Software-Wartung – eine Taxonomie

Stefan Opferkuch und Jochen Ludewig
Abteilung Software Engineering, Institut für Softwaretechnologie
Universität Stuttgart
www.iste.uni-stuttgart.de/se

1 Einführung

Das Thema „Software-Wartung“ ist schon so alt wie das Thema „Softwareentwicklung“. Trotzdem fällt es Softwareingenieuren an den Hochschulen und in der Industrie auch heute noch schwer, sich über das Gebiet der Software-Wartung zu verständigen.

Dagegen gibt es kaum Verständigungsschwierigkeiten zwischen Softwareingenieuren über die Entwicklung von Software. Man kann sich sofort darüber abstimmen, in welchem Teilgebiet der jeweilige Gesprächspartner tätig ist. Analyse, Spezifikation, Entwurf, Implementierung, Test, dies sind alles Tätigkeiten, unter denen sich Softwareingenieure, bei allen Differenzen im Detail, etwas vorstellen können.

Und bei der Software-Wartung? Da wird die Verständigung plötzlich schwierig. Da fällt es schwer, überhaupt die Grenzen der Wartung von Software zu benennen. Was ist noch Erstentwicklung, was schon Wartung? In welche Gebiete lässt sich die Software-Wartung aufteilen?

Viele Schwierigkeiten sind auf eine fehlende oder unzureichende Taxonomie für dieses Gebiet zurückzuführen. Im Nachfolgenden soll ein kleiner Beitrag dazu geleistet werden, dass zukünftig nicht immer wieder erneut die grundlegenden Begriffe geklärt werden müssen und dass die Kommunikation über Software-Wartung vereinfacht wird.

2 Abgrenzung des Wartungsbegriffs

Der Wartungsbegriff ist alt und kann darum nicht einfach frei definiert werden. Welche Konnotationen sind üblicherweise mit der Wartung verbunden?

Bei technischen Geräten und Einrichtungen findet die Wartung in der Regel statt, um den **Verschleiß** zu kompensieren. Autos, Flugzeuge und andere Maschinen werden darum regelmäßig gewartet. Diese Bedeutung spielt aber in der Software-Wartung keine Rolle, weil Software immateriell ist und daher keinem Verschleiß unterliegt.

Während der Verschleiß vorhersehbar, die entsprechende Wartung also planbar ist, lassen sich überraschende Defekte und Änderungen der Anforderungen nicht vorhersehen. Die dadurch erforderliche Wartung ist also im Detail nicht planbar; man kann höchstens Vermutungen darüber anstellen, welcher Aufwand für diese Art der Wartung einzuplanen ist. Hier ist zu betonen, dass die Defekte in der Software wohl überraschend auftreten, aber natürlich vorher bereits latent vorhanden sind.

Der von Lehman und Belady [1] postulierte Entropie-Anstieg, der bei Software-Systemen als ein Effekt der Wartung eintritt, hat eine gewisse Ähnlichkeit mit dem Verschleiß. Entsprechend kann man bei einem laufend eingesetzten Softwaresystem abschätzen, wann eine Strukturverbesserung (ein Re-Engineering) notwendig wird. Wir halten es daher für sinnvoll, das Software-Re-Engineering als eine von der Software-Wartung verschiedene Tätigkeit zu behandeln. Es scheint zweckmäßig, einen Oberbegriff einzuführen („**Software-Pflege**“), der Wartung und Re-Engineering einschließt.

Wir halten daher fest: Software-Wartung ist im Einzelnen **nicht vorhersehbar**, mit ihr ist ein **Überraschungsmoment** verbunden. Wir sehen als weiteres Merkmal, dass die Software-Wartung im unmittelbaren Interesse der Benutzer liegt. Daher definieren wir:

Software-Wartung ist jede Arbeit an einem bestehenden Software-System, die nicht von Beginn der Entwicklung an geplant war oder hätte geplant werden können und die unmittelbare Auswirkungen auf den Benutzer der Software hat.

Die Wartung findet in der Regel nach Abschluss der Entwicklung statt, sie kann aber auch bereits während der Entwicklung nötig werden.

Umgekehrt endet die **Entwicklung** nicht automatisch mit der Inbetriebnahme der Software. Wenn ein System in einer Folge von Ausbauschritten realisiert wird, handelt es sich nicht um Wartung, es sei denn, dass es beim Ausbau zu überraschenden Änderungen der Planung kommt.

Auch die **Komposition** bestehender Komponenten oder Systeme zu einem neuen System ist keine Wartung, soweit nicht die einzelnen Bestandteile des neuen Systems angepasst oder erweitert werden müssen.

3 Arten und Aspekte der Wartung

Wartungsaktivitäten kann man nach verschiedenen Kriterien klassifizieren. Als Kriterien bieten sich an:

- der **Anlass** der Wartung
- der **Ausgangszustand** der Wartung
- die **Art der Anforderungen**, die die Wartung erforderlich machen
- der **Prozess**, nach dem die Wartung abläuft

Abb. 1 zeigt eine Übersicht der Taxonomie für die Software-Wartung, sowie deren Verhältnis zu Software-Pflege und Software-Re-Engineering.

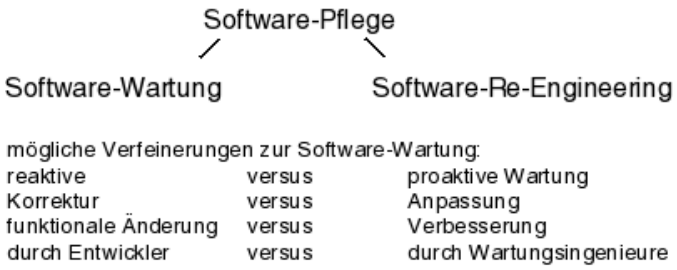

Abbildung 1: Taxonomie der Software-Wartung

3.1 Der Anlass der Wartung

Zu Beginn der Wartung einer Software sind vor allem Korrekturen erforderlich. Diese erfolgen in der Regel **reaktiv**, d.h. nach Auftreten von Fehlern oder Problemen.

Wenn man die Wartung über den gesamten Lebenszyklus einer Software betrachtet, so wird deutlich, dass der überwiegende Teil der Wartungsaktivitäten aus Anpassungen oder Erweiterungen besteht, also **proaktiv** bearbeitet wird.

Teilweise werden auch Korrekturen *proaktiv* durchgeführt, wenn Fehler oder Probleme vorhergesehen oder vermutet werden.

Beim Jahr-2000-Problem wurde ganz überwiegend versucht, die Probleme proaktiv zu beheben. Ebenso geht man meist proaktiv ans Werk, wenn man bei einer Änderung der Umgebung bestimmte Inkompatibilitäten vorhersieht.

In der Literatur wird an Stelle von „proaktiv“ das Wort „präventiv“ verwendet. Wir folgen dem nicht, weil die Prävention durch die Medizin eine ganz andere Konnotation hat. Prävention soll Defekte (Krankheit) verhindern, während sich proaktive Wartung mit Defekten befasst, bevor sie Schaden angerichtet haben.

3.2 Der Ausgangszustand der Wartung

Wird in der Software ein Fehler auffällig, so besteht ein Widerspruch zwischen Anforderungen und Implementierung, das System ist **inkonsistent**. Hier ist eine **Korrektur** fällig; die Spezifikation bleibt unverändert, die Implementierung wird verändert, um Konsistenz herzustellen.

Ändern sich die Anforderungen (z.B. durch eine Portierung auf ein anderes Betriebssystem), so ist der Ausgangszustand (prinzipiell) **konsistent**. Wir sprechen in diesem Fall von einer **Anpassung**: Die Veränderung der Spezifikation wird in der Implementierung nachvollzogen, das System wird von einem konsistenten Zustand in einen *anderen* konsistenten Zustand überführt.

3.3 Die Art der Anforderungen, die die Wartung nötig machen

Eine Wartung ist dann und nur dann notwendig, wenn eine **Diskrepanz zwischen Anforderungen und Realisierung** aufgetreten oder absehbar ist. Handelt es sich um funktionale Anforderungen, so sprechen wir von einer (funktionalen) **Korrektur** oder einer (funktionalen) **Erweiterung**. Handelt es sich um nicht-funktionale Anforderungen, die die Gebrauchsqualität betreffen, so ist das Ziel der Wartung eine **Verbesserung** des Systems (für den Benutzer oder Kunden).

Offensichtlich kann es sich bei der Verbesserung auch um eine Korrektur handeln, beispielsweise dann, wenn die Antwortzeiten des Systems oder seine Wartbarkeit nicht den Anforderungen entsprechen.

3.4 Der Prozess, nach dem die Wartung abläuft

Wartung kann ganz unterschiedlich durchgeführt werden: In vielen Fällen übernehmen die Entwickler selbst, soweit sie noch verfügbar sind, die Wartung neben anderen (Entwicklungs-)Arbeiten. Dies bezeichnen wir als **Wartung durch die Entwickler**. Eine weitere Möglichkeit ist, spezielle Personen mit der Wartung zu betrauen. Wir sprechen dann von einer **Wartung durch Wartungsingenieure**.

Software-Re-Engineering wurde zu Beginn als eigene, von der Wartung verschiedene Aktivität definiert. Wir bezeichnen als Re-Engineering Änderungen an einer Software, die ausschließlich einer Verbesserung hinsichtlich von Wartungsqualitäten dienen. Wartungsqualitäten sind Eigenschaften der Software, die direkt keinen Einfluss auf die Qualität aus Sicht eines Anwenders haben, sondern nur die Entwickler und Wartungsingenieure betreffen.

In der Praxis werden Re-Engineering und Wartung oft vermischt; das ist in der Regel nicht sinnvoll, weil das reine Re-Engineering durch Regressionstests sehr gut überwacht werden kann; sobald funktionale Änderungen dazukommen, ist die klare Regel „Alles muss nachher funktionieren wie vorher.“ aufgehoben.

Literatur

[1] LEHMAN, M.M. und L.A. BELADY: *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.