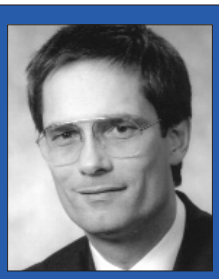
Fehlerfreie Software durch wissensorientierte Softwareentwicklung



Martin Rösch

Softwarefehler sind keine Geißel der Menschheit, sondern sicher vermeidbar. Hierfür ist ein bewusster Umgang mit Wissen in der Softwareentwicklung erforderlich. Der Artikel gibt einen ersten Überblick über die wissensorientierte Softwareentwicklung und beschreibt erste praktische Erfahrungen aus Projekten.

Fröhlich fährt der Informatik-Manager mit seinem neuen Auto zum Termin. Die Sonne scheint, das Auto macht einfach nur Spaß und von der CD kommt seine Lieblingsmusik. Die freundliche Frauenstimme im Navigationssystem sagt „jetzt rechts fahren", während die Abbiegespur zu seiner Rechten schon längst außer Reichweite ist. Er denkt etwas sehr Unfreundliches über die Softwareentwickler des Navigationssystems und schaut auf seine Uhr, die ihm mit bewundernswerter Präzision die Sicherheit vermittelt, dass er nicht mehr pünktlich ankommen wird. Doch noch während er auf die nächste Ausfahrt zum Wenden wartet, legt sich sein Groll, denn er denkt an die eigene Software, für die er selbst die Verantwortung trägt. Und an seine Kunden, die sich auch schon mal ärgern. Und an die eigenen Entwickler. Er weiß, dass sie gut sind, denn er kennt auch die anderen. Warum bloß ist Software nicht so gut wie sein Auto? Und wie seine Uhr?

Nach unseren Erfahrungen ist die Antwort sehr einfach: In Softwareprojekten geht viel zu viel Wissen verloren oder es kommt dort gar nicht erst an. Softwarefehler sind Symptome von unzureichender Sorgfalt beim Umgang mit Wissen.

Dipl.-Inf. Martin Rösch (E-Mail: info@roesch.com) ist Geschäftsführer der „Rösch Consulting Gesellschaft für innovative Softwareentwicklung mbH" in Kaarst, die sich auf die Entwicklung von „Software-Robotern" und die wissensorientierte Softwareentwicklung spezialisiert hat.

Unternehmen sind organisiertes Wissen

Wissen ist der Rohstoff der Informationsgesellschaft. Wissen organisiert die klassischen drei Produktionsfaktoren der Betriebswirtschaftslehre (vgl. [Gut90]) *Rohstoffe, Arbeit* und *Kapital* und erlaubt uns den bewussten Umgang mit ihnen.

Software ist automatisiertes Wissen

Software automatisiert Wissen in Form von Programmen. Über die Drähte unserer Computernetzwerke bringen Softwareprogramme das Wissen der Unternehmen an jeden Arbeitsplatz, im LAN und im Internet. Doch welches Wissen wird uns da gebracht? Wenn wir ehrlich sind: meistens wissen wir das nicht so ganz genau. Wo gibt es schon Beschreibungen von Softwaresystemen, die uns präzise darüber informieren, was eine Software macht oder – genauer – welches Wissen sie enthält. Und was sie *nicht* enthält und wer dafür die Verantwortung trägt. Wenn heute Software nicht macht, was sie soll, oder macht, was sie nicht soll, dann wissen wir nur selten, ob dieses Zuviel oder Zuwenig an automatisiertem Wissen Absicht oder ein Fehler ist.

Das Wissen in Software von heute: dezentral und „verantwortungslos"

Das klingt unfair, denn jeder von uns gibt sein Bestes, und kein Softwareentwickler handelt bewusst verantwortungslos im Sinn einer schädlichen Absicht. Doch wenn man die Emotionen einmal beiseite lässt und den Prozess der bisherigen Softwareentwicklung nüchtern betrachtet, kann man erkennen, dass wir bisher nur *das* Wissen sicher besitzen, das aus dem Prozess der Softwareentwicklung *herauskommt*, in Form von Programmcode, verantwortet von seinem Ersteller. Das Wissen, das in den Prozess *hineinfließt*, kommt dagegen *dezentral* von vielen Seiten, wird nur selten präzise und vollständig erfasst und noch seltener ist es mit einer klaren Verantwortung versehen. So gesehen ist es *„verantwortungslos"*.

Programmcode = Ist-Zustand des Wissens

Die einzige präzise, vollständige und verantwortete Beschreibung des Wissens in Software ist heute noch ihr Programmcode. Er beschreibt den *Ist-Zustand* des Wissens in einer Software. Im Code ihrer Programme steht alles, was eine Software wirklich macht. Nur kann man es dort nicht mehr klar erkennen. „The essence and the accidents" ([Bro88]) sind kaum zu unterscheiden: Welcher Code ist wirklich wichtig („the essence")? Und was wurde später noch dazugestrickt („the

accidents“)? Aus diesem Grund haben Entwickler von Anfang an versucht, anhand von Kommentaren das Wissen im Code ihrer Software zu zeigen. Dann kam die „strukturierte Programmierung“ mit dem Ziel, das Ablaufwissen noch klarer zu zeigen: Schleifen und IF-Anweisungen ersetzten die GO TOs. Noch expliziter wurde das Wissen im Code durch Objekte erkennbar. Die Klassen der Objektorientierung beschreiben das Wissen ihrer Objekte: Wissen über die enthaltenen Informationen und Wissen über die möglichen Verarbeitungen. Doch egal, wie gut man den Code strukturiert, mit oder ohne Objekte: In ihm ist das Wissen begraben, man bekommt es nicht sicher wieder heraus. Man sieht weder, was es war, noch, wer es verantwortet – und damit auch nicht, ob es verantwortungs-los ist.

Soll-Zustand des Wissens in Software

Deshalb versuchte bisher noch jede Generation von Programmierern, das Wissen ihrer Programme für Auftraggeber sichtbar zu machen und die Auftraggeber auf diese Weise mit in die Verantwortung zu ziehen: zuerst mit Ablauf-Diagrammen, die noch die Sprunganweisungen zeigten, dann mit Nassi-Shneiderman-Diagrammen für strukturierten Code und schließlich mit der „Unified Modeling Language“ (UML) für objektorientierte Systeme.

Dabei erzeugte jede Generation Tonnen von Schrankware, die mit jeder codierten Zeile wertloser wurde. Denn welcher Programmierer kümmert sich noch um das Wissen aus den frühen Projektphasen, wenn erst einmal der hektische und kreative Prozess des detaillierten Entwurfs und der Codierung begonnen hat? Immer genauere Abstimmungen mit den Anwendern brachten ständig neue Erkenntnisse, deren sofortige Umsetzung in Code allen Seiten Glücksgefühle bescherte. Doch wehe, wenn etwas schief lief: Dann begann das Fingerzeigen. Schuld waren immer die anderen. Diese Grabenkriege haben in allen Unternehmen, besonders in den großen, eine lange Tradition – und eine einfache Ursache: Ständig geht Wissen verloren.

Der bewusste Umgang mit Wissen ist der Softwareentwicklung noch weitgehend fremd. Vereinzelt und in besonders wichtigen Projekten wird das Wissen bewahrt*, doch nur auf dem Papier oder in textbasierten Recherchesystemen mit Hyperlinks. Abweichungen zwischen dem Wissen in der Software und dem Wissen in ihren zugehörigen Texten sind dabei nur mit viel Aufwand erkennbar. Und Widersprüche zwischen dem Wissen auf den Seiten 1037 und 2037 merkt man erst, wenn die Software fertig ist und nicht macht, was sie soll.

Die Größe und die Komplexität der heutigen Informationssysteme machen es für Menschen immer schwerer, den Überblick zu behalten, sowohl über die Software selber als auch über das Wissen, das sie automatisieren soll. Die heutige Art der Softwareentwicklung bringt eine ständige und systematische Überforderung aller beteiligten Menschen. Das macht die Softwareentwicklung langsam, teuer und anfällig für Fehler.

Wissen wird in der heute noch üblichen, technikorientierten Softwareentwicklung nicht bewusst behandelt und nicht zuverlässig gesichert. Auf dem Weg des Wissens – von den Köpfen der Wissensträger bis hinein in den Programmcode der fertigen Software – geht noch zu viel Wissen unkontrolliert verloren. Wenn wir die Softwareentwicklung verbessern wollen, muss das Wissen präzise, vollständig und mit klar zugeordneten Verantwortlichkeiten beschrieben und ins Zentrum der Softwareentwicklung gestellt werden, denn Software ist automatisiertes Wissen.

Hierfür brauchen wir den bewussten Umgang mit Wissen in der Softwareentwicklung, sowohl bei der Anlieferung von Wissen für Softwareprojekte als auch bei seiner weiteren Verarbeitung und Dokumentation. Die Softwareentwicklung muss sich am Wissen orientieren.

Wissen als Motor des Fortschritts

Zum Glück ist die Softwareentwicklung nicht das erste Arbeitsgebiet, in dem Menschen vor der Aufgabe stehen, Wissen zu sichern. Seit Beginn der Industrialisierung haben Menschen in vielen Bereichen ähnliche Herausforderungen bewältigt. Das bekannteste Beispiel ist vielleicht die Automobilindustrie, deren Entwicklungsschritte sehr ausführlich dokumentiert sind (vgl. [Wom94]).

Für Automobilhersteller ist der bewusste Umgang mit Wissen seit mehr als hundert Jahren selbstverständlich und blanke Notwendigkeit für das Überleben im Markt. Der größte, bekannteste und damals beste Autohersteller der Jahrhundertwende, Panhard & Levassor (P&L) aus Paris (der unter anderem auch Motoren von Carl Benz bezog) war wenige Jahre später nicht mehr konkurrenzfähig. P&L hatte nicht begriffen, dass Henry Ford mit seinem Insistieren auf maßgenauem Arbeiten (vgl. [Wom94]) und wesentlich verbesserter Präzision nicht eine persönliche Marotte pflegte, sondern ein knallhartes unternehmerisches Kalkül verfolgte: Durch die verbesserte Präzision sparte er alle Anpassungsarbeiten ein, die bis dahin bei Montagevorgängen in der Automobilindustrie noch üblich waren. Hierdurch bekam er als erster kalkulierbare Montagezeiten, die ihm in der Folge den Einsatz des Fließbands erlaubten und die Herstellungskosten noch weiter herabsetzten. Zusätzlich reduzierte er seine Abhängigkeit von der Knappheit ausgebildeter Facharbeiter, denn bei ihm konnte fast jeder nach kurzer Einarbeitungszeit beginnen. So konnte Henry Ford die moderne Massenproduktion aus der Taufe heben, denn Personalknappheit kannte er nicht.

Was war Henry Fords Geheimnis? Aus heutiger Sicht ist es klar zu erkennen: Er ging bewusster mit dem Wissen des Unternehmens um und er sicherte das Wissen im gesamten Produktionsprozess. Um dieses Ziel zu erreichen, gab er das Wissen nicht nur – wie alle anderen Autohersteller – in Form von Plänen an seine internen und externen Zulieferer weiter, sondern er sicherte es auch, indem er ihnen zusätzlich zu den Plänen Messwerkzeuge an die Hand gab. Diese Messwerkzeuge waren Gestalt gewordenes Wissen, das er beliebig vervielfältigen konnte. Sie wurden zu Wissen, das sofort anwendbar war, auch ohne erfahrene Spezialisten der Metallbearbeitung. Und während andere Hersteller noch unter der Knappheit an ausgebildeten

* I. Peterson berichtet in dem Buch „Fatal Defect-Chasing Killer Computer Bugs“ ([Pet95]) über David Parnas und 30 Ordner für die Softwareanforderungen eines Kernkraftwerks; Peter Coad und Edward Yourdon berichten in [Coa91] von gigantomanischen Dokumentationsanforderungen in Defense-Projekten.

Facharbeitern litten, konnte Ford schon in Massen produzieren, denn das Wissen steckte im Prozess. Das war für alle Beteiligten ein Gewinn: Henry Ford konnte *mehr Autos* bauen, die Kunden bekamen *bessere Preise* und seine ungelernten Arbeiter bekamen den *doppelten Lohn* ihrer gelernten Kollegen bei der Konkurrenz. Das Geheimnis von Henry Ford war der bewusste Umgang mit Wissen.

Wird die Wissensorientierung die Softwareentwicklung in einen „Anlern-Job" verwandeln? Mit Sicherheit nicht, denn heute beschäftigt die Automobilindustrie mehr Ingenieure, als es damals Arbeiter gab. Die Autos von heute sind mit über 40.000 Teilen so komplex, dass kein einzelner Mensch mehr das ganze Wissen über ein Modell besitzen kann. Beim Ford Model T von 1908 war das vielleicht noch möglich. Doch Unternehmen wie BMW oder Mercedes benötigen heute viele hundert Ingenieure, um das gesamte Wissen eines einzigen Modells zu bewahren. Ingenieure machen heute die spannenden Arbeiten, während die Fertigung fast vollständig automatisiert ist. In den Montagehallen stehen Industrie-Roboter, die immer größere Teile des Fertigungswissens besitzen und es automatisch, schnell und zuverlässig anwenden.

Wissensorientierte Softwareentwicklung

Die wissensorientierte Softwareentwicklung beruht auf derselben grundlegenden Zweiteilung wie die heutige Fertigung von Automobilen. Auf der einen Seite stehen *Wissens-Ingenieure*, die das gesamte in Software einfließende Wissen sammeln, festhalten und in UML-Modellen integrieren. Auf der anderen Seite gibt es *Software-Roboter*, die das Wissen, nachdem es verifiziert ist, automatisch, schnell und zuverlässig in fertige Software verwandeln. Probleme bringt dies nur für Programmierer, die Spaß an langweiligen und immer wiederkehrenden Programmieraufgaben haben. Diese Arbeiten werden schon bald von Software-Robotern erledigt, 100.000-fach schneller.

Die wissensorientierte Softwareentwicklung erfasst das Wissen an seiner Quelle: bei den Wissensträgern des Unternehmens. Dort wird es verständlich, verantwortbar und präzise beschrieben, mit Anforderungen und Akzeptanzkriterien. So wird sichergestellt, dass man es auch bei den folgenden Arbeitsschritten – und bei der späteren Wartung – nicht aus Versehen verliert.

Ähnlich wie die Wissensbereiche bei einem Automobil (z. B. Lichtmaschine, Motor und Zündung) werden auch die Wissensbereiche von Softwaresystemen bei der wissensorientierten Softwareentwicklung organisatorisch gegliedert (z. B. in Teilprojekte) und mit Wissensträgern bestückt. Jedes Teilprojekt liefert dann sein Wissen in Form einer Komponente ab, die von anderen Teilprojekten ohne Kenntnis ihres inneren Aufbaus verwendet werden kann. Nur das nach außen sichtbare Verhalten muss den Verwendern einer Komponente noch bekannt sein. Diese Vorgehensweise klingt wie ein alter Hut, doch der Unterschied liegt – ähnlich wie bei Henry Ford – in der Präzision der Beschreibung und in der einfachen sowie automatischen Prüfbarkeit aller Komponenten und ihrer Schnittstellen untereinander. Jede Komponente kapselt das Wissen ihrer Wissensträger und kann jederzeit, isoliert und in wenigen Sekunden, gegen alle ihre eigenen Anforderungen und gegen alle an sie gerichteten Anforderungen ihrer Nachbarn geprüft werden. So wird es beispielsweise möglich, in einem aktuellen Projekt über 100 Entwickler in mehr als einem Dutzend Teilprojekten parallel und ohne nennenswerten Abstimmungsaufwand mit ihren Nachbarn arbeiten zu lassen. Das bringt Geschwindigkeit. Die Abstimmung mit den Nachbarn setzt erst später ein, wenn jedes Teilprojekt seine eigene Arbeit komplett, in sich rund und mit Annahmen und Wünschen bezüglich seiner Nachbarn beschrieben hat.

Dieses Vorgehen erweckt vordergründig den Anschein, dass die anfangs unterbleibende Abstimmung zwischen den Teilprojekten später zu überflüssigen Aufwänden für nachträgliche Anpassungen führt. Doch dieser vermeintliche Nachteil wird mehr als aufgewogen, weil die Abstimmung mit viel größerer Sicherheit und damit auch mit viel höherer Geschwindigkeit und Sicherheit erfolgt.

Auf diese Weise kann das gesamte in die Softwareentwicklung einfließende Wissen sicher beherrscht werden und die beteiligten Menschen wissen, dass es nicht mehr verloren geht.

Erfahrungen der ersten Projekte

Die wissensorientierte Softwareentwicklung ist bereits in mehreren Projekten eingesetzt worden. Das größte Projekt koordiniert die Arbeit von über 100 Entwicklern in mehr als einem Dutzend Teilprojekten und setzt einen „Wissensberg" von mehreren Tausend Anforderungen in Software um. Eines der Teilprojekte wurde von der Firma Rösch Consulting als Festpreisprojekt mit Qualitätsgarantie übernommen und Anfang Oktober 2000 an den Kunden übergeben. Die Garantie sah für jeden vom Kunden entdeckten Fehler einen Abzug in Höhe von 5% der Auftragssumme vor, d. h. bei 20 Fehlern bräuchte der Kunde nicht mehr zu zahlen. Ähnlich wurde der Termin garantiert: 10 % Abzug für jede begonnene Woche nach dem vereinbarten Termin.

Die bisher vorliegenden Erfahrungen (keine Fehler gefunden, Termin eingehalten, zwei weitere Projekte angelaufen) legen die Vermutung nahe, dass die wissensorientierte Softwareentwicklung die Erstellung von nachweisbar fehlerfreier Software erlaubt. Es ist absehbar, dass Ersteller von Individual- und Standardsoftware ihren Kunden schon bald ähnliche Garantien werden geben können und müssen, wenn 100%ige Qualität zum Wettbewerbsfaktor wird.

Nachdem die bisher üblichen Problemquellen von Softwareprojekten sicher vermieden werden konnten, war es für uns interessant, die neuen Herausforderungen kennen zu lernen, die bei wissensorientierten Softwareprojekten in den Vordergrund treten.

Die wichtigste Beobachtung war, dass sicheres Wissen sehr schnell beschrieben werden kann, während unsicheres oder fehlendes Wissen auch nach sehr langen Bearbeitungszeiten nicht stabil wird. Deshalb gehen wir im Umkehrschluss jetzt davon aus, dass ein Wissensthema, das von seinem Wissensträger nicht innerhalb eines Tages abschließend und vollständig beschrieben werden kann, dem unsicheren Wissen zugerechnet werden muss und aus dem Projektumfang herausgenommen werden sollte. Diese Art der Identifikation von *Wissenslöchern* erschien uns zunächst gewagt, doch sie hat sich in den Folgeprojekten bereits bewährt. Wissenslöcher melden sich eben nicht von selber. Man erkennt sie nur

indirekt und daran, dass der Fortschritt zäh wird. Wer gibt schon gerne zu, dass er bei einem Thema noch Lücken hat, für das ihn Kollegen und Chefs als Experte betrachten?

Ausblick

Große Softwaresysteme haben heute mehrere tausend Anforderungen und ein Vielfaches davon an Akzeptanzkriterien, die zusammen das durch die Software automatisierte Wissen repräsentieren. Das mag zunächst recht viel erscheinen, doch im Vergleich zu den 40.000 Teilen eines Autos, von denen jedes vielleicht ein Dutzend Anforderungen erfüllt, nehmen sich auch die größten heute bekannten Softwaresysteme noch eher bescheiden aus. Unter der Annahme, dass das oben angenommene Dutzend stimmt, repräsentiert ein Automobil von heute ein Wissen im Umfang von ca. 500.000 Anforderungen, mit ca. 1,5 Mio. Prüfbedingungen. Uns ist noch kein Softwaresystem bekannt, das diese Menge an Wissen enthält. Wir kennen aber viele, die mehr Fehler haben als unsere Autos.

Zusammenfassung

Weitere Fortschritte in der Softwareentwicklung werden nur dann möglich sein, wenn die Softwareentwicklung ihren Horizont erweitert und das gesamte Wissen betrachtet, das sich letztlich in der fertigen Software niederschlagen soll. Dieses Wissen muss an der Quelle – dort, wo es noch verantwortet werden kann – präzise und vollständig erfasst werden. Es muss im gesamten Prozess der Softwareentwicklung sicher bearbeitet werden, mit einem entsprechenden Nachweis an jeder Station. Diese wissensorientierte Softwareentwicklung wird die Erstellung wesentlich größerer Softwaresysteme erlauben, als wir heute beherrschen, und für einen flächendeckenden E-Commerce wird sie voraussichtlich zur zwingenden Voraussetzung werden. Aber schon heute bringt die wissensorientierte Softwareentwicklung Vorteile, denn sie ermöglicht die Erstellung von fehlerfreier Software – mit Garantie.

Literatur

[**Bro88**] F.R. Jr. Brooks, No Silver Bullet, in: IEEE Software, April 1988

[**Coa91**] P. Coad, E. Yourdon, Object-oriented Analysis, Prentice Hall, 1991

[**Gut90**] E. Gutenberg, Einführung in die Betriebswirtschaftslehre, Gabler, 1990

[**Pet95**] I. Peterson, Fatal Defect – Chasing Killer Computer Bugs, Random House 1995

[**Wom94**] J.P. Womack, D.T. Jones, D. Roos, Die Zweite Revolution in der Automobilindustrie – Konsequenzen aus der weltweiten Studie des Massachussetts Institute of Technology, Campus, 1994