Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3487

Visuelle Analyse von Stored Procedures in Datenbanksystemen

Matthias Meyer

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Steffen Lohman , M. Sc. Dipl.-Inf. Fabian Beck
Beginn am:	20. Mai 2013
Beendet am:	19. November 2013
CR-Nummer:	H.2.7, I.3.3, K.6.3

Kurzfassung

In vielen großen Datenbanksystemen kommen Stored Procedures zum Einsatz, wenn es um die effiziente Verarbeitung der Daten geht. Diese bilden nach einiger Zeit ein schwer zu überblickendes System aus komplexen Abhängigkeiten. Fehleranalysen und Optimierungen sind dann nur noch mit hohem manuellem Aufwand und intensiver Kenntnis des Systems möglich. In dieser Diplomarbeit wird ein interaktives System zur visuellen Analyse von Stored Procedures entwickelt. Der aktuelle Stand der Verarbeitung, zeitliche Verzögerungen sowie die Abhängigkeiten der Stored Procedures sind in der Visualisierung unmittelbar ersichtlich. Mit diesem Werkzeug kann der Anwender leicht den Prozess über viele Stored Procedures verfolgen und einfach Lastspitzen sowie Leerläufe identifizieren. Außerdem können wichtige Metriken statistisch analysiert werden. Der entwickelte Prototyp wird mit Hilfe von Experteninterviews evaluiert.

Abstract

In many big database systems Stored Procedures are used to efficiently transform the data. Over time these procedures become a difficult system with complex and obscure dependencies. Analyses of failures and for optimization purposes can only be done with huge manual effort and in-depth knowledge of the system. In this thesis a new interactive system for the visual analysis of Stored Procedures is proposed. The current status of processing, delays and the dependencies are directly visible in the visualization. Using this system, the analyst can easily trace the process across many Stored Procedures and find load peaks and idle states. Statistical analysis of important metrics of the processing is also directly possible. The prototype of the system is evaluated using interviews with database specialists.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1. Einleitung

Jeden Tag werden in großen Data Warehouse-Systemen hunderte oder gar tausende Prozessschritte in Form von Stored Procedures durchgeführt. Diese laden Daten aus externen Systemen, transformieren diese in das richtige Datenmodell oder aggregieren sie zu aussagekräftigen Berichten für Endanwender. So wird die Funktion eines Data Warehouse sichergestellt: Die richtigen Daten zum richtigen Zeitpunkt am rechten Ort zu haben um die richtigen Entscheidungen zu unterstützen [JLVV02].

Im Laufe der Zeit kommen neue Stored Procedures hinzu, alte werden geändert oder gelöscht und es entsteht ein Dickicht aus Abhängigkeiten zwischen den Prozeduren, das nur schwer überblickt werden kann. Die Verarbeitungsketten sind nicht immer gut gepflegt, Fehler in der Steuerung oder fehlerhafte Datenlieferungen aus externen Systemen stellen große Gefahren für die Abläufe in der Datenbank dar. Viele unterschiedliche Arten von Auslösern, die Stored Procedures starten, führen zu weiteren Unwägbarkeiten.

Zeitliche Abhängigkeiten können in verschiedenster Form auftreten: Für einige Prozesse ist ein bestimmter Zeitpunkt fest vordefiniert, manche Datenlieferungen werden per Dateitransfer in das Filesystem verschoben und müssen verarbeitet werden, sobald sie ankommen. Wieder andere warten auf Daten aus mehreren Quellen, die wiederum ganz eigene Abhängigkeiten haben. Nicht zuletzt gibt es Prozesse, die von Benutzern gesteuert werden und für die daher gar keine Vorhersage möglich ist, wann sie durchgeführt werden müssen.

Hinzu kommen Einschränkungen in den Systemen, die die Einhaltung der richtigen Reihenfolge sicherstellen, ganz gleich ob es sich dabei um ein modernes Workflowmanagementsystem oder ein einfaches Schedulingsystem für Batchprozesse handelt. Es kommt vor, dass Abhängigkeiten existieren, die mit dem System nicht abgebildet werden können oder die Abhängigkeiten sind so undurchsichtig und vielschichtig, dass sie vom Entwickler übersehen werden. Weiterhin können sich Annahmen, die bei der Entwicklung der Prozesse noch wahr waren, schnell ändern. Beispielsweise weil neue Hardware die Prozesse so beschleunigt, dass es zu Konstellationen kommt, die vorher durch entsprechend lange Rechenzeiten ausgeschlossen waren. Die Auswirkungen solcher Änderungen sind oft nicht abzuschätzen oder lassen sich nur durch Ausprobieren evaluieren.

Die Verarbeitung in Datenbanksystemen hat jedoch den großen Vorteil, dass überall Software beteiligt ist und Datensammeln kein Problem ist. Die erzeugten Aktivitätslogs der Stored Procedures entsprechen somit eher der Wahrheit und sind umfassender vorhanden als bei Prozessen, die teilweise von Menschen in der realen Welt ohne EDV durchgeführt werden [ADH+03].

In Datenbanken werden die Logs von den Prozessen selbst aufgezeichnet, sei es durch Audit-Funktionalitäten der Datenbank oder durch selbst programmierte Funktionen. Die Möglichkeiten, die Prozesse zu überwachen und zu steuern, sind vielfältig und lassen sich auf das jeweilige System und Umfeld anpassen. Dabei entstehen jedoch schnell riesige Datenmengen, die manuell nur schwer zu analysieren sind. Auch ist die Fragestellung nicht immer so konkret und präzise formulierbar, dass sie durch Algorithmen beantwortet werden könnte.

Nach der Definition von Tamara Munzner [Mun09], erfordert dieses Problem demnach eine Visualisierung:

> Visualization is used when the goal is to augment human capabilities in situations where the problem is not sufficiently well defined for a computer to handle algorithmically[.]

Das Anliegen dieser Arbeit ist also, eine visuelle Repräsentation dafür zu finden, wann welche Stored Procedures durchgeführt wurden, wie lange sie liefen und welche Abhängigkeiten zwischen ihnen galten. Diese Darstellung soll Entwickler bei der Fehleranalyse oder auch bei der Optimierung der Abläufe unterstützen. Das umfasst Fragestellungen wie „Wie ist der aktuelle Stand der Verarbeitung?“ oder „Wann sind bestimmte Datenlieferungen vollständig verarbeitet?“ aber auch Analysen wie „Wann ist die Last auf dem System am höchsten?“ oder „Welche Stored Procedures brauchen besonders lange und bedürfen möglicherweise einer Überarbeitung?“. Außerdem soll ein Vergleich möglich sein zwischen der aktuellen Verarbeitung und relevanter vergangener Läufe. Damit soll sich feststellen lassen, ob die Verarbeitung „normal“ läuft, oder ob es zu besonderen Ereignissen kam. Eine Evaluation soll am Ende prüfen, ob die Anwendung hilfreich ist und ob es die Arbeit von Datenbankentwicklern unterstützen kann.

Die restliche Arbeit ist wie folgt strukturiert: Um in das Thema einzuführen, werden im nächsten Kapitel einige grundlegende Technologien und Begriffe erläutert. Anschließend findet sich ein Überblick über bestehende Arbeiten und Ansätze, die ähnliche Probleme lösen.

Darauf folgt der Hauptteil der Arbeit, die Umsetzung. Darin wird am Anfang die grundlegende Idee der Diplomarbeit erläutert. Dann wird die Anwendung anhand der Verarbeitungspipeline vorgestellt. Von der Datengrundlage der Visualisierung geht es über die Bereitstellung und Weiterleitung der Daten an den Browser zum Layoutalgorithmus und dessen Anpassung and das Problem. Darauf folgt die Farbgebung und die interaktiven Elemente der Visualisierung. Die Steuerung der Anwendung steht sowohl am Ende der Pipeline als auch am Ende des Hauptteils.

Das vierte Kapitel beschreibt die Evaluation, beginnend mit dem Versuchsaufbau und dem Personenkreis, darauf folgt der Ablauf sowie die Ergebnisse und eine Diskussion. Ein Ausblick und das Fazit bilden das Ende der Arbeit.

2. Grundlagen

Für das Verständnis der Arbeit ist es notwendig, einige grundlegende Technologien zu erläutern und Begriffe abzugrenzen. Das Grundlagenkapitel ist thematisch zweigeteilt. Am Anfang stehen Technologien und Begriffe der Datenebene, des *Was* der Visualisierung. Dieser Teil beginnt mit relationalen Datenbanken als Grundlage für die darauf folgenden Stored Procedures. Darauf folgt ein kurzer Absatz über die Batchsteuerung. Eine Beschreibung des Datensatzes, der in der Arbeit verwendet wird, bildet den Abschluss des ersten Teils.

Der zweite Teil besteht aus der Visualisierungsebene, also dem *Wie*. Dabei erläutert ein Abschnitt die Begriffe des Graphzeichnen und des Layout-Algorithmus. Anschließend wird die dynamische Analyse erläutert. Der darauf folgende Abschnitt handelt von der Visualisierung zur dynamischen Analyse und diskutiert einige verwandte Arbeiten.

2.1. Relationale Datenbanken

Eine Datenbank ist eine wohlorganisierte Sammlung von sinnvoll verknüpften Daten, die auf mehreren logischen Ebenen verwendet werden kann [SS07]. In relationalen Datenbanken werden diese Daten in Tabellenform vorgehalten. Diese Tabellen sind nach dem relationalen Datenmodell organisiert, wie es Edgar Codd 1970 definiert hat [Cod70]. Das relationale Datenbankmanagementsystem (RDBMS) verwaltet die Datenbank und bietet Möglichkeiten, die Daten abzurufen, zu ändern, neu anzulegen und zu löschen. Heute besteht diese Möglichkeit meist in der Standard-Abfragesprache SQL (Structured Query Language). Sie entwickelte sich aus SEQUEL (Structured English Query Language), die 1974 genau für diesen Zweck entwickelt wurde.

SQL in Standardform bietet Befehle für die Manipulation von Daten (Data Manipulation Language, DML), für die Definition des Schemas (Data Definition Language, DDL) und für die Definition der Zugriffsrechte und der Transaktionskontrolle (Data Control Language, DCL). DML und DDL sind die am häufigsten benutzten Teile. DML umfasst die üblichen Kommandos `SELECT`, `INSERT`, `UPDATE`, `DELETE` zum Abfragen, Einfügen, Ändern und Löschen von Datensätzen. DDL bietet die Möglichkeit Datenbankobjekte, wie Tabellen, Views oder Indizes anzulegen, sowie diese wieder zu löschen oder zu ändern. Bekannte Vertreter sind `CREATE TABLE`, `DROP TABLE` oder `ALTER TABLE`.

2.2. Stored Procedures

Stored Procedures sind Codestücke, die in der Datenbank als Datenbankobjekt gespeichert werden. Sie wurden 1996 als eine Erweiterung für den SQL Standard der ISO von Andrew Eisenberg definiert [Eis96]. Die Erweiterung wurde drei Jahre später, zusammen mit anderen Erweiterungen in der Revision SQL:1999 zusammengefasst.

Stored Procedures ermöglichen es, mehrere DML-Operationen[1] nacheinander innerhalb eines atomaren Kontextes auszuführen. Der SQL Standard in der Revision SQL-92 sah dafür Module vor, die jedoch in einer anderen Programmiersprache geschrieben wurden und von der Datenbank nur aufgerufen wurden. Stored Procedures werden im Gegensatz dazu innerhalb des Datenbankprozesses (innerhalb desselben Adressraums) ausgeführt und bieten so Vorteile in Bezug auf Geschwindigkeit der Verarbeitung, Sicherheit sowie Verwaltbarkeit [Eis96].

Mit Stored Procedures können komplexe Verarbeitungen der Daten vorgenommen werden, die mit einzelnen SQL-Anweisungen nicht möglich wären. Der Standard definiert dafür die wohlbekannten Schleifenkonstrukte LOOP, WHILE und REPEAT sowie eine abgewandelte FOR-Schleife die direkt auf einem Cursor operiert. Damit lassen sich Datensätze aus einem SELECT-Statement einzeln durchlaufen und verarbeiten. Stored Procedures können Parameter definieren, die, wie auch bei anderen Programmiersprachen üblich, als Ein- und/oder Ausgabe Parameter deklariert werden können. Auch ein Exception-System ist vorgesehen um Fehler zu melden und zu behandeln (vgl. [Eis96]).

Stored Procedures werden daher häufig verwendet, um große Datenmengen in der Datenbank zu verwalten und zu transformieren. Stored Procedures werden von vielen Datenbanksystemen unterstützt. Wie bei vielen Elementen des SQL-Standards haben auch hier die Hersteller proprietäre Elemente eingeführt, oder sind auf andere Weise vom Standard abgewichen, so dass die Interoperabilität nur eingeschränkt verfügbar ist . Das Grundprinzip ist jedoch in den meisten vorhanden (Vgl. Tabelle 2.1).

Stored Procedures können besonders bei großen Datenmengen ihre Geschwindigkeitsvorteile ausspielen. Daher spielen sie in Data Warehouses eine große Rolle. Data Warehouses sammeln Daten aus verschiedenen Quellsystemen und integrieren sie in ein zentrale Datenbank, das an die Bedürfnisse der Benutzer angepasst ist [CCR02]. Es gibt mehrere Möglichkeiten ein Data Warehouse mit aktuellen Daten zu versorgen, eine davon ist die Stapel-Verarbeitung oder Batchverarbeitung.

[1] DML: Data Manipulation Language. DML Operationen umfassen alle SQL-Operationen, die die gespeicherten Daten ändern können. Beispielsweise INSERT, UPDATE und DELETE.

Produkt	Name
ANSI/ISO Standard	SQL/PSM
IBM DB2	SQL PL
IBM Informix	SPL
Microsoft / Sybase	T-SQL
MySQL	SQL/PSM
Oracle	PL/SQL
PostgreSQL	PL/pgSQL, PL/PSM
Sybase	Watcom-SQL

Tabelle 2.1.: Datenbanksysteme und ihre Programmiersprachen für Stored Procedures.

2.3. Batchverarbeitung

Bei der Batchverarbeitung für Data Warehouses werden die Änderungen an den Daten im Quellsystem über einige Zeit gesammelt (meist ein Arbeitstag) und dann im Data Warehouse gebündelt. Batchverarbeitung (zu Deutsch Stapelverarbeitung) beschreibt eine Vorgehensweise zur Verarbeitung von Daten. Dabei werden zumeist seriell große Datenmengen aus den Quellsystemen exportiert, in das Data Warehouse geladen und in dessen Datenmodell transformiert und integriert [RKZ00, Wid95].

Das bietet sich insbesondere dann an, wenn das Quellsystem nicht 24 Stunden am Tag zur Verfügung stehen muss, da die Batchverarbeitung typischerweise mehrere Stunden benötigt und nur statische Datenbestände verarbeiten kann, die Daten im Quellsystem dürfen sich also während der Verarbeitung nicht ändern. Der Vorteil liegt dabei in der gesteigerten Performance beider Systeme während des Arbeitstages, da sie unabhängig agieren. Das Quellsystem braucht die Daten nicht direkt weiterleiten und das Data Warehouse kann sich ebenfalls voll auf die Analysen der Anweder konzentrieren.

Dem gegenüber stehen moderne OLAP-Systeme, die Änderungen in den Quellsystemen direkt oder nur mit kleiner Verzögerung anzeigen [RKZ00].

2.4. Beschreibung des Datensatzes

Die Daten stammen aus dem Data Warehouse eines mittelständischen Großhandelsunternehmen mit bundesweit ca. 2500 Mitarbeitern und 400 Anwendern, die das Data Warehouse regelmäßig verwenden. Sie bilden ein Protokoll aller Stored Procedures, die in der Datenbank ausgeführt wurden, um die Daten zu laden, zu aggregieren, zu transformieren oder zu exportieren.

Die Daten des Data Warehouse werden in mehreren Batch-Prozessen verarbeitet. Eine grobe Klassifizierung der Prozesse ist nach der Laufhäufigkeit möglich: Es gibt tägliche, wöchentliche, monatliche und jährliche Batch-Prozessketten.

Typ	Bedeutung
A	Aggregation
C	Technischer Prozess, Serviceprogramme, CTL-Tool, etc.
E	Exportprogramm
L	Ladeprozess (außer Warenwartschaftssystem)
T	Transformation

Tabelle 2.2.: Die verschiedenen Typen von Programmen.

Diese Ketten sind nicht zusammenhängend und beinhalten nur einen (wenn auch sehr großen) Teil der Procedures. Sie definieren sich über eine Vorgänger-Nachfolger-Relation der Stored Procedures. Diese Relation wird von den Entwicklern bei der Erstellung von neuen oder bei der Anpassung bestehender Stored Procedures definiert und gepflegt. Die Entwickler definieren aber nur pro Procedure die direkten Vorgänger. Diese Beziehungen werden in Tabellen in der Datenbank gespeichert und über eine Microsoft-Access-Applikation verwaltet.

Nicht alle tatsächlich auftretenden Abhängigkeiten lassen sich mit diesem Werkzeug abbilden. So gibt es beispielsweise Abhängigkeiten zwischen täglich laufenden und monatlich laufenden Programmen, die aber nicht in das Tool eingegeben werden können.

Die Prozesse in den Ketten berechnen zumeist die entsprechenden Aggregationsebenen. Daneben gibt es noch einige Programme, die von diesen Intervallen abweichen und beispielsweise mehrmals täglich, mehrmals im Monat oder auch von Benutzern gestartet werden. Diese müssen in der Visualisierung entsprechend angezeigt werden. Außerdem gibt es Programme, die eine andere Aggregationsebene berechnen, als ihre Laufhäufigkeit vermuten lässt. So gibt es beispielsweise ein Programm, das zwar täglich ausgeführt wird, aber in eine Tabelle schreibt, die Jahreswerte enthält. Dies geschieht, da ein rollender Jahreswert berechnet wird, der für die letzten 12 Monate gilt und nicht für das jeweilige Kalenderjahr. Diese Ausreißer verlangen nach einer gewissen Unschärfe bei der Kategorisierung.

Ein Prozess oder Programm ist dabei jeweils eine Stored Procedure, die aus einer oder mehreren Tabellen der Datenbank liest und zumeist in eine, selten aber auch in mehrere Tabellen schreibt. Die Begriffe Prozess, Programm und Stored Procedure werden aufgrund dieser Definition im Weiteren synonym verwendet.

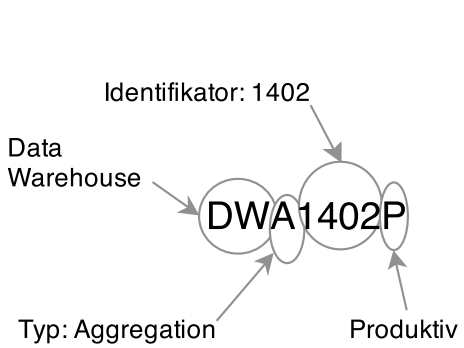
Die Stored Procedures werden nach einem festen System benannt. Der gesamte Name hat eine Länge von genau acht Zeichen: „D W _ _ _ _ _ P“

Die ersten beiden (DW) und das letzte (P) sind bei allen gleich und beschreiben, dass es sich um Programme aus dem **D**ata **W**arehouse handelt und es **p**roduktive Programme sind.

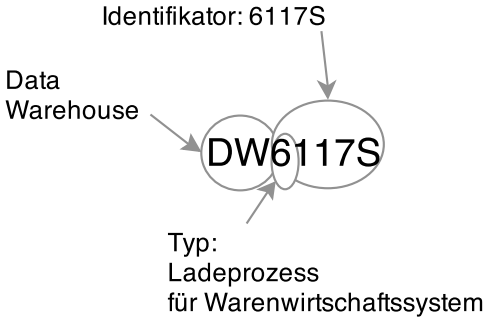
Es bleiben also fünf Zeichen zur Identifikation der Stored Procedure. Das erste dieser fünf kodiert den Typ des Programms. Die möglichen Typen sind in Tabelle 2.2 beschrieben. Die verbleibenden vier Zeichen sind Ziffern und die Entwickler halten Programme, die ähnliche Daten verarbeiten, in enger Namensnachbarschaft.

Eine wichtige Ausnahme sind Ladeprozesse, die Daten aus dem Warenwirtschaftssystem des Unternehmen laden. Sie tragen keine Typdefinition wie die anderen, ihre Namen sind also nur sieben Zeichen lang. Auch fehlt ihnen das P am Ende. Ihre Namen bestehen also nur aus **DW** und einem 5 Zeichen langen Identifikator. Dieser wird vom Warenwirtschaftssystem vergeben und ist daher eindeutig. Aus diesem Grund wurde auch ein eigener Namensraum für diese Programme verwendet, um Namenskollisionen zu vermeiden.

Das Namenssystem ist in Abbildung 2.1 an zwei Beispielen visualisiert.



(a) Namensgebung für eine Aggregation.



(b) Namensgebung für einen Ladeprozess des Warenwirtschaftssystems.

Abbildung 2.1.: Visualisierung der Namensgebung für Stored Procedures.

Die Stored Procedures sind strukturell sehr ähnlich zueinander aufgebaut. Der Grundbau einer solchen Stored Procedure ist in Algorithmus 2.1 dargestellt. Als Parameter wird ein Datum erwartet, das den zu verarbeitenden Zeitraum definiert. Bei täglichen Programmen ist das zumeist der Vortag, bei monatlichen der Vormonat und so weiter. Es gibt aber auch Programme, die von diesem Standard abweichen und beispielsweise für den aktuellen Tag aufgerufen werden. Dieser Parameter, auch Laufparameter oder (am häufigsten) Intervall genannt, trägt in den allermeisten Fällen den Namen ProDat für Processing Date. Daneben wird ein Parameter CheckPre definiert, der die Stored Procedure anweist, zu überprüfen, ob alle ihre Vorgänger für das angegebene Laufdatum bereits beendet sind und der Lauf stattfinden darf.

Um diese Überprüfung zu ermöglichen, trägt jedes Programm, wenn es mit CheckPre = „Y“ gestartet wurde, den Aufruf und das Ergebnis in eine Tabelle ein. Auch fehlerhafte Läufe oder falsche Aufrufe werden so protokolliert. Genauso werden parallele Läufe, die doppelte Verarbeitung von Daten oder die Ausführung in der falschen Reihenfolge verhindert.

Durch die Steuerung über Parameter, ist es möglich Programme ohne die Überprüfung zu starten. Dies ist insbesondere für die Fehlersuche oder das manuelle Eingreifen in die Prozessketten notwendig. Diese Dunkelziffer von Läufen ist jedoch irrelevant, da es nur in

Algorithmus 2.1 Vereinfachte Grundstruktur einer Stored Procedure.

```
1  CREATE OR REPLACE PROCEDURE TRANSFORM_SOMETHING
2   ( I_PRODAT NUMBER DEFAULT TO_CHAR (SYSDATE-1,'YYYYMMDD')
3   , I_CHECKPRE CHAR DEFAULT 'Y')
4  AS
5    PGM_NAM VARCHAR2(100) := 'TRANSFORM_SOMETHING';
6  BEGIN
7   IF (I_CHECKPRE == 'Y') THEN
8     IF (IS_RUN_ALLOWED (PGM_NAM,I_PRODAT) <> 'Y') THEN
9       RAISE GLOBAL_ERROR;
10    END IF;
11    INSERT_START_OF_RUN (PGM_NAM,I_PRODAT);
12  END IF;
13
14  /* DO SOMETHING */
15
16  IF (I_CHECKPRE == 'Y') THEN
17    INSERT_END_OF_RUN(PGM_NAM,I_PRODAT);
18  END IF;
19 EXCEPTION
20   WHEN OTHERS THEN
21     /* handle exception */
22 END;
```

seltenen Ausnahmefällen vorkommt. Außerdem entstehen diese Läufe nur in Ausnahmesituationen, die generell nur falsche Informationen liefern würden. Die visuelle Analyse wird also nicht beeinträchtigt, wenn diese Daten fehlen. Statistisch sind diese wenigen Läufe ebenfalls irrelevant, wenn man die Datenmenge betrachtet.

Die Daten werden seit dem 01.01.2000 gesammelt und umfassen eine Gesamtzahl von mehr als 3,5 Millionen Läufen von 2.808 verschiedenen Programmen. Zur Bearbeitung der Diplomarbeit wurde eine Kopie des Datensatzes angefertigt um eine stabile Datenbasis zu bekommen. Dieser beschreibt den Zeitraum vom 01.01.2000 bis zum 18.05.2013. In diesem Zeitraum kam es zu keiner großen Hardware-Erweiterung. Kleinere Speichererweiterungen oder Software-Upgrades der Datenbank oder Betriebssystem hatten nach Aussage der Mitarbeiter keine großen Auswirkungen auf die Performance der Stored Procedures.

Im Juli 2013 (also während der Entwicklung der Diplomarbeit) wurde ein Umzug der Datenbank auf eine sehr viel schnellere Maschine (laut Hersteller fünf mal schneller) durchgeführt. Die Auswirkungen dieses Umzuges werden in der Validierung kurz beleuchtet. Vor diesem Umzug gab es keine großen Änderungen an der Hardware oder wichtigen Systemkomponenten. Daher können auch die sehr alten Datenbestände als relevant eingestuft werden. Natürlich erfordert dieser Umfang jedoch eine gewisse Sensibilität. Insbesondere wenn sehr alter Code plötzlich elementar geändert wird, kann das zu schwerwiegenden Änderungen in den Laufzeiten führen. Das muss bei der Verwendung des Tools immer beachtet werden.

Damit sind alle Themen des *Was* der Visualisierung erläutert und es kann zum *Wie* übergegangen werden.

2.5. Graphzeichnungen und Layoutalgorithmen

Eine in jedem Fall gute visuelle Repräsentation für Graphen zu finden, ist ein nach wie vor ungelöstes Problem der Informatik. Ganz besonders, weil nicht klar definiert werden kann, wann eine Repräsentation gut ist und wann nicht. Diese Einschätzung hängt immer von der Fragestellung ab, die mit dem Graph beantwortet werden soll.

So müssen beispielsweise die angebotenen U-Bahnlinien für den Reisenden anders dargestellt werden als für den Verkehrsplaner. Es gibt mehrere visuelle Metaphern für graphähnliche Strukturen. Die bekannteste und verbreitetste ist wohl „Node-Link“. Dabei werden die Knoten als Punkte und die Kanten als Linien zwischen den Knoten dargestellt. Weitere Darstellungsformen umfassen Matrix-Repräsentationen und Mischformen zwischen diesen beiden Metaphern. Auch hierarchische Informationen können in den Diagrammen untergebracht werden, beispielsweise durch eine entsprechende Verteilung der Knoten auf Ebenen oder durch Gruppierung in Matrixdarstellungen.

Layout-Algorithmen haben zum Ziel, unter der Eingabe eines Graphen (V, E) paarweise verschiedene Positionen (x_i, y_i) für alle Knoten v_i aus V und den Verlauf aller Kanten aus E zu berechnen [HSMM00]. Ästhetische Aspekte finden als Randbedingungen Beachtung. Herman et al. listen die folgenden auf:

- Minimierung der Kantenkreuzungen
- Ausschließlich gerade Kanten zeichnen
- Gleichmäßige Verteilung der Knoten und Kanten
- Gleichmäßige Länge der Kanten
- Isomorphe Strukturen im Graphen gleich darstellen

Eine Übersicht über die Literatur zu Layout-Algorithmen im Kontext der Informationsvisualisierung gibt [HSMM00].

Es existiert eine Fülle an Layout-Algorithmen. Allen gemein ist die hohe Laufzeit in Abhängigkeit von der Knotenzahl und die stark ansteigende Unübersichtlichkeit, wenn die Knotenzahl sehr groß wird. „Visual Clutter“ist ein Begriff, der von Ruth Rosenholtz in 2005 definiert wurde. Er beschreibt den Zustand, in dem ein Übermaß an Objekten oder deren Unordnung zu einem Abfall in der Bearbeitungsgeschwindigkeit einer Aufgabe führt. Faktoren, die zu Visual Clutter in Node-Link-Diagrammen führen, sind u.A. Kantenkreuzungen oder Überlappungen von Knoten [RLN07].

In dieser Arbeit wird ein force-directed Layout-Algorithmus verwendet. Dieser wird im Hauptteil näher erläutert.

Damit sind alle grundlegenden Technologien erklärt und vorgestellt und es kann zum eigentlichen Zweck der Visualisierung übergegangen werden: Der dynamischen Analyse von Software.

2.6. Dynamische Analyse von Software

Thomas Ball definiert die dynamische Analyse als die Analyse von Eigenschaften eines laufenden Programms [Bal99]. Sie steht damit im Kontrast zur statischen Code-Analyse, die allein den Source-Code untersucht und ist ein Teil der Programmvisualisierung, die nach B.A. Myers verwendet wird, um einen Aspekt des Programms oder seiner Ausführung zu illustrieren [Mye86].

Bei der Analyse von Stored Procedures ist es besonders wichtig, auf die dynamischen Eigenschaften zu achten, da die Laufzeit von SQL-Statements nur äußerst selten an der Syntax abgelesen werden können. Einfache Statements wie `SELECT COUNT(*) FROM MY_TABLE` können auf der selben Hardware zwischen Millisekunden und Stunden dauern, je nach Datenmenge und Tabellenstruktur. Mit Hilfe von Ausführungsplänen[2] des DBMS und Kenntnis der Daten können zwar grobe Abschätzungen abgegeben werden, aber auch schon bei der Generierung des Ausführungsplans verwendet das DBMS dynamische Informationen wie beispielsweise Menge und Verteilung der Datensätze in den abgefragten Tabellen (vgl. hierzu beispielsweise die Oracle Dokumentation über Ausführungspläne [Ora09b].). Eine rein statische Analyse kann hier also kein vollständiges Bild ergeben.

Dynamische und statische Analysen unterscheiden sich nach Ball [Bal99] in den drei folgenden Dimensionen.

- *Vollständigkeit*. Die dynamische Analyse erfasst nur tatsächlich ausgeführte Pfade, wobei die statische Analyse zwangsweise auch toten Code untersucht. Bei Stored Procedures kann die Suche nach totem Code sehr interessant sein, beispielsweise wenn sich die Datenbasis stark geändert hat und dadurch die Datenverarbeitung von Cursor-Verarbeitung auf Bulkstatements umgestellt werden kann.
- *Umfeld*. Die dynamische Analyse kann Abhängigkeiten zu externen Systemen oder Einflüssen aufzeigen, die der statischen Analyse verborgen bleiben. Insbesondere bei dynamischem Code, der sich während der Laufzeit ändert, kann die statische Analyse keine Aussagen mehr treffen. Bei Stored Procedures kann es geschehen, dass die Programme von außen gestartet werden, ohne dass innerhalb des Systems die Notwendigkeit ersichtlich ist, beispielsweise weil eine Datenlieferung im Dateisystem stattgefunden hat. Der statischen Analyse würde dieser Teil als toter Code erscheinen, da er nirgends referenziert wird.
- *Genauigkeit*. Die dynamische Analyse untersucht die konkreten Ausführungen, die statische Analyse enthält immer einen Abstraktionsaspekt, der die Aussage verfälschen kann. Die Abstraktion wird benötigt, damit die statische Analyse garantiert terminiert und sich nicht in Endlosschleifen verfängt.

[2] Ausführungspläne geben an, wie welche Tabellen nach den gesuchten Datensätzen durchsucht werden und wie die Datenmengen zusammengejoint werden. Beispielsweise ob Indizes verwendet werden oder ob die gesamte Tabelle durchlaufen wird.

Es ist also immer eine Abwägung, wann welche Form der Analyse eingesetzt wird. Je nach Fragestellung eignet sich eine Form besser als die andere. Visualisierungen können bei der dynamischen Analyse sehr hilfreich sein, da die dynamische Analyse schnell sehr große Datenmengen erzeugt, die nicht vollautomatisch ausgewertet werden können [KG06]. Im nächsten Kapitel werden Arbeiten vorgestellt, die Visualisierungen für die dynamische Analyse einsetzen und damit einen ähnlichen Ansatz wie diese Diplomarbeit verfolgen.

2.7. Verwandte Arbeiten

In den Arbeiten zur Software-Visualisierung wird häufig die Nähe zum Quelltext gesucht [DPJM+02, DPLP+05, BDPS94, JSB97]. Große Prozessketten können damit nur schwer analysiert werden. Die Daten zur dynamischen Analyse in bisherigen Arbeiten stammen oft nicht aus Produktiv-Umgebungen, sondern aus Debug-Läufen, die der Entwickler entweder selbst ausgeführt hat, oder in seltenen Fällen von Usern ausführen ließ.

In diesem Fall ist das anders. Es existiert zu jedem produktiven Lauf aller Programme ein Eintrag in der Log-Tabelle. Die Dunkelziffer der nicht erfassten Programmläufe oder der Programme, deren Läufe generell nicht aufgezeichnet werden, ist sehr gering und kann vernachlässigt werden. Natürlich ist die Datenmenge auch immer dadurch beschränkt, wie viel von den Programmen tatsächlich geloggt wird [CZD+09]. In diesem Fall stehen nur relativ wenige Informationen pro Lauf zur Verfügung. Jedoch gibt es für jeden Lauf dieselben Informationen, so dass sich eine hohe Genauigkeit innerhalb dieser wenigen Kennziffern erzielen lässt.

In der Arbeit von de Pauw et al. in 2002 [DPJM+02] werden die Methodenaufrufe während der Ausführung eines Java-Programms visualisiert. Die Zeit verläuft von oben nach unten, in der Horizontalen ist die Tiefe des Ausführungsstacks dargestellt. Die Farbe kodiert verschiedene Klassen. Die Länge eines Streifens repräsentiert die Zeit, die in einer Methode verbracht wurde. Durch Interaktion kann diese Ansicht weiter erkundet werden und es werden Klassen- und Methodennamen angezeigt. Damit verhält sich dieses System sehr ähnlich wie das in dieser Diplomarbeit vorgestellte. Als Abhängigkeiten können jedoch nur direkte Call-Referenzen dargestellt werden. Komplexere Wirkzusammenhänge wie bei den Stored Procedures können nicht dargestellt werden.

In der Prozessvisualisierung spielen dagegen Codezeilen kaum eine Rolle, es geht um Geschäftsprozesse, von der Technik darunter wird absichtlich abstrahiert, da die Prozesse zu komplex sind, um Code anzuzeigen. Auch ist die zugrundeliegende Programmierung der Module für viele Fragestellungen der Prozessvisualisierung irrelevant oder gar störend [DK76]. Hinzu kommen weitere Aspekte des *Programming in the Large*. Wenn der Prozess definiert wird, sind die Module unter Umständen noch nicht geschrieben, manchmal ist sogar die Programmiersprache noch nicht festgelegt. Besonders deutlich wird das in Sprachen wie BPEL oder BPMN 2.0, in denen Prozesse aus abstrakten Webservices bestehen. Bei BPMN 2.0 wird das Prozessdiagramm bereits vom Prozess-Designer gezeichnet, so dass sich hier keine Notwendigkeit für das automatische Generieren eines Layouts ergibt.

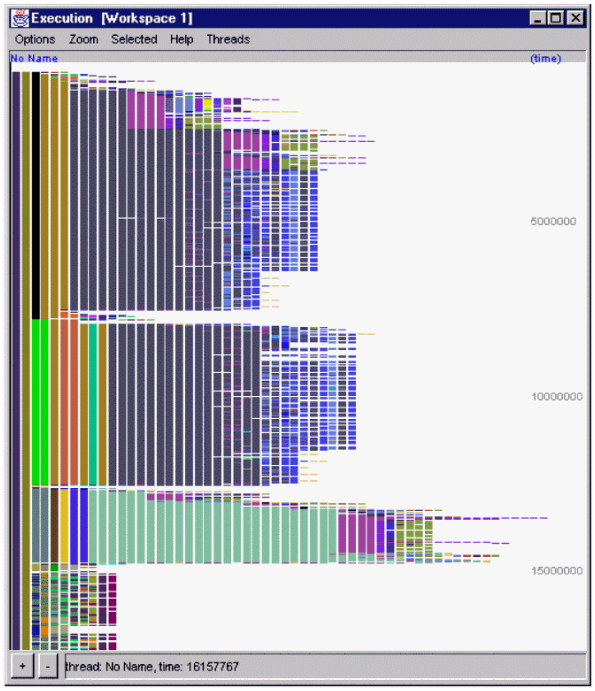
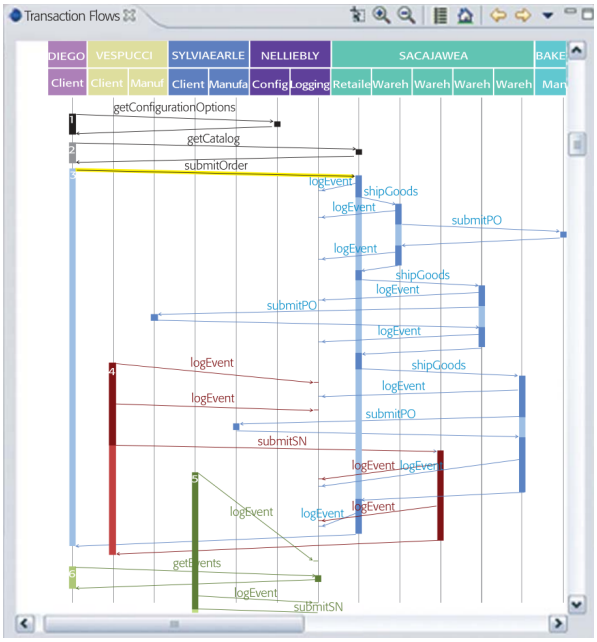


Abbildung 2.2.: Executions-Ansicht von de Pauw et al. [DPJM+02].

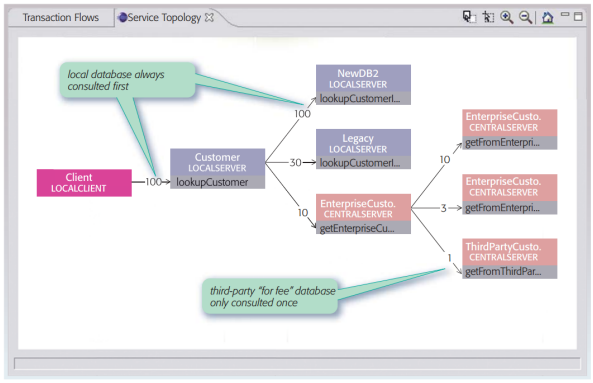
Für die architektonisch direkt darunter liegende Ebene der Webservices haben de Pauw et al. 2005 ein neues System vorgestellt [DPLP+05]. Zwei Ansichten daraus sind in Abbildung 2.3 dargestellt. Damit kombinieren die Autoren die tiefe Ebene der direkten Kommunikation auf Methodenebene und die logische, „Business Process“ Sicht der topologischen Ansicht. Durch Interaktion kann zwischen den beiden Ansichten gewechselt werden. Das System bietet weiterhin Funktionalitäten um Wiederholungen von Aufrufmustern zu identifizieren und farblich hervorzuheben.

Die detaillierte Ansicht Transaction Flow ist der älteren Arbeit von de Pauw et al. von 2002 [DPJM+02] sehr ähnlich. Genau wie dort, wird in der Horizontalen die Aufruftiefe dargestellt, mit dem Unterschied, dass hier die Webservices und deren Endpoints als Gruppierungselement verwendet werden können und die Kommunikationswege eindeutig sind. Auch können auf den Kommunikationswegen Verzögerungen auftreten, beispielsweise durch überlastete Netze. Diese werden ebenfalls visualisiert.

Im Vergleich zu der vorliegenden Diplomarbeit lässt sich sagen, dass die detaillierte Transaction Flow Ansicht kaum hilfreich wäre, da die Aufruf-Tiefe sehr flach ist. Eine Stored Procedure wird von außen, vom Schedulingsystem gestartet und verarbeitet direkt die Daten aus den Tabellen. Für die Geschäftslogik relevante Aufrufe von anderen Stored Procedures kommen nur äußerst selten vor. Die zweite Ansicht des Prozesses mit den Aufrufhäufigkeiten ist deutlich vielversprechender. Jedoch steht zu befürchten, dass der Layout-Algorithmus mit der Größe der Prozessketten überfordert wäre.



(a) Transaction Flow Ansicht. Hier ist die detaillierte Kommunikation von Webservices ersichtlich.



(b) Topologische Ansicht. Hier werden die logischen Einheiten dargestellt, zusammen mit den Abhängigkeiten, die sich aus den Aufrufen ergeben. An den Kanten sind die Anzahl der Aufrufe dargestellt.

Abbildung 2.3.: Zwei Ansichten des Web Services Navigator [DPLP+05].

Natürlich gibt es trotz allem noch viele Prozesse, die noch nicht mit BPMN 2.0 abgebildet sind, aber trotzdem einer Visualisierung bedürfen. Es gibt bereits viele Ansätze, solche Prozesse zu visualisieren, Rinderle et al. listet einige auf [RBRB06]. Einen interessanten Ansatz liefern Six und Tollis [ST02], die ein orthogonales Layout für Prozesse vorschlagen. Sie erzeugen dieses in linearer Zeit und mit maximal drei Abbiegungen in den Kanten. Durch die Generalität des Ansatzes lässt es sich für viele unterschiedliche Zwecke einsetzen. Für die dynamische Analyse ist sie jedoch weniger gut geeignet, da Start- und Laufzeiten nicht angezeigt werden. Es wird lediglich der logische Ablauf dargestellt, ein Ist/Soll-Vergleich ist nur schwer möglich. Auch steht wie bei de Pauw et al. [DPLP+05] zu befürchten, das die Größe der Prozesse in dieser Arbeit die Visualisierung überfordern würden.

Einen ebenfalls interessanten Ansatz, in einem thematisch etwas weiter entfernten Bereich, verfolgen Tory et al. in ihrer Arbeit [TSFH+13]. Darin werden Ablaufpläne für komplexe Bauvorhaben zur Analyse visualisiert. Das System ermöglicht auch das Verändern und das Vergleichen von Plänen um Alternativen erzeugen zu können oder What If-Analysen durchzuführen. Der Vergleich findet auf Basis von Gantt-Charts statt, wie in Abbildung 2.4 dargestellt. Die Analysen finden jeweils nur in der Metaebene der Pläne statt, ein Vergleich des Solls mit dem Ist ist nicht möglich.

Obwohl die Abbildung 2.4 sehr ähnlich zur Grundidee dieser Arbeit in Abbildung 3.1 auf Seite 26 ist, gibt es große Unterschiede. Zum Einen visualisieren Tory et al. mit Gantt-Charts, die voraussichtlich zu unflexibel in der Darstellung der komplexen Abhängigkeiten von Stored Procedures sind. Abhängigkeiten die mehrere andere, unbeteiligte Tasks überspannen,

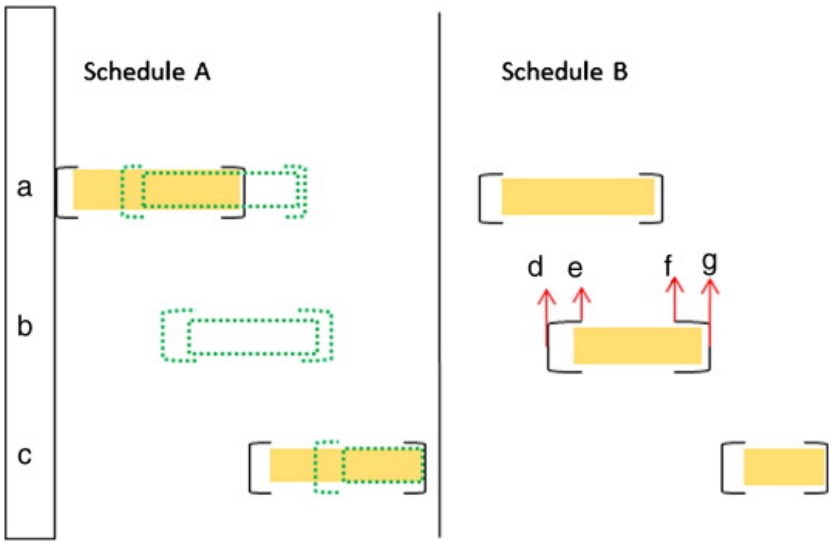


Abbildung 2.4.: Vergleich von Ablaufplänen auf Basis von Gantt-Charts [TSFH+13].

erzeugen in Gantt-Charts Kanten, die diese anderen Tasks überdecken und zu visual cludder führen, wie man in Abbildung 2.5 gut erkennen kann. Das ist insofern relevant, da es sehr viele solcher langen Kanten im verwendeten Datensatz gibt.

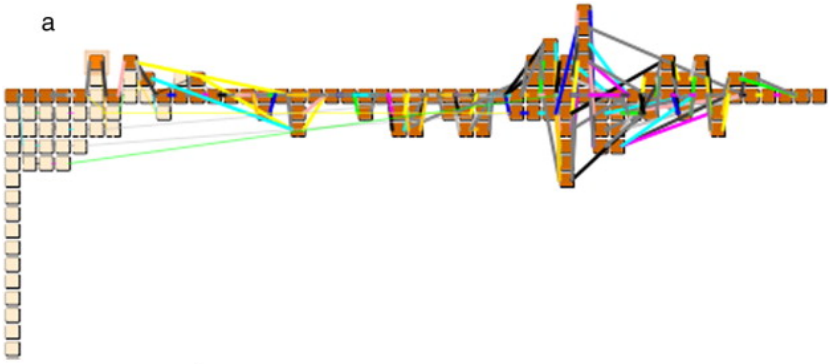


Abbildung 2.5.: Visual Clutter durch Kanten, die Knoten überdecken [TSFH+13].

Mindestens genauso interessant ist das Paper von Stott et al. [SRB+05]. Die Autoren benutzen die Metapher von U-Bahn-Verlaufsplänen um Projektpläne abzubilden, wie in Abbildung 2.6 dargestellt. Die Autoren verwenden interaktive Elemente um Teilketten hervorzuheben. Allerdings werden Aufgaben nur nach Startzeit angezeigt, eine Dauer der Aktivitäten wird nicht dargestellt und lässt sich nur durch die Länge der nachfolgenden Kanten grob abschätzen. Diese können jedoch aber auch durch andere Knoten induziert sein, so dass nicht sofort ersichtlich ist, welche Aufgabe wie lange dauert. Der Ist-Ablauf wird ebenfalls nicht angezeigt, so dass auch diese Arbeit für das in dieser Diplomarbeit behandelte Problem unpassend ist.

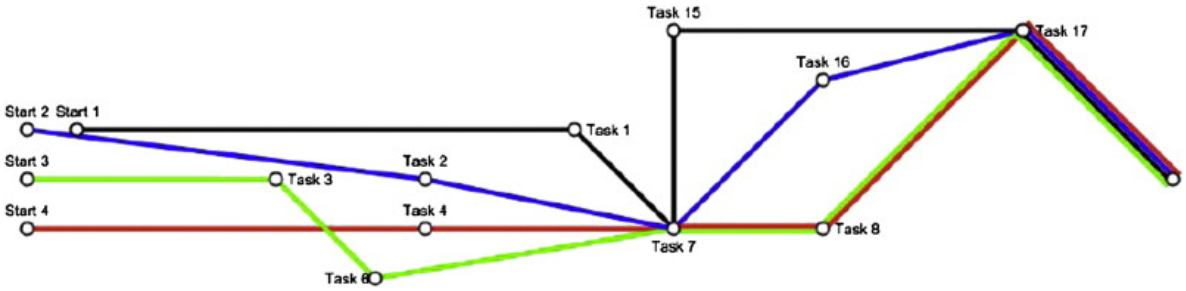


Abbildung 2.6.: U-Bahn-Metapher auf Projektpläne angewendet [SRB+05].

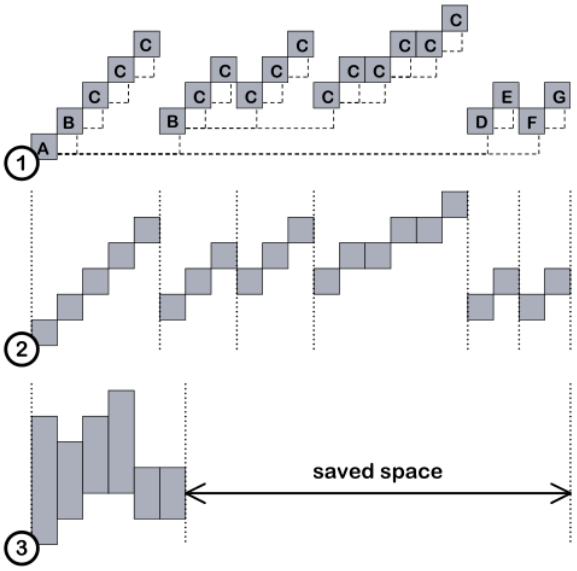


Abbildung 2.7.: Transformation der Methodenaufrufe in Signale. 1) zeigt die Methodenaufrufe, 2) zeigt die Methodenaufrufe in komprimierter Form ohne Namen und markierten Trennpunkten zwischen den einzelnen Features. 3) zeigt die fertig komprimierten Feature-Signale, die später visualisiert werden [KG06].

Kuhn und Greevy [KG06] verfolgen einen ganz anderen Ansatz. Sie versuchen der riesigen Datenmenge durch Abstraktion Herr zu werden. Die Analogie basiert auf der Signalanalyse aus der Physik und der dynamischen Analyse von Software. Dabei werden in prozeduralen Systemen zu jedem Funktionsaufruf Logeinträge erzeugt. Diese Logeinträge werden dann als Signal interpretiert und entsprechend visualisiert. Da diese traces sehr groß werden (Millionen von Einträgen, die gleichzeitig visualisiert werden sollen), müssen sie zusammengefasst werden, bevor sie dargestellt werden können. Die Idee von Kuhn und Greevy basiert darauf, alle Aufrufe zusammenzufassen, die einer einzelnen Aktion des Benutzers folgen und somit einem *Feature* der Anwendung zuzuordnen sind. Die gesammelten Aufrufe werden dann als ein Signal dargestellt, die Größe skaliert mit der Tiefe des Aufrufs, d.h. wie viele Methodenaufrufe bei der Realisierung des Features beteiligt waren. So wird bis zu 50% Platz gespart (vgl. Abbildung 2.7).

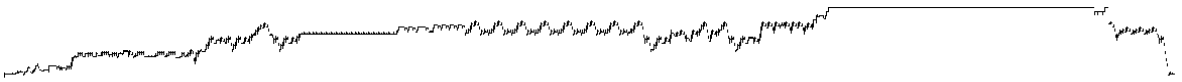


Abbildung 2.8.: Visualisierung von Kuhn und Greevy. Dargestellt sind die Balken aus 3) von Abbildung 2.7 reduziert auf die Höheninformationen [KG06].

Ganz so groß ist die Datenbasis der vorliegenden Diplomarbeit nicht. Es sind zwar 3,8 Millionen Datensätze in der CTL_PROC_STAT, jedoch werden nur Bruchteile davon gleichzeitig visualisiert. Auch ist die Calltiefe der Prozeduren sehr flach. Es kommt nur äußerst selten vor, dass eine Stored Procedure eine andere aufruft, die nochmal eine andere aufruft. Wenn gemeinsame Funktionalität verwendet wird, ist diese meist sehr allgemeiner Natur und wird von allen Stored Procedures des Systems verwendet (bspw. die Funktionen des CTL-Tools). Die Darstellung dieser Information würde also keinen Mehrwert schaffen. Die komplexen Abhängigkeiten zu möglicherweise lange zurückliegenden Programmen könnte mit diesem Ansatz ebenso nicht dargestellt werden, da die Arbeit nur direkte Call-Beziehungen visualisiert.

Es gibt noch einige Ansätze mehr, die aber alle ähnliche Probleme aufweisen, wie die hier vorgestellten Arbeiten. Eine exzellente Übersicht über die riesige Menge an Literatur im Gebiet der dynamischen Analyse liefert [CZD+09].

3. Visualisierung der Stored Procedures

Nachdem alle Grundlagen und Begrifflichkeiten geklärt sind, kann nun die Arbeit als solches vorgestellt werden. Den Anfang macht eine Beschreibung des Ansatzes, d.h. wie die eingangs genannte Problemstellung von dieser Diplomarbeit gelöst werden soll. Anschließend wird, wie im Grundlagenkapitel auch, zuerst das *Was* der Visualisierung erläutert um dann zum *Wie* überzugehen.

Als roter Faden dient also die Verarbeitungspipeline. Als erstes wird beschrieben, welche Daten visualisiert werden sollen. Anschließend, wie sie aus der Datenbank geholt werden und an den Client weitergeleitet werden. Dann wird erläutert, wie diese Daten vom Layoutalgorithmus dargestellt werden und wie dieser angepasst wurde, um in diesem Problembereich besser zu funktionieren. Danach werden die interaktiven Möglichkeiten der Visualisierung erläutert um abschließend die Möglichkeiten vorzustellen, wie die Anwendung gesteuert werden kann und welche zusätzlichen Elemente es im Tool gibt.

3.1. Ansatz

Die Idee dieser Arbeit ist die Visualisierung des Ablaufs vieler Stored Procedures und deren Abhängigkeiten. Die Stored Procedures bilden die Knoten und die Abhängigkeiten die Kanten (mit 2 in der Zeichnung markiert) eines Graphen. Die Stored Procedures sollen als Balken (mit 1 markiert) dargestellt werden, deren Position die Startzeit und deren Länge die Laufzeit repräsentiert. Die X-Achse kodiert die Zeit, die Y-Achse dient der Verteilung der Programme und Kanten, so dass eine übersichtliche Darstellung erreicht wird. So kann auf einen Blick entschieden werden, ob die Stored Procedure erfolgreich und pünktlich lief. Die Grundidee ist als Zeichnung in der Abbildung 3.1 dargestellt. Die Boxplot-ähnlichen Balken vor und nach den Programmen stellen die Pünktlichkeit des Programmes dar. Der linke Whisker (mit 3 markiert) repräsentiert das erste Dezil der Startzeit, der Anfang des Balken die tatsächliche Startzeit, die Balkenlänge die Laufzeit und somit das Ende des Balken die tatsächliche Endzeit. Der rechte Whisker (mit 4 markiert) repräsentiert das neunte Dezil der Endzeit. So gibt die gesamte Darstellung einen Eindruck, wann die Stored-Procedure „normalerweise“ läuft und wann sie in der betrachteten Situation gelaufen ist.

Die Boxplots wurden jedoch schlussendlich nicht umgesetzt. Stattdessen wurden Informationen über die Verteilung von Pünktlichkeit und Laufzeit der Programme in Histogrammen dargestellt, die im Abschnitt 3.9.3 beschrieben sind. Die Boxplots wurden nicht implementiert, da sie zu lang geworden wären. Bei vielen Programmläufen würden die Enden nicht mehr auf der Zeichenfläche liegen und hätten so keine Informationskraft mehr gehabt.

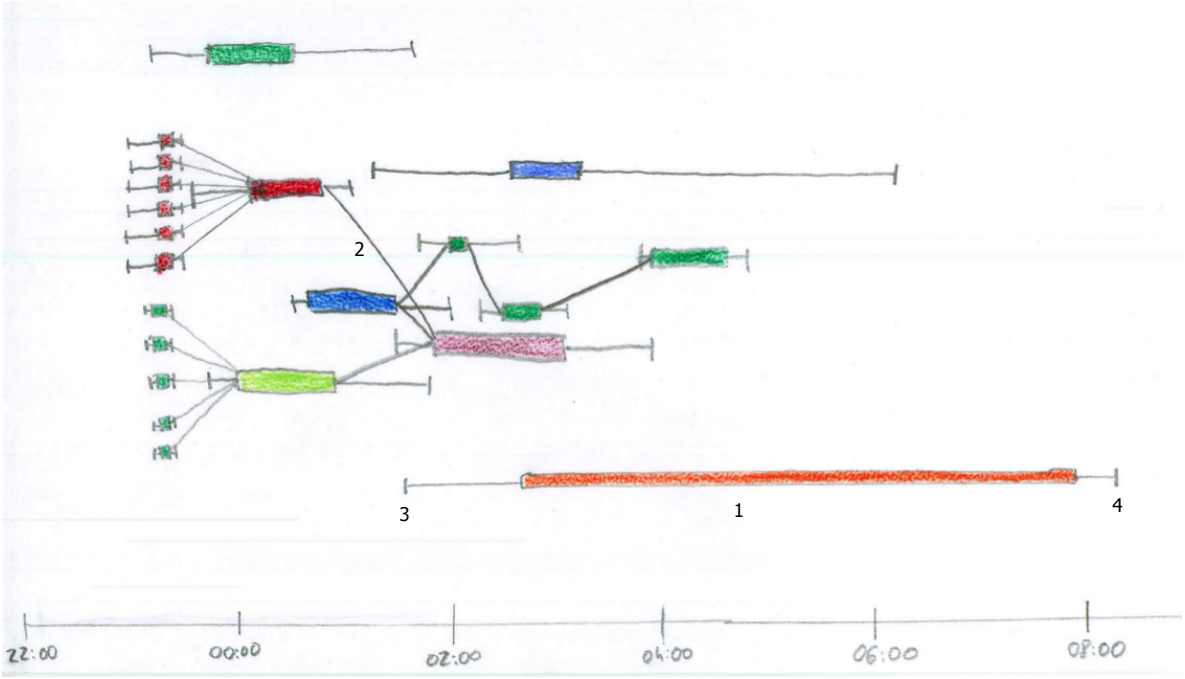


Abbildung 3.1.: Zeichnung der Grundidee. 1) Ein Lauf einer Stored Procedure. Der orange Balken definiert Start und Endzeitpunkt dieses einen Laufs. 2) Die Kante definiert eine Abhängigkeit zwischen dem roten und dem violetten Programm. 3) Das erste Dezil der Startzeit des orangen Programms. 4) Das neunte Dezil der Endzeit des orangen Programms.

Die Kanten sollen zum einen die Abhängigkeiten aus dem CTL-Tool darstellen als auch die Abhängigkeiten, die sich aus den Metadaten der Datenbank ergeben. Dem User soll die Möglichkeit gegeben werden, auszuwählen, welche Daten angezeigt werden. Es wurde auch versucht, beide Kantentypen gleichzeitig in der Visualisierung anzuzeigen, das hat jedoch die Analyse deutlich erschwert. Auch spielen die Kanten beim Layout-Algorithmus eine wichtige Rolle, so dass die Wahl der angezeigten Kanten bereits ganz am Anfang vom Benutzer festgelegt werden muss.

Die Farbe kodiert die Pünktlichkeit in den üblichen Regenbogenfarben. So entsprechen violett und blau überdurchschnittlich frühen Startzeitpunkten und gelb bzw. rote Töne besonders späten Läufen. Grün gefärbte Balken entsprechen pünktlichen Startzeitpunkten. Die Farbgebung erleichtert die Identifikation von stark abweichenden Programmen.

Die Programme sollen mit Hilfe eines force-directed Layout-Algorithmus auf der Zeichenfläche verteilt werden. Die Wahl fiel auf einen force-directed Ansatz, da diese Algorithmen die Möglichkeit bieten, vielfältig auf die Verteilung Einfluss zu nehmen. Es können einfach neue Kräfte hinzugefügt werden, um mehr Ordnung in das System zu bringen.

Insgesamt wird so dem Anwender die Möglichkeit gegeben, die Situation einzuschätzen. Fragestellungen wie „Was ist heute anders als sonst?“ oder „Gab es heute besondere Vorkommnisse?“ werden leichter beantwortbar. Genauso sollen Analysen wie „Warum sind diese Daten noch nicht verfügbar?“ mit der Visualisierung vereinfacht werden, da ersichtlich ist, was schon gelaufen ist und was nicht.

Die Optimierung der Abläufe wird damit ebenfalls unterstützt. Lange Wartezeiten zwischen einzelnen Programmen sind an langen Kanten ersichtlich. Hohe Lastspitzen werden von einem generell hohen Graphen dargestellt (in Abbildung 3.1 bspw. die Zeit von ca. 23:00 bis 01:00 Uhr), da viele Programme zu dieser Zeit aktiv sind. Zeitspannen, in denen kaum Aktivität stattfindet, sind ebenso leicht zu erkennen. Genauso klar zu erkennen sind Programme, die besonders lang laufen und eventuell eine Überarbeitung benötigen. So bietet das Tool einen allgemeinen Einstiegspunkt für die Analyse der Stored Procedures in der Datenbank.

3.2. Datenmodell der Stored Procedures

Zur Einführung in die Visualisierung wird hier das Datenmodell der Stored Procedures erläutert. Damit ist klar definiert, was visualisiert werden soll und wie die Darstellung zu interpretieren ist.

Die hier definierten Tabellen und einige PL/SQL-Packages bilden das *CTL-Tool* (sprich: *Controltool*). Die erste Version davon wurde 1998 erstellt und dient seither als Sicherungsschicht vor Fehlern in der Jobsteuerung. Da die Stored Procedures potentiell gefährliche Statements für das Data Warehouse wie INSERT, UPDATE, DELETE oder gar TRUNCATE[1] enthalten, ist diese Sicherungsschicht notwendig. Es ist damit eher Teil der defensiven Programmierung und nicht der Jobsteuerung, da es keine aktiven Komponenten enthält, es kann lediglich überprüfen, ob ein Parametersatz für eine Stored Procedure gültig ist.

Im Zentrum steht die Stored Procedure, definiert in der Tabelle CTL_PROC_DESC. Sie hat eine eindeutige Nummer (PROC_ID), einen Namen und einen Typ (TYPE_ID) (*Loader* (1) für Ladeprozess,*Transformation* (2) oder *Aggregation* (3)).

Außerdem sind ihr zwei Flags zugeordnet: Das erste ist *PARTIAL_FLAG*, das angibt, ob sie für ein Intervall mehrmals laufen kann. Das ist für Ladeprozesse notwendig, die Daten für den selben Tag mehrmals erhalten können, beispielsweise weil Nachlieferungen stattfinden können. Diese Stored Procedures können dann mit einer separaten Methode dem CTL-Tool mitteilen, dass ein Intervall abschließend erfolgreich verarbeitet wurde und dass ein weiterer Lauf falsch wäre. Das zweite Flag ist *PREPROC_FLAG*. Dieses gibt an, ob ein Programm zwingend auf seine Vorgänger warten muss.

Auf der linken Seite ist die Relation *should run on* dargestellt. Sie ist in der Tabelle CTL_PROC_INTVL definiert und gibt an, wann die Programme laufen sollen. Es gibt

[1] Löscht den gesamten Inhalt einer Tabelle

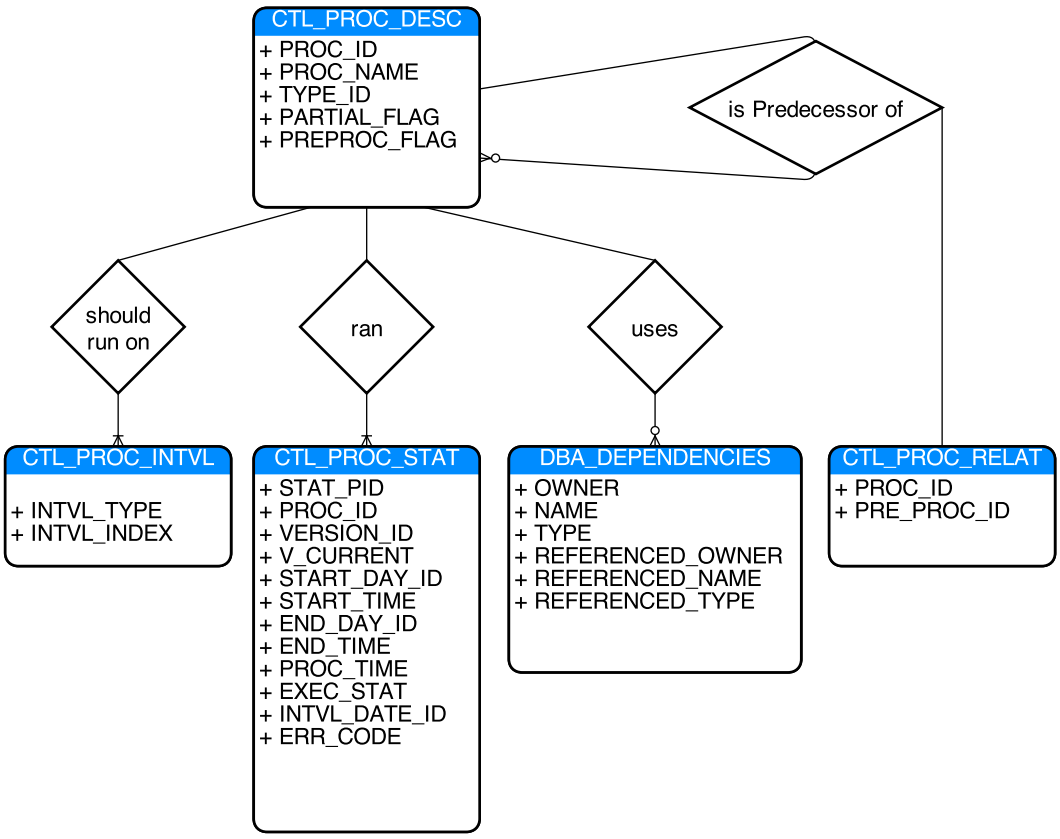


Abbildung 3.2.: Das Datenmodell der Stored Procedures als ER-Diagramm.

fünf *INTVL_TYPE*s: Täglich (1), wöchentlich (2), monatlich (3), quartalsweise (4), halbjährlich (5) und jährlich (6). Zusätzlich wird hier der *Interval_Index* definiert, der angibt, an welchen Intervallen des jeweiligen Typs das Programm laufen soll. Für tägliche Programme also die Wochentage mit den Indizes 1-7. Für wöchentliche Programme ist hier die Kalenderwoche hinterlegt (1-52), für monatliche der Kalendermonat (1-12) und so weiter.

Beispiel: Die Zeile (PROC_ID:17,INTVL_TYPE: 1,INTVL_INDEX: 2) besagt, dass Programm 17 ein tägliches Programm ist und Dienstage verarbeiten darf (der erste Tag der Woche ist Montag). Gibt es für Programm 17 keine weiteren Einträge in der Tabelle, darf das Programm ausschließlich Dienstage verarbeiten.

In der Tabelle *CTL_PROC_STAT* wird jede Ausführung der Stored Procedure protokolliert. Jede Ausführung erhält automatisch eine eindeutige ID. Die Zeile wird, während die Stored Procedure läuft, aktualisiert. Am Anfang wird das Startdatum, die Uhrzeit und das Intervall aufgezeichnet. Der Status wird auf *started* gesetzt. Nach erfolgreicher Beendigung wird das Enddatum und die Dauer eingetragen. Der Status wird auf *beendet* gesetzt. Im Fehlerfall wird ebenfalls das Enddatum aktualisiert und der Status entsprechend auf *Fehler* gesetzt. Partiell fertig bedeutet, dass das Programm bereits einen Teil (beispielsweise einige Niederlassungen)

EXEC_STAT	Bedeutung
1	Erfolgreich beendet
2	Gestartet
3	Fehler
4	Partiell fertig
6	Manuell eingefügt (für Feiertage)
8	Manuell zurückgesetzt
9	Manuell beendet (terminiert)

Tabelle 3.1.: Mögliche Status der Ausführungen von Stored Procedures.

verarbeitet hat, aber noch nicht alle. Dieser Status steht also im direkten Kontakt mit dem *PARTIAL_FLAG* aus der CTL_PROC_STAT. Die restlichen Status werden verwendet um manuelle Eingriffe zu dokumentieren. 6 beschreibt Läufe, die für Konsistenzgründe eingefügt wurden, aber nie durchgeführt wurden. Dies ist in seltenen Fällen für nicht bundeseinheitliche Feiertage notwendig. Manuell zurückgesetzt wird verwendet, wenn ein Programm abgebrochen ist und nochmal gestartet werden soll. So kann dokumentiert werden, dass ein fehlerhafter Lauf stattgefunden und ein erneuter Start zum Erfolg geführt hat. 9 ist dem sehr ähnlich, mit dem Unterschied das hier der Administrator zusätzlich den Lauf abgebrochen hat und das Programm nicht von allein einen Fehler erzeugt hat. Einen Überblick über die verschiedenen Status gibt Tabelle 3.1.

Auf der rechten Seite sind die Abhängigkeitsrelationen dargestellt, die als Kanten visualisiert werden sollen. Die ganz rechts dargestellte Relation *Is Predecessor of* definiert Vorgänger-Nachfolger Beziehungen zwischen den Stored Procedures. Ein Eintrag in dieser Tabelle wie beispielsweise (PROC_ID:17,PRE_PROC_ID:23) bedeutet, dass das Programm 17 erst dann ein Intervall verarbeiten darf, wenn dieses von Programm 23 bereits erfolgreich verarbeitet wurde.

Die mittlere rechte Tabelle DBA_DEPENDENCIES ist eine von Oracle bereitgestellte View und listet alle Abhängigkeiten auf, die der Datenbank bekannt sind, also alle, die von statischen Compiler gefunden werden. Abhängigkeiten die sich erst zur Laufzeit ergeben, bspw. durch die dynamische Generierung von Queries in Stored Procedures, können offensichtlich nicht dargestellt werden (vgl. [Ora09a]).

3.3. Bestimmung der anzuzeigenden Daten

Hier wird erläutert, welche Daten wie aus der Datenbank geladen werden.

Da der User nur den anzuzeigenden Zeitraum auswählen kann, gestaltet sich die Auswahl der Läufe relativ einfach. Es handelt sich um alle Läufe, deren Startzeitpunkt innerhalb des ausgewählten Intervalls liegen. Dafür genügt eine einfache Abfrage auf die Log-Tabelle der Stored Procedures (CTL_PROC_STAT).

Die Bestimmung der Kanten ist etwas schwieriger. Für die Kanten aus dem CTL-Tool genügt ein Join auf die Tabelle CTL_PROC_RELAT, in der genau diese Informationen gespeichert sind.

Für die Kanten aus den Metadaten muss die Oracle-interne View DBA_DEPENDENCIES verwendet werden. Diese View enthält jedoch auch alle Abhängigkeiten zu system-internen Objekten und deren Abhängigkeiten. Daher werden hier nur solche Objekte betrachtet, die im selben Schema wie die Stored Procedures liegen. So ist sicher gestellt, dass nur Objekte betrachtet werden, die tatsächlich eine Bedeutung für die Programme haben. Desweiteren werden einige zentrale Objekte des Data Warehouse ausgeschlossen. Andernfalls würden alle Programme mit allen anderen in Abhängigkeit stehen.

Als nächstes müssen die Werte für die Farbgebung bestimmt werden. Dazu muss erst bestimmt werden, wann ein Programm normalerweise laufen muss. Da dieser Wert a priori allgemein nicht festgelegt werden kann und auch nicht in den Daten vorhanden ist, muss hier die Historie zu Rate gezogen werden, um eine Abschätzung zu erhalten.

Dafür wird der **Offset** eines Laufes i eines Programmes j für einen Intervalltyp k definiert. Beim Offset wird auf den Intervalltyp Rücksicht genommen, da Programme an unterschiedlichen Tagen unterschiedlichen Ausführungsplänen folgen. Viele Programme laufen beispielsweise nur für Montag bis Samstag, da Sonntags keine Daten anfallen, die verarbeitet werden könnten. Auch die Datenmenge unterscheidet sich häufig an Wochentagen, was großen Einfluss auf die Laufzeit der Programme haben kann. Der Offset o_i sei wie folgt:

$$o_i = \text{START\_DATE}_i - \text{INTVL\_DATE}_i \tag{3.1}$$

Daraus ergibt sich ein Stundenwert, der angibt, wie viel Zeit zwischen dem verarbeiteten Datum und der tatsächlichen Startzeit liegt. Die Intervalle werden dabei durch den Beginn des Intervalls definiert. D.h. Tagesintervalle sind auf 00:00:00 Uhr festgelegt, Wochenintervalle auf den ersten Tag der Woche, Monatsintervalle auf den ersten Tag des Monats um 00:00:00 Uhr usw.. **Beispiel**: Das Programm *DWA1402P* läuft am 16.10.2013 um 06:32:15 Uhr und verarbeitet das Intervall 15.10.2013 (also den Vortag). Der Offset beträgt also 30,5375 (30h, 32min und 15s) Stunden.

Über alle Läufe i eines Programmes j für einen Intervalltyp k wird der Median gebildet und **erwarteter Startzeitpunkt** für dieses Programm und Intervalltyp genannt. Hier wurde der Median und nicht der Durchschnitt gewählt, da es zu großen Ausreißern in den Daten kommen kann und eine Reaktion auf diese Ausreißer vermieden werden soll.

Die **Verspätung** eines Programmlaufes i ist nun die Abweichung vom erwarteten Startzeitpunkt. Die Verspätung kann sowohl negativ (wenn es früher als sonst läuft) als auch positiv sein (wenn es zu einer Verspätung im Wortsinn kommt). Die Verspätung v_i ist also:

$$v_i = o_i - \tilde{o}_{j,k} \tag{3.2}$$

3.4. Bereitstellung der Daten

Um die Daten aus der Datenbank zu holen und an den Client weiterzuleiten, wurde eine Serverapplikation in Python 2.7 mit Hilfe des Frameworks Flask[2] als REST-Webservice implementiert. Sie liefert JSON[3] aus und kann so bequem vom JavaScript-Client verwendet werden.

Python und Flask wurden eingesetzt, um den Aufwand für die Serverapplikation gering zu halten. Der Geschwindigkeitsnachteil von Python gegenüber kompilierten Sprachen wie C(++) oder Java sollte nicht so sehr ins Gewicht fallen, da die aufwändige Selektion und Transformation der Daten in der Datenbank in einem großen Query erledigt wird. Mit Python werden die HTTP-Requests verarbeitet sowie die Zusammensetzung der Ausgabe-arrays und deren Übersetzung nach JSON übernommen.

Es wurden die folgenden REST-Methoden implementiert:

1. **GET** */getgraph/*

   URL-Parameter:

   - *start*: Startdatum
   - *end*: Enddatum

   Diese Methode ist das Herz der Anwendung, da sie alle Daten liefert, die für die Visualisierung initial benötigt werden. Sie liefert die Programme und deren Abhängigkeiten die zwischen *start* und *end* gestartet wurden. Die Antwort besteht aus einem JSON-Objekt mit drei Feldern, dem nodes-Array, das alle Knoten (d.h. Programme) enthält, dem links-Array, das alle Abhängigkeiten aus dem CTL-Tool enthält, und dem links_dep-Array, das alle Abhängigkeiten beschreibt, die aus den Metadaten der Datenbank ausgelesen wurden.

   Beim Auslesen der Daten aus der Datenbank werden jedem Knoten seine jeweiligen Nachbarn in ein Array eingetragen. Außerdem wird hier bestimmt, ob ein Knoten Teil eines Fächers ist oder ob er die Transformation nach einem Fächer ist.

2. **GET** */getstarthistogram/<proc_id>/<intvl_index>*

   URL-Parameter:

   - *intvl_date*: Intervall-Datum

   Diese Methode liefert die Daten für das Histogramm der Offsets. Dafür wird sowohl die PROC_ID , die das Programm identifiziert, benötigt, als auch der INTVL_INDEX, der angibt, was für ein Tag verarbeitet wurde, damit nur relevante Läufe verwendet werden. Außerdem werden nur solche Offsets zurückgegeben, die innerhalb von 3,3 Standardabweichungen um den Median liegen. Alle Offsets werden auf das als

[2] http://flask.pocoo.org/
[3] JavaScript Object Notation, http://www.json.org/json-de.html

URL-Parameter angegebene Intervall-Datum addiert. Dies geschieht im Server, da die Datenbank diese Berechnung am schnellsten und bequemsten vornimmt. Wird der URL-Parameter nicht gesetzt, werden die Rohwerte ausgeliefert.

3. **GET** */getdurationhistogram/<proc_id>/<intvl_index>*

   Diese Methode liefert die Daten für das Histogramm der Laufzeiten. Sie funktioniert genauso wie die vorherige Methode. Auch hier werden nur Laufzeiten ausgegeben, die nicht mehr als 3,3 Standardabweichungen vom Median entfernt sind.

4. **GET** */gettrcfile/<proc_name>*

   URL-Parameter:

   - *start*: Startdatum

   - *end*: Enddatum

   Hier kann das Logfile eines Programms abgerufen werden. Dafür wird der Name des Programms benötigt, da die Logfiles jeweils als Dateinamen den Programmnamen tragen. Weiterhin können Start- und Enddatum angegeben werden um den Bereich des Logfiles einzuschränken. Sind diese beiden URL-Parameter leer, wird das gesamte Logfile zurückgegeben.

   Es wurde eine recht rudimentäre Methode implementiert um die Logfiles aufzufinden: Sie müssen im Filesystem des Servers liegen, der die Serverapplikation der Visualisierung hostet. Das ist aber eher unrealistisch, da die Logfiles auf dem Datenbankserver liegen. Es bieten sich mehrere Lösungsmöglichkeiten an: Entweder man transferiert die Logfiles über den Umweg Datenbank, wo man sie per Stored Procedure ausliest, oder aber man gewährt dem Tool Zugriff (bspw. per Netzlaufwerk) auf das Verzeichnis des Datenbankservers, wo die Logfiles abgelegt werden. Oder man bietet die Files mit Hilfe eines FTP-Servers der Serverapplikation an.

   Die rudimentäre Lösung wurde implementiert, da in der Kürze der Zeit die Implementierung der Stored Procedure zur Weiterleitung nicht mehr möglich war und die anderen beiden Optionen aus sicherheitstechnischen Gründen in der Firma nicht mehr durchgeführt werden konnten. Dieser Teil ist für die prototypische Implementierung der Visualisierung nicht bedeutend, da eine praktische Lösung existiert und für die Evaluation jeweils kurz vorher die Logfiles auf den richtigen Server übertragen werden können.

Damit ist erläutert, was visualisiert wird und wie es zum Browser gelangt, nun wird beschrieben, wie die Visualisierung entsteht und welche Möglichkeiten der Interaktion dem Benutzer geboten werden.

3.5. Der Layout-Algorithmus

Damit die Daten übersichtlich dargestellt werden, wurde ein force-directed Ansatz gewählt. Dieser wurde vornehmlich von Fruchterman und Reingold [FR91] sowie Kamada und Kawai [KK89] entwickelt.

Er besteht darin, dass sich Knoten wie geladene Teilchen verhalten und sich gegenseitig abstoßen. Kanten wirken wie Federn und ziehen verbundene Knoten zusammen. Diese Kräfte werden in einem iterativen Prozess simuliert und bewegen die Knoten auf der Zeichenfläche. Wichtig ist, dass die Kräfte keine Beschleunigungen wie in echten physikalischen Systemen, sondern Geschwindigkeiten erzeugen. Das rührt von dem Ziel, ein statisches Gleichgewicht (in dem sich kein Knoten mehr bewegt) zu erreichen anstatt eines dynamischen, wie das Planeten unter der Wirkung der Gravitation tun [FR91]. Durch die Bewegungen der Knoten entsteht nach Ablauf des Algorithmus ein ausgeglichenes Layout, das Symmetrien hervorhebt, die gesamte Zeichenfläche nutzt, gleichmäßig lange Kanten anstrebt und Kantenkreuzungen zwar nicht minimiert (es findet keine Planarisierung statt) aber deutlich verringert [FR91]. Das entspricht auch den Zielen von Kamada und Kawai, die erkannten, dass eine minimale Anzahl Kantenkreuzungen nicht notwendigerweise Übersichtlichkeit garantiert. Der große Vorteil dieses Ansatzes ist die Einfachheit der Implementierung und die Hervorhebung von Symmetrien im Graphen. Zusätzliche Kräfte können einfach hinzugefügt werden.

In dieser Arbeit wurde der Force-directed Layout-Algorithmus aus der JavaScript Bibliothek *D3.js* [BOH11] verwendet. Dabei kommen noch einige Kräfte mehr außer der Abstoßung und der Anziehung zum Einsatz (vgl. [Bos13]):

- **Abkühlung:** Im Gegensatz zu dem Ansatz von Kamada und Kawai kommt hier nicht Simulated Annealing zum Einsatz, sondern ein Kühlungsfaktor *alpha* (im Weiteren und in allen weiteren Formeln mit a gekennzeichnet) der als Faktor bei jeder Bewegung eingerechnet wird und so die Bewegungen verlangsamt. Bei jeder Iteration wird dieser um 1% gesenkt, daher die Verlangsamung. In gewisser Weise simuliert dieser Faktor Simulated Annealing, indem er die Bewegungen kontinuierlich abbremst bis die Knoten zum Stillstand kommen (wenn alpha gegen Null konvergiert). Tatsächlich handelt es sich beim Abbruchkriterium aber nur um eine konstante Anzahl Iterationen, da die Kräfte keinerlei Einfluss auf a haben. Alpha wird auch zum „Wiederaufwärmen“ der Simulation verwendet, beispielsweise wenn neue Knoten hinzugefügt werden oder wenn das Ergebnislayout nicht zufriedenstellend ist.
- **Gravitation:** Diese Kraft g sorgt dafür, dass die Knoten nicht davonfliegen, da sie die Knoten bei jeder Iteration auf den Mittelpunkt der Zeichenfläche zubewegt. Umso schneller, je weiter sie davon entfernt sind. Besonders am Ende der Iterationen, wenn die anderen Kräfte schwächer geworden sind, wirkt sich die Gravitation aus. Details zur Berechnung finden sich in der Dokumentation zu d3js [Bos13].
- **Reibung:** Die Begriff Reibung ist etwas missverständlich. Es handelt sich nicht um einen Reibungskoeffizienten, sondern um einen Faktor r, der auf die durch den Algo-

rithmus induzierte Bewegung multipliziert wird. So kann die Stärke der Bewegungen kontrolliert werden. Er wird am Ende jeder Iteration mit der Gesamtheit aller Bewegungen des Knotens multipliziert. Die Gesamtheit der Bewegung ergibt sich aus der Differenz der *neuen Position* (x, y) für Knoten v und seiner *alten Position* (x', y'). Die alte Position bezeichnet dabei die Position, an der sich der Knoten vor der Iteration befand. Die neue Position ergibt sich aus der Berechnung sämtlicher anderer Kräfte während der Iteration. Allerdings werden hier nur die Kräfte betrachtet, die von der Bibliothek berechnet werden. Eigene Kräfte, die über das tick-Event (dazu später mehr) ausgelöst werden, finden hier keine Betrachtung.

- **Kantendistanz und -stärke:** Kanten werden in diesem Algorithmus nicht als Federn modelliert, sondern als schwache geometrische Beschränkungen. In jeder Iteration werden die verbundenen Knoten so bewegt, dass die Kantenlänge in Richtung der definierten Distanz konvergiert. Diese Bewegungen werden mit der Kantenstärke skaliert. So kann jeder Kante eine „Wunschdistanz“ vorgegeben werden.
- **Ladung:** Die Knoten werden als Ladungen modelliert. Um nicht den Einfluss von jedem Knoten auf jeden anderen Knoten einzeln berechnen zu müssen, wird hier eine Approximation auf Basis des Barnes-Hut-Algorithmus vorgenommen. Näheres dazu in der Dokumentation von d3js [Bos13].

In Abbildung 3.3 sind die täglich laufenden Stored Procedures mit ihren Abhängigkeiten dargestellt. Man sieht darin die Effekte der Gravitation, die zu einem runden Gesamtlayout führen. Die leeren Flächen ergeben sich aus den besonders starken Kräften, die von der großen Menge Knoten im Zentrum ausgehen und die anderen Knoten auf Distanz halten.

Diese Grafik ist am Anfang der Entwicklung entstanden und war sehr überraschend. Sie hat gezeigt, dass ein großer Teil der Stored Procedures keine definierten Vorgänger oder Nachfolger hat.

Dafür gibt es verschiedene Gründe. Meistens liegt es daran, dass keine Abhängigkeit definiert wurde, weil durch externe Systeme sichergestellt ist, dass sie in der richtigen Reihenfolge laufen, beispielsweise durch den Scheduler der Batchverarbeitung. Weiterhin gibt es Abhängigkeiten, die mit dem bestehenden System nicht modelliert werden können, bspw. wenn Programme für unterschiedliche Intervalle aufgerufen werden, aber dieselben Daten verarbeiten. Dies passiert beim Laden von Daten aus externen Systemen, die ihre Daten mit dem Export-Datum anstatt dem Erzeugungsdatum versehen. Dann läuft das Import-Programm mit dem Export-Datum und die darauf folgende Transformation mit dem Erzeugungsdatum.

Außerdem gibt es Programme, die schlicht keine Abhängigkeiten haben, entweder weil sie isoliert Daten verarbeiten und zeitgesteuert gestartet werden oder weil sie Daten verarbeiten die von einem externen System A zu einem externen System B geleitet werden sollen. Diese müssen generell den Umweg über Data Warehouse nehmen, egal ob sie dort verwendet werden oder nicht. Diese Daten haben dann keinerlei Bezug zum Rest der Daten und damit auch keine Abhängigkeit in der Verarbeitung. Dies geschieht, damit das Data Warehouse

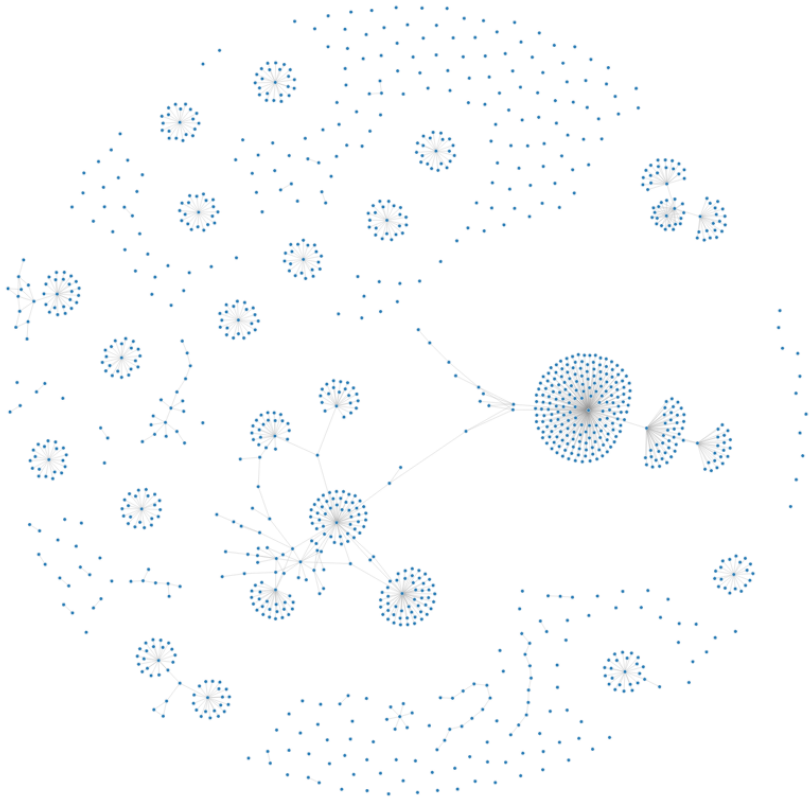


Abbildung 3.3.: Einfaches force-directed Layout der täglichen Stored Procedures mit ihren Abhängigkeiten.

über alle Daten verfügt und es nicht zu Fehlentwicklungen in anliegenden Systemen kommt. Beides kommt jedoch eher selten vor.

Interessant sind außerdem die Kreise mit einem Programm in der Mitte, das Nachfolger von vielen anderen ist, die keine weiteren Beziehungen zu anderen Knoten haben. Diese Strukturen repräsentieren Ladeprozesse, die für jede der 20 Niederlassungen separat definiert sind und eine Transformation, die die geladenen Daten verarbeitet. Die Ladeprozesse sind programmatisch identisch, aber für jede Niederlassung definiert, um zu prüfen, ob alle geladen wurden. Die darauf folgende Transformation verarbeitet dann alle Daten auf einmal.

3.6. Anpassung des Layout-Algorithmus

Damit die Visualisierung die Zeit korrekt darstellen kann, muss der Layout-Algorithmus angepasst werden. Er darf die Knoten nur auf der Y-Achse verteilen, da die X-Achse die Zeit kodiert und somit für jeden Knoten fest vordefiniert ist.

In der ersten Iteration wurde der Layout-Algorithmus nicht verändert und die vom Algorithmus berechneten X-Koordinaten schlicht ignoriert. Das Ergebnis ist in Abbildung 3.4 zu sehen. Man erkennt gut die Fächerform der Ladeprozesse und der nachfolgenden Transformation (in Abbildung 3.3 wurden diese Strukturen durch die Kreise dargestellt). Die Kantenlängen sind nur von bedingter Aussagekraft, da hier nur die Medianwerte für Offset und Laufzeit angezeigt werden.

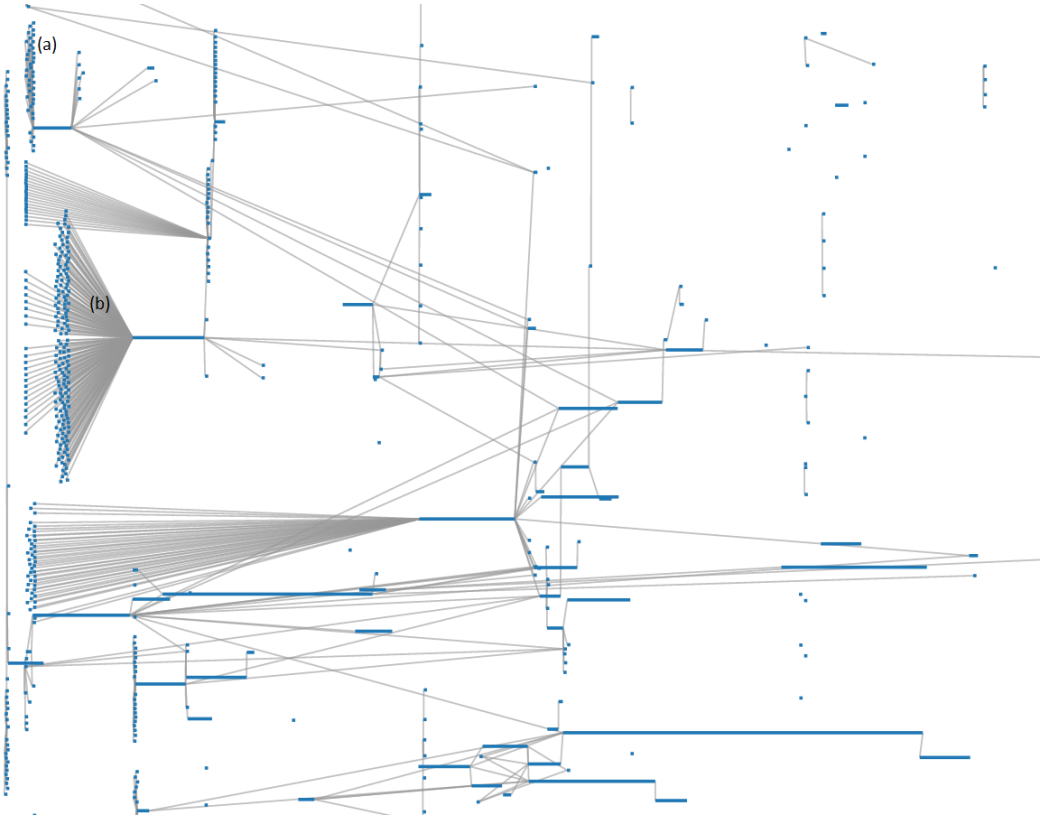


Abbildung 3.4.: Ausschnitt aus dem force-directed Layout, von dem nur die Y-Achse übernommen wurde. Dargestellt sind die täglichen Prozesse mit ihrem Median-Offset und Median-Laufzeit.

Viele Kantenkreuzungen, Überdeckungen der Kanten durch Knoten und ein generell unausgeglichenes Layout trüben die Übersicht. Der Algorithmus schien nicht am Ende, eine Verlängerung der Laufzeit (mehr Iterationen) brachte allerdings keine Besserung. Es musste also der Algorithmus an die neue Situation angepasst werden. Schließlich muss er wissen, dass Bewegungen auf der X-Achse keinen Effekt haben. Auch stimmten in dieser Form die Abstandsberechnungen nicht, da der Einfluss der X-Koordinate völlig falsch war.

In der ersten Iteration wurden schlicht alle Codezeilen aus dem Algorithmus entfernt, die die X-Position der Knoten beeinflussten. So steuerte der Algorithmus nur noch die Y-Position und die X-Koordinate repräsentierte fest die Zeit. Das brachte etwas Besserung. Allerdings

brachten die vielen Ladeprozesse, die alle an einer Transformation hingen, viel Chaos hinein. Diese sind in Abbildung 3.4 gut zu erkennen, sie bilden die fächerartigen Strukturen.

Damit die Fächer die Übersicht nicht stören, wurden diese ausgeblendet. um anzudeuten, dass das nachfolgende Programm noch weitere, aktuell ausgeblendete, Vorgänger hat, wurde dem Programm ein schmaler und etwas höherer Knoten vorangestellt . Ein Klick mit der Maus auf diesen Knoten zeigte die Ladeprozesse. Ein weiterer Klick versteckte sie wieder.

Ein Knoten wird genau dann Teil eines Fächers wenn:

a) er keinen Vorgänger hat

b) der Nachfolger mindestens fünf Vorgänger hat

c) alle Vorgänger des Nachfolgers (also alle Geschwisterknoten) keine Vorgänger haben

Im nächsten Schritt wurden die Bewegungen der Knoten beeinflusst. Dafür wurden dem force-directed Algorithmus weitere Kräfte hinzugefügt. Dies geht mit dem Framework d3js sehr einfach, da bei jedem Schritt ein „tick"-Event ausgelöst wird, auf das mit eigenen Methoden reagiert werden kann.

Allen Kräften gemein ist, dass sie mit dem Faktor alpha (a) multipliziert werden. Dadurch unterliegen sie genauso dem Geschwindigkeitsverfall wie die Kräfte aus der Bibliothek, beispielsweise Gravitation oder Ladung. Der Reibung unterliegen sie jedoch nicht, wie dort bereits erwähnt.

Bei jedem Tick wird folgendes getan:

1. **Knoten auf die Zeichenfläche beschränken**

   Am Anfang entstehen durch die hohe Knotenzahl, die alle auf einer Linie aufgereiht sind, sehr große Kräfte, die die Knoten so weit weg katapultieren, dass die Gravitation sie nicht mehr zurückholen kann. Daher werden die Ränder der Zeichenfläche schlicht als Wand simuliert und jeder Knoten, der darüber hinaus bewegt wurde, wird auf den Rand zurückgesetzt. Die Geschwindigkeit des Knotens wird ebenfalls auf 0 gesetzt, indem die vorherige Position (x', y') mit der neuen Position am Rand synchronisiert wird. Es wird also für Knoten v auf Höhe y und vorheriger Höhe y' und einer Höhe der Zeichenfläche h gerechnet:

   $$y = \begin{cases} h * -4 & \text{für } y < h * -4 \\ h * 4 & \text{für } y > h * 4 \end{cases} \tag{3.3}$$

   Der Faktor 4 hat sich während der Entwicklung als gute Grenze ergeben. Kleinere Werte beschränkten die Bewegungen zu sehr, größere Flächen waren für die Gravitation nicht mehr kontrollierbar.

2. **Einzelne Knoten und verbundene Knoten trennen**

   Einzelne Knoten (Knoten ohne Vorgänger oder Nachfolger) werden von den verbundenen Knoten getrennt, da diese oft zwischen den Ketten gefangen waren und die entstandenen Kraftgrenzen nicht überwinden konnten. Trotzdem sorgten sie für viel Unruhe in den Ketten, die wiederum nicht weg konnten, weil sie von den Kanten zusammengehalten wurden. Das Wegbewegen der einzelnen Knoten hat überraschend gute Ergebnisse erzielt. Die Simulation kam deutlich schneller zum Gleichgewicht und die visuelle Ordnung war deutlich besser. Die Kraft wird in Kombination mit der nächsten berechnet. Diese kam jedoch erst später dazu. Vorher wurde für einen Knoten auf Höhe y gerechnet (a ist der Kühlungsfaktor, wie unter 3.5 beschrieben):

   $$y = y + \begin{cases} +30 * a & \text{für Knoten mit Nachbarn} \\ -30 * a & \text{für Knoten ohne Nachbarn} \end{cases} \tag{3.4}$$

   Die 30 haben sich während der Entwicklung als guter Wert herausgestellt. Größere Werte wären für die Gravitation schwierig geworden und hätten große Leerräume erzeugt. Kleinere Werte hätten die beiden Gruppen nicht ausreichend getrennt.

3. **Knoten mit ähnlicher Verspätung zusammen bringen**

   Nachdem in einer ersten Iteration nur die einzelnen Knoten von den Ketten getrennt wurden, hat sich gezeigt, dass die gleichzeitig laufenden einzelnen Programme sehr unterschiedliche Verspätungen und Laufzeitvarianzen haben. Dadurch entstanden bunte Klumpen aus Programmen, die es schwer machten, die Programme zu bestimmen, die verspätet oder in sonstiger Weise auffällig waren. Um das Farbchaos etwas einzudämmen sollen Knoten mit ähnlicher Verspätung zusammengebracht werden. Dadurch wird die Farbverteilung homogener und man kann auf einen Blick die Programme bestimmen, die beispielsweise zu spät starteten. Um die Verspätung abzuschätzen, wird das Sextil verwendet, in dem sich die aktuelle Verspätung befindet. Die genaue Erläuterung dazu findet sich im Abschnitt 3.7 bei der Beschreibung der Farbgebung.

   Diese und die vorherige Kraft wird für einen Knoten auf Höhe y und im Verspätungssextil z so verrechnet:

   $$y = y + \begin{cases} +(30 - 7 * (z - 3)) * a & \text{für Knoten mit Nachbarn} \\ -(30 - 7 * (z - 3)) * a & \text{für Knoten ohne Nachbarn} \end{cases} \tag{3.5}$$

   Durch diese Kombination der Zahlenwerte ist sichergestellt, dass Knoten mit Nachbarn auf jeden Fall nach unten bewegt werden, und von den einzelnen Knoten getrennt werden, unabhängig von ihrer Verspätung. Trotzdem hat die Verspätung einen deutlich sichtbaren Einfluss. Von der Sextilzahl werden 3 abgezogen, damit Programme, die nahe dem Verspätungsmedian ($z = 3$) liegen, in der Mitte landen. Außerdem wird so verhindert, dass die Verspätung die Trennung der verbundenen und unverbundenen Knoten überdeckt, da $-21 <= 7 * (z - 3) <= 21$, da $0 <= z <= 6$.

4. **Knoten in Fächern auf ähnliche Höhe wie Nachfolger bewegen**

   Um die Symmetrie zu erhöhen, werden die (ausgeblendeten) Knoten auf ihren Nachfolger zubewegt. Dadurch entstehen Strukturen wie am linken Rand der Abbildung 3.4 (mit (b) markiert). Verschobene Fächer, mit vielen Kantenkreuzungen wie (a) werden vermieden. Für einen Knoten v auf Höhe y_v aus dem Fächer von Programm w mit der Y-Position y_w wird gerechnet:

   (3.6) $y_v = y_v + (y_w - y_v) * a$

5. **Ähnliche Knoten aufeinander zubewegen**

   Diese Kraft wurde erst am Ende der Entwicklung eingeführt, hatte aber große Auswirkungen. Beim Empfang der Daten durch den Browser werden jedem Knoten, der nicht Teil eines Fächers ist, ähnliche Knoten gesucht (die *Freunde* des Knoten). Fächerknoten werden hier nicht beachtet, weil sie bereits durch den Nachfolger zusammengehalten werden. Die Freunde werden auf Basis der Namen bestimmt. Da diese aus einem fest definierten Namensraum stammen, funktioniert diese Gruppierung sehr gut (vgl. Kapitel 2.4). Um die ähnlichen Knoten zu bestimmen werden die Buchstaben des Namens an den Stellen 3 bis 8 untersucht. D.h. das „DW“ am Anfang und das „P“ am Ende werden ignoriert, da diese bei allen gleich sind. Stimmen mehr als 3 dieser 5 Literale überein, werden die Knoten als ähnlich gewertet, solange die zeitliche Distanz nicht mehr als 10% des vom Benutzer selektierten Zeitbereichs ausmacht. So soll verhindert werden, dass Knoten ganz links am Rand Einfluss auf Knoten in der Mitte oder ganz rechts nehmen. Bei Ladeprogrammen, die Daten aus dem Warenwirtschaftssystem verarbeiten, wird verlangt, dass alle Literale übereinstimmen, da diese keine Typ-Definition tragen und dort die Namensnähe nicht unbedingt Datennähe impliziert.

   Um die Knoten aufeinander zuzubewegen kommt ein ähnlicher Ansatz wie bei der vorherigen Kraft zum Einsatz. Für ein Programm v an Position (x_v, y_v) mit befreundeten Knoten $f \in f_1, .., f_n$ mit Y-Positionen $y_i, 1 <= i <= n$ und dem Durchschnitt dieser Y-Positionen $\overline{y}$ wird gerechnet:

   (3.7) $y_v = y_v + (\overline{y} - y_v) * a$

   Anschließend wird das Array der Freunde neu gemischt, da sich sonst die Position des befreundeten Knotens zu stark auswirkt, der ganz am Ende des Knoten-Arrays steht. Seine Position wird ja zuletzt verändert, hat aber auf alle anderen vorherigen Knoten Einfluss.

   In Abbildung 3.5 ist der Effekt der Kraft sehr gut zu erkennen: Ganz links sind vier Gruppen (die mittlere sind eigentlich zwei, aber sehr nah beieinander, da es einige Elemente gibt, die Freunde in beiden Gruppen haben). Rechts sieht man eine Kette aus unverbundenen Programmen. Dies ist jeweils ein und dasselbe Programm, das alle 15 Minuten gestartet wird und nur einige Sekunden läuft.

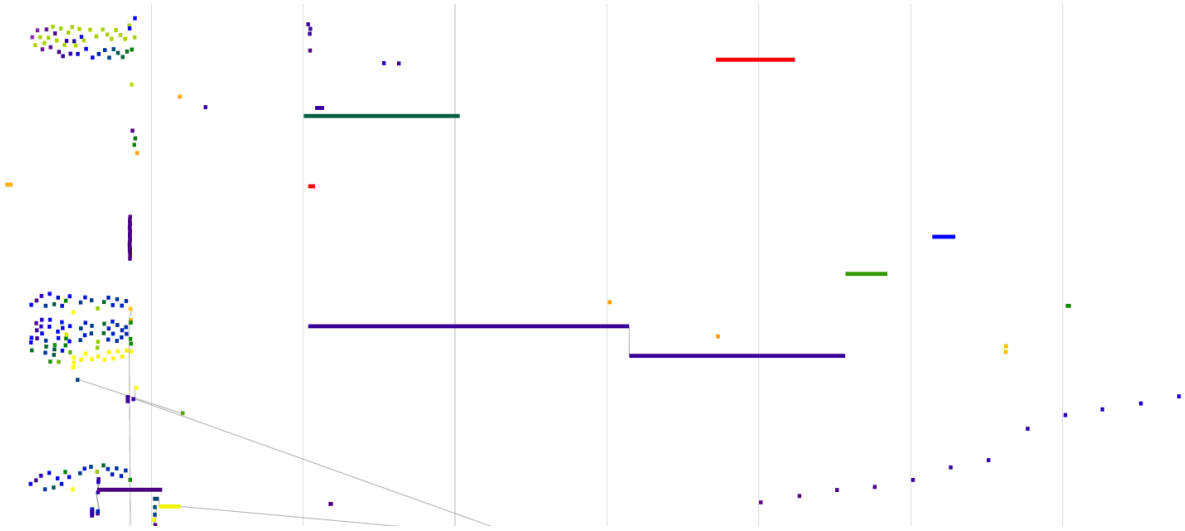


Abbildung 3.5.: Ähnliche und daher gruppierte Knoten. Die dünnen senkrechten Linien stellen die Zeit dar und repräsentieren im aktuellen Fall einen Stunden-Takt.

3.7. Farbgebung

Farbe ist ein wichtiger Teil der Visualisierung. Mit Hilfe von Farbe lässt sich einfach und schnell eine grobe Einschätzung von einem Wert vermitteln. Dies wird in dieser Arbeit für die Darstellung der Programme genutzt. Die Kanten werden nicht eingefärbt, da diese sonst zu sehr von den Programmen ablenken und damit zu visual clutter führen würde. Wie im Abschnitt 3.9.1 erläutert, kann der Anwender auswählen, welche Metrik (Verspätung oder Laufzeit) von der Farbe kodiert wird.

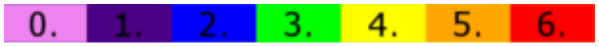
Die Transformation wird als abschnittsweise lineare Funktion implementiert. Die sieben Regenbogenfarben aus Abbildung 3.6a bestimmen die Abschnitte. Jeder Regenbogenfarbe wird der Wert eines Sextils der ausgewählten Metrik zugewiesen. Da die Abstände zwischen den Sextilen nicht notwendigerweise gleich lang sind, handelt es sich nur um eine abschnittsweise lineare Funktion. Zwischen den Werten kann jedoch linear interpoliert werden. Diese Interpolation wird von d3js durchgeführt [Bos13].

So ist sichergestellt, dass die Verspätung oder Laufzeit eines Programms immer im richtigen Kontext dargestellt wird. Für ein Programm, dessen Offset oft um mehrere Stunden schwankt, ist eine Verspätung von einigen Minuten weniger stark zu vermerken, als für ein Programm, das jeden Tag auf die Sekunde genau gestartet wird. Durch die abschnittsweise lineare Abbildung bedarf damit ein orange bei einem stark schwankenden Programm die gleiche Aufmerksamkeit wie bei einem sehr pünktlichen Programm. Würde nur linear durchinterpoliert werden, wäre dem nicht so.

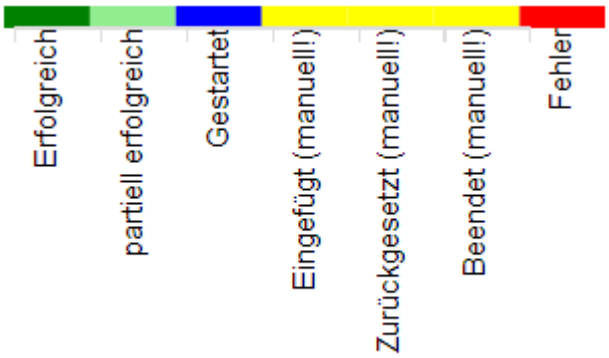
Es wurden Sextile gewählt, da mit Quintilen der Median nicht enthalten wäre und mit allen Quantilen darunter nicht genug Farben vorhanden wären. Der Median ist wichtig, da er den

erwarteten Startzeitpunkt bzw. die erwartete Laufzeit darstellt. Er sollte also grün markiert werden. Mehr als sieben Farben wären zu schwer zu unterscheiden. Damit sind Sextile die beste Wahl.

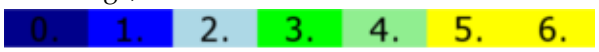
Außerdem kann der Nutzer die Farbgebung ändern und eine dichromatische Farbskala auswählen, beispielsweise um Farbfehlsichtigkeiten zu umgehen. In diesem Fall werden nicht die Regenbogenfarben verwendet, sondern nur Blau, Grün und Gelb, wie in Abbildung 3.6c und 3.6d dargestellt.



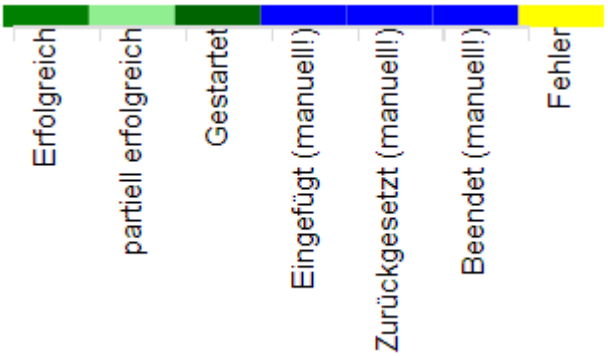
(a) Die Regenbogenfarbskala für Laufzeit und Verspätung mit Violett, Indigo, Blau, Grün, Gelb, Orange, Rot.



(b) Farbkodierungen aller möglichen Status in Regenbogenfarben.



(c) Die dichromatische Farbskala für Laufzeit und Verspätung mit Dunkelblau, Blau, Hellblau, Grün, Hellgrün, Gelb, Gelb.



(d) Farbkodierungen aller möglichen Status in dichromatischen Farben.

Abbildung 3.6.: Alle möglichen Farbskalen der Anwendung. Oben die Regenbogenfarben, unten die dichromatischen Skalen. Bei den Farbskalen für die Laufzeit und Verspätung (3.6a und 3.6c) wird zwischen den einzelnen Farben von d3js linear interpoliert. Bei den Skalen für den Status ist dafür kein Bedarf, da die Statuswerte diskret sind.

Die Statuscolorierung funktioniert sehr viel einfacher: Jedem Status ist eine Farbe zugeordnet, grob nach Schweregrad sortiert. Diese Farbskala ist in Abbildung 3.6b und 3.6d dargestellt. Den manuellen Status wurde gelb (bzw. blau) zugewiesen, da sie einen manuellen Eingriff

repräsentieren, der bei der Analyse von großer Wichtigkeit sein kann. Er deutet an, dass ein Fehler passiert ist und erfordert die Aufmerksamkeit des Anwenders.

3.8. Interaktive Elemente

In der Gesamtansicht ist die Visualisierung zu unübersichtlich. Nur durch Interaktion können die Daten erkundet und zu hilfreicher Information verwandelt werden.

3.8.1. Pan und Zoom

Daher wurde ein semantischer Zoom implementiert. Dieser brachte jedoch den Layout-Algorithmus durcheinander, da sich durch das Zoomen die Distanzen zwischen den Knoten änderten. Um dieses Problem zu lösen, wurde dem Layout-Algorithmus ein eigenes Koordinatensystem mit fester Breite und Höhe gegeben, das am Anfang anhand der vom Browser festgestellten Anzeigegröße definiert wird. Mit Hilfe von Scales wird dieses von d3js transformiert. Im äußeren Koordinatensystem wird dann auch der Zoom und das Panning[4] eingerechnet [BOH11].

Während der Entwicklung hat sich gezeigt, dass der Zoom zwar gut funktioniert um eng beieinander liegende Programme zu untersuchen, jedoch war dann zwischen den Programmen viel Platz. Daher wurden die Knoten beim Zoomen nicht nur in der Länge verändert, sondern auch in der Höhe. Folgende Formel für die Knotenhöhe h und Zoomfaktor z wurde verwendet:

$$z' = \begin{cases} (z-4)*2 & \text{für } z >= 4 \\ 0 & \text{sonst} \end{cases} \tag{3.8}$$

$$h = \begin{cases} 3 & \text{für } z < 1 \\ 3+z+z' & \text{für } 1 <= z <= 5\frac{2}{3} \\ 12 & \text{sonst} \end{cases} \tag{3.9}$$

So wird sichergestellt, dass die Knotenhöhe 3 nicht unterschreitet und 12 nicht überschreitet, egal wie weit hinaus- bzw. hineingezoomt wird. Durch z', das sich nur für einen Zoomfaktor zwischen 4 und $5\frac{2}{3}$ auswirkt, wird das Anwachsen der Knoten in diesem Bereich etwas beschleunigt. Da in diesem Bereich der Effekt des Zooms am höchsten ist, wird hier der Knotenhöhe etwas nachgeholfen. Durch die größeren Knoten ließen sich diese auch leichter mit der Maus anfahren.

[4]Panning ist das Verschieben des Sichtfeldes in der Anwendung.

3.8.2. Hervorhebung

Um Teile der Ketten besser untersuchen zu können, wurde Highlighting implementiert. So erkennt man besser, welche Programme Abhängigkeiten zu dem ausgewählten Programm haben und wann diese gelaufen sind. Beim Hervorheben werden auch die Namen aller hervorgehobenen Programme angezeigt, damit man diese nicht einzeln anfahren muss um sie zu identifizieren. Diese Funktion ist in Abbildung 3.7 dargestellt.

Bei der Hervorhebung werden alle Knoten, die nicht Teil der Kette sind, mit 90% Transparenz ausgeblendet. So stechen die hervorgehobenen deutlich hervor. Auch hier wurde mit mehreren Werten experimentiert. Weniger Transparenz hätte zu Unsicherheiten geführt, welche Knoten nun ein- oder ausgeblendet sind, insbesondere bei hellen Farben wie Gelb. Ganz ausblenden erschien ebenfalls nicht sinnvoll, da möglicherweise durch die Suche ein Programm gefunden wird und dann die damit verbundenen Knoten analysiert werden sollten. Würden diese nicht mehr angezeigt, weil die Suche auf sie nicht zuträfe, müsste die Suche erst abgeschaltet werden. Dann müsste sich der Anwender aber merken, wo sein gesuchtes Programm in der Visualisierung war. Daher wurde der Weg der starken Transparenz gewählt.

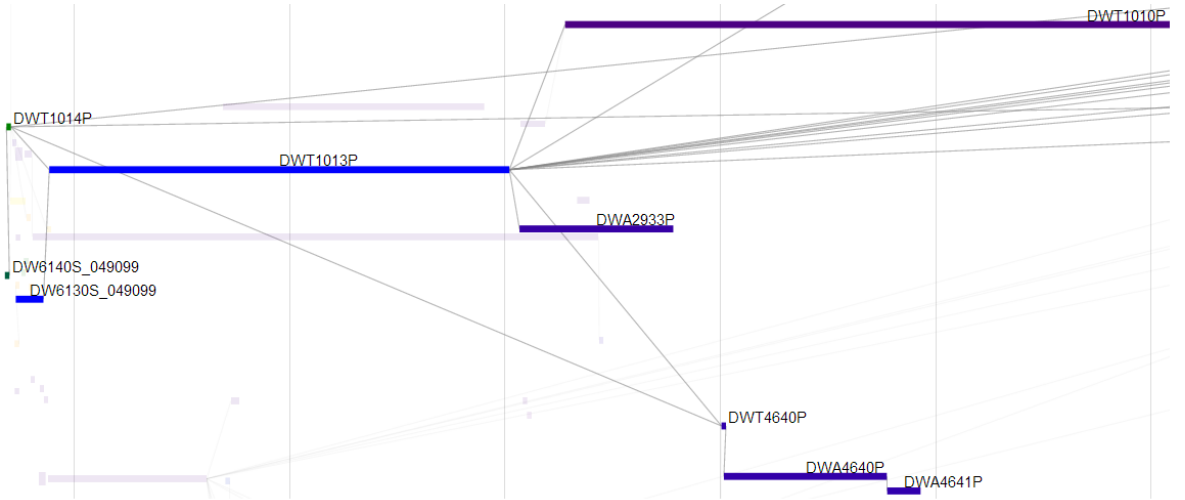
3.9. Steuerung der Anwendung

Um die Visualisierung zu steuern und zusätzliche Informationen zu liefern, die nicht direkt in der Visualisierung angezeigt werden können, wurden zwei zusätzliche Bereiche geschaffen. Zum einen die Navigationsleiste am linken Rand zur Steuerung der Anwendung. Zum anderen das Popup-Menü am rechten oberen Rand, das Informationen zum jeweils ausgewählten Programm liefert.

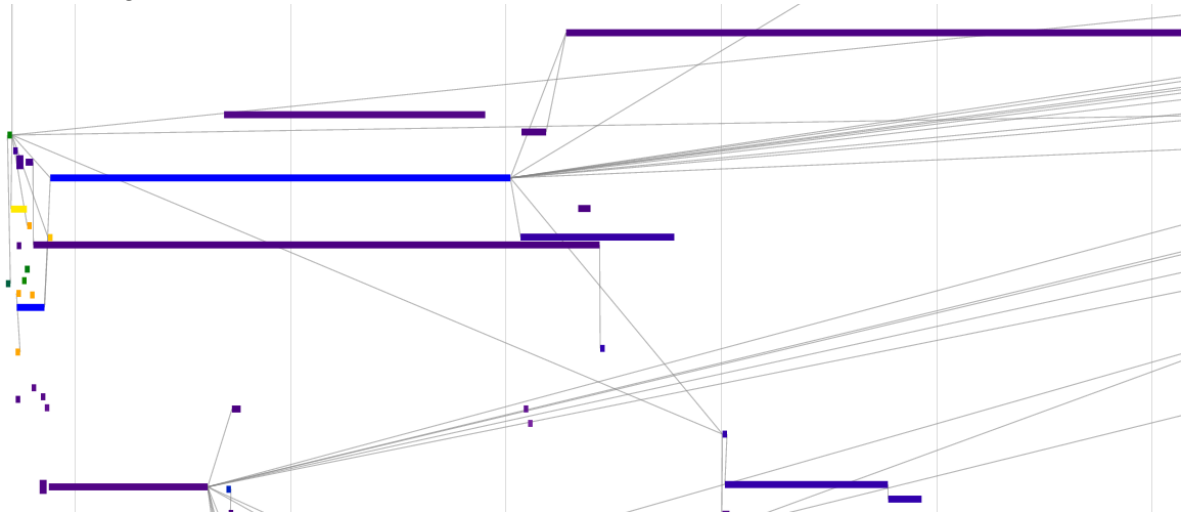
3.9.1. Navigationsleiste

Die Navigationsleiste ermöglicht die Auswahl des Zeitraums, das Starten der Anwendung und die Suche nach Programmen, sowie weitere grobe Filterungen nach Status des Laufs sowie nach Intervalltyp . Außerdem lässt sich hier die Farbskala auswählen, die verwendet werden soll. Es stehen die Regenbogenfarben zur Auswahl sowie eine dichromatische Farbskala auf Basis von Blau und Gelb für Rot-Grün-Blinde. Die Navigationsleiste kann in Abbildung A.1 auf Seite 68 betrachtet werden.

Zusätzlich lässt sich hier die Metrik auswählen, die von der Farbe kodiert wird. Es stehen drei Metriken zur Auswahl. Standardmäßig wird der Offset dargestellt, der Benutzer kann aber zusätzlich die Laufzeit oder den Status als farbgebend auswählen. So werden viele Analysen optimal unterstützt. Mit der Auswahl des Status kann sofort erkannt werden, ob ein Programm abgebrochen ist, oder ob noch welche aktiv sind. Mit der Möglichkeit die Laufzeit farblich darzustellen, können Programme identifiziert werden, die möglicherweise



(a) Hervorgehobene Kette des *DWT1013P*.



(b) Gleicher Ausschnitt ohne Hervorhebung zum Vergleich.

Abbildung 3.7.: Hervorhebung.

mit einem schlechten Ausführungsplan laufen oder die besonders viele Daten zu verarbeiten haben und daher für Verzögerungen sorgen.

3.9.2. Popup-Menü

Dieses Menü wurde ursprünglich als Kontextmenü konzipiert und in einer Entwicklungsstufe auch so implementiert. Es sollte beim Überfahren eines Knoten erscheinen. Jedoch hat sich herausgestellt, dass es zu groß war und zu viel der Visualisierung überdeckte. Auch die Möglichkeit es zu verschieben brachte keine Besserung. Außerdem sind nicht immer die angezeigten Informationen von Interesse, beispielsweise wenn man nur die Kette

verfolgt und sich für die Details der Programme nicht interessiert. Dabei wurde das ständige Auftauchen des Popups sowie das dafür nötige Laden der Daten vom Server als sehr störend empfunden.

Daher wurde das Popupfenster fest in die rechte obere Ecke gesetzt. Da sehr viele Informationen dort untergebracht werden sollten, wurde es mit Tabs versehen, so dass der Anwender genau die Ansicht auswählen kann, die für ihn relevante Daten enthält. Die Daten werden jeweils aktualisiert, wenn ein Programm in der Visualisierung angeklickt wird. So können einzelne Programme anhand verschiedener Metriken schnell und einfach verglichen werden.

Dank der Tabs finden hier auch die Einträge aus den Logfiles Platz. Dazu mehr im Abschnitt 3.9.4.

3.9.3. Histogramme

Die Histogramme repräsentieren das, was die Whisker in der Zeichnung der Grundidee in Abbildung 3.1 auf Seite 26 eigentlich darstellen sollten. Während der Entwicklung wurde deutlich, dass die Programme eine sehr große Streuung aufzeigen, vor allem im Offset. Die Whisker wurden zeitweise implementiert, jedoch wurden sie bei einigen Programmen so groß, dass die Enden nicht mehr angezeigt werden konnten. Sie hätten also keinen Informationsgewinn gebracht.

Als nächstes wurde probiert, die Laufzeitvarianz direkt in der Visualisierung darzustellen. Dazu wurden beim Mouseover für jedes Sextil der Laufzeit ein Balken in die Zeichnung eingefügt und entsprechend eingefärbt. Allerdings war das nur schwer zu interpretieren und wenig intuitiv. Weiterhin überdeckten die eingeblendeten Balken andere Programme und Kanten, wie in Abbildung 3.8 dargestellt. Daher wurde beschlossen, die Daten in Histogrammform anzuzeigen. Histogramme erlauben eine detaillierte und intuitive Analyse der Verteilung.

Die Histogramme zeigen die Offsets bzw. Laufzeiten von allen Läufen des ausgewählten Programms. Es werden jedoch nur die Läufe verwendet, die am selben INTVL_INDEX liefen. D.h. für ein tägliches Programm, das einen Montag verarbeitet hat, werden nur Läufe angezeigt, die ebenfalls einen Montag verarbeitet haben. So werden nur die relevanten Daten angezeigt.

Es werden 2 Histogramme erzeugt. Eins für die Verspätungen, eins für die Laufzeit. Beide werden mit Hilfe von d3js erzeugt, wie es die Dokumentation in einem Beispiel erläutert[5] [Bos13].

Um die Analyse noch weiter zu vereinfachen, werden die Histogrammdaten so transformiert, dass sie angeben, wie sich die restlichen Offsets im aktuellen Fall verhalten hätten. Es werden also nicht die rohen Offset-Werte angezeigt, sondern die Offsets werden mit dem aktuellen

[5] http://bl.ocks.org/mbostock/1933560

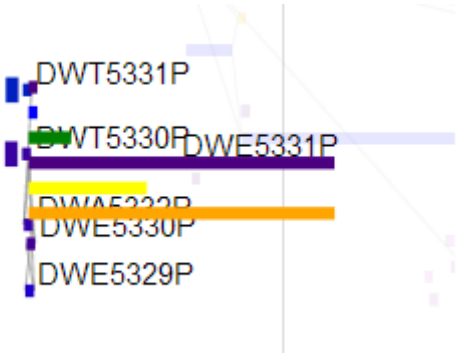


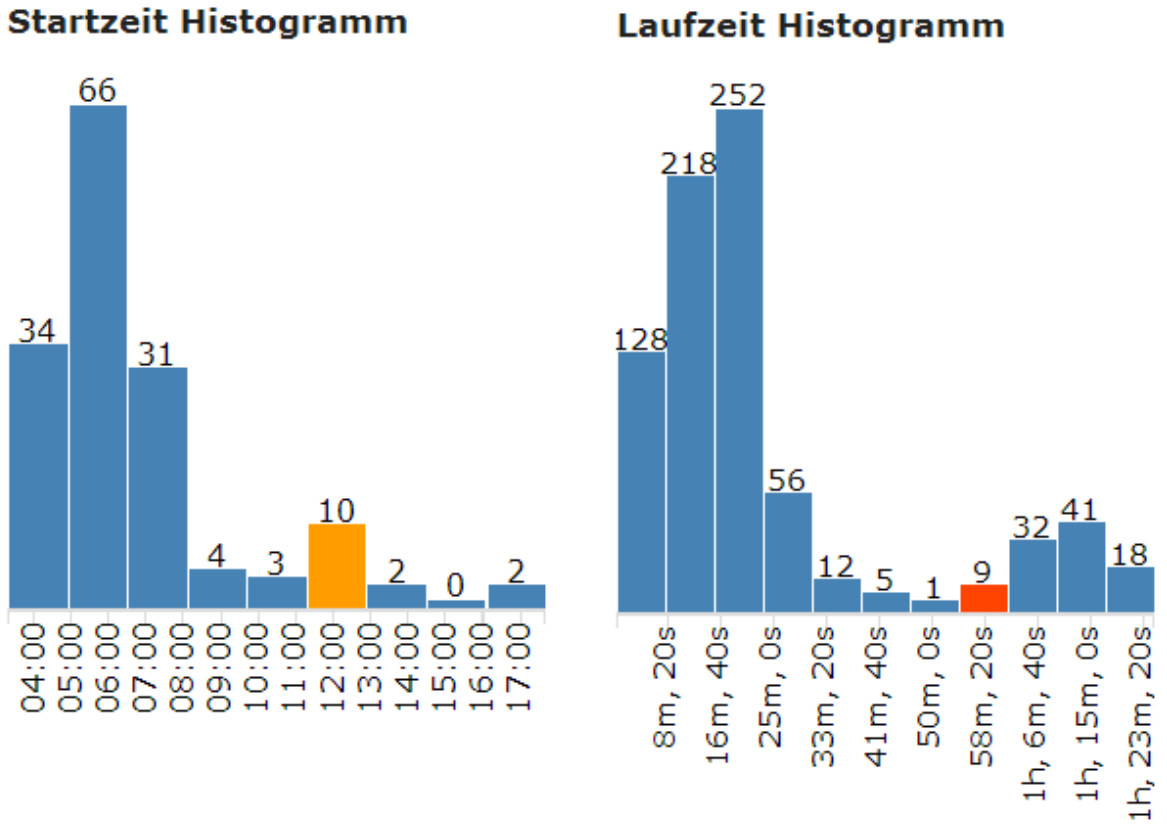
Abbildung 3.8.: Darstellung der Varianz der Laufzeiten in der Visualisierung. Ausgewählt wurde *DWE5331P*, der lange violette Balken. Die violette Färbung kommt von der Startzeitmetrik. Die drei Balken direkt darüber (violet, blau, grün) sowie die 2 darunter (gelb, orange) repräsentieren die Sextile der Laufzeitverteilung. Gut zu erkennen ist der erzeugte Visual Clutter durch Überdeckung anderer Knoten, Kanten und Namen.

Intervalldatum addiert. So sieht man auf einen Blick, wann die Mehrheit der Programmläufe im aktuellen Fall fertig geworden wäre.

Ein Beispiel: Das Programm *DWA1402P* verarbeitet am 07.03.2013 den 06.03.2013 (ein Mittwoch) und startet um 6:30 Uhr. Der Offset dieses Programmlaufs beträgt 30 Stunden, 30 Minuten. Nun werden aus der Datenbank alle anderen Offsets für dieses Programm geholt, die auch einen Mittwoch verarbeitet haben. Anschließend werden diese Offsets mit dem Verarbeitungsdatum (06.03.2013 00:00:00 Uhr) addiert und diese Daten werden als Histogramm dargestellt. Der Balken, in dem sich der aktuelle Offset befindet, wird mit der Farbe eingefärbt, die entsprechende Farbskala vorgibt. So muss nicht die X-Achse nach dem entsprechenden Balken abgesucht werden.

Das Laufzeithistogramm ist einfacher in der Berechnung. Hier werden lediglich alle Laufzeiten aus der Datenbank selektiert, die den entsprechenden INTVL_INDEX verarbeitet haben und als Histogramm dargestellt. Auch hier wird wieder der aktuelle Balken mit dem Farbwert für die aktuelle Laufzeit eingefärbt.

Bei der Entwicklung hat sich gezeigt, dass es bei sehr vielen Programmen jeweils eine kleine Anzahl extremer Ausreißer gibt, die die Histogramme stark verzerren. Daher wurde die Selektion der verwendeten Offsets und Laufzeiten eingeschränkt: Es werden nur die Offsets bzw. Laufzeiten angezeigt, die innerhalb von 3,3 Standardabweichungen um den Median liegen. Unter einer Normalverteilung betreffen das 99,9% aller Daten. Auch wenn die Offsets und Laufzeiten sicherlich nicht normalverteilt sind, hat sich gezeigt, dass das gute Werte liefert.



(a) Histogramm der Startzeit. Der gelb markierte Balken enthält den aktuellen Offset. Gut zu erkennen ist, dass das dargestellte Programm eine deutliche Verspätung von mehreren Stunden hat.

(b) Histogramm der Laufzeit. Der orangerote Balken enthält die aktuelle Laufzeit. Hier ist ein Programm dargestellt, das deutlich länger lief als üblich. Allerdings gab es auch schon Läufe, die noch länger dauerten.

Abbildung 3.9.: Die Histogramme.

3.9.4. Logfiles

Die Logfiles der Stored Procedures enthalten viele interessante Informationen. Beispielsweise welche Tabellen verändert werden, wieviele Datensätze von der Stored Procedure verwendet, geschrieben oder geändert wurden und Fehlermeldungen. Die Logfiles werden jedoch nicht automatisch erzeugt, sondern durch explizite Aufrufe in der Stored Procedure, geschrieben von den Entwicklern. Diese waren bisher erste Anlaufstelle bei der Analyse, wo der Fehler liegt und was ihn ausgelöst hat. Daher müssen diese auch unbedingt in der Visualisierung auftauchen.

Um die Analyse angenehmer zu gestalten, werden die Logfiles von der Server-Applikation geparsed und nach Datum und Uhrzeit aufgeteilt. Im ersten Schritt (also wenn der Anwender das Programm in der Visualisierung anklickt) wird nur der Teil des Logfiles geladen

und angezeigt, der für diesen Lauf relevant ist. Dies geschieht über die Zeitstempel des Laufeintrags. Am Ende des Logfiles wird aber ein Knopf eingeblendet, mit dem das gesamte Logfile geladen werden kann. Dann wird der aktuelle Teil farblich hervorgehoben, um die Orientierung zu erleichtern.

Weiterhin werden die Zeilen des Logfiles in eine 2-spaltige Tabelle übernommen, um mit der kleinen Breite des Popup-Fensters zurecht zu kommen. In die linke Spalte wird das Datum eingetragen, in die rechte Spalte kommt die eigentliche Nachricht. Dies funktioniert sehr einfach, da für das Logging eine einheitliche Schnittstelle in den Stored Procedures verwendet wird, die alle Nachrichten nach dem folgenden Schema formatiert: `YYYY-MM-DD HH:MI:SS - Nachricht` Eine typische Zeile aus den Logfiles sieht so aus: `2013-08-01 01:12:22 - Start Einlesen Datei DW1153S_049001.txt_20130731`.

Die Einbindung der Logfiles ist in Abbildung 3.10 dargestellt.

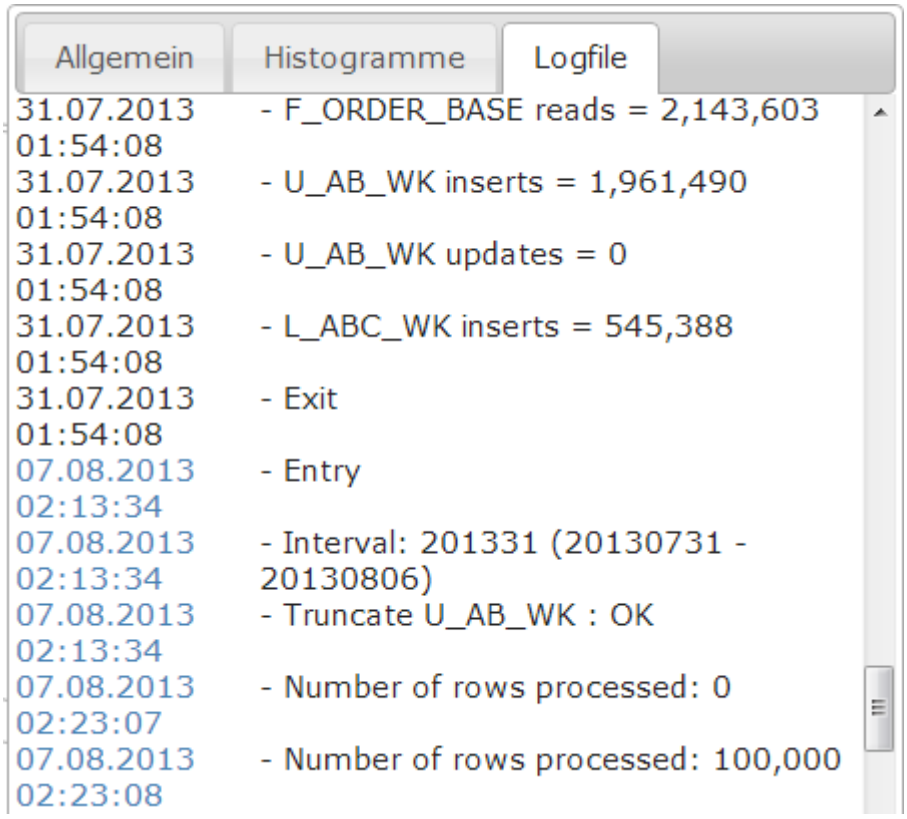


Abbildung 3.10.: Darstellung des Logfiles.

Damit sind alle Elemente der Anwendung erläutert und vorgestellt.

4. Evaluierung

Um zu überprüfen, ob die Visualisierung hilfreich und korrekt ist, wurden Experteninterviews durchgeführt. Als erstes wird der Versuchsaufbau erläutert. Der Personenkreis stellt die Probanden anhand einiger relevanter Attribute vor. Daran schließt sich der Ablauf der Evaluation an. Am Ende stehen die Ergebnisse und eine Diskussion derselben.

4.1. Versuchsaufbau

Der Versuchsaufbau bestand aus einem Laptop in einem Besprechungsraum der Firma mit einem Browserfenster das die Applikation bereits geladen hatte. Angeschlossen an den Rechner war ein 24″-Monitor mit einer Auflösung von 1920x1080 sowie Tastatur und Maus. Der Laptop lief unter Windows XP, hatte 3 GB RAM und einen Intel Core 2 Duo Prozessor mit 2 Kernen. Als Browser kam Chrome (Version 29) zum Einsatz.

Der Server der Anwendung wurde mit der Produktivdatenbank des Unternehmens verbunden um aktuellste Daten anbieten zu können. Außerdem wurden jeweils kurz vor Beginn der Interviews die Logfiles der Stored Procedures vom Datenbankserver auf den Server mit dem Tool kopiert.

Vor Beginn jeder Evaluierung wurde eine Instanz des Tools geladen. Als Zeitraum für die Visualisierung wurde der 06.03.2013 20:00 Uhr bis 07.03.2013 08:00 Uhr ausgewählt.

Außerdem standen den Probanden sämtliche Entwicklungstools zur Verfügung, die er auch an seinem normalen Arbeitsplatz hat. Dazu gehören die beiden Entwicklungsumgebungen Quest Toad for Oracle[1] sowie SQL Developer[2] von Oracle, Token 2[3] als Telnet-Client und Filezilla[4] als FTP-Client.

Als Probanden sollten Experten befragt werden, die sich mit Datenbanken und Stored Procedures professionell beschäftigen und voraussichtlich von der Visualisierung profitieren können. Weiterhin sollten sie ein Verständnis für die visualisierten Daten mitbringen um die Einführung kurz zu halten und damit sich die Evaluierung auf die Visualisierung konzentrieren kann. Ein Einfluss der Qualität der Erläuterungen sollte vermieden werden. Da der zu erwartende Personenkreis sehr klein ist, wurde auf qualitative Merkmale wert gelegt.

[1] http://www.quest.com/toad-for-oracle/
[2] http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html
[3] http://www.choung.net/products/discontinued/
[4] http://filezilla-project.org/

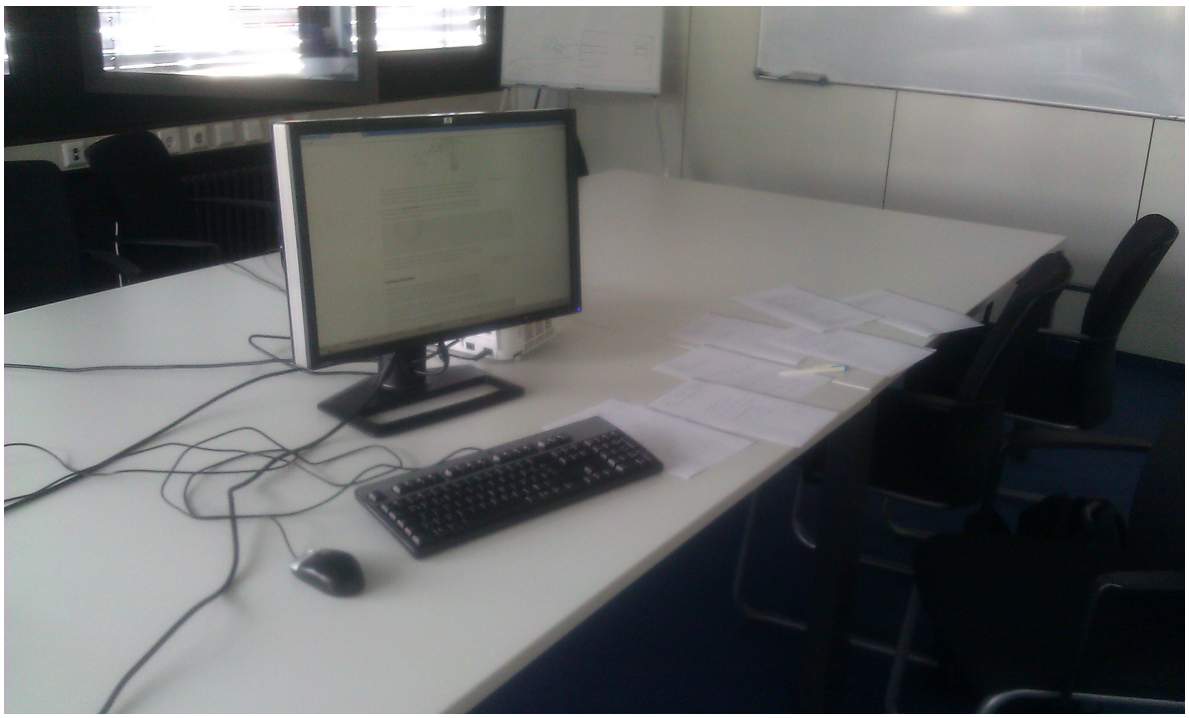


Abbildung 4.1.: Der Versuchsaufbau. Dargestellt ist der Monitor, Maus und Tastatur im Besprechungsraum.

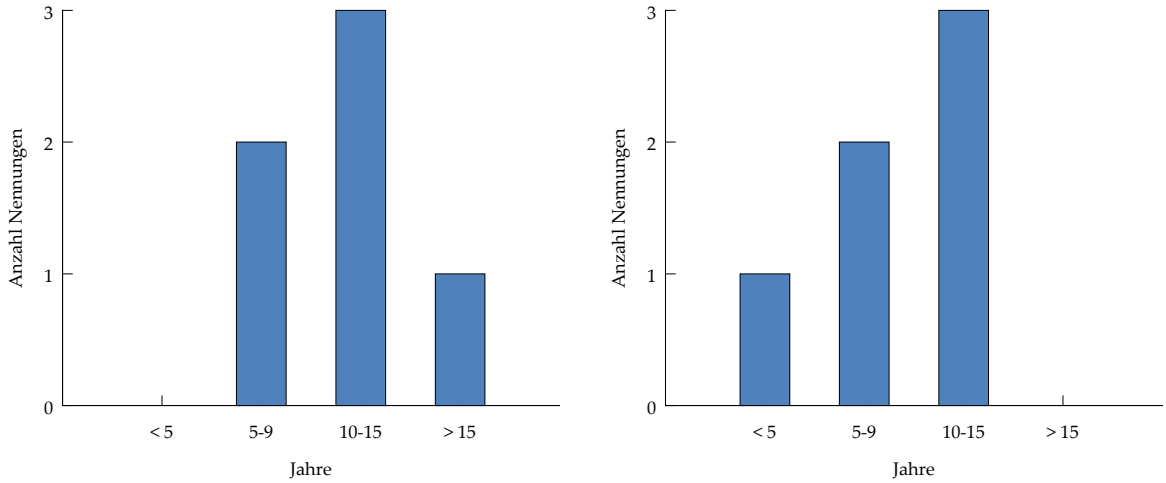
4.2. Personenkreis

Insgesamt wurden sieben Personen befragt. Alle arbeiten in der Entwicklungsabteilung für das Data Warehouse des Unternehmens, das die Daten für diese Diplomarbeit bereit gestellt hat. Sie entwickeln die Stored Procedures und arbeiten täglich mit den visualisierten Daten. Ihre Erfahrung mit Datenbanken liegt im Mittel bei 11,5 Jahren (Median: 11, Standardabweichung: 4,9) und reicht von 5 bis 20 Jahren. In Anbetracht dessen, dass es relationale Datenbanken als kommerzielles Produkt erst seit 1981 gibt [SS07, S. 112], deckt das einen sehr großen Teil der möglichen Erfahrungsspanne ab. Die Erfahrung mit dem CTL-Tool liegt im Schnitt leicht darunter (9,3), im Median jedoch gleichauf (11). Das Maximum liegt hier bei 15 Jahren. Das liegt wiederum daran, das es das CTL-Tool erst seit 1998 gibt. Die Standardabweichung beträgt 5,5 Jahre.

Weitere persönliche Details der Personen wurden nicht erfasst.

4.3. Ablauf der Validierung

Der Ablauf der Validierung wurde so geplant, dass die Gesamtdauer eine Stunde nicht übersteigt, um Ermüdungseffekte ausschließen zu können. Der grobe Ablauf unterteilte sich



(a) Erfahrung der Teilnehmer mit Datenbanken in Jahren.

(b) Erfahrung der Teilnehmer mit dem CTL-Tool in Jahren.

Abbildung 4.2.: Aggregierte Erfahrung der Personen.

in drei Teile: Einführung und Erklärung, Aufgaben durchführen und eine abschließende Befragung. Der Ablauf ist in Abbildung 4.3 dargestellt.

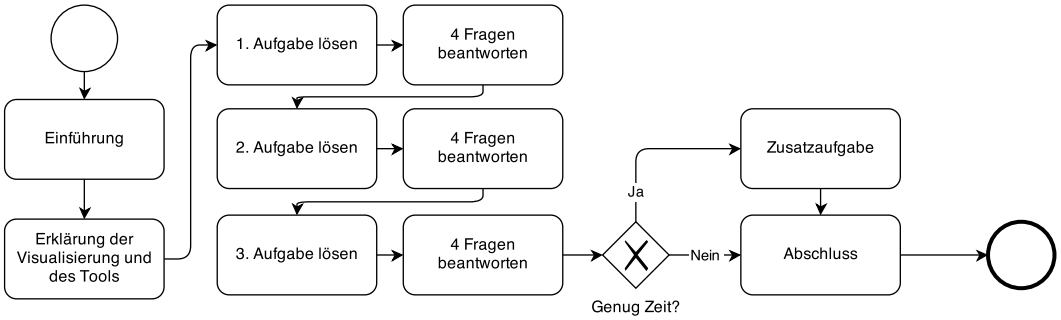


Abbildung 4.3.: Der Ablauf eines Experteninterviews in BPMN 2.0-ähnlicher Notation.

Einführung (5 Minuten) Der Versuchsperson wurde die Vorgehensweise der Validierung erläutert. Es wurde darauf hingewiesen, dass in erster Linie der Ansatz der Visualisierung getestet werden sollte und nicht das Tool in seiner Gesamtheit. Die Kernfrage wurde darauf festgelegt, ob die Visualisierung nützlich sei, ob sie bei der täglichen Arbeit hilfreich sein könne und ob sie besser zu verwenden sei als bisher verfügbare Methoden.

Erklärung der Visualisierung und des Tools (5 Minuten) Als erstes wurde das Konzept des Offsets und der Verspätung erläutert. Anschließend wurde die Oberfläche erklärt, die

Möglichkeiten zur Interaktion vorgestellt und dem Probanden das Popup-Menü mit allen Histogrammen und dem Logfile gezeigt. Anschließend hatten die Teilnehmer Gelegenheit, das Tool auszuprobieren, Fragen zu stellen und eigenen Fragestellungen mit dem Tool nachzugehen. So konnten die Teilnehmer die erste Neugier befriedigen, ein Gefühl für das Werkzeug und die Visualisierung bekommen und die Aufgaben anschließend flüssig bearbeiten.

Aufgaben (30 Minuten) Nun mussten die Experten drei Aufgaben lösen und dazu jeweils drei Fragen beantworten.

Die Aufgaben waren:

1. Finde das Programm, das am 2.8.2013 am zweitlängsten lief.
2. Finde heraus, wann der letzte Nachfolger des DWT1030P im Tagesabschluss vom 6.3.2013 fertig wurde.
3. Finde heraus, warum das Programm DWT2215P im Tagesabschluss vom 2.8.2013 nicht wie üblich um 9 Uhr morgens fertig war.
4. Löse eine alltägliche Aufgabe mit dem Tool.

Die Lösungen für die Aufgaben sind im Anhang in den Abbildungen A.1 bis A.3 dargestellt. Das gesuchte Programm ist jeweils mit einem roten Pfeil markiert.

Bei Aufgabe 1 wurde das Zweitlängste gewählt, damit die Probanden mit dem Zoom und dem Popupmenü arbeiten müssen, da es mehrere mögliche Kandidaten gibt. Diese Aufgabe ist relevant, wenn es darum geht, Kandidaten für die Optimierung zu suchen. Gerade bei Langläufern lohnt es sich, die Programme und deren Laufzeiten regelmäßig zu analysieren, da hier die Möglichkeiten zum Zeitgewinn am größten sind.

Bei Aufgabe 2 wurde eine Situation nachgestellt, bei der zu klären ist, wann eine bestimmte Kette von Daten, beispielsweise vom einen externen Datenlieferanten, abschließend verarbeitet ist, der Tagesabschluss also im Sinne dieser Daten fertig ist. Das ist sowohl bei der Suche nach Optimierungsmöglichkeiten interessant als auch in Situationen, in denen Wartungsfenster geplant werden sollen.

Aufgabe 3 testet eine typische Situation aus der Fehleranalyse. Häufig kommen von Benutzern Anfragen, wann bestimmte Daten verfügbar sind oder warum Berichte keine Daten liefern, die um diese Zeit normalerweise funktionieren.

Nach jeder Aufgabe wurden den Teilnehmern vier generische Fragen gestellt und die Antworten aufgeschrieben. Die Fragen waren:

1. Ist das Tool hilfreich bei der Beantwortung der Frage?
2. Geht das besser als jetzt? Wieso?
3. Welche Funktionalität fehlt, damit es besser ginge?
4. War die Aufgabe realistisch?

Damit sollte überprüft werden, ob die Visualisierung eine Besserung darstellt. Außerdem sollten so Schwachpunkte der Visualisierung im realen Einsatz gefunden werden und ob die Annahmen über die Aufgaben korrekt waren.

Zusatzaufgabe (5 Minuten) Je nachdem ob noch genug Zeit verfügbar war, wurde den Probanden nach diesen drei vordefinierten Aufgaben eine Zusatzaufgabe gegeben. Dabei wurden die Probanden gebeten zu versuchen, eine alltägliche Aufgabe mit dem Tool zu lösen und ihre Erkenntnisse und Eindrücke mitzuteilen.

Abschließende Befragung (10 Minuten) Nachdem die Probanden alle Aufgaben erledigt haben, wurden einige abschließende Fragen gestellt:

1. Für welche Aufgaben könntest du dir vorstellen, dass das Tool hilfreich ist?
2. Welche Erwartungen an die Visualisierung wurden nicht erfüllt?
3. Was war gut?
4. Ist die Gesamtübersicht über den Tagesabschluss hilfreich?

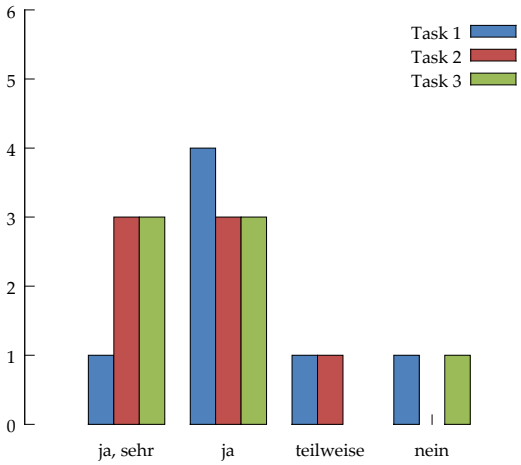
Damit sollten Schwächen und Chancen des Tools identifiziert werden.

Außerdem wurden einige statistische Informationen abgefragt, um untersuchen zu können, ob die Erfahrung bei der Bewertung der Visualisierung eine Rolle spielt.

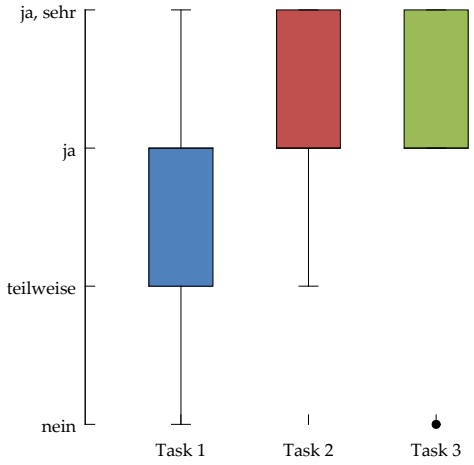
1. Wie lange arbeitest du schon im Datenbankbereich?
2. Wie lange arbeitest du schon mit dem CTL-Tool?
3. Wie viel Zeit verbringst du wöchentlich mit der Fehler- und Statusanalyse?

4.4. Ergebnisse der Validierung

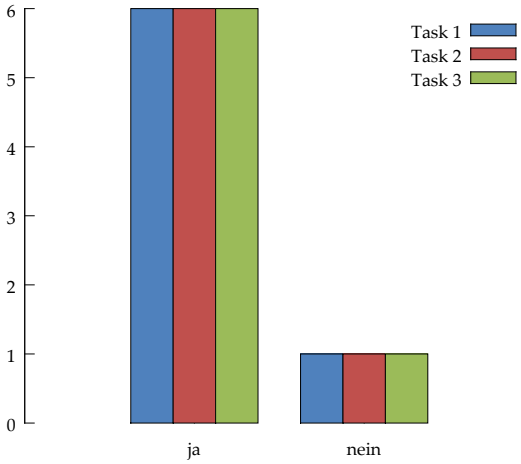
Insgesamt fiel das Urteil sehr positiv aus. Bei jeder Aufgabe befanden sechs Teilnehmer, dass die Aufgabe mit dem Tool besser zu lösen sei als ohne. Ebenso wurde das Tool mit einer großen Mehrheit als hilfreich bei der Beantwortung der Fragen bewertet. Die aggregierten Antworten sind in den Abbildungen 4.4a bis 4.4f dargestellt.



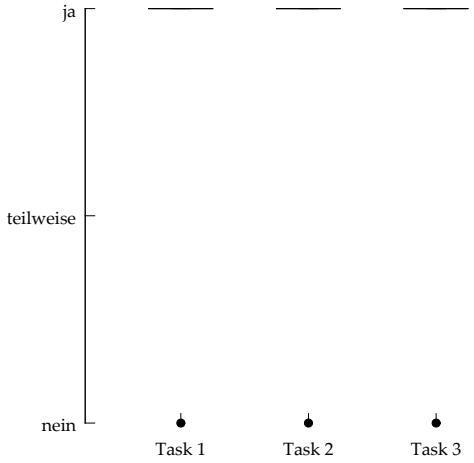
(a) Ist das Tool hilfreich?



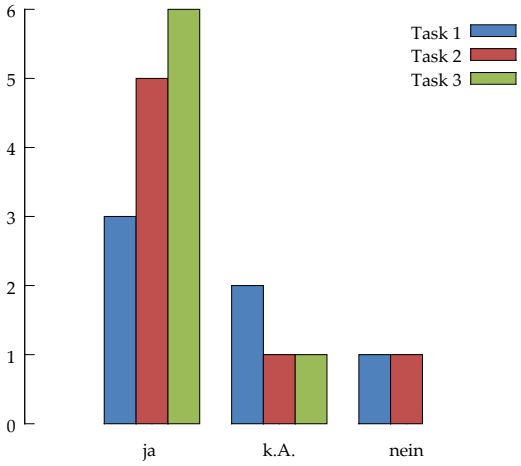
(b) Boxplot zu Abbildung 4.4a.



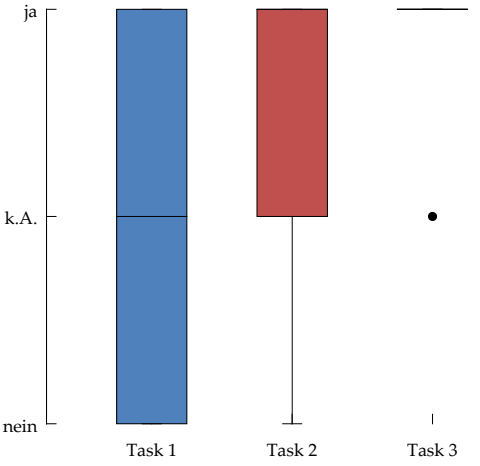
(c) Geht es mit dem Tool besser?



(d) Boxplot zu Abbildung 4.4c.



(e) Ist die Aufgabe realistisch?



(f) Boxplot zu Abbildung 4.4e.

Abbildung 4.4.: Aggregierte Antworten auf die Fragen nach den Aufgaben.

4.4.1. Aufgabe 1

Bei der ersten Aufgabe fielen die Ergebnisse am schlechtesten aus (vgl. Abbildung 4.4b). Ein Grund für die schlechte Bewertung ist, dass ein einfaches SQL-Statement die Frage leichter und schneller beantwortet, als dass das die Visualisierung tut. Es wurde bemängelt, dass die Reihenfolge nach absoluter Laufzeit nicht klar zu erkennen sei und der Längenvergleich der Balken schwer falle. Es mussten also jeweils Kandidaten bestimmt werden und diese einzeln, anhand der Werte im Popup-Menü, verglichen werden. Das war jedoch geplant und erwünscht, trotzdem war es überraschend, dass dieser Aufwand so negativ aufgenommen wurde.

Alle Kandidaten bemerkten jedoch, dass die Auswahl des längsten Programms sehr leicht fällt, wie auch in Abbildung A.1 gut zu erkennen ist. Die Mehrheit der Teilnehmer (4) vermissten eine Hervorhebung der Top X mit einer Einstellmöglichkeit, welche Metrik verwendet wird (Laufzeit, Abweichung von der Normallaufzeit, Offset oder Verspätung) um so die Frage auf einen Blick beantworten zu können. Ebenso oft wurde eine direkte Filtermöglichkeit basierend auf diesen Metriken verlangt, so dass beispielsweise nur noch besonders stark verspätete Programme angezeigt werden.

Ein Teilnehmer vermisste bei dieser Aufgabe eine Verlaufsanzeige für die Laufzeit und den Offset, um entscheiden zu können, ob eine Veränderung dieser Werte eine stabile Entwicklung sei, oder ob es sich um einen einmaligen Ausreißer handle.

Die Aufgabe wurde als am wenigsten realistisch bewertet. Ein Teilnehmer begründete diese Bewertung mit Zeitmangel für Reengineering oder Performance-Optimierung bestehender Stored Procedures, was darauf schließen lässt, dass die Aufgabe an sich realistisch sein könnte, wäre mehr Zeit dafür vorhanden oder würde Performance-Tuning konkret vom Management oder den Kunden angefordert werden. Die anderen Teilnehmer gaben keine Gründe für ihre Bewertung an.

4.4.2. Aufgabe 2

Aufgabe 2 wurde als realistischer empfunden (Fünf Teilnehmer wählten „ja“). Wenn Gründe dafür angegeben wurden, waren es dieselben Gründe, die auch für die Auswahl der Aufgabe gesorgt hatten, also Benutzeranfragen bezüglich der Frage, wann welche Daten verfügbar seien. Diese Annahme war also richtig.

Hier fehlte den meisten Teilnehmern (vier) eine Möglichkeit, die Hervorhebung festzuhalten („Sticky Highlighting“). In der jetzigen Form musste man sich merken, welche Programme Teil der Kette sind, während man die Maus zum gesuchten Objekt bewegte. Sobald der Mauszeiger das Objekt verlässt, geht die Hervorhebung verloren und man verliert schnell den Blickkontakt zur Kette. Besonders bei starkem Zoom, wenn das Programm am Anfang der Kette aus der Anzeige gerät, wird die Pfadverfolgung sehr schwierig.

Trotzdem wurde das Tool als sehr hilfreich empfunden (3x „ja, sehr“, 3x „ja“). Ein Teilnehmer sagte wörtlich: „Um Faktor 50 schneller als die bisherige Methode!“. Die visuelle Darstellung

wurde am häufigsten als Grund für die Besserung durch das Tool genannt, sowie der Wegfall des hohen Aufwands. Ein Teilnehmer bezeichnete jedoch die Analyse mit dem Tool als „schneller, aber nicht besser“, da es visuell schwierig wäre, zu entscheiden, welches Programm nun das letzte gewesen sei. Eine tabellarische Darstellung, chronologisch sortiert, wäre besser gewesen. Ein weiterer Teilnehmer bevorzugte die bisherige Analysemethode mit Hilfe der Logfiles und bemängelte, dass der Sollzustand nicht ausreichend angezeigt werde. So könne nicht entschieden werden ob die Kette tatsächlich fertig wäre, weil Programme, die noch gar nicht gelaufen sind, nicht angezeigt werden.

Der Sollzustand war auch Teil eines weiteren Kritikpunkts. Es wurde vorgeschlagen, eine Möglichkeit zu schaffen, manuell Sollwerte für bestimmte Programme zu definieren. So könnten auch Prozesse untersucht werden, bei denen die bisher verwendete, statistische Methode fehlschlägt und falsche Ergebnisse liefert. Des Weiteren wurde von einem Teilnehmer die Farbgebung kritisiert. Da die Farbskala nicht helligkeitsstabil ist, sind gelb gefärbte Programme sehr schwer zu erkennen.

4.4.3. Aufgabe 3

Aufgabe 3 war die realistischste Aufgabe. Alle Teilnehmer hatte sie bereits im Alltag erlebt. Positiv bemerkt wurde die übersichtliche Darstellung des Tools. Die 10-Stunden-Lücke zwischen DWL2202P und seinen Vorgängern wird sofort erkannt und als Grund für die Verspätung von DWT2215P angebracht.

Negativ fiel auf, dass diese Situation nicht abschließend mit dem Tool gelöst werden konnte, da nicht klar wird, warum DWL2202P so lange nicht lief. Allerdings wurde von vielen Probanden gemutmaßt, dass ein externes System für das Starten der Prozedur und somit für die Verspätung verantwortlich sei. Dass ein externes System in Verbindung mit dem Programm steht, konnte aus dem Logfile geschlossen werden, in dem die verwendeten Tabellen genannt werden, deren Namen wiederum auf das externe System hindeuteten.

Mehr als die Hälfte (4) wünschten sich eine größere Datenbasis zur Visualisierung. Beispielsweise Logs des Systems, des Schedulers oder gar der externen Systeme sollten einen tieferen Einblick gewähren, sodass die Frage abschließend beantwortet werden könnte. Erneut wurde Sticky Highlighting vermisst (2 Nennungen), damit man die Kette des DWT2215P hervorheben könnte und dann am Anfang besser mit Zoom und Panning sehen könnte, was die Vorgänger seien. Ein Teilnehmer vermisste die Möglichkeit, die Datenbasis vor der Anfrage an den Server bereits einzuschränken, um die Ladezeiten zu verringern.

4.4.4. Weitere Ergebnisse

Im Rahmen der eigenen Aufgaben und dem freien Herumprobieren am Anfang der Evaluation haben die Probanden meist Programme untersucht, die entweder sehr im Zentrum liegen und deshalb von großem Interesse waren oder von Ihnen selbst geschrieben wurden.

Anhand dieser bekannten Programme wurde am Anfang meist untersucht, ob das Tool vertrauenswürdig ist, d.h. ob die Programme wie erwartet angezeigt werden. Zwei Teilnehmer überprüften für sie überraschende Ergebnisse mit dem Entwicklungstool Toad for Oracle.

Ein Teilnehmer entdeckte während der Validierung einen Fehler in einem Programm. Dort wurden langwierige Aktivitäten durchgeführt bevor die Meldung abgegeben wurde, dass das Programm gestartet ist. Dadurch kam es in der Visualisierung zu einer großen Lücke zwischen dem Vorgänger und dem Programm, für die es scheinbar keine Erklärung gab und die, wie die Analyse der jeweiligen Histogramme ergab, auch regelmäßig auftrat.

Ein weiterer Teilnehmer untersuchte die aktuelle Tagesverarbeitung und stellte fest, dass einige wichtige Programme noch nicht gelaufen waren. Dabei bemerkte er jedoch, dass es schwierig ist, zu beurteilen, ob tatsächlich alles gelaufen ist, da nicht angezeigt wird, was noch nicht gelaufen ist. So muss man wissen, was laufen muss, und dann mit Hilfe der Suche und dem Hervorheben von Ketten analysieren, ob und wann die Programme gelaufen sind.

Desweiteren zeigt sich die bereits im Grundlagenkapitel 2.4 erwähnte Hardware-Erweiterung sehr deutlich. Wählt man die Verspätung als Metrik für die Farbe sind die Verarbeitungen nach der Erweiterung deutlich blauer bzw. violetter als vorher. Gut sichtbar ist das in A.1. Dort ist der Lauf vom 2.8.2013 (also nach der Erweiterung statt fand) dargestellt und ein großer Teil der langen Ketten ist deutlich lila gefärbt, wurde also früher als sonst gestartet.

Es steht zu überlegen die Statistiken zur Analyse nur für den jeweiligen Zeitraum zu wählen, d.h. vor oder nach der Erweiterung, um diesen Effekt auszuschließen. Schließlich sind die Programme nur deswegen als verfrüht dargestellt, weil der Server schneller ist und nicht aus Gründen wie geringe Datenmenge oder gar ein Problem der Jobsteuerung.

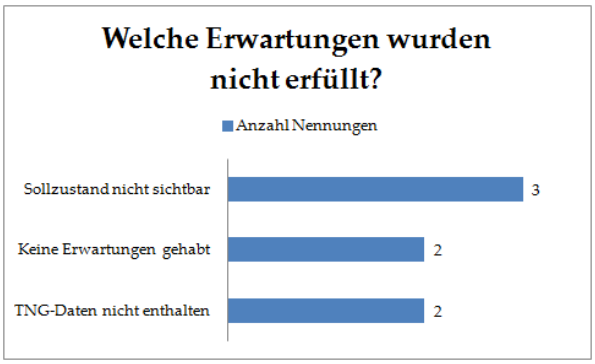
Allerdings gibt es für nach der Erweiterung für viele Programme noch nicht ausreichend Vergleichsdaten, sodass dort gar keine Statistik betrieben werden könnte. Daher wurden die Daten gerade auch für die Evaluation, die im September stattfand, so belassen. Für die Zukunft, sollte das Tool produktiv eingesetzt werden, müsste dieser Umstand aber nochmal beleuchtet werden. Dann stehen möglicherweise auch wieder ausreichend Vergleichsdaten zur Verfügung um die statistischen Analysen zu ermöglichen.

In dieser Situation wäre eine Verlaufsanzeige für Verspätung und Laufzeit sicherlich ein nützliches Analysewerkzeug, wie es ja auch von einem Teilnehmer gefordert wurde. So könnte der Analyst schnell bestimmen, ob die Verspätung seit der Hardware-Erweiterung gesunken ist oder ob möglicherweise ein Problem besteht.

4.4.5. Abschluss

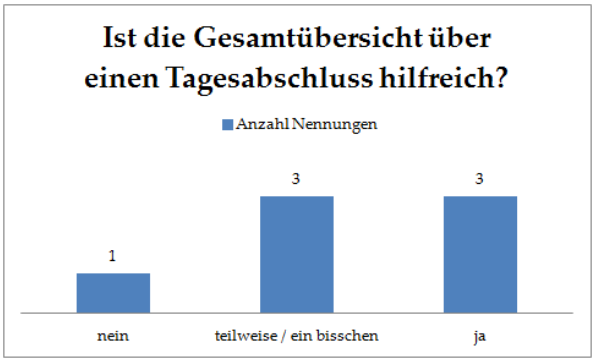
Überraschend ist, dass alle Teilnehmer das Tool für Performance-Tuning einsetzen wollen, jedoch nur vier Teilnehmer einen Nutzen beim Fehlertracking oder der Fehleranalyse sehen, wofür das Werkzeug eigentlich entwickelt wurde (vgl. Abbildung 4.5a). Zwei Teilnehmer sehen einen Nutzen bei der Erstellung von Dokumentation und Ablaufplänen. Die einmal

(a) Mögliche Einsatzgebiete.



(b) Unerfüllte Erwartungen.

(c) Was war gut?



(d) Nützlichkeit der Übersicht.

Abbildung 4.5.: Aggregierte Antworten auf die Fragen im Abschluss.

genannte What-If-Analyse betrifft vor allem die Darstellung der gemeinsamen Abhängigkeiten, da hier beispielsweise überprüft werden kann, welche Programme alle davon betroffen wären, würde eine Datenquelle wegfallen.

Wie in Abbildung 4.5d dargestellt, wird die grobe Übersicht über den Tagesabschluss von drei Teilnehmern als hilfreich eingeschätzt, weitere drei können ihr zumindest teilweise etwas abgewinnen, beispielsweise in der Retrospektive oder um einen ganz groben Eindruck davon zu bekommen, „wie der Tagesabschluss lief“. Wenn er als nicht hilfreich bewertet wurde, wurde betont, dass mit der Interaktion und der Möglichkeit zum Zoomen, sich diese Einschätzung ändert.

Auf die ganz allgemeine Frage, was gut gefallen hat, wurde am häufigsten die gesamte Visualisierung genannt, was darauf schließen lässt, dass das Werkzeug gern eingesetzt werden würde, sobald einige Anfangsschwierigkeiten behoben sind. Die Übersicht empfanden weitere drei Teilnehmer als besonders hervorzuheben. Jeweils einem Teilnehmer gefielen die Navigationsmöglichkeiten sowie die Darstellung der Logfiles besonders gut.

Gefragt, was denn fehlen würde und welche Erwartungen nicht erfüllt wurden, antworteten alle, zunächst keine Erwartungen gehabt zu haben. Die Entwicklung fand auch ausschließlich an der Universität statt, Vorpräsentationen oder Abstimmung mit dem Unternehmen oder

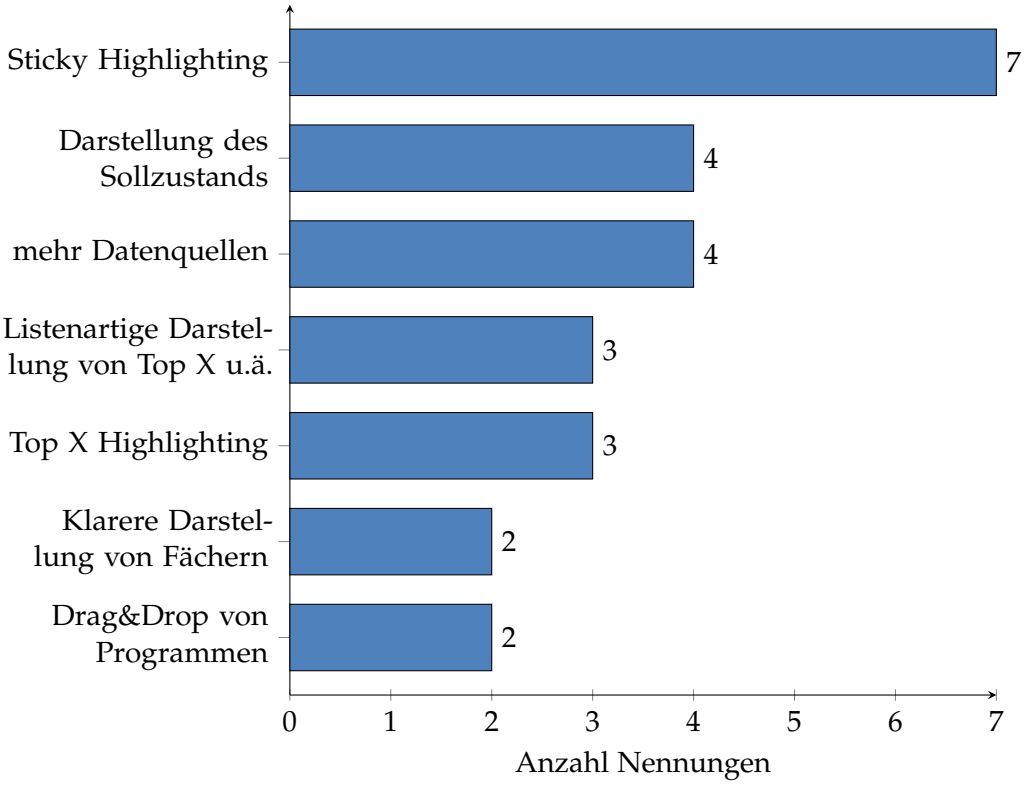


Abbildung 4.6.: Häufig genannte fehlende Funktionen der Visualisierung.

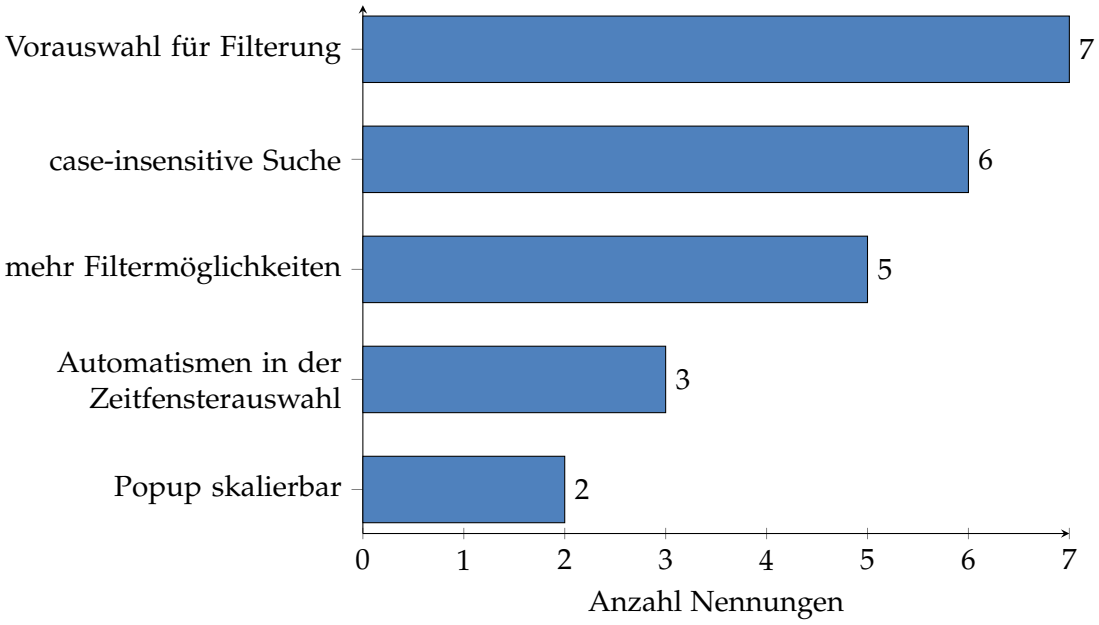


Abbildung 4.7.: Häufig genannte fehlende Funktionen der Anwendung.

gar den Teilnehmern fanden nicht statt. Auf Nachfragen, was denn generell nicht gefallen habe oder was noch fehlen würde, damit das Tool hilfreich sei oder bei der Arbeit helfen könnte, wurde von allen Sticky Highlighting genannt. Darauf folgen der Sollzustand und der Wunsch nach mehr Datenquellen. Da das Tool immer nur anzeigt, was tatsächlich gelaufen ist, könne nicht mit letzter Sicherheit gesagt werden, ob das alles war oder ob noch Programme fehlen. Diese fehlenden Programme sollten als Liste dargestellt werden. Alle drei Teilnehmer die das bemängelten gaben jedoch auch an, dass die Bestimmung dieser Liste nicht 100%-ig möglich ist, da die Definitionen im CTL-Tool nicht unbedingt der Realität entsprechen. Die Ausnahmen verlangen eine gewisse Unschärfe sowie Vorsicht beim Umgang mit Vorhersagen.

Auch der Wunsch nach Integration von mehr Datenquellen zeigt, dass die Visualisierung zwar als gut empfunden wird und der Ansatz als vielversprechend angenommen wurde, jedoch die Datenbasis nicht ausreicht, damit das Tool im Alltag hilfreich sein kann. Als konkrete Quellen wurde interflex[5] sowie awr[6] genannt.

Auf die Frage nach nicht erfüllten Erwartungen kamen im Weiteren dann auch Aufzählungen von Funktionen, die das Tool besser bedienbar oder die Visualisierung aussagekräftiger machen würden. Diese Antworten sind in Abbildung 4.7 bzw. 4.6 dargestellt. Darin sind nur solche Antworten aufgezeichnet, die von mehr als einer Person genannt wurden. Von jeweils einer Person wurden die folgenden Funktionen vermisst:

- Verlaufsanzeige für Verspätung / Laufzeit.
- Anzeige der Namen ab einem bestimmten Zoomlevel.
- Darstellung der rein logischen Abhängigkeiten, ohne Kodierung der Zeit.
- Möglichkeit zur manuellen Definition von Soll- und Schwellwerten für die Metriken.
- Helligkeitsstabile Farbskala.

Der letzte Punkt ist durchaus interessant. Die Experteninterviews wurden in einem Besprechungsraum durchgeführt, bei dem die Fenster mit einer Jalousie verdunkelt waren, sodass der Monitor sehr gut ablesbar war. Trotzdem hatten die Teilnehmer hin und wieder Schwierigkeiten hellgelbe Programme zu lokalisieren. Auch war bei hell gefärbten Programmen die Unterscheidung zwischen hervorgehoben und nicht hervorgehoben schwierig. Eine helligkeitsstabile Farbskala würde diese Probleme umgehen. Daher ist die Nachfrage möglicherweise höher als die Anzahl Nennungen vermuten lässt, explizit gefordert wurde sie jedoch nur einmal.

[5] zur Zeit in der Entwicklung befindliches Framework zum Laden von Daten aus dem Warenwirtschaftssystem der Firma.

[6] Oracle Automatic Workload Repository. Sammelt Informationen über Auslastung der Datenbank und dem System.

4.5. Diskussion der Ergebnisse

Das überraschendste Ergebnis der Evaluation ist sicherlich, für welche Aufgaben das Tool als hilfreich eingeschätzt wurde. Hier wurde Performance Analyse und Optimierung der Abläufe zusammen mit What-If-Analysen doppelt so häufig genannt wie die Fehleranalyse. Ein Grund dafür ist sicherlich die mangelnde Datenbasis, die auch am häufigsten kritisiert wurde. Hinzu kommt der nicht dargestellte Sollzustand.

Durch unklare Aufgabentrennung innerhalb der Firma zwischen Betriebsabteilung und Entwicklungsabteilung kommt es außerdem zu Widerständen im Entwicklungsteam im Hinblick auf Tätigkeiten, die der Fehleranalyse zuzuordnen sind. Es ist nicht ganz auszuschließen, dass diese Situation Einfluss auf die Ergebnisse der Validierung hatte.

Mindestens genauso interessant ist die Schlussfolgerung, die bereits früh in der Diplomarbeit aus Abbildung 3.3 gezogen wurde: Viele Programme haben keinen Vorgänger oder Nachfolger. Dies überraschte viele Teilnehmer. Es wurde vermutet, dass entweder das CTL-Tool schlecht gepflegt ist oder eben nicht alle Abhängigkeiten abbilden kann.

Abbildung 4.8 zeigt den Einfluss der Erfahrung auf die Bewertungen. Dabei wurden die Bewertungen in Punkte umgerechnet. Die Umrechnung fand auf Basis der Antworten zu den ersten beiden generischen Fragen pro Task statt. Es wurden nur die ersten beiden Fragen beachtet, da nur dort annähernd quantitative Aussagen getroffen wurden. Die genaue Umrechnung ist in Tabelle 4.1 dokumentiert.

Aufgrund der kleinen Anzahl Teilnehmer erscheinen statistische Auswertungen nicht sinnvoll. Beachtenswert ist trotzdem der Anstieg und Abfall mit dem Höhepunkt bei 11 Jahren Erfahrung. Übersteigt die Erfahrung diesen Wert, wird das Tool als weniger hilfreich eingeschätzt. Möglicherweise liegt das daran, dass bei 20 Jahren Erfahrung, wie am rechten Rand, die kritischen Ketten bereits auswendig bekannt sind und bei der Fehleranalyse nicht mehr separat bestimmt werden müssen. Am Anfang, mit wenig Erfahrung, erschlägt die Visualisierung möglicherweise und ist daher nur beschränkt hilfreich. Mehr Hilfestellungen um die angezeigten Daten zu interpretieren, würden hier eventuell helfen. Allerdings müssten für solche Schlüsse noch deutlich mehr Evaluationen durchgeführt werden, um verlässliche Zahlen zu erhalten.

Als weiterer Nachteil lässt sich sagen, dass Hintergrundwissen notwendig ist, wenn einzelne Programme untersucht werden. Ein Beispiel dafür ist der DWT1500P, der zwar als tägliches Programm definiert ist und normalerweise den Vortag verarbeitet, aber jeden Tag auch regulär für bereits weiter zurückliegende Tage gestartet werden kann. Dabei verarbeitet er „nachgelieferte“ Daten für diese vergangenen Tage. Diese Läufe verfälschen die Statistik und lassen eigentlich keinen Vergleich der Verspätung zu.

Das war ein Grund für einen Wunsch eines Teilnehmers. Es sollte möglich sein, manuell Sollwerte für Programme zu definieren um solche Situationen besser behandeln zu können. Das zeigt jedoch, dass der Ansatz der Visualisierung angenommen wird und „nur“ die Datenbasis verbessert werden muss.

Bewertung	Punkte
ja, sehr hilfreich	3
ja, hilfreich	2
teilweise hilfreich	1
nicht hilfreich	0

(a) Ist das Tool hilfreich?

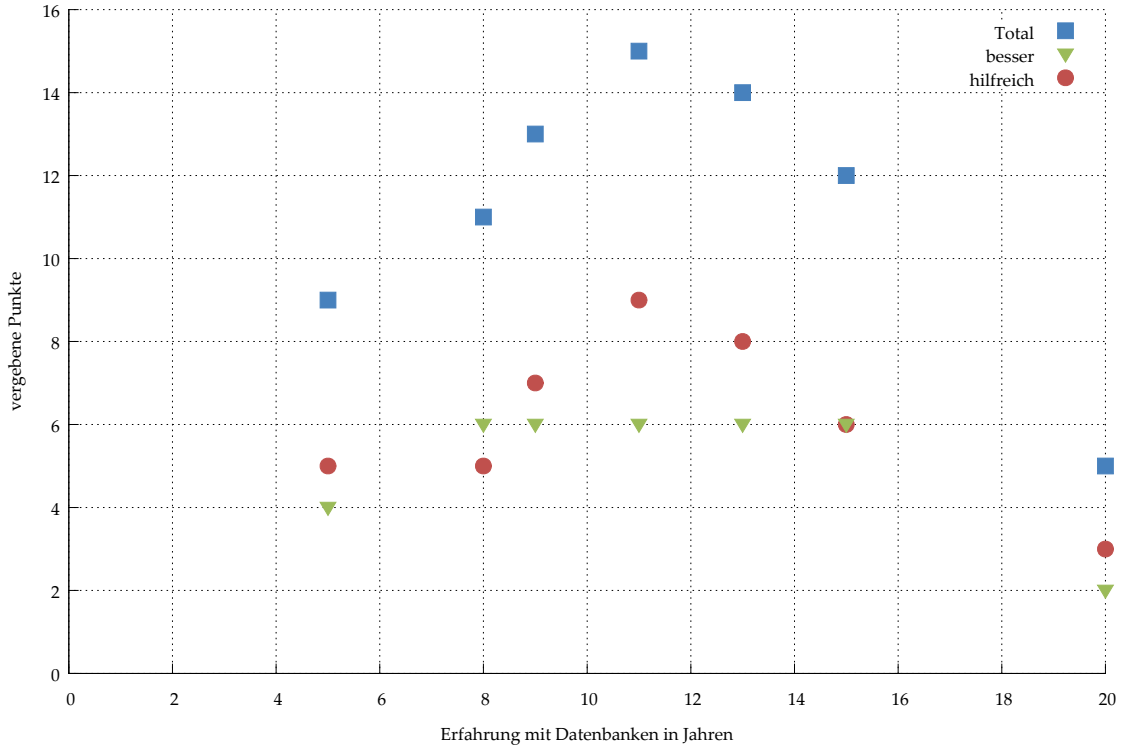
Bewertung	Punkte
ja, besser	2
nein, nicht besser	0

(b) Geht es mit dem Tool besser?

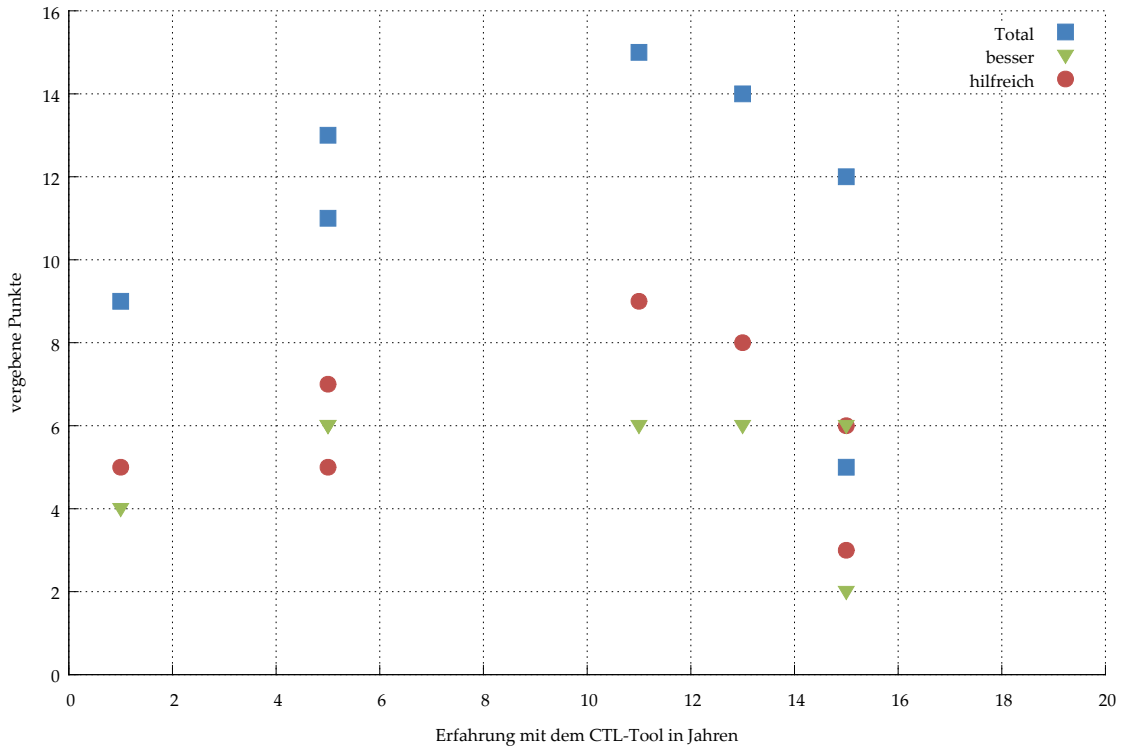
Tabelle 4.1.: Umrechung der Bewertungen in Punkte.

Das Sticky Highlighting so stark vermisst wurde, deutet darauf hin, dass die Untersuchung einzelner Ketten als wichtigstes Element der Visualisierung wahrgenommen wurde. Darunter fällt auch die Fehleranalyse im Falle eines abgebrochenen Programmes, wo herausgefunden werden muss, warum dieses Programm abgebrochen ist. Als erstes wird untersucht, ob bei den Vorgängern etwas passiert ist, das den Fehler hätte auslösen können. Die anderen Ketten sind in dem Fall erst mal irrelevant.

Überraschend war, dass *Drag & Drop* von Programmen nicht häufiger gefordert wurde. Insbesondere da ja so viele Programme keine Vorgänger haben, könnte die *Drag & Drop*-Funktionalität bei der Analyse sehr hilfreich sein. Genauso für Dokumentationszwecke, wenn das Layout einen Knoten zu weit wegschiebt. Andererseits spricht es natürlich für das Layout, wenn diese Funktion nicht so häufig vermisst wurde.



(a) Bewertungen vs. Erfahrung mit Datenbanken.



(b) Bewertungen vs. Erfahrung mit dem CTL-Tool.

Abbildung 4.8.: Bewertungen nach Erfahrung aufgeschlüsselt.

5. Fazit und Ausblick

Es wurde gezeigt, dass mit Hilfe eines force-directed Algorithmus die Abläufe von Stored Procedures in einem aussagekräftigen Layout angeordnet werden können. Durch Interaktion und sinnvolles Ausblenden von unwichtigen Elementen können die Information sehr gut erkundet werden.

In der Evaluation hat sich gezeigt, dass es großes Interesse an einem solchen Ansatz gibt und damit neue Erkenntnisse gewonnen werden können. Bisherige Analysemethoden benötigten sehr viel manuellen Aufwand und tiefste Kenntnis des Systems. Mit Hilfe der Visualisierung bekommt man eine Übersicht geliefert und visuelle Indikatoren, wo ein Problem liegen könnte.

In seiner ursprünglichen Form konnte der Layoutalgorithmus kein zufriedenstellendes Ergebnis liefern, was hauptsächlich an der Menge der Knoten lag, die bei durchschnittlich 1000-1200 dargestellten Knoten deutlich über dem liegt, wofür force-directed Layout-Algorithmen nach allgemeiner Meinung geeignet sind. Durch das Hinzufügen von problemspezifischen Kräften, die dem Algorithmus die Verteilung vereinfachten, konnte das Ausmaß der Unordnung jedoch stark reduziert werden. Ein weiterer großer Beitrag zur Lösung des Problems war die Reduktion der Knoten durch Zusammenfassung und Ausblenden der Fächer. Trotzdem bleiben einige Strukturen übrig, die zu visual clutter durch viele Kantenkreuzungen führten. Dieses Problem sollte in zukünftigen Arbeiten beleuchtet werden. Beispielsweise könnte der Layout-Algorithmus weiter an die Situation angepasst werden. Die Abstandsbestimmung und die Kräfteberechnungen sind noch nicht vollständig an die neue Situation angepasst und könnten eventuell noch mehr Kantenkreuzungen verhindern.

Gleichzeitig sollten die Möglichkeiten zur Interaktion ausgebaut werden, wie es auch mehrheitlich von den Teilnehmern der Validierung gefordert wurde. Zusätzliche Ansichten, wie die Darstellung rein logischer Abhängigkeiten, könnten das Tool noch deutlich verbessern.

Im jetzigen Zustand bietet das Tool einen Einstieg und Indikatoren, aber nicht genug Möglichkeiten, die Situation abschließend zu bewerten. Die bessere Darstellung der Sollwerte sowie die Anzeige von nicht gelaufenen Programmen gehört zu den Dingen, die das verbessern könnten. Weitere Datenquellen würden umfassendere und detailliertere Analysen erlauben.

Trotz der genannten Einschränkungen wurde das Tool sehr positiv aufgenommen. Ein großer Teil des bisher manuellen Aufwands wird überflüssig, durch die Darstellung der Prozesskette und der einfachen Möglichkeit, das Logfile zu betrachten. Zusätzliche Hilfestellungen wie die Histogramme ermöglichen eine schnelle Einschätzung der Situation.

Die Anwendung erlaubt es, viele Fragestellungen rund um Stored Procedures detailliert zu analysieren. Durch die visuelle Darstellung und die intuitive Bedienung kann der Datensatz schnell erkundet werden. Problemstellen werden deutlich markiert und mit detaillierten Informationen versehen.

A. Lösungen für die Aufgaben der Evaluation

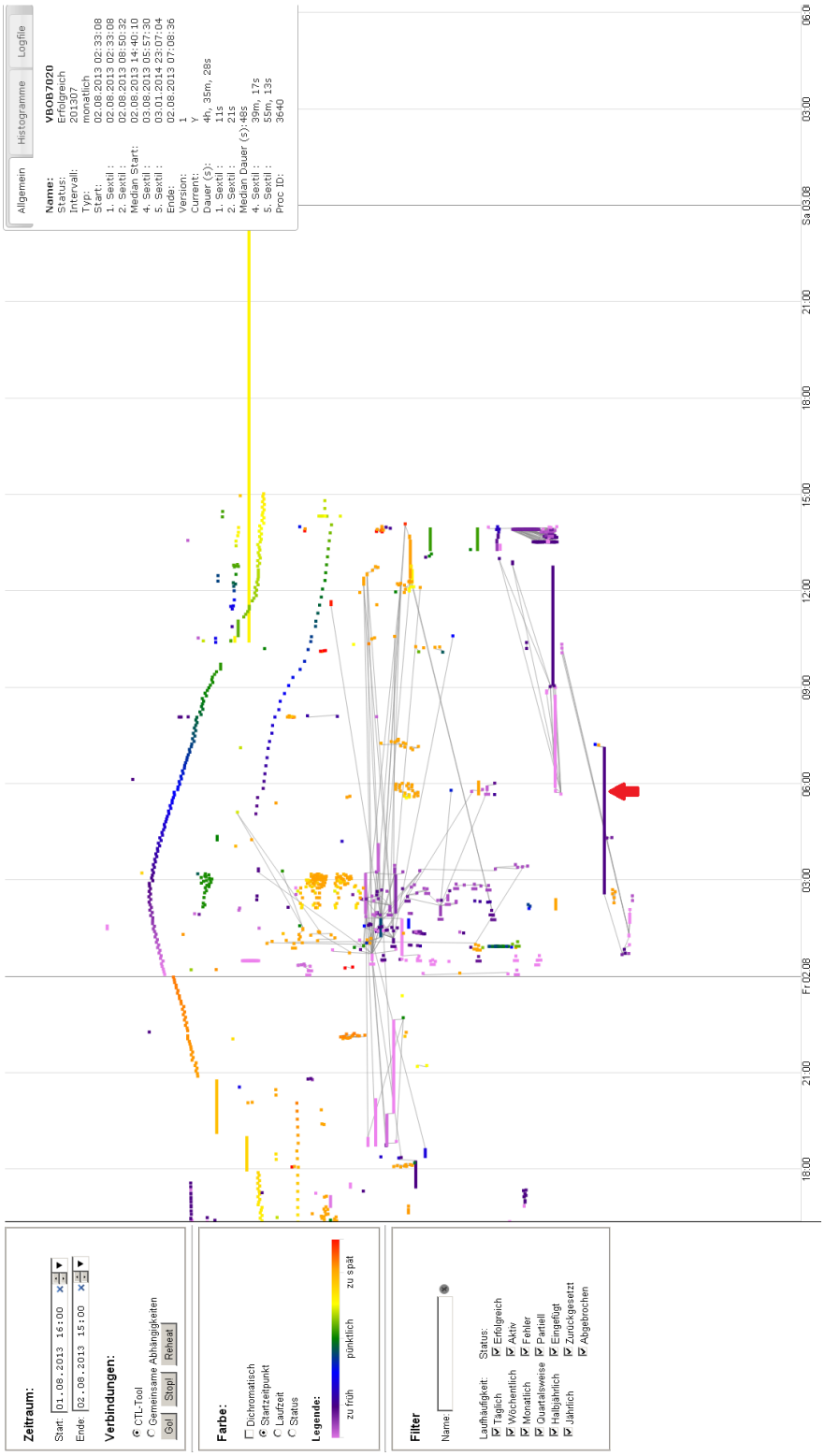


Abbildung A.1.: Lösung für Aufgabe 1.

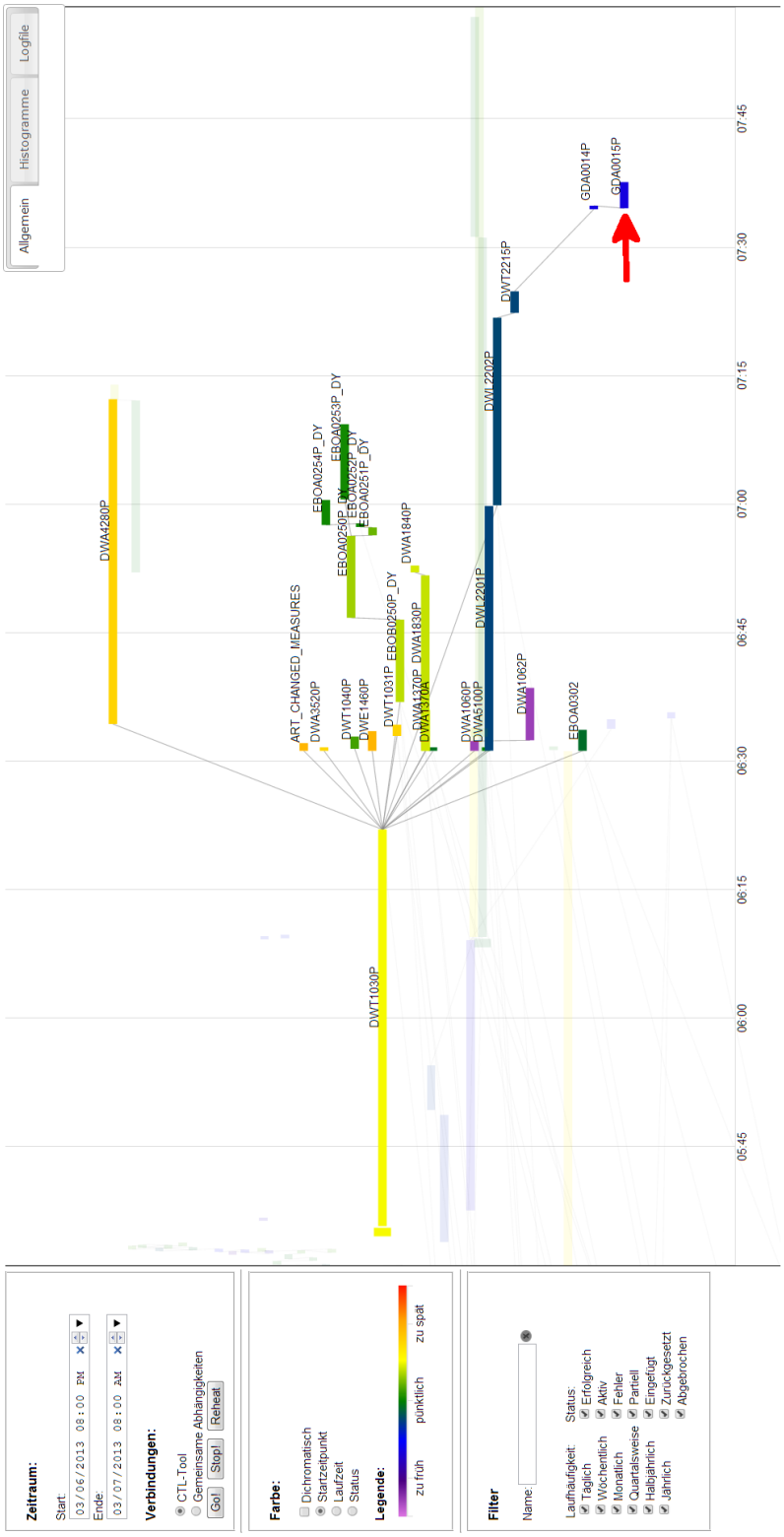


Abbildung A.2.: Lösung für Aufgabe 2.

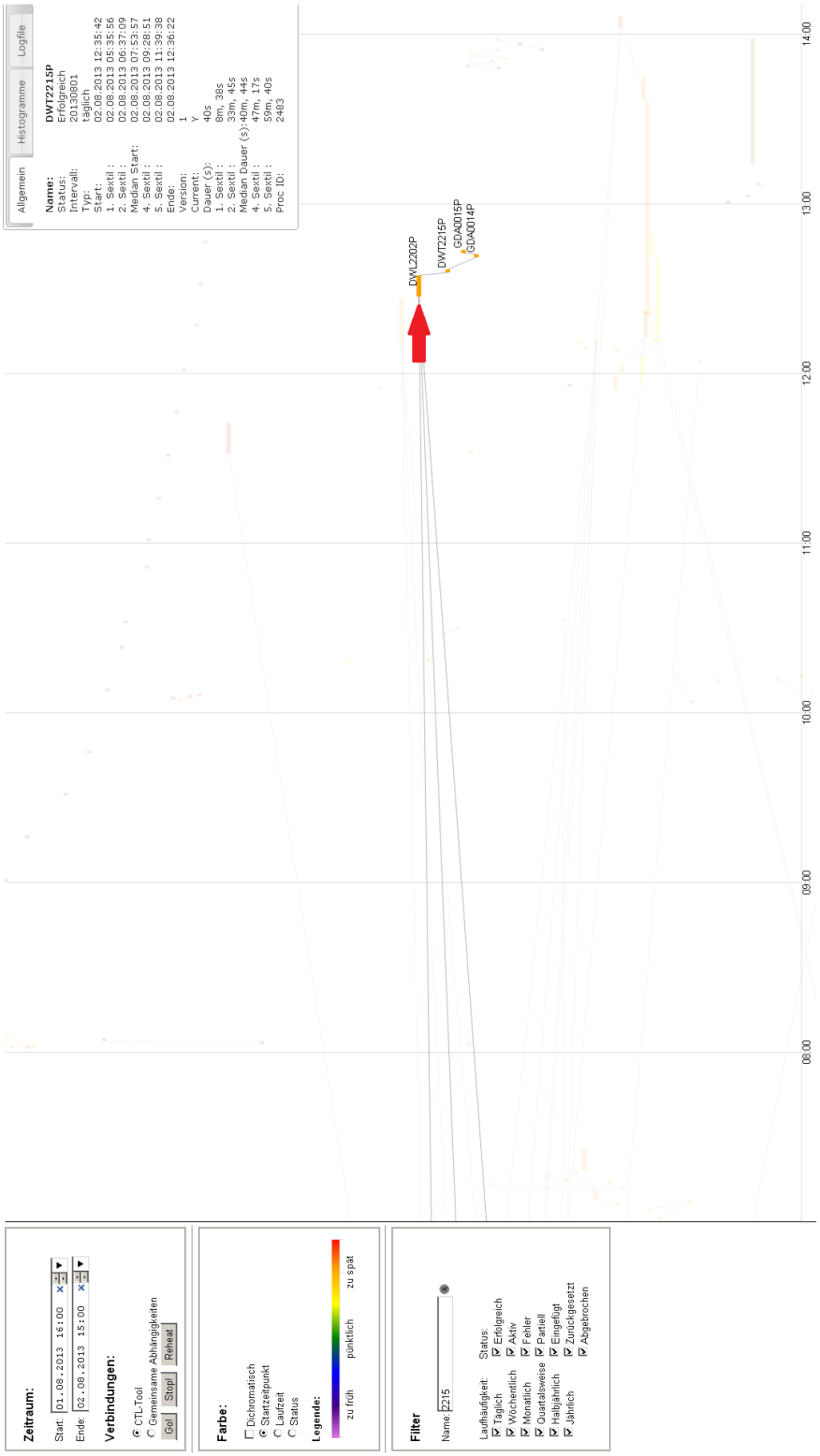


Abbildung A.3.: Lösung für Aufgabe 3.

Literaturverzeichnis

[ADH+03] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237 – 267, 2003. doi:http://dx.doi.org/10.1016/S0169-023X(03)00066-1. URL http://www.sciencedirect.com/science/article/pii/S0169023X03000661. (Zitiert auf Seite 9)

[Bal99] T. Ball. The Concept of Dynamic Analysis. In O. Nierstrasz, M. Lemoine, Herausgeber, *Software Engineering, ESEC/FSE '99*, Band 1687 von *Lecture Notes in Computer Science*, S. 216–234. Springer Berlin Heidelberg, 1999. doi:10.1007/3-540-48166-4_14. URL http://dx.doi.org/10.1007/3-540-48166-4_14. (Zitiert auf Seite 18)

[BDPS94] M. Brown, J. Domingue, B. Price, J. Stasko. Software visualization: a CHI '94 workshop. *SIGCHI Bull.*, 26(4):32–35, 1994. doi:10.1145/191642.191651. URL http://doi.acm.org/10.1145/191642.191651. (Zitiert auf Seite 19)

[BOH11] M. Bostock, V. Ogievetsky, J. Heer. D3; Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi:10.1109/TVCG.2011.185. (Zitiert auf den Seiten 33 und 42)

[Bos13] M. Bostock. d3js.org API-Reference, 2013. URL https://github.com/mbostock/d3/wiki/API-Reference/688db96635667b9d326517443ab2e641fe809e8b. Bei commit 688db96. (Zitiert auf den Seiten 33, 34, 40 und 45)

[CCR02] J. Chen, S. Chen, E. A. Rundensteiner. A Transactional Model for Data Warehouse Maintenance. In *Proceedings of International Conference on Conceptual Modeling, ER'02*, S. 247–262. 2002. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.1076. (Zitiert auf Seite 12)

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970. doi:10.1145/362384.362685. URL http://doi.acm.org/10.1145/362384.362685. (Zitiert auf Seite 11)

[CZD+09] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke. A Systematic Survey of Program Comprehension through Dynamic Analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009. doi:10.1109/TSE.2009.28. (Zitiert auf den Seiten 19 und 24)

[DK76] F. DeRemer, H. Kron. Programming-in-the-Large Versus Programming-in-the-Small. *IEEE Transactions on Software Engineering*, 2(2):80–86, 1976. doi:10.1109/TSE.1976.233534. (Zitiert auf Seite 19)

[DPJM+02] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. M. Vlissides, J. Yang. Visualizing the Execution of Java Programs. In *International Seminar, Revised Lectures on Software Visualization*, S. 151–162. Springer-Verlag, London, UK, UK, 2002. URL http://dl.acm.org/citation.cfm?id=647382.724791. (Zitiert auf den Seiten 6, 19 und 20)

[DPLP+05] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, J. F. Morar. Web services navigator: visualizing the execution of web services. *IBM Syst. J.*, 44(4):821–845, 2005. doi:10.1147/sj.444.0821. URL http://dx.doi.org/10.1147/sj.444.0821. (Zitiert auf den Seiten 6, 19, 20 und 21)

[Eis96] A. Eisenberg. New standard for stored procedures in SQL. *SIGMOD Rec.*, 25(4):81–88, 1996. doi:10.1145/245882.245907. URL http://doi.acm.org/10.1145/245882.245907. (Zitiert auf Seite 12)

[FR91] T. M. J. Fruchterman, E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102. URL http://dx.doi.org/10.1002/spe.4380211102. (Zitiert auf Seite 33)

[HSMM00] I. Herman, I. C. Society, G. Melancon, M. S. Marshall. Graph Visualization and Navigation in Information Visualization: a Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000. (Zitiert auf Seite 17)

[JLVV02] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis. *Fundamentals of Data Warehouses*. Springer, 2nd Auflage, 2002. (Zitiert auf Seite 9)

[JSB97] D. F. Jerding, J. T. Stasko, T. Ball. Visualizing interactions in program executions. In *Proceedings of the 19th international conference on Software engineering*, ICSE '97, S. 360–370. ACM, New York, NY, USA, 1997. doi:10.1145/253228.253356. URL http://doi.acm.org/10.1145/253228.253356. (Zitiert auf Seite 19)

[KG06] A. Kuhn, O. Greevy. Exploiting the Analogy Between Traces and Signal Processing. In *22nd IEEE International Conference on Software Maintenance, 2006, ICSM '06.*, S. 320–329. 2006. doi:10.1109/ICSM.2006.29. (Zitiert auf den Seiten 6, 19, 23 und 24)

[KK89] T. Kamada, S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7 – 15, 1989. doi:10.1016/0020-0190(89)90102-6. URL http://www.sciencedirect.com/science/article/pii/0020019089901026. (Zitiert auf Seite 33)

[Mun09] T. Munzner. Visualization. In P. Shirley, S. Marschner, Herausgeber, *Fundamentals of Computer Graphics*, Kapitel 27, S. 675–707. AK Peters, 3 Auflage, 2009. (Zitiert auf Seite 10)

[Mye86] B. A. Myers. Visual programming, programming by example, and program visualization: a taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, S. 59–66. ACM, New York, NY, USA, 1986. doi:10.1145/22627.22349. URL http://doi.acm.org/10.1145/22627.22349. (Zitiert auf Seite 18)

[Ora09a] Oracle. Oracle Documentation - ALL_DEPENDENCIES, 2009. URL http://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_1066.htm. (Zitiert auf Seite 29)

[Ora09b] Oracle. Oracle Documentation - Using EXPLAIN PLAN, 2009. URL http://docs.oracle.com/cd/B28359_01/server.111/b28274/ex_plan.htm. (Zitiert auf Seite 18)

[RBRB06] S. Rinderle, R. Bobrik, M. Reichert, T. Bauer. Businesss Process Visualization - Use Cases, Challenges, Solutions. In *Proceedings of 8th Internationall Conference on Enterprise Information Systems, ICEIS'06. Track on Information Systems Analysis and Specification*, S. 204–211. 2006. URL http://dbis.eprints.uni-ulm.de/112/. (Zitiert auf Seite 21)

[RKZ00] E. A. Rundensteiner, A. Koeller, X. Zhang. Maintaining Data Warehouses over Changing Information Sources. *Commun. ACM*, 43(6):57–62, 2000. doi:10.1145/336460.336475. URL http://doi.acm.org/10.1145/336460.336475. (Zitiert auf Seite 13)

[RLN07] R. Rosenholtz, Y. Li, L. Nakano. Measuring visual clutter. *Journal of Vision*, 7(2), 2007. doi:10.1167/7.2.17. URL http://www.journalofvision.org/content/7/2/17.abstract. (Zitiert auf Seite 17)

[SRB+05] J. Stott, P. Rodgers, R. Burkhard, M. Meier, M. Smis. Automatic layout of project plans using a metro map metaphor. In *Proceedings of Ninth International Conference on Information Visualisation, 2005.*, S. 203–206. 2005. doi:10.1109/IV.2005.26. (Zitiert auf den Seiten 6, 22 und 23)

[SS07] S. Sumathi, S.Esakkirajan. *Fundamentals of Relational Database Management Systems*. Studies in Computational Intelligence. Springer, 2007. (Zitiert auf den Seiten 11 und 50)

[ST02] J. Six, I. Tollis. Automated Visualization of Process Diagrams. In P. Mutzel, M. Jünger, S. Leipert, Herausgeber, *Graph Drawing*, Band 2265 von *Lecture Notes in Computer Science*, S. 45–59. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45848-4_4. URL http://dx.doi.org/10.1007/3-540-45848-4_4. (Zitiert auf Seite 21)

[TSFH+13] M. Tory, S. Staub-French, D. Huang, Y.-L. Chang, C. Swindells, R. Pottinger. Comparative visualization of construction schedules. *Automation in Construction*, 29(0):68 – 82, 2013. doi:http://dx.doi.org/10.1016/j.autcon.2012.08.004. URL http://www.sciencedirect.com/science/article/pii/S0926580512001458. (Zitiert auf den Seiten 6, 21 und 22)

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the Fourth International Conference on Information and Knowledge Management*, CIKM '95, S. 25–30. ACM, New York, NY, USA, 1995. doi:10.1145/221270.221319. URL http://doi.acm.org/10.1145/221270.221319. (Zitiert auf Seite 13)

Alle URLs wurden zuletzt am 17. 11. 2013 geprüft.

Danksagung

Mein Dank gilt..

Herrn Prof. Dr. Ertl, für die Möglichkeit, diese Diplomarbeit am Institut für Visualisierung und Interaktive Systeme durchzuführen.

Steffen Lohman M. Sc. und Dipl. Inf. Fabian Beck, für die Betreuung und Motivation und viele hochinteressante Gespräche.

Meinem Arbeitgeber und ganz besonders meinen Chefs, für die Bereitstellung der Daten und der Ermöglichung der Evaluation.

Meinen Kollegen, für die Teilnahme an der Evaluation und das sehr hilfreiche Feedback.

Meiner Mitbewohnerin Sarah Fieß, für die großartige Unterstützung, Motivation und unsere Frühstücke.

Meiner guten Freundin Dorothea Wahl, für die riesige Unterstützung, für die Motivation und für die wunderbaren Urlaube.

All meinen Freunden, meiner Familie und ganz besonders meinen Eltern Ruth und Edmund Meyer, für die Unterstützung, die Motivation und die vielen kleinen Dinge, die man viel zu leicht vergisst.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift