4 Konzept

Inhaltsangabe

Wie in Abschnitt 1.1 beschrieben, ist die zentrale Idee dieser Diplomarbeit, bereits vorhandene Daten aus Change-Request-Systemen zu analysieren und visualisieren, um Softwareentwicklungs-Prozesse zu verbessern. In Kapitel 3 wurde deutlich gemacht, inwiefern dieser Ansatz bereits in anderen Arbeiten angegangen wurde. Dieses Kapitel beschreibt die Konzepte hinter dieser Arbeit und ihre Auswirkungen auf die zu erstellende Werkzeugunterstützung *River*.

Nach einer Beschreibung der Vorgehensweise dieser Arbeit folgen Überlegungen über die Interessen und Ziele der Stakeholder im Kontext der Softwareentwicklungs-Prozessverbesserung, sowie über die Idee, die Visualisierung eines Softwareentwicklungs-Prozesses analog zur Visualisierung eines industriellen Prozesses in einer Messwarte eines Chemiebetriebes zu gestalten. Daraus werden Anforderungen an die Werkzeugunterstützung erhoben, die anschließend iterativ in zwei Prototypen validiert, ergänzt und auch verworfen werden. Das Kapitel schließt mit einer Auflistung der konsolidierten Anforderungen für das endgültige Werkzeug.

4.1 Vorgehensweise

In diesem Abschnitt wird die Vorgehensweise beschrieben, nach der diese Diplomarbeit bearbeitet wurde. Sie orientiert sich stark an der Methodik, die Cook, Votta und Wolf zur Analyse und Auswertung von Daten bereits vorhandener Software-Systeme vorschlagen[CVW98]:

1. *Verstehen der Organisation, des Projekts oder des Prozesses*: Was sind die Charakteristika des zu untersuchenden Prozesses? Welche Personen innerhalb der Organisation interessieren sich für die Ergebnisse der Prozess-Analyse? Mit diesen Fragen beschäftigen sich die beiden folgenden Abschnitte: Abschnitt 4.2 und Abschnitt 4.3. Insbesondere tragen jedoch die Evaluierungen der beiden Prototypen und des finalen Werkzeugs zum Verständnis der dazustellenden Prozesse bei.

2. *Identifikation der möglichen Datenquellen*: Aus welchen im Prozess verwendeten Software-Systemen können die den Metriken zugrundeliegenden Daten erhoben werden? Dieser Frage widmet sich der gleichnamige Abschnitt 4.4.

3. *Identifikation der Metriken, die den Erfolg des Prozesses messen*: Metriken, die den Erfolg eines Prozesses messen oder vorhersagen können, sind von besonderem Interesse bei der Analyse und Optimierung von Prozessen. Gleichzeitig sind insbesondere Vorhersage-Metriken in der Regel nicht über direkte und triviale Messungen erhebbar. Stattdessen werden Hypothesen über ihre Relation zu berechenbaren Metriken aufgestellt und geprüft. Dieser Punkt in der Methodik von Cook et al. wird innerhalb dieser Arbeit nicht bearbeitet und ist ggf. Gegenstand einer Folgearbeit. Diese Diplomarbeit legt den Fokus auf die visuelle Präsentation der Metriken.

4. *Identifikation der aus den Datenquellen berechenbaren Metriken*: Wie können die erhobenen Daten weiterverarbeitet und zu Informationsquellen über den Zustand der Prozesse verdichtet werden?

5. *Extraktion der Daten und Durchführung der Analyse*: Dieser Punkt sieht die Entwicklung von Programmmodulen zur Datenerhebung aus den vorhanden Software-Systemen sowie von Algorithmen zur Berechnung der Metriken vor.

6. *Interpretation der Ergebnisse*: Welche Schlüsse können aus den berechneten und dargestellten Metriken gezogen werden? Inwiefern tragen sie zur Verbesserung des Prozesses bei?

Punkt 3 (Identifikation der Metriken, die den Erfolg des Prozesses messen) der Methodik von Cook et al. wird innerhalb dieser Arbeit nicht bearbeitet und ist ggf. Gegenstand einer Folgearbeit. Diese Diplomarbeit legt den Fokus auf die Punkte 4-6 der Methodik und setzt diese anhand der erstellten Prototypen und der Werkzeugunterstützung *River* sowie den zugehörigen Evaluationen um.

4.2 Stakeholder

Als Stakeholder, d.h. Personengruppen, die ein besonderes Interesse an der Optimierung des Softwareentwicklungs-Prozesses haben, wurden Softwareprozess-Manager und Projektleiter identifiziert. In diesem Abschnitt werden ihre jeweiligen Aufgabenbereiche und Ziele beschrieben, um daraus Anforderungen für die Werkzeugunterstützung *River* abzuleiten.

Softwareprozess-Manager

Die Aufgabe des Softwareprozess-Managers ist der Entwurf, die Einführung, die Überwachung und kontinuierliche Optimierung von projektübergreifenden und organisationsweit standardisierten Softwareentwicklungs-Prozessmodellen[1]. In konkreten Projekten unter-

[1] `http://www.heise.de/developer/artikel/Erfolgreiche-Einfuehrung-von-Business-Process-Management-BPM-1715608.html` abgerufen am 25.05.2013

stützt er bei der ggf. notwendigen Anpassung des Prozessmodells an die projektspezifischen Anforderungen. Um diese Aufgaben, insbesondere die Optimierung, zu erfüllen, benötigt er Sichten auf die Umsetzung der verwendeten Prozesse.

Projektleiter

Der Projektleiter hat die Aufgabe, ein Software-Projekt erfolgreich durchzuführen, d.h. die Projektziele im Rahmen des angesetzten Zeit- und Kostenbudgets zu erreichen. Dazu wird das Projekt gemäß des verwendeten Softwareentwicklungs-Prozesses bearbeitet. Treten in dem Prozess unerwartete Abläufe auf, die die erfolgreiche Durchführung des Projekts gefährden oder behindern, ist es die Aufgabe des Projektleiters, gegenzusteuern. Auch er benötigt dazu Informationen über die Umsetzung des gewählten Prozesses.

Die Ziele des Softwareprozess-Managers sind eher langfristiger/strategischer Natur: Erfahrungen aus der Umsetzung eines Prozesses in einem Projekt fließen in die Verbesserung des organisationsweit standardisierten Prozesses zurück. Zur Überprüfung der getroffenen Verbesserungsmaßnahmen eignen sich insbesondere Trend-Darstellungen der Prozess-Eigenschaften, die die frühere Umsetzung des Prozesses der jetzigen gegenüber stellt.

Das wichtigste Ziel des Projektleiters - der Erfolg des Projekts - ist im Vergleich dazu kurzfristig. Weitere Ziele, wie der Aufbau von Erfahrungen bei der Umsetzung von Projekten, um ein nachfolgendes Projekt besser zu leiten, sowie die Kontrolle und Beobachtung dieses Fortschritts, sind zwar auch wichtig, gemessen am Erfolg des aktuellen Projekts aber zweitrangig. Daher bieten sich initial für Projektleiter Darstellungen an, die den aktuellen Zustand des Softwareentwicklungs-Prozesses charakterisieren. Daraus kann er ggf. notwendige Steuerungsmaßnahmen ableiten. Im Anschluss daran sind, ähnlich wie beim Softwareprozess-Manager, Trend-Darstellungen interessant, um Auskunft über die Wirksamkeit der Maßnahmen zu erlangen.

4.3 Leitstand

In industriellen Anlagen wie etwa Chemieanlagen oder Kraftwerken werden zur Überwachung der dort ablaufenden Prozesse Leitstände verwendet. Diese SCADA (Supervisory-Control-And-Data-Acquisition)-Systeme greifen auf Sensoren (z.B. Temperatursensoren) zu, um den Zustand des Prozesses zu messen. Ihre Daten werden von einem Prozessüberwachungssystem erfasst und als Kontrollvariable auf einer Anzeige oder einem Bildschirm des Leitstandes visualisiert. Zusätzlich findet meist eine automatische Überwachung der Kontrollvariablen statt. Dazu wird der Wertebereich der Kontrollvariablen in vier Kategorien unterteilt:

Normal

Die Kontrollvariable befindet sich innerhalb ihres Soll-Wertebereichs. Es sind keine Korrekturmaßnahmen erforderlich.

1. Warnstufe

Die Kontrollvariable befindet sich leicht außerhalb ihres Soll-Wertebereichs. Durch Regeln definierte Automatismen (wie z.B. das Ausschalten einer Heizung bei Überschreiten einer Höchsttemperatur) dienen dazu, die Variable wieder in die Kategorie “Normal” zu überführen. Alternativ greift ein Bediener in den Prozess ein. Es ist noch kein Schaden an der Anlage oder dem Produkt entstanden.

2. Warnstufe

Die Kontrollvariable weicht stark von ihrem Soll-Wertebereich ab. Schaden an der Anlage oder dem Produkt steht unmittelbar bevor. Hier greifen in der Regel automatische Notfall-Korrekturmaßnahmen, um fehlendes oder fehlerhaftes menschliches Eingreifen zu kompensieren, Schaden zu vermeiden und den Prozess wieder in einen sicheren Zustand zu führen.

Schaden entstanden

Die Kontrollvariable hat die 2. Warnstufe zeitlich dauerhaft überschritten und die Korrekturmaßnahmen waren erfolglos. Es ist Schaden an der Anlage oder dem Produkt entstanden.

Außerdem ermöglichen es SCADA-Systeme dem Bediener, steuernd in den Prozess einzugreifen. Er kann, wenn erlaubt, z.B. Grenzwerte für Kontrollvariablen festlegen oder Elemente des kontrollierten Systems wie etwa eine Pumpe manuell ein- und ausschalten.

Ein wesentliches Merkmal eines Leitstands ist die Zentralisierung der Überwachung und Steuerung. Er ersetzt bzw. ergänzt die Anzeigen vor Ort (etwa die Temperatur- und Druckanzeigen direkt an einem Kessel) und ermöglicht die ferngesteuerte Einflussnahme auf den Prozess. Muss der Messwart in den Prozess eingreifen, werden so Latenzzeiten reduziert bzw. vermieden, die z.B. entstehen würden, wenn er manuell ein Ventil öffnen oder schließen müsste.

Der Leitstand stellt die Elemente und Arbeitsschritte des Prozesses ikonisch dar und setzt sie logisch, zeitlich und/oder örtlich zueinander in Beziehung. So wird es dem Benutzer, der mit dem Prozess vertraut ist, erleichtert zu erkennen, welches Element der Visualisierung welchem Element des Prozesses entspricht.

Abbildung 4.1 zeigt ein typisches SCADA-System am Beispiel eines industriellen Prozesses. Die Darstellung ist darauf ausgelegt, den realen Prozess in einem Fließbild mit Prozessvariablen wiedererkennbar und intuitiv abzubilden. Selbst ein Laie erkennt schnell, ohne ein Handbuch konsultieren zu müssen, dass in der Mitte der Darstellung die Durchschnittstemperaturen einer Flüssigkeit in einem Kessel in 3 verschiedenen Höhen angezeigt werden. Daneben befinden sich die eingestellten Grenzwerte für das Ein- und Ausschalten der rechts dargestellten Wärmepumpe. Unten im Bild gibt es Knöpfe zur Konfiguration der Stellgrößen des Prozesses und zur Navigation zu Detailansichten sowie zur Visualisierung anderer Teile des Prozesses. Einen Prozess-Experten versetzt eine solche Darstellung in die Lage, auf einen Blick zu erkennen, ob sich der Prozess wie gewünscht verhält, oder ob und welche Korrektur-Maßnahmen erforderlich sind.

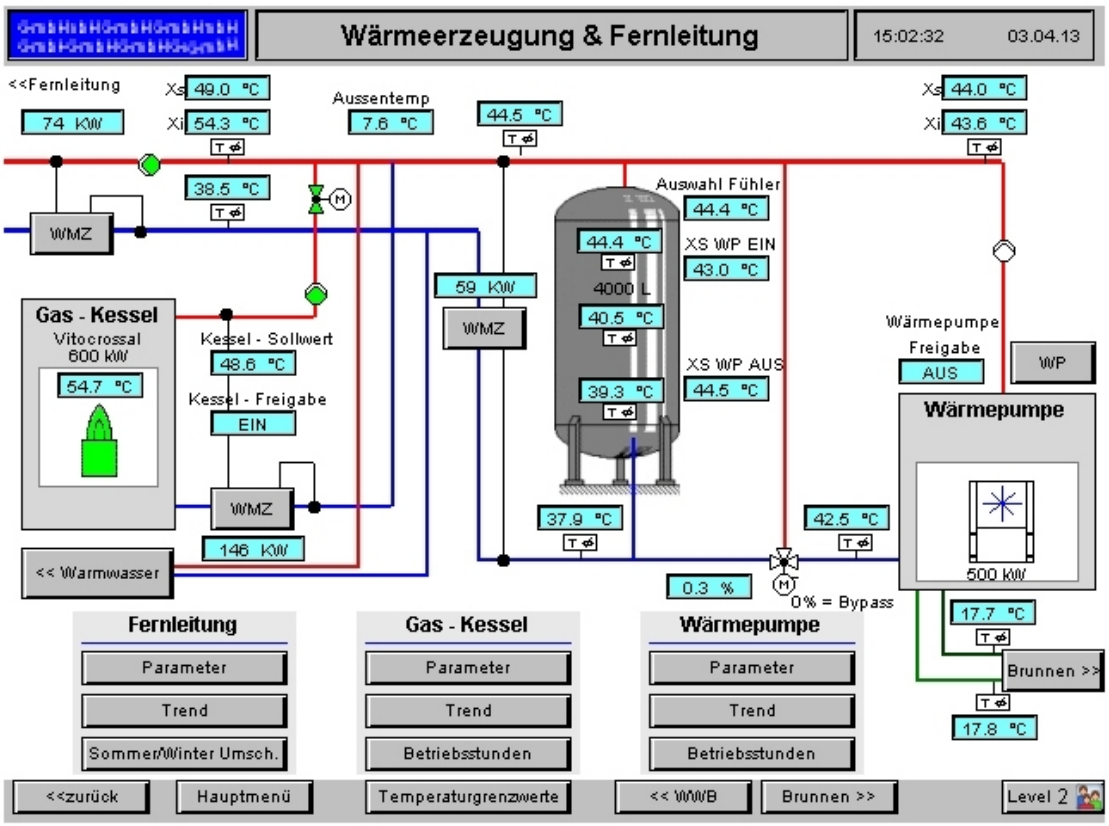

Abbildung 4.1: Schaltzentrale eines industriellen Prozesses
Quelle: `http://www.heise.de/security/meldung/Kritische-Schwachstelle-in-hunderten-Industrieanlagen-1854385.html`
abgerufen am 26.05.2013

Leitstände im Kontext von Softwareentwicklungs-Prozessen

Leitstände sind eine lange bewährtes Mittel zur Überwachung und Steuerung von industriellen Prozessen. Kann ein Softwareentwicklungs-Prozess auf ähnliche Art überwacht und gesteuert werden? Die Beantwortung dieser Frage erfordert ein genaues Verständnis, was eigentlich überwacht und gesteuert werden soll. Daher zunächst ein paar Begriffsdefinitionen:

Prozess: ein "sich über eine gewisse Zeit erstreckender Vorgang, bei dem etwas [allmählich] entsteht, sich herausbildet"[2].

Softwareentwicklungs-Prozess: The process by which user needs are translated into a software product.[ISO10]

Der Softwareentwicklungs-Prozess gliedert sich weiter auf in Phasen und Aktivitäten, an deren Ende jeweils Zwischenergebnisse stehen. Diese werden in späteren Phasen und Aktivitäten weiterverarbeitet, bis schließlich das Endprodukt entstanden ist.

Überwachung

Abbildung 4.2 zeigt einen Softwareentwicklungs-Prozess am Beispiel der Scrum-Methode[Abr02]. Die Aufteilung in Phasen ist durch senkrechte gestrichelte Linien gekennzeichnet. Die Aktivitäten befinden sich in rechteckigen Kästen und die Zwischenergebnisse sind durch Kästen mit geschwungenem unteren Rand dargestellt. Pfeile verdeutlichen die zeitliche Abfolge von Aktivitäten und stellen dar, welches Zwischenergebnis aus welcher Aktivität resultiert.

Diese Visualisierung eines Softwareentwicklungs-Prozesses ähnelt stark dem statischen Fließbild einer SCADA-Visualisierung. Es fehlt lediglich eine Anzeige von Kennwerten an und zwischen den angezeigten Prozess-Elementen, die Auskunft über den Zustand bzw. die Umsetzung der repräsentierten Phasen, Aktivitäten oder Zwischenergebnisse geben.

Ein Beispiel: Im Rahmen des "Sprint Planning Meetings" wird das "Sprint Backlog" mit denjenigen Funktionalitäten (in Form von "User-Stories") der zu entwickelnden Software befüllt, die im nächsten Sprint umgesetzt werden sollen. Diese Funktionalitäten sind jeweils mit einer Priorität und einer Aufwandsschätzung versehen. Der Prozess sieht vor, dass während des ein- bis vierwöchigen Sprints alle Entwickler gemeinsam an der Umsetzung der User-Story mit der höchsten Priorität arbeiten.

Um den Fortschritt und die Einhaltung des Prozesses während eines Sprints zu überwachen, könnte die Darstellung des Scrum-Prozesses um eine Angabe über den Zustand des "Sprint Backlogs" angereichert werden (siehe Abbildung 4.3). Die Darstellung erinnert an einen Kessel aus dem langsam Flüssigkeit abgelassen wird. Die User-Stories sind durch Abschnitte innerhalb dieser "Flüssigkeit" repräsentiert, wobei die Höhe eines solchen Abschnitts die Aufwandsabschätzung wiedergibt. Darüber hinaus sind sie nach Priorität sortiert und eingefärbt. Oben befindet sich die User-Story mit der höchsten (rot), unten die mit der niedrigsten Priorität (grün). Eine im Verlauf des Sprints von oben nach unten wandernde horizontale Linie zeigt den Fortschritt der Zeit an. Erledigte

[2] `http://www.duden.de/rechtschreibung/Prozess` abgerufen am 26.05.2013

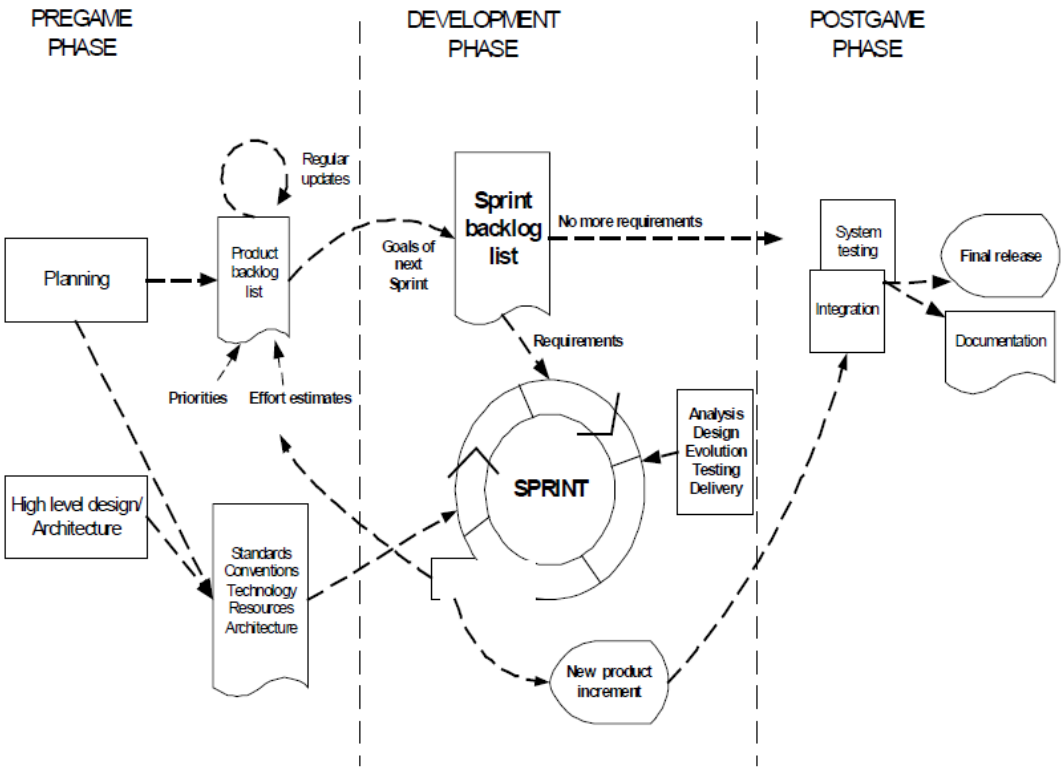

Abbildung 4.2: Der Scrum Prozess
Quelle: [Lic12]

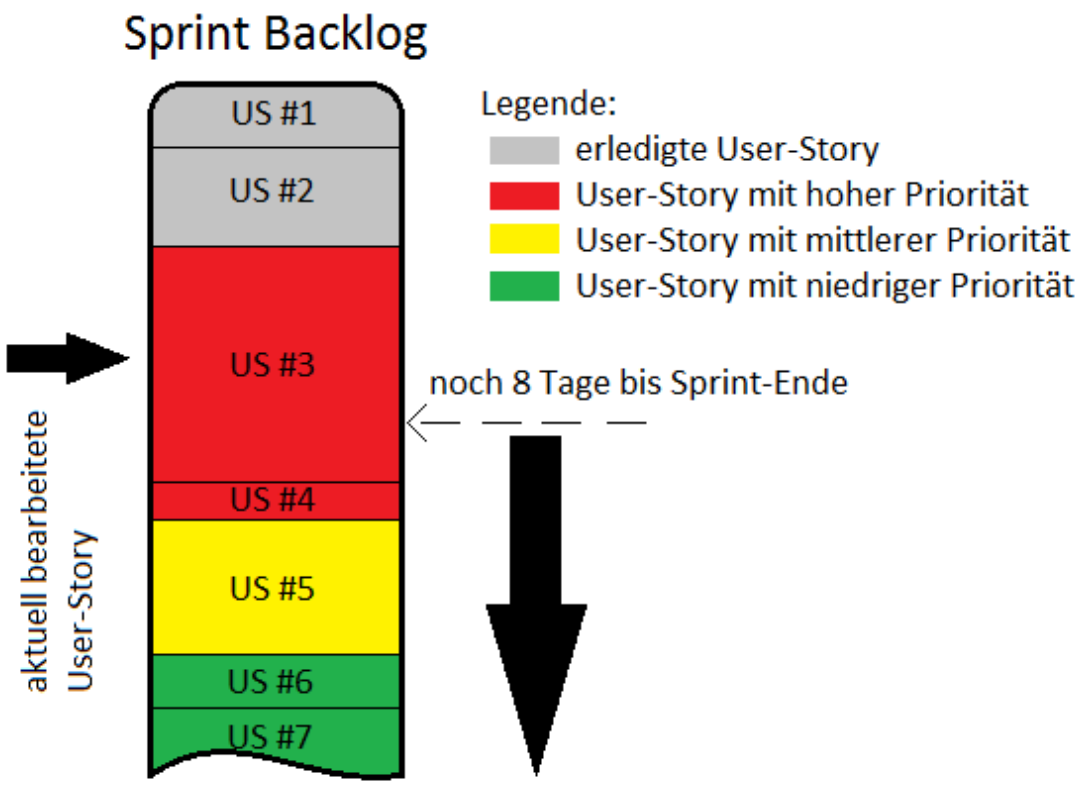

Abbildung 4.3: Backlog mit Kennwerten

User-Stories werden aus dem “Kessel” entfernt, indem sie grau dargestellt werden. Die aktuell bearbeitete User-Story ist mit einem Pfeil markiert.

Anhand dieser Visualisierung kann der Fortschritt des Sprints mit den Aufwandsabschätzungen der User-Stories auf einen Blick abgeglichen werden. Liegt der graue Teil der “Flüssigkeit” komplett über der Zeitlinie, und der farbige Teil komplett unter ihr, befindet sich der Sprint genau im Zeitplan. Bei einem farbigen Anteil oberhalb der Zeitlinie werden die User-Stories zu langsam abgearbeitet, und bei einem grauen Anteil unterhalb der Zeitlinie schneller als erwartet. Im Beispiel ist der Bearbeitungs-Fortschritt einer einzelnen User-Story nicht feingranular mit einem Prozentwert aufgeschlüsselt. Das Scrum-Team liegt hier also im Plan, sollte aber in Kürze die Bearbeitung von User-Story #3 fertigstellen.

Die Einhaltung des Prozesses ist ebenfalls erkennbar. Es sollte sich jeweils nur eine einzige User-Story (und nicht mehrere gleichzeitig) in Bearbeitung befinden, d.h. mit einem Pfeil markiert sein. Außerdem sollte es nur oben einen grauen Bereich geben, während der farbige Bereich unten liegt. Ist dies nicht der Fall und der farbige Bereich von grauen Anteilen durchsetzt, bedeutet dies, dass die User-Stories nicht entsprechend der im “Sprint Planning Meeting” festgelegten Priorität bearbeitet werden.

Steuerung

Dieses Beispiel zeigt, dass die Idee einer von industriellen Leitständen inspirierten Visualisierung und Überwachung von Softwareentwicklungs-Prozessen grundsätzlich umsetzbar ist. Wie aber sieht es mit der Steuerung des Entwicklungs-Prozesses aus? Können wie im obigen SCADA-Beispiel Knöpfe und Regler eingefügt werden, mit denen direkt Einfluss auf den Prozess genommen werden kann? Die Antwort lautet hier leider “Nein”. Im Gegensatz zu Prozessen in industriellen Anlagen sind die Maßnahmen, die als Reaktion einer Abweichung vom gewünschten Prozess-Verlauf zu treffen sind, im Kontext von Softwareentwicklungs-Prozessen in der Regel nicht technischer Natur und können somit auch nicht von einem technischen System umgesetzt werden. Was ist zu tun, wenn die obige Visualisierung des Scrum-Backlogs anzeigt, dass der Zeitplan im Sprint nicht eingehalten wird? Der “ScrumMaster” hat dann die Aufgabe, die Ursache der Prozessabweichung zu ermitteln und zu beseitigen. Eine mögliche Ursache wäre, dass die Aufwandsabschätzungen, die die Scrum-Teilnehmer den User-Stories zugeordnet haben, zu niedrig ausgefallen sind. In der Folge wurden zu viele User-Stories in das Sprint-Backlog übernommen, so dass es nicht gelingt, ihn innerhalb des Sprints vollständig abzuarbeiten. Mögliche Maßnahmen sind:

- Verlängerung des Sprints
- Verschiebung von User-Stories mit niedriger Priorität auf den nächsten Sprint
- Mehr Mitarbeiter zur Bewältigung der Aufgaben einsetzen (dies ist aber durch die kurze Dauer eines Sprints oft nicht zielführend. Stichwort: Einarbeitungszeit)
- Beim nächsten Sprint Planning Meeting besonderes Augenmerk auf die Aufwandsabschätzungen legen

Offensichtlich können diese Maßnahmen nicht einfach durch das Setzen einer technischen Stellgröße bewerkstelligt werden. Stattdessen trifft der ScrumMaster gemeinsam

mit dem Kunden, dem Product Owner und/oder den Scrum-Teilnehmern Entscheidungen bzw. weist auf die Prozessabweichung hin, um dann gemeinsam gegenzusteuern.

Zusammenfassung

Aus den obigen Überlegungen, wie Leitstände als Vorbild für die Analyse und Optimierung von Softwareentwicklungs-Prozessen genutzt werden können, ergibt sich der Fokus dieser Diplomarbeit: Der Überwachungs- bzw. Analyseanteil eines Leitstands lässt sich gut auf Softwareentwicklungs-Prozesse übertragen. Insbesondere eignet er sich zur Realisierung und Evaluierung einer Werkzeugunterstützung.

Der Steuerungsanteil hingegen findet vor allem über werkzeugferne Kanäle, wie Kommunikation unter den Mitarbeiten und organisatorischen Maßnahmen, statt. Eine gewisse Integration in ein Werkzeug ist zwar möglich, wie etwa die Benachrichtigung von Stakeholdern bei wichtigen Ereignissen im Prozess. Der wesentliche Anteil der Steuerung ist aber nicht automatisierbar. Daher wird der Steuerungsanteil im weiteren Verlauf dieser Diplomarbeit ausgeblendet und nicht weiter betrachtet.

4.4 Identifikation der Datenquellen

Dieser Abschnitt stellt die verschiedenen Datenquellen vor, die als Grundlage für die Analyse von Softwareentwicklungs-Prozessen in Frage kommen. Aufgrund der initialen Annahme dieser Diplomarbeit, dass sich die Prozesse in den verwendeten Werkzeugen und den von ihnen erzeugten Daten widerspiegeln, liegt der Fokus genau auf denjenigen Werkzeugen, die in praktisch jedem Softwareentwicklungsprojekt verwendet werden. Diese werden im Folgenden beschrieben und auf ihre Eignung als Datenquelle untersucht.

Zeit- und Kostenerfassungssysteme

In der Zeit- und Kostenerfassung notieren Mitarbeiter den von ihnen erbrachten Zeitaufwand und entstandene Kosten und ordnen sie Tätigkeiten und Arbeitspaketen zu. Auf Basis dieser Daten lassen sich leicht Aussagen über den Prozess treffen. In welchen Phasen und Aktivitäten entstehen besonders hohe Kosten? Wo weicht der tatsächliche Zeitaufwand von der Planung ab? Dieser Themenbereich ist allerdings bereits ausgiebig erforscht und mit Werkzeugunterstützung versehen. Als Beispiel sei hier die Earned-Value-Analyse[Lic12] genannt, die etwa mit dem verbreiteten Projektmanagement-Werkzeug MS-Project durchgeführt werden kann[3].

Versionsverwaltung

“Eine Versionsverwaltung ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird. Alle Versionen werden in einem Archiv mit Zeitstempel und Benutzerkennung gesichert und können später wiederhergestellt werden.”[4] Oft sind einer Änderung auch weitere Informationen zugeordnet. Dies kann eine Commit-Nachricht in Form eines Freitextes sein, die eine unstrukturierte Beschreibung

[3] `http://office.microsoft.com/en-us/project-help/analyze-project-performance-with-earned-value-analysis-HA101819808.aspx` abgerufen am 27.05.2013

[4] `https://de.wikipedia.org/wiki/Versionsverwaltung` abgerufen am 27.05.2013

der Änderung darstellt. Aber auch strukturierte Informationen wie die Zuordnung der Änderung zu einem bearbeiteten Änderungswunsch bzw. behobenen Fehler sind üblich. Anhand dieser Daten können vor allem Metriken erstellt werden, die die Software selbst vermessen. Z.B. könnte eine Metrik darstellen, in welchen Dateien, Modulen oder Komponenten besonders viele Fehler behoben wurden. Erfahrungsgemäß gilt für Softwarefehler das Pareto-Prinzip: Insbesondere dort wo viele Fehler gefunden wurden, die Fehlerdichte also hoch ist, sind weitere Fehler zu erwarten[CJ02]. Eine solche Metrik liefert also Hinweise, welche Teile der Software besonders intensiv getestet oder mit Reviews geprüft werden sollten.

Bezüglich des Softwareentwicklungs-Prozesses scheint die Aussagekraft der in Versionsverwaltungen enthaltenen Daten allerdings begrenzt zu sein. Welche Aussage kann über den Prozess getroffen werden, wenn die Fehlerdichte in einer Komponente niedrig ist? Zwei gegensätzliche Interpretationen liegen nahe:

1. Einerseits könnten die Fehlervermeidungsmaßnahmen, die der Prozess vorsieht, die Fehlerdichte erfolgreich niedrig gehalten haben. So könnte beispielsweise das Anforderungsdokument eine hohe Qualität besitzen und klar definierte Schnittstellen und Funktionalitätsbeschreibungen enthalten, wodurch Missverständnisse und Fehlinterpretationen in Design und Implementierung vermieden wurden.

2. Eine andere Möglichkeit ist eine schlechte Testabdeckung und/oder ungenügende Reviews. Es existieren zwar Fehler, diese werden aber nicht gefunden.

Diese beiden Interpretationen betreffen unterschiedliche Aspekte des Prozesses und bewerten sie gegensätzlich. Der Zusammenhang zwischen den Metrikmesswerten und der Schlussfolgerung durch die Interpretation ist hier also fragwürdig.

Change-Request-Systeme

Change-Request-Systeme wurden bereits in Abschnitt 2.3 näher beschrieben. Die Daten in Change-Request-Systemen korrelieren stark mit den Aktivitäten eines Softwareentwicklungs-Prozesses. Am Beispiel von Scrum werden User-Stories weiter zu kleineren Aufgaben (Tasks) aufgespaltet, die wiederum in das Change-Request-System in Form von Tickets eingetragen werden. Die Bearbeitung dieser Tasks wird durch eine Serie von Aktivitäten im Prozess vorangetrieben, im Einzelnen sind dies Analyse, Design, Evolution, Testing und Delivery (vgl. Abbildung 4.2). Der Fortschritt der Bearbeitung dieser Tasks spiegelt sich direkt im Status der Tickets wider. Auf Basis dieser Daten kann etwa ein stockender Sprint schnell erkannt werden. Über eine Auswertung der zu einer Task gehörenden Aufwandsabschätzung und den Abgleich mit dem Lebenszyklus einer Task (Zeit zwischen dem Status “neu” bis zum Status “abgeschlossen”) kann z.B. eine Metrik definiert werden, die die Güte der Aufwandsabschätzungen bewertet. Aus den gleichen Daten kann etwa auch die Teamvelocity errechnet werden, eine Größe mit deren Hilfe die Anzahl und der Umfang der User-Stories, die ein Team innerhalb eines Sprints sinnvoll bearbeiten kann, vorhergesagt wird.

Die genannten Beispiele stellen Informationen über den Zustand und den Ablauf des Prozesses zur Verfügung bzw. stellen Daten zur Verfügung, die zur Optimierung der Prozessumsetzung nützlich sind. Insgesamt ist die Datenquelle Change-Request-Systeme

damit die vielversprechendste und dient als Grundlage für die Metriken, die im während dieser Diplomarbeit entstandenen Werkzeug *River* visualisiert werden.

4.5 Erster Prototyp

Ausgehend von den gesetzten Zielen der Arbeit, den Überlegungen zu industriellen Leitständen und den Zielen und Interessen der Stakeholder wurde mit einem ersten Prototyp untersucht, wie ein GUI für eine Werkzeugunterstützung bei der Analyse und Verbesserung von Softwareentwicklungs-Prozessen aussehen könnte. Darüber hinaus wurden mit dem Prototyp erste Experimente mit den Technologien Trac, Python und JavaEE unternommen. Ein dritter Zweck des Prototyps ist die Erkundung der Datenerhebung: Wie können Daten aus dem im Prozess verwendeten Change-Request-System extrahiert werden?

In diesem Abschnitt werden daher zunächst funktionale wie auch nicht-funktionale Anforderungen an den Prototyp entwickelt. Diese Anforderungen werden anschließend anhand des erstellten Prototyps hinsichtlich ihrer Zweckmäßigkeit und Umsetzbarkeit überprüft. Die Anforderungen sind zwecks Referenzierbarkeit jeweils mit einem Kategoriekürzel und einer fortlaufenden Nummer markiert. Das Kategoriekürzel “RB” zeichnet eine Randbedingung aus, die Einschränkungen für den Entwurf definiert. “Q” steht für Qualitätsanforderung. In dieser Diplomarbeit werden hierunter vor allem Anforderungen an die Benutzbarkeit des Werkzeugs gestellt. “RB” und “Q” bilden zusammen die nicht-funktionalen Anforderungen. Eine mit “FA” markierte Anforderung ist eine funktionale Anforderung. Sie definiert, was das System tun soll.

- (RB1) Um den Einsatz und die Evaluierung des Werkzeugs in existierenden Softwareentwicklungs-Projekten zu erleichtern und eine hohe Akzeptanz zu erreichen, soll das Werkzeug aus Sicht der Software-Entwickler eines Projekts “nicht-störend” sein (vgl. [Joh01]. D.h. es soll von den Entwicklern keinen über ihre normalen Tätigkeiten hinausgehenden zusätzlichen Aufwand erfordern, wie etwa die manuelle Eingabe von Informationen über ihre Aktivitäten im Rahmen des Softwareentwicklungs-Prozesses.

- (Q1) Die Einarbeitungszeit in das Werkzeug soll für mit dem Softwareentwicklungs-Prozess vertraute Personen kurz sein.

- (Q2) Der Konfigurationsaufwand vor und während der Benutzung des Werkzeugs durch die Stakeholder soll gering sein.

- (Q3) Die Visualisierung soll übersichtlich und leicht verständlich sein.

- (FA1) Das Werkzeug soll den Benutzer bei der Analyse des Softwareentwicklungs-Prozesses unterstützen. Dazu zeigt es Informationen über den Prozess-Zustand an und hilft, Abweichungen vom gewünschten Prozess zu erkennen und zu bewerten.

- (FA2) Das Werkzeug soll Daten aus Change-Request-Systemen erheben, weiterverarbeiten und visualisieren.

Die genannten Anforderungen sind noch sehr grobgranular. Es ist aber gerade Zweck und Ergebnis des explorativen Prototyps, diese zu verfeinern. Des Weiteren lassen die Anforderungen bzgl. ihrer Überprüfbarkeit zu wünschen übrig. Da es sich bei diesem Werkzeug aber nicht um ein Projekt handelt, an dessen Ende ein Kunde anhand überprüfbarer Eigenschaften über Abnahme oder Ablehnung entscheidet, wird dies hier in Kauf genommen. An die Stelle der Abnahme tritt die Evaluierung, die Aufschluss darüber gibt, welche Metriken des Werkzeugs von den Evaluierungs-Teilnehmern als zweckmäßig zur Prozessanalyse und -optimierung bewertet werden. Die Formulierung der Anforderungen dient primär als Grundlage für Designentscheidungen in den Prototypen und dem endgültigen Werkzeug.

Beschreibung des Prototyps

Im Folgenden werden die aus den Anforderungen gefolgerten Designentscheidungen zum ersten Prototyp erläutert. Eine Beschreibung der technischen Realisierung befindet sich im Anhang A.1.

Um der Anforderung (Q1) nachzukommen, soll die Visualisierung im ersten Prototyp den Grundsatz der Wiedererkennbarkeit des Prozesses erfüllen. Wie in Abschnitt 4.4 beschrieben, korrelieren die Status-Felder von Tickets in Change-Request-Systeme gut mit den Aktivitäten im Prozess. Im ersten Prototyp wurde daher der Workflow des Change-Request-Systems, in Form eines gerichteten Graphen, als Basis für die Visualisierung gewählt. Die Knoten in diesem Graph repräsentieren die verschiedenen Status des Workflows, die Kanten die möglichen Übergänge von einem Status in einen anderen. Abbildung 4.4 zeigt die Oberfläche der ersten und einzigen[5] Version des ersten Prototyps anhand des Standard-Workflow eines Trac Issue Tracking Systems.

Moderne Change-Request-Systeme wie Trac, Jira oder Redmine stellen einerseits Standard-Workflows zur Verfügung, bieten andererseits aber die Möglichkeit, eigene Workflows zu definieren und zu verwenden. Dies dient dazu, das Change-Request-System an den Prozess anzupassen, in dem es verwendet wird. Wegen (Q2) soll vor der ersten Verwendung des Prototypen mit einem angepassten Workflow nicht zunächst manuell ein Graph zu zeichnen sein. Stattdessen verwendet der Prototyp einen Auto-Layouting-Algorithmus, der den Workflow-Graph in der üblichen Darstellung $G = (V, E)$ entgegennimmt, und nach einer Layout-Strategie den Graphen zeichnet.

In einem weiteren Schritt (d.h. weiteren Versionen des ersten Prototyps) sollte ursprünglich[5] dieser Graph um Informationen angereichert werden, die den Zustand des durch das Change-Request-System widergespiegelten Prozesses charakterisieren. Bei den Knoten wurde dabei z.B. an Füllstände gedacht. Wie viele Tickets befinden sich in welchem Zustand? Welche Prioritätsverteilung haben diese Tickets? Eine Visualisierung, die diese Fragen beantwortet, könnte mit einer Darstellung der Knoten ähnlich wie in Abbildung 4.3 bewerkstelligt werden. Für die Kanten liegt eine Visualisierung der Durchflussrate nahe. Z.B. wird über die Dicke des Kantenpfeils dargestellt, wie viele Tickets pro Zeiteinheit von einem Status in einen anderen wechseln. Eine solche Darstellung, so die Überlegungen zur Interpretation des Graphen, könnte genutzt werden, um eine stockende Bearbeitung von Tickets zu entdecken oder sogar vorherzusagen. “Fließen” etwa

[5] Der erste Prototyp wurde nicht weiterentwickelt. Eine Erläuterung der Gründe dafür ist unter “Gewonnene Erkenntnisse” zu finden.

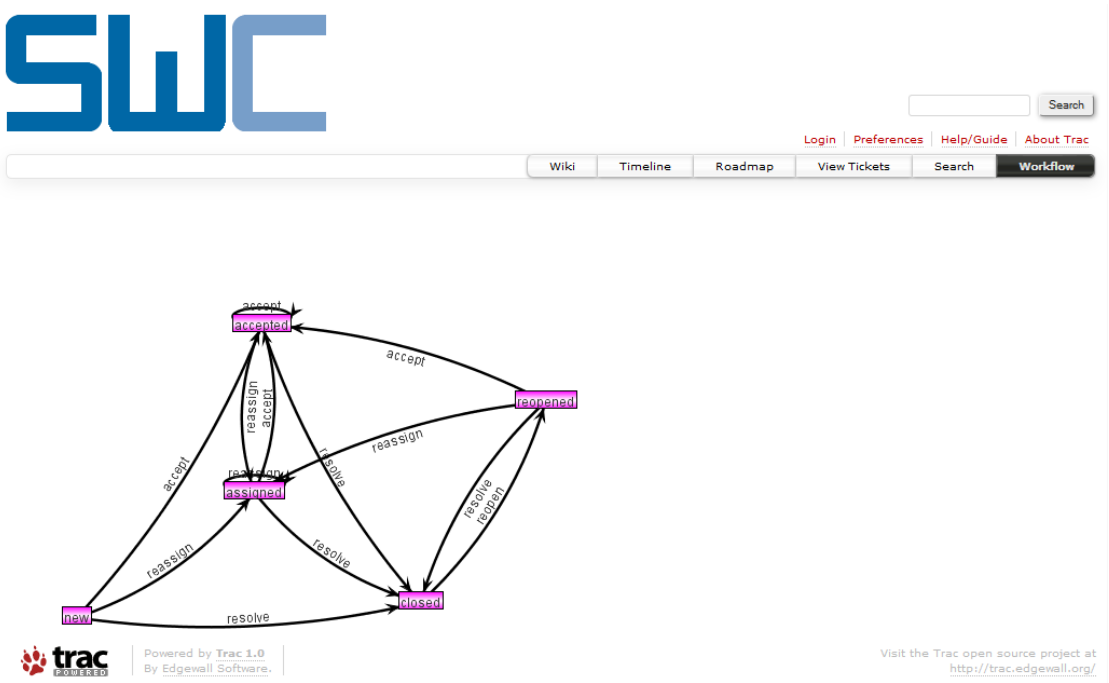

Abbildung 4.4: Erster Prototyp

über einen längeren Zeitraum mehr Tickets aus dem Status “assigned” in den Status “accepted” als von “accepted” zu “closed”, nehmen die Ticket-Bearbeiter mehr Tickets neu in Bearbeitung als sie fertigstellen. Es ist ggf. ein Anstieg der durchschnittlichen Ticket-Bearbeitungszeit zu erwarten. “Stauen” sich Tickets im Status “assigned” auf und “fließen” kaum weiter, wurden sie womöglich versehentlich einem in Urlaub befindlichen Mitarbeiter zugewiesen und wichtige Tätigkeiten bleiben unbemerkt unerledigt.

Neben Metriken, die direkt über die Darstellung des Knotens oder der Kante selbst visualisiert werden können (Füllhöhe, Pfeildicke), sollte es aber auch möglich sein, andere Metrikdarstellungen Knoten und Kanten zuzuordnen, z.B. in MeDIC-Dashboard genutzte Visualisierungen (siehe Abbildung 4.5). Liniendiagramme (etwa mit Füllstand oder Durchflussrate auf der y-Achse und der Zeit auf x-Achse) sind beispielsweise geeignet, um Trends darzustellen, wie sie insbesondere für Softwareprozess-Manager interessant sind. Histogramme setzen die durch den Knoten oder die Kante dargestellten Informationen in den Kontext eines weiteren Kriteriums, wie etwa die Ticketpriorität. Bulletgraphen helfen mit der Einordnung in Kategorien wie “gut”, “akzeptabel” und “schlecht” (Ampeldarstellung) und der Bereitstellung eines Vergleichswerts (das kann ein Sollwert sein, aber auch z.B. der Wert der vorherigen Messung) bei der Interpretation der dargestellten Metrik. M. Gora beschreibt diese und weitere Visualisierungen in seiner Bachelor-Arbeit[Gor13].

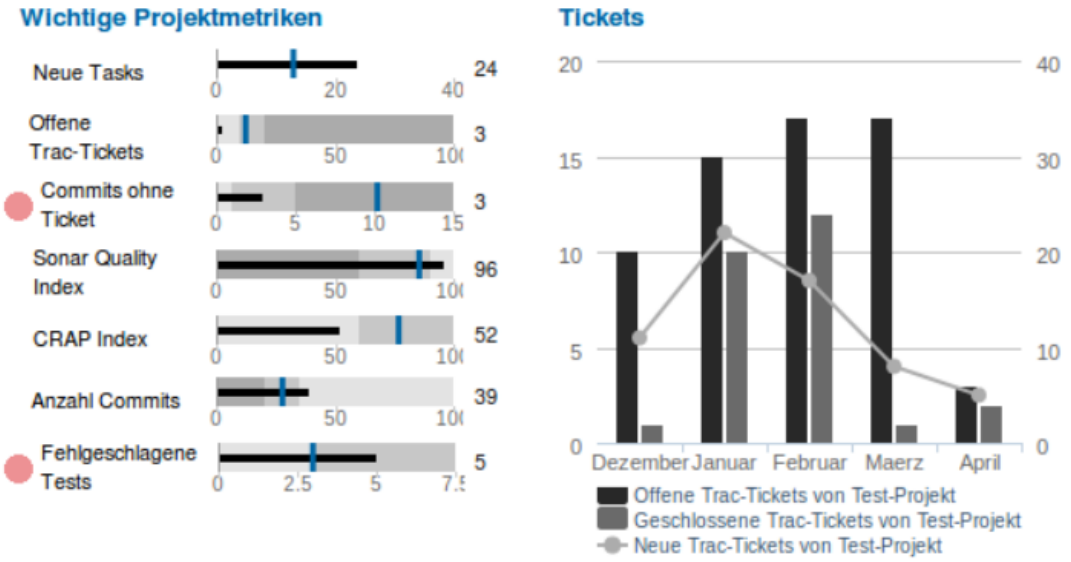

Abbildung 4.5: Bulletgraph (links) und kombiniertes Liniendiagramm/Histogramm (rechts) aus MeDIC-Dashboard
Quelle: [Gor13]

Gewonnene Erkenntnisse

Wie können diese zusätzlichen Metriken unter Beachtung von (Q3) (Übersichtlichkeit, Verständlichkeit) in den Workflow-Graph integriert werden? Zwei Bedingungen müssen dazu erfüllt sein:

1. Die Zuordnung muss eindeutig sein. Es muss auf eine Blick ersichtlich sein, welche Metrik zu welchem Graphelement (Knoten, Kante) gehört.

2. Die Metrik muss lesbar bleiben. Insbesondere darf die Skalierung der Metrik nicht zu klein werden, und die Metrik muss sich vollständig in einem freien Bereich zwischen den Graphelementen befinden. In keinem Fall darf etwa eine Graphkante mitten durch ein Liniendiagramm verlaufen.

Schon am obigen Beispiel des simplen Trac-Standard-Workflow ist erkennbar, dass die Erfüllung dieser Bedingungen keine triviale Aufgabe ist, und die gewählte Auto-Layout-Strategie dafür ungeeignet ist. Es ist nicht offensichtlich, wie etwa ein Liniendiagramm so in diesem Graphen untergebracht werden kann, sodass intuitiv erkennbar ist, dass es zur Kante von “accepted” nach “assigned” gehört und gleichzeitig gut lesbar bleibt.

Eine mögliche Lösung für dieses Problem stellen planare, orthogonale Graph-Layout-Algorithmen dar[ET94]. Diese wurden ursprünglich im Bereich der Very-Large-Scale-Integration (VLSI) dazu entwickelt, Transistoren in einem integrierten Schaltkreis optimal (nur senkrechte und waagerechte Leiterverbindungen, keine/wenig Biegungen, keine Leiterkreuzungen, kleinstmögliche Fläche) anzuordnen. Abbildung 4.6 zeigt den Standard-Trac-Workflow in einem orthogonalen, planaren Graphen. Die Knoten sind dabei an einem groben Raster bzw. Gitternetz ausgerichtet, während die Kanten über die Linien eines feineren Gitternetzes verlaufen (es ist feiner als das Knotenraster, um

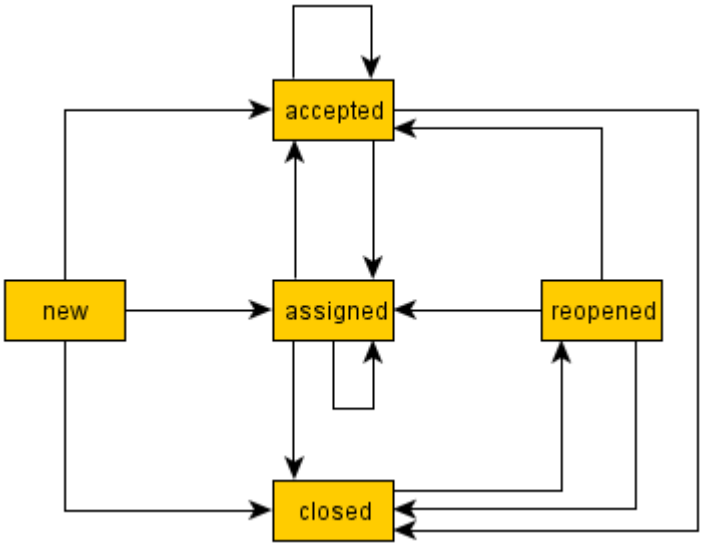

Abbildung 4.6: Standard Trac-Workflow in orthogonaler, planarer Darstellung

mehrere parallele verlaufende Kanten zu ermöglichen)[Fö97]. Um zusätzliche Metrik-Visualisierungen an den Kanten und Knoten anzubringen, könnte der Graph gestreckt werden, z.b. über die Vergrößerung des Abstandes zwischen den Gitternetzlinien, bis genügend Platz neben bzw. über oder unter den Graphelementen vorhanden ist, um einerseits die Metrik-Visualisierung in einem freien Bereich lesbar darzustellen, und sie andererseits so zu positionieren, dass durch ihre Nähe zu genau einem Graphelement intuitiv ersichtlich ist, auf welches Element sie bezogen ist. Es sind allerdings nicht alle Graphen planar oder planar darstellbar. Der vollständige Graph mit 5 Knoten K_5 ist ein solcher nicht planar darstellbarer Graph[6]. Durch die in modernen Change-Request-Systemen vorhandene Möglichkeit, Workflows frei zu definieren, kann nicht ausgeschlossen werden, dass hierbei nicht planare Graphen entstehen, wie etwa ein Workflow mit 5 Status, der den Übergang von jedem Status in jeden anderen erlaubt (und somit isomorph zu K_5 ist). In nicht planaren, orthogonalen Graphen, d.h. orthogonalen Graphen mit Kreuzungen, wird es aber wiederum schwieriger, geeignete Positionen für Metrik-Annotationen, die die beiden oben definierten Bedingungen erfüllen, zu berechnen.

In Absprache mit dem Betreuer der Diplomarbeit wurde an dieser Stelle die weitere Untersuchung dieser Visualisierung unterbrochen. Der Fokus dieser Arbeit ("Wie können in Softwareentwicklungs-Werkzeugen vorhandene Daten über Softwareentwicklungs-Prozesse ausgewertet und zur Prozessverbesserung verwendet werden?") sollte nicht aus den Augen verloren werden und sich nicht vorwiegend auf graphentheoretische Überlegungen verlagern. In einem zweiten Prototyp (siehe Abschnitt 4.7) sollte daher eine Visualisierung der Daten eines Change-Request-Systems gefunden werden, die sich "besser" zur Darstellung der Vorgänge im Softwareentwicklungs-Prozess eignet.

[6] `https://de.wikipedia.org/wiki/Planarer_Graph` abgerufen am 28.05.2013

4.6 Vom Workflow zum Sankey-Diagramm

Was ist mit der Forderung nach einer "besseren" Visualisierung am Ende des letzten Abschnitts genau gemeint? Einerseits eine technisch leichtere Umsetzung der Darstellung bei gleichzeitiger Beachtung der Anforderung nach Übersichtlichkeit und Verständlichkeit (Q3). Andererseits wurde im ersten Prototypen die Darstellung des Ticket-Workflows als Visualisierung hauptsächlich ausgewählt, um die Wiedererkennbarkeit des Softwareentwicklungs-Prozesses zu gewährleisten. In diesem Abschnitt wird die Frage untersucht, welche Arten von Informationen in einem Change-Request-System vorhanden sind und welche zum Workflow alternativen Darstellungsformen zu ihrer Visualisierung geeignet sind.

Im vorhergehenden Abschnitt wurden Annotationen von Change-Request-System-Workflows mit "Füllständen" an Knoten und Transitionsraten an Kanten skizziert. Diese beiden Aspekte ergeben sich gerade aus den typischen Daten, die Change-Request-Systeme produzieren. Sie halten einerseits die aktuellen Zustände der Tickets vor, aus denen der "Füllstand" resultiert. Anderseits stellt die Historie der Tickets die Datengrundlage für die Transitionsraten bereit.

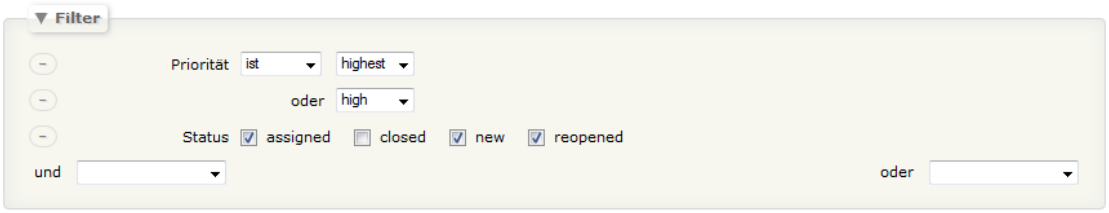

Abbildung 4.7: Eingabemaske zur Angabe eines Filters bei Ticketabfragen in Trac

Auswertungen über die aktuellen Ticket-Zustände unterstützen Change-Request-Systeme bereits über ihre Ticket-Abfrage-Mechanismen sehr gut. Der Anwender gibt dazu über einen Filter Kriterien vor, und erhält eine Liste aller Tickets, deren Eigenschaften die Kriterien erfüllen. Abbildung 4.7 zeigt die Oberfläche zur Eingabe eines Filters am Beispiel des Change-Request-Systems Trac. Nach einer Ticket-Abfrage mit diesem Filter erzeugt das System Ergebnisliste mit allen Tickets, deren Priorität "highest" oder "high" ist, und die sich in einem der Status "assigned", "new" oder "reopened" befinden. Über die Combo-Boxen "und" bzw. "oder" lässt sich der Filter leicht um zusätzliche Kriterien erweitern. Teil der Ergebnisliste ist selbstverständlich auch die Anzahl der zu den Kriterien passenden Tickets.

Im Vergleich zur Darstellung des Prozess-Workflows besitzt die Ergebnisliste einer Abfrage sowohl Vor- als auch Nachteile. Im Workflow ist die aktuelle Verteilung der Tickets gemäß des Ticketstatus "auf einen Blick" erkennbar. Das leistet die Ergebnisliste nicht, punktet dafür aber bei der Flexibilität. Über den Filter können Tickets nach einer beliebigen Kombination von Ticketeigenschaften ausgewählt werden. Im Workflow werden die Tickets fest nach dem Kriterium "Status" unterteilt, und ggf. mit Kontextinformationen über genau eine weitere Ticketeigenschaft versehen (vgl. Prioritäts-Schichten in Abbildung 4.3).

Die Historie der Tickets lässt sich hingegen mit Bordmitteln von Change-Request-Systemen derzeit (Stand: Mai 2013) überhaupt nicht (Trac, Redmine) oder nur rudi-

mentär (JIRA: Closed Vs Resolved Statistik) auswerten. Fragen wie: “Welche und wie viele Tickets mit hoher Priorität sind innerhalb der letzten Woche vom Status “in Bearbeitung” in den Status “im Test” übergegangen?”, oder: “Welche und wie viele Tickets haben den Status “im Test” in ihrem Lebenszyklus übersprungen und sind ungetestet geschlossen worden?”, oder: “Wie hoch ist der Anteil dieser Tickets an den allen Tickets?”, lassen sich aufgrund der fehlenden Bezugsmöglichkeit auf die Historie nicht über die verfügbaren Filterkriterien formulieren.

Eine Fokussierung auf die Visualisierung der Historie von Tickets ist also allein deshalb für den zweiten explorativen Prototyp schon interessant, weil sie von den Change-Request-Systemen bisher nicht bereitgestellt wird. Hier verbergen sich möglicherweise bislang nicht sichtbare Informationen, die die Stakeholder Softwareprozess-Manager und Projektleiter zur Einschätzung des Prozesszustands und zur Prozessverbesserung nutzen können.

Die Ticket-Historie beschreibt die Veränderungen der Ticketeigenschaften im Verlauf der Zeit. Dabei bilden mehrere Tickets, die in einer bestimmten Eigenschaft, wie etwa dem Ticketstatus, dieselbe Veränderung aufweisen, einen “Ticket-Fluss”: Die Tickets fließen von einem Status in einen anderen. Zur quantitativen Visualisierung dieser Flüsse eignen sich insbesondere, wie in Abschnitt 2.4 beschrieben, Sankey-Diagramme. Dieser Diagrammtyp wurde daher als zentrales Visualisierungselement für den zweiten Prototyp verwendet. Eine genaue Beschreibung des zweiten Prototyps sowie der Semantik eines Sankey-Diagramms im Kontext von Ticket-Flüssen befindet sich im folgenden Abschnitt 4.7.

4.7 Zweiter Prototyp

Das primäre Ziel des zweiten Prototypen ist die Evaluierung von Sankey-Diagrammen zur Darstellung von Ticket-Flüssen. Ein weiteres Ziel ist die technische Erkundung der Datenerhebung (Wie können Daten aus dem im Prozess verwendeten Change-Request-System extrahiert werden?).

Dieser Abschnitt beschreibt zunächst die Oberfläche des Prototyps, die Semantik des dargestellten Sankey-Diagramms und die getroffenen Design-Entscheidungen. Der fertige Prototyp wurde im Rahmen des “GDIS Info Cafe” Führungskräften der Generali Deutschland Informatik Services GmbH (GDIS) vorgestellt. Das dabei erhaltene Feedback wird am Ende dieses Abschnitts wiedergegeben. Die technische Realisierung des zweiten Prototyps ist im Anhang A.2 erläutert.

Beschreibung des Prototyps in Variante 1

Abbildung 4.8 zeigt die Oberfläche des Prototyps in Variante 1 (eine verbesserte Variante 2 wird weiter unten beschrieben) anhand eines beispielhaften Ticketflusses eines Trac-Change-Request-Systems, das den Trac-Standard-Workflow verwendet (vgl. Abbildung 4.4). Die Visualisierung stellt die Veränderung der Status der im Trac-System enthaltenen Tickets innerhalb eines Betrachtungs-Zeitraums in einem Sankey-Diagramm dar.

Das Diagramm wird von links nach rechts “gelesen”. Die Knoten sind dazu spaltenweise übereinander angeordnet. Es gibt eine erste Spalte (ganz links), deren Knoten im

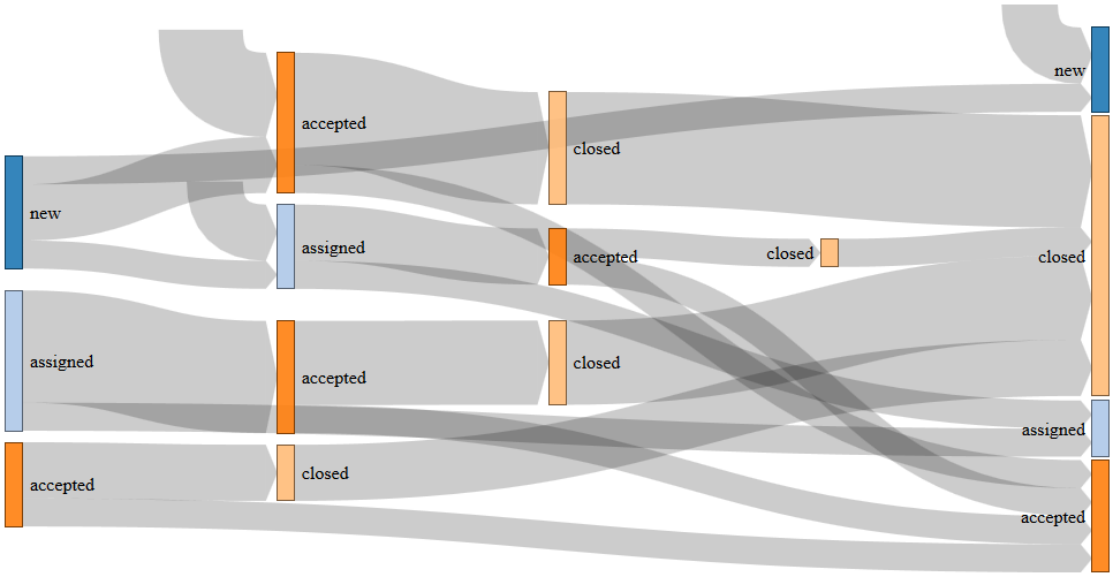

Abbildung 4.8: Variante 1 des zweiten Prototyps

Folgenden Startknoten genannt werden. Die Knoten der letzten Spalte (ganz rechts) heißen Endknoten. Beide Arten nehmen eine besondere Rolle gegenüber den Knoten (innere Knoten) der dazwischenliegenden Spalten ein. Die Startknoten in der ersten Spalte veranschaulichen die Verteilung der Ticket-Status zu Beginn des Betrachtungs-Zeitraums. Tickets mit gleichem Status sind dabei in einem Startknoten zusammengefasst. Die Anzahl der Tickets mit gleichem Status wird über die Höhe des entsprechenden Knotens visualisiert. In diesem Beispiel befinden sich zu Beginn des betrachteten Zeitraums 4 Tickets im Status “new”, 5 Tickets im Status “assigned” und 3 Tickets im Status “accepted” (insgesamt 12 Tickets). Die Endknoten in der letzten Spalte zeigen, wie viele Tickets welchen Status am Ende des Betrachtungszeitraums innehaben (hier: 3x“new”, 10x“closed”, 2x“assigned” und 4x“accepted”, insgesamt 19 Tickets). Die Visualisierung lässt zeigt zunächst nur die relative Verteilung der Tickets. Die absolute Zahl der durch Knoten (und Kanten) repräsentierten Tickets kann der Benutzer durch den Mouse-Over-Tooltip über dem entsprechenden Sankey-Diagramm-Element abfragen.

Die Startknoten, die Tickets mit gleichem Status zu Beginn des Betrachtungszeitraums zusammenfassen, fächern sich in nach rechts verlaufende Pfade auf. Ein solcher Pfad repräsentiert eine Gruppe von Tickets, die sich zu Beginn des Betrachtungs-Zeitraums im gleichen Status befanden und innerhalb des Betrachtungs-Zeitraums die gleiche Abfolge an Status-Änderungen durchlaufen haben. Jeder innere Knoten auf einem solchen Pfad steht somit anschaulich für Tickets mit identischer Status-Historie (seit Beginn des Betrachtungs-Zeitraums) und der Weg von einem Startknoten zu diesem inneren Knoten ist eindeutig. Nach einem inneren Knoten kann sich der Pfad weiter aufspalten, und so die weitere (unterschiedliche) Status-Entwicklung der durch den inneren Knoten repräsentierten Tickets darstellen.

Auf diese Weise bilden die Pfade, die den gleichen Startknoten besitzen, eine Art Korridor (gut erkennbar in Variante 2 des Prototyps, siehe Abbildung 4.9). Zwischen diesen Korridoren gibt es an den inneren Knoten keine Querverbindungen, d.h. Tickets mit un-

terschiedlichem Startknoten bzw. unterschiedlicher Statushistorie werden nicht wieder zu gemeinsamen Pfaden zusammengeführt, auch wenn sie etwa ab einem gemeinsamen Status identische restliche Statusveränderungen vollziehen. Erst die letzte Transition jedes Pfades führt zu einem nicht mehr nach Korridoren unterschiedenen Endknoten, der alle Tickets mit identischem Status zum Ende des Betrachtungs-Zeitraum unabhängig von ihrer Status-Historie konsolidiert.

Die jeweils letzte Transition jedes Pfades steht allerdings nicht mehr für eine tatsächliche Ticketstatus-Änderung. Sie dient lediglich der Zusammenführung von Tickets mit identischem Status zum Ende des Betrachtungs-Zeitraums. Die letzte Transition jedes Pfades verbindet daher auch ausschließlich gleich bezeichnete Knoten und kann durchaus auch die einzige Transition des Pfades sein, wie z.B. der Pfad mit nur einem Ticket vom Startknoten "new" links oben zum Endknoten "new" rechts oben. Die Semantik ist hierbei: Eines der vier Tickets, die zu Beginn des Betrachtungs-Zeitraums den Status "new" innehatten, hat im Laufe des Zeitraums keine Status-Änderung vollzogen und befindet sich zum Ende des Zeitraums nach wie vor im Status "new".

Neben den bisher beschriebenen Elementen des Sankey-Diagramms, die die Status-Änderungen existierender Tickets innerhalb des Betrachtungs-Zeitraums visualisieren, stellen die von oben "hinein fließenden" Transitionen Tickets dar, die während des Betrachtungs-Zeitraums neu erstellt wurden. Sie führen ausschließlich in die 2. Spalte im Korridor des Ticketstatus "new" (siehe links in Abbildung 4.8 oder an den Endknoten "new" (siehe rechts oben in Abbildung 4.8). Erstere repräsentieren neu erstellte Tickets, deren Status sich im Betrachtungs-Zeitraum ändert, letztere stehen für Tickets, die im Status "new" verbleiben. Die Sonderrolle des Status "new" ergibt sich daraus, dass sich alle Tickets bei ihrer Erstellung zunächst in diesem Status befinden.

Eine ähnliche Sonderrolle könnte auch der Status "closed" einnehmen. Der Lebenszyklus eines Tickets endet in der Regel mit dem Schließen des Tickets. Es kann dabei durchaus mehrere schließende Status geben, die z.b. nach Lösungsart differenziert sind. Einige Beispiele: "closed/resolved", "closed/cannot reproduce bug", "closed/won't fix". Es liegt nahe, in der Visualisierung die Transition zu einem "closed"-Knoten und den "closed"-Knoten selbst durch einen "herausfließenden" Pfeil zu substituieren. Zwei Diagramm-Elemente werden damit zu einem verschmolzen, ohne den Informationsgehalt zu reduzieren. Die Übersichtlichkeit der Darstellung verbessert sich. Allerdings zeichnen "closed"-Knoten nicht zwingend und eindeutig das Ende des Lebenszyklus eines Tickets aus, wie dies beim Status "new" und dem Beginn des Lebenszyklus der Fall ist. Der Trac-Standard-Workflow sieht beispielsweise vor, Tickets im Status "closed" über die Transition "reopen" wieder zu öffnen und in den Status "reopened" zu überführen. Die Visualisierung des Prototyps müsste daher Tickets, die im Betrachtungs-Zeitraum geschlossen werden und geschlossen bleiben, von Tickets differenzieren, die wieder geöffnet werden, indem sie z.B. erstere über "hinaus fließende" Transitionen darstellt und letztere wie in Abbildung 4.8 mit expliziter Transition und Knoten. Diese feingranulare Differenzierung verschlechtert jedoch die intuitive Verständlichkeit des Sankey-Diagramms. Daher wurde auf die Umsetzung einer gesonderten Darstellung von schließenden Ticket-Status verzichtet.

Änderungen bei Variante 2 des Prototyps

Bereits bei dieser einfachen (wenige verschiedene Pfade, einfacher Trac-Standard-Workflow) Visualisierung eines Ticketflusses tritt ein Problem dieser Darstellung hervor. Sie wird schnell unübersichtlich. Viele Pfade überlagern sich, und es ist schwer zu erkennen, welche Knoten sie miteinander verbinden.

Aufgrund der Korridor-Bildung ausgehend von einem Startknoten und entlang der inneren Knoten entstehen hier keine Überlagerungen. Das Problem tritt erst bei der jeweils letzten Transition jedes Pfades auf, die zu den Endknoten führt. Werden diese Transitionen entfernt, gewinnt die Visualisierung deutlich an Übersichtlichkeit. Abbildung 4.9 zeigt denselben Ticketfluss ohne finale Transitionen wie Abbildung 4.8. Übriggeblieben sind in dieser Darstellung alle Transitionen, die Ticketstatus-Änderungen repräsentieren. An den Startknoten sowie an den inneren Knoten existieren nun im Gegensatz zur Variante 1 des Prototyps Bereiche, die nicht von ausgehenden Transitionen "weiterverarbeitet" werden. Diese Bereiche sind implizit mit dem gleichnamigen Endknoten in der letzten Spalte verbunden.

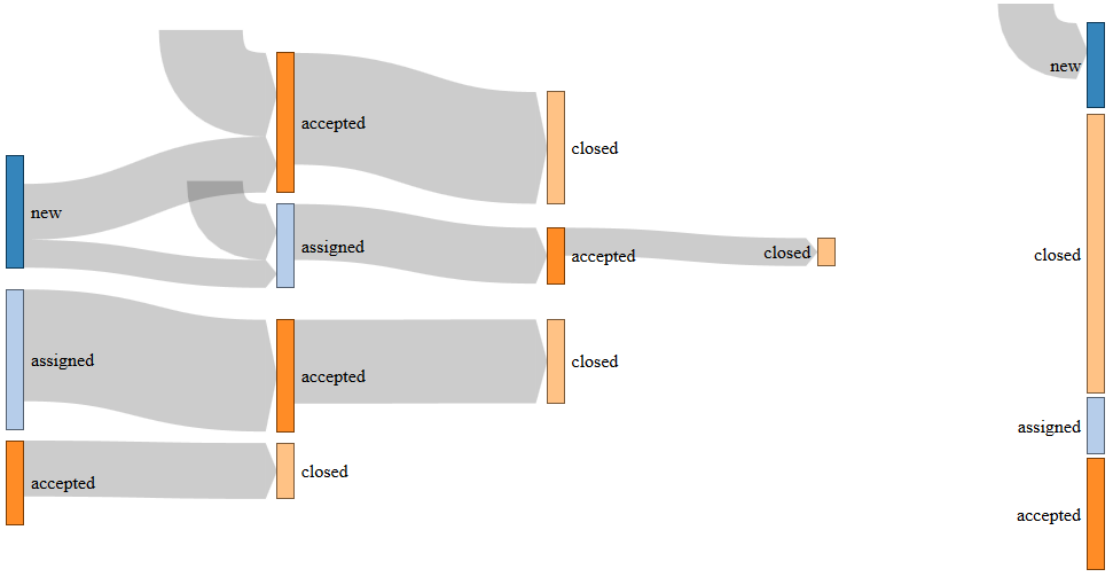

Abbildung 4.9: Variante 2 des zweiten Prototyps

Dem Vorteil der Übersichtlichkeit steht allerdings die Verletzung der Energieerhaltungs-Konvention bei Sankey-Diagrammen gegenüber. Sankey-Diagramme besitzen Quellen (hier: Startknoten) und Senken (hier: Endknoten). Für die dazwischenliegenden Knoten (hier: innere Knoten) gilt: Die Summe der Eingänge ist gleich der Summe der Ausgänge. Die Variante 2 des Prototyps hält diese Regel nicht explizit ein.

Hinweise zur Visualisierung

Wie oben bereits erläutert stellt die erste Spalte anhand der Höhe der Startknoten die Verteilung der Ticket-Status zu Beginn des Betrachtungs-Zeitraums dar. Die letzte Spalte visualisiert analog die Verteilung zum Ende des Zeitraums. Nun liegt die Vermutung nahe, dass die dazwischenliegenden Spalten diesen Zeitraum in gleich lange Intervalle

unterteilen. Das ist aber gerade nicht der Fall. Die Pfade der Sankey-Darstellung erzeugen lediglich eine Halbordnung und stellen die durch den Pfad repräsentierten Tickets zeitlich in eine Vorher-Nachher-Beziehung. Die folgende Liste verdeutlicht zulässige und unzulässige Interpretationen der Visualisierung:

- Ein Pfad sagt nicht aus, wann genau die Status-Änderungen bei den einzelnen Tickets stattgefunden hat. Insbesondere ist der Abstand zwischen den Spalten konstant und steht nicht in Bezug zum Änderungszeitpunkt.
- Die Spaltenposition von Knoten auf verschiedenen Pfade stellt keine zeitliche Relation dar. Insbesondere lässt sich aus einer weiter rechts stehenden Transition auf einem Pfad und einer weiter links angezeigten Transition auf einem anderen Pfad nicht folgern, dass erstere Status-Änderung zeitlich nach letzterer stattgefunden hat.
- Die in einer Transition zusammengefassten Status-Änderungen einer Gruppe von Tickets müssen nicht zeitgleich stattgefunden haben.
- Ein Pfad lässt nur folgende Schlussfolgerung zu: Die durch ihn repräsentierten Tickets haben jeweils zu unbestimmten Zeitpunkten innerhalb des Betrachtungs-Zeitraums die Status der Knotenliste des Pfades (in dieser Reihenfolge) angenommen.

Eine weitere nicht offensichtliche Besonderheit der Visualisierung betrifft geschlossene Tickets, deren Status sich innerhalb des Betrachtungs-Zeitraums nicht ändert. In Change-Request-Systemen werden Tickets in der Regel nach ihrer Bearbeitung geschlossen, aber nicht gelöscht. Daher entsteht nach und nach ein hoher Bestand an geschlossenen Tickets, die nicht weiter bearbeitet werden, während der Anteil aktiver, d.h. nicht geschlossener, Tickets relativ dazu unter der Annahme eines konstanten Ticketdurchsatzes abnimmt. Eine Darstellung aller Tickets des Change-Request-Systems im Sankey-Diagramm würde sehr hohe, wenig aussagekräftige Knoten mit dem Status “closed” in der ersten und letzten Spalte erzeugen, während die interessanten Ticketflüsse der aktiven Tickets auf einer kleinen, dem Verhältnis zwischen aktiven und geschlossenen Tickets entsprechenden Fläche angezeigt würden.

Um dieser mit der Zeit der Nutzung des Change-Request-Systems fortschreitenden Degradierung der Visualisierung entgegenzuwirken, bezieht der zweite Prototyp in beiden Varianten geschlossene Tickets, deren Status sich im Betrachtungs-Zeitraum nicht ändert, nicht in die Darstellung ein.

Interpretation der Visualisierung

Die Interpretation der im Sankey-Diagramm visualisierten Status-Änderungen der Tickets eines Change-Request-Systems überlässt der Prototyp vollständig dem Anwender. Er nutzt dazu seine Kenntnisse über den der Darstellung zugrundeliegenden Softwareentwicklungs-Prozess, um beispielsweise Abweichungen oder ungewöhnliche Vorgänge zu erkennen. Die Bereitstellung einer automatisierten Unterstützung bei der Interpretation ist nicht trivial. Sie erfordert die Entwicklung und Evaluierung eines Modells, das die Zusammenhänge zwischen den Daten eines Change-Request-Systems und

dem tatsächlichen Zustand des widergespiegelten Softwareentwicklungs-Prozesses herstellt und beschreibt. Ein solches Modell wurde im Rahmen dieser Diplomarbeit nicht erstellt, wird aber ggf. von einer nachfolgenden Arbeit behandelt (siehe Abschnitt 7.1). Dennoch werden im Folgenden mögliche Interpretationen der Visualisierung anhand einiger Beispiele beschrieben. Die dabei gefolgerten Schlüsse erscheinen dem Diplomanden plausibel, wurden jedoch nicht systematisch entwickelt oder validiert:

Als Basis für die Beispiele dient folgendes Szenario: Angenommen, eine Organisation verwendet das Scrum-Prozessmodell zur Entwicklung von Software und nutzt ein Change-Request-System zur Verwaltung des Sprint-Backlogs (vgl. Abbildung 4.2). In diesem Change-Request-System sei ein Workflow mit folgenden möglichen Status definiert: “ausstehend”, “in Analyse”, “im Entwurf”, “im Test” und “erledigt”. Dabei sei der direkte Wechsel von jedem Status in jeden anderen erlaubt. Der Scrum-Prozess sieht vor, dass das Scrum-Team mit der Bearbeitung eines Tickets beginnt, wodurch sich sein Status von “ausstehend” zu “in Analyse” ändert. Daraufhin wird ein oder mehrere Male die Abfolge “in Analyse”, “im Entwurf” und “im Test” durchlaufen, bis das Ticket schließlich den Status “erledigt” erhält.

Abweichung von einem vorgegebenen Prozess

In der Visualisierung des Prototyps kann der Anwender “gute” (Prozess-einhaltende) von “schlechten” (Prozess-verletzenden) Pfaden unterscheiden. “Gute” Pfade sind diejenigen, die die im Szenario beschriebene Abfolge von Ticket-Status widerspiegeln. Ebenfalls potenziell “gut” sind Pfade, die ein Suffix bzw. ein Präfix dieser Abfolge abbilden. Hier findet die Bearbeitung der Tickets nur zum Teil innerhalb des in der Visualisierung gewählten Betrachtungs-Zeitraums statt. Ob die Bearbeitung der Tickets außerhalb des Betrachtungs-Zeitraums dem Prozess entspricht, ist unbestimmt. “Schlechte” Pfade sind alle übrigen: Z.B. ein Pfad direkt von “ausstehend” zu “erledigt” oder ein Pfad ohne den Status “im Test”.

Die Schlussfolgerungen, die der Prototyp-Anwender aus dem Vorhandensein von “schlechten” Pfaden ziehen kann, können ohne die oben angesprochene Modellbildung und -evaluierung kaum generisch hergeleitet werden. Weitere, nicht in der Visualisierung sichtbare Faktoren beeinflussen die Schlussfolgerungen ebenso wie die Rolle des Prototyp-Anwenders.

Ein Beispiel: Besitzen die vom Wunschprozess abweichenden “schlechten” Pfade einen kleinen Anteil am Gesamtbild, wird der Prozess weitgehend eingehalten und es sind keine Maßnahmen durch den Projektleiter oder Softwareprozess-Manager erforderlich. Ist der Anteil der “schlechten” Pfade jedoch hoch, sollte der Projektleiter nach den Gründen für die Abweichung vom Prozess forschen. Möglicherweise passt der Prozess nicht optimal zum Projekt. Vielleicht sind viele Tickets derart trivial, dass die Mitarbeiter den Aufwand, den Fortschritt des Tickets im Change-Request-System zu erfassen im Vergleich zum Aufwand zur Bearbeitung des Tickets als unverhältnis hoch empfinden, kurzerhand das Ticket bearbeiten und es direkt vom Status “ausstehend” in den Status “erledigt” versetzen. In diesem Fall könnte der Softwareprozess-Manager den Prozess modifizieren und z.B. den direkten Übergang von “ausstehend” zu “erledigt” für triviale Tickets erlauben.

Dieses Beispiel verdeutlicht den Bedarf, die im Sankey-Diagramm dargestellten “Füll-

stände" und Transfermengen um weitere Informationen (wie in diesem Fall die Aufwandsabschätzung der Tickets) zu ergänzen.

Stockender Prozess

Eine Lagerbildung, d.h. Anhäufung von Tickets, in einem nicht-schließenden Zustand kann auf Probleme im Prozess hindeuten. In einem Scrum-Prozess bearbeitet das gesamte Scrum-Team in der Regel zu einem Zeitpunkt genau ein aktives Ticket. Eine Lagerbildung z.B. im Ticket-Status "in Analyse" weist also auf eine Abweichung vom Prozess hin: Es werden mehrere Tickets gleichzeitig bearbeitet. In einem System, in dem kontinuierlich Tickets erstellt und bearbeitet werden, wie etwa einem Kundenbetreuungssystem, kann eine solche Lagerbildung darauf hinweisen, dass mehr Tickets erstellt als erledigt werden und ggf. nicht genug (etwa personelle) Ressourcen für die Bearbeitung der Tickets zur Verfügung stehen. In einem nicht-iterativen, von Phasen geprägten Softwareentwicklungs-Modell, wie etwa dem klassischen Wasserfallmodell, kann eine Lagerbildung wiederum typisch für den Prozess und ganz normal sein. Z.B. befinden sich etwa alle in Tickets formulierten Anforderungen zu Beginn des Projekts im Status "in Analyse".

Auch hier ist die Diagnose, die aus der Visualisierung des Prozesses gefolgert werden kann, stark Kontext-abhängig.

Gewonnene Erkenntnisse

Die obigen Überlegungen zur Interpretation der Visualisierung zeigen, dass die Ticketfluss-Darstellung durchaus auf Probleme im Softwareentwicklungs-Prozess hinweisen kann. Um die Analyse zu verbessern, sollte die Visualisierung um weitere Kontextinformationen ergänzt werden. Der nächste Schritt, aus der Darstellung des Prozesses Ursachen für Probleme im Prozess und Maßnahmen für deren Lösung abzuleiten, ist jedoch nur schwer automatisierbar und beruht auf dem Expertenwissen des Anwenders.

Die Variante 2 des Prototyps wurde im Rahmen des "GDIS Info Cafe" in einer 20-minütigen Präsentation einigen Führungskräften der Generali Deutschland Informatik Services GmbH vorgestellt. Ziel dieser Veranstaltung war es, einerseits den Kooperationspartner GDIS über Entwicklungen in der Forschungsgruppe Software Construction ins Bild zu setzen. Andererseits bietet sie den Studenten die Möglichkeit, die eher theoretischen Ideen ihrer Abschlussarbeit mit den praktischen Erfahrungen aus der Softwareindustrie abzugleichen. Das in der der Präsentation folgenden Diskussion erhaltene Feedback wird hier wiedergegeben:

Der Ansatz, vorhandene Daten aus Softwareentwicklungs-Werkzeugen und insbesondere aus Change-Request-Sytseme zu visualisieren und zur Prozessanalyse zu verwenden, wurde grundsätzlich positiv bewertet. Es kam allerdings schnell die Frage auf, warum ausgerechnet der Ticketstatus und dessen Veränderung im Zentrum der Visualisierung steht. Wie oben bereits beschrieben, sind die möglichen Schlussfolgerungen aus der Visualisierung uneindeutig.

Stattdessen schlugen die Diskussionsteilnehmer vor, andere Ticket-Aspekte als den Status zu visualisieren, die konkreter interpretiert werden können. Sie verdeutlichten dies anhand von Szenarien:

Kundenbetreuung mit Service-Level-Agreement

Es sei in einem Service-Level-Agreement mit einem Kunden vereinbart worden, dass Fehlermeldungen innerhalb einer Frist bearbeitet werden. Weiterhin werde bei der Meldung des Fehlers anhand der Beschreibung eine Einschätzung vorgenommen, welche Abteilung innerhalb der Organisation für die Behebung des Fehlers zuständig ist. Diese Information wird in einer Eigenschaft des Tickets, einem Feld namens "zuständige Abteilung", festgehalten. Kommt es bei der Erfassung des Fehlers zu einer Fehleinschätzung, welche Abteilung zuständig ist, verbleibt er zunächst in der Abarbeitungswarteschlange der falschen Abteilung, bis er dort bearbeitet werden soll. Nun fällt die initiale Fehleinschätzung auf und der Fehler wird der richtigen Abteilung zur Bearbeitung zugewiesen. Ein solcher Ablauf kann die Bearbeitungszeit signifikant erhöhen und ggf. die Einhaltung des Service-Level-Agreements gefährden.

Abbildung 4.10 zeigt eine Visualisierung der Ticketeigenschaft "zuständige Abteilung". Wie oben beschrieben, ist es "gut", wenn die zuständige Abteilung anhand der Fehlerbeschreibung direkt bei der Fehlererfassung richtig eingeschätzt wird. Dies ist in der Visualisierung durch fehlende Änderungspfade erkennbar. Viele Änderungspfade deuten hingegen auf häufige Fehleinschätzungen hin. Abbildung 4.10 zeigt einen Prozess mit hoher Genauigkeit bei der Einschätzung der für die Fehlerbehebung zuständigen Abteilung.

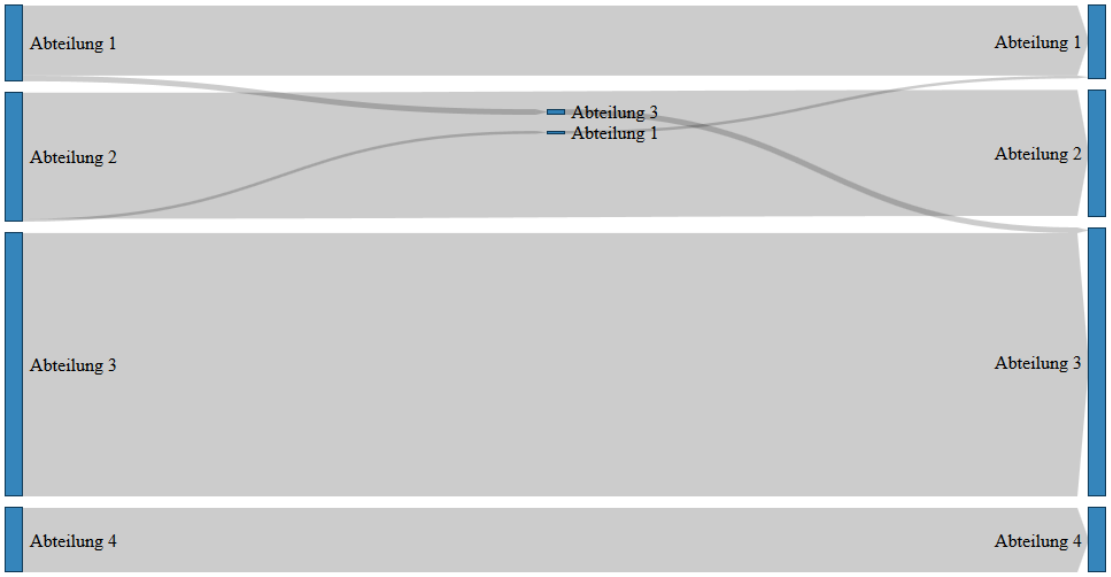

Abbildung 4.10: Visualisierung der Ticketeigenschaft "zuständige Abteilung"

Gegenüberstellung "Fehler gefunden durch" vs "Fehler entstanden in"

In diesem Szenario wird die Verwendung eines V-Modells zur Softwareentwicklung vorausgesetzt. Dabei stehen Phasen und Aktivitäten, in denen Fehler entstehen bzw. gemacht werden können, Prüfungen und Tests gegenüber, die diese Fehler aufdecken sollen. Fehler im Anforderungsdokument sollten etwa durch ein Anforderungsdokument-Review gefunden werden. Modultests finden Fehler in der Codierung. Die Quelle des Fehlers und

die Phase oder Aktivität (Fehleraufdecker), die ihn aufgedeckt hat, wird dabei in den Ticketeigenschaften “Fehler entstanden in” bzw. “Fehler gefunden durch” notiert. Die Visualisierung wurde wie folgt skizziert: Links im Sankey-Diagramm befinden sich mit den Fehlerquellen bezeichnete Knoten. Rechts stehen Knoten mit den Bezeichnungen der Fehleraufdecker. Jeder Fehlerquelle wird ein Fehleraufdecker zugeordnet, indem sie horizontal auf gleicher Höhe dargestellt werden. Die Pfade im Diagramm stellen dar, wie viele Tickets die Fehlerquelle und den Fehleraufdecker besitzen, die der Pfad miteinander verbindet.

Horizontale Pfade sind in dieser Darstellung “gute” Pfade, denn sie verbinden die Phase, die den Fehler enthält mit der Prüfung, die den Zweck hat, ihn aufzudecken. Sie zeigen also, dass der Prozess funktioniert. Geschwungene, s-förmige Pfade sind “schlechte” Pfade: Ein Architekturfehler etwa, der erst im Entwurfs-Review entdeckt wird, und dazu führt, dass zunächst die Architektur korrigiert und anschließend der bereits erstellte Entwurf neu angefertigt werden muss.

4.8 Konsolidierte Anforderungen

In diesem Abschnitt werden die Anforderungen an die Werkzeugunterstützung *River* formuliert. Die ursprünglichen, an den ersten Prototypen gerichteten Anforderungen, gelten weiter. Sie sind hier erneut wiedergegeben:

- (RB1) Um den Einsatz und die Evaluierung des Werkzeugs in existierenden Softwareentwicklungs-Projekten zu erleichtern und eine hohe Akzeptanz zu erreichen, soll das Werkzeug aus Sicht der Software-Entwickler eines Projekts “nichtstörend” sein (vgl. [Joh01]. D.h. es soll von den Entwicklern keinen über ihre normalen Tätigkeiten hinausgehenden zusätzlichen Aufwand erfordern, wie etwa die manuelle Eingabe von Informationen über ihre Aktivitäten im Rahmen des Softwareentwicklungs-Prozesses.
- (Q1) Die Einarbeitungszeit in das Werkzeug soll für mit dem Softwareentwicklungs-Prozess vertraute Personen kurz sein.
- (Q2) Der Konfigurationsaufwand vor und während der Benutzung des Werkzeugs durch die Stakeholder soll gering sein.
- (Q3) Die Visualisierung soll übersichtlich und leicht verständlich sein.
- (FA1) Das Werkzeug soll den Benutzer bei der Analyse des Softwareentwicklungs-Prozesses unterstützen. Dazu zeigt es Informationen über den Prozess-Zustand an und hilft, Abweichungen vom gewünschten Prozess zu erkennen und zu bewerten.
- (FA2) Das Werkzeug soll Daten aus Change-Request-Systemen erheben, weiterverarbeiten und visualisieren.

Darüber hinaus werden die Anforderungen anhand der mit den beiden Prototypen gewonnenen Erkenntnissen wie folgt ergänzt:

- (FA3) Das Werkzeug soll verschiedene Ticketeigenschaften visualisieren können, nicht nur den Ticket-Status.

- (FA4) Das Werkzeug soll die in den Prototypen erprobte Sankey-Diagramm-Darstellung um weitere Kontextinformationen ergänzen, die die Analyse des Prozesses unterstützen.

Schließlich folgen noch einige, nicht aus den Prototypen abgeleitete Anforderungen, die die Benutzbarkeit und Wiederverwendbarkeit des Werkzeugs steigern sollen:

- (FA5) Das Werkzeug soll über eine Schnittstelle verfügen, über die Change-Request-Systeme das Werkzeug über Änderungen an ihren Tickets informieren können, um die im Werkzeug erhobenen Daten zu aktualisieren.
- (Q4) Das Werkzeug soll mit verschiedenen Change-Request-Systemen zusammenarbeiten können.
- (RB2) Das Werkzeug soll aus wiederverwendbaren Komponenten bestehen und in der Technologie JavaEE entwickelt werden. Insbesondere sollen die Datenerhebung, die Aufbereitung der Daten und die Visualisierung durch einzelne Komponenten bereitgestellt werden, um eine ggf. später erfolgende Integration des Werkzeugs in MeDIC zu erleichtern.