, morgen und übermorgen. In: R. Marzi, V. Karavezyris, H.-H. Erbe & K.-P. Timpe (Hrsg.): Bedienen und Verstehen. 4. Berliner Werkstatt Mensch-Maschine-Systeme. 10. bis 12. Oktober 2001. Fortschritt-Berichte VDI Reihe 22 Nr. 8. Düsseldorf: VDI-Verlag 2002. S. 42/54.

[11] Leuchter, S.; Urbas, L. & Röse, K.: Engineering and Evaluation of Community Support in useworld.net. In Constantine Stephanidis & Julie Jacko (Hrsg.), Human-Computer Interaction. Theory and Practice (Part II). Mahwah, NJ: Lawrence Erlbaum Publishers 2003. S. 959/963

[12] Leuchter, S.; Rothmund, T. & Kindsmüller, M. C.: Ergebnisse einer Tätigkeitsbefragung zur Vorbereitung der Entwicklung eines Web-Portals für Mensch-Maschine-Interaktion. In: Ar-

beitswissenschaft im Zeichen gesellschaftlicher Vielfalt. 48. Kongress der Gesellschaft für Arbeitswissenschaft. Johannes Keppler Universität Linz 20.-22. Februar 2002. Düsseldorf: GfA-Press 2002. S. 129/132.

[13] Kindsmüller, M. C.; Razi, N.; Leuchter, S. & Urbas, L.: Zur Realisierung des Konzepts "Nutzer als Redakteure" für einen Online-Dienst zur Unterstützung der MMI-Forschung im deutschsprachigen Raum. In: Arbeitswissenschaft im Zeichen gesellschaftlicher Vielfalt. 48. Kongress der Gesellschaft für Arbeitswissenschaft. Johannes Keppler Universität Linz 20.-22. Februar 2002. Düsseldorf: GfA-Press 2002. S. 133/137.