Adaption, Umsetzung, Grenzen und Nutzen von Six Sigma in der Softwareentwicklung

Martin Mikusz, Georg Herzwurm

Lehrstuhl für ABWL und Wirtschaftsinformatik II
Universität Stuttgart
Breitscheidstraße 2c
70174 Stuttgart
herzwurm@wi.uni-stuttgar.de
mikusz@wi.uni-stuttgart.de

Abstract: Der Beitrag befasst sich mit der Umsetzung des aus dem industriellen Kontext stammenden Prozessverbesserungsansatzes Six Sigma in der Softwareentwicklung. Um den Rahmenbedingungen der Softwareentwicklung gerecht zu werden, muss Six Sigma größtenteils durch Analogien und bereits vorhandene Ansätze in der Softwareentwicklung adaptiert werden. Hierauf wird einzeln für jede Phase des Six Sigma-Vorgehensmodells zur Qualitätsverbesserung und Prozessoptimierung „DMAIC“ eingegangen. Anschließend werden die Grenzen und Potentiale eines Software-spezifischen Six Sigma aufgezeigt.

1 Six Sigma

Six Sigma wurde in den 80er-Jahren bei Motorola entwickelt und ist ein auf statistischen Methoden basierender Prozessverbesserungsansatz. Die publizierten Erfolge von Six Sigma, beginnend 1988 mit Motorolas Gewinn des *Malcolm Baldridge Quality Award*, bis hin zu dem umfassenden Six Sigma-Einsatz bei General Electric, trugen ebenso zu der großen Beachtung in der produzierenden Industrie bei, wie auch mehrere Management-Bestseller [HS00]. Zunächst bei hochvolumiger industrieller Produktion eingesetzt, anschließend in der im Six Sigma-Jargon sog. *Zweiten Welle* im Dienstleistungssektor, wird Six Sigma aktuell in der sog. *Dritten Welle* in die Softwareentwicklung eingebracht.

Six Sigma liegt die Qualitätsphilosophie von *G. Taguchi* zugrunde. Variationen innerhalb der Prozessergebnisse lassen sich direkt auf Veränderungen der Prozesselemente und -aktivitäten als steuerbare Parameter zurückführen. Das Hauptaugenmerk von Six Sigma liegt dabei auf der Reduzierung von Variation bei den qualitätskritischen Steuerparametern, die damit das Ergebnis besonders stark beeinflussen [RY05, S. 27f.].

Gute Prozesse liefern fortlaufend qualitativ hochwertige Ergebnisse, die kaum variieren. In einem Prozess, der das Niveau von 6 *Sigma* erfüllt, entstehen bezogen auf 1 Million Möglichkeiten nur 3,4 fehlerhafte Ergebnisse (*DPMO; defects per million opportunities*). Es liegen also nur 3,4 von einer Million Werten außerhalb der gesetzten unteren und oberen Toleranzgrenze i. H. v. (*Mittelwert+/- 6 Sigma* bzw. *Standardabweichungen*). Das Prozessergebnis variiert damit nahezu überhaupt nicht. Die Wahrscheinlichkeit, dass eine Produkteinheit bei 3,4 *DPMO* den gesamten Produktionsprozess fehlerfrei passiert, die sog. *rolled throughput yield (rty), beträgt* 99,9966%. Die *rty,* bzw. die hierin konvertierbaren Größen *DPMO* und der *Sigma-Wert* eines Prozesses, sind die zentralen Metriken bei Six Sigma [HS00].

Six Sigma-Projekte folgen einer formalisierten Vorgehensweise mit den Projektphasen *Define*, *Measure*, *Analyze*, *Improve* und *Control* (*DMAIC*). Wie die Mehrzahl aller Vorgehensmodelle zur Qualitätsverbesserung und Prozessoptimierung ist auch *DMAIC* an das von *Deming* eingeführte PDCA-Modell (Plan, Do, Check, Act) angelehnt. Den einzelnen Projektphasen des *DMAIC* sind phasenspezifische Methoden sowie Meilensteine mit definierten Teilnehmern und Zwischenergebnissen als Phasenübergänge zugeordnet (vgl. Abbildung 1).

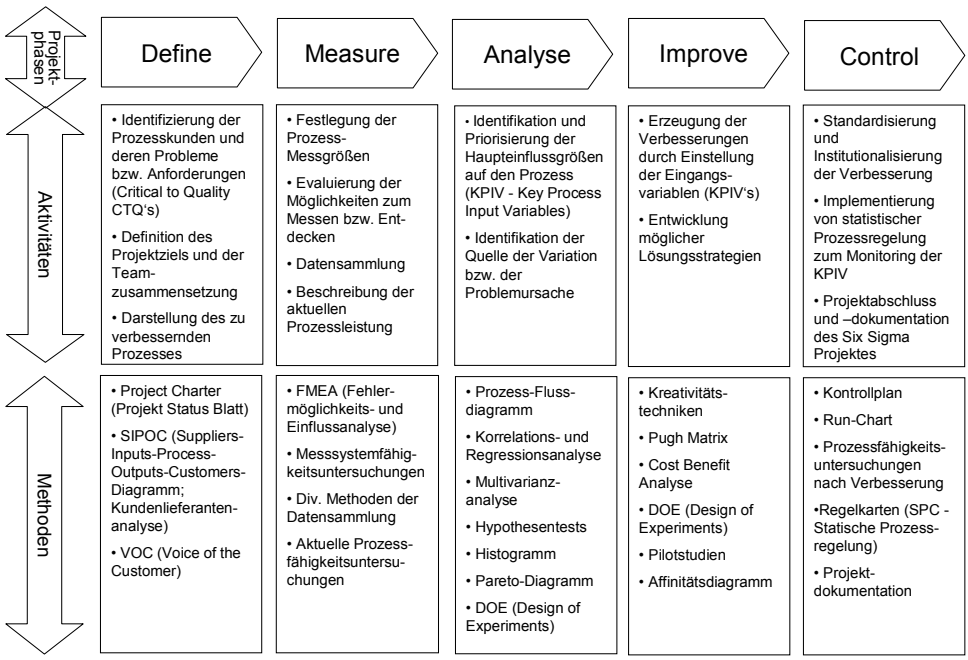

Abbildung 1: Six Sigma Projektphasen [RY05, S. 98]

Zum strikten Projektmanagement gehören des Weiteren definierte Rollen, die parallel zu ihrem ersten Six Sigma-Projekt eine entsprechende Ausbildung mit Schwerpunkt in statistischen Methoden und im Projektmanagement erfahren. Im Vordergrund stehen hier die sog. Black- und Green Belts. Black Belts übernehmen in Vollzeit die Rolle des Projektinitiators, -leiters und Change Managers. Sie werden durch die Green Belts bei der operativen Ausführung von Six Sigma-Projekten unterstützt. Die Dauer eines Six Sigma-Projektes wird üblicherweise auf ca. vier Monate ausgelegt.

Ursprünglich wurde von Motorola und General Electric ein vierstufiges Vorgehensmodell, das MAIC (Measure, Analyse, Improve, Control), entwickelt. Erst im Zuge der „Anreicherung" von Six Sigma wurde das *MAIC* um die *Define*-Phase erweitert, um so die Bedeutung des Projektziels, der Ressourcenplanung, und der eigentlichen Projektdefinition zu betonen. Die bedeutendste Rolle kommt jedoch der Datensammlung und -analyse bzw. der *Measure*- und *Analyse*-Phase zu. Alle Maßnahmen in einem Six Sigma Projekt sollen ausschließlich auf Basis von Daten bzw. Fakten festgelegt werden; Entscheidungen nach dem "Bauchgefühl" sind möglichst auszuschließen. Im Einzelnen bilden quantitative Daten

- die Grundlage für die Visualisierung des Ist-Prozesses und die Bestimmung seiner aktuellen Reife in der *Measure*-Phase, damit
- den Input für die *Analyse*-Phase, in der mit Hilfe der Messwerte die Problemursachen statistisch bestimmt werden, sowie damit
- die Grundlage für Entscheidungen bei der Auswahl der Lösungsalternativen in der *Improve*-Phase und für den Beweis deren Wirkung.

Six Sigma zielt also über den punktuellen Einsatz von statistischen Methoden hinaus auf die Entdeckung von Zusammenhängen zwischen den betrachteten Qualitätscharakteristika und Einflussfaktoren auf den Prozess. Verbesserungen werden aus diesen Beziehungen abgeleitet; vermutete Beziehungen werden gegen empirische Daten getestet, bevor sie als wahr akzeptiert werden. Der gesamte Problemlösungsprozess ist bei Six Sigma durch statistische Analysen (der Variation) als Erkenntnisprozess getrieben.

Zusammenfassend hebt sich Six Sigma v. a. durch drei Eigenschaften von konkurrierenden Ansätzen ab: a) rein datenorientierte Betrachtungsweise und auf statistischen Analysen beruhende Entscheidungsfindung, b) ein klar definiertes Rollenkonzept, sowie c) starke Projektfokussierung mit einer klar festgelegten Vorgehensweise bei der Lösungsfindung (*DMAIC*) [BA04]. Six Sigma ist jedoch keineswegs eine vollkommen neue Methodik und Denkweise. Elemente, die bereits früher vorhanden waren und in geschickter Weise mit den, ebenso wohlbekannten, statistischen Methoden als Six Sigmas methodischem Kern kombiniert wurden, sind Leadership-Ideen, Elemente von TQM wie die ausgesprochene Kunden- und Prozessorientierung, Teambildung, und der frühe Einbezug aller Betroffenen [Se03, S. 268].

2 Software-spezifische Umsetzung der Six Sigma-basierten Vorgehensweise *DMAIC*

2.1 Define-Phase

Hall et al. [HAB02] weisen in ihrer Sekundäranalyse von Erfolgsfaktorenuntersuchungen Merkmale erfolgreicher Prozessverbesserungen in der Softwareentwicklung nach. Interessanterweise bezieht sich keiner der Erfolgsfaktoren auf die konkreten (Verbesserungs-)Methoden, sondern ausschließlich auf die sog. *Soft Facts* (Management Commitment, Mitarbeiterbeteiligung, Kommunikation, Klarheit der Zielsetzung, u. a.). Gerade die Ziele und Aktivitäten der *Define*-Phase decken sich mit den wesentlichen Merkmalen erfolgreicher Prozessverbesserungen [HAB02]. So werden in der *Define*-Phase alle Prozesskunden bzw. Betroffenen identifiziert und mit einbezogen. Management Commitment wird durch die Nominierung eines Projekt-Champion aus dem Management in unmittelbarem Prozessumfeld sichergestellt. Die Festsetzung konkreter, von allen Beteiligten verstandener und akzeptierter Ziele ist Voraussetzung für den Übergang in die *Measure*-Phase. All diese Aktivitäten sind domänenneutral und bedürfen daher keiner Anpassung an die Besonderheiten der Softwareentwicklung. Insofern darf Six Sigma in der Softwareentwicklung nicht ausschließlich anhand der Anwendbarkeit der im Vordergrund stehenden statistischen Methoden bewertet werden.

2.2 Measure-Phase

Die Möglichkeiten zum Messen sind in der Softwareentwicklung nicht in dem Umfang gegeben, wie dies in der Massenproduktion der Fall ist. Die Datenqualität und -verfügbarkeit kann in der Softwareentwicklung als mangelhaft bezeichnet werden [Ka03, S. 470]. Oftmals ist kein direkter Zugang zur Messquelle gegeben, so dass auf Indikatoren, welche lediglich mit der Messquelle korrelieren, ausgewichen werden muss. Dadurch, dass in der Softwareentwicklung nicht nur Fehlerentstehung, sondern auch die Fehlerbeseitigung in allen Phasen stattfindet, kann zudem das Konzept der *rty* und der anderen sinngemäßen Metriken nicht direkt übernommen werden. Mit der *defect removal effectiveness* (*dre*) existiert aber bereits ein analoges Konzept in der Softwareentwicklung, das Aussagen über die Effizienz der Fehlerbeseitigung macht und dabei softwarespezifische Besonderheiten wie u. a. die neuerliche Fehlereinführung bei der Berichtigung berücksichtigt (vgl. Abbildung 2).

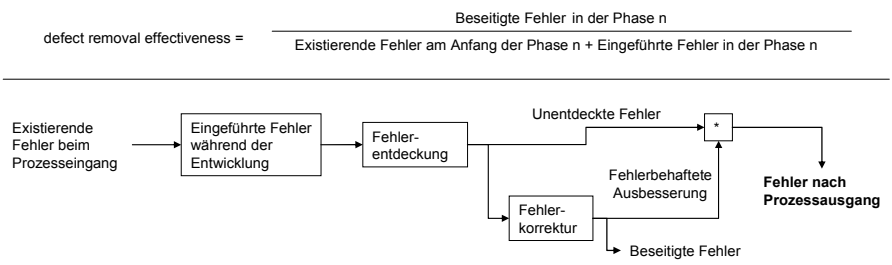

Abbildung 2: defect removal effectiveness [Ka03, S. 164f.]

Die *dre* ist ein ex post-Modell, da die eingeführten Fehler in der Phase *n* zum dortigen Zeitpunkt noch nicht bekannt sein können. Die Operationalisierung des Ansatzes erfolgt mit einer Matrix (Tool-Unterstützung ist zumeist Excel-basiert), in der Fehler der entsprechenden Phase, in der sie gefunden wurden, sowie der Phase, in der sie ihren Ursprung haben, zugeordnet werden. Basierend auf der Matrix und der Definition der *dre* können Effektivitätsmetriken für die einzelnen Phasen der Softwareentwicklung kalkuliert, über mehrere oder alle Phasen zu aggregierten Effektivitätsmetriken verdichtet [Ka03, S. 165ff.], und somit als Datenbasis für Six Sigma-Projekte herangezogen werden.

Einen solchen Ansatz verwirklicht in der Praxis Hallowell [Ha07]. Hier werden, in Analogie zu einem Reifegradmodell, aufeinander aufbauenden Six Sigma-Zielen die erforderlichen und ermöglichten Prozesse sowie, ausgehend vom zu *dre* analogen Konzept der *defect containment effectiveness,* die Implementierung der entsprechenden Metriken zugeordnet (vgl. Abbildung 3).

Six Sigma-Ziel	Erforderliche Prozesse	Ermöglichte Prozesse	Metriken
1. Reduzierung der Defekte (Released); Fokus auf Fehlerkorrektur	Modul-, Integrations-, Systemtest; Fehlerklassifizierung und Fehlerdatenbank-basierte Fehlerlösung	Auswertung von Fehlern vor vs. nach Release; Kausalanalyse	*Total Containment Effectiveness (TCE);* Fehler pro Fehlerklasse
2. Fehlerbeseitigung unmittelbar beim Ursprung	Software-Inspektionen	Analyse der Fehlereinführung und der Effizienz der Fehlerbeseitigung pro Phase	*Phase- und Defect Containment Effectiveness (PCE und DCE),* Fehler pro Modul pro Phase
3. Vorhersage der Fehlereinführung und der Effizienz der Fehlerbeseitigung vor und nach Release	Anwendung von Zuverlässigkeitsmodellen (bspw. Rayleigh)	Ermittlung des Inspektions- und Testaufwands; Schätzungen der Fehlerdichte des ausgelieferten Produktes	*Best fit Rayleigh Model* *Predictive Rayleigh Model*

Abbildung 3: Six Sigma als Metriken-Initiative basierend auf der *defect containment effectiveness* [Ha07]

2.3 Analyze-Phase

Die Prozesse der Softwareentwicklung sind unbeständig und teilweise undefiniert. Auch bei Verwendung eines Vorgehensmodells kann eine leichte Variation der kognitiven Prozesse zu unvorhersehbaren Ergebnissen führen. Besonders wenn die Leistung von Menschen unmittelbar über die Mitarbeiterqualifikation oder mittelbar über die Unternehmenskultur in den Prozess einfließt, müssen grundlegende Schwankungen als gegeben hingenommen werden. Eine zweite, wesentliche Quelle der Prozessvariation sind Produkteinflüsse. Produktcharakteristika wie bspw. Produktkomplexität sowie Produktanforderungen wie bspw. die geforderte Zuverlässigkeit haben ebenso signifikanten Einfluss auf die Prozessproduktivität und -ergebnisse. In der Summe ist die Variation des Softwareentwicklungsprozesses enorm hoch. Nach empirischen Erkenntnissen variiert für ein typisches Entwicklungsprojekt in der Größenordnung von 80.000 LOC (Lines of Code) die Entwicklungszeit zwischen 3 und 100 Monaten, der Aufwand zwischen 8 und 2000 Mann-Monaten und die Zahl der gefundenen Fehler zwischen 10 und 4.000 [PM97, S. 30 und 35]. Für die Anwendung von Six Sigma in der Softwareentwicklung ergeben sich hieraus Konsequenzen und ein Bedarf zur Anpassung der Methodik.

Da statistische Verfahren bestimmte Anforderungen an die Qualität von Auswertungsdaten stellen, steht bei einem Six Sigma-Projekt in der Softwareentwicklung nicht das vollständige statistische Methodeninstrumentarium der *Analyze*-Phase zur Verfügung. Bspw. ist die bei Six Sigma sehr wichtige Methode *DoE* (*Design of Experiments*) in der Softwareentwicklung nur bedingt anwendbar. Denn ein exaktes Messen wie bei physischen Produkteigenschaften in der Massenproduktion und anschließendes Optimieren über eine exakt bestimmbare Faktoreinsatzkombination ist hier nicht realistisch.

Auf der Methodenebene sprechen Magnusson et al. [MKB04] jedoch von insgesamt über 450 potentiellen Six Sigma-Methoden, wobei zu beachten ist, dass diese für sich genommen nicht Six Sigma-spezifisch sind. Die Herausforderung liegt daher in der Verwendung der für die Domäne der Softwareentwicklung und ihrer Besonderheiten zielführenden Methoden. Aufgrund der enormen Variation und der dadurch teilweise fehlenden direkten Korrelation zwischen Prozessinput (Eingangsvariablen) und -output (Produktqualität bzw. Prozessproduktivität) [Ka03, S.143] ist der mögliche Erkenntnisgewinn durch statistische Analysen der Variation in der Softwareentwicklung begrenzt. Sie führen nicht unmittelbar zum Verständnis des Problems und der Ursache-Wirkungszusammenhänge. Daher muss der Fokus bei der Auswahl der richtigen Methoden in der Softwareentwicklung auch auf qualitative Beobachtungs- und Analysetechniken, wie bspw. Post Mortem-Analysen abgeschlossener Projekte, erweitert werden.

2.4 Improve-Phase

Der kognitive Charakter des Softwareentwicklungsprozesses und seine enorme Variation führen in der *Improve*-Phase zu Konsequenzen analog zur *Analyze*-Phase. Denn nicht nur die Problemursachen, sondern auch die Effekte der Prozessverbesserung, werden durch die Produkt- und Mitarbeitereinflüsse überdeckt. So ist bspw. eine moderate Erhöhung der Testeffizienz durch den Einsatz von CASE-Tools nur schwer nachweisbar, da überlagernde Produktivitätsschwankungen durch die Mitarbeiterqualifikation den Faktor 10 erreichen können. Die Effekte der Prozessverbesserung müssen daher normiert werden. Die anforderungsneutrale Definition der Prozesseffizienz durch den Aufwand pro *Function Point* [BF04, S. 78] ist hier ein probates Mittel, um Prozessverbesserungen in der Softwareentwicklung losgelöst von der Produktkomplexität messbar zu machen.

Six Sigma bietet grundsätzlich keine konkreten Lösungsvorschläge oder Ansatzpunkte für die Prozessverbesserung, sondern versteht sich als ein Ansatz, der allgemeine Problemlösungskompetenz zur Lösung sehr verschiedener inhaltsspezifischer Probleme bietet. Verbesserungen resultieren daher neben dem Optimieren der Eingangsvariabeln auf den Prozess i. e. S. (als ein mathematisches bzw. statistisches Problem) mit Hilfe von *DoE* oder durch Änderungen von Prozesstoleranzen, auch aus der Anwendung von Kreativitätstechniken. Während das Optimieren bei kognitiven Prozessen ausscheidet, ist die Anwendung der von Six Sigma bereitgestellten Kreativitätstechniken domänenneutral.

Six Sigma-Projekte versuchen die Probleme aus möglichst vielen Perspektiven zu betrachten, um sie optimal erfassen zu können. Hierfür kann das Projektteam temporär in der *Improve*-Phase um Spezialisten aus dem unmittelbaren Prozessumfeld erweitert werden. Die Lösungsgenerierung erfolgt dabei systematisch mit Hilfe von Kreativitätstechniken wie bspw. Brainstorming. Kreativitätstechniken wenden heuristische Prinzipien in formalisierter Form an. Die Prinzipien Assoziieren und Strukturieren fördern das intuitive Hervorbringen von Ideen, während Variieren, Kombinieren und Abstrahieren durch systematisch analytisches Vorgehen zu neuen Ansätzen führen [Ba98, S. 172f.].

Die *Improve*-Phase von Six Sigma in der Softwareentwicklung ist nicht scharf vom sog. *Design for Six Sigma*-Ansatz (*DFSS*) abzugrenzen. Als *DFSS*(-Prozesse) werden generische Produktentwicklungsprozesse bezeichnet, die den Anspruch haben, kundenorientierte, innovative und fehlerfreie Produkte zu entwickeln [YE03]. Dabei spielt die Methode *Quality Function Deployment* (*QFD*) [HSM00] eine wesentliche Rolle. Es sind Entwicklungsprozesse im Sinne der Philosophie von Six Sigma. Gegenstand von Verbesserungen bei Six Sigma in der Softwareentwicklung ist überwiegend die Einführung von ebensolchen Verbesserungen, die der Philosophie von Six Sigma entsprechen. In der Praxis sind dies:

- Konzepte, die zur datenbasierten Entscheidungsfindung im Entwicklungsprozess führen: Aufwandschätzung, Risikomanagement, RoI, etc. [DC02; Da92];
- Konzepte, die zu weniger Fehlern führen: Inspektionen, Zuverlässigkeitsmodelle, *dre*-Metriken (vgl. Abbildung 3) [Ha07];
- Konzepte, die zur Erhöhung der Kundenzufriedenheit führen: QFD, Kosten-Nutzen-basierte Anforderungspriorisierung [Fe05; DC03; HE00].

2.5 Control-Phase

Ziel der Statistischen Prozessregelung ist es, aus Vergangenheitsdaten die zukünftige Variabilität des Prozesses vorherzusagen, um so präventiv eingreifen zu können, bevor der Prozess mit hoher Wahrscheinlichkeit Ausschuss produzieren würde. Basis dafür bildet die Qualitätsregelkartentechnik, mit der die Stichprobenentnahme im Zeitverlauf visualisiert wird und damit rechtzeitig Unregelmäßigkeiten in den Messdatentrends erkannt werden können [Zo01, S. 1101ff.]. Es wird also im Prozessschritt *n* von der Qualität der letzten baugleichen *x-1*, *x-2*, etc. Zwischenprodukte auf die Qualität des Zwischenproduktes *x* geschlossen. Häuften sich bspw. systematisch nach jedem 1000. Werkstück die Fehler im Prozessschritt *n*, ist es plausibel anzunehmen, dass sich dieses Muster fortsetzen wird. Dabei spielt es weder eine Rolle, welche Prozessschritte *n-1*, *n-2* etc. durchlaufen wurden, noch wie dies der Fall war.

Da in der Softwareentwicklung diese Schlussfolgerung v. a. aufgrund der Heterogenität der Artefakte (bspw. Produktkomplexität) und der Nichtbeachtung des bisherigen Entwicklungsverlaufes bez. Prozessqualität nicht zwingend sinnvoll ist, bietet es sich an, das Konzept wiederum auf Analogien zu vorhandenen Ansätzen der Softwareentwicklung zu untersuchen. Statistische Prozessregelung wird zwar in der dargestellten Form in der Softwareentwicklung thematisiert. Doch sie wurde für die Massenproduktion entwickelt und basiert damit auf dortigen Gesetzesmäßigkeiten, die in der Softwareentwicklung so nicht gegeben sind. Allen voran ist Softwareentwicklung nicht Massenproduktion und nimmt im Gegensatz zu dieser einen relativ langen Zeitraum sowie mehrere Projektphasen in Anspruch.

Daher ist hier konzeptionell ein Lebenszykluskonzept zweckmäßiger als Qualitätsregelkarten, die auf sequentiellen Daten basieren [Ka03, S. 144]. Bei diesem ist die Schlussfolgerung eine in der Softwareentwicklung ungleich sinnvollere, nämlich vom bisherigen Entwicklungsverlauf (bspw. Zeitdruck, Change History, *dre*, etc.) in den Entwicklungsphasen *n-1*, *n-2*, etc. des Moduls *x* auf seine Qualität in der Entwicklungsphase *n*. Wurde bspw. ein komplexes Modul unter hohem Zeitdruck, vielen Änderungsanforderungen, sowie einer hohen Personalfluktuation entwickelt, ist es plausibel anzunehmen, dass in der Phase *n* (bspw. Modultest) viele Fehler gefunden werden.

Solche auf dem Lebenszykluskonzept basierenden Modelle existieren in der Softwareentwicklung und werden als Zuverlässigkeitsmodelle bezeichnet. Six Sigma Ansätze in der Softwareentwicklung adaptieren diese bereits (vgl. Abbildung 3). Die Fehlerdichte in der Phase *n* ist hier eine Funktion der Produkteigenschaften, des bisherigen Projektverlaufs und eben der aktuellen Projektphase.

Zuverlässigkeitsmodelle bedürfen einer zeit- und kostenintensiven Einführung bzw. Justierung im Unternehmen, um verlässliche Vorhersagen treffen zu können. Beispielhaft sei hier das Zuverlässigkeitsmodell von *Rayleigh* genannt. Es kann zur Ermittlung des Inspektions- und Testaufwands bei einem gegebenen Qualitätsziel und zur Schätzungen der Fehlerdichte des ausgelieferten Produktes eingesetzt werden [Ka03, S. 187ff.].

3 Nutzen und Grenzen von Six Sigma in der Softwareentwicklung

Bei der Gestaltung der Softwareprozessverbesserung sind wie bei jedem anderen Gestaltungsvorgang diverse Rahmenbedingungen zu beachten. Sie stellen zum einen Anforderungen an die Gestaltungsmaßnahmen und üben zum anderen Einfluss auf die Wirkung dieser Gestaltungsmaßnahmen aus [Me04, S. 39ff.].

Kap. 2 verdeutlichte, dass aufgrund der besonderen Rahmenbedingungen in der Softwareentwicklung die direkte Übertragbarkeit von Six Sigma aus dem industriellen Kontext nicht bei allen Methoden und Phasen von Six Sigma zielführend ist. Deren Umsetzung muss vielmehr durch die aufgezeigten Analogien und bereits vorhandenen Ansätze in der Softwareentwicklung erfolgen. Grundsätzlich können erst unter dieser Voraussetzung die tatsächlichen Nutzen, Grenzen und Potentiale von Six Sigma in der Softwareentwicklung aufgezeigt werden. Im Einzelfall ist die erfolgreiche Umsetzung und ggf. weitere Anpassung von Six Sigma jedoch von weiteren Rahmenbedingungen abhängig, nämlich von der

- Problemart, vom
- Reifegrad des Softwareentwicklungsprozesses, sowie von der
- Unternehmens- und Problemlösungskultur.

3.1 Problemart als Rahmenbedingung

Die systematische Lösung von Problemen lässt sich in die beiden Phasen Ursache- und Lösungsanalyse aufteilen. Beide Phasen sind vollständig durch das *DMAIC* abgedeckt. Sie müssen jedoch nur bei Problemen i. e. S. durchlaufen werden, d. h. wenn weder die Ursache noch die Lösung des Problems bekannt sind. Ist die Ursache bekannt oder deren Kenntnis nicht notwendig, entfällt die Ursachenanalyse (*Measure* und *Analyze*). Ist zudem gar die Lösung bekannt bzw. vorab beschlossen, kann auch die Lösungsanalyse (*Improve*) entfallen. Denn bei einer offensichtlichen und allen Beteiligten bekannten Ursache erübrigt sich deren Analyse. Analog erübrigt sich eine aufwendige Lösungsanalyse, wenn die Lösung bereits feststeht.

Bei Problemen mit bekannter oder nicht wissenswerter Ursache ist daher ein relativ aufwändiges Six Sigma-Projekt nicht gerechtfertigt. Es bietet es sich daher an, ein lediglich als ein Projektmanagementansatz für Softwareprozessverbesserungen verstandenes Six Sigma anzuwenden. Hierunter fällt v. a. die *Define*-Phase. Somit kann eine zielgerichtete Einführung von vorab festgelegten Verbesserungen in einem projektorganisatorischen Rahmen erfolgen, der den Merkmalen erfolgreicher Prozessverbesserungen in der Softwareentwicklung entspricht (vgl. Kap. 2 zu *Define*).

Das gesamte Potential von Six Sigma kann also nur bei Problemen i. e. S. ausgeschöpft werden. Hier kann Six Sigma in seinem ursprünglichen Sinn als ein systematischer Problemlösungsansatz mit seinem gesamten methodischen Kern, der Problem- und Lösungsanalyse mit statistischen Methoden, Anwendung finden. Voraussetzung für den Erfolg ist jedoch neben der methodischen Anpassung an die Besonderheiten der Softwareentwicklung (vgl. Kapitel 2) eine Problemstellung, die für statistische Methoden zugänglich ist. Bedauerlicherweise gilt dies nur bedingt für die drei bedeutendsten Einflussfaktoren auf die Produktivität des Softwareenwicklungsprozesses, nämlich für die Bereiche Unternehmenskultur, Qualifikation der Mitarbeiter, sowie den Grad der Wiederverwendung. Die Reduzierung von Variation bei diesen qualitätskritischen Steuerparametern ist zwar prinzipiell möglich - bei der Mitarbeiterqualifikation bspw. über eine standardisierte und qualitativ hochwertige Ausbildung. Doch unternehmenskulturellen Problemen, Defiziten in der Mitarbeiterqualifikation sowie mangelnder Wiederverwendung kann nur begrenzt mit statistischen Methoden beigekommen werden.

3.2 Prozessreifegrad der Softwareentwicklung als Rahmenbedingung

Das *CMMI* unterscheidet fünf Reifestufen des Softwareentwicklungsprozesses. Jede Reifestufe zeichnet sich durch bestimmte Prozessfähigkeiten aus, wobei die Stufen geordnet sind und aufeinander aufbauen. Auf Stufe 3 (*Defined*) existiert ein unternehmensweit gültiger Softwareentwicklungsprozesses, der dokumentiert und standardisiert ist. Auf Stufe 4 (*Managed*) sind quantitative Ziele für Produkte und Prozesse vorhanden; die Qualität wird gemessen. Der Einfluss von verwendeten Maßnahmen auf Produktqualität sowie Produktivität ist verstanden, sodass Zielabweichungen mit Hilfe vorhandener Daten erklärt werden können. Auf der *CMMI*-Stufe 5 (*Optimizing*) ist das Unternehmen auf kontinuierliche Prozessverbesserung ausgerichtet. Erfahrungen werden mit quantitativen Daten beschrieben sowie neue Methoden zunächst in Pilotanwendungen erprobt. Anschließend wird ihre Eignung auf Basis quantitativer (Kosten-Nutzen-)Analysen festgestellt [Me04, S. 331f.].

Offensichtlich bestehen zwischen *CMMI* und Six Sigma Berührungspunkte. *CMMI*-Stufe 5 kann durch die Verwendung von Six Sigma umgesetzt werden. *CMMI*-Stufe 4 beschreibt treffend wesentliche Bestandteile der Six Sigma-Philosophie, aber gleichzeitig auch wesentliche implizite Voraussetzungen für die erfolgreiche Anwendung von Six Sigma. Das CMMI als Reifegradmodell geht davon aus, dass eine bestimmte Stufe nicht ohne Prozessfähigkeiten erreicht werden kann (und darf), die den darunter liegenden Stufen zugeordnet werden. Insofern muss auch für die erfolgreiche Anwendung von Six Sigma ein gewisser Reifegrad des Softwareentwicklungsprozesses vorausgesetzt werden. Fehlen bspw. wohl definierte und auf Kundenanforderungen basierte Metriken, kann die Softwareprozessverbesserung mit Six Sigma mit einer aufwendigen Metriken-Initiative einhergehen.

So berichtet Daskalantonakis [Da92] von einem langwierigen Prozess der Schaffung einer Softwaremetrik-Infrastruktur im Zuge der flächendeckenden Anwendung von Six Sigma bei Motorola. Neben der eigentlichen Definition der Metriken, die in Arbeitsgruppen mit Hilfe der *Goal Question Metric*-Methode (*GQM*) abgeleitet wurden, umfasste der Prozess entsprechende Trainingsmaßnahmen, die Einführung von Tools sowie Beraterunterstützung bei der Implementierung der Metriken [Da92, S. 107f.].

3.3 Unternehmens- und Problemlösungskultur als Rahmenbedingung

Eine breite Akzeptanz von Six Sigma im Softwareunternehmen setzt voraus, dass die Softwareentwicklung als eine Ingenieurwissenschaft angesehen wird und nicht als eine vorwiegend künstlerisch-kreative Tätigkeit. Eine Methode in einer Ingenieurwissenschaft wird stets an ihrer Nützlichkeit gemessen, welche sich in den meisten Fällen ausschließlich empirisch nachweisen lässt [Pr01, S. 30]. Six Sigma ist in diesem Punkt rigoros, sodass im Softwareunternehmen vor den Resultaten einer empirischen Bewertung bestehender sowie neu eingeführter Methoden nicht zurückgewichen werden darf. Damit ist Six Sigma gleichzeitig auch wissenschaftlich akzeptabel. Denn sowohl bei Six Sigma als einem Praxisansatz, als auch bei der empirischen Softwaretechnik als einem wissenschaftlichen Ansatz, stützt sich der Wissensgewinn auf die tatsächliche und quantifizierte Wirkung von (Entwicklungs-)Methoden. Im Gegensatz dazu steht die spekulative Bewertung, die auf Basis mehr oder weniger plausibler und größtenteils unausgesprochener Annahmen die erwarteten Eigenschaften ohne Empirie herleitet [Pr01, S. 30]. Insofern steht Six Sigma vor denselben Herausforderungen auf dem Weg zu einer messwertgesteuerten Softwareentwicklung wie die empirische Softwaretechnik.

Auf der anderen Seite können an einer mangelnden Datenverfügbarkeit und -qualität leidende Six Sigma-Projekte die Metrikproblematik im Unternehmen unmissverständlich aufzeigen und sukzessiv aus den Projekten heraus Metriken institutionalisieren. Die rationale Vorgehensweise bei der Entscheidungsfindung in einem Six Sigma-Projekt ist bei allen Entscheidungen im Softwareentwicklungsprozess wünschenswert. Six Sigma trägt zu einer Abkehr von der häufig intuitiven, subjektiven, und im Nachhinein nicht nachvollziehbaren Entscheidungsfindung, zu einem systematischen und rationalen Vorgehen bei der Bewältigung von Problemen und Entscheidungen im gesamten Softwareentwicklungsprozess bei.

4 Zusammenfassung

Um den Rahmenbedingungen in der Softwareentwicklung gerecht zu werden, muss Six Sigma größtenteils durch Analogien und bereits vorhandene Ansätze in der Softwareentwicklung adaptiert werden. So bezeichnet bspw. die SAP ihr derart abgeleitetes Methodenset als SAP Sigma; Sun´s Ansatz lautet Sun Sigma. Resultierende Verbesserungen durch Six Sigma in der Softwareentwicklung liegen u. a. in den Bereichen Aufwandschätzung, Metriken-basierte Entscheidungsfindung, Fehlervermeidung und Erhöhung der Kundenzufriedenheit.

Im Einzelfall ist die erfolgreiche Umsetzung und ggf. weitere Anpassung von Six Sigma von zusätzlichen Rahmenbedingungen abhängig – v. a. muss das ursächliche Problem für statistische Methoden zugänglich sein und es muss bereits ein gewisser Prozessreifegrad mit einer Softwaremetrik-Infrastruktur vorhanden sein. Auf der anderen Seite können an einer mangelnden Datenverfügbarkeit leidende Six Sigma-Projekte die Metrikproblematik aufzeigen und sukzessiv aus den Projekten heraus Metriken institutionalisieren. Insgesamt trägt Six Sigma zu einer Abkehr von der häufig intuitiven, subjektiven, und im Nachhinein nicht nachvollziehbaren Entscheidungsfindung, zu einem systematischen und rationalen Vorgehen bei der Bewältigung von Problemen und Entscheidungen im gesamten Softwareentwicklungsprozess bei.

Literaturverzeichnis

[Ba98] Balzert, H.: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Band 2. Spektrum, Heidelberg 1998.

[Ba04] Basu, R.: Six-sigma to operational excellence: role of tools and techniques. In: International Journal of Six Sigma and Competitive Advantage 1 (2004) 1, S. 44–64.

[BF04] Bundschuh, M.; Fabry, A.: Aufwandschätzung von IT-Projekten. 2. Aufl., mitp-Verlag, Bonn 2004.

[Da92] Daskalantonakis, M.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. In: IEEE Trans. Software Eng. 18 (1992) 11, S. 998-1010.

[DC03] Denne, M.; Cleland-Huang, J.: Software by Numbers. Prentice Hall, Upper Saddle River 2003.

[Fe05] Fehlmann, T.: Six Sigma in der SW-Entwicklung. Vieweg, Wiesbaden 2005.

[FS07] Fröschle, H.-P.; Strahringer, S. (Hrsg.): IT-Industrialisierung, HMD-Praxis der Wirtschaftsinformatik, Heft 256. dpunkt Verlag, Heidelberg 2007.

[HAB02] Hall, T.; Austen, R.; Badoo, N.: Implementing Software Process Improvement: An empirical Study. In: Software Process Improvement and Practice (2002) 7, S. 3-15.

[Ha07] Hallowell, D.: Six Sigma Software Metrics, Part 2, iSixSigma LLC, 2007, abgerufen am 06.04.2007: http://software.isixsigma.com/library/content/c031001a.asp.

[HS00] Harry, M.; Schroeder, R.: Six sigma: the breakthrough management strategy revolutionizing the world's top corporations. Doubleday, New York 2000.

[HSM00] Herzwurm, G.; Schockert, S.; Mellis, W.: Joint requirements engineering - QFD for rapid customer-focused software and Internet-development. Vieweg, Wiesbaden 2000.

[Ka03] Kan, S. H.: Metrics and Models in Software Quality Engineering. 2. Aufl., Addison-Wesley, Upper Saddle River 2003.

[MKB04] Magnusson, K.; Kroslid, D.; Bergman, B.: Six Sigma umsetzen. 2. Aufl., Hanser, München 2004.

[Me04] Mellis, W.: Projektmanagement der SW-Entwicklung. Vieweg, Wiesbaden 2004.

[Pr01] Prechelt, L.: Kontrollierte Experimente in der Softwaretechnik. Springer, Berlin 2001.

[PM97] Putnam, L. H.; Myers, W.: Industrial Strength Software: Effective Management Using Measurement. IEEE Computer Society Press, Washington 1997.

[RY05] Rehbehn, R.; Yurdakul, Z. B.: Mit Six Sigma zu Business Excellence. 2. Aufl., Publics Corporate Publishing, Erlangen 2005.

[Se03] Seghezzi, H. D.: Integriertes Qualitätsmanagement: das St. Galler Konzept. 2. Aufl., Hanser, München 2003.

[YE03] Yang, K.; El-Haik, B.: Design for Six Sigma: a roadmap for product development. McGraw-Hill, New York 2003.

[Zo01] Zollondz, H.-D.: Lexikon Qualitätsmanagement. Oldenbourg, München 2001