# Softwareprozessmetriken und agile Methoden – eine industrielle Fallstudie

Stefanie Löper
T-Systems Multimedia Solutions GmbH Berlin
Wittestraße 30N, 13509 Berlin
E-Mail: stefanie.loeper@t-systems.com

Gabriele Schmidt
Fachhochschule Brandenburg, Fachbereich Informatik und Medien
Magdeburger Straße 50, 14770 Brandenburg an der Havel
E-Mail: gabriele.schmidt@fh-brandenburg.de

**Abstrakt:** Vor einigen Jahren sind im Bereich der Softwareentwicklung agile Softwareentwicklungsmethoden entworfen worden, welche u.a. für den Entwicklungsprozess schnellere Entwicklungszeiten und Flexibilität als Vorteile versprechen und damit einen Ansatz darstellen, die Nachteile traditioneller Entwicklungsmethoden zu überwinden. Softwaremetriken definieren Kenngrößen eines Software-Produktes und/oder –Prozesses, die gemessen werden. Sie werden als integraler Bestandteil und Stärke der traditionellen Softwareentwicklungsmethoden angesehen. Die Anwendung von Prozessmetriken auf agile Softwareentwicklungsmethoden ist bis zum heutigen Zeitpunkt nur geringfügig untersucht. Es existieren einige Prozessmetriken für agile Methoden. Inwieweit diese Metriken Aussagen über die Effizienz agiler Softwareentwicklungsmethoden treffen und sich daraus Optimierungspotentiale bei der Methode ableiten lassen, wird in einer industriellen Fallstudie untersucht. Dabei werden empirische Daten präsentiert und analysiert, die auf der Grundlage von ausgewählten Prozessmetriken gesammelt wurden.

## 1 Motivation und Einleitung

Zu Beginn des 21. Jahrhunderts, als der Begriff Agilität geprägt wurde, haben viele Anwender und Benutzer von agilen Entwicklungsmethoden festgestellt, dass agile Softwareentwicklungsmethoden viel schneller einen Geschäftsnutzen liefern, der die Effizienz und Qualität verbessert und die Teammoral steigert [Ba04].

Im Fokus der folgenden Arbeit steht der agile Softwareentwicklungsprozess. Dabei bilden geeignete Softwaremetriken, im Folgenden als Prozessmetriken bezeichnet, die Grundlage für folgende Maßnahmen:

- Sie helfen einen bestimmten Zielwert zu definieren, welcher nötig ist, um das Projektziel zu erreichen.
- Sie unterstützen die Transparenz des Projektstatus.

- Sie liefern den Projektmitarbeitern Feedback, z.B. über den Projektfortschritt, welches das Projektmanagement unterstützt.
- Sie fördern das Sammeln von Daten für zukünftige Projektarbeiten.

Prozessmetriken für agile Softwareentwicklungsprozesse sind im Gegensatz zu Prozessmetriken für traditionelle Entwicklungsprozesse noch nicht so weit entwickelt. Leider können Prozessmetriken von traditionellen Entwicklungsmethoden nicht einfach für agile Methoden verwendet werden, da agile Softwareentwicklungsmethoden variierende Funktionalität, feste Zeit und feste Ressourcen fokussieren, während es für traditionelle Softwareentwicklungsmethoden umgekehrt gilt.

Aus diesem Grunde erklärt Liz Barnett: "During 2004 and 2005, vendors, consultants, and user organisations must collectively address the following gap for agile processes to become pervasive: Develop standard metrics to measure the value delivered by agile projects [Ba04]." Im zweiten Abschnitt werden die derzeit gebräuchlichsten agilen Metriken vorgestellt. Diese Metriken werden detailliert präsentiert, da sie für die im Folgenden beschriebene Fallstudie von Nutzen sind.

Das übergeordnete Ziel dieser Arbeit ist es, empirische Daten in einer industriellen Fallstudie zu sammeln, welche die Validität von Prozessmetriken und deren praktischen Einsatz in agilen Entwicklungsmethoden untersucht. Hierzu ist ein spezieller agiler Softwareentwicklungsprozess der T-Systems Multimedia Solutions GmbH gegeben. Zusätzlich ist diese Fallstudie durch das derzeitige Interesse am Messen von agilen Softwareentwicklungsmethoden motiviert. Der Fokus liegt dabei auf Softwareprozess-metriken und deren Einfluss auf die Effizienz agiler Softwareentwicklungsmethoden. Die Studie untersucht, inwieweit Prozessmetriken für agile Softwareentwicklungsteams hilfreich sind, Informationen zu sammeln, um Anpassungen und Verbesserungen an der gesamten Entwicklungsmethode vornehmen zu können.

Die Fallstudie, die erwarteten Resultate und erste Ergebnisse werden in den Abschnitten drei und vier vorgestellt. Abschließend wird eine Zusammenfassung präsentiert.

## 2 Metriken

Derzeit sind nur einige Ansätze bezüglich der Softwaremessung bekannt, welche auf agile Entwicklungsmethoden angewendet werden. Zusammenfassend sind die meist befürworteten agilen Prozessmetriken nachfolgend aufgelistet:

- Software Inventory
  Software Inventory ist die Summe der Arbeit, welche zu einem Projekt hinzugefügt wurde. Die Arbeit ist dabei in Pakete aufgeteilt, welche durch das System laufen und verschiedene Reifegrade z.B. geplant, in Bearbeitung, abgeschlossen erreichen.
- Work-In-Progress
  Work-In-Progress verdeutlicht die historische Summe aller Ideen, welche sich im System befinden, bevor sie in funktionierende Software umgewandelt werden.
- Velocity / Lead Time
  Mit Velocity kann bestimmt werden, wie hoch der Aufwand ist, funktionierende Software herzustellen.

Die Metriken wie Inventar (Software Inventory) und Geschwindigkeit (Velocity) sind wichtige Softwareprozessmetriken, die Aufschluss über die Effizienz des Entwicklungsprozesses liefern. Diese Metriken werden im Weiteren näher vorgestellt.

## 2.1 Software Inventory

Software Inventory beschreibt die Menge der Arbeit, die durch eine definierte Inventory-Einheit ausgedrückt wird, welche von der Softwareentwicklungsmethode abhängt [An04]. Zum Beispiel kann im eXtreme Programming diese Einheit als User Story betrachtet werden. In der folgenden Fallstudie wird ein Feature-Driven-Development-Ansatz untersucht und daher die Inventory-Einheit Feature definiert. Ein Feature kann dabei als der kleinste Bestandteil einer laufenden Software definiert werden. Software Inventory kann verschiedene Zustände erreichen. Aufgrund der Umwandlung von Arbeitspaketen in den Phasen wie Analyse, Design, Implementierung oder Test kann das Inventar zum Beispiel den Status geplant, gestartet, implementiert, getestet oder fertig gestellt annehmen.

Die gesamte Summe der Arbeit im System wird berechnet durch die Differenz zwischen geplanter und fertig gestellter Arbeit. Dieser Wert wird als Work-In-Progress bezeichnet.

## 2.2 Velocity bzw. Lead Time

Mit Velocity kann der Aufwand, funktionierende Software herzustellen, bestimmt werden [AH03]. Mit Hilfe der Velocity-Metrik kann vorhergesagt werden, wann der wahrscheinliche Fertigstellungstermin erreicht wird [PP03]. In Referenzen erwähnen Forscher auch Lead Time, wenn sie von Velocity sprechen, und meinen denselben Sachzusammenhang. Velocity wird als die Zeit charakterisiert, welche gebraucht wird, um Software Inventory von Input in Output umzuwandeln. Velocity beschreibt demnach, die Geschwindigkeit des Teams, geplante Features in auslieferbare Features umzuwandeln.

Der Wert der Velocity kann auf zwei Granularitätsebenen bestimmt werden – Iterationen und Releases. Berechnet wird Velocity wie folgt: Die derzeit fertig gestellte Arbeit geteilt durch die benötigten Arbeitstage. Auf der Iterationsebene wird die Formel angepasst zu: Anzahl von fertig gestellten Features oder User Stories pro Iteration.

## 2.3 Visualisierung von Messungen

Als Darstellungsmöglichkeit von Messungen werden Cumulative-Flow-Diagramme (CFD) gewählt. Dieser Diagrammtyp stellt eine Methode zur Verfügung den Projekt-fortschritt, auf der Basis von Burn-up-Diagrammen darzustellen. Burn-up Diagramme visualisieren die Anzahl von fertig gestellten Features. Cumulative-Flow-Diagramme zeigen neben den gesamten Umfang des Projektes auch den Fortschritt von bestimmten Features [An04].

Abbildung 2.3.1 stellt ein Cumulative-Flow-Diagramm mit Software Inventory, Work-In-Progress und Velocity dar.

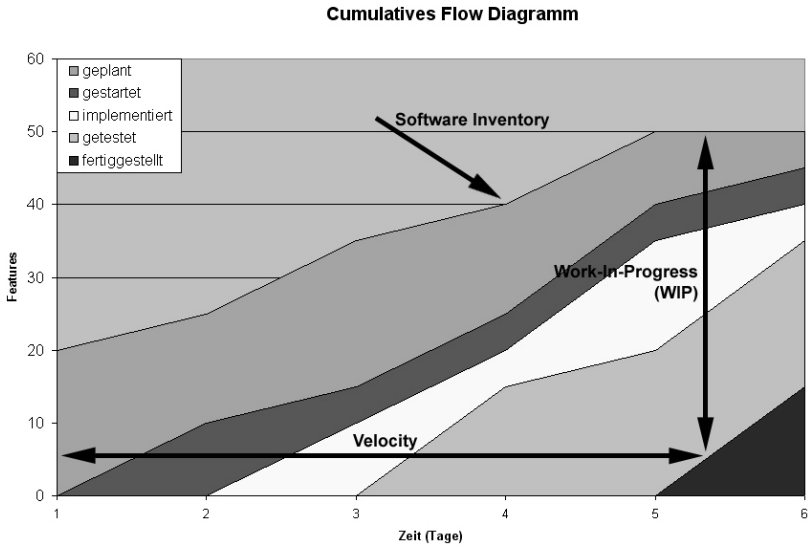


Abbildung 2.3.1: Cumulative-Flow-Diagramm mit geplanten, gestarteten, implementierten, getesteten und fertiggestellten Features, Work-In-Progress und Velocity

In dem Diagramm wird demnach ein Inventory von 20 Features zu Beginn des Projektes dargestellt. Am Ende der fünften Woche ist das Inventory gestiegen. Software Inventory zählt nun 50 Features. Neben den finanziellen Metriken wie Software Inventory können auch Metriken wie Work-In-Progress und Velocity in den CFDs erkannt werden. Somit sind in der ersten Woche 20 Features in Bearbeitung, in der vierten Woche 40 Features und am Ende des Projektes 35 Features.

Inventory-basierte Metriken werden als einfach, relevant, selbst-generierend und vorhersehbar bewertet [An04].

# 3 Beschreibung der Fallstudie

## 3.1 Das Team

Die T-Systems Multimedia Solutions GmbH ist ein High-End Multimedia Dienstleister, der sich auf die Entwicklung webbasierter Anwendungen, Portale und Marktplätze spezialisiert hat. Das Unternehmen wurde im Jahr 2004 zum zweiten Mal in Folge die Nr. 1 im New Media Service Ranking.

Das Team des Berliner Standortes, welches für die Fallstudie ausgewählt wurde, entwickelt komplexe J2EE-Anwendungen in Verbindung mit Content Management

Systemen und arbeitet seit mehreren Jahren nach den Prinzipien agiler Softwareentwicklung. In dieser Zeit wurde die Methodik immer weiter verfeinert und entspricht keiner Lehrbuch-Methodik mehr[1]. Der gewählte Ansatz wird durch niedrige Fehlerraten und höchste Kundenzufriedenheitswerte bestätigt.

### 3.2 Die agile Methode

Das Team legt Wert auf die vier im Agilen Manifest [@Ag] niedergeschriebenen Punkte: Individuen und Interaktion, funktionierende Software, Kundeneinbeziehung und die Fähigkeit auf Änderungen zu reagieren. Schlüsselfaktoren für die Messungen sind:

- Releases
  Die Auslieferung von funktionierender Software ist das Hauptziel von jedem Release. Ein Release dauert mindestens ein bis zwei Monate. Das Release stellt die Grundlage dar und kann als unabhängiges Projekt angesehen werden. Jedes Release beinhaltet mehrere Iterationen.
- Iterationen
  Das Arbeiten mit Iterationen ist ein Schlüsselprinzip der agilen Softwareentwicklung. Eine Iteration ist durch spezifische Arbeitsschritte charakterisiert, welche in jeder Iteration wiederholt werden. Iterationen dauern mindestens eine Woche und maximal zwei Wochen.

Das Arbeiten mit Iterationen und das Vorhandensein einer direkten Kommunikationslinie zum Kunden hilft dem Team auf der einen Seite, funktionierende Software so schnell wie möglich zu realisieren, und auf der anderen Seite, auf eventuelle Änderungswunsche des Kunden flexibel zu reagieren. Lindvall et al. sagen dazu: "frequent iterations are very important for frequent monitoring of warning signs" [Li02].

### 3.3 Die Messwerkzeuge

Das OpenSource Tool Xplanner ist essentiell für die Fallstudie. Dieses Werkzeug ist ein Planungs- und Zeiterfassungswerkzeug, welches das Verwalten von Projekten, Iterationen, Features und Aufgaben ermöglicht. Die graphischen Darstellungsmöglichkeiten von Microsoft Excel wurden angewendet, um die gesammelten Daten zu analysieren und interpretieren.

### 3.4 Angewandte Metriken

Basierend auf dem Goal-Question-Metric Ansatz (GQM) [RB87] wurden Softwareprozessmetriken abgeleitet, welche während der Fallstudie gesammelt wurden. Dabei liefert der GQM-Ansatz "a structure from which managers and developers can derive a set of crucial project goals, plus questions that must be answered in order to tell if the goals have been met." [FP97].

[1] Kontakt für weitere Informationen: Stefan Bessing <stefan.bessing@tsi-mms.de>

Als Ziel der Fallstudie wird die Steigerung der Effizienz der angewandten agilen Methode durch das Identifizieren von Optimierungspotentialen bei der Methode gesehen. Daraus leiten sich folgende Fragen und Prozessmetriken, wie im GQM Modell in Abbildung 3.4.1 ersichtlich, ab.

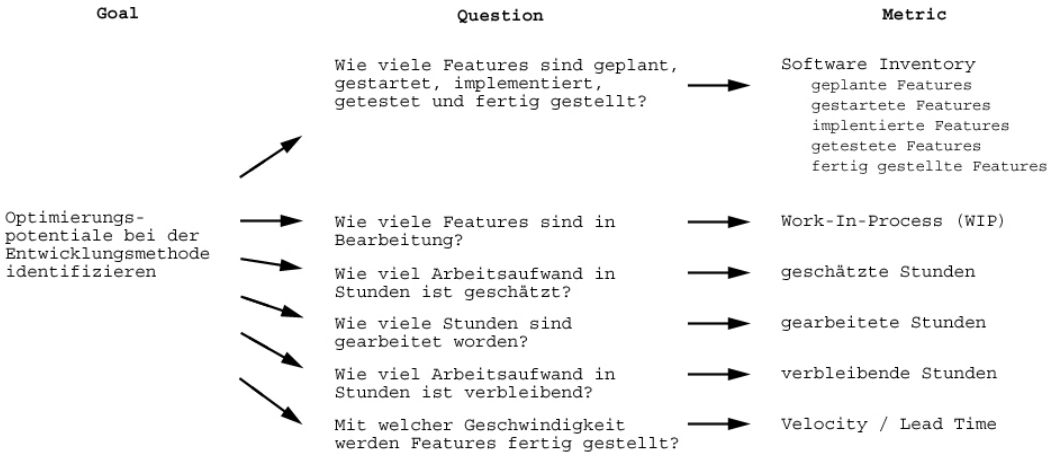


Abbildung 3.4.1: Goal-Question-Metric-Modell für Fallstudie

## 3.5 Erwartete Ergebnisse und Darstellung

Die erwarteten Ergebnisse der Fallstudie basieren hauptsächlich auf Erfahrungen von Forschern wie zum Beispiel David Anderson. Sowohl die erwarteten als auch die erzielten Ergebnisse werden in Cumulative-Flow-Diagrammen dargestellt und präsentiert.

Anderson befürwortet das in Abbildung 3.5.1 dargestellte Cumulative-Flow-Diagramm.

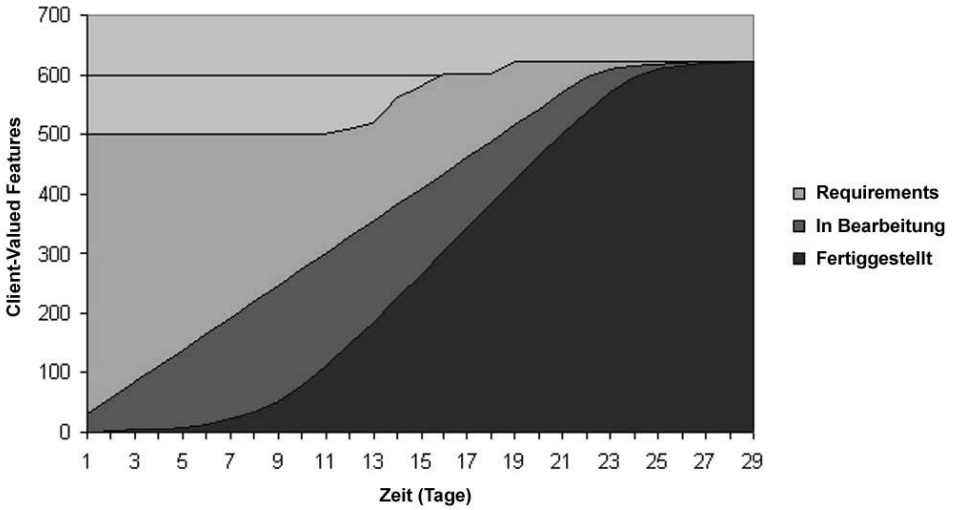


Abbildung 3.5.1: Ideales Projekt nach David Anderson

Dieses Diagramm visualisiert ein ideales Projekt mit Requirements (geplante Features), Features In Bearbeitung und fertig gestellten Features.

Der Trend der verschiedenen Kurven ist durch folgende Argumente begründet [An04]:

- Scope Creep und Dark Matter sind ausschlaggebend für eine steigende Kurve der Requirements (geplante Features). Scope Creep definiert die Anforderungen, die im fortgeschrittenen Projekt hinzugefügt werden. Dark Matter ist die Summe der Arbeit, welche bereits zu Beginn des Projektes existierte, aber für die Teammitglieder bis zu diesem Zeitpunkt nicht ersichtlich war.
- Basierend auf einer konstanten Produktivitätsrate werden die für den Kunden gewinnbringenden Features als ein kontinuierlicher Strom angesehen, während sie in Bearbeitung sind.
- Die Kurve der fertig gestellten Features folgt einer S-Kurve. Der Anfang der S-Kurve ist dadurch begründet, dass gewinnbringende Features mit Beginn des Projektes nicht sofort umgesetzt werden. Das Auslaufen der S-Kurve ist dadurch begründet, dass am Ende des Projektes Aufgaben wie Projektintegration, Fehlerbehebung und Refactoring zu bewältigen sind.

Grundsätzlich werden in CFDs nur die Verläufe von Software Inventory (geplante, gestartet, etc. Features) abgebildet. Damit werden einige Metriken, die im GQM-Model dargestellt sind, vernachlässigt. Um alles auf einen Blick sichtbar abzubilden, ist zu befürworten, dass geschätzte, gearbeitete und verbleibende Zeitaufwände in den CFDs hinzugefügt werden.

## 4 Erste Ergebnisse der Fallstudie

Abbildung 4.1 beinhaltet alle Informationen, die für das Projektteam wichtig sind, um eine Aussage über Optimierungspotentiale der Entwicklungsmethode zu treffen.

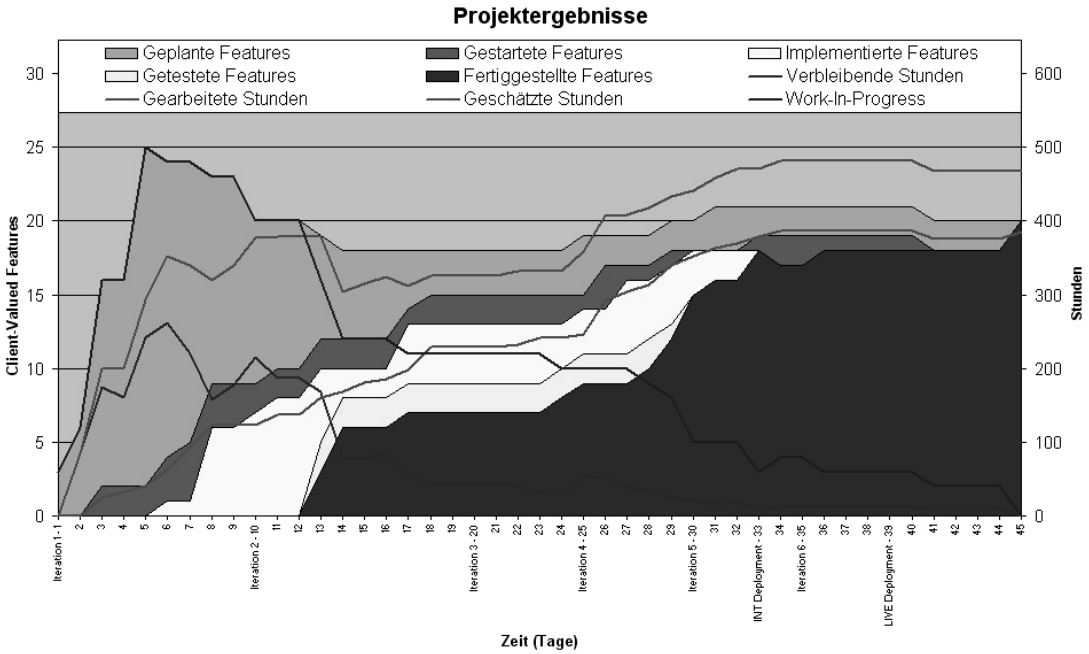


Abbildung 4.1: Projektresultate (CFD)

Dabei spielen (1) die Entwicklung aller Software-Inventory-Kurven, (2) die Anzahl der geplanten Features vs. geschätzter Arbeitsaufwand und (3) Work-In-Progress vs. verbleibender Arbeitsaufwand eine wichtige Rolle und werden folgend näher betrachtet.

(1) Die Entwicklung der Software-Inventory-Kurven spiegelt den Verlauf der Projekt-aktivitäten während des Releases wieder. Das bedeutet, die Arbeitspakete (Features) des Projektes wandern sichtbar durch die verschiedenen Entwicklungsphasen wie zum Beispiel Planung, Implementierung oder Testen. Somit wird sichtbar, dass Features hauptsächlich zu Beginn des Projektes geplant wurden. Der Verlauf der geplanten Features entspricht damit nicht dem erwarteten Verlauf. Grundlegend wird zu Beginn des Projektes ein konstanter Verlauf der Requirements-Kurve erwartet. Zu Beginn des Projektes ist jedoch ein steiler Anstieg der Requirements-Kurve zu verzeichnen. Maximal 25 Features wurden zum Projekt hinzugefügt.

Erste Features sind am sechsten Projekttag implementiert und können getestet werden. Am Projekttag 13 werden erste Features getestet und fertig gestellt. Die Kurve für Features in Bearbeitung verläuft damit wie erwartet linear gleichbleibend. Die Kurve für fertig gestellte Arbeitspakete zeichnet sich im Ansatz als S-Kurve ab, jedoch sind sehr oft Kurven-Sprünge zu verzeichnen. Diese Sprünge sind dadurch begründet, das der Build- und Deployment- Prozess wöchentlich durchgeführt wird. Am Ende des Projektes existieren noch immer nicht gestartete Features. Dieses ist durch Features mit Aufgaben wie Konfiguration oder Support begründet, welche erst nach der Auslieferung des Produktes an den Kunden realisiert werden können.

(2) Aus den angewandten Metriken kann als zweite wichtige Information das Zusammenspiel der geplanten Features und des geschätzten Arbeitsaufwandes herausgefiltert werden. In Abbildung 4.2 wird dieses verdeutlicht.

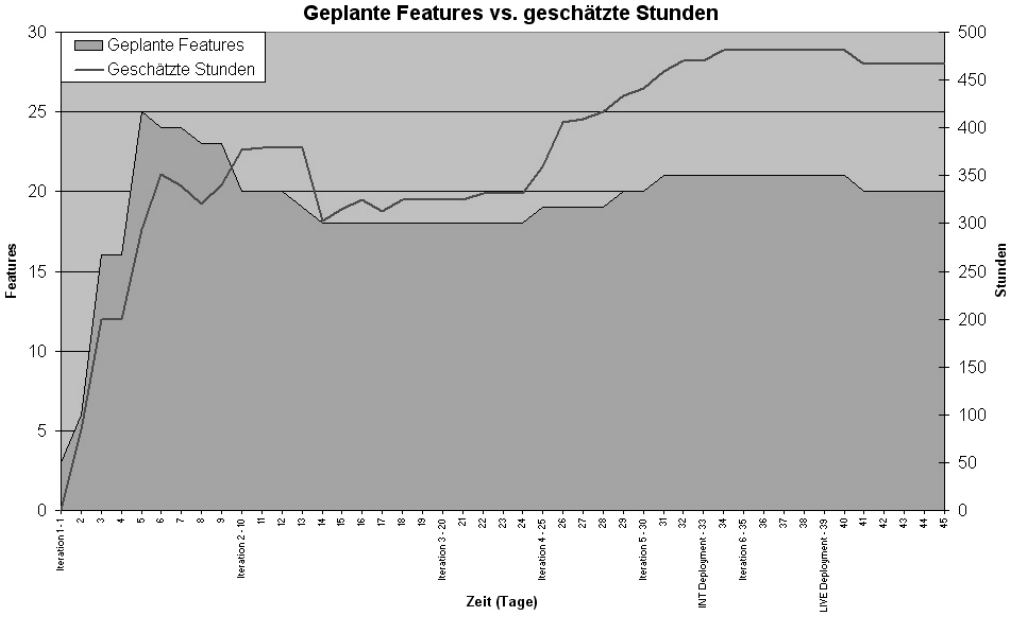


Abbildung 4.2: Geplante Features vs. geschätzter Zeitaufwand

Mit Hilfe dieser Information kann eine Aussage darüber getroffen werden, wie viel Stundenaufwand nötig ist, um einen spezifische Menge von Features zu realisieren.

Zu Beginn des Projektes wird eine Anzahl Features geplant und der Stundenaufwand dafür geschätzt. Sowohl die Kurve für geplante Features als auch die Kurve für den geschätzten Stundenaufwand steigen. Später im Projektverlauf werden Features gestrichen, aber der geschätzte Aufwand steigt trotzdem weiterhin. Zum Ende des Projektes werden Features mit einem hohen geschätzten Stundenaufwand hinzugefügt. Das Hinzufügen umfangreicher Features zum Projektende bedeutet ein erhöhtes Risiko für den Projekterfolg.

(3) Auf der Grundlage von gearbeiteten und verbleibenden Stunden kann eine Aussage über den Projektaufwand getroffen werden. Diese Information im Vergleich zu den Daten über den geschätzten Zeitaufwand liefert den Wert der Schätzungsdifferenz. In Abbildung 4.3 sind die Kurvenverläufe von den verbleibenden Stunden, den gearbeiteten Stunden, den geschätzten Stunden und dem Projektaufwand dargestellt. Für das in der Fallstudie beobachtete Projekt kann ein positiver Fortschritt beobachtet und Rückschluss auf die Effizienz des Projektes gezogen werden. Der reelle Aufwand lag zu jedem Projektzeitpunkt unter dem geschätzten Aufwand. Ein Überschreiten der geschätzten Aufwandskurve durch die Projektaufwandskurve hätte ein frühes Warnsignal bezüglich des Projektgesundheitszustandes geliefert.

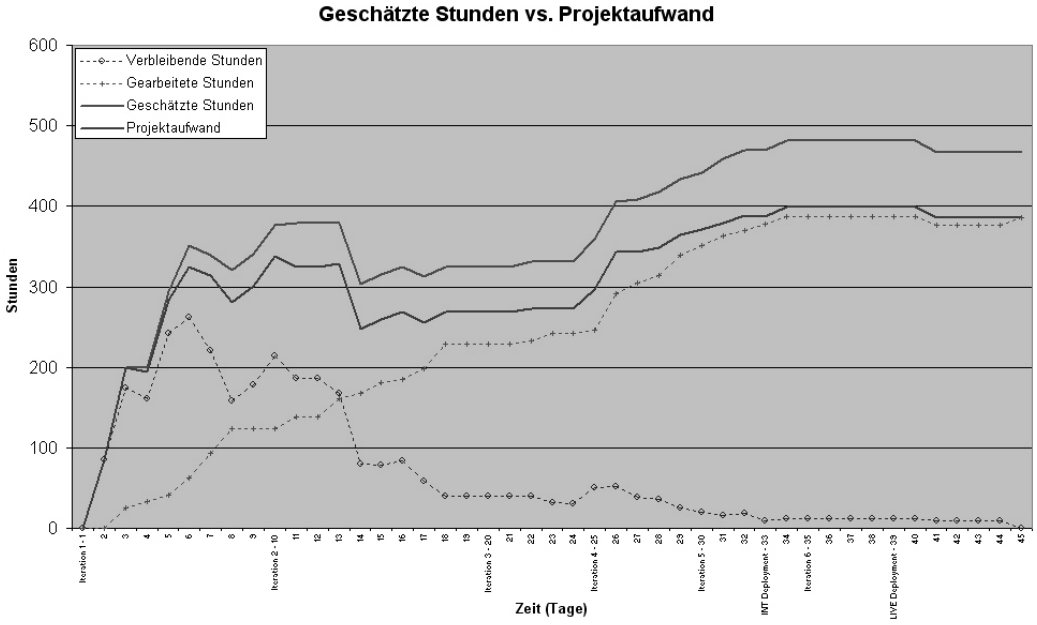


Abbildung 4.3: Geschätzte Stunden vs. reeller Projektaufwand

Weitere Rückschlüsse über den Projektfortschritt und damit die Effizienz des Projektes können mit Hilfe der Metrik Velocity gezogen werden. Die Velocity wurde in der durchgeführten Fallstudie für jede Iteration bestimmt.

In Abbildung 4.4 ist das Cumulative-Flow-Diagramm mit der Zeitspanne für jede Iteration und der Anzahl der in der Iteration fertig gestellten Features dargestellt.

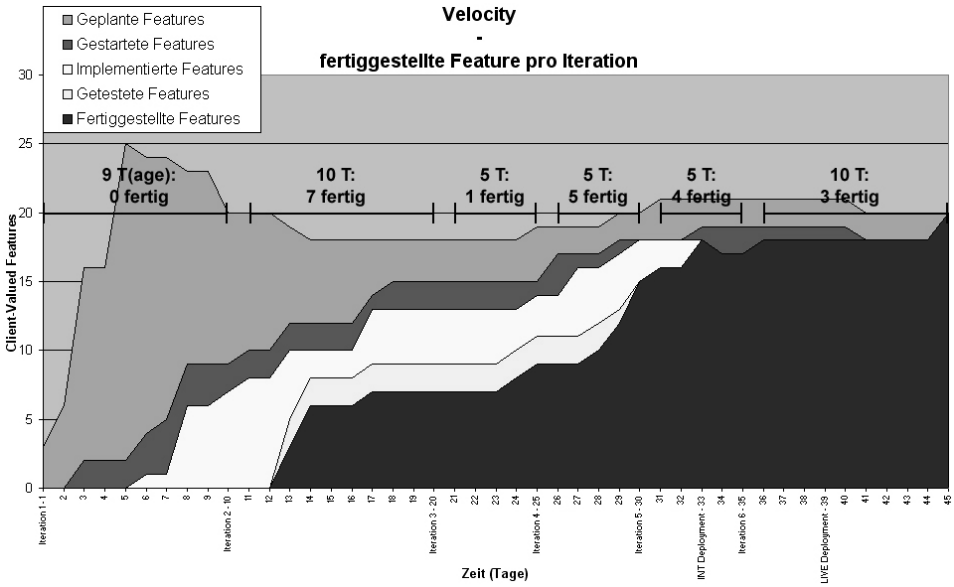


Abbildung 4.4: Velocity – fertig gestellte Features pro Iteration

Wie bereits erwähnt werden Features nicht sofort mit Beginn des Projektes fertig gestellt. Am Ende des Projektes sind hauptsächlich Konfigurations-, Support- und Integrationsaufgaben zu realisieren. Damit können die erste und letzte Iteration aus der Betrachtung herausgenommen werden. Für die Bestimmung der Velocity verbleiben somit vier Iterationen, in denen 17 Features fertig gestellt wurden. Im Durchschnitt arbeitet das Team demnach mit einer Velocity von vier Features pro Iteration. Diese Information ist hilfreich zur Planung und Bestimmung des wahrscheinlichen Endtermins.

# 5 Zusammenfassung

Softwaremetriken sind ein Hilfsmittel im Bereich Software Engineering, um Probleme im Softwareentwicklungsprozess aufzudecken. Als Vorteile von Softwaremetriken werden das Verstehen, Kontrollieren, Vorhersagen und Verbessern der gesamten Software-Engineering-Methodologie und ihre Ergebnisse verstanden. Allerdings hat eine im Jahr 2003 durchgeführte Untersuchung zum Einsatz von Softwaremetriken gezeigt, dass nur 47 Prozent der Softwareingenieure Metriken während des gesamten Softwareentwicklungsprozesses sammeln [LZ03]. Diese Daten lassen auf eine Abneigung bzgl. Softwaremetriken unter den Softwareingenieuren schließen. Nach Schlussfolgerungen der Autoren beruht diese Abneigung einerseits offensichtlich auf mangelnder Ausbildung. Andererseits könnte sie möglicherweise auf der Ablehnung des

Gebrauchs von Softwaremetriken beruhen. Immerhin haben Softwareingenieure die Erfahrung gemacht, dass es schwierig ist, Softwaremetriken zu sammeln und zu analysieren.

Um die Abneigung bezüglich des Gebrauchs von Softwaremetriken zu überwinden, sollten Metriken einfach und leicht erhältlich, relevant, genau definiert und zuverlässig sein. Anderson legt fest, dass Metriken eher den genannten Anforderungen entsprechen sollten, als behindernd oder blockierend zu sein [An04].

Die Fallstudie über Softwareprozessmetriken und agile Entwicklungsmethoden zeigt, dass Prozessmetriken wie Software Inventory, Work-In-Progress und Velocity mit den aufgeführten Anforderungen an Softwaremetriken übereinstimmen. Sie liefern in Kombination mit geschätzten, gearbeiteten und verbleibenden Stunden feste Anhaltspunkte über die Effizienz des während der Fallstudie beobachteten Projektes.

Darüber hinaus helfen die Messergebnisse dem Entwicklungsteam, Informationen darüber zu erhalten, an welchen Stellen ihre Entwicklungsmethode verbessert und angepasst werden könnte. Somit erkannte das Team, dass Optimierungsbedarf bei der Planungsphase zu Beginn des Projektes und bei dem Build- und Deployment-Prozess besteht. Indem alle geplanten Features vor Beginn der ersten Iteration zum Projekt hinzugefügt werden, ist eine Steigerung der Effizienz möglich. Der Build- und Deployment-Prozess wurde bis jetzt einmal in der Woche durchgeführt. Ein täglicher Build- und Deployment-Prozess würde dem Team helfen, mehr Testgelegenheiten zu haben und so Features sehr viel schneller fertig zu stellen.

Alles in allem liefern Cumulative-Flow-Diagramme eine Methode, den Projektfortschritt von agilen Methoden mit Hilfe von Software Inventory und Work-In-Progress sichtbar zu machen. Aber wie die Fallstudie zeigt, ist es sehr wichtig, neben den Informationen über die Anzahl der Features auch Informationen über den geschätzten und verbleibenden Arbeitsaufwand zu haben. Deshalb befürworten die Autoren, die Kurven für geschätzte und verbleibende Stunden ebenfalls den Cumulative-Flow-Diagrammen hinzuzufügen.

Neben den erwähnten Metriken wie Software Inventory und Work-In-Progress ist die Metrik Velocity in Cumulative-Flow-Diagrammen sichtbar. Velocity beschreibt die Geschwindigkeit des Softwareentwicklungsteams, geplante Features in abgeschlossene Features umzuwandeln. Das Bestimmen dieser Geschwindigkeit ist ein sehr wichtiger Aspekt für die Agilität. Agil zu sein, ist die “ability to both create and respond to change in order to profit in a turbulent business environment” [Hi02]. Um schnell handeln und fertige Software so oft wie möglich ausliefern zu können, ist es wichtig zu wissen, wie schnell ein Team sein kann. Das bedeutet, ein Softwareentwicklungsteam kann nur auf eine flexible Art entwickeln und auf Änderungen reagieren, wenn es seiner Team Velocity bewusst ist.

Auf der Basis der Metrik Velocity kann auch eine Vorhersage zu dem möglichen Endtermin des Projektes getroffen werden. Während der Fallstudie wurde die Velocity auf Ebene der Iteration bestimmt. Jedoch war dieser Wert nicht so hilfreich wie erwartet. Es stellte sich heraus, dass für die Vorhersage des Endtermins mehr Information als nur die Velocity des Teams benötigt wird. In dieser Hinsicht befürworten die Autoren, in der

Zukunft mehr Velocity-Daten des spezifischen Teams zu sammeln. Mit Hilfe dieser Daten könnte ein zuverlässigerer Fertigstellungstermin vorhergesagt werden.

Softwareprozessmetriken, wie in dieser industriellen Fallstudie gezeigt, helfen agilen Teams, den Projektfortschritt zu verdeutlichen und die Effizienz der verwendeten Entwicklungsmethode zu bewerten. Damit wird dem Team die Möglichkeit gegeben, Optimierungspotentiale im Prozess bzw. der Methode selbst zu erkennen und zu nutzen und somit ihren agilen Entwicklungsprozess flexibel anzupassen, zu gestalten und zu verbessern.

## Literaturverzeichnis


[@Ag] http://www.agilemanifesto.org; 30.10.2004

[AH03] Alleman, G.B.; Henderson, M.; „Making Agile Development Work in a Government Contracting Environment – Measuring velocity with Earned Value“; Submitted to Agile Development; Salt Lake City, Utah; June 2003

[An04] Anderson, D. J.; “Agile Management for Software Engineering – Applying the Theory of Constraints for Business Results”; Pearson Education Inc.; 2004

[Ba04] Barnett, L.; “Adopting Agile Development Processes”; Forrester Research 2004.

[BT04] Boehm, B.; Turner, R.; “Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods”; Proceeding of the 26th International Conference on Software Engineering (ICSE) 2004

[Co02] Cohn, M.; “A Measured Response”; Software Quality Engineering Magazine; September/October 2002; pp.21-25

[Fa03] Favaro, J.; “Value-Based Management and Agile Methods”; Proceedings of 4th International Conference on XP and Agile Methods; May 2003

[FP97] Fenton, N.E.; Pfleeger, S.L.; “Software Metrics – A Rigorous and Practical Approach”; PWS Publishing Company; 1997

[Hi02] Highsmith, J. “Agile Software Development Ecosystems”; The Agile Software Development Series, Addison Wesley; 2002

[Li02] Lindvall, M.; Basili, V.; Boehm, B.; Costa, P.; Dangle, K.; Shull, F.; Tesoriero, R.; Williams, L.; Zelkowitz, M.; “Empirical Findings in Agile Methods”; Proceedings of Extreme Programming and Agile Methods; XP/Agile Universe 2002; pp. 197-207

[LZ03] Loeper, S.; Zehle, M.; „Evaluation of Software Metrics in the Design Phase and their Implication on CASE Tools“; Blekinge Institute of Technology; Department of Software Engineering and Computer Science; June 2003

[PP03] Poppendieck, M.; Poppendieck, T.; “Lean Software Development – An Agile Toolkit”; The Agile Software Development Series, Addison Wesley; 2003

[RB87] Rombach, H. D.,Basili, V. R., “Quantitative Software-Qualitätssicherung”, in: Informatik-Spektrum 10/1987, S. 145-158