Sonderdruck aus OBJEKTspektrum

Vom Künstler zum Handwerker

Wie Softwareentwickler von den Erfolgsstrategien des Handwerks profitieren

Obwohl Software immer komplexer wird und häufig nur noch von sehr großen Teams erstellt werden kann, ist die Softwareentwicklung selbst stark von persönlichen Präferenzen und Individualleistungen geprägt. Das erschwert oft die Weiterentwicklung bestehender Systeme, die Integration der Ergebnisse mehrerer Teams, den Wechsel eines Entwicklerteams oder den Austausch einzelner Teammitglieder. Eine stärkere Orientierung am Handwerk und dessen Erfolgsstrategien kann die Situation in der Softwareentwicklung verbessern.

Wenn wir an eine medizinische Operation denken, assoziieren wir damit verschiedene Erwartungen. Dazu gehören sterile Operationsinstrumente genau so wie die frisch gewaschenen Hände des Operationsteams. Es ist heute einfach unvorstellbar, dass der Chefarzt in einem Kittel am OP-Tisch erscheint, an dem noch das Blut des Vorgängers klebt.

Das war aber nicht immer so. Noch vor 150 Jahren waren Hygiene und Sterilität für Ärzte nur ein Randthema. Über Jahrtausende führten Mediziner Operationen ohne spezielle Vorbereitungen durch und waren dabei auch erfolgreich. Denn die Operationen erhöhten die Überlebenschancen der Patienten, auch wenn viele in der Folge verstarben. Ohne die Operationen hätten die Patienten meist sowieso nicht überlebt.

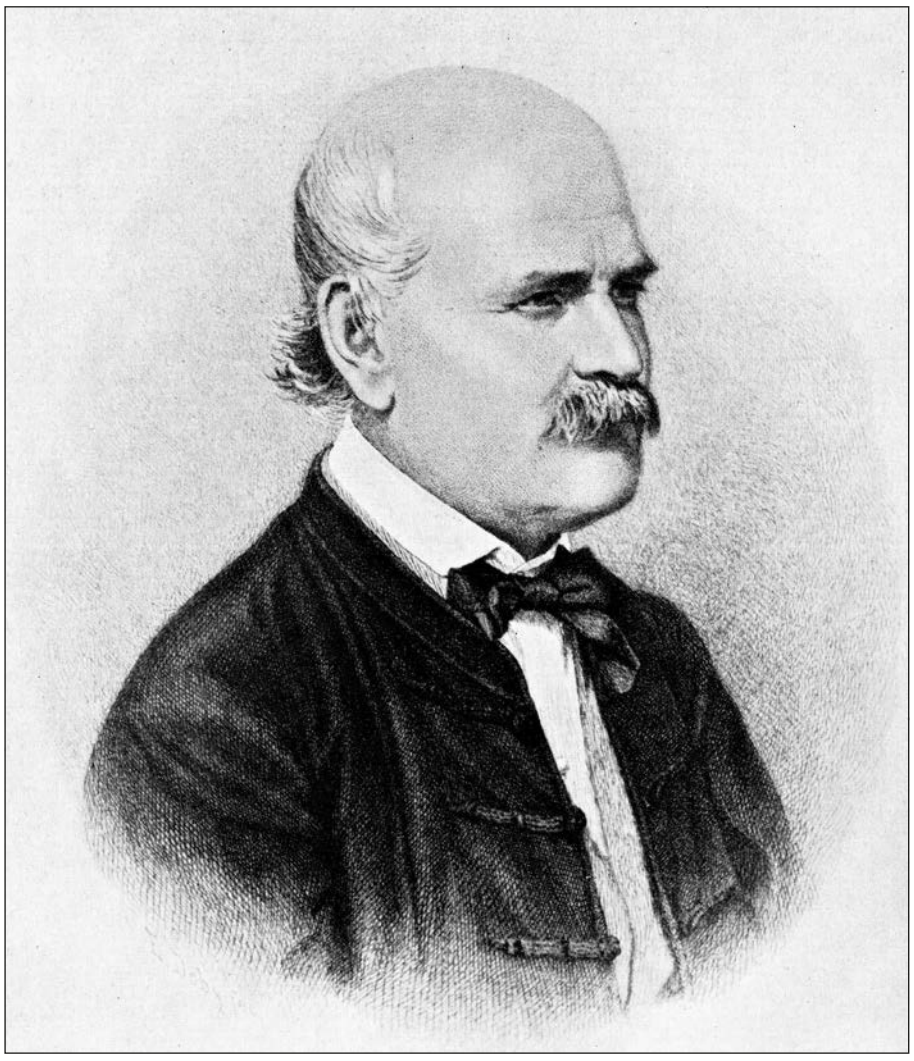

Abb. 1: Ignaz Semmelweis [Dob]

Als Ignaz Semmelweis (**siehe Abbildung 1**) 1847 aufgrund seiner empirischen Studien vorschlug, Hygienevorschriften bei der Patientenbehandlung einzuführen, schlug ihm der Widerstand der gesamten Zunft entgegen. Seine Zeitgenossen lehnten seine Erkenntnisse als „spekulativen Unfug" ab, denn Hygiene war für sie Zeitverschwendung und unvereinbar mit den damals geltenden Theorien über Krankheitsursachen. Erst nach seinem Tod begann die Medizin, seine Ideen anzunehmen, was letztendlich zum heutigen hohen Standard führte. [SMAG, Wiki-b] Heute gibt es sogar eine Vorschrift für medizinisches Personal, die vorgibt, wie die Hände gewaschen werden sollen. [Wiki-a]

Von der Spinnerei zur Routine – Parallelen zwischen Softwareentwicklung und Medizin

In der Softwareentwicklung stehen wir derzeit vor einem ähnlichen Problem wie die Medizin vor der „Ära Semmelweis". Es gibt viele erfolgreiche Programmierer. Deren Produkte werden mit zunehmender Projektlaufzeit immer komplexer. Durch die höhere Komplexität steigt das Risiko für Fehler und zwar nicht nur für die neuen Programmteile, sondern auch für bereits existierende, soweit diese von den Änderungen betroffen sind.

In der Folge erreichen viele Softwareprojekte einen Punkt, an dem die Komplexi-

tät für den einzelnen Programmierer auch in Teilbereichen nicht mehr überschaubar ist, eine Weiterentwicklung aus ökonomischen Gründen nicht mehr sinnvoll erscheint und das Produkt daher von Grund auf neu erstellt werden muss. Oft wird dieser Punkt erreicht, wenn der Hauptentwickler das Projekt verlässt. Das Projekt ist dann „tot".

Was Hygienevorschriften für die Medizin sind, ist *Test-Driven Development* (TDD, [Fre09]) für die Softwareentwicklung. Im Zusammenspiel mit anderen modernen Arbeitstechniken wie *Pair Programming* und *Clean Code* [Mar16] werden Auswirkungen der wachsenden Komplexität mit TDD besser beherrschbar und Softwareprojekte so deutlich länger „am Leben" gehalten. Die Vorteile sind theoretisch aufbereitet und praktisch belegt [Erd05].

Obwohl die Idee vom *Test First*-Ansatz bereits mehr als einer Generation propagiert wird [McC57], sind *Test-Driven Development*, *Pair Programming* und *Clean Code* nur selten in aktuellen Projekten zu finden. Oft werden sie nur sporadisch oder nur ansatzweise eingesetzt. Viele Entwickler bekunden zwar, diese Techniken positiv einzuschätzen, lehnen deren konsequenten Einsatz aber aus vermeintlichem Kosten- und Termindruck ab. Dieselben Aussagen machen Vorgesetzte und Projektleiter.

Obwohl *Test-Driven Development* und *Clean Code* also ein positives Echo aus der Branche erhalten, stehen wir heute an der gleichen Stelle wie Semmelweis, als er seine bahnbrechenden Erkenntnisse veröffentlichte: Der Wille zur konsequenten Umsetzung in der Praxis fehlt.

Laie, Profi, Künstler – Was macht den Unterschied?

Programmieren ist eine kreative Tätigkeit vergleichbar mit dem Schreiben eines Romans. Genau wie ein Roman ist ein Programm nur dann erfolgreich, wenn es sich inhaltlich qualitativ von anderen Programmen unterscheidet. Naturgemäß orientiert sich ein Programm in seiner Form und Funktionalität an dem Problem, zu dessen Lösung es beitragen soll, aber wie es das tut, ist eng mit der Persönlichkeit des Programmierers verknüpft, so wie der Schreibstil des Autors einen Roman prägt. Ein Programm stellt also genau wie ein Roman ein Unikat mit persönlicher Prägung dar. Die Schaffung von Unikaten mit persönlicher Prägung ist die universelle Definition für die Tätigkeit eines Künstlers. Man könnte also einen Programmierer genau wie einen Romanautor als Künstler betrachten.

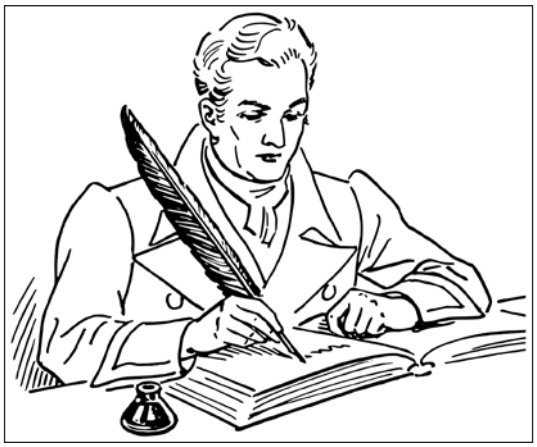

Abb. 2: Künstler [Fore]

Was jedoch Programme von Romanen unterscheidet, ist, dass erstere meist „Gemeinschaftswerke" sind, also von mehreren Personen gleichzeitig erstellt werden, während Romane üblicher Weise von einer Einzelperson (**siehe Abbildung 2**) geschrieben werden. Darüber hinaus werden Programme heute meist auch nach ihrer Veröffentlichung inhaltlich geändert. Während ein Roman nach seiner Erstveröffentlichung nur noch formale Überarbeitungen erfährt, werden Programme oft umfassend weiterentwickelt. Neue Funktionen werden hinzugefügt oder bestehende geändert.

Oft erledigt dies eine andere Person als der ursprüngliche Ersteller. (Wer schon einmal eines seiner eigenen Programme nach mehreren Jahren ändern musste, weiß, was ich meine ...) In diesem Fall treten nun die persönlichen Vorlieben der beteiligten Programmierer in Konflikt. In der Konsequenz führt die Eigenschaft von Programmierern, die ihre Tätigkeit als Kunst betreiben, also ihren persönlichen Programmierstil in das Produkt einfließen lassen, zu vermeidbaren Problemen im Projektalltag.

Doch welches Selbstverständnis soll ein Programmierer dann haben? Naheliegend ist, dass ein Programmierer „professionell" sein sollte. Doch was genau bedeutet das?

Der Volksmund sagt:

„Der Laie macht zufällig etwas richtig, der Profi mit Absicht."

Der Kern dieser Erkenntnis ist, dass der Profi seine Werkzeuge nicht nur bedienen kann, sondern im Gegensatz zum Laien über Prinzipien verfügt, nach denen er sie einsetzt und die letztendlich den Erfolg seiner Arbeit sicherstellen. Diese Prinzipien sind entweder aus eigenen Erfahrungen hervorgegangen oder sie wurden erlernt.

Das Handwerk, zu dem die Medizin lange Zeit gehörte, hat in seiner Geschichte Strategien entwickelt, die Weitergabe von Wissen und Erfahrungen zu organisieren. Was mit den Zünften und der Tradition der Wanderschaft begann, findet heute in konsolidierten Lehrplänen seine Fortsetzung.

In der Softwareentwicklung gibt es diese übergeordnete Instanz der Zunft nicht in dieser strikten Form. Das liegt insbesondere daran, dass Softwareentwickler über verschiedene Wege in ihren Beruf eintreten. So gibt es zwar eine Berufsausbildung zum Programmierer, die meisten Softwareentwickler beginnen ihre Karriere aber mit einem Informatikstudium, in dem das Programmieren nur ein kleiner Teilbereich ist, oder sie beginnen ohne fundierte Grundkenntnisse als Quereinsteiger, die zwar die eine oder andere Programmiersprache gut beherrschen, jedoch nicht die Prinzipien, die Software über lange Zeiträume „am Leben" hält.

Was uns bisher fehlt, ist eine übergeordnete Instanz, die in der Lage ist, das Niveau für diese verschiedenen Karrierewege anzugleichen. Aus meiner Sicht sollte sich die „Community" zu dieser Instanz entwickeln. Damit die „Community" diese Rolle einnehmen kann, müsste sie jedoch zunächst selbst zu einer Einheit zusammenwachsen. Das aktuelle Konglomerat aus diversen *User Groups* ist nicht in der Lage, diese Vereinheitlichung zu bewirken.

Meine Hoffnung ist, dass die *Clean Code Developer*-Bewegung, die sich als weiteres Element der Entwickler-Community gebildet hat, der Kristallisationspunkt wird, den es benötigt, um die erwähnte übergeordnete Instanz zu schaffen.

Clean Code Developer's Manifest

Die von Stefan Lieser und Ralf Westphal initiierte *Clean Code Developer*-Bewe-

Literatur & Links

[CCD] St. Lieser, R. Westphal, Clean Code Developer – Eine Initiative für mehr Professionalität in der Softwareentwicklung, siehe: **http://clean-code-developer.de**

[CCDS] R. Westphal, St. Lieser, Clean Code Developer School, siehe: **http://ccd-school.de/coding-dojo/**

[CodeRe] C. Haines, Coderetreat Community Network, siehe: **http://coderetreat.org/**

[Dob] Gemeinfreies Bild des Kupferstichs von Ignaz Semmelweis von J. Doby von 1869, siehe: **https://de.wikipedia.org/wiki/Datei:Ignaz_Semmelweis.jpg**

[Erd05] H. Erdogmus, On the effectiveness of test-first approach to programming, in: Proc. of the IEEE Trans. on Software Engineering, (31), 1 2005

[Fore] Pearson Scott Foresman, Quill (PSF), siehe: **https://es.wikipedia.org/wiki/Archivo:Quill_(PSF)_vector.svg**

[Fow] M. Fowler, TestPyramid, siehe: **https://martinfowler.com/bliki/TestPyramid.html**

[Fre09] St. Freeman, N. Pryce, Growing Object-Oriented Software, Guided by Tests, Pearson Education, 2009

[Gar87] S. Garrett, G. Ballard, Man in the mirror, 1 1988. From the album Bad, released August 31, 1987, siehe auch: **https://de.wikipedia.org/wiki/Man_in_the_Mirror**

[Gre13] J. Grenny u. a., Influencer: The New Science of Leading Change, McGraw-Hill, 2. Aufl., 2013

[Mar16] R. C. Martin, Clean Code – Refactoring, Patterns, Testen und Techniken für sauberen Code, Dt. Ausgabe, MITP-Verlags GmbH & Co. KG, 2013

[McC57] D. D. McCracken, Digital Computer Programming – One of Series Written by General Electric Authors for the Advancement of Engineering Practice, London, Chapman & Hall, 1957

[SMAG] Schülke & Mayr AG, Geschichte der Hygiene, siehe: **https://www.schuelke.com/ch-de/Wissensportal/article/Geschichte-der-Hygiene.php**

[Wiki-a] Wikipedia-Community, Händedesinfektion/Methode, siehe: **https://de.wikipedia.org/w/index.php?title=Händedesinfektion&oldid=180008317#Methode**

[Wiki-b] Wikipedia-Community. Ignaz Semmelweis, siehe: **https://de.wikipedia.org/w/index.php?title=Ignaz_Semmelweis&oldid=181327568**

gung [CCD] propagiert qualitative Verbesserung in der Softwareentwicklung durch Orientierung an einem gemeinsamen Wertesystem. Dieses Wertesystem basiert auf dem Gedanken, dass es minimale Anforderungen an *Professionalität* gibt und es in der Entwickler-Community einen gemeinsamen Konsens über diese Anforderungen geben sollte.
Nach Meinung der Initiatoren stellt Robert C. Martins Buch *Clean Code* [Mar16] aktuell die beste Zusammenstellung von Arbeitsprinzipien in der Softwareentwicklung dar und sollte deshalb zur Pflichtlektüre jedes Softwareentwicklers gehören.
Aus diesem Buch leiten Lieser und Westphal die vier Werte ab, an denen Softwareentwicklung grundsätzlich ausgerichtet sein sollte:

- Evolvierbarkeit,
- Korrektheit,
- Produktionseffizienz und
- kontinuierliche Verbesserung.

Als praktische Empfehlung stellen sie Praktiken und Prinzipien vor, die konkrete Hilfestellung bei der Umsetzung dieser Werte geben.
Selbstverständlich ist es unmöglich, die Praktiken und Prinzipien sofort zu verinnerlichen und anzuwenden. Es braucht Zeit zum Lernen und Üben. Aus diesem Grund wurde in Anlehnung an den asiatischen Kampfsport ein System von *Graden* entwickelt, das diesem Umstand Rechnung trägt und es dem Programmierer ermöglicht, sich strukturiert mit dem Thema zu befassen und sich selbst weiterzuentwickeln.
Genau diese, von einer konkreten Programmiersprache unabhängige Herangehensweise unterscheidet die „Clean Code Developer"-Initiative von den Gruppierungen, die sich bisher in der Entwicklergemeinde etabliert haben. Sie ist notwendig für eine breite Akzeptanz über die gesamte Community hinweg.

Den Wechsel fördern

Wie bereits erläutert, existiert ein Widerspruch zwischen dem zurückhaltenden Einsatz von *Test-Driven Development* und *Clean Code* in realen Projekten und der doch grundsätzlich positiven Einstellung aller Beteiligten dazu. Die Ursache dafür liegt in den komplexen Zusammenhängen, die in *The Influencer* beschrieben werden:

"For every behavior you see the world around that person is perfectly designed for that behavior to happen."
[Gre13]

Tabelle 1 zeigt, welche Aspekte das konkrete Verhalten einer Person bestimmen. Werfen wir also einen Blick auf konkrete Gründe, die uns von dem eigentlich als sinnvoll erachteten Verhalten abhalten.
Ein wesentlicher Grund ist aus meiner Sicht, dass IT-Projekte oft mithilfe externer Berater gestartet werden. Diese Personen begleiten die Projekte oft nur bis zur initialen Inbetriebnahme. Sie sind nur selten an der Weiterentwicklung der Programme, die sie entwickeln, beteiligt. Aus diesem Grund können sie nicht von den Vorteilen, die *Test-Driven Development* und *Clean Code* bieten, profitieren, leiden andererseits auch nicht, wenn der Code nicht *clean* und die Testabdeckung gering ist. Für sie sind straffe Terminpläne, die

	Motivation	Fähigkeiten
persönlich	Ist die Person vom positiven Verhalten überzeugt?	Ist die Person physisch in der Lage, positives Verhalten zu zeigen bzw. negatives Verhalten zu vermeiden?
Gruppe (sozial)	Reagieren die Gruppenmitglieder direkt auf negatives Verhalten?	Folgt die Person anderen, die negatives Veralten zeigen?
Unternehmen (Umgebung)	Unterstützt die Organisation negatives Verhalten bzw. wird positives Verhalten nicht unterstützt?	Fördern die physikalischen Bedingungen negatives Verhalten?

Tabelle 1: Verhaltenseinflüsse [Gre13]

eng mit ihrem wirtschaftlichen Erfolg verknüpft sind, die Regel. In so einem Umfeld gehört schon viel Enthusiasmus dazu, *Clean Code* als Qualitätsanspruch und *Test-Driven Development* als Weg dahin konsequent durchzuhalten.

Eine Besserung dieser Situation wird eintreten, wenn *alle* Beteiligten, also nicht nur die Entwickler selbst, sondern auch deren Vorgesetzte und Kunden, nicht nur der „sichtbaren" Qualität eines Programms, also seiner optischen Erscheinung, der Usability und der fachlichen Korrektheit, sondern auch seiner „versteckten" Qualität, also der Umsetzung von Programmierrichtlinien und dem Vorhandensein automatisierter Tests in verschiedenen Ebenen entsprechend der Testpyramide [Fow], einen Wert zuweisen.

Davon sind wir aber noch weit entfernt und so liegt es an uns Programmierern, die Idee vom *Clean Code* mithilfe der Arbeitsmethode *Test-Driven Development* umzusetzen. Vorgesetzte und Kunden können uns unterstützen, indem sie begünstigende Umstände organisieren, doch wir schaffen am Keyboard Tatsachen. Wir können uns die notwendigen Fähigkeiten und Fertigkeiten unabhängig von der Arbeitssituation aneignen. Die „Clean Code Developer"-Initiative gibt uns das notwendige theoretische Rüstzeug und *Community Events* wie *Coding Dojos* [CCDS] oder der jährliche *Global Day of Coderetreat* [CodeRe] geben uns die Gelegenheit zum Üben. Es liegt an jedem einzelnen Entwickler, diese Gelegenheiten zu nutzen.

Fazit

Die Medizin benötigte mehr als eine Generation, um die Erkenntnisse von Ignaz Semmelweis in alltäglich gelebte Praxis umzusetzen. Obwohl die Softwareentwicklung im Allgemeinen als „schnelllebig" betrachtet wird, benötigen anscheinend auch hier die Veränderungen der Standards für professionelles Arbeiten ihre Zeit.

Auf der anderen Seite sind wir Softwareentwickler genau wie Mediziner gehalten, unseren Kunden und Anwendern mit unserer Tätigkeit nicht zu schaden. Deshalb sollten wir Entwickler mit derselben Selbstverständlichkeit, mit der wir heute von Medizinern erwarten, dass sie sich die Hände waschen, unseren Code professionell, *clean* und *test driven* erstellen.

Der Weg dahin führt aus meiner Sicht über das Bekenntnis jedes einzelnen Programmierers zum *Clean Code Developer's Manifest*. Deshalb appelliere ich an alle Kollegen mit den Worten von Michael Jackson [Gar87]:

And No Message Could Have
Been Any Clearer
If You Wanna Make The World
A Better Place
Take A Look At Yourself, And
Then Make A Change

Der Autor

Thomas Papendieck
(thomas.papendieck@opitz-consulting.com) ist sowohl Elektroingenieur als auch Informatiker. Nach zwölf Jahren als Radartechniker bei der NVA/Bundeswehr ist er seit nunmehr 13 Jahren Entwickler im Java- und PL/SQL-Umfeld.