Software Craftsmanship - Softwareentwicklung ist ein Handwerk

Im Mittelalter war die Produktion von Gütern noch in Manufakturen organisiert. Meistens mit sehr wenigen Mitarbeitern. Geleitet wurde die Manufaktur üblicherweise von einem Meister, der in langjähriger Ausbildung alles über sein Handwerk gelernt hat. Die Qualität der von der Manufaktur gefertigten Waren hing damit auch direkt mit den Qualitäten und Fähigkeiten des Meisters zusammen.

Dann kam die Industrialisierung. Die komplexen Produktionsprozesse der Manufakturen wurden in ganz viele Einzelschritte aufgeteilt. Am besten so klein, daß sie auch von Menschen ohne grosse Ausbildung durchgeführt werden können. Meistens waren es tatsächlich nur wenige Handgriffe, die man in relativ kurzer Zeit lernen konnte.

Am Ende der vielen Einzelschritte stand dann wieder das fertig produzierte Produkt. Gegenüber den Manufakturen gab es gleich eine Reihe von Vorteilen:

- Die Qualität der Produkte war immer relativ gleich
- Die Fabrikbesitzer konnten viel Geld sparen, da ungelernte Arbeiter natürlich weniger verdienen als Handwerksmeister

Und das Wichtigste: Die Produktivität war ungleich höher. Der Ausstoss einer nach diesem neuen Muster organisierten Fabrik war ungleich höher als der einer traditionellen Manufaktur. Dazu mit gleichbleibender Qualität und zu wesentlich niedrigeren Stückkosten.

Man muss keinen Abschluss in BWL haben um zu erkennen, warum dieses Vorgehen die Industrie revolutioniert hat.

Lange Zeit war alles in Ordnung. Dann kam die Softwareentwicklung. Und 100 Jahre Industrialisierung hatten natürlich ihre Spuren hinterlassen: Getreu dem oben beschriebenen Muster wurde nun auch hier probiert, den komplexen Prozess der Softwareentwicklung in viele kleine Einzelschritte aufzuteilen.

Dies kann man z.B. schön im Wasserfallmodell sehen: Erst werden die Anforderungen gestellt, danach Grob- und Feinspezifikation erstellt, und im Anschluss die Software implementiert, getestet und abgenommen. Sehr viele einzelne kleine Schritte, deren Ziel es ist, Softwareentwicklung zu einem wiederholbaren Prozess zu machen – ähnlich wie z.B. die Produktion von Möbeln oder Autos.

Es gibt nur ein Problem dabei: Es funktioniert nicht!

In den letzten Jahrzehnten hat sich sehr deutlich gezeigt, daß sich dieses Vorgehen für Softwareentwicklung einfach nicht eignet. Nachlesen kann man das z.B. im Chaos Report der Standish Group, wonach nur ca. 16% aller IT-Projekte erfolgreich abgeschlossen werden (siehe [1]).
Softwareentwicklung ist anscheinend eben kein trivialer, mess- und wiederholbarer Ingenieursprozess. Doch was ist es dann? Die Antwort ist so einfach wie überraschend:

Softwareentwicklung ist ein Handwerk!

So werden in der Individualsoftwareentwicklung ganz spezifische, auf den jeweiligen Kunden und Anwendungsfall maßgeschneiderte Produkte hergestellt – ähnlich wie es z.B. auch im Handwerk der Fall ist.

Und wie im Handwerk, so gehört auch hier wesentlich mehr dazu als das reine Erzeugen des Werkstücks (also das Schreiben von Quellcode). Und natürlich verfügt auch ein "Softwerker" über eine Handwerksehre:

Dies bedeutet, die Software zu erstellen, die der Kunde wirklich braucht, und dabei Wert sowohl auf innere, als auch auf äußere Qualität zu legen.

Diese Qualität zu erreichen ist jedoch alles andere als einfach, und braucht viel Erfahrung. Diese erreicht man aber nur durch jahrelanges Training und Berufserfahrung. Und selbst dann ist die Erlangung einer „Meisterschaft“ keineswegs sicher – so mancher bleibt sein ganzes Berufsleben lang auf einem eher durchschnittlichen Niveau.

Um ein Softwareprojekt auf hohem handwerklichen Niveau durchzuführen sind für mich aktuell zumindest die folgenden Fähigkeiten notwendig:

- Kundenberatung: Jeder Kunde hat Anrecht auf eine spezifische, genau auf die jeweiligen Bedürfnisse zugeschnittenen Beratung. Ein professioneller Softwareentwickler soll verantwortungsbewusst handeln und sich dabei an Werten wie Wirtschaftlichkeit, Qualität und Machbarkeit orientieren.
- Stakeholder-Management: Es gilt, alle relevanten Stakeholder zu identifizieren und auf die jeweils angebrachte Art und Weise in das Projekt einzubinden. Je nach Art der Betroffenheit kann dies verschiedene Ausprägungen annehmen - von aktiver Mitarbeit im Projektteam, über aktive und passive Informationspolitik, bis hin zu abwartenden Maßnahmen oder gar aktiver Abgrenzung. Diese initiale Analyse muss über die ganze Projektdauer hinweg aktiv nachverfolgt werden, da sich Einstellung und Betroffenheit von Stakeholdern während des Projekts jederzeit ändern können.
- Anforderungsmanagement: Ein sehr komplexes und umfangreiches Thema, dem (zu Recht) eine sehr hohe Bedeutung im Projektmanagement zukommt. Hier kann bereits der Grundstein für einen späteren Projekterfolg, oder natürlich auch ein Scheitern gelegt werden. Bevor eine Anforderung umgesetzt wird sollte man mindestens die folgenden Punkte klären:
    - Sind die Anforderungen klar und eindeutig beschrieben?
    - Gibt es Interpretationsspielräume?
    - Sind eindeutige Abnahmekriterien formuliert?
    - Haben Entwickler und Kunde ein gemeinsames Verständnis des Ergebnisses?
    - Sind die Anforderungen sowohl einzeln, als auch alle zusammen, in einem vernünftigen finanziellen Rahmen umsetzbar?
- Systemarchitektur: Der „Softwerker“ kennt die relevanten aktuellen Technologien und berät den Kunden dahingehend kompetent und individuell. Am Ende steht eine Systemarchitektur, die genau dem gewünschten Einsatzzweck entspricht, und weder zu klein noch zu groß dimensioniert ist. Zudem berücksichtigt sie den kompletten Lebenszyklus – von der Erstellung über die Inbetriebnahme und den Produktivbetrieb bis hin zu Aus- und Umbau, sowie am Ende die Einstellung des Systems bzw. Ablösung durch ein Nachfolge- oder Alternativprodukt.
- Entwicklungsprozess: Da Softwareentwicklung immer mit nachträglichen Änderungen verbunden ist (auch wenn sich beide Seiten noch so viel Mühe geben, das zu vermeiden), sieht der „Softwerker“ einen Prozess vor, der Änderungen an den Anforderungen auch spät in der Umsetzungsphase noch ermöglicht. Die Möglichkeit, auf veränderte Anforderungen auch kurzfristig noch reagieren zu können gibt dem Kunden einen Vorteil gegenüber seinen Mitbewerbern. In vielen Bereichen, z.B. öffentlichen Institutionen sind aber auch heute noch bestimmte Entwicklungsprozesse üblich oder gar vorgeschrieben (etwa das V-Modell). Als Craftsman sollte man alle relevanten Entwicklungsprozesse kennen.

Bei allen bis jetzt beschriebenen Tätigkeiten wurde noch keine einzige Zeit Quellcode erzeugt – mithin das, was allgemein unter „Softwareentwicklung“ verstanden wird. Auch bei der Umsetzung der Anforderungen in Quellcode gibt es Techniken und Vorgehensweisen, die man 2014 von einem handwerklich gut gemachten Softwareprojekt erwarten darf:

- Versionsverwaltung: Sämtlicher Quellcode wird versioniert. In der Vergangenheit waren hier zentrale Systeme üblich (etwa CVS oder SVN). In jüngerer Zeit sind auch dezentrale

Systeme auf dem Vormarsch (z.B. Git oder Mercurial). Ein Software Craftsman weiss um die Vor- und Nachteile beider Ansätze, und kann mit beiden gleichermaßen professionell umgehen

- Kontinuierliche Integration: Eine Technik, die sich in den vergangenen Jahren immer mehr durchgesetzt hat, und in vielen Projekten mittlerweile Standard wird. Dabei wird der Quellcode in regelmäßigen (sehr kurzen) Abständen neu gebaut, wenn möglich sogar bei jeder Änderung des Quellcodes im Versionsverwaltungssystem. In Folge dessen werden auch alle Tests ausgeführt, sowie die Dokumentation erzeugt. Am Ende jedes „Builds“ steht eine Gesamtnote, die den Entwicklern angezeigt wird. Ein Build sollte üblicherweise nur wenige Minuten dauern. Ist dies nicht der Fall, so kann dies mittels technischer (Gegen-) Maßnahmen wieder erreicht werden (z.B. Aufteilen in mehrere kleinere Pakete).
- Kollektives Code-Eigentum: Jede Zeile Code darf von jedem Entwickler geändert werden. Es gibt kein Eigentum einzelner Personen an bestimmten Teilen des Quellcodes
- Testgetriebene Entwicklung: Es soll keine Zeile Code geschrieben werden, zu der es nicht auch einen Test gibt. Idealerweise werden die Tests sogar vor dem eigentlichen Quellcode geschrieben („Test First Development“). Die „Testabdeckung“, d.h. der Prozentsatz von Quellcode, der von einem Test abgedeckt wird, soll wenn möglich 100% erreichen. Es existieren verschiedene Arten von Tests. U.a. Unit Tests, Performance Tests, Lasttests, Smoke Tests usw.. In jüngster Zeit erfreut sich auch der Ansatz des „Behaviour Driven Development“ (BDD) steigender Beliebtheit. Dabei wird der Quellcode nicht nur auf richtiges „Rechnen“, wie bei Unit Tests, geprüft, sondern zusätzlich auch auf richtiges Verhalten im Bezug zur Anforderung. Üblicherweise ist es möglich, Testfälle in strukturierter natürlicher Sprache zu formulieren, die sowohl menschen- als auch maschinenlesbar sind.
- Werkzeuge: Zu jeder der verbreiteten Programmiersprachen existieren heute eine Vielzahl an Werkzeugen. Eines der wichtigsten Werkzeuge jedes Entwicklers ist die „IDE“ (Integrated Development Environment). Auch hier existiert eine ganze Auswahl an Alternativen, jede mit ihren Vor- und Nachteilen, sowie eigenem Aussehen und Bedienkonzept. Als Craftsman kennt man alle für seinen Bereich wichtigen Werkzeuge und bleibt auf dem laufenden Stand ihrer Entwicklung. Gerade in den Mainstream-Programmiersprachen konnte man in den letzten Jahren eine hohe Dynamik im Bereich der Werkzeuge und ihrer Möglichkeiten beobachten.
- Programmiersprachen: Jede der verbreiteten Programmiersprachen hat eigene Konzepte, und damit verbunden natürlich auch ganz eigene Stärken und Schwächen. Zudem folgen sie teils ganz unterschiedlichen Ansätzen (z.B. objektorientierte vs. funktionale Sprachen).

Lebenslanges Lernen

Diese Ansprüche sind natürlich recht schnell formuliert. Sie zu erreichen dürfte aber sehr schwer sein. Neben dem notwendigen theoretischen Wissen gehört dazu hauptsächlich eines:
Üben, Üben und Üben – ein Leben lang!

Das kontinuierliche Üben ist eines der zentralen Bestandteile des Software Craftsmanship. Hierfür haben sich in den letzten Jahren verschiedene Methoden etabliert. Dem Verständnis des Handwerks folgend bestehen diese nicht nur aus theoretischem Wissen, sondern auch aus praktischen Übungen und dem direkten Austausch mit anderen Craftsmen.

Die wichtigsten Methoden möchte ich in den folgenden Absätzen kurz vorstellen:

Community of Professionals

Wie bereits beschrieben ist es eine zentrale Idee des Software Craftsmanship, den direkten Kontakt zwischen den Craftsmen zu initiieren und zu fördern. Hierzu ist mittlerweile eine weltweite Bewegung entstanden, mit Ablegern in vielen Ländern der Welt.

Auch in Deutschland existiert mit der „Softwerkskammer“ (siehe [5]) ein solcher Ableger. Sie wurde im Jahre 2011 gegründet und verfügt mittlerweile über 15 Regionalgruppen in Deutschland, 2 in Österreich und 1 in der Schweiz.

Üblicherweise werden monatliche Treffen organisiert, bei denen miteinander diskutiert, gelernt, und natürlich auch programmiert wird.

Eine der wichtigsten, und bei vielen Treffen gerne und häufig praktizierte Technik ist die sog. „Coding Kata“.

Coding Katas

Ein Kata ist ein feststehender Bewegungsablauf aus den fernöstlichen Kampfkünsten, genauer gesagt dem Karate. Dieser wird von den Schülern immer und immer wieder durchlaufen, bis die Bewegungen sprichwörtlich ins Unterbewusstsein übergegangen sind. Ein ähnliches Konzept verbirgt sich auch hinter einem "Coding Kata". Hier geht es jedoch nicht ums Kämpfen, sondern ums entwickeln: Ein Coding Kata ist dabei eine kleine ritualisierte Programmierübung zu einem vorgegebenen Problem. Sie dient Programmieren dazu, ihre Fähigkeiten zu steigern, indem sie sie immer wieder lösen und dabei jedes mal versuchen, einen besseren Weg zu finden. Das wird vor allem auch dadurch interessant, daß sich bis heute weltweit eine Reihe von festen Katas zu verschiedenen gängigen Problemen aus dem Bereich der Softwareentwicklung herausgebildet haben. Diese wurden schon inzwischen schon viele tausend Male gelöst - auf alle nur erdenklichen Arten, und in so gut wie allen gängigen Programmiersprachen. In vielen Fällen kann man daher die eigene Lösung mit bereits bekannten Lösungen vergleichen. Manchmal steht die Schönheit des Quelltextes im Vordergrund, manchmal ist aber auch der Weg das Ziel. So gibt es einige grossartige Mitschnitte von Katas, die im Takt eines begleitenden Musikstückes gelöst werden, und die so der eigentlich "mechanischen" Tätigkeit des Programmierens eine ganz eigene Ästhetik verleihen.

Coding Dojo

Natürlich muss man Coding Katas nicht alleine lösen. Treffen sich 2 oder mehr Menschen, um gemeinsam Coding Katas zu lösen, so spricht man von einem "Coding Dojo". Es gibt verschiedene Formate, ein Coding Dojo durchzuführen. Das klassische Format ist es dabei, 45 Minuten lang „Conways Game of Life“ zu programmieren. Da man dieses Problem in der zur Verfügung stehenden Zeit nicht lösen kann macht es gar keinen Sinn, dies mittels Abkürzungen dennoch zu versuchen. Stattdessen kann man sich darauf konzentrieren, zusammen mit einem Partner im „Pair Programming“-Stil möglichst sauberen Quellcode zu schreiben (z.B. mit einem striken „Test First“-Ansatz). Am Ende der 45 Minuten hört man auf zu programmieren und beginnt eine 15-minütige Retrospektive. Danach löscht man sämlichen zuvor geschriebenen Quellcode, sucht sich einen neuen Partner, und beginnt von vorne. Normalerweise wiederholt man diesen Zyklus 6 mal innerhalb eines Tages. Am Ende hat man am selben Problem mehrmals mit verschiedenen Partnern gearbeitet und eine Menge gelernt. Dies können einfache Dinge sein wie neue Tastenkombinationen in der IDE, aber auch neue Werkzeuge, Vorgehensweisen oder Architektur-Patterns. Ich habe bisher in jedem Coding Dojo, an dem ich bisher teilgenommen habe, so viel Neues gelernt, wie es sonst in einem einzigen Tag sonst noch nicht erlebt habe

Craftsman Swaps

Zu einer Handwerksausbildung im Mittelalter gehörte in vielen Berufen auch eine Verpflichtung zur Wanderschaft (schön beschrieben z.B. auf Wikipedia unter [2]). Gesellen mussten teils für mehrere Jahre auf Wanderschaft gehen, bevor sie sich für die Prüfung zum Handwerksmeister einschreiben konnten. Meistens waren daran auch noch strenge Auflagen geknüpft. So wurde z.B. ein “Sperrgebiet” um den Heimatort vorgeschrieben, und ein Abbruch der Wanderschaft war nur in genau festgelegten Ausnahmefällen möglich. Im Gegenzug waren viele Zünfte dazu verpflichtet, Gesellen auf Wanderschaft aufzunehmen und ihnen Arbeit zu geben.

Auf diese Art und Weise wurde der Wissensaustausch gefördert, und das Ganze in einer Zeit, bevor Internet, Fax oder sogar das Telefon erfunden war.
Beide Seiten, sowohl der Geselle als auch die ihn aufnehmende Firma, hatten davon einen Vorteil:

Der Geselle lernte in verschiedenen Städten und von verschiedenen Meistern, was es ihm erlaubte, in kurzer Zeit viele Aspekte seines Handwerks zu erlernen und zu perfektionieren. Und der ihn aufnehmende Betrieb profitierte von dem neuen Wissen und frischen Ideen, die der Geselle in den Betrieb mit einbrachte.

Dieser Ansatz wird in letzter Zeit auch zunehmend in die Welt der Softwareentwicklung übertragen.

Die “Craftsman Swap”-Bewegung hat sich aus diesem Grund nun daran gemacht, das oben beschriebene Konzept der Wanderschaft nun auch in unsere Zeit zu übertragen. Natürlich mit stark abgeschwächten Regeln. Es gibt keine Mindestdauer und auch keine Sperrzonen o.ä..

Dabei tauschen 2 Personen einfach für einige Zeit die Plätze. Das Ganze geht zurück auf eine Initiative der beiden Unternehmen “8th Light” und “Obtiva” von Anfang 2009. Ziel war es, voneinander zu lernen, neue Einblicke zu erhalten, und nicht zuletzt auch Vertrauen zwischen den beiden Unternehmen zu schaffen. Der originalen Blogpost von Dave Hoover, in dem er das Konzept beschreibt, findet sich unter [3]. Seitdem haben viele dieses Konzept aufgegriffen und ebenfalls Swaps durchgeführt. Ihre Erfahrungen findet man z.B. durch eine entsprechende Google-Suche nach dem Begriff „Craftsmanship Swap“.
Wer das nun für eine gute Idee hält, und sich vorstellen kann, das evtl. sogar selber mal auszuprobieren, findet auf der Seite der „Softwerkskammer“ eine entsprechende Rubrik mit mehr Infos zu diesem Thema (siehe [4]).

All dies soll nur eine kurze Einführung in die Hintergründe und Konzepte des Software Craftsmanship sein.
Wer mehr zum Thema erfahren, oder gar selbst aktiv werden möchte, dem empfehle ich einen Besuch auf den Seiten der Softwerkskammer (siehe [5]), oder einen Besuch eines Treffen der nächstgelegenen Regionalgruppe.

Links

[1]: http://de.wikipedia.org/wiki/Chaos-Studie

[2]: http://de.wikipedia.org/wiki/Wanderjahre

[3]: http://nuts.redsquirrel.com/post/80855433/craftsman-swap

[4]: https://www.softwerkskammer.org/wiki/craftsmanswap/index

[5]: https://www.softwerkskammer.org

Bücher

Dave Thomas, Andrew Hunt: Der Pragmatische Programmierer, ISBN: 020161622X

Pete McBreen: Software Craftsmanship, ISBN: 0201733862