Entwicklungsprozesse für eingebettete Software

Link:
http://www.springer.com/spektrum+akademischer+verlag/informatik/informatik+und+it+übergreifend/book/978-3-8274-1533-2

Bibliographic information:

Jürgen Münch, Dieter Rombach. Software Engineering eingebetteter Systeme: Grundlagen – Methodik – Anwendungen, chapter Entwicklungsprozesse für eingebettete Software, pages 13-38. Elsevier, 2005.

3 | Entwicklungsprozesse für eingebettete Software

Von Jürgen Münch und Dieter Rombach

Explizite Modelle von Entwicklungsprozessen und fortschrittliche Prozesstechnologien gehören zu den wesentlichen methodischen Grundlagen des Software-Projektmanagements und der Prozessoptimierung. Da Software-Entwicklungsprojekte in ihrer Kombination von projektspezifischen Zielen und Randbedingungen einzigartig sind, ist die Vorgabe „idealer" und allgemeingültiger Entwicklungsprozesse keine Lösung für die Praxis. Vielmehr bedarf es der Bereitstellung effektiver und effizienter Software-Entwicklungsprozesse, die für ein Projekt maßgeschneidert werden können. Im Folgenden wird ein Überblick über Software-Prozesse und ihre Bedeutung im Rahmen der Entwicklung eingebetteter Software gegeben. Basierend auf einer Definition der Grundkonzepte werden unterschiedliche Arten von Prozessen vorgestellt. Daraufhin wird exemplarisch gezeigt, wie besondere Eigenschaften eingebetteter Systeme den Entwicklungsprozess beeinflussen. Abschließend wird dargestellt, wie domänenspezifische Prozesse etabliert und verbessert werden können.

Übersicht

3.1 Motivation

Eingebettete Software gewinnt zunehmend an Bedeutung für die Marktfähigkeit zahlreicher Produkte. Bei steigendem Kostendruck, sich drastisch verkürzenden Innovationszyklen und immer wichtiger werdenden Qualitätsanforderungen wird die Software-Entwicklung vor besondere Herausforderungen gestellt. Bei heutigen Entwicklungen eingebetteter Software passiert es nicht selten, dass Software- und Hardwareentwicklung unzureichend aufeinander abgestimmt sind, Kosten unterschätzt werden oder Qualitätsziele nicht erreicht werden. Eine wesentliche Ursache hierfür ist, dass die Entwicklung von eingebetteter Software bisher nicht nach ingenieurmäßigen Grundsätzen erfolgt. Detaillierte Vorgaben von Projektzielen, eine systematische Auswahl von Entwicklungsprozessen und -ansätzen, die projektbegleitende Qualitätssicherung sowie die Verbesserung von Entwicklungsprozessen auf der Basis von Erfahrungen sind heute noch die Ausnahme. Im Gegensatz zu anderen Ingenieurdisziplinen sind wir im Bereich der Software-Entwicklung noch weit davon entfernt, vorgegebene Qualitätsanforderungen an ein Softwareprodukt durch ingenieurmäßige Vorgehensweisen garantieren zu können. Erfolgreiche Software-Projekte sind im Allgemeinen nicht das Ergebnis von Softwareentwicklungskompetenz, sondern werden eher zufällig durch begabte und engagierte Entwickler erreicht. Die technische Software-Entwicklung wird der Kreativität der Entwickler überlassen. Somit sind Erfolge selten wiederholbar.

Ingenieurmäßige Software-Entwicklung

Erfahrungen aus fortschrittlichen Software-Entwicklungsorganisationen wie beispielsweise dem Software Engineering Laboratory (SEL) der NASA [10] zeigen, dass sich Verbesserungen in Produktivität, Qualität und Vorhersagbarkeit erreichen lassen, indem man Software-Entwicklung grundsätzlich auf eine ingenieurmäßige Basis stellt. Dies bedeutet, dass die Industrie Software nach expliziten Prozessmodellen entwickelt, dass sie quantitative Methoden der Softwareentwicklung einführt und Prozess- und Vorhersagemodelle etabliert, die kontinuierlich verbessert werden.

Effektive und effiziente Software-Entwicklungsprozesse und ihre Anwendung in der Praxis haben eine zunehmende Bedeutung und werden essenziell für den Geschäftserfolg. Die Ursachen hierfür sind vielfältig. Beispiele sind der Trend zu immer komplexeren Systemen, ständig wachsende und sich rascher ändernde Anforderungen und ein ungenügendes Verständnis des Zusammenwirkens einzelner Entwicklungsaufgaben.

Reifegrad von Entwicklungsorganisationen

Die Relevanz systematischer Software-Entwicklungsprozesse sowie expliziten Prozesswissens ist industriell erkannt und hat zu einer Bewertung von Unternehmen entsprechend so genannter Reifegradmodelle geführt (z. B. CMM [9], CMMI, SPICE). Diese Modelle berücksichtigen den Stand des Wissens bezüglich Software-Engineering und das Vorhandensein und die Anwendung expliziter Entwicklungsprozesse. Der Reifegrad einer Softwareentwicklungsorganisation gewinnt als Beurteilungskriterium für die Vergabe von Entwicklungsaufträgen signifikant an Bedeutung.

Darüber hinaus ist die explizite Repräsentation von Entwicklungsprozessen eine wichtige Voraussetzung für Prozessverbesserungen. Explizite Prozesse sind notwendig, um den Ist-Zustand darzustellen und zu analysieren, um darauf aufbauend Prozessoptimierungen durchzuführen. Watts Humphrey charakterisiert dies treffend: „If you don't know where you are, a map won't help" [9].

Maßgeschneiderte Software-Entwicklungsprozesse

Eine wichtige Herausforderung ist, den passenden und geeigneten Entwicklungsprozess für eine Software-Entwicklungsorganisation bzw. ein Projekt zu finden. Da es weder einen universellen Entwicklungsprozess noch eine optimale Entwicklungsmethode gibt, muss der Entwicklungsprozess immer auf die Entwicklungsziele sowie die übergeordneten Geschäftsziele abgestimmt sein und in dem ihn umgebenden Kontext betrachtet werden. Der Entwicklungskontext umfasst Aspekte wie z.B. die Anwendungsdomäne, die vorhandenen Entwicklungstechniken, die Erfahrung der Entwickler, den organisatorischen Rahmen oder Budgetgrenzen. Prozesse müssen somit an die jeweiligen Rahmenbedingungen angepasst werden können.

Die systematische Planung von Software-Projekten nutzt geeignete Software-Prozessmodelle, passt diese an die Projektziele und -randbedingungen an und fügt weitere Informationen (z. B. Resourcen) hinzu. Es entsteht ein Projektplan, der eine wichtige Grundlage für die Steuerung und Verfolgung des Software-Entwicklungsprojekts ist. Daher muss eine hohe Übereinstimmung zwischen der tatsächlichen Durchführung von Prozessen in der realen Welt und ihrer Repräsentation in der so genannten Software-Engineering-Modellwelt angestrebt werden. Das Verständnis des Unterschiedes zwischen beiden Welten ist wichtig, da aus der Modellwelt Schlüsse für Handlungen in der realen Welt gezogen werden und umgekehrt die reale Welt Grundlage für die Bildung der Modellwelt ist. In der realen Welt werden Software-Projekte mit Menschen durchgeführt. Sie haben bestimmte Tätigkeiten durchzuführen, wobei sie Produkte konsumieren, produzieren oder modifizieren können. Dabei haben sie bestimmte Qualitätsvorgaben einzuhalten. In der Modellwelt existieren Prozess-, Produkt-, Ressourcen und Qualitätsmodelle. Durch Instanzierung der Modelle erhält man Objekte, die bei der Projektdurchführung auf Elemente der realen Welt abgebildet werden.

3.2 Definitionen und Konzepte

3.2.1 Grundbegriffe

Software-Produkt

Ein *Software-Produkt* ist jedes Artefakt, das im Rahmen der ingenieurmäßigen Software-Entwicklung benötigt oder erzeugt wird. Produkte können materiell oder immateriell sein. Beispiele für Artefakte sind Anforderungsdokumente, System-Entwürfe, Programmierrichtlinien, Quellkodedateien, Binärkode, Projektfortschrittsberichte, Messdaten oder Feh-

lerberichte. Software-Produkte können durch andere Produkte verfeinert werden, so dass sich Produkthierarchien bilden. Beispielsweise kann ein Entwurfdokument durch den Grobentwurf und den Feinentwurf verfeinert werden. Produkte, die nicht weiter verfeinert sind, werden atomare Produkte genannt.

Software-Prozess

Ein *Software-Prozess* ist eine zielorientierte Aktivität zur Erzeugung eines Produkts oder zur Durchführung einer anderen Aufgabe im Rahmen der ingenieurmäßigen Software-Entwicklung. Ein Software-Prozess wird in der realen Welt durchgeführt, d.h. in einer konkreten Entwicklungsorganisation. Ein Beispiel für einen Software-Prozess ist die Kodierung der Systemkomponente Nr. 15 im Projekt Alpha bei der Firma ACSoft. Ein Software-Prozess transformiert in der Regel ein oder mehrere Eingangsprodukte unter Zuhilfenahme weiterer Produkte in ein oder mehrere Ausgangsprodukte. Beispielsweise gehen in die Aktivität „Kodierung der Systemkomponente Nr. 15“ als Eingangsprodukte die Spezifikation für diese Komponente und der Entwurf der Komponente ein. Die Kodierung erfolgt mithilfe des Produkts „Kodierungsrichtlinien“. Das Ausgangsprodukt dieser Aktivität ist der Quellkode für die Systemkomponente. Software-Prozesse können von Personen oder Maschinen oder von beiden zusammen durchgeführt werden. So ist die Kompilierung von Quellkode ein typisches Beispiel für einen automatisierten Software-Prozess, der von einer Maschine durchgeführt wird. Testprozesse oder Fehlerbeseitigungsprozesse werden vielfach semiautomatisch durchgeführt, d.h. sie werden manuell von Personen unter Zuhilfenahme von Werkzeugen durchgeführt. Software-Prozesse können durch andere Prozesse verfeinert werden, so dass sich eine Prozesshierarchie ergibt. So kann beispielsweise der Testprozess durch die Prozesse Testgenerierung, Testdurchführung und Diagnose verfeinert werden. Diese können wiederum verfeinert werden. Prozesse, die nicht weiter verfeinert sind, werden atomare Prozesse genannt.

Projekt und Projektphase

Ein *Projekt* ist ein einmaliges Vorhaben, das durch einen Anfangs- und Endtermin zeitlich begrenzt ist und ein Ziel erreichen soll.

Eine *Projektphase* (kurz: Phase) ist ein zeitlicher Projektabschnitt, der inhaltlich getrennt von anderen Projektphasen abläuft. Im Unterschied zu einem Prozess ist eine Phase immer durch einen Anfangszeitpunkt und einen Endzeitpunkt definiert. Ist dieser Zeitraum vorbei, ist auch die Phase beendet. So ist beispielsweise die Analysephase eines Software-Projekts zu einem bestimmten Zeitpunkt im Projekt abgeschlossen. Prozesse können jedoch mehrfach aktiviert werden. So kann beispielsweise der Prozess „Bearbeitung des Anforderungsdokuments“ auch nach dem Abschluss der Analysephase noch aktiviert werden, etwa bei Änderungen der Anforderungen in der Entwurfsphase oder zur Korrektur von Fehlern in der Testphase.

Modell

Ein *Modell* ist eine abstrakte und vereinfachende Repräsentation eines Objekts oder Phänomens der realen Welt. Ein Modell beschreibt nur

diejenigen Aspekte des Objekts oder Phänomens, die bekannt und für das Verständnis und die beabsichtigte Nutzung des Modells relevant sind (oder von denen der Modellentwickler glaubt, dass sie relevant seien). Modelle sind eine Möglichkeit, Erfahrungen explizit darzustellen. Modelle können für unterschiedliche Zwecke genutzt werden, beispielsweise zur Planung, Kontrolle oder Vorhersage. Modelle durchlaufen einen eigenen Lebenszyklus, d.h., sie werden entworfen, analysiert, benutzt, beurteilt und überarbeitet.

Software-Prozessmodell

Ein *Software-Prozessmodell* ist die Beschreibung eines Software-Prozesses. Ein Software-Prozessmodell ist somit nicht das Gleiche wie ein Software-Prozess, es gibt einen wesentlichen Unterschied: Ein Software-Prozessmodell beschreibt die Durchführung der Aktivität in der realen Welt, der Software-Prozess ist die Aktivität als solche. So wie ein Software-Prozessmodell die Durchführung des Software-Prozesses spezifiziert, beschreibt ein Backrezept die Aktivität des Kuchenbackens; das Backrezept ist die Prozessbeschreibung, der Vorgang des Backens ist der Prozess. Im Alltag finden sich viele Prozessbeschreibungen: Die von einem Routenplaner berechnete schnellste oder kürzeste Strecke ist eine Prozessbeschreibung, wohingegen es sich beim Abfahren der Strecke mit einem Auto um einen Prozess handelt. Eine Anleitung zur Wartung einer Maschine ist ein Prozessmodell, wohingegen der Vorgang der Wartung (z. B. das Nachjustieren oder Ölen) ein Prozess ist. Die Nutzung von Prozessbeschreibungen ist weit verbreitet und wird vielfach als Mittel zur Problemlösung genutzt. Prozessmodelle können erweitert werden um die Angabe relevanter quantitativer Messgrößen (z.B. Aufwand, Effizienz, Effektivität, Prozesstreue).

Software-Prozessmodelle können unterschiedlich abstrakt beschrieben werden. Es lassen sich wenigstens zwei Ebenen unterscheiden, der Prozessmodelltyp und die Prozessmodellinstanz (oder Prozessmodellausprägung). Der *Prozessmodelltyp* beschreibt eine Klasse ähnlicher Prozessmodelle. Die *Prozessmodellinstanz* ist die Beschreibung einer Ausprägung dieser Klasse. Beispielsweise gehört die Instanz „Inspektion der Anforderungen durch Lesetechnik x“ zum Prozessmodelltyp „Anforderungsinspektion“.

Mit Prozessmodellen lassen sich Erfahrungen explizit beschreiben. Die Nutzung von Software-Prozessmodellen kann somit auch als Nutzung von gewonnenem Wissen betrachtet werden. Ingenieurmäßige Software-Entwicklung erfordert generell die Nutzung von existierendem Software-Wissen während der Projektplanung und -durchführung sowie die Gewinnung und Aufbereitung neuen Software-Wissens für die Nutzung in zukünftigen Projekten.

Software-Produktmodell

Ein *Software-Produktmodell* ist die Beschreibung eines Software-Produkts bzw. einer Klasse von Software-Produkten. Analog zu Software-Prozessmodellen spezifiziert das Produktmodell ein Produkt, ist aber nicht mit ihm gleichzusetzen. Es gibt Produktmodelle auf Instanzenebene (z. B.

Vorlage für ein Anforderungsdokument in einem bestimmten Projekt) und auf Typebene (z. B. Beschreibung der Struktur von Entwurfsdokumenten, Datenschema für Messdaten). In der Regel beinhalten Software-Produktmodelle eine Beschreibung der elementaren Informationsbausteine eines Software-Produkts (z. B. funktionale Anforderungen, nichtfunktionale Anforderungen, Entwurfsentscheidungen) und eine Vorgabe zur Strukturierung dieser Informationsbausteine (z. B. Gliederung des Anforderungsdokuments). Darüber hinaus können Produktmodelle erweitert werden um die Angabe relevanter quantitativer Messgrößen für das Produkt (z. B. Anzahl der Function Points, Kopplungsmaß zur Beschreibung des Grades der Abhängigkeit von Entwurfskomponenten).

Prozessarchitektur

Eine *Prozessarchitektur* (auch: Prozessschema, Prozess-Metamodell) ist ein konzeptueller Rahmen für die konsistente Beschreibung von Prozessen und ihrer Beziehungen. Eine Prozessarchitektur beschreibt einerseits die Bausteine, aus denen sich Prozessmodelle zusammensetzen lassen, und andererseits, auf welche Art und Weise dies geschehen darf. So gibt es beispielsweise Prozessarchitekturen, die Eingabe-, Ausgabe-, und Hilfsprodukte erlauben, wohingegen andere nur Ausgabeprodukte zulassen. In einigen Prozessarchitekturen wird der Kontrollfluss (also die Reihenfolgebeziehung zwischen einzelnen Prozessschritten) imperativ beschrieben; andere Prozessarchitekturen sehen eine Beschreibung des Kontrollflusses mittels Eingangs- und Ausgangsbedingungen vor. Es hat sich bisher noch keine einheitliche Prozessarchitektur etabliert. Oftmals verwenden unterschiedliche Organisationen unterschiedliche Prozessarchitekturen. Auch in verschiedenen Prozessstandards sind unterschiedliche Architekturen vorzufinden. Prozesswerkzeuge (z. B. Modellierungswerkzeuge) sind derzeit noch häufig an feste Prozessarchitekturen gebunden. Nur wenige Prozesswerkzeuge können mit verschiedenen Prozessarchitekturen umgehen bzw. individuelle Prozessarchitekturen importieren. Vielfach wird eine Prozessarchitektur ad hoc im Zuge der Erstellung einer Prozessbeschreibung entworfen. Allerdings werden dabei häufig Fehler gemacht (z. B. Verfeinerung von Phasen durch Aktivitäten), die vermieden werden könnten, wenn Prozessexperten herangezogen würden. Neuere Prozessmanagementansätze sollten flexibel mit unterschiedlichen Prozessarchitekturen umgehen können, um in unterschiedlichen Umgebungen Anwendung zu finden.

Agent und Rolle

Die Schnittstelle zwischen Prozessen und den sie durchführenden Personen und Maschinen basiert auf dem Agenten- und Rollenkonzept. Ein *Agent* ist ein Akteur (Mensch oder Maschine), der einen Prozess oder Teile eines Prozesses ausführt. Eine *Rolle* definiert eine Menge von zielgerichteten Aktivitäten im Rahmen eines Software-Entwicklungsprojekts oder der dazu notwendigen organisatorischen Umgebung. Diese Aktivitäten werden einem oder mehreren Agenten zugewiesen. Beispiele für Rollen sind Tester, Qualitätssicherer, Projektmanager oder Entwurfsingenieur. Ein einzelner Agent kann mehrere Rollen einnehmen (z. B. Anforderungsingenieur *und* Tester) und eine einzelne Rolle kann von mehreren Agenten (z. B. Mitglieder eines Entwurfsteams) eingenommen

werden. Rollendefinitionen entsprechen Rollen in Filmen oder Theaterstücken. So wie ein Schauspieler eine Rolle in einem Film oder einem Theaterstück spielt, nehmen Agenten Rollen bei der Durchführung von Prozessen ein. Das Rollenkonzept erlaubt die Definition der notwendigen Voraussetzungen (z. B. Qualifikationen von Agenten, Leistungsmerkmale von Maschinen) für die Durchführung zusammenhängender Aufgaben. Hierdurch ist eine getrennte Beschreibung der notwendigen Fähigkeiten für das Einnehmen einer Rolle und den spezifischen Eigenschaften eines Agenten, der mehrere unterschiedliche Rollen einnehmen kann, möglich. Die Trennung von Rolle und Agent betont also den Unterschied zwischen einer Rollenbeschreibung, die prozessspezifisch ist, und der weitgehend prozessunabhängigen Beschreibung der fachlichen Fähigkeiten und Erfahrungen einer Person (bzw. den notwendigen Eigenschaften einer Maschine). Falls Rollendefinitionen in unterschiedlichen Prozessbeschreibungen denselben Namen haben, sollten sie identisch sein. Eine Rolle wird im Einzelnen definiert durch

- personenunabhängig beschriebene Aktivitäten, gegebenenfalls erweitert um Angaben zur Reihenfolge der Tätigkeiten (z. B.: erstelle zuerst Nutzungsszenarien und leite danach Testfälle ab);
- Produkte, die in den Aktivitäten benötigt oder erzeugt werden (z. B. Anforderungsdokument, Checklisten, Testtreiber);
- Verantwortlichkeiten, Rechte und Pflichten (z. B.: halte Budgetvorgaben ein, lesender Zugriff auf Personaldaten erlaubt, erfasse den Fehlerbehebungsaufwand);
- notwendige Qualifikationen und Erfahrungen der Agenten, die die Rolle einnehmen (z. B.: Training in OO-Analyse ist absolviert, zertifizierter Function-Point-Schätzer, mindestens 5 Jahre Programmiererfahrung, mindestens 3 Jahre Erfahrung mit eingebetteten Systemen);
- den Grad der Beteiligung an Aktivitäten (z. B.: ist für die Bearbeitung verantwortlich, muss der Durchführung zustimmen, arbeitet bei der Durchführung mit, muss über die Durchführung der Aktivität informiert werden).

Ein wesentlicher Vorteil von Rollen ist, dass zusammenhängende Aktivitäten personenunabhängig definiert und die dafür notwendigen Kompetenzen klar umrissen werden können. Darüber hinaus wird die Projektplanung vereinfacht, da Ressourcen unabhängig von Personen und Maschinen bestimmt werden können und Ressourcen-Engpässe frühzeitig erkannt werden können.

Prozessmanagement-rollen

Wir unterscheiden technische Entwicklerrollen (z. B.: Anforderungsingenieur, Kodierer, Verifizierer) von organisations- und management-orientierten Rollen (z. B. Produktmanager, Projektmanager, Qualitätssicherer). Zu den organisations- und management-orientierten Rollen gehören auch die Prozessmanagement-Rollen. Im Folgenden wird beispielhaft eine Definition gängiger Prozessmanagement-Rollen gegeben:

- *Prozessverantwortlicher* (Process owner). Der Prozessverantwortliche trägt die fachliche Verantwortung für die effiziente und effektive Durchführung der ihm zugeordneten Prozesse. Er ist für die Überwachung und kontinuierliche Verbesserung seiner Prozesse verantwortlich. Hierbei wird er durch andere Rollen (z. B. Prozess-Designer) unterstützt.
- *Prozess-Designer*. Der Prozess-Designer definiert und modelliert Prozesse, die jeweils zur Erreichung einer Klasse von Zielen (z. B. Erstellung von Testfällen) in bestimmten Kontexten (z. B. Steuergeräte im Automobil) dienen. Dies kann beispielsweise durch Interviews mit den Fachabteilungen, durch Übernahme von Best Practices oder durch Anpassung von Standards an firmenspezifische Gegebenheiten erfolgen. Beim Prozess-Design ist zu beachten, dass die Schnittstellen des zu entwerfenden Prozesses auf die Schittstellen der umgebenden Prozesse abgestimmt sind. Zu den Aufgaben des Prozess-Designers gehört auch, dass er beschreibt, für welche Arten von Projekten und Situationen der Prozess geeignet ist und wie er entsprechend angepasst wird.
- *Prozess-Dokumentationsspezialist*. Der Prozess-Dokumentationsspezialist generiert aus einem Prozessmodell rollen- und zweckorientierte Beschreibungen. Hierbei kann es sich beispielsweise um Prozesshandbücher für Entwickler, Qualitätshandbücher für Qualitätssicherer, Trainingsmaterialien, Audit-Unterlagen oder um Inter-/ Intranet-Beschreibungen des Prozesses handeln.
- *Prozessingenieur*. Der Prozessingenieur entwickelt die Prozessarchitektur oder wählt eine geeignete Prozessarchitektur aus. Darüber hinaus definiert er die Prozessmanagement-Richtlinien, wählt Prozessmanagement-Werkzeuge aus und berät deren Anwender. Er unterstützt vor allem den Prozess-Designer.
- *Prozess-Qualitätsmanager*. Der Prozess-Qualitätsmanager sichert die Qualität der Prozessmodelle in Bezug auf definierte Qualitätskriterien (z. B. Vollständigkeit, Konsistenz, Akkuratheit) und verwaltet die Prozessmodelle in einem Repository. Zusätzlich ist er in Kooperation mit dem Prozess-Designer für die Wartung und Weiterentwicklung der Prozessmodelle verantwortlich.

Explizites und implizites Software-Wissen

Unter *Software-Wissen* verstehen wir die Gesamtheit an Informationen und Fähigkeiten zur Lösung von Software-Engineering-Problemen in bestimmten Situationen. Wissen kann implizit in den Köpfen von Experten existieren (sog. tacit knowledge); dieses Wissen ist oftmals unbewusst vorhanden und daher nur schwierig kommunizierbar. Der Wert von implizitem Wissen wird häufig erst bemerkt, wenn Experten plötzlich fehlen und mit ihnen das Wissen zur Lösung anstehender Probleme. Daher ist es wichtig, implizites Wissen zu externalisieren, d. h. in explizites Wissen zu überführen. Explizites Wissen ist in Dokumenten niedergelegt und kann kommuniziert werden. Ein wesentliches Mittel zur Beschreibung expliziten Wissens sind Modelle wie z. B. Software-Prozessmodelle, Produktmodelle oder Qualitätsmodelle.

3.2.2 Prozessarten

Klassifikation von Prozessmodellen

Abb. 3.1 stellt unterschiedliche Arten von Prozessmodellen dar (in Anlehnung an [15]). Prozessmodelle können auf unterschiedlichen Abstraktionsgraden beschrieben werden. *Lebenszyklusmodelle* sind Prozessmodelle, die Aktivitäten eines Projekts auf einem hohen Abstraktionsgrad beschreiben. Charakteristisch für Lebenszyklusmodelle ist, dass sie die komplette Entwicklung eines Software-Produkts bzw. Systems beschreiben. Ein Beispiel ist das Wasserfallmodell [16].

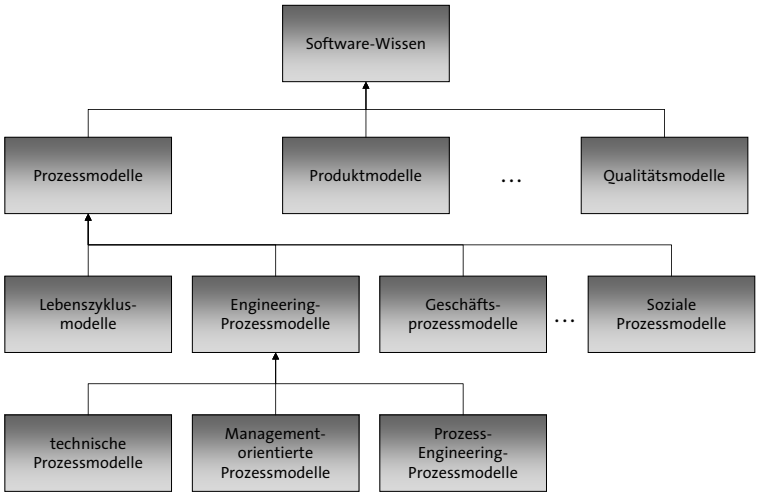

Abbildung 3.1 *Arten von Software-Prozessmodellen*

Engineering-Prozessmodelle

Engineering-Prozessmodelle fokussieren auf bestimmte Aspekte (z. B. Akzeptanztest, Aufwandskontrolle, Messen) und Rollen (z. B. Tester, Projektmanager, Qualitätssicherer). In der Regel beschreiben Engineering-Prozessmodelle Aktivitäten auf niedrigeren Abstraktionsebenen als Lebenszyklusmodelle. Beispiele für Engineering-Prozessmodelle sind Prozesse für den OO-Entwurf, die Projektplanung oder die Datensammlung. Engineering-Prozessmodelle können technische Prozesse, managementorientierte Prozesse und Prozess-Engineering-Prozesse beschreiben.

Technische Prozessmodelle

Technische Prozesse haben hauptsächlich das Ziel, Produktinformation von der Problembeschreibung bis zum endgültigen System systematisch zu transformieren und weiterzuentwickeln (siehe Abb. 3.2 auf der nächsten Seite). Im Folgenden ist eine Auswahl typischer technischer Prozesse beschrieben, wie sie ähnlich in den meisten Software-Entwicklungsprojekten vorkommen. Die Auswahl ist nicht vollständig und die einzelnen technischen Prozesse können in Projekten anders geartet sein. So kann beispielsweise die Bezeichnung der Prozesse oder die Aggregation von Einzeltätigkeiten zu Prozessen in Projekten variieren. Darüber hinaus haben zugrunde liegende Entwicklungsmethoden (z. B. eine objektorien-

tierte Entwicklungsmethode) Einfluss auf die Gestaltung der technischen Prozesse).

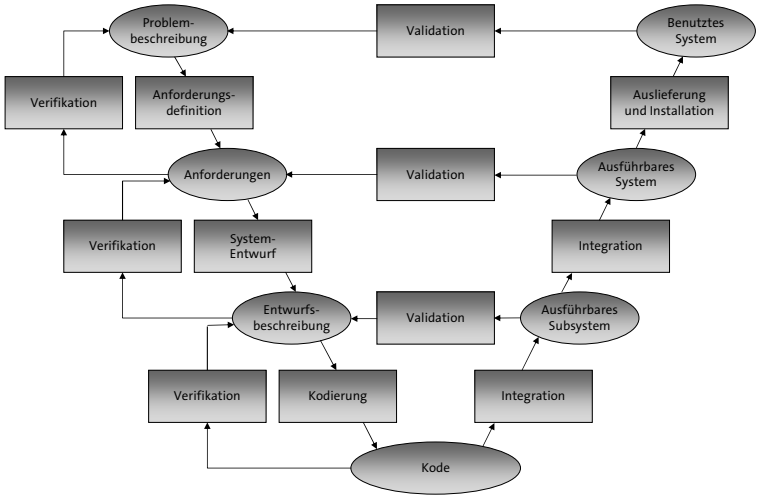

Abbildung 3.2 *Technische Prozesse und zugehörige Produkte*

- *Anforderungsdefinition*. Die Anforderungsdefinition ist derjenige Prozess, in dem die Eigenschaften festgelegt werden, die ein zu entwickelndes oder zu wartendes System haben sollte.
- *System-Entwurf*. Im System-Entwurf wird eine technische Lösung entwickelt, die überprüfbar den Anforderungen entspricht. Hier werden die Systemkomponenten, die Schnittstellen zwischen den Systemkomponenten und die Komponentenanforderungen festgelegt.
- *Kodierung*. Die Kodierung ist derjenige Prozess, bei dem der Entwurf in Anweisungen einer Programmiersprache transformiert wird.
- *Integration*. Die Integration ist derjenige Prozess, der aus Kode ausführbare Subsysteme bzw. das ausführbare System erzeugt.
- *Auslieferung und Installation*. Die Auslieferung bringt das ausführbare System von der Entwicklungsorganisation zum Anwender. Die Installation integriert das ausführbare System in die Umgebung des Anwenders.
- *Verifikation*. Die Verifikation umfasst alle Aktivitäten, die zum Nachweis der Konsistenz zwischen einem Dokument und seiner Spezifikation auf der nächst höheren Abstraktionsebene notwendig sind. Beispielsweise können die Anforderungen gegen die Problembeschreibung verifiziert werden, der Systementwurf kann gegen die Anforderungen verifiziert werden, und der Kode kann gegen den Systementwurf verifiziert werden.
- *Validation*. Die Validation umfasst alle Aktivitäten, die notwendig sind, um sich oder andere zu überzeugen, dass ein ausführbares

System oder ein ausführbarer Teil eines Systems (z. B. Komponente oder Subsystem) keine Inkonsistenzen zu den zugehörigen Anforderungen enthält.

Management-orientierte Prozessmodelle

Management-orientierte Prozesse haben hauptsächlich das Ziel, eine Umgebung und Dienstleistungen für die Entwicklung von Software bzw. Systemen bereitzustellen, die die Produktanforderungen und Projektziele erfüllen. Beispiele für management-orientierte Prozesse sind:

- *Projektplanung*. Die Projektplanung ist derjenige Prozess, der für eine Menge von Projektzielen und -randbedingungen passende Modelle auswählt, instanziert und in einem Projektplan integriert. Der Projektplan beschreibt im Wesentlichen Aktivitäten, Meilensteine, Zwischen- und Endergebnisse sowie die Zuordnung von Personen zu Rollen bzw. Aktivitäten. Bei der Projektplanung wird das Zusammenspiel unterschiedlicher Modelle deutlich: Beispielsweise wird ausgehend von einem gewählten Prozessmodell mit einem Schätzmodell der Gesamtaufwand und die Projektdauer geschätzt. Basierend auf diesen Schätzungen werden die Aufwände und Durchführungszeiträume einzelner Aktivitäten festgelegt. Darauf aufbauend werden Ressourcenzuordnungen vorgenommen. Neben dem eigentlichen Projektplan werden im Rahmen der Projektplanung weitere Pläne erstellt (z. B. Konfigurationsplan, Messplan, Kostenplan).
- *Projektmanagement*. Das Projektmanagement steuert und kontrolliert die durchgeführten Prozesse entsprechend des Projektplans während der gesamten Projektdauer. Die Kontrolle und Steuerung des Projekts durch das Projektmanagement konzentriert sich vor allem auf die Aspekte Kosten, Zeit und Inhalte (im Sinne von realisierter Funktionalität). Bei Abweichungen vom Projektplan leitet das Projektmanagement Gegenmaßnahmen ein bzw. initiiert Umplanungen.
- *Qualitätssicherung*. Die Qualitätssicherung umfasst alle Aktivitäten zur Überprüfung von Prozessen und Produkten in Bezug auf Qualitätsziele oder Qualitätsanforderungen. Im Gegensatz zum Projektmanagement, das vor allem auf die Kontrolle von Kosten, Terminen und Funktionalität abzielt, fokussiert die Qualitätssicherung auf Qualitätsaspekte (z. B. Wartbarkeit von Kode) und Faktoren, die zu Qualitätsproblemen führen können (z. B. Schnittstellenfehler im Entwurf). Qualitätssicherungsprozesse müssen eng mit dem Projektmanagement verzahnt sein, damit Abweichungen frühzeitig gemeldet und entsprechende Gegenmaßnahmen rechtzeitig eingeleitet werden können.
- *Produktmanagement*. Das Produktmanagement umfasst alle Aktivitäten zur Verwaltung und Bereitstellung aller im Rahmen eines Projekts benötigten Produkte. Hierzu gehört Versionierung und Konfiguration der Produkte sowie die Definition und der Einsatz eines Zugriffskonzepts.

Prozess-Engineering-Prozessmodelle

Prozess-Engineering-Prozesse sind Aktivitäten, die die Evolution von Wissen unterstützen. Hierzu gehören Messprozesse, Modellierungsprozesse, Verbesserungsprozesse und Wiederverwendungsprozesse.

Geschäftsprozesse und soziale Prozesse

Neben den Lebenszyklusmodellen und den Engineering-Prozessmodellen gibt es Prozessmodelle, die keine ingenieurmäßigen Aspekte direkt betreffen, aber dennoch in der Software-Entwicklung eine wichtige Rolle spielen. Hierzu gehören Geschäftsprozesse (z. B. Rechnungsprüfung, Buchhaltung) und soziale Prozesse (z. B. Gruppenprozesse, Kommunikation, Konfliktlösung). Die Software-Entwicklung ist eine mensch-basierte Disziplin, deren Erfolg entscheidend davon abhängt, dass Personen zusammenarbeiten und effektiv kommunizieren. Daher spielen soziale Prozesse und weitere Prozesse aus dem nicht-ingenieurmäßigen Bereich eine wichtige Rolle. Die Interaktion solcher verschiedener Prozessarten muss bei der Planung, Durchführung und Analyse von Software-Projekten beachtet werden.

Vorgehensmodelle

Im Deutschen wird häufig der Begriff *Vorgehensmodell* verwendet. Hiermit ist meist ein standardisiertes Prozessmodell gemeint, das ein Lebenszyklusmodell und mehrere Engineering-Prozessmodelle umfasst. Beispiele für Vorgehensmodelle sind nationale Standards (z. B. das britische PRINCE2-Vorgehensmodell für Software-Projekte der öffentlichen Hand) oder firmenspezifische Standards. Oftmals werden auch zu Methoden gehörende Prozessmodelle als Vorgehensmodelle bezeichnet (z. B. der Rational Unified Process).

Prozesse aus unterschiedlichen Blickwinkeln

Der Begriff „Prozess“ wurde bisher in einem absoluten Sinne als Aktivität in der realen Welt definiert. In der Praxis existieren jedoch unterschiedliche Vorstellungen von einem Prozess. Ein Prozess, der gerade durchgeführt wird, wird beispielsweise anders wahrgenommen als ein Prozess, der bereits durchgeführt wurde oder ein Prozess, der erst in der Zukunft durchgeführt werden soll. Darüber hinaus gibt es unterschiedliche Vorstellungen über einen Prozess, wenn man ihn aus den Blickwinkeln unterschiedlicher Rollen betrachtet. Beispielsweise hat derjenige, der einen Prozess vorgibt, oftmals eine andere Vorstellung von einem Prozess als derjenige, der den Prozess ausführt. Nach Bandinelli et al. [1] kann man den Sollprozess, den offiziellen Prozess, den wahrgenommenen Prozess, den tatsächlichen Prozess und den beobachteten Prozess unterscheiden (siehe Abb. 3.3 auf der nächsten Seite).

Prozessterminologie im Sprachgebrauch

Die in diesem Kapitel eingeführten Begriffe werden im täglichen Sprachgebrauch häufig unterschiedlich und unscharf verwendet, da sich noch keine reife Terminologie im Gebiet „Software-Prozesse“ gefestigt hat. Dies beruht einerseits auf unterschiedlichen parallelen Entwicklungen wie z. B. der Entwicklung verschiedener Prozessnotationen. Andererseits unterliegt das Gebiet wichtigen Einflüssen aus anderen Disziplinen wie z. B. der Geschäftsprozessmodellierung. Es ist daher häufig notwendig, aus dem Zusammenhang auf die Bedeutung eines Begriffes zu schließen: Statt „Software-Prozess“ wird beispielsweise vielfach einfach von einem

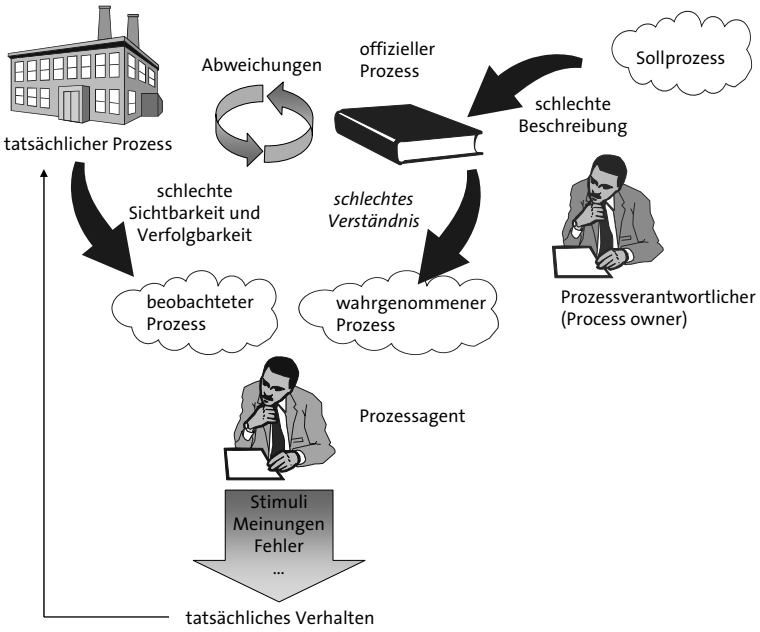

Abbildung 3.3 *Transformation vom Sollprozess zum tatsächlichen Prozess [1]*

„Prozess“ gesprochen, sofern klar ist, dass es sich um eine Aktivität im Rahmen der ingenieurmäßigen Software-Entwicklung handelt. Der Begriff „Prozess“ wird häufig auch benutzt, wenn eigentlich ein „Prozessmodell“ gemeint ist. Oftmals werden unter dem Begriff „Prozess“ neben der Aktivität auch alle zugehörigen Produkte, Rollen und weitere Elemente verstanden.

3.3 Besonderheiten von Entwicklungsprozessen für eingebettete Software

Die besonderen Eigenschaften eingebetteter Software (insbesondere ihre Kapselung in Hardware und ihre starke Interaktion mit dem physikalischen System in der Umgebung) haben erheblichen Einfluss auf den Entwicklungsprozess. Die Prozessarchitektur ist in der Regel nicht betroffen, da sie die konzeptuellen Bausteine von Prozessmodellen beschreibt, jedoch nicht die Inhalte.

An dieser Stelle sollen ausgewählte typische Eigenschaften eingebetteter Systeme und ihre Auswirkungen auf den Entwicklungsprozess aufgezeigt werden. Aufgrund der Unterschiede zwischen verschiedenen Arten eingebetteter Systeme (z. B. kritische, unkritische Systeme) lässt sich kein allgemein gültiger Entwicklungsprozess für eingebettete Software definieren. Für die Entwicklung eingebetteter Software ist daher die Etablierung bzw. Auswahl und Anpassung domänenspezifischer Prozesse

notwendig, die auf den speziellen Typ des eingebetteten Systems und den Entwicklungskontext abgestimmt sind.

3.3.1 Interdisziplinäre Entwicklung

Software-Entwicklung als Teil der System-Entwicklung

Der Prozess der Entwicklung eingebetteter Software ist einer von mehreren Subprozessen des System-Engineerings und somit Teil eines Entwicklungsvorhabens, an dem in der Regel unterschiedliche Disziplinen (z. B. mechanisches Engineering, elektrisches Engineering, Software-Engineering) beteiligt sind. Die abgestimmte Zusammenarbeit dieser unterschiedlichen Disziplinen ist eine der größten Herausforderungen der Entwicklung eingebetteter Systeme. Die Entwicklung eingebetteter Software hat hierbei einen zunehmend wichtigeren Anteil, da immer mehr Funktionalität anderer Disziplinen durch Software realisiert wird. Software-Funktionen sind beispielsweise zur Realisierung von elektronischen Steuerungsfunktionen geeignet und ermöglichen in der Regel höhere Freiheitsgrade und Entwurfsspielräume als Hardware-Realisierungen. Dies führt zu einer größeren Verflechtung der Software-Entwicklung mit anderen Disziplinen und hat zur Konsequenz, dass die Entwicklung eingebetteter Software nicht isoliert betrachtet werden kann. Nach einer Studie von Graaf et al. [7] wird die Entwicklung eingebetteter Systeme derzeit hauptsächlich von der Hardware-Entwicklung, also von mechanischen und elektronischeren Gesichtspunkten, getrieben. Dies kann soweit gehen, dass Software-Architekten nicht an Entwurfsentscheidungen auf Systemebene beteiligt werden und in der Konsequenz die Software-Entwicklung häufig erst dann startet, wenn Hardware-Änderungen bereits teuer werden. Durch getroffene Entwurfsentscheidungen für Hardware wird somit der Gestaltungsraum für Software oftmals unnötig eingeschränkt. Software dient dann vielfach als behelfsmäßiger Integrator, d.h., Probleme aus der Hardware-Domäne werden in der Software-Domäne gelöst. Erschwerend kommt hinzu, dass eingebettete Software oftmals von fachfremden Personen entwickelt wird. Durch systematische Berücksichtigung der Software-Entwicklung im System-Entwicklungsprozess und Einbeziehung von Software-Ingenieuren (z. B. Software-Architekten) kann dem begegnet werden.

Konsequenzen für den Entwicklungsprozess

Die Entwicklung von Subsystemen unterschiedlicher Disziplinen (Mechanik, Elektronik, Software, Optik etc.) erfordert, dass die verschiedenen Entwicklungsprozesse aufeinander abgestimmt, synchronisiert und in einen übergeordneten System-Entwicklungsprozess integriert werden. Darüber hinaus ist ein gemeinsames Verständnis des Problems und der Prozesse wichtig. Insbesondere ist für die Softwareentwicklung ein genaues Verständnis der umgebenden Hardware essentiell, da eingebettete Software in ein sie umgebendes Hardware-System integriert ist, das vielfach anwendungsspezifisch ist. Das Vorhandensein offener Plattformen, die eine einfache Integration unterschiedlicher Systembestandteile ermöglichen, ist bei eingebetteten Systemen (noch) unüblich.

Vielfach ist die Komplexität eingebetteter Systeme ursächlich in ihren dynamischen Verhalten begründet (und weit weniger in ihrer Architektur). Aus der starken Interaktion eingebetteter Software mit der physikalischen Umgebung resultiert dann auch eine Verhaltensorientierung des Prozesses, d. h., dass Aktivitäten zur Spezifikation, Modellierung und Überprüfung dynamischer Zusammenhänge eine größere Bedeutung haben als beispielsweise bei Informationssystemen und dort andere Verfahren eingesetzt werden.

Nach Graaf et al. [7] erfolgt eine Aufteilung in Subsysteme in der Praxis häufig gemäß folgendem Schema: Ausgehend von einer System-Anforderungsdefinition wird ein System-Entwurf erstellt, der eine Partitionierung in multidisziplinäre und monodisziplinäre Subsysteme vornimmt. Dies beinhaltet in der Regel auch eine Partitionierung der funktionalen und nichtfunktionalen Anforderungen auf die entstehenden Subsysteme. Ein multidisziplinäres Subsystem wird durch verschiedene Disziplinen (z. B. Software, Mechanik, Optik, Elektronik) implementiert. Ein monodisziplinäres Subsystem wird durch eine Disziplin implementiert. Darauf werden die Anforderungen an die Subsysteme festgelegt und der Partitionierungsprozess wird für jedes Subsystem wiederholt. Schließlich werden die Subsysteme auf die Ebene von monodisziplinären Subsystemen oder Komponenten heruntergebrochen. In der Praxis variiert das Abstraktionsniveau der resultierenden monodisziplinären Subsysteme bzw. Komponenten. In einigen Fällen resultiert schon der erste Dekompositionsschritt in einer Menge von monodisziplinären Subsystemen, in anderen Fällen bleiben multidisziplinäre Subsysteme über mehrere Dekompositionsschritte erhalten. Durch die parallele Entwicklung unterschiedlicher Disziplinen bekommt die horizontale Kommunikation (zwischen Entwicklern unterschiedlicher Subsysteme auf gleichem Abstraktionslevel) neben der vertikalen Kommunikation (zwischen System-, Subsystem und Komponentenentwicklern auf unterschiedlichen Abstraktionsebenen) eine große Bedeutung. Sie muss durch den Prozess unterstützt werden. Dies kann durch entsprechende Prozessschnittstellen und Kommunikationskanäle erfolgen.

Schäuffele und Zurawka [17] geben ein Beispiel für einen solchen Partitionierungsprozess in der Fahrzeugentwicklung: Auf übergeordneter Systemebene erfolgt eine Partitionierung in die Subsysteme Antriebsstrang, Fahrwerk, Karosserie, sowie Multi-Media. Darauf erfolgt stufenweise die weitere Unterteilung der Subsysteme in weitere Subsysteme und Komponenten. Nach der arbeitsteiligen Entwicklung der einzelnen Subsysteme/Komponenten erfolgt die stufenweise Integration mit begleitender Validation. Abschließend werden die Subsysteme Antriebsstrang, Fahrwerk, Karosserie und Multi-Media zum Fahrwerk integriert. Auf der Ebene der Entwicklung elektronischer Systeme im Fahrzeug kann eine Partitionierung in die Subsysteme Steuergeräte-Software, Steuergeräte-Hardware, Sollwertgeber, Sensoren und Aktuatoren erfolgen. Nach deren arbeitsteiliger Entwicklung werden die Subsysteme wieder stufenweise integriert zu einem elektronischen System. In Vorgehensmodellen

für eingebettete Systeme führt die Partitionierung in Subsysteme unterschiedlicher Disziplinen und deren Integration häufig zu einer Vervielfachung des logischen V-Modells auf der Ebene von Subsystemen (siehe Abb. 3.4). Im Unterschied zu Informationssystemen ist bei eingebetteten Systemen auch die klassische Produktion zu beachten, so dass der Entwicklungsprozess letztlich noch auf den Produktionsplanungs- und Produktionsprozess abgestimmt werden muss.

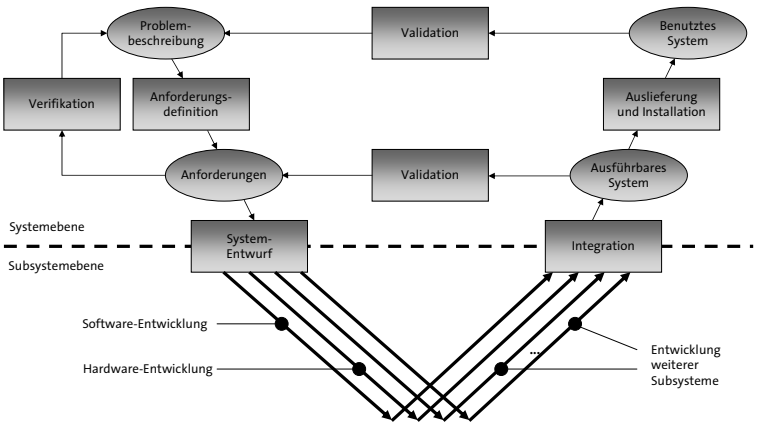

Abbildung 3.4 *Multidisziplinäre Entwicklung*

3.3.2 Unternehmensübergreifende Entwicklung und Simultaneous Engineering

Firmenübergreifende, verteilte Entwicklung

Da an der Entstehung eingebetteter Systeme oftmals unterschiedliche Fachdisziplinen beteiligt sind, ist firmenübergreifende, verteilte Entwicklung typisch. Beispielsweise besteht in der Automobilindustrie eine enge Verzahnung zwischen den Entwicklungsprozessen von OEMs (Original Equipment Manufacturers) und Zulieferern. Es ist abzusehen, dass mit steigender Komplexität eingebetteter Systeme immer mehr Firmen bzw. Entwicklungsorganisationen zusammenarbeiten müssen. Ähnlich den Lieferantenketten in der Automobilindustrie kann es auch bei Entwicklungsvorhaben mehrstufige Abhängigkeiten geben.

Ein weiterer Trend in diesem Zusammenhang ist die vermehrte Einbindung vorgefertigter Komponenten in eingebettete Systeme. Kürzere Entwicklungszeiten erfordern darüber hinaus mehr und mehr die Parallelisierung von Entwicklungsaufgaben, die auch mit dem Begriff Simultaneous Engineering bezeichnet wird. Simultaneous Engineering erfordert die verzahnte Entwicklung einzelner Inkremente bzw. Funktionen. Eine große Herausforderung hierbei ist die durchgängige Synchronisation der Inkremententwicklungen.

Die firmenübergreifende Entwicklung erfordert klare Arbeitsteilungen, definierte Schnittstellen und die Festlegung von Verantwortlichkeiten. Die Gestaltung des Entwicklungsprozesses muss hierauf ausgerichtet sein. Darüber hinaus sind organisatorische und rechtliche Aspekte zu beachten. Bei verteilter Entwicklung ist die Etablierung von Integrationsmechanismen erforderlich (z. B. in Bezug auf Entwicklungs- und Testumgebungen oder Werkzeuge). Es gibt eine Vielzahl von Methoden und Prozessen, die eine firmenübergreifende Entwicklung und die Einbeziehung vorgefertigter Komponenten unterstützen, wie etwa Prozesse für Lieferantenmanagement oder die COTS-Auswahl [13].

Konsequenzen für den Entwicklungsprozess

3.3.3 Nichtfunktionale Eigenschaften

Bedeutung und Ausprägung nichtfunktionaler Eigenschaften

Nichtfunktionale Eigenschaften wie Echtzeitfähigkeit, Zuverlässigkeit, Fehlertoleranz und Sicherheit haben bei eingebetteter Software eine besondere Bedeutung. Dies hat seine Ursache vor allem in den Anwendungsgebieten eingebetteter Systeme. Echtzeitfähigkeit ist beispielsweise in der Robotersteuerung von großer Bedeutung. Zuverlässigkeit und Fehlertoleranz spielen eine große Rolle bei sicherheitskritischen Anwendungen wie Flugzeugsteuerungen. Funktionssicherheit ist im Automobilbau sehr wichtig, da es bei Unfällen zu Personenschäden kommen kann. Eingebettete Software muss oftmals (weitgehend) fehlerfrei sein: Zum einen erwartet der spätere Benutzer ein korrekt arbeitendes System, zum anderen ist der Austausch der Software manchmal zu kostenintensiv und somit kann auch keine Software-Wartung stattfinden. Hinzu kommt, dass einige nichtfunktionale Eigenschaften (z. B. Benutzbarkeit und Ergonomie) im Umfeld von eingebetteten Systemen anders ausgeprägt sind als bei Informationssystemen.

Konsequenzen für den Entwicklungsprozess

Die Behandlung nichtfunktionaler Eigenschaften bei der Entwicklung eingebetteter Systeme erfordert in der Regel spezielle Techniken bzw. Produktnotationen, die in den Prozess integriert werden müssen. Hohe Qualität eingebetteter Software wird insbesondere durch frühzeitige Erkennung bzw. systematische Vermeidung von Fehlern erzielt. Entsprechende Fehlerentdeckungs- und Fehlerbehebungsprozesse sowie Strategien zur systematischen Fehlervermeidung sollten daher Bestandteil des Entwicklungsprozesses sein.

3.3.4 Wirtschaftlichkeit

Wirtschaftliche Herausforderungen

Entwicklungskosten und -zeit spielen bei der Entwicklung eingebetteter Systeme eine große Rolle. Der Markt für eingebettete Systeme ist einer der am stärksten wachsenden Märkte im Informations- und Telekommunikationssektor. Hieraus resultiert die Forderung nach immer kürzer werdenden Entwicklungszeiten sowie kosteneffizienteren Entwicklungs- und Produktionsverfahren.

Konsequenzen für den Entwicklungsprozess

Aus den wirtschaftlichen Anforderungen resultiert die besondere Beachtung der Plan- und Kontrollierbarkeit des Prozesses, die beispielsweise durch quantitative Messungen unterstützt werden kann. Außerdem werden die Herstellungskosten in der Regel durch die Preise für die Elektronik-Hardware dominiert, was sich in sehr begrenzten Hardware-Ressourcen (z. B. hinsichtlich Speicherplatz, Energieverbrauch, Rechenleistung) niederschlagen kann. Entsprechend müssen die Software-Entwicklungsprozesse angepasst werden (z. B. durch Einführung von Optimierungsschritten).

3.3.5 Lange Produktlebenszyklen

Hohe Wartbarkeitsanforderungen

Lange Produktlebenszyklen sind für viele Arten von eingebetteten Systemen charakteristisch. Beispielsweise geht man im Automobilbau von rund drei Jahren Entwicklungszeit, etwa sieben Jahren Produktionszeit und von bis zu 15 Jahren für eine anschließende Betriebs- und Servicephase aus. Insgesamt ist der Produktlebenszyklus rund 25 Jahre lang [17]. Lebenszyklen elektronischer Bauteile sind aufgrund der raschen technologischen Fortschritte wesentlich kürzer. Dies verursacht Kompatibilitätsprobleme und stellt Herausforderungen an die Lagerung passender Bausteine.

Konsequenzen für den Entwicklungsprozess

Die langen Produktlebenszyklen müssen bei der Entwicklung eingebetteter Systeme berücksichtigt werden, insbesondere müssen die Entwicklungsprozesse auf Wartungs- und Serviceprozesse abgestimmt sein. So ist z. B. schon bei der Entwicklung eingebetteter Software darauf zu achten, dass Elektronikkomponenten sich im Laufe der Zeit ändern können. Dem kann beispielsweise durch standardisierte Architekturen und hardwareunabhängig programmierte Funktionen begegnet werden. Manche Komponenten eingebetteter Systeme müssen auch in regelmäßigen Abständen ausgetauscht werden. So werden heutzutage beispielsweise Druckerpatronen vielfach softwaremäßig deaktiviert, wenn sie eine bestimmte Einsatzzeit überschritten haben. Bei immer mehr eingebetteten Systemen besteht heute die Möglichkeit, die Software im Laufe des Lebenszyklus zu modifizieren (z. B. durch Flash-Technologie). Auch hierfür müssen Vorkehrungen getroffen werden.

Es gibt eine Reihe weiterer typischer Charakteristika eingebetteter Systeme, die Einfluss auf den Entwicklungsprozess haben. Erwähnt sei die zunehmende Notwendigkeit, Varianten eingebetteter Systeme (z. B. Motorsteuerungen für unterschiedliche Fahrzeugtypen) systematisch zu entwickeln. Hierzu eignet sich insbesondere der Einsatz von Produktlinienansätzen (wie z. B. PuLSE® [3]).

Eine wesentliche Herausforderung für software-entwickelnde Organisationen ist die Etablierung domänenspezifischer Prozessmodelle, die an die jeweiligen Entwicklungsumgebungen und Randbedingungen angepasst sind. Hierzu müssen entsprechende Prozessmodelle, Techniken,

Methoden und Werkzeuge, die den besonderen Eigenschaften eingebetteter Systeme gerecht werden, ausgewählt, an die konkrete Entwicklungsumgebung angepasst, erprobt und in einen übergeordneten systematischen Entwicklungsprozess integriert werden. Für eine solche Auswahl und Anpassung ist es besonders wichtig, die Auswirkungen unterschiedlicher Prozesse, Techniken, Methoden und Werkzeuge im Hinblick auf die Erreichung von Projekt- und Qualitätszielen in unterschiedlichen Kontexten zu kennen. Erkenntnisse dieser Art können mit Hilfe von empirischen Studien gewonnen werden.

3.4 Etablierung domänenspezifischer Prozessmodelle

Es gibt keinen Standardprozess für die Entwicklung eingebetteter Software, der uneingeschränkt anwendbar ist. Vielmehr erfordert die Entwicklung eingebetteter Software effiziente Entwicklungsprozesse, die auf die Besonderheiten der Entwicklungsorganisation sowie weitere Randbedingungen (Anwendungsdomäne, technische Besonderheiten etc.) abgestimmt sind [11]. Die Gründe sind unter anderem, dass Software individuell erstellt (oder: entwickelt) wird und dass die Prozesse weitgehend mensch-basiert sind. Eine wichtige Aufgabe einer Entwicklungsorganisation ist, effektive und an ihre spezifischen Randbedingungen angepasste Entwicklungsprozessmodelle zu definieren, zu dokumentieren und zu etablieren, so dass sie auch tatsächlich gelebt werden. Nur gelebte Prozessmodelle können vermessen, analysiert und verbessert werden.

Gewinnung von Prozessmodellen

Es gibt verschiedene Strategien, passende Entwicklungsprozessmodelle zu definieren. Eine Möglichkeit ist die *theoretische Modellgewinnung*, bei der basierend auf theoretischen Überlegungen Prozesse entwickelt werden. Dies kann insbesondere dann erforderlich sein, wenn drastische Änderungen an Prozessen vorgenommen werden müssen. Hierbei können existierende Vorgehensmodelle, die Spezifika von eingebetteter Software berücksichtigen, als Orientierung oder Vorlage dienen. Eine andere Möglichkeit ist die *deskriptive Modellgewinnung*, die sich weitgehend auf die Beobachtung der tatsächlich durchgeführten Prozesse stützt. Oftmals findet man eine Kombination beider Herangehensweisen bei der Definition von Prozessen. Im Folgenden wird eine Auswahl existierender Vorgehensmodelle vorgestellt, der deskriptive Ansatz der Modellgewinnung näher erläutert sowie beispielhaft ein Ansatz zur Prozessverbesserung vorgestellt.

3.4.1 Existierende Vorgehensmodelle für die Entwicklung eingebetteter Software

Auf der Ebene von Engineering-Prozessen gibt es eine Vielzahl von Methoden, Techniken und Werkzeugen, die bestimmte Aspekte der Entwicklung eingebetteter Systeme unterstützen. Beispiele sind RTSA (Real-Time Structured Analysis), DARTS (Design Approach for Real-Time Systems) und MCSE (Electronic System Design Methodology). Oftmals fokussieren diese Technologien auf Produktnotationen und Analysetechniken für Produkteigenschaften und weniger auf den eigentlichen Entwicklungsprozess. Auf höheren Abstraktionsebenen gibt es nur wenige Vorgehensmodelle, die den gesamten Lebenszyklus abdecken. Hierzu zählen nationale und internationale Standards, domänenspezifische Modelle, umfassende Entwicklungsmethoden sowie firmenspezifische Modelle. Viele dieser Vorgehensmodelle basieren auf theoretischen Vorgaben und sind nicht empirisch abgeleitet.

Vorgehensmodelle für eingebettete Software und eingebettete Systeme

Im Folgenden wird eine Liste ausgewählter Vorgehensmodelle gegeben, die u.a. für die Entwicklung eingebetteter Software bzw. eingebetteter Systeme erstellt wurden:

- V-Modell XT
  - Vorgehensmodell für Entwicklungsvorhaben im öffentlichen und Verteidigungsbereich
  - Quelle: http://www.v-modell-xt.de/
- ISO/IEC 15288 Standard
  - Internationaler Standard mit Vorgehensmodell für die System-Entwicklung
  - Quelle: www.iso.org
- ROPES (Rapid Object Oriented Process for Embedded Systems)
  - Iteratives, UML-basiertes Vorgehensmodell
  - Quelle: [6]
- Automotive Software Engineering
  - Domänenspezifisches Vorgehensmodell für die Automobiltechnik
  - Quelle: [17]
- WISEP (Wireless Internet Service Engineering Process)
  - Domänenspezifisches Vorgehensmodell für drahtlose Internetdienste
  - Quelle: [14]

- MARMOT
  - Vorgehensmodell zur Entwicklung eingebetteter, sicherheitskritischer Realzeit-Systeme
  - Quelle: www.marmot-projekt.de
- RTE-Vorgehensmodell
  - Agiles Vorgehensmodell für Echtzeitsysteme und eingebettete Systeme
  - Quelle: [8]
- SPRINT
  - Firmenspezifisches Vorgehensmodell der Porsche AG zur Einbettung von Software-Prozessen in die Fahrzeugentwicklung
  - Quelle: [19]

3.4.2 Deskriptive Prozessmodellierung

Deskriptive versus präskriptive Prozessmodellierung

Deskriptive Prozessmodelle entstehen durch Beobachtung der tatsächlich durchgeführten Prozesse. Deskriptive Prozessmodelle eignen sich besonders für Prozessanalysen und als Ausgangspunkt für Prozessverbesserungsmaßnahmen. Sie beschreiben Prozesse, wie sie in der Realität stattfinden (engl. as-is-models), und können daher besonders gut für die Analyse des Ist-Standes angewendet werden. Man kann sie insbesondere für das Aufdecken von Schwachstellen und Verbesserungspotentialen nutzen. Darüber hinaus lassen sie sich als Baseline verwenden, auf der Prozessänderungen durchgeführt bzw. neue Prozesse definiert werden.

Präskriptive Prozessmodelle hingegen beschreiben, wie Software entwickelt werden soll und machen Vorgaben für die Durchführung von Prozessen. Solche Modelle werden typischerweise zur Anleitung (engl. should-be models) oder zur Erzwingung (engl. to-be models) von Prozessen in Organisationen verwendet.

Die Bezeichnungen deskriptiv und präskriptiv deuten nicht auf inhärente Eigenschaften von Prozessmodellen hin, sondern geben vielmehr Auskunft über ihren Verwendungszweck. In der Regel findet die Evolution von Prozessmodellen durch zyklische Erstellung deskriptiver und präskriptiver Prozessmodelle statt (siehe Abb. 3.5 auf der nächsten Seite). Hierbei fließen sowohl theoretische Überlegungen als auch Beobachtungen in die Definition von Prozessmodellen ein. Ein solcher Zyklus führt darüber hinaus auch immer wieder zu einem Abgleich der vorgegebenen Modelle mit der Realität.

Deskriptive Modellierung ist in der Praxis mit einigen Herausforderungen konfrontiert: Beispielsweise finden Prozesse auf Instanzenebene

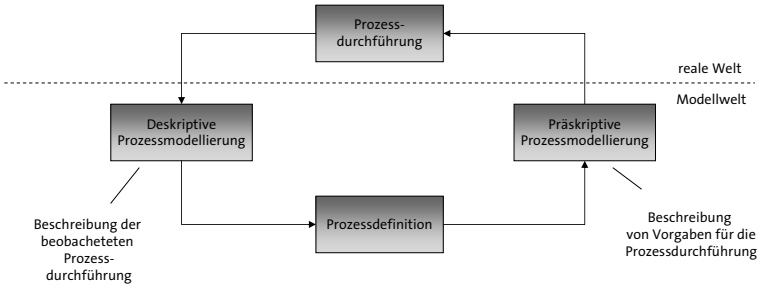

Abbildung 3.5 *Deskriptive und präskriptive Prozessmodellierung*

statt, präskriptive Prozessmodelle beschreiben Prozesse hingegen eher auf Typebene. Hier gilt es, den richtigen Abstraktionsgrad zu finden. Darüber hinaus sind Software-Entwicklungsprozesse nicht repetitiv. Dies bedeutet, dass gemeinsame und unterschiedliche Anteile von Prozessvarianten beschrieben werden müssen und die jeweiligen Differenziatoren identifiziert werden müssen. Weitere Herausforderungen in der Praxis sind, dass das Wissen über Prozesse vielfach „verstreut" ist und es oftmals keine klare Trennung von Rollen gibt, besonders wenn sie von einer Person gespielt werden.

Vorgehensweise zur deskriptiven Prozessmodellierung

Zur Durchführung der deskriptiven Modellierung wird von Becker, Hamann und Verlage [5] eine aus zwei Phasen bestehende Prozedur bestehend aus acht Schritten empfohlen. Die Konfigurierungsphase umfasst die Auswahl und Bereitstellung des Prozessmodellierungsansatzes und der benötigten Infrastruktur. Die Schritte dieser Phase werden selten durchgeführt. Die Ausführungsphase wird in der Regel pro Modellierungszweck einmal ausgeführt. Die Phasen und Schritte sind im Folgenden aufgezählt:

- Phase I: *Konfigurierung („set-up")*
  - Schritt 1: Definition der Ziele und des Modellierungsbereichs
  - Schritt 2: Auswahl einer Prozessarchitektur
  - Schritt 3: Auswahl einer (mehrerer) Prozessmodellierungsnotation(en)
  - Schritt 4: Auswahl und/oder Anpassung von Werkzeugen
- Phase II: *Ausführung („execution")*
  - Schritt 5: Erfassung von Prozesswissen („process elicitation")
  - Schritt 6: Erstellung des Modells
  - Schritt 7: Analyse des Prozessmodells
  - Schritt 8: Analyse des Prozesses

Die deskriptive Prozessmodellierung hat sich in vielen Bereichen bewährt, beispielsweise zur Erfassung rollenspezifischer Sichten auf Entwicklungsprozesse [18] oder zur Definition von Entwicklungsprozessen

für neue Anwendungsdomänen [4]. Deskriptive Prozessmodellierung löst zum Teil sofortige Verbesserungen der Prozesse bzw. ein gründliches Hinterfragen und Überdenken der existierenden Prozesse aus. Erfahrungen mit der deskriptiven Modellierung zeigen, dass die Absicht hinter der Modellierung einen großen Einfluss auf das Modell hat (beispielsweise in Bezug auf Detaillierungsgrad oder Darstellung). Graphische Prozessdarstellungen sind hilfreich für die Erstellung und die Überprüfung des Prozessmodells durch die beteiligten Personen. Zu beachten ist auch, dass die befragten Personen nicht immer den tatsächlichen Prozess schildern (siehe Abb. 3.3 auf Seite 31).

3.4.3 Kontinuierliche Prozessverbesserung

Zertifizierung, Assessment und kontinuierliche Prozessverbesserung

Ein Grundprinzip ingenieurmäßiger Software-Entwicklung ist die Gewinnung und Nutzung von Erfahrungen zum Zwecke der Verbesserung. Im Software-Bereich haben sich unterschiedliche Ansätze etabliert, die zur Prozessverbesserung herangezogen werden. Solche Ansätze bieten einen Rahmen zum Thematisieren von Problemen und Suchen von Lösungen, wobei die Beteiligten durch eine systematische Herangehensweise unterstützt werden. Die Ansätze lassen sich grob klassifizieren in Zertifizierungsansätze (z. B. ISO 9000), Assessment-Ansätze (z. B. SPICE, CMMI), und kontinuierliche Verbesserungsansätze (z. B. PDCA, QIP). Während Zertifizierungsansätze zur Überprüfung von grundlegenden Praktiken dienen, bieten Assessment-Ansätze in der Regel zusätzlich einen Verbesserungsweg an. Kontinuierliche Verbesserungsansätze fokussieren auf wichtige Aspekte und ermöglichen eine selbstgesteuerte, kontinuierliche Verbesserung.

Quality Improvement Paradigm (QIP)

Ein Beispiel für einen kontinuierlichen Verbesserungsansatz ist das speziell auf die Belange der Software-Entwicklung zugeschnittene Qualitätsverbesserungsparadigma QIP (Quality Improvement Paradigm) [2], das im Folgenden kurz skizziert wird. Das QIP besteht aus sechs Aktivitäten, die für jedes Entwicklungsprojekt zu planen und durchzuführen sind:

- Schritt 1: *Charakterisiere („characterize"):* Stelle diejenigen Charakteristika des Projekts fest, die notwendig sind, um die geeigneten Modelle (z.B. Prozessmodelle, Qualitätsmodelle) zu identifizieren.
- Schritt 2: *Definiere Ziele („set goals"):* Definiere die Ziele des Projekts in messbarer Form. Dazu sind Metriken und Soll-Werte für alle Dimensionen eines Ziels nötig (Objekt, Blickwinkel, Qualitätsmerkmal).
- Schritt 3: *Wähle Projektplan („choose process"):* Erstelle den Projektplan durch (a) Auswahl der für die identifizierten Charakteristika und Ziele geeigneten Modelle, (b) Instrumentierung der Modelle entsprechend geeigneter Qualitätsmodelle und (c) Auswahl der unterstützenden Methoden und Werkzeuge (Entwicklung und Messen).

- Schritt 4: *Führe aus („execute"):* Führe das Projekt entsprechend dem gewählten Plan aus. Dies beinhaltet die Erfassung der Ist-Werte für alle Metriken und das Erstellen von Rückkopplung an das Projektteam auf der Basis eines Vergleichs von Ist- und Sollwerten.
- Schritt 5: *Analysiere („analyze"):* Nach Beendigung des Projekts wird untersucht, inwieweit verwendete Modelle Stärken und Schwächen offenbart haben. Derartige Erkenntnisse bilden die Grundlage für die Überarbeitung der Modellbasis.
- Schritt 6: *Überarbeite die Modellbasis („package"):* Auf der Basis von Erkenntnissen aus Projekten werden bestehende Modelle überarbeitet. Dies kann z. B. eine Formalisierung der Modellpräsentation, eine Abstraktion oder Präzisierung oder die Formulierung eines neuen Modells zur Folge haben. Entsprechend den Unstetigkeiten von Modellen (d. h., für unterschiedliche Umgebungen können verschiedene Modelle gelten) kann es vorkommen, dass sich Modellhierarchien herausbilden. Generelles Ziel dieser Aktivität ist es, die Wahrscheinlichkeit der Wiederverwendung von Modellen in zukünftigen Projekten zu erhöhen.

Die Integration von Prozessmodellen in den Projektplan erfolgt beim QIP im dritten Schritt auf der Grundlage der in den ersten beiden Schritten erzielten Ergebnisse. Im letzten Schritt werden die Prozessmodelle auf Verbesserungsmöglichkeiten hin untersucht, um gewonnene Erfahrungen für zukünftige Projekte zu sichern. Sie werden in einer so genannten Erfahrungsdatenbank gespeichert, so dass ein projektübergreifender Verbesserungsprozess ermöglicht wird.

3.5 Zusammenfassung und Ausblick

Voraussetzung für die entscheidende Verbesserung von Entwicklungsprozessen für eingebettete Software bzw. eingebettete Systeme ist ihre Integration in ein ingenieurmäßiges Software-Entwicklungsmodell. Dies erfordert a) die genaue (messbare) Definition von Zielen für das zu entwickelnde (Software-)System, b) die Auswahl und geeignete Anpassung von Prozessen, Techniken, Methoden und Werkzeugen zur Erfüllung der vorgegebenen Ziele, c) die Gewährleistung der Einhaltung vorgegebener Ziele durch geeignete Maßnahmen sowie d) die Aufbereitung von Erkenntnissen und Erfahrungen aus Projekten für die zukünftige Nutzung. Die explizite Modellierung von Entwicklungsprozessen ist eine wesentliche Grundlage der ingenieurmäßigen Software-Entwicklung.

Prozessmanagement-Trends

Eine Auswahl zukünftiger Trends und Entwicklungen auf dem Gebiet „Prozessmanagement" ist im Folgenden aufgeführt:

- Kostengünstige Generierung von konsistenten und aktuellen Prozessdokumentationen und Webguides für unterschiedliche Zwecke (z. B. Anleitung, Audit-Unterstützung, Training)

- Entwicklung von anpassbaren Entwicklungsprozessen für neue Domänen (z. B. für Ambient Intelligence)
- Quantitatives Prozessmanagement (unterstützt durch Prozessleitstände [12])
- Simulation von Software-Entwicklungsprozessen zur Entscheidungsunterstützung bei der Projektplanung oder Priorisierung von Verbesserungsmaßnahmen
- Umfassende Prozessassessments, die den Einfluss von Prozessen auf Produktqualitäten berücksichtigen

Literaturverzeichnis

[1] Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M. und Picco, G.: *Modeling and improving an industrial software process*. IEEE Transactions on Software Engineering, 1995.

[2] Basili, V. R., Caldiera, G. und Rombach, D.: *The Experience Factory*. In: Marciniak, J. (Herausgeber): *Encyclopedia of Software Engineering*, Band 1, Seiten 469–476. John Wiley & Sons, 1994.

[3] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T. und DeBaud, J.-M.: *PuLSE®: A Methodology to Develop Software Product Lines*. In: *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, Seiten 122–131, Los Angeles, CA, USA, 1999.

[4] Becker-Kornstaedt, U., Boggio, D., Münch, J., Ocampo, A. und Palladino, G.: *Empirically Driven Design of Software development Processes for Wireless Internet Services*. In: Oivo, Markku und Komi-Sirviö, S. (Herausgeber): *Proceedings of the 4th International Conference on Product Focused Software Process Improvement (Profes 2002)*, Lecture Notes in Computer Science 2559, Seiten 351–366. Springer-Verlag, 2002.

[5] Becker-Kornstaedt, U., Hamann, D. und Verlage, M.: *Descriptive Modeling of Software Processes*. In: *Proceedings of the Third Conference on Software Process Improvement (SPI '97)*, Barcelona, Spain, 1997.

[6] Douglass, B. P.: *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Band 1. Addison-Wesley Professional, 1999.

[7] Graaf, B., Lormans, M. und Toetenel, H.: *Embedded Software Engineering: The State of the Practice*. IEEE Software, 20(6): 61–69, 2003.

[8] Hruschka, P. und Rupp, C.: *Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML*, Band 1. Hanser, München, Wien, 2002.

[9] Humphrey, W. S.: *Managing the Software Process*. Addison Wesley Reading, Massachusetts, 1989.

[10] McGarry, F., Pajerski, R., Page, G., Waligora, S., Basili, V. R. und Zelkowitz, M. V.: *An Overview of the Software Engineering Laboratory*. Software Engineering Laboratory Series Report SEL-94-005, Software Engineering Laboratory, Greenbelt, MD, USA, 1994.

[11] Münch, J.: *Anpassung von Vorgehensmodellen im Rahmen Ingenieurmässiger Softwarequalitätssicherung*. In: Kneuper, Ralf und Verlage, M. (Herausgeber): *Tagungsband des 6. Workshops der Fachgruppe 5.1.1 (GI): Vorgehensmodelle, Prozessverbesserung und Qualitätsmanagement*, Seiten 33–42, Kaiserslautern, 1999.

[12] Münch, J. und Heidrich, J.: *Software Project Control Centers: Concepts and Approaches*. International Journal of Systems and Software, 70(1–2): 3–19, 2004.

[13] Morisio, M., Seaman, C. B., Basili, V. R., Parra, A. T., Kraft, S. E. und Condon, S. E.: *COTS-based software development: Processes and open issues*. Journal of Systems and Software, 61(3): 189–199, 2002.

[14] Ocampo, A., Boggio, D., Münch, J. und Palladino, G.: *Towards a Reference Process for Developing Wireless Internet Services*. IEEE Transactions on Software Engineering, 29(12): 1122–1134, 2003.

[15] Rombach, H. D. und Verlage, M.: *Advances in Computers 41*, Kapitel Directions in software process research, Seiten 1–63. Academic Press, 1995.

[16] Royce, W. W.: *Managing the development of Large Software Systems*. In: *Proceedings of Westcon*, Seiten 328–339. IEEE CS Press, 1970.

[17] Schäuffele, J. und Zurawka, T.: *Automotive Software Engineering*. Vieweg, Wiesbaden, 2003.

[18] Verlage, M.: *Multi-view Modelling of Software Processes*. In: Warboys, B. C. (Herausgeber): *Proc. 3rd European Workshop on Software Process Technology*, Nummer 772 in *Lecture Notes Comput. Sci.*, Seiten 123–127, Grenoble, France, 1994. Springer-Verlag.

[19] Zöller, R., Dorn, R., Kohley, A. und Marx, D.: *Prozess SPRINT – Anpassung und Einbettung etablierter Software-Entwicklungsprozesse in die Gesamtfahrzeug-Entwicklung der Porsche AG*. In: *Tagungsband 11. internationaler VDI Kongress „Elektronik im Kraftfahrzeug“*, Baden-Baden, 25.–26. September 2003.

Index