CASSANDRA CROSSINGS

Bureau indépendant d'analyses et d'études

Werden Sie wieder ein bisschen Aufsicht übernehmen?

Eine sozial-technische Studie

Cassandra Crossings, ca. Dezember 2025

Unsere Berufserfahrung gab uns die Möglichkeit, die Gegebenheiten bei der Herstellung einer IT-Anwendung, von den Designprämissen bis hin zu Nachhaltigkeitsproblemen, genau zu beobachten.

Bevor wir Lösungen anbieten, schauen wir uns die IT-Entwicklungsprozesse an.

++++

Wenn der Weise auf den Mond zeigt, sieht der Idiot nur den Finger…

In dieser Geschichte ist der Weise der Kunde !

Er wünscht sich: Er bittet den Idiot, ihm den Mond zu holen. Und den Mond, den er mit Nachdruck bezeichnet, ist eine Software, die seinen Erwartungen und Bedürfnissen entspricht.

Der Kunde benötigt eine Anwendung, die bestimmte Aufgaben übernehmen kann. Der Kunde erstellt eine Ist-Analyse und verfasst mit diesen Erkenntnissen ein Pflichtenheft, das er dem Lieferanten zur Verfügung stellt.

Beide vereinbaren ihre jeweiligen Erwartungen und Anforderungen. Der Kunde verpflichtet sich, dem Entwickler alle Elemente zur Verfügung zu stellen, die dieser anfordert, und der IT-Spezialist verpflichtet sich, eine Anwendung zu entwickeln, die die bereitgestellten Informationen berücksichtigt.

Nach Beendigung der Arbeit stellt der Lieferant dem Kunden das so hergestellte Produkt zur Verfügung und der Kunde zahlt dem Lieferanten den vereinbarten Betrag.

Der Idiot ist die Computerwelt.

Angesichts der Kundennachfrage verpflichtet sich die IT-Welt, qualitativ hochwertige Arbeit zu erbringen, die die Erwartungen des Kunden erfüllt. Dazu folgt sie einem Verfahren, einem Arbeitsplan, der durch die Erfahrung ihres Fachverbandes validiert wurde. Die gewählte Struktur soll als Leitplanke dienen, um mögliche Abweichungen zu verhindern.

Die IT-Welt unterscheidet sich nicht von anderen. Als das Bedürfnis nach Führung wurde, beobachtete die IT-Welt die Lösungen anderer Berufsfelder und baute auf den Analogien, die zum Zeitpunkt dieser Überlegungen am zielführendsten erschienen, ihre eigene Lösung auf.

Wir werden uns diese Lösungen ansehen, die die IT-Welt in den letzten Jahrzehnten implementiert hat, und wir werden sehen, ob sie ihre Versprechen gehalten haben, das heisst, ob sie dem Idioten geholfen haben, dem Weisen das zu liefern, was er von ihm verlangt hat…

Situation in den 1960er Jahren

Mitte des Jahrzehnts kam es zu einem Umschwung: Der Preis der Software übersteigt den der Hardware, und die Kluft zwischen ihnen wird immer grösser werden.

1968 zog die erste **NATO-Konferenz über Software-Engineering** eine schlechte Bilanz:

Programme werden verspätet ausgeliefert und Budgets werden nicht eingehalten.

Aber diese Unannehmlichkeiten hätten hingenommen werden können, wenn die Software nicht zu schwerwiegenderen Folgen geführt hätte: Die Programme entsprechen nicht den Anforderungen. Ihre Anwendung ist zu komplex und es treten häufig Fehler auf.

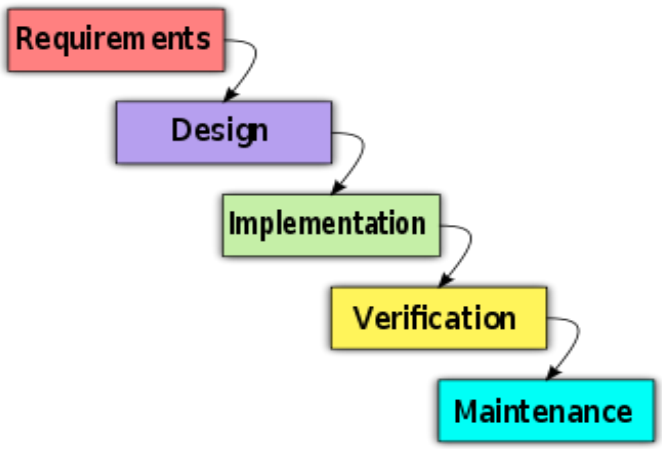
Rosine auf die Torte (wenn man es so ausdrücken will): Die interne Struktur der Programme ist anarchisch, die Wartung ist sehr schwierig, das Debuggen ist kompliziert und folglich kann die Skalierbarkeit nicht kostengünstig garantiert werden.

Ab diesem Zeitpunkt wird die Verringerung der Toleranz gegenüber Konstruktions- und Programmierungsfehlern zu einem Bewusstsein für deren finanzielle und organisatorische Folgen führen. Der zunehmende Druck auf bessere Softwarequalität wird dazu führen, dass sich viele mehr oder weniger wirksame Lösungen entwickeln.

Situation in den 1970er Jahren

Auf diese berechtigten Kritikpunkte wurde ein Framework geschaffen, welches die Entwicklung seitens Lieferanten strukturiert.

Als erste Struktur wurde die Kaskadenmethode (Wasserfall) angewendet, die von den im Ingenieurbau gebräuchlichen Methoden inspiriert wurde.



Jeder Schritt muss in einer logischen Reihenfolge erfolgen: Zuerst fragt man den Kunden, was er erwartet, dann zeichnet man die Pläne, dann baut man das Fundament, dann die Wände und dann das Dach. Und wenn ein Schritt erreicht ist, gilt er als endgültig abgeschlossen. Daher der Name «Wasserfall»: Wenn man eine Treppe überschritten hat, kommt man nicht mehr zurück.

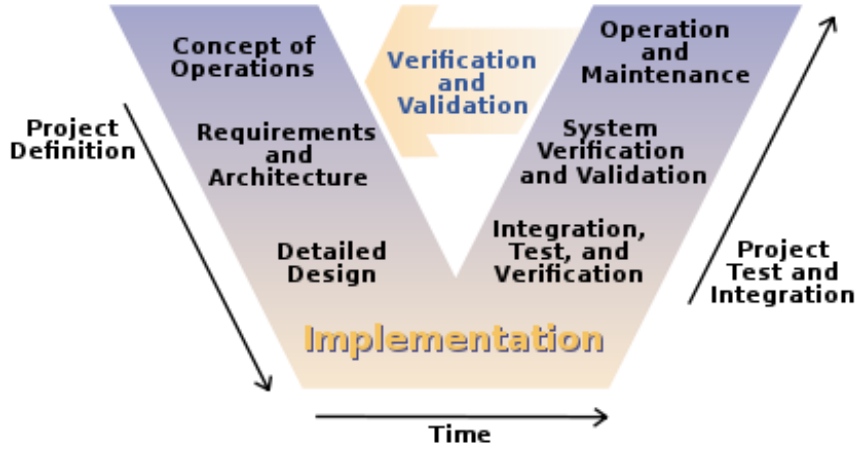
Neben diesen Arbeiten, die darauf abzielen, den Anforderungen des Anbieters besser gerecht zu werden, wurden in der IT-Community Empfehlungen entwickelt: das Hinzufügen von Kommentaren in den Code, um ihn besser verständlich zu machen, die Versionalisierung der Arbeiten, um die Rückkehr zu einem stabilen früheren Zustand zu erleichtern, die Definition eindeutiger und gut dokumentierter Testverfahren, um eine reproduzierbare Überwachung des Arbeitsfortschritts zu gewährleisten. Diese Empfehlungen gelten weiterhin.

Situation in den 1980er Jahren

Das Wasserfallmodell war sehr unflexibel in zeitlicher und organisatorischer Hinsicht. Jeder Schritt musste abgeschlossen werden, bevor der nächste in Angriff genommen wurde – zumindest theoretisch. Infolgedessen war es sehr schwierig, eine in einem früheren Schritt getroffene Entscheidung nachträglich in Frage zu stellen.

Insbesondere die Überprüfungen am Ende des Prozesses konnten zu Problemen führen, deren Behebung sehr schwierig und kostspielig war. Jede Korrektur konnte auch bedeuten, dass ein mehr oder weniger grosser Teil der Arbeit verloren ging.

Das Modell namens «V», das aus der Weiterentwicklung des Wasserfall-Modells hervorgegangen ist, hat es ermöglicht, diese Einschränkungen besser zu berücksichtigen.



Der absteigende Zweig des V stellt die Konzeption und Programmierung dar, die von der Abstraktion der allgemeinen Konzepte bis hin zur Konkretisierung der einzelnen Elemente geht.

Der aufsteigende Zweig repräsentiert die Tests, deren Generalisierungsgrad mit dem entsprechenden Element des absteigenden Zweigs parallel gesetzt werden kann. Die einzelnen Elemente, die beim Erreichen des Fusses des absteigenden Zweigs programmiert werden, entsprechen den Einheitstests derselben Elemente. Wenn zu diesem Zeitpunkt ein Fehler auftritt, ist die Feedbackschleife zur Korrektur des betreffenden Elements sehr kurz, d.h. sehr kostengünstig.

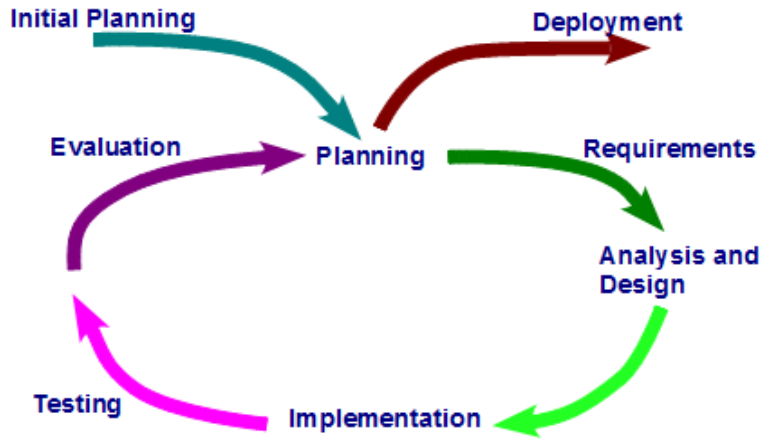
Der nächste Schritt im aufsteigenden Zweig ist die Validierung der Module. Wenn ein Modul ausfällt, wird die Feedbackschleife länger sein, da ein oder mehrere Elemente des betreffenden Moduls korrigiert werden müssen. In der Praxis ist je höher der Arbeitsfortschritt entlang des aufsteigenden Zweigs ist, desto geringer ist die Wahrscheinlichkeit, dass ein Fehler eine korrigierende Rückmeldung erfordert. Aber umso schwerwiegender ist die mögliche Korrektur.

Situation in den 1990er Jahren

Die Realisierung immer komplexererer Software mit vielen Modulen hat zu einem erheblichen Eigengewicht des Modells V geführt: Die Modifikation eines Moduls ist zwar teuer, aber das Problem wird deutlich verschärft, wenn die Korrektur eines Moduls zu einer Kaskade von Korrekturen in anderen Modulen der Anwendung führt.

Der Kostenanstieg kann in einer solchen Situation explosiv werden. Zudem kann der Kunde erst in einem letzten Schritt schlussendlich prüfen, ob die Arbeit seinen Erwartungen entspricht.

Aus einer Weiterentwicklung des Modells V entstand das inkrementelle Spiralmodell. Aus diesem inkrementellen Modell stammen die heutigen iterativen Prozesse wie Rapid Application Development, Scrum oder Extreme Programing. Die meisten dieser Prozesse werden unter dem Agilen Manifest zusammengefasst.



Das Agile Manifest

Die gängige agile Denkweise umfasst mehrere Methoden, die gemeinsame Werte und Praktiken anerkennen. Diese Methoden fordern einen pragmatischen Ansatz, bei dem der Kunde während des gesamten Entwicklungsprozesses stark eingebunden wird.

Werte Agil

Die vier Grundwerte sind:

- **Das Team**: Laut Agile sind die einzelnen Personen des Projektteams wichtiger als die eingesetzten Tools, und der Teamgeist wird stark gefördert.
- **Die App**: Die korrekte Funktionsweise der App ist entscheidend. Eine funktionierende App ist besser als eine vollständige Dokumentation.
- **Zusammenarbeit**: Der Kunde wird in die Entwicklung einbezogen. Er muss mit dem Team zusammenarbeiten und explizit Feedback geben, dass das Produkt seinen Bedürfnissen entspricht.
- **Akzeptanz der Veränderung**: Produktplanung und -struktur müssen flexibel sein, damit das Produkt an die sich verändernde Kundennachfrage angepasst werden kann.

Agile Prinzipien

Diese vier Werte sind in 12 Prinzipien untergliedert, die allen agilen Methoden gemeinsam sind:

I. Kundenzufriedenheit durch regelmässige Bereitstellung von Funktionen mit hohem Mehrwert.

II. Offenheit für Veränderungen, auch spät, um dem Kunden einen Wettbewerbsvorteil zu verschaffen.

III. Die häufige Auslieferung von lauffähiger Software in kürzest möglichen Zyklen.

IV. Die Zusammenarbeit zwischen den Anwendern sowie deren Vertretern und den Entwicklern über den gesamten Projektverlauf.

V. Die Motivation der Beteiligten.

VI. Der Dialog, möglichst physisch.

VII. Eine funktionsfähige Software ist der wichtigste Schritt vorwärts.

VIII. Nachhaltiges Entwicklungstempo.

IX. Ein kontinuierliches Augenmerk auf technische Exzellenz und gutes Design.

X. Einfachheit – also die Kunst, unnötige Arbeit zu minimieren – ist entscheidend.

XI. Die besten Architekturen, Spezifikationen und Designs entstehen aus selbstorganisierten Teams.

XII. Das Team überlegt in regelmässigen Abständen, wie es effektiver werden kann und justiert und verändert sein Verhalten entsprechend.

All diese Werte und Grundsätze sollen sicherstellen, dass ein Produkt geliefert wird, das den Wünschen des Kunden entspricht, da der Kunde vom Anfang bis zum Ende des Prozesses involviert war und die Teillieferungen und Kontrollen in jedem Schritt sicherstellen sollten, dass die Anwendung genau auf die Erwartungen und Bedürfnisse des Kunden abgestimmt ist.

Wie lässt sich unter diesen Umständen solch schlimme Fehlschläge wie das Fiasko der IT-Umgebung des USA Militärflugzeugs F-35 erklären?

Wie erklärt die Schweizerische Eidgenossenschaft den Verzicht auf die Entwicklung von Programmen, die hundert Millionen Franken gekostet haben? (ESTV INSIEME, 150 mio CHF oder VBS FIS, 700 mio CHF)

Oder der Untergang der neuen Arbeitslosenmanagement-App des SECO (Schweiz, Staatssekretariat für Wirtschaft), eine dedizierte App für die Arbeitslosenverwaltung, die mehr als 6 Monate lang Korrektur-Patches erfordert hat ?

Unser Wissen über diese Welt hat uns gelehrt, dass sich die Gesetze der IT-Entwicklung nach der Einführung der agilen iterativen Umgebungen nicht mehr verändert haben.

Schon Charles Louis de Secondat, Baron von Montesquieu, lehrte uns in seinem fast 300 Jahre alten Werk: «Wenn ein Gesetz schlecht ist, muss es geändert werden.»

Worauf wartet die IT-Welt, um ihre Gesetze zu ändern?

Wir werden diese sehr berechtigte Frage in den nächsten Absätzen beantworten, und unsere Antwort wird genauso explosiv sein wie die Kostensteigerung der beanstandeten Computerprogramme.

Wir werden die Gründe beleuchten, die eine Verbesserung der Qualität von IT-Anwendungen verhindern, wir werden erklären, warum Budgets nur mehrfach überschritten werden können. Wir werden aufzeigen, warum wir in eine IT-Krise zurückfallen, die grösser ist als die ursprüngliche Krise von 1968, aber wir werden auch die Schlüssel zu den Abhilfemassnahmen aufzeigen.

Krise? Welche Krise?

Krise verstehen

(Anmerkung: Diese Analyse stammt aus dem Jahr 2007, wir übernehmen sie unverändert, da sich die Situation, die sie veranlasste, nicht grundlegend geändert hat.)

Mehr als 40 Jahre nach der NATO-Konferenz waren die zur Softwareentwicklung verwendeten Techniken nicht in der Lage, diese Fragen zu beantworten, und die zunehmende Komplexität der grossen Systeme hat das Problem noch verschärft.

Modelle im Vergleich

Wir haben diese Muster analysiert und herausgefunden, wie sie auf als «Software-Krise» bekannte Schwachstellen reagieren.

	Übergreifende Steuerung	Testing	RETEX (Erfahrungsaustausch)	Anforderungen	Budgetierung
Wasserfall	I.O.	-	-	-	Nicht bestanden
V-Model	I.O.	+/-	-	-	Nicht bestanden
Iterativ/Agil	I.O.	I.O.	+/-	+/-	Nicht bestanden

Das **Wasserfall-Modell** bietet eine umfassende Kontrolle über den Projektlebenszyklus, löst aber nicht die anderen Probleme, z. B. die Tatsache, dass Tests erst am Ende des Prozesses Fehler erkennen, wenn das Budget nicht mehr ausreicht, um sie zu beheben.

Das **V-Modell** erfüllt auch die globale Kontrolle und implementiert ein effektives Testverfahren für kleine Teile der Software (Unit-Tests), kann aber das Fehlerproblem eines ganzen Moduls nicht beheben. Mit dem V-Modell ist es möglich, Fehler zu erkennen und innerhalb des Budgets zu korrigieren, sofern diese Fehler klein sind und geringe Auswirkungen haben. Deshalb wird der Test mit (+/-) bewertet. Bei grösseren Fehlern schlägt die Budgetkontrolle fehl.

Die Familie der **iterativen Modelle** verfügt über eine ziemlich effektive globale Kontrolle und testet genau den geschriebenen Code. Das Feedback der Benutzer wird in die nachfolgenden Iterationen eingespeist, um Fehler zu beheben. Diese Rückmeldungen sind ein grosser Vorteil von iterativen Familien und ermöglichen eine mehr oder weniger gute Erfüllung der Anforderungen des Benutzers, d.h. um genau die Anforderungen zu erfüllen, kann es erforderlich sein, mehr Iterationen durchzuführen als erwartet, was zu einer Budgetüberschreitung führen kann. Daher werden die Anforderungen (+/-) notiert und das Budget scheitert.

Soweit zur Theorie.

In der Praxis haben Waterfall- und V-Modelle so gravierende Einschränkungen, dass sie zugunsten von iterativen Modellen aufgegeben wurden. Aber selbst solche Modelle, die alle Mängel beheben sollen, scheitern kläglich und führen manchmal zum völligen Abbruch des Projekts.

Das Scheitern von iterativen Modellen erklärt

Das Scheitern von iterativen Modellen ist auf zwei Ursachen zurückzuführen, die miteinander keine Verbindung haben.

Der erste Faktor ist ein Skaleneffekt: Alle Mitglieder der Gruppe, die das Agile Manifest verfasste, waren Experten auf höchstem Niveau, jeder auf seinem Gebiet. Sie waren es gewohnt, mit kleinen Gruppen von Followern mit sehr hoher Kompetenz zu arbeiten.

Diese Doppelauswahl (kleine Gruppe und hochrangige Gruppe) hatte einen besonders perversen Effekt: Alle Empfehlungen der Mitglieder des Manifests konnten nur in Arbeitsgruppen mit denselben Merkmalen, d. h. in kleinen Expertengruppen, umgesetzt werden.

Unglücklicherweise wurden Kunden, die Apps kauften, die unter der Ägide des Agilen Manifests entstanden sind, ihre Produkte von gemeinen Menschen gebaut, die in grossen Strukturen arbeiteten. Die Skalierung konnte nur zum völligen Scheitern führen, weil keine der Bedingungen für die Modellerstellung gegeben war.

Um das Agile Manifest auf grosse Teams mit normal kompetenten Personen anwendbar zu machen, hätte es von ganz normalen Leuten geschrieben werden müssen, die Mitglieder grosser Teams sind. Sie hätten Banalitäten aufgestellt, und niemand hätte ihnen die geringste Aufmerksamkeit geschenkt.

Die Verfasser des Manifests wurden angehört, weil sie anerkannte und respektierte Gurus waren, und genau dieser Status als Gurus unter den Gurus führte zum Scheitern des Modells.

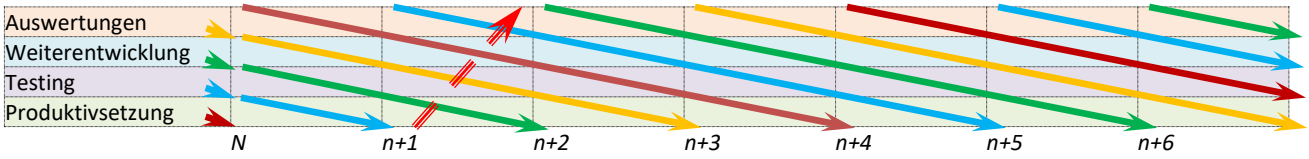
Ein weiterer Grund für das Versagen iterativer Modelle, qualitativ hochwertige Anwendungen bereitzustellen, ist der Paradigmenwechsel im Finanzbereich.

Mit den Modellen Waterfall und V bekam der Kunde immer eine voll funktionsfähige, entwickelte und getestete App als fertiges Produkt. So blieb es den Kunden frei, je nach Neuerungen in der App eine neue Version zu erwerben oder auf diese zu verzichten.

Für den Käufer bedeutete dies grosse Handlungsfreiheit und erhebliche Kosteneinsparungen. Für den Hersteller war jede neue Version ein Risiko. Würden die neuen Funktionen einem Publikum begegnen, das sie kaufen wollte? Es sei denn, der Hersteller verpasste das Häkchen einer neuen Nische, die zu plötzlich auftauchte, als dass sie vorhergesehen werden konnte und das Produkt von einem Tag auf den anderen obsolet machen würde?

Durch die iterative Entwicklung hat sich die IT-Welt grundlegend verändert.

Nun würde das Produkt stetig weiterentwickelt: Das Konzept der Stable-Version wurde abgeschafft und durch den Sprint ersetzt, der das Allheilmittel sein sollte, um Kundenwünsche schnell zu integrieren.



Bei den Iterationen laufen die vier Hauptschritte der Softwareentwicklung parallel ab. Sobald ein Team die Arbeiten für einen bestimmten Schritt abgeschlossen hat, übergibt es seinen Entwurf an das nächste Team für den nächsten Schritt und beginnt mit den Arbeiten für Schritt ***n* + 1** in dem Entwurf, den es ihm vom vorherigen Team übergeben hat. Man kann buchstäblich von einer Entwicklung auf dem Laufband sprechen.
Wenn ein Problem auftritt, z. B. wenn der Benutzer einen Fehler in der produktiven Anwendung findet, wird dieser Fehler an das Analyseteam weitergeleitet, das eine Iteration hat, um den Fehler zu prüfen und eine Korrektur zu erstellen. Diese Korrektur wird im nächsten Schritt an die Entwickler zur Korrektur weitergeleitet, welche Korrektur wird später getestet usw., bis die Anwendung an den Client zur Produktion übergeben wird. Bei einer monatlichen Iteration dauert es 5 bis 6 Monate, bis der Fehler behoben wird.

Wenn eine neue Iteration auf den Markt kommt, folgen bereits drei Nachfolger, die sich bereits in der Entwicklung befinden und sich jeweils in einer anderen Phase befinden. Das Feedback einer Version *n* wird daher in die Version *n* + 4 oder 5 integriert. Ebenso müssen Fehlerbehebungen auf eine ähnliche Dauer warten.

Wie wir bereits in unserer Studie zum **Kultur- und Sprachvergleich** beschrieben haben, haben wir es auch mit einer deutlichen Diskrepanz zwischen dem Verhalten, das im kommerziellen Diskurs behauptet wird, und dem Verhalten, das tatsächlich ausgedrückt wird, zu tun.

Der kommerzielle Diskurs ist, dass der Anbieter zum Wohle seiner Kunden arbeitet. Tatsächlich gibt es für den Anbieter nicht nur keinen Anreiz, Qualität zu produzieren, sondern es hilft sogar, fehlerhafte Anwendungen zu produzieren.

Es gibt keine Black Magic hinter dieser Behauptung. Es gibt die einfache Feststellung, dass man den Kunden, der an seinem Abonnement festgehalten wird, durstig machen muss, damit er weiter bezahlt.

Unter der Annahme, dass ein Provider (zu seinem grössten Unglück) eine Iteration herausbringt, die alle vom Kunden gewünschten Funktionen enthält, riskiert er, dass mehr oder weniger Kunden ihm mitteilen, dass sie das Abonnement kündigen möchten, um ein Verkaufsangebot für die betreffende Version anzufordern.

Eine solche Situation wäre für den Zulieferer dramatisch: Statt einer Kuh die so regelmässig wie eine Uhr gemolken wird, würde er plötzlich sehr viel Geld einfahren, aber ohne Folgen.

Und wenn dieser Anbieter nicht schnell neue Kunden findet, die die gerade verlassenen Kunden ersetzen, riskiert er, den Betrieb einzustellen. Zusammenfassend lässt sich sagen, dass eine gute App ein wahrer Alptraum ist !

Iterative Modelle sind daher die perfekte Antwort auf diesen wirtschaftlichen Alptraum. Sie versprechen eine unzerstörbare Verbindung zwischen dem Lieferanten und seinem Kunden. Leider muss die Medaille ohne Rückschläge noch erfunden werden. Iterative Modelle haben den Kunden zu seinem Unglück in eine Milchkuh verwandelt, zum Glück des Lieferanten, der sich nicht einmal mehr anstrengen muss, um Qualität zu produzieren.

Diese Situation stellt uns in Frage, denn sie entspricht nicht der Realität vor Ort: Der Bedarf an IT-Anwendungen ist so gross und wächst so schnell, dass IT-Unternehmen nie einen Mangel an Kunden haben werden. Es wäre also durchaus möglich, qualitativ hochwertige Anwendungen zu produzieren, ohne wirtschaftliche Hungersnot befürchten zu müssen.

Tatsächlich deutet alles darauf hin, dass eher das Gegenteil der Fall sein könnte, d.h. ein Anbieter, der sich durch optimale Qualität auszeichnet, wird mit Sicherheit mit Entwicklungsanforderungen von Kunden überflutet, die sich mit fehlerfreien Applikationen ausrüsten möchten.

Zusammenfassend lässt sich sagen, dass der Fokus, der seit der NATO-Konferenz von 1968 auf Entwicklungsmodelle gelegt wurde, völlig verloren ging, da diese Frage der Entwicklungsmodelle viel mehr mit dem wirtschaftlichen Kontext der IT-Entwicklung zu tun hatte als mit der Erzielung besserer Qualität.

Update 2025

Wir wurden vor kurzem eingeladen, an der periodischen Überprüfung der Entwicklungsgruppen eines IT-Departments teilzunehmen und konnten die Veränderungen in der Führung, seit wir in diesem Bereich im letzten Jahrtausend anfingen, miterleben.

Das, was uns am meisten überraschte, war, dass alles nur noch aus Metriken, Fortschrittsdiagrammen und Erfolgsquoten besteht. Wo bleibt der Mensch, der Benutzer-Kunde in diesem Selbstbeweihräucherungs-Wirrwarr?

Natürlich, diese endlosen Litaneien von Erfolgserklärungen schmeichelten diesem kleinen Kreis, aber dieser hyperaktive Narzissmus verdeckt die Tatsache, dass alles, was mit der Benutzererfahrung zu tun hat, völlig vernachlässigt wurde.

Tatsächlich dienten all diese Präsentationen nur dazu, zu zeigen, wie jedes Team mit den im letzten Quartal vorgestellten Meilensteinen übereinstimmt und wie die heutigen Ergebnisse als Sprungbrett für die Ziele des nächsten Quartals dienen.

All diese Managementmethoden haben übertriebene Namen, die ihre transatlantische Herkunft verraten, und in allen Mitteilungen wurde ein Fachjargon verwendet, das vor allem dazu diente, den Insider vom Ouotsider zu unterscheiden…

Am beunruhigendsten ist jedoch die Vorherrschaft der zweistufigen US-amerikanischen Denkweise, deren Grenzen und Gefahren wir in unserer **Studie zur sozialen Rückentwicklung** aufgezeigt haben.

Ein Manager beschrieb stolz die Reaktionsfähigkeit seines Teams und lobte dessen Fähigkeit, Fehler aus der Produktion in kürzester Zeit zu beheben.

Eine Tabelle sagt mehr als tausend Worte :

#	**Approche Agile**	**Approche "à la française[1]"**
1 – 1	Ein Fehler in der Produktion entdeckt wurde.	Der Endbenutzer ist nie in die ergonomische Suche nach Lösungen einbezogen (was gegen die französischen ergonomischen Grundsätze verstösst, die sich von den amerikanischen unterscheiden und eine solche Beteiligung verlangen)
2 – 2	Der Fehler innerhalb weniger Tage behoben wurde.	Die Aktivitätsergonomie kann ihre positiven Effekte nicht entfalten.
3 – 3	Champagne !! 	Die Merkmale der realen Arbeit werden nicht berücksichtigt.
| - 4		Die Testverfahren können die Realität der Arbeit des Benutzers nicht widerspiegeln.
| - 5		Das Endprodukt entspricht nicht dem, was der Benutzer braucht.
| - 6		Selbst wenn es mit grosser Professionalität entwickelt wurde, kann das Produkt die Mängel des Begleitverfahrens nicht überwinden und scheitert an der Aufgabe.
| - 7		Wäre es nicht besser gewesen, ein Werkzeug zu entwickeln, das an den realen Bedürfnissen der Benutzer näher kommt? Die Herstellung des Produkts wäre nicht teurer gewesen, hätte aber keine Diskrepanz zwischen den Erwartungen der Benutzer und dem Produkt gezeigt und wäre nicht an der Aufgabe gescheitert.
| - 8		Das Produkt wäre billiger gewesen – *da es sofort so entwickelt wurde, wie es die Benutzer erwartet haben* – und hätte weder zu Produktivitätsverlusten geführt – *die erste Folge des Fehlers in der Produktion* – noch den Return on Investment verschlechtert

Es ist offensichtlich, dass die Werkzeuge existieren, man muss sie nur einsetzen. Der erste Hinderungsgrund ist also nicht das Fehlen einer Lösung, sondern der Wille, sie umzusetzen.

Es ist offensichtlich, dass es nicht einfach ist, sich mit einem 60 Jahre alten Problem auseinanderzusetzen, das durch sein Alter dazu geführt hat, dass jedermann von der Unmöglichkeit seiner Beseitigung überzeugt ist. Aber die einzigen unlösbaren Probleme sind die, die man nicht angeht.

Es ist auch bemerkenswert, dass die Computerwelt es nicht geschafft hat, das Problem zu lösen, sondern die Art und Weise, wie man es betrachtet, so sehr verändert hat, dass man glauben könnte, es sei gelöst.

---

[1] Die französische Herangehensweise ist die theoretische Übung, die aus der möglichen Anwendung der auf der französischen Ergonomie basierenden Aktivitätsergonomie resultiert, wie sie vor zwei Jahrzehnten an technischen Hochschulen gelehrt wurde, und die Nutzung von Software, die mit Hilfe der, in der Fachliteratur beschriebenen, am Ende des letzten Jahrtausends entwickelten Werkzeuge, entwickelt wurde.
Wir möchten betonen, dass diese Werkzeuge nie in der industriellen Produktion eingesetzt wurden, sondern nur Labor-Kuriositäten geblieben wurden.

Hoffnung auf die Softwarekrise: Hochwertige App

Von Entwicklungsmodellen ist nichts zu erwarten, keines ist allein in der Lage, eine Lösung für die Qualitätskrise zu finden.

Die Antwort wird nur von der Projektleitung des IT-Projekts kommen, wenn es jemanden gibt, der es hören kann.

Was verlangt der Kunde, wenn er sich über die Softwarekrise beschwert?

Die Kundenreklamation ist klar:

- Die Software ist unpassend, sie entspricht nicht den Bedürfnissen.
- Sie enthalten zu viele Fehler, sie sind schwer zu warten.
- Sie sind zu teuer, Termine und Budget werden nicht eingehalten

Und was hat die IT-Welt von dieser Forderung verstanden?

Die IT-Welt war sicher überrascht, dass sie so abgetrennt wurde, denn wenn man nicht bewusst schlecht arbeitet, ist jeder überzeugt, sein Bestes zu geben. Für jeden ist es deshalb selbstverständlich, dass seine Arbeit gut ist.

Die erste Antwort, das Wasserfall-Modell, war richtig. Die Validierung der Arbeit am Ende jedes Schrittes führte zu einer Verbesserung der durchschnittlichen Qualität.

Das Hauptproblem der Kostenkontrolle blieb jedoch bestehen, da ein Fehler, der bei den abschliessenden Tests festgestellt wurde, die Lebensfähigkeit des Produkts gefährden könnte. Das Problem der Untauglichkeit blieb bestehen: Der Kunde konnte die Eignung des Produkts für seine Bedürfnisse erst überprüfen, wenn es fertig war, d.h. wenn das Entwicklungsbudget ausgeschöpft war.

Das später angewendete Modell V beseitigte zum Teil das Problem der späten Entdeckung von Fehlern während des abschliessenden Testverfahrens. Die Unit-Tests und dann die modularen Tests ermöglichten eine Befreiung von den eklatantesten Fehlern und eine gewisse Kostenkontrolle. Die Frage der Untauglichkeit blieb jedoch ungelöst, da der Kunde erst bei der Lieferung des fertigen Produkts die Möglichkeit hatte, die Mängel zu entdecken.

Durch die Einführung inkrementeller und iterativer Modelle konnte ein Teil dieser Probleme behoben werden. Bei diesen Modellen wird dem Kunden ein Modul ausgeliefert, sobald es fertig und getestet ist. Der Kunde erhält also seine Anwendung, die in einzelnen Teilen validiert werden muss. Wenn bei einem dieser Modelle ein Problem auftritt, kann die Korrektur innerhalb kürzester Zeit vorgenommen werden.

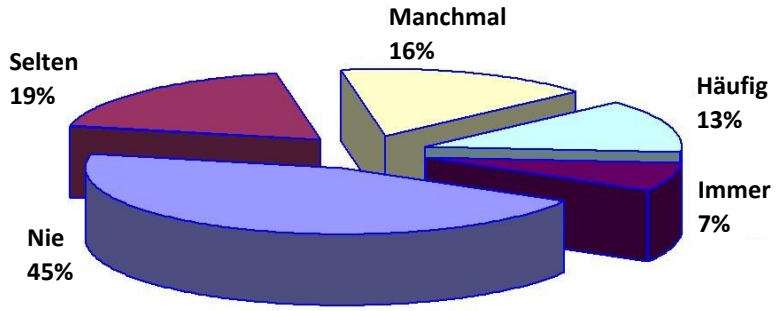
Das Problem der Unfähigkeit bekam endlich eine Antwort: Wenn ein Mangel festgestellt wurde, konnte dies im Rahmen der noch verfügbaren Budget korrigiert werden.

Aktueller Stand

Trotz aller Versprechungen der Befürworter dieser unterschiedlichen Methoden hat sich die Situation in der Softwarewelt seit der NATO-Konferenz 1968 kaum verbessert. Die Projekte werden immer noch schlecht gemanagt, die Budgets und Termine immer wieder überschritten, die Nutzerzufriedenheit und die Qualität immer noch schlecht.

Wie kann eine solch schwere Feststellung festgestellt werden, wenn die angewandten Methoden eine IT-Entwicklung ohne solche Abweichungen gewährleisten sollen?

Der von der Standish Group herausgegebene CHAOS Report von 2002 veröffentlichte eine sehr interessante Statistik:



Der Kreisdiagramm zeigt die Nutzungsrate der Funktionen durch die Nutzer einer App.

Wie man sieht, sind fast die Hälfte der entwickelten (und vom Kunden bezahlten) Features umsonst, da sie nie genutzt werden. Der Funktionsblock, der selten oder nie genutzt wird, macht zwei Drittel der Funktionen aus, während der Funktionsblock, der häufig oder immer genutzt wird, 20% der Gesamtmenge ausmacht.

Die Frage, welche Funktionalitäten nie genutzt werden, ist ein sehr wichtiges Problem. Die Studie der Standish Group befasst sich mit *Ad-hoc-Software*, die für den professionellen Einsatz entwickelt wurde. Das heisst, jede Funktionalität, die in einer solchen Software enthalten ist, wurde explizit gefordert und im Pflichtenheft definiert, das als Grundlage für die Entwicklung des Endprodukts diente.

Es ist also ein eklatanter Widerspruch, dass eine bestimmte Funktionalität benötigt, analysiert, entwickelt, bezahlt, aber nicht verwendet wird.

Verwendungshäufigkeit und Ausweisung

Eine erste Reaktion liegt darin, dass Features, die selten genutzt werden, schlecht beherrscht sind und hohe Aufmerksamkeit bei ihrer Nutzung erfordern. Folglich führt der hohe Arbeitsaufwand für die Implementierung solcher Features zu einer Verschlechterung der Produktivität.

Dies führt dazu, dass der Benutzer eine gewisse Abneigung gegen Funktionen entwickeln wird, die schwer zu bedienen sind und eine hohe Fehlerquote aufweisen.

Daher entwickelt der Anwender Vermeidungsstrategien, um sie nicht anwenden zu müssen. Und da er sie nicht nutzt, hat er keine Chance zu lernen, sie zu meistern.

Wir befinden uns in einem Teufelskreis:

- Wenn die Funktionalität schwer erlernbar ist, wird der Benutzer diese nicht benutzen.
- Wenn er es nicht benutzt, hat er keine Möglichkeit, seine Feinheiten zu beherrschen.
- Daher wurde das Feature vergeblich entwickelt und das Geld, das in dieses Feature investiert wurde, ist reiner Verlust. Eine Software, die solche Funktionen enthält, ist im Vergleich zu ihrer Verwendung zu teuer, die Rentabilität wird schlecht sein und sowohl die Zufriedenheit der Nutzer als auch die des Finanzierers wird gering sein.

Letzteres ist genau die Feststellung, die 1968 gemacht wurde, als die Softwarekrise zum ersten Mal angeprangert wurde und alle seither gelieferten Antworten das Ziel verfehlten, ungeachtet der tatsächlichen oder vermeintlichen Verdienste, mit denen ihre Befürworter sie befasst haben.

Ursache des Scheiterns von Weiterentwicklungsmethoden

Die bisher verwendeten Methoden haben alle in unterschiedlichem Ausmass beinahe versagt, da sie das grundlegende Merkmal von IT-Produkten nicht berücksichtigt haben: Sie sind alle einzigartig.

Anders als in der Industrie gibt es in der Computertechnik keinen Prototyp. Das programmierte Objekt ist genau das gleiche, das später beim Kunden verwendet wird. Es ist keine Kopie im industriellen Sinne, bei der zwei verschiedene Objekte aus einer Form herauskommen.

Je nach den Gefahren des eingespritzten Materials sind die Lebensdauer und die Widerstandsfähigkeit eines Industrieprodukts nicht identisch. Das erste Produkt kann nach zwei Tagen ausfallen, während das nächste Produkt dem Benutzer über viele Jahre hinweg einen qualitativ hochwertigen Service bietet.

Kopien der gleichen Software, die auf zwei verschiedenen Computer installiert sind, sind jedoch ein und dasselbe Objekt. Werden dieselben Manipulationen parallel ausgeführt, versagen beide Klone gleichzeitig und identisch.

In dieser Hinsicht kommt die Entwicklung einer Software einer künstlerischen Tätigkeit wie der Bildhauerei viel näher als einer industriellen Produktion. Die meisten bekannten Probleme sind auf die mangelnde Berücksichtigung dieses Merkmals zurückzuführen.

Verschärft wird das Missverständnis durch die industriellen Praktiken: Wenn die Industrie von einem Prototyp spricht, handelt es sich um ein Objekt, das aus den Konstruktionsbüros kommt und mit dem Endprodukt nichts gemein hat. Beispielsweise wurde der Prototyp Stück für Stück mit Werkzeugmaschinen bearbeitet oder mit einem Polymerharz-3D-Drucker hergestellt, während das Endprodukt in eine Form gespritzt wird.

Jede dieser Fertigungsmethoden erzeugt Objekte, deren Verhalten völlig unterschiedlich sein kann. Beispielsweise ist die Reaktion auf Spannungen nicht vergleichbar: Die Bearbeitung neigt dazu, Mikrorisse zu erzeugen, die als Bruchkerbe dienen können; der 3D-Druck hat ein eher isotropes Verhalten, während das Giessen in Abhängigkeit von der Giessgeschwindigkeit und den Erstarrungsbedingungen Spannungen im Werkstück hervorruft.

In der Computerwelt sollte ein Prototyp als Entwurf bezeichnet werden, da alles, was darin enthalten ist, im Endprodukt wiederzufinden wird; so wie die fertige Statue alle Mängel auf der Skizze enthält, kann die Entwicklung *quick and dirty* auf keinen Fall etwas anderes als ein *quick and dirty* Endprodukt ergeben.

In der Tat wird jede Komponente, die im Rohling entsteht, während der gesamten Produktentwicklung bis hin zum Vertrieb integriert bleiben, und jede Unvollkommenheit, die darin verbleibt, wird eine potenzielle Schwachstelle sein.

Diese Methode ist umso gefährlicher, da sie auf der Illusion beruht, dass der Code ständig überprüft wird, um Fehler zu finden. In der Realität wird ein schmutziger Code, der den Tests standhält, unverändert in das Endprodukt integriert, denn das Kriterium für die Einführung eines Codes in das Endprodukt ist nicht die Sauberkeit der Programmierung, sondern die Fähigkeit, die Tests zu bestehen.

Das Agile Manifest

Diese Achillesferse zeigt, dass alle Methoden, die sich auf das Agile Manifest berufen, nur qualitativ minderwertige Anwendungen erzeugen können. Diese Lücke zwischen iterativen und inkrementellen Methoden wird durch die finanziellen Gegebenheiten des wirtschaftlichen Umfelds noch verschärft. Würde die IT die Industrie tatsächlich kopieren, würde sie einen Prototyp des Produkts erstellen und den gesamten Code neu schreiben, wenn sie die endgültige Version zur Vermarktung schreiben würden.

Schon in finanzieller Hinsicht ist eine solche Methode unrealistisch, obwohl sie theoretisch in der Lage ist, einen Code zu garantieren, der fast alle Fehler frei ist.

Die Erklärung für das Scheitern der agilen Methoden liegt darin, dass sie davon in den Ausgang gehen, dass jeder Code während des Entwicklungsprozesses mindestens einmal revidiert wird. Infolgedessen steht niemand unter Qualitätsdruck, da jeder mit Recht davon ausgeht, dass der Aufwand zur Entdeckung möglicher Fehler auf einer späteren, implizit durch die Implementierung des agilen Prozesses vorgesehenen Kontrolle beruht.

Die Definition des Arbeitsrahmens ist von entscheidender Bedeutung, denn nur ein expliziter Rahmen auf der Grundlage klar definierter Verfahren kann zufriedenstellende Qualität gewährleisten. Wenn Qualität auf impliziten Verfahren beruht, wird sie von den Menschen abhängig, die diese Verfahren anhand ihrer Eigenheiten interpretieren.

Dadurch kann die Zuverlässigkeit des Systems nicht mehr gewährleistet werden, da jedes menschliche Versagen automatisch zum gesamten Entwicklungsprozess führt.

Eine Neudefinition des Systems

Folglich muss das System von Menschen unabhängig gemacht werden und sich nur auf die Verfahren stützen, die von den Menschen ausgeführt werden. Der Mensch verdrängt sich hinter dem Verfahren, das zum Träger der Qualität wird.

Ein solches System ist im Gegensatz zum Anschein sehr vorteilhaft für Menschen mit stark reduziertem Stresslevel. Die Stabilität des Systems wird dadurch verstärkt, dass sich die Mitarbeitenden mit aller Ruhe ihrer Aufgabe widmen können.

Da alle derzeitigen Methoden aufgrund der fehlenden Berücksichtigung des menschlichen Faktors keine zufriedenstellende Qualität erreichen, wird in den folgenden Kapiteln untersucht, welche Werkzeuge sich für diese Herausforderungen am besten eignen.

Rousseau und die Manager

Es scheint *so* zu sein, dass Mitarbeiter systematisch korrekt arbeiten; Fehler sind unglückliche Ereignisse, auf die sowohl das Management als auch die ausführenden Personen keinen Einfluss haben.

Die Angestellten werden von ihren Vorgesetzten intuitiv als eine Person wahrgenommen, die unwissentlich Fehler begeht, die jedoch aufgrund ihres guten Willens erziehend ist.

Die logische Konsequenz daraus ist, dass nichts unternommen wird, um Fehler in der Produktentwicklung zu verhindern.

Dieser industrielle Rousseauismus kann übersetzt werden mit: «*Der Angestellte wurde gut geboren, aber die Zwänge der Arbeit verdrehen ihn. Eine angemessene Ausbildung wird seine Seele retten und seine Produktivität steigern*.»

Im Gegensatz zu dieser Melasse guter Gefühle hat der Verhaltenalismus eine Position, die man als zynisch oder desillusioniert bezeichnen könnte: «*Der Angestellte arbeitet immer sein Bestes, aber er ist nicht in der Lage, Perfektion zu erreichen. Es gibt keine Methode, seine ursprüngliche Unvollkommenheit zu sublimieren. Nur eine Veränderung seiner Umwelt kann seine Mängel überwinden, um das Entstehen der gewünschten Eigenschaften zu ermöglichen*.»

Zwar wehren sich HR-Fachleute und die Hierarchie damit ab, dass sie viele sehr wirksame Qualitätsverbesserungsstrategien haben. Die Auswertung zeigt aber, dass diese Praktiken sowohl sportliche Trainingsvorgaben als auch militärische Trainingsübungen erfüllen und in gewissem Masse der Selbstbeeinflussung-Methode (Autosuggestion) zu verdanken sind.

Ihre sehr relativen und punktuellen Erfolge, gepaart mit der eklatanten Unfähigkeit, die Lehren aus diesen Erfolgen langfristig zu stabilisieren, zeugen von der Schwäche dieser Ansätze.

Wenn Informatiker Schafe wären

Wir schlagen vor, die Probleme im Zusammenhang mit der hierarchischen Struktur der IT-Entwicklung zu verdeutlichen, indem wir ein Unternehmen mit einer Schafherde vergleichen.

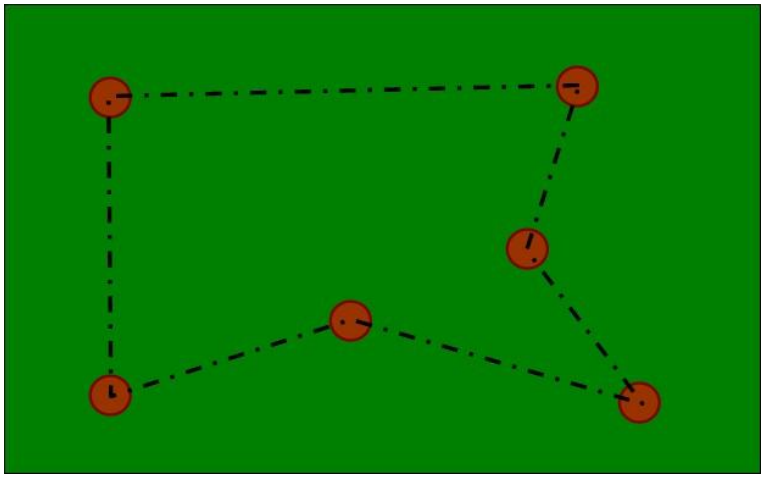
Dabei ist der Schäfer der Spiegel der Unternehmensführung. Die hierarchische Führung, vom Top- bis zum Low-Level-Management, wird durch die Hunde des Schäferhundes symbolisiert und die Produktivkräfte durch die Schafe, aus denen die Herde besteht.

Es wäre sinnlos, irgendeine Übereinstimmung zwischen dem Ruf, den diese Avatare normalerweise geniessen, und dem tatsächlichen Verhalten der so beschriebenen Personen zu suchen. Diese Studie ist keine Sozialsatire im Sinne der Fabeln von Esope oder Jean de la Fontaine.

Bei der Untersuchung geht es nicht um Tier oder Mensch, sondern um die Beziehungen zwischen den Beteiligten.

Der Hirte und seine Schafe

Wie in der Präambel erklärt, ist der IT-Anbieter wie ein Hirte, der seine Schafe über das Land seines Kunden transportieren muss. Es gibt Gebiete, in denen Schafe grasen können; und verbotene oder gefährliche Gebiete. Das Bild unten zeigt ein solches Gebiet.



Der Landowner befahl dem Schäfer, nur das Gras zwischen den Bäumen weiden zu lassen.

Der richtige Lieferant, der gute Schäfer, muss seine Angestellten, seine Schafe, dazu bringen, dem Kunden das für ihn geeignete Produkt zu liefern bzw. die erlaubten Flächen zu grasen, ohne den Rest zu berühren. Dabei wird dem Schäfer von seinen Hunden geholfen, genauso wie der Unternehmer ein Management, eine angeblich wirksame Hierarchie hat.

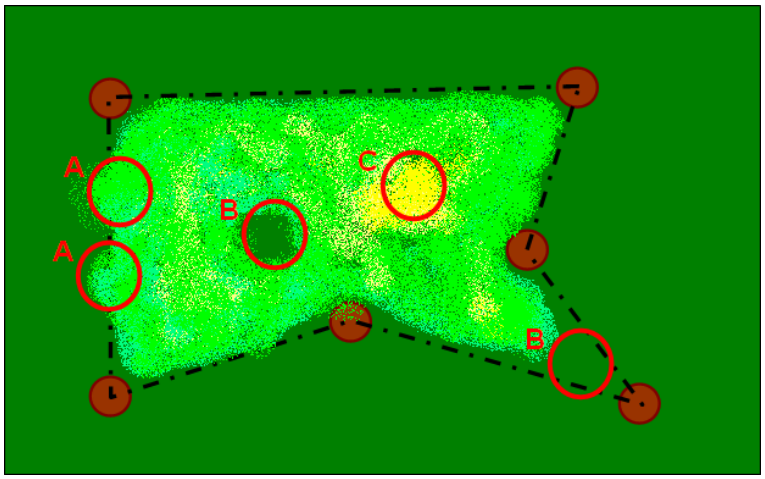
Durch langes, moduliertes Pfeifen wird der Hirte von seinen Hunden dazu gebracht, die Herde auf dem vereinbarten Gelände zu halten; so wie das Management Anweisungen und Dienstvorschriften herausgibt, damit die Mitarbeiter ordnungsgemäss arbeiten.

Doch hinter dieser idyllischen Vision steckt ein enormer Stress: Das Management setzt viel Energie auf, um das Projekt im Rahmen zu halten, genauso wie die Hunde in alle Richtungen rennen, um die Schafe im genehmigten Bereich zu halten.

Die Schafe geraten durch Pfeifen und Gebell unter Stress. Dies entspricht auch denjenigen, die an der Entwicklung eines Produkts beteiligt sind und ihren Arbeitsfortschritt pro Teammeeting begründen müssen.

Wenn die Arbeit getan ist

Nachdem die Schafe durchgereist sind, könnte das betreffende Gebiet so aussehen:

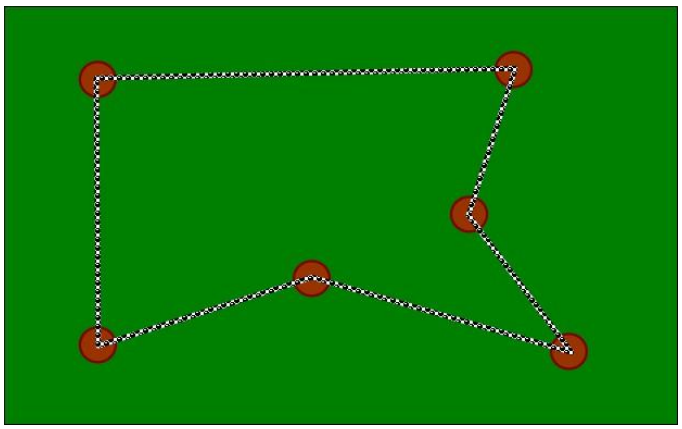


Bei genauer Betrachtung dieses Bildes stellen wir mehrere Probleme fest:

- In **A** haben die Schafe die erlaubte Grenze überschritten. Im Falle einer Software kann dies dazu führen, dass die Datenintegrität nicht gewährleistet ist oder eine Hintertür entsteht, die eine Sicherheitslücke öffnet.
- In **B** haben die Schafe einen Teil nicht abgegrast, der hätte abgegrast werden müssen. Bei einer Software kann dies dazu führen, dass die Funktion fehlt, obwohl sie im Pflichtenheft korrekt spezifiziert ist, das dem Lieferanten übergeben wird.
- In **C** haben die Schafe das Gras übermässig gemäht, was zu Nachwuchsproblemen führen wird; mit anderen Worten, es gibt einen Wertverlust des Grundstücks. Im Falle einer Software bedeutet dies, dass die Analysten zu viel spezifizieren oder die Funktionalität redundant ist. Dies führt dazu, dass der Kunde für die betreffende Funktion einen überhöhten Preis zahlen muss. Infolgedessen werden die Entwicklungskosten für das Endprodukt über das Optimum hinausgehen, was den Return on Investment umso verschlechtert.

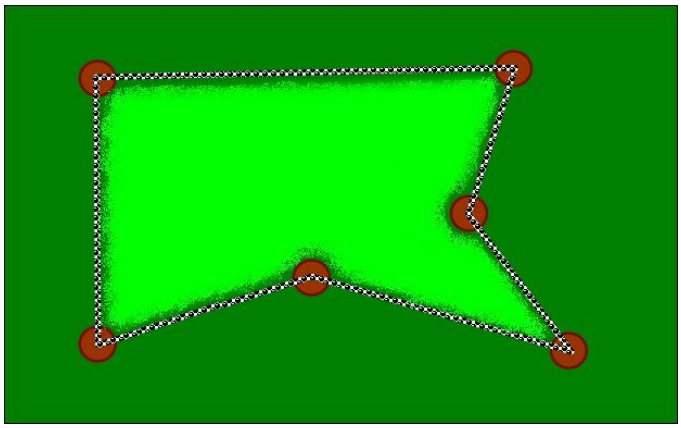
Eine einfache Lösung: der Zaun !

Der Hirte ist angesichts dieses Problems nicht ohne Ressourcen: Anstatt gestresst zu sein, das Verhalten der Schafe nicht wirksam kontrollieren zu können oder die Herde unter Stress zu sehen, wenn sie das erlaubte Gebiet verlassen, ist es viel einfacher, dieses mit einem Zaun abzugrenzen.



Während der Rousseauist Schäfer hofft, dass seine Schafe eines Tages gut genug erzogen werden, um selbst zu entscheiden, welche Teile zu grasen sind und welche nicht, weiss der verhaltensorientierte Schäfer, dass er seinen Schafen kein Vertrauen schenken kann, und er kann sich nicht vorstellen, diese Möglichkeit zu versuchen.

Er errichtet einen Zaun, weil er weiss, dass dies der einzig wirkungsvolle Weg ist, seine Schafe nur dort zu grasen, wo es erlaubt ist, mit oder ohne Hilfe von Hunden.



Die Produktivkräfte, ob Schafe oder Beschäftigte, werden den Stressniveau stark reduziert sehen, da sie nicht mehr einem übermässigen Druck durch die Führungskräfte ausgesetzt sind.

Im Gegensatz dazu wird sich in der Welt der Angestellten etwas ereignen, das in der pastoralen Welt keine Entsprechung hat: Die Einschliessung der Schafe wird in einen virtuellen Rahmen umgewandelt, der die Angestellten in ihre Verantwortung einbindet.

Externer Stress, der Druck der Führungskraft, wird interner Stress die Einhaltung der Pflichtenhefte ausgetauscht. Doch je stärker externer Stress als negativer Faktor empfunden werden kann, dem Mitarbeitende ausgesetzt sind, desto stärker kann interner Stress als positiver Faktor empfunden werden, denn die Mitarbeitenden wissen, dass sie diese Belastung im Griff haben.

Um diese Verantwortung nachzukommen, kann der Arbeitnehmer wählen, ob er sich freiwillig unterwerfen will. Zwang wird gegebenenfalls nicht mehr als Belastung, sondern als Motivation empfunden.

Dahinter verbirgt sich nichts Neues, die wissenschaftlichen Studien zur freiwilligen Submission sind fast 50 Jahre alt und die daraus resultierenden Empfehlungen haben ihre Stärke im Marketing bereits unter Beweis gestellt.

Dieser verhaltensorientierte Vorschlag ähnelt einem Fabry-Perot-Interferometer. Selbst ein sehr hochwertiger Laser kann keine vollkommen kohärente Strahlung emittieren. Er ähnelt einem Mitarbeiter, der sein Bestes gibt, aber nicht in der Lage ist, eine bestimmte Qualitätsgrenze zu überschreiten.

Um die Wellenlängengenauigkeit eines Laserstrahls zu erhöhen, wird dem Sender ein Interferometer hinzugefügt, das aus einer dünnen Klinge besteht, die mit halbreflektierenden Flächen bedeckt ist.

Diese Klinge erzeugt destruktive Interferenzen, die andere Wellenlängen als im gewünschten Bereich aufheben. Nur der Wellenlängenbereich, der mit der Dicke der optischen Scheibe resoniert, kann das optische Gerät durchqueren und der Strahl verlässt das Interferometer gereinigt von unerwünschten Frequenzen, wodurch eine bessere Qualität erreicht wird.

Genau das schlagen wir in diesem Vortrag vor: eine auf den Erkenntnissen der Verhaltenspsychologie basierende Herangehensweise an die Softwareentwicklung zu definieren, um die inhärenten Schwächen des Menschen zu eliminieren.

<u>Ein Bild ist mehr als tausend Wörter</u>

Wir haben gezeigt, dass ein Umfeld, das auf die Masse und nicht auf hypothetische Gurus zugeschnitten ist, viel bessere Erfolgschancen hat. Das obige Beispiel ist zwar nur ein Beispiel für das, was wünschenswert ist, aber es zeigt vor allem, dass man besser machen kann, wenn man die Situation richtig angeht und das Ziel richtig festlegt.

Besten Dank für die Kenntnisnahme.

Anhang Mitte April 2026

Ein vielversprechender Neuling?

Die Medien haben sich kürzlich über die Leistungen eines KI-Motors gefreut, der angeblich Tausende von Schwachstellen in Software entdeckt haben soll, Software die teilweise seit Jahrzehnten existieren und hatten den Ruf, sehr robust zu sein.

Hinter diesen Meldungen steckt keine «Rocket Science», der KI-Motor hat das getan, wofür er entwickelt wurde, nämlich nach Inkonsistenzen zu suchen, und wie jedes Computerprogramm hat er das sehr schnell und zuverlässig getan. Aber er konnte nicht über seine inhärenten Grenzen hinausgehen: Er ist ein Software, er wurde als solcher geboren und er bleibt ein solcher.

Eine Inkonsistenz zu finden ist eine Sache, sie zu beheben eine andere, und das kann der Software (noch) nicht.

Wie die Journalisten, die sich mit diesem Thema befasst haben, gerade aufgedeckt haben, gibt es zwei praktische Konsequenzen aus diesem Erfolg:

- Die erste ist, dass Unternehmen, die Zugang zu diesem Tool haben, nun die Möglichkeit haben, Schwachstellen zu finden und sie so schnell wie möglich zu beheben. Das ist eine «**White Hat**»-Strategie (der gute Hacker mit edlen Motiven.)
- Aber die Kehrseite ist ebenso gültig: Das Unternehmen, das den KI-Motor besitzt, hat den Zugang sorgfältig gesichert, um zu verhindern, dass er von «**Black Hats**» (den bösen Hackern mit finstersten Absichten) genutzt wird.

Aber was einmal gemacht wurde, kann kopiert werden und wenn der Zugang zu der einzigen derzeit funktionierenden Instanz derzeit gesperrt ist, wird dies nur ein weiterer Anreiz sein, einen anderen KI-Motor zu programmieren, der diese Funktionen nachahmt.

Wir sollten also in naher Zukunft einen Tsunami an Cyberkriminalität erleben.

Gibt es Hoffnung?

Hoffnung ist das letzte, was stirbt, solange wir nicht tot sind, gibt es immer Hoffnung !

Diese Hoffnung ist einfach das, was wir gerade beschrieben haben : mit geeigneten Überwachungstools ist es möglich, fehlerfreie Software zu erstellen, indem man die Bemühungen der Ingenieure und Architekten unterstützt, um potenzielle Schwachstellen zu vermeiden.

Wir haben bereits einen Demonstrator ausgeführt, der diese Realität voll und ganz bestätigt. Nicht nur, dass die Arbeit dadurch stark vereinfacht wurde, sondern auch die Entwicklungszeit konnten wir halbieren und die Kosten um ein Drittel gesenkt werden.

Wir müssen jedoch einen entscheidenden Punkt hervorheben: So wie es für eine KI möglich ist, einen Code zu überprüfen und Fehler zu finden, so ist es konzeptionell unmöglich, eine KI mit der Entwicklung eines fehlerfreien Programms zu beauftragen.

Die Erschaffung *ex nihilo* ist (und wird so noch lange bleiben) für ein automatisches Werkzeug unmöglich. Nur ein Mensch kann die Abstraktionsfähigkeiten besitzen, die für den Erfolg dieses Unterfangens notwendig sind.