Softwareentwicklung und Softwareevolution als Problemlösungsstrategien

Carsten von Schwichow
Technische Universität Ilmenau
carsten.von.schwichow@tu-ilmenau.de

Maritta Heisel
Universität Duisburg-Essen
heisel@fb9dv.uni-duisburg.de

Abstract: Abstract.

1 Einleitung

2 Ein allgemeines Modell des Problemlösens

Erstellung und Wartung von Software sind problemlösende Verfahren. Von der Erstellung eines neuen Softwareprodukts erhofft sich der Auftraggeber einen bestimmten Nutzen. Oftmals sollen Arbeitsvorgänge effizienter oder zuverlässiger ablaufen als bisher. Bei der Wartung bestehender Software soll in den meisten Fällen Performanz und Zuverlässigkeit verbessert oder die Funktionalität erweitert werden.

In allen genannten Fällen handelt es sich um eine Problemstellung, welche durch die Entwicklung oder Veränderung von Software gelöst werden soll. Um sich bewusst zu machen, mit welchen Aufgaben und welchen Schwierigkeiten die Entwicklung und Wartung von Software verbunden ist, empfiehlt es sich, zunächst eine Vorstellung davon zu erhalten, wie das Lösen von Problemen im Allgemeinen abläuft. Abbildung 1 zeigt die wichtigsten Artefakte, die beim Lösen von Problemen eine Rolle spielen, sowie die Beziehungen zwischen ihnen. In diesem Abschnitt wird auf jedes dieser Artefakte genauer eingegangen und die Beziehungen zwischen ihnen werden erläutert.

2.1 Ziele

Am Anfang einer jeden Problemlösung steht das Problem. Ein wichtiger Aspekt der Beschreibung eines Problems ist die Formulierung der zu erreichenden *Ziele*. Darüber hinaus müssen der gegenwärtige Zustand (Ausgangszustand) der Problemumgebung, sowie die zur Verfügung stehenden Hilfsmittel angegeben werden.

Bei der Entwicklung von Software werden die Ziele überlichweise in Form von Anforderungen formuliert. Diese sollten ausschließlich auf den Problembereich Bezug nehmen. Das heißt, sie sagen nur aus, wie sich das Gesamtsystem aus der fertigen Software und ihrer Umgebung verhalten soll. Die Anforderungen sagen hingegen nichts darüber, wie die

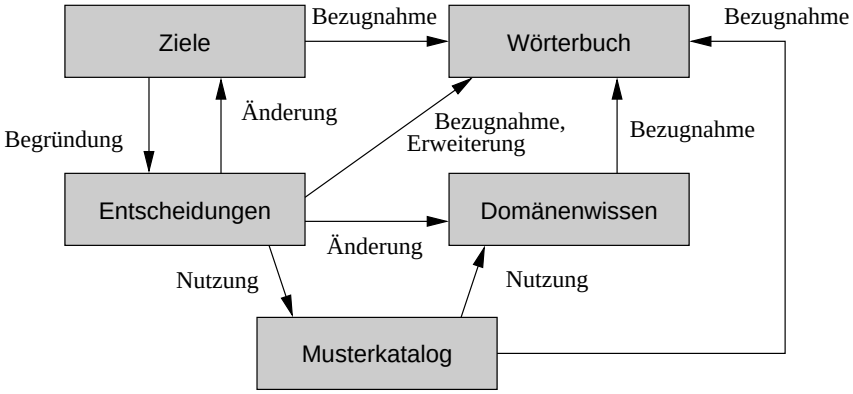

Abbildung 1: Ein allgemeines Modell des Problemlösens

Software dieses Verhalten bewirkt. Das Vokabular, das für die Formulierung der Anforderungen verwendet wird, ist demzufolge ausschließlich Vokabular aus der Problemwelt.

Die Ziele können aber im Verlauf der Problemlösung verfeinert oder anderweitig verändert werden. Dann wird neues Vokabular eingeführt, das nicht zwangsweise aus dem Problembereich stammen muss. In Abschnitt 2.4 wird näher auf diese Problematik eingegangen.

2.2 Wissen

Wer ein Problem systematisch lösen möchte, muss Wissen über die Problemwelt besitzen, denn wer die Vorgänge in der Problemwelt nicht versteht, kann bestenfalls zufällig eine Lösung finden. Das Wissen über den Aufbau und die Funktionsweise der Problemwelt wird üblicherweise als *Domänenwissen* bezeichnet.

Bei ingenieursmäßigem Vorgehen, welches die Softwaretechniker für sich beanspruchen, spielt neben dem Domänenwissen auch die Erfahrung eine wichtige Rolle. Beispielsweise ist es allgemein bekannt, dass eine Brücke eine geeigne Konstruktion ist, um Personen und Fahrzeugen die einfache Überquerung eines Tals oder eines Flusses zu ermöglichen. Auch hierbei handelt es sich um Wissen. Dieses Wissen aber ist – im Gegensatz zum einfachen Domänenwissen – problembezogen. Das heißt, es liefert zu einer bestimmten Art von Problemstellung einen konkreten Lösungsansatz. Bei dieser Art von Wissen spricht man von *Mustern*.

Auch in der Softwareentwicklung sind Muster seit Jahren bekannt, beispielsweise in der Form von Entwurfsmustern. Aber auch Architekturstile, Klassenbibliotheken (z.B. für grafische Oberflächen) und Komponenten können als Muster aufgefasst werden. Typischerweise werden solche Softwaremuster in einem *Musterkatalog* abgelegt, der ein Auffinden von geeigneten Mustern zu einem gegebenen Problem erleichern soll.

2.3 Begriffe

Wie oben erwähnt, werden Begriffe aus der Problemwelt verwendet, um die initialen Ziele zu formulieren. Beispiele wären etwa „Fluss“, „Fahrzeug“, „Fußgänger“ und „überqueren“, wenn es um die oben genannte Überquerung eines Flusses geht. Gleiches gilt für das Domänenwissen und für die Muster. Letztere verwenden darüber hinaus Begriffe aus der Lösungswelt, denn sie ordnen jeder Problemklasse einen Lösungsansatz zu. Beispiele wären hier: „Pfeiler“, „Spannseil“ und „Verankerung“. Um sich über die Bedeutung all dieser Begriffe verständigen zu können, ist es sinnvoll, ein *Wörterbuch* anzulegen, dass für jeden Begriff eine Definition enthält.

2.4 Entscheidungen

Während des eigentlichen Lösungsprozesses werden die Ziele durch *Entscheidungen* sukzessive verfeinert oder anderweitig verändert. Beispielsweise könnte die Entscheidung eine Brücke zu bauen, das ursprüngliche Ziel „Fluss überqueren“ durch eine verfeinerte Variante „Brücken bauen, um Fluss zu überqueren“ ersetzen. Auf diese Weise werden die Ziele durch Entscheidungen beeinflusst. Aber auch in umgekehrter Richtung muss es eine Beeinflussung geben, denn es muss sichergestellt werden, dass die geänderten Ziele im Einklang mit den bisherigen Zielen stehen. Um dies zu erreichen, müssen alle Entscheidungen begründet werden. Als Grundlage dieser Begründungen dienen wiederum die bisher definierten Ziele. Auf diese Art und Weise gibt es eine welchselseitige Beeinflussung zwischen Zielen und Entscheidungen.

Die Absicht ist, durch die immer weiter fortschreitende Verfeinerung der Ziele, die Problembeschreibung so feingranular werden zu lassen, dass sie schließlich unmittelbar als Beschreibung einer Lösung verwendet werden kann. In der Softwareentwicklung kann das zum Beispiel so aussehen, dass zunächst Entscheidungen über die Einführung bestimmter Architekturkomponenten und deren Interaktion getroffen werden. Danach wird entschieden, durch welche Klassen diese Komponenten realisiert werden, und schließlich wird festgelegt, wie die Klassen zu implementieren sind.

3 Einpassung bestehender Softwareprozesse

Das im vorigen Abschnitt beschriebene Modell macht keinerlei Einschränken bezüglich der Formulierung von Zielen und Entscheidungen. Weil dadurch nur sehr wenig Struktur in den Entwicklungsprozess gebracht wird, ist das Modell nicht als Softwareprozess geeignet. Es soll stattdessen als Grundgerüst dienen, um existierende Softwareprozesse besser bewerten und vergleichen zu können. Softwareprozesse können dem entsprechend als Instanzen des allgemeinen Modells aus Abschnitt 2 aufgefasst werden.

Softwareprozesse zeichnen sich gegenüber dem oben beschriebenen Entwicklungsmodell dadurch aus, dass sie zahlreiche Einschränkungen vornehmen, insbesondere hinsichtlich

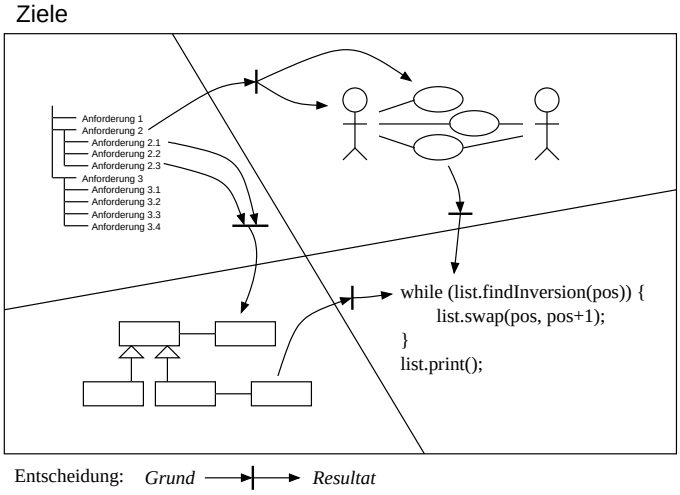

Abbildung 2: Verwendung von Modellen zur Strukturierung des Entwicklungsprozesses

Formulierung von Zielen und Entscheidungen. Für die Beschreibung der Ziele definiert jeder Softwareprozesse eignene Modelle. Jedes solche Modell dient dem Zweck, einen bestimmten Aspekt der Zielbeschreibung herauszustellen und sich dabei auf ein bestimmtes Abstraktionsniveau zu beschränken. Außerdem wird in der Regel für jedes Modell eine Notation vorgeschrieben. Somit bewirkt die Einführung von Modellen eine Partitionierung der Zielbeschreibung, wobei jede Partition einem Modell entspricht.

Da Ziele gemäß Abschnitt 2.4 aufgrund von Entscheidungen formuliert werden, muss jeder Softwareprozess auch Einschränken bezüglich der Entscheidungen vornehmen. Diese Einschränkungen müsses sicherstellen, dass nur zulässige Modelle entsprechend den Definitionen des Softwareprozesses als Ziele erzeugt werden können, und dass die Modelle untereinander konsistent sind. In Abbildung 2 sind Entscheidungen durch Pfeile angedeutet, welche Gründe und Wirkungen der jeweiligen Entscheidung verbinden. Die Entscheidung selbst ist – in Analogie zu einer Transition in einem Petrinetz – als schwarzer Balken dargestellt.

Jeder Softwareprozess definiert eine eigene Menge von Modellen und beschreibt deren Beziehungen untereinander. Im Folgenden werden einige der bekanntesten Softwareprozesse kurz beschrieben und in das allgemeine Entwicklungsmodell aus Abschnitt 2 eingepasst. Es wird dabei ausschließlich auf die für das Entwicklungsmodell relevanten Eigenschaften des jeweiligen Softwareprozesses eingegangen. Andere Aspekte, wie Prozess- und Projektmanagement werden nicht berücksichtigt.

3.1 Rational Unified Process

- iterativ und inkrementell (Cycles)

- 4 Phasen
  - Inception (Vorbereitung)
  - Elaboration (Ausarbeitung)
  - Construction (Herstellung)
  - Transition (Übergabe)
- 4 Meilensteine
  - Lifecycle Objective
  - Lifecycle Architecture
  - Initial Operational Capability
  - Product Release
- Architektur(-zentriert)
- Use-Case(-getrieben)

3.2 Cleanroom

- inkrementell
- Architektur
- Box Structure
  - black box
  - state box
  - clear box
- Verfeinerung und Verfikation
- Komponenten
- statistische Tests
- Nutzungsmodelle (usage models)

3.3 Extreme Programming

- System Metaphor
- Testfall(-getrieben) (statt Spezifikation)

- CRC Cards
- Story Cards
- Refactoring
- Erst Testfälle definieren, dann implementieren

3.4 Fusion

- Fusion
- Klassenmodell
- Use-Case-Modell
- Szenarien
- Operationenmodell
- Systemklassenmodell
- Lifecycelmodell
- Objektinteraktionsmodell
- Referenzierungsmodell
- Klassenschnittstellenmodell
- Vererbungsmodell

4 Nutzen für die Softwareevolution

5 Verwandte Arbeiten

6 Zusammenfassung

7 Ausblick