Wie kann man risikoorientiert Software entwickeln? Risikobegriffe und Software-Paradigmen

Artur Nowodworski

Studierender
der Beuth-Hochschule für Technik Berlin
s55732@beuth-hochschule.de

ABSTRACT

Die Entwicklung von Software hängt mit technologischen und organisatorischen Herausforderungen zusammen und ist unterschiedlichen Risiken ausgesetzt. Wie kann man nun Software risikoorientiert entwickeln? Mit dieser Frage beschäftigt sich dieses Paper. Zuerst werden der Begriff des Risikos sowie ein systematischer Prozess zur Risikobehandlung, d.h. das Risikomanagement erklärt. Anschließend wird darauf eingegangen, wie der Softwareentwicklungsprozess organisiert werden kann. Dementsprechend erfolgt eine Erläuterung der Grundzüge von drei grundsätzlichen Softwareparadigmen: dem sequentiellen, dem iterativen und dem agilen Paradigma sowie jeweils einem für sie stehenden Vorgehensmodell. Hierbei wird darauf hingedeutet, dass jedes der Paradigmen durch eine spezifische Risikoorientierung charakterisiert wird. Abschließend folgt ein zusammenfassendes Fazit.

Categories and Subject Descriptors

D.2.9 [SOFTWARE ENGINEERING]**:** Management – *Life cycle, Software process models*

General Terms

Management

Keywords

Software engineering, software development, risk management, process models

1. EINLEITUNG

Allein in Deutschland hat die Informations- und Kommunikationstechnologie-Branche im Jahr 2014 in Höhe von 4,6% zur gewerblichen Wertschöpfung in Deutschland beigetragen, weshalb sie sich bereits auf dem Platz 6 – knapp hinter dem Fahrzeug- und vor dem Maschinenbau platziert [BWE15, S. 17 - 18]. Darüber hinaus erwirtschaften 27% der Unternehmen in Deutschland mehr als 60% ihres Umsatzes digital [BWE15, S. 6]. Dementsprechend spielt Software eine grundlegende Rolle in der Wirtschaft.

Da Softwaresysteme in immer neueren Anwendungsgebieten verwendet werden und immer schwierigere Aufgaben erledigen, wächst ihre Komplexität sowie die Anforderungen an ihre Qualität [Bal09, S. 11 - 12]. Des weiteren wird Software in immer kürzeren zeitlichen Rahmen erstellt [Ahr08, S. 37]. Daher bringt die Realisierung von Softwareprojekten viele Risiken mit sich.

Im Rahmen der Studie „CHAOS Report“ untersucht das Forschungsunternehmen *Standish Group* regelmäßig seit 1994 IT-Projekte auf ihre Erfolge. Der Chaos-Studie aus dem Jahr 2001 zufolge sind 23% der IT-Projekte gescheitert und 49% der IT-Projekte konnten verspätet und nur mit höheren als ursprünglich geplanten Kosten realisiert werden. Nur 28% der IT-Projekte wurden im vorausgesetzten Zeit- und Finanzrahmen abgeschlossen, wobei 67% dieser Projektgruppe alle am Projektanfang geplanten Funktionen umsetzten. [WAL14, S. 106 - 107]

Von diesem Hintergrund benötigt man entsprechende Methoden, Vorgehensweisen, Techniken sowie Werkzeuge, die die systematische Erstellung von Softwareprodukten als Gegenstand haben. Mit diesen Themen beschäftigt sich die Softwaretechnik.

Der Begriff Softwaretechnik wurde Ende der 1960er Jahre geprägt, und zwar als Reaktion auf die sog. *Softwarekrise,* mit Hilfe derer auf die Schwierigkeiten bei der Softwareerstellung hingedeutet wurde. Mitte der 1960 Jahre erkannte man nämlich, dass Softwaresysteme zu langsam entwickelt werden und ihre Kosten zu hoch sind. Außerdem erfüllten sie die Anforderungen der Anwender nicht und zeichneten sich durch ihre geringe Qualität aus.

H. Balzert definiert die Softwaretechnik wie folgt: „Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen. Zielorientiert bedeutet die Berücksichtigung z. B. von Kosten, Zeit, Qualität.” [Bal09, S. 17]

Eine Entwicklung von Software sollte man nicht isoliert betrachten. Zu berücksichtigen ist viel mehr der gesamte Software-Lebenszyklus, der aus fünf Phasen besteht (siehe Abbildung 1). Das Ziel der ersten Phase ist die Softwarespezifikation als Resultat des Requirements Engineering. Im nächsten Schritt wird die Softwarearchitektur erstellt, die dann im Rahmen der dritten Phase in den Code zu implementieren ist. Anschließend erfolgt die Installation und die Abnahme sowie die Einführung des erstellten Softwareproduktes. Während der letzten Phase, d.h. des Betriebs, werden eventuelle Softwarefehler eliminiert bzw. wird die Software angesichts neuer Umstände geändert oder erweitert. Der Lebenszyklus wird durch die Abschaltung, Ablösung bzw. Weiterentwicklung eines Softwareproduktes beendet. [Bal11, S. vi, 1 - 2]

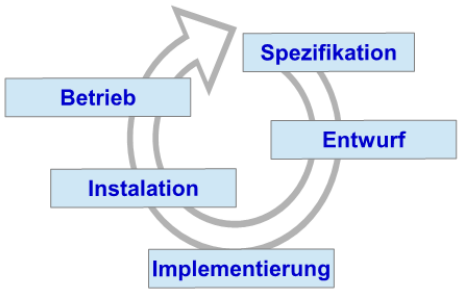

Abbildung 1. Der Lebenszyklus eines Softwaresystems, in Anlehnung an [Bal11, S. 1]

Die Softwareentwicklung erfolgt im Rahmen eines Prozesses. M. Hölzl, und J. Rinke definieren den Begriff des Prozesses als „eine Abfolge von definierten Schritten in einer spezifischen Reihenfolge um ein bestimmtes Ziel zu erreichen“ [Höl16, S. 2]. Ein essentielles Feld der Softwaretechnik bilden die sogenannten Vorgehensmodelle, die zum Ziel haben, den Softwareentwicklungsprozess abstrakt zu beschreiben. Sie definieren die durchzuführenden Aktivitäten und koordinieren die Zusammenarbeit der Beteiligten [Som12, S. 343].

Diese Ausarbeitung beschäftigt sich mit der Risikoorientierung in der Softwareentwicklung. In Kapitel 2 wird der Begriff des Risikos sowie das Risikomanagement erläutert. In Kapitel 3 werden drei Software-Prozess-Paradigmen - das sequentielle, das iterative sowie das agile Software-Prozess-Paradigma [vgl. Wir07, S. 6] – sowie konkrete Vorgehensmodelle dargestellt. Die drei Software-Prozess-Paradigmen unterscheiden sich voneinander in ihrer Risikoorientierung.

2. Risiko und Risikomanagement

Die Softwareentwicklung ist mit Risiken verbunden. In jedem Softwareentwicklungsprojekt können Schwierigkeiten entstehen, die zur Überschreitung der Projektkosten bzw. des Projektzeitplans führen oder das Projekt selbst zum Scheitern bringen können.

Als Beispiel kann die webbasierte Software A2ll dienen, die zur Bearbeitung von Anträgen auf Leistungen des Arbeitslosengeldes II für Ende 2004 geplant wurde. Die Anwendung wurde mit einer Verspätung von drei Wochen und mit nicht unerheblichen Fehlern in Betrieb genommen. So wurden beispielsweise die Kontonummern der Antragsteller, die weniger als zehn Stellen hatten, nicht linksbündig, sondern rechtsbündig aufgeführt. Dies führte zur Zahlungsverspätung. Darüber hinaus wurde die Gebrauchstauglichkeit der Software von Sachbearbeitern negativ beurteilt. Als Gründe für die Schwierigkeiten des Projektes A2ll stellten sich falsche Zeitplanung sowie mangelhafte Beurteilung der Komplexität der geplanten Anwendung heraus. [Aßm06, S. 108]

Das Ziel des Risikomanagements ist es, Risiken rechtzeitig zu erkennen, zu analysieren, entsprechende Maßnahmen gegen sie zu definieren sowie sie zu überwachen. Damit wird die Wahrscheinlichkeit des Projekterfolgs gesteigert. Im vorliegenden Kapitel wird der Begriff des Risikos sowie des Risikomanagements erläutert.

2.1 Begriff des Risikos

Es existieren viele Definitionen dieses Begriffs. Grundsätzlich enthalten sie zwei Merkmale: zum einen die Unsicherheit des Eintretens bzw. Nichteintretens eines Ereignisses und zum anderen negative Auswirkungen dieses Ereignisses [Wal04, S. 6]. Ch. Ebert stellt das Risiko als Multiplikation von zwei Faktoren dar:

Risiko = Auswirkungen x Eintrittswahrscheinlichkeit [Ebe13, S. 8].

Risiken kann man in zwei Kategorien einordnen: in externe oder interne Risiken. Zu der ersten Gruppe zählen verschiedene Einflüsse von Umgebungsbedingungen, die nicht direkt durch das eigene Handeln beeinflussbar sind. So sind dies beispielsweise Markt- sowie Wettbewerbsveränderungen, Konjunkturwandel und Rahmenbedingungen hinsichtlich der Technologie und des Rechts. Als Reaktionen darauf werden hierbei strategische Unternehmensanpassungen, und seltener operative Maßnahmen vorgenommen. Im Gegensatz dazu lassen sich die internen Risiken durch Handlungen der Mitglieder eines Projektteams und damit durch operative Maßnahmen unmittelbar beeinflussen. Das Definieren und die Koordination von operativen Maßnahmen erfolgt im Rahmen des Risikomanagements. [Ahr08. S. 10 -11]

Es gibt diverse Studien über Standardrisiken. Tabelle 1 stellt die zehn nach Ch. Ebert wichtigsten Risikopotenziale in Software- und IT-Projekten dar [Ebe13, S. 8].

1. Unzureichende Organisation	6. Over-Engineering
2. Falsche und fehlende Anforderungen	7. Lieferantenprobleme
3. Sich ändernde Anforderungen	8. Fehler und Qualitätsmängel
4. Unrealistische Planung	9. Architekturdefizite
5. Personelle Schwächen	10. Technologiekomplexität

Tabelle 1. Die Top-10 Risiken in Software und IT, Quelle: [Ebe,13 S. 8]

2.2 Risikomanagement

„Risikomanagement ist der geplante Umgang mit Risiken in einem Projekt.“ [Ahr08. S. 11] Im Risikomanagement geht es nicht darum, die aus Risiken entstandenen Probleme zu behandeln. Es konzentriert sich darauf, mögliche Probleme vor ihrem Eintreten, d.h. Risiken, zu erkennen und angemessene Maßnahmen zum Umgang mit Risiken festzulegen und umzusetzen. Die Risiken sollen also nicht zu Problemen werden, die wiederum geplanten Projektziele gefährden könnten [Ebe13, S. 11].

Risikomanagement ist ein strukturierter und iterativer Prozess, der während der einzelnen Projektphasen im Rahmen des gesamten Lebenszyklus wiederholt wird. I. Sommerville teilt Risikomanagement in die folgenden vier Phasen ein: Risikoerkennung, Risikoanalyse, Risikoplanung und Risikoüberwachung [Som12, S. 652] (siehe Abbildung 2).

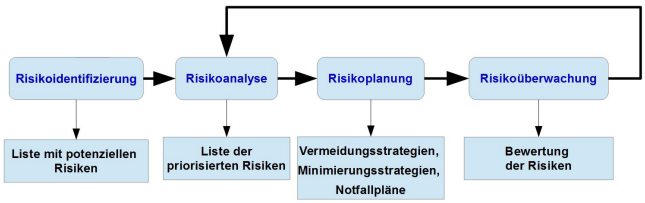

Abbildung 2. Ablauf des Risikomanagements, in Anlehnung an [Som12, S. 652]

2.2.1 Risikoerkennung

Die erste Phase des Risikomanagements: Risikoerkennung besteht darin, Risiken zu identifizieren. Nach I. Sommerville sind hierbei Risiken sowohl für die zu erstellende Software und den

Softwareentwicklungsprozess als auch für die Entwicklungsfirma zu berücksichtigen [Som12, S. 652].

Es existieren unterschiedliche Techniken zur Risikoerkennung. Eine der wichtigsten sind Checklisten [Ebe13, S. 27], die unterschiedliche Arten von Risiken umfassen. Hierzu schlägt I. Sommerville die folgenden sechs Gruppen vor:

- technologische Risiken, deren Risikoquelle in den Software – und Hardwaretechnologien der zu entwickelnden Software liegt,
- personenbezogene Risiken, die mit den Mitgliedern des Projektteams verbunden sind,
- unternehmensbezogene Risiken, die auf das Unternehmensfeld, im Rahmen dessen die Software zu entwickeln ist, zurückzuführen sind,
- Risiken durch Werkzeuge, die bei der Softwareentwicklung zur Verwendung kommen,
- Anforderungsrisiken, die sich aus den Änderungen der Anforderungen ergeben,
- Schätzrisiken, die mit dem Managementschätzungen bzgl. der erforderlichen Ressourcen für die zu erstellende Software zusammenhängen [Som12, S. 652].

Darüber hinaus kann zur Risikoerkennung die sog. Postem-mortem-Analyse, die Probleme und Erfahrungen aus früheren Projekten untersucht, herangezogen werden [Bal08, S. 362]. Des Weiteren lassen sich viele Risiken innerhalb eines kurzen Zeitraums mit Hilfe eines Brainstormings ermitteln. Zur Identifizierung von strategischen Risiken bzgl. der Projektumgebung eignet sich z.B. die sog. SWOT-Analyse, mit der Stärken und Schwächen des Unternehmens sowie Chancen und Gefahren bzgl. des Projektes sowie des Markts untersucht werden [Ebe13, S. 25-26].

2.2.2 Risikoanalyse

Während der Risikoanalyse wird zum einen eine Eintrittswahrscheinlichkeit für jedes identifizierte Risiko untersucht. Der Wahrscheinlichkeit kann einer der folgenden fünf Wertebereiche zugewiesen werden [Bal08, S. 364]:

- sehr wahrscheinlich: [75% ,100%]
- wahrscheinlich: [50%, 75%[
- möglich: [25%, 50% [
- unwahrscheinlich: [5%, 25%[
- fast unmöglich: [0%, 5%[

Zum anderen erfolgt während der Risikoanalyse die Ermittlung der Schadenshöhe der Risiken. Sie kann als unbedeutend, gering, mittel, schwerwiegend oder existenzbedrohend eingeschätzt werden [Bal08, S. 364].

Zum Schluss dieser Phase werden die Risiken nach Wichtigkeit geordnet. Je höher die Eintrittswahrscheinlichkeit und die Schadenshöhe des jeweiligen Risikos ist, desto größer ist das Risiko und seine Priorität. Die Ermittlung der Hauptrisiken kann mittels einer Matrix, dessen y-Achse die Wahrscheinlichkeit und die x-Achse die Schadenshöhe darstellt, erfolgen [Bal08, S. 363].

2.2.3 Risikoplanung

Eine Grundlage für die Risikoplanung bilden die Risiken, die bei der Risikoanalyse wegen ihrer hohen Prioritäten identifiziert wurden. Für diese Risiken werden entsprechende Maßnahmen entwickelt.

I. Sommerville unterschiedet hierfür die drei folgenden Kategorien: Vermeidungsstrategien, Minimierungsstrategien sowie Notfallpläne. Vermeidungsstrategien bestehen darin, die Wahrscheinlichkeit des Risikoeintritts zu reduzieren. Minimierungsstrategien verfolgen den Zweck, die Konsequenzen des jeweiligen Risikos zu verringern. Im Rahmen der Notfallpläne werden Lösungen für kritische Risiken wie z.B. finanzielle Schwierigkeiten der Entwicklungsfirma vorbereitet. [Som12, S. 655 - 656]

2.2.4 Risikoüberwachung

Bei der Risikoüberwachung wird untersucht, ob es zu Änderungen der Parameter, die die Hauptrisiken kennzeichnen, kommt. Demnach erfolgt eine erneute Prüfung der Eintrittswahrscheinlichkeit sowie die Schadenshöhe der Hauptrisiken. Wichtig hierbei ist, diverse Indikatoren zu observieren, die Indizien für Änderungen des jeweiligen Risikos darstellen können. Beispielsweise könnten Beschwerden des Auftragsgebers einen Indikator für mögliche Änderungen der Anforderungsrisiken bilden. [Som12, S. 656 - 657]

3. SOFTWAREPROZESS-PARADIGMA UND VORGEHENSMODELLE

„Ein Vorgehensmodell ist eine Sammlung aufeinander abgestimmter Richtlinien, die die Durchführung von Projekten unterstützen.“ [Wir07, S. 6] In einem Vorgehensmodell werden Schnittstellen des Projektes zum Unternehmen, in dem das Projekt durchgeführt wird, festgelegt [Bro13, S. 86]. Ein Vorgehensmodell beschreibt einen abstrakten Prozess, den man für unterschiedliche Projekte anwenden kann [Höl16. S. 3] und bildet einen Umsetzungs- und Organisationsrahmen für Softwareprojekte.
M. Wirsing unterscheidet drei Softwareprozessparadigmen, die Grundprinzipien für Vorgehensmodelle beschreiben: das sequentielle / phasenorientierte, das iterative sowie das agile Paradigma [Wir07, S. 6]. Im Rahmen dieser Paradigmen wird die Risikoorientierung jeweils unterschiedlich realisiert. Nachfolgend werden diese drei Paradigmen mit Hilfe von beispielhaft ausgewählten Vorgehensmodellen erläutert.

3.1 Das sequentielle Softwareprozess-Paradigma

Nach dem sequentiellen Paradigma wird die Softwareentwicklung in Phasen geordnet. Diese Phasen werden aufeinander folgend umgesetzt. Nachdem die vorgängige Phase komplett beendet wurde, fängt die nächste Phase an. Der Zweck dieses Vorgehens garantiert die Vollständigkeit und Stabilität jeder Phase [Bal08, S. 518]. Bekannte Vertreter des sequentiellen Paradigmas sind das Wasserfall-Modell sowie das V-Modell. Im Nachfolgenden werden Grundzüge des Wasserfall-Modells erläutert.

3.1.1 Das Wasserfall-Modell

Im Rahmen des Wasserfall-Modells wird der Entwicklungsprozess in die folgenden Phasen aufgeteilt (siehe auch Abbildung 3): Systemanforderungen,

Softwareanforderungen, Analyse, Entwurf, Codierung, Testen, Betrieb [Bal08, S. 520].

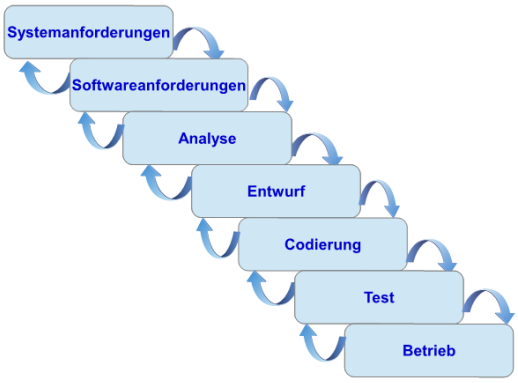

Abbildung 3. Wasserfall-Modell, in Anlehnung an [Bal08, S. 520]

Die Ergebnisse der vorigen Phase bilden eine Grundlage für die nächste Phase. Als Erweiterung des stufenweisen Modells von Herbert D. Benington kennzeichnet sich das Wasserfall-Modell durch Rückkopplungen zwischen den Phasen aus. Die Rückkopplungsschleifen binden jeweils zwei miteinander angrenzende Stufen. [Bal08, S. 519 - 520]

Jede der Phasen umfasst vorgeschriebene Aktivitäten, die in einer bestimmten Reihenfolge zu realisieren sind [Bro13, S. 90]. Demnach wird die Folgeaktivität in voller Breite und erst nach dem Abschluss der vorgängigen Aktivität durchgeführt [Wir07, S. 9]. Da zum Abschluss jeder Aktivität eins oder mehrere Dokumente erstellt werden, gilt das Wasserfall-Modell als dokumentengetrieben [Bal08, S. 520].

Zusammenfassend besteht ein Vorteil des Wasserfall-Modells in der Durchschaubarkeit seiner Struktur sowie im relativ kleinen Aufwand bezüglich der Projektleitung. Durch das Dokumentieren jeder Phase wird die Transparenz des Softwareentwicklungsprozess sowie des Projektfortschritts erreicht.

Auf der anderen Seite kennzeichnet sich dieses Vorgehensmodell durch eine Inflexibilität in Bezug auf die Durchführung von nachträglichen Änderungen in vorigen Phasen aus [Bro13, S. 91]. Sollten z.B. während der Entwurfsphase Inkonsistenzen bezüglich der Anforderungen oder während der Codierung Entwurfsfehler identifiziert werden, so kann sich eine Rückkehr in die vorgängigen Entwicklungsstufen als notwendig erweisen [Som12, S. 58], um eine erforderliche Korrektur zu vollziehen. Die umgesetzten Änderungen sind dann in späteren Phasen zu berücksichtigen. Dies wirkt sich wiederum negativ auf die Projektkosten sowie den Projektzeitplan auf.

3.1.2 Fazit

Die Vorgehensmodelle nach dem sequentiellen Paradigma eignen sich im Prinzip für Projekte mit kleinen Unsicherheiten. Der Softwareentwicklungsprozess ist starr in mehrere Phasen eingeteilt. Bereits zu Beginn des Projektes werden Verpflichtungen festgelegt, sodass es schwierig ist, im Laufe des Prozesses Änderungen, wie z.B. Modifikationswünsche des Auftragsgebers in Bezug auf Anforderungen, durchzuführen. Ein Weiteres Hindernis für Änderungen und Rückkopplungen zwischen den Phasen bilden die hohen Kosten bezüglich der Dokumentationsanfertigung. [Som12, S. 58]

Demnach sollen die Anforderungen an die zu erstellende Software frühzeitig gut abgewogen werden. Gleiches gilt für andere zu erwartende Risikofaktoren wie beispielsweise technische Rahmenbedingungen. Risiken werden anderenfalls zu spät erkannt. Es besteht dann die Gefahr, dass die erst während des Entwicklungsprozesses identifizierten Risikofaktoren bzw. Fehler ignoriert werden, damit der geplante Entwicklungsablauf realisiert wird [Bal08, S. 521]. Dies kann wiederum zur Folge haben, dass spätestens während des Betriebs der Software fehlende Funktionalität, Bugs bzw. schlechte Softwarestruktur festgestellt werden [Som12, S. 58].

Sequentielle Vorgehensmodelle können in einem eindeutigen Vertragsverhältnis sinnvoll sein, durch das die umzusetzenden Anforderungen sowie weitere zu erfüllende Verpflichtungen zwischen dem Auftraggeber und dem Softwareunternehmen klar definiert sind. Dies ist z.B. der Fall bei öffentlich-rechtlichen Ausschreibungsverfahren, für welche Vorgaben gesetzlich geregelt sind. [Säf10, S. 348]

3.2 Das iterative Softwareprozess-Paradigma

Das iterative Paradigma bildet eine Erweiterung des sequentiellen Paradigmas. Eine Grundlage hierfür ergibt sich aus der Feststellung, dass die Lebenszeit eines Softwareprodukts länger als angenommen ist. Demnach ist die Software auch in Bezug auf das Ausmaß seiner Funktionalität zu pflegen. [Wir07, S. 20]
Nach dem iterativen Paradigma werden die einzelnen Phasen bzw. Zyklen der Softwareentwicklung mehrmalig durchgeführt. Auf diese Weise erfolgt der jeweils nachfolgende Durchlauf mit einer Wissenserweiterung, die durch Lern- und Erfahrungseffekte der bisher erreichten Entwicklung gewonnen werden. Ein Merkmal des Paradigmas ist die Tatsache, Anforderungsänderungen während der Iterationen berücksichtigen zu können. [Han15, S. 353]
Die bekanntesten Vertreter des iterativen Paradigmas sind das Spiralmodell sowie Rational Unified Process [Wir07, S. 20]. Nachfolgend werden Grundzüge des Spiralmodells erläutert.

3.2.1 Das Spiralmodell

Das Spiralmodell von Barry W. Boehm ist nicht nur als ein Vorgehensmodell sondern auch als ein Metamodell zu betrachten. Demzufolge ermöglicht es, für jede Entwicklungsphase ein entsprechendes Vorgehensmodell zu verwenden [Bal08, S. 557].

Das Modell stellt den Softwareentwicklungsprozess als eine Spirale dar (siehe Abbildung 4). Demnach bildet jede Spiralwindung - beginnend mit der inneren Windung - eine Entwicklungsphase. Die erste Phase beschäftigt sich mit dem Konzept, in der nächsten werden Anforderungen definiert, Anschließend wird der Systementwurf erstellt. Die letzte Spiralwindung umfasst Detailentwurf, Programmierung, Testen sowie Betrieb. [Som12, S. 76 – 77]

Alle Spiralwindungen bestehen aus den vier folgenden Schritten:

1. Analyse:
2. Evaluierung,
3. Realisierung,

4. Planung [Bro13, S. 91 – 92].

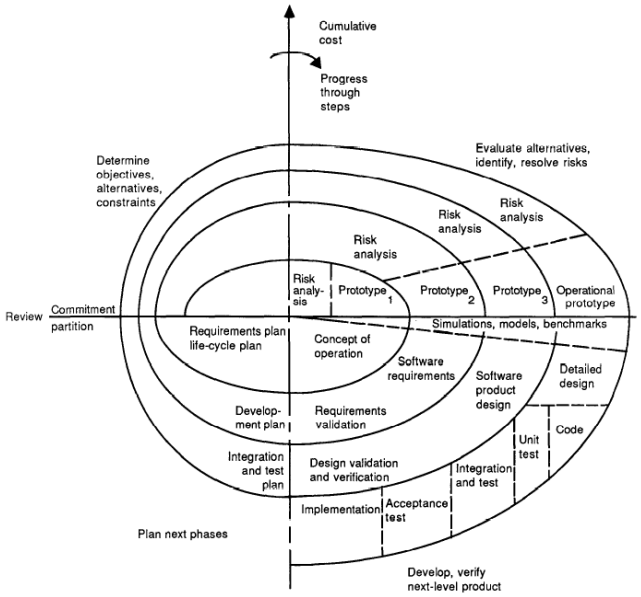

Abbildung 4. Spiralmodell, Quelle [Boe88, S. 64]

Im Rahmen des ersten Schritts werden Projektziele definiert. Den Zielen entsprechend erfolgt dann die Ausarbeitung von alternativen Lösungen. Des Weiteren werden hierbei Randbedingungen der zu erstellenden Software festgelegt. [Som12, S. 77]

Im zweiten Schritt werden die Alternativen ausgewertet. Sollten infolgedessen Projektrisiken identifiziert werden, dann sind entsprechende Strategien zur Risikoüberwindung zu entwickeln und durchzuführen. Dies kann beispielsweise mittels der Entwicklung von Prototypen oder der Durchführung von Simulationen erfolgen. [Bal08, S. 556]

Im nächsten Schritt wird adäquat zu noch zu berücksichtigenden Risiken ein Entwicklungsmodell gewählt, z.B. ein sequentielles Vorgehensmodell. Entsprechend dem festgelegten Modell wird dann die Entwicklung der Software schrittweise fortgesetzt. [Bal08, S. 556]

Im Rahmen des letzten Schritts werden die drei vorigen Schritten der aktuellen Spiralwindung überprüft. Abschließend wird die nächste Spiralwindung geplant. [Bal08, S. 556]

3.2.2 Fazit

Ein Vorteil des iterativen Paradigmas ist die Tatsache, dass es eine frühe Erkennung von Projektrisiken und Fehlern ermöglicht**.** So kann es beispielsweise während der Entwicklung eines großen Systems zu Änderungen von Architekturen kommen. Des Weiteren kann der Auftraggeber seine Anforderungen ändern bzw. die zu erstellende Software genauer präzisieren, nachdem die Entwicklung ein Stück weit erfolgt ist. Mittels der Iterationen und der regelmäßigen Überprüfung des Projektes ist es möglich, auf solche Situationen frühzeitig zu reagieren und die Entwicklung entsprechend umzulenken.
Das iterative Paradigma fördert auch die inkrementelle Entwicklung [Wir07, S. 21, 27]. In ihrem Rahmen wird die Implementierung schrittweise entwickelt und dann dem Auftraggeber präsentiert. Die Feedbacks des Auftraggebers werden anschließend bei der Anpassung des aktuellen Teilprodukts sowie bei der Entwicklung des übrigen Teils des Gesamtsystems berücksichtigt. Damit wird die Eintrittswahrscheinlichkeit von unerwarteten Anforderungsänderungen minimiert.
Ein Nachteil des iterativen Paradigmas ist unter anderem ein erheblicher Projektmanagementaufwand. Denn im Laufe der Entwicklung ist es häufig nötig über den weiteren Ablauf des Prozesses zu entscheiden. [Bal08, S. 560]

3.3 Das agile Softwareprozess-Paradigma

Das agile Softwareprozessparadigma baut auf dem iterativen Softwareprozessparadigma auf, indem Iterationen im Softwareentwicklungsprozess dynamisch konzipiert werden. Ein Ziel der agilen Vorgehensmodelle ist es also, die jeweiligen Änderungen während der Softwareentwicklung zu adaptieren. [Wir07, S. 29]

Gleichzeitig bildet das Paradigma eine Gegenbewegung im Verhältnis zu den formalisierten Vorgehensmodellen. Die folgenden Kerngedanken der Agilität wurden in dem sog. Manifest für agile Softwareentwicklung von 2001 formuliert:

> „Individuen und Interaktionen [werden] mehr als Prozesse und Werkzeuge [geschätzt.]
>
> Funktionierende Software [wird] mehr als umfassende Dokumentation [geschätzt.]
>
> Zusammenarbeit mit dem Kunden [wird] mehr als Vertragsverhandlung [geschätzt.]
>
> Reagieren auf Veränderung [wird] mehr als das Befolgen eines Plans [geschätzt]“ [Man17].

Zwar finden die Befürworter des Manifests die Werte auf der jeweils rechten Seite relevant, für sie sind aber die Werte auf der jeweils linken Seite wichtiger [Man17].

Vertreter des agilen Softwareprozessparadigmas sind unter anderem XP sowie Scrum. Nachfolgend werden Grundzüge von Scrum erläutert.

3.3.1 Scrum

Scrum ist ein iterativ-inkrementelles Vorgehensmodell, das die folgenden drei Rollen vorsieht: Product Owner, Team und Scrum Master. Der Product Owner vertritt den Auftraggeber während des Entwicklungsprozesses und hat somit Einfluss auf die in der jeweiligen Iteration zu realisierenden Anforderungen. Das Team umfasst bis ca. zehn Personen und hat die Aufgabe, die geplante Software zu erstellen, wobei es für seine Organisation selbst verantwortlich ist. Der Scrum Master koordiniert die Arbeit des Teams und gewährt die Einhaltung der Prinzipien von Scrum. [Säf10, S. 382]

Der Ablauf des Entwicklungsprozesses nach Scrum ist der Abbildung 5 zu entnehmen. Der Product Owner beginnt den Entwicklungsprozess mit einer Vision der zu erstellenden Software [Säf10, S. 382]. In einem sog. Product Backlog listet der Product Owner anschließend die Anforderungen des Auftraggebers als Product Backlog Items auf und definiert ihre Prioritäten. Jeder Anforderung im Product Backlog weist das Team einen Wert zu, der den erwarteten Realisierungsaufwand darstellt. [Bal08, S. 672]

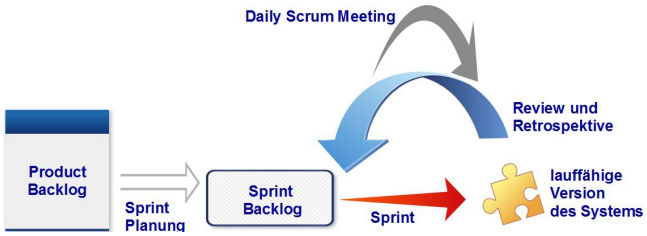

Abbildung 5. Scrum, in Anlehnung an [Bra13, S. 14]

Anhand des Product Backlogs wird ein sog. Release Plan entwickelt. In diesem Dokument werden sog. Sprints, d.h. Iterationen, mit den jeweils wichtigsten zu realisierenden Anforderungen geplant. Nach jedem Sprint kann eine Aktualisierung des Release Plans erfolgen. [Lud13, S. 231]

Während der bereits oben erwähnten Sprints findet die eigentliche Softwareentwicklung statt. Sie stellen einen essentiellen Bestandteil von Scrum dar. In einem sog. Sprint Backlog werden Anforderungen aus dem Product Backlog festgelegt, die das Team in dem jeweiligen Sprint umzusetzen hat. Dies muss innerhalb einer bestimmten Dauer erfolgen, die von dem Team nicht überschritten werden darf. In der Regel umfasst die Sprintdauer bis zu ca. dreißig Tagen. Notfalls kann das Team die zu realisierende Funktionalität beschränken, um die Frist einzuhalten. [Bal08, S. 672]

Während der Sprints findet jeden Tag eine kurze Teambesprechung, d.h. ein sog. Daily Scrum Meeting statt. Während dieses Treffens werden folgende Informationen von jedem Teammitglied mitgeteilt: seinen aktuellen Arbeitsfortschritt, eventuelle Projektschwierigkeiten sowie sein Vorhaben bis zum nächsten Daily Scrum Meeting. Auf diese Weise sollen alle Projektbeteiligten Meldungen über den aktuellen Projektstand erlangen. [Bro13, S. 101]

Mittels eines sog. Sprint Burndown Berichts wird - in der Regel grafisch – der Aufwand dargestellt, der im Rahmen des aktuellen Sprints durch das Team noch erbracht werden muss. Die Aktualisierung des Berichts erfolgt täglich. Das Dokument ermöglicht eine laufende Analyse des Entwicklungsprogresses und zeigt im Laufe des Sprints, ob die Umsetzung des Sprint Backlogs mit den vorausgesetzten Aufwänden realistisch ist. [Lud13, S. 231]

Zum Schluss jedes Sprints wird in einem sog. Review die aktuelle, lauffähige Version des Systems präsentiert. Eventuelle Anforderungsänderungen, die der Auftraggeber während des Reviews meldet, werden in das Product Backlog aufgenommen. Anschließend wird von dem Team eine sog. Retrospektive durchgeführt. Hierbei werden Probleme und Erfahrungen bezüglich des bereits realisierten Sprints besprochen, was der Realisierung der nächsten Sprints dienen soll. [Bro13, S. 101]

3.3.2 Fazit

Bei agilen Vorgehensmodellen ist es signifikant, dass sie in geringem Maße formal, und dadurch flexibel sind. Ihr Fokus liegt hauptsächlich auf Entwurfs- sowie Implementierungstätigkeiten, in deren Rahmen dann wiederum andere Aktivitäten stattfinden, wie Erhebung von Anforderung sowie Testen. Die Planung des Entwicklungsprozesses erfolgt hierbei fortlaufend – entsprechend den im Laufe der Iterationen gewonnenen Ergebnissen und Erfahrungen. [Som12, S. 55, 92]

Hiernach eignen sich agile Vorgehensmodelle für Softwareprojekte mit unpräzisen Zielen sowie mit zu erwartenden Anforderungsänderungen [Wir07, S. 39]. Maßgeblich ist hierbei die Tatsache, dass der Auftraggeber in den Entwicklungsprozess integriert wird [Som12, S. 89]. In kurzen Iterationen wird lauffähige Software entwickelt und dem Auftraggeber demonstriert. Seine Hinweise und Änderungen werden anschließend in den nächsten Iterationen berücksichtigt.

Einzelne Versionen der zu erstellenden Software werden während der Iterationen kontinuierlich getestet. Dadurch werden Fehler und Risiken früh erkannt und rechtzeitig behandelt. So wird beispielsweise anhand von regelmäßigen Integrationstests ermöglicht, noch während der Implementierung Integrationsrisiken zu identifizieren und anschließend bei Bedarf die Software entsprechend zu adaptieren.

Des weiteren besteht eine der agilen Regeln in der Einfachheit der geplanten Software, sodass nach Möglichkeit auf komplexe Elemente verzichtet wird [Som12, S. 89]. Es wird also versucht, die Funktionalität der Software ohne unnötige Funktionen zu implementieren. Dadurch wird die Eintrittswahrscheinlichkeit des Over-Engineerings reduziert.

Eine Einschränkung der agilen Vorgehensmodelle resultiert daraus, dass die Entwicklerteams relativ klein sind. So eignen sie sich für Projekte mit einer Dauer bis zu ca. einem Jahr [Bro13, S. 98].

Des weiteren wird in den agilen Vorgehensmodellen auf eine umfassende Anforderungsspezifikation verzichtet. Bei kritischen Systemen wie z.B. Flugzeugsteuerungssystemen ist es jedoch notwendig, Anforderungen präzise und systematisch zu erheben und zu analysieren [Lud13, S. 234], bevor die Entwurfs- und Implementierungsaktivitäten begonnen werden. Für die Entwicklung von solchen Systemen wären agile Vorgehensmodelle wesentlich umzuformen [Som12, S. 90].

4. Fazit

Zwar sind Risiken mit potenziellen Gefahren verbunden, auf der anderen Seite charakterisieren sie aber ebenfalls Chancen, denn ohne Risiken kann man nicht mit einem positiven Geschäftsergebnis rechnen [Ebe13, S. 7]. Auch bei der Softwareentwicklung ist dies der Fall. In jedem Softwareprojekt sind daher Risiken selbstverständlich zu erwarten.

Einen systematischen Umgang mit Risiken bei der Softwareentwicklung ermöglicht das Risikomanagement. In einem iterativen Prozess erfolgt die Risikoidentifizierung, die Risikoanalyse, die Planung von entsprechenden Maßnahmen sowie die Risikoüberwachung. Das frühe Erkennen von Risiken und die Durchführung von adäquaten Schritten steigert die Wahrscheinlichkeit, dass das Projekt erfolgreich abgeschlossen wird. Im Gegensatz dazu können Probleme, die aus nicht identifizierten und behandelten Risiken resultieren, sehr teuer sein bzw. die Projektrealisierung verhindern.

Eines der wichtigsten Risiken bildet eine unzureichende Organisation der Softwareentwicklung. Die Softwareentwicklung darf nicht ungeordnet erfolgen, sondern im Rahmen eines Prozesses. Eine abstrakte Beschreibung eines Entwicklungsprozess stellen Vorgehensmodelle dar, die einen Umsetzungs- und Organisationsrahmen für Softwareprojekte bilden.

Die Grundprinzipien der Vorgehensmodelle beschreiben die drei Softwareparadigmen: das sequentielle, das iterative sowie das agile Paradigma. Bei der Auswahl eines Vorgehensmodells sind unter anderem die Art der geplanten Software, die Größe des Projektes sowie das im Projekt geltende Auftragsverhältnis zu berücksichtigen. So ist beispielsweise ein sequentielles Vorgehensmodell für ein Projekt auf Grundlage eines öffentlich-rechtlichen Ausschreibungsverfahrens sinnvoll. Ein agiles Vorgehensmodell wiederum eignet sich gut für kleine Projekte, deren unpräzise Ziele sowie Anforderungen während des Entwicklungsprozesses noch zu spezifizieren sind.

5. Literatur

[Agl17] Manifest für Agile Softwareentwicklung, http://agilemanifesto.org/iso/de/manifesto.html, (letzter Zugriff: Januar 2017).

[Ahr08] F. Ahrendts, A. Marton, IT-Risikomanagementleben! Wirkungsvolle Umsetzung für Projekte in der Softwareentwicklung, e-ISBN 978-3-540-30025-0,Springer-Verlag 2008.

[Aßm06] U. Aßmann, B. Demuth, F. Hartmann, Risiken in der Softwareentwicklung, Wissenschaftliche Zeitschrift der Technischen Universität Dresden, 55**,** 2006, Heft 3 – 4.

[Bal08] H. Balzert, Lehrbuch der Softwaretechnik: Softwaremanagement, ISBN 978-3-8274-1161-7**,** Spektrum Akademischer Verlag 2008.

[Bal09] H. Balzert, Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, ISBN 978-3-8274-1705-3, Spektrum Akademischer Verlag 2009.

[Bal11] H. Balzert, Lehrbuch der Softwaretechnik: Entwurf Implementierung, Installation und Betrieb, ISBN 978-3-8274-1706-0, Spektrum Akademischer Verlag Heidelberg 2011.

[Boe88] B. W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer. Vol. 21, Ausg. 5, Mai 1988, S. 61-72.

[Bra13] J. Brandstäter, Agile IT-Projekte erfolgreich gestalten. Risikomanagement als Ergänzung zu Scrum, ISBN 978-3-658-04430-5, Springer Fachmedien 2013.

[Bro13] M. Broy, M. Kuhrmann, Projektorganisation und Management im Software Engineering, ISBN 978-3-642-29290-3, Springer-Verlag 2013.

[BWE15] Bundesministerium für Wirtschaft und Energie, Monitoring-Report Wirtschaft DIGITAL 2015.

[Ebe13] Ch. Ebert, Risikomanagement kompakt. Risiken und Unsicherheiten bewerten und beherrschen, ISBN 978-3-642-41048-2, Springer-Verlag 2006, 2013.

[Han15] H. R. Hansen, J. Mendling, G. Neumann, Wirtschaftsinformatik. Grundlagen und Anwendung, e-ISBN 978-3-11-033529-3, Walter de Guyter GmbH 2015.

[Höl16] M. Hölzl, Matthias, J. Rinke, Vorlesung Softwaretechnik, Softwareprozesse und Vorgehensmodelle, LMU München 2016, http://www.pst.ifi.lmu.de/Lehre/wise-15-16/swtechnik/softwareprozesse, (letzter Zugriff: Mai 2016).

[Lud13] J. Ludewig, H. Lichter, Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, ISBN 978-3-86491-298-6, dpunkt.verlag GmbH 2013.

[Säf10] W. Schäfer, Softwareentwicklung: Einstieg für Anspruchsvolle, ISBN 978-3-8273-2851-9, Addison Wesley 2010.

[Som12] I. Sommerville, Software Engineering, ISBN 978-3-86894-099-2, Pearson Deutschland GmbH 2012.

[Wal04] E. Wallmüller, Risikomanagement für IT- und Software-Projekte. Ein Leitfaden für die Umsetzung in der Praxis, ISBN 3-446-22430-0, Carl Hanser Verlag 2004.

[Wal14] E. Wallmüller, Risiko- und Chancen-Management für IT- und Software-Projekte. Ein Leitfaden für die Umsetzung in der Praxis, ISBN 978-3-446-43477-6, Carl Hanser Verlag 2014.

[Wir07] M. Wirsing, Koch, Störrle, Vorlesung „Methoden des Software Engineering“, Block E „Software-Prozess und Projektmanagement“, Vorgehensmodelle, LMU München 2007, http://www.pst.ifi.lmu.de/lehre/WS0607/mse/material/22-MSEE1-Vorgehen.pdf, (letzter Zugriff: Mai 2016).