Master's Thesis

Smart assistive script breakdown and its integration into business processes on the example of film production

Karl Heinz Struggl, BSc

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology

Supervisor: Assoc. Prof. Dr. Andreas Holzinger, PhD, MSc, MPh, BEng, CEng, DipEd, MBCS

Graz, December 2010

This page intentionally left blank

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

Intelligente Unterstützung bei der Aufgliederung von Manuskripten und ihre Integration in Geschäftsprozesse am Beispiel der Filmproduktion

Karl Heinz Struggl, BSc

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz

Betreuer: Univ.-Doz. Ing. Mag. Mag. Dr. Andreas Holzinger

Graz, December 2010

Abstract

The process behind the production of a motion picture, be it a documentary, an action movie or an animation film, consists of a number of cascaded phases and steps. As such, it can be viewed upon from various angles, each with its own characteristics and important aspects, like the perspectives of the creative minds behind the production, or from the position of a financially accountable manager. One important task for the latter group, especially at the beginning of the planning and budgeting phase, is breaking down all elements and aspects of the script and creating a solid estimate of production costs.

Established industry standards pertain to this phase, and the first goal of the work at hand is analyzing how the inherent processes can be improved and accelerated with the assistance of expert applications, as well as finding out whether, and to what extent, existing software solutions provide such help. As a result of these findings, this work will then suggest an improved solution reflecting these observations, as well as present a working prototype as proof of concept, that will be developed with a strong focus on Usability Engineering.

Keywords
film production, script breakdown, nlp, software architecture, usability study

ÖSTAT classification
1108 40%, 1109 30%, 1157 15%, 1161 15%

ACM classification
D.2.11, D.3.3, H.1.2, H.5.2, I.2.7, K.4.3

This page intentionally left blank

Kurzfassung

Der Prozess hinter einer Filmproduktion, sei es eine Dokumentation, ein Actionfilm oder eine Animation, besteht aus einer Vielzahl verschachtelter Phasen und Schritte. Dabei entstehen verschiedene Blickwinkel auf das Projekt, mit jeweils unterschiedlichen Charakteristiken und Aspekten, wie zum Beispiel die Perspektive der kreativen Köpfe, oder die Sicht der für die Finanzierung verantwortlichen Manager. Zu deren Hauptaufgaben, vor allem in den frühen Planungs- und Budgetierungsphasen, gehört das Herunterbrechen der verschiedenen Elemente des Manuskripts. Dieser Schritt ist essentiell für eine erste, aussagekräftige Kostenabschätzung und damit entscheidend für den möglichen weiteren Verlauf der Produktion.

In dieser Phase kommen etablierte Industriestandards und Normen zum Tragen. Das erste Ziel dieser Arbeit ist es dabei, die darin enthaltenen Prozesse und Vorgehensweisen zu analysieren und Möglichkeiten zu finden, diese mittels Expertenlösungen zu verbessern und beschleunigen, sowie aufzuklären, inwieweit dies bereits von bestehenden Branchenlösungen umgesetzt wird. Als Ergebnis dieser Untersuchungen wird schließlich eine verbesserte Lösung präsentiert, deren Konzept und Umsetzbarkeit anhand eines Prototypen gezeigt werden. Bei der Entwicklung dieses Prototypen außerdem wird ein starker Fokus auf Usability gelegt werden.

Schlüsselwörter

filmproduktion, script breakdown, nlp, softwarearchitektur, usability studie

ÖSTAT Klassifikation

1108 40%, 1109 30%, 1157 15%, 1161 15%

ACM Klassifikation

D.2.11, D.3.3, H.1.2, H.5.2, I.2.7, K.4.3

This page intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, December 3rd, 2010

Karl Heinz Struggl

This page intentionally left blank

Acknowledgements

First and foremost I would like to express my sincerest gratitude to my family and close friends for their patience and their seemingly never-ending support towards me, my work and my studies.

I feel deeply indebted to my advisor, Andreas Holzinger, who imparted inestimable knowledge and insights onto me over the course of the last few years, for which I am truly grateful.

I would also like to thank my fellow students and colleagues, especially Martin, Martin and Martin of Nimblo, for many inspiring discussions and both their amicability and professionalism.

Of the many as yet unmentioned people who certainly endorsed this work in various other ways, I would like to thank Helen Ashton for revising the written thesis, as well as everyone who helped with conducting the usability study and all who participated in it.

Karl Heinz Struggl

Graz, December 2010

This page intentionally left blank

Table of Contents

1. Introduction and Motivation for Research

The implementation of audio-visual (AV) projects has become an increasingly software-assisted process over the last few years and decades (Singleton, 1996, 1991). While there is an astonishingly wide range of types and genres, the majority of these productions - from film students' graduation pieces to extensively produced hollywood blockbusters - follow some specific schemes and rules.

Among these conventions, there are ideas and patterns that have proven suitable in dealing with certain circumstances of such projects. One such convention of particular importance deals with the need to find ways of enabling creative minds (such as authors of screenplays, directors, costume designers) and people carrying out positions of financial (among others) responsibilities to communicate.

In order to establish such an interface, one central point of communication, as well as documentation, has been identified: the script (Clevé, 2005). It not only describes all scenes including their locations, but also hints at how specific roles should be cast, what special effects may be needed and much more. This data, however, first needs to be filtered, structured, amended and supplemented by various project members with different amounts of experience and insights. The resulting set of mostly formal information represents the very foundation of the project and is used as a basis for carrying out scheduling and budgeting tasks.

In this fundamentally important step, which is called the *Script Breakdown*, the script is read through and elements of specific importance and relevance to certain categories are marked accordingly, for example all speaking roles may be highlighted in red, make-up/hair elements are marked with asterisks and so on. The result is a structured set of data defining exactly which kinds of elements each scene is comprised of. In another step following this breaking down of elements, additional information is gathered and estimations about the required number of shooting days and overall costs are given (Clevé, 2005).

As hinted before, this task is usually re-iterated several times by various team members who are experts in differing fields of production. What is more, as the project progresses in production, there are inevitably frequent changes in various parts of the script and therefore also in the scenes. This often causes a need to repeatedly revise the breakdown in order to reflect said changes, which furthermore brings along the necessity to update all derived documents as well, for instance to re-assess, print and distribute breakdown sheets.

It is obvious at this point that several aspects of this process are well suited for automation and assistance by software, especially considering large scripts with several dozens of scenes containing recurring elements. In practice, however, script breakdown is still most often carried out using highlighters on printouts of the scripts and later transferring the data manually into sometimes decades-old software with numerous deficiencies ranging from lacking functionality to inadequate usability.

This thesis presents an approach aiming to assist professionals in a number of tasks pertaining to the breaking down of scripts and using the gathered data to rapidly communicate information among various groups of specialists in the project team, as well as create estimations on production time and costs. In addition to the theoretical work described herein, a prototype application will be implemented and presented as well. It will act as a proof of concept to demonstrate the practicality and usability of the devised ideas.

The first part of this thesis will establish a fundamental understanding of the theoretical background (Chapter 2) and provide an overview of works related to the topics presented thereby (Chapter 3). An observation of the methods and systems currently employed (Chapter 4) concludes these first three chapters.

A presentation of concepts and ideas on how to improve the process of script breakdown will then follow (Chapter 2.3) and lead into the practical installation thereof. A detailed report on the implementation and testing of the prototype application as well as the usability study and its evaluation (Chapters 6.1 and 6.2 respectively) complement the second part.

Finally, a discussion of the findings of this thesis (Chapter 7), the lessons learned from designing, implementing and evaluating the presented prototype (Chapter 8) as well as an outlook into possible future work (Chapter 9) will close this work.

2. Theoretical Background

In order to better understand some special requirements and intricate details about several aspects discussed in later chapters, this section will provide a brief introduction of the film production process, starting with one of its most important staff members, the production manager.

The four phases of a film production project, as well as the task of script breakdown and its importance for the rest of production will then be discussed in more detail, whereas film scheduling and budgeting will be summed up briefly as the later chapters will provide required detailed information as necessary.

A discussion of the economical value of a smart script breakdown application, its potential for process improvement and an assessment of the associated risks will then be presented in the form of a business case.

2.1 The production manager

According to Clevé (2005), the role of a production manager (PM) is one of the most essential appointments in the production of a film, because the PM is one of the few members of the whole production staff that is involved in most (if not all) production phases. As such, the PM is not only an up-to-date, central point of information about the most various aspects of the production for the other staff members, but also a mediator and balancing factor in human conflicts.

Depending on the size of production, the exact areas of responsibility and authority, as well as the distinction of job titles (PM, unit production manager or UPM, line producer, producer), can become imprecise and overlap, but the general idea of the PM is as pointed out. The Directors Guild of America (DGA) provides a formal job description for the role of unit production manager, stating that the

UPM:

> *"[. . .] is required to coordinate, facilitate and oversee the preparation of the production unit or units [. . .] assigned to him or her, all off-set logistics, day-to-day production decisions, locations, budget schedules and personnel."* (Directors Guild of America, Inc, 2005)

It also clearly points out that supervision and participation in breakdown, preliminary shooting schedule and preparation and coordination of the budget are among their main duties. These latter points will be discussed in the next sections, after the differing features and aspects of the four phases of film production have been established.

For a better understanding of the organizational structure in film production and how the PM is embedded in the role's hierarchical and functional context, see figure 2.1.

2.2 The four phases of film production

As previously noted, the film production process is segmented into four phases that are usually traversed consecutively with the possibility of small overlaps, and differ in their staff requirements and outcomes. The four phases, as characterized by Clevé (2005), are:

1. **Development**
   In this phase, the producer conceives ideas for movies, considering a number of possible sources like novels, real-life stories and screenplays or movies which already exist. The producer needs to clear required rights and permits for use of the screenplay or the screenplay adaption of the source before proceeding to find attractive *Name* talents, such as actors, directors and similar. This package will then be presented to possible production companies and studios for further negotiation on a production.

   At this point, a PM can be assigned by the producer to conduct a first breakdown of the screenplay and create preliminary cost estimations in the form of rudimentary budgets.

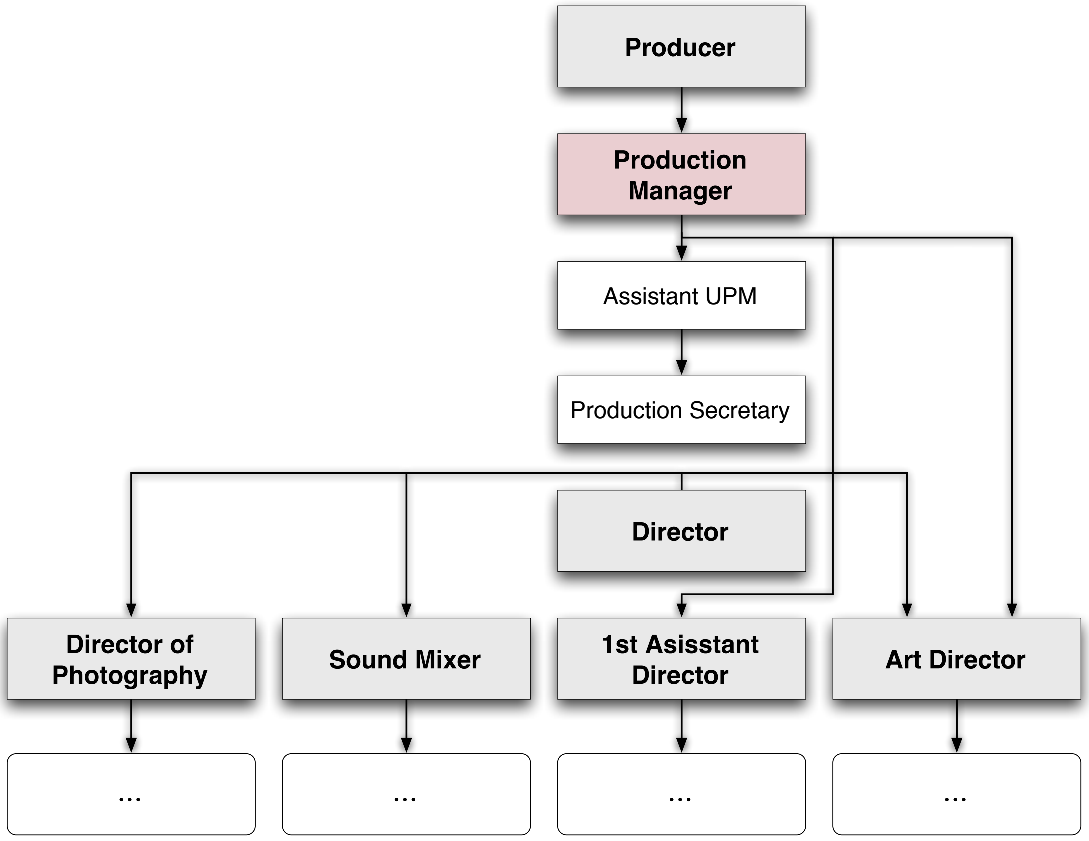


Figure 2.1: Organizational chart (excerpt) of a motion picture production team in the preproduction and production phases (adapted from (Clevé, 2005)).

2. **Preproduction**

   Once the project has been green-lighted, a number of tasks in preparation of the actual production have to be carried out, namely a more thorough breakdown of the screenplay and its elements, as well as the construction of reliable shooting schedules and an obligatory budget. This requires organizational work carried out by the PM in a number of areas, including location scouting, casting, hiring of staff and crew, equipment rental, insurance and much more.

   This phase, especially the breakdown of the screenplay along with providing mechanisms for enabling efficient integration into scheduling and budgeting, will be the main focus of all discussions in the sections following this chapter.

3. **Production**

   This phase, also known as principal photography, represents the actual production of the movie. The responsibilities of the PM herein are the organization and coordination of all staff members, actors, equipment and other elements to guarantee efficient and glitch-free shooting.

As depicted in 2.1, the PM communicates mostly with the assistant directors (AD) in order to supervise production units. How exactly shooting on set is carried out depends strongly on the kind of motion picture and the scene itself.

4. **Postproduction**

   Once production has been completed, the produced material is edited and extended with additional material at this stage. This includes the classical editing of the film, sound mastering, creation of visual effects and computer generated (CG) content.

   Due to the fact that a number of these tasks have to be considered beforehand in shooting, the PM will already have made preparations for many of these postproduction elements in the production and preproduction phases.

The organizational chart in 2.1 focuses on the preproduction and production phases. In development and postproduction, some chains of responsibility change and less people are usually involved throughout these phases, resulting in variations of the organizational charts, as detailled in (Clevé, 2005).

2.3 Script breakdown

One of the first tasks for a PM upon entering a film production is to read through the screenplay script thoroughly and start to analyze it. This allows for an estimation of the production's size and scope. Following this, the PM conducts a very in-depth analysis of the script by employing the established industry standard of script breakdown (Singleton, 1996).

Clevé (2005) points out that thorough and correct script breakdown and breakdown sheet generation is one of the single most important factors of the whole film production, as the provided information is used (and is relied upon) in many other tasks within the production process. This includes preproduction steps such as the writing of the shooting schedule and therefore its sub-tasks like scheduling staff availability times, trips, equipment leasing, and thus also the preliminary and planning budgets of the production.

What is more, it also greatly affects the production itself, not only because flawed schedules might cause missed delivery dates and budgets could be wrongly estimated, but also because elements missing in the breakdown sheets will also very likely be missing on the shooting location. This affects a vast number of other formal and informal aspects like shooting efficiency, emotional stress and even the resulting film's quality.

The process of script breakdown itself can be broken down into the following steps (Singleton, 1991; Clevé, 2005):

1. All scenes are identified by their scene headings or sluglines, i.e., lines stating name, location, whether shooting is interior (INT) or exterior (EXT), as well as the scene's time of day, for which the most common classifiers are DAY and NIGHT (Clevé, 2005; Singleton, 1991).

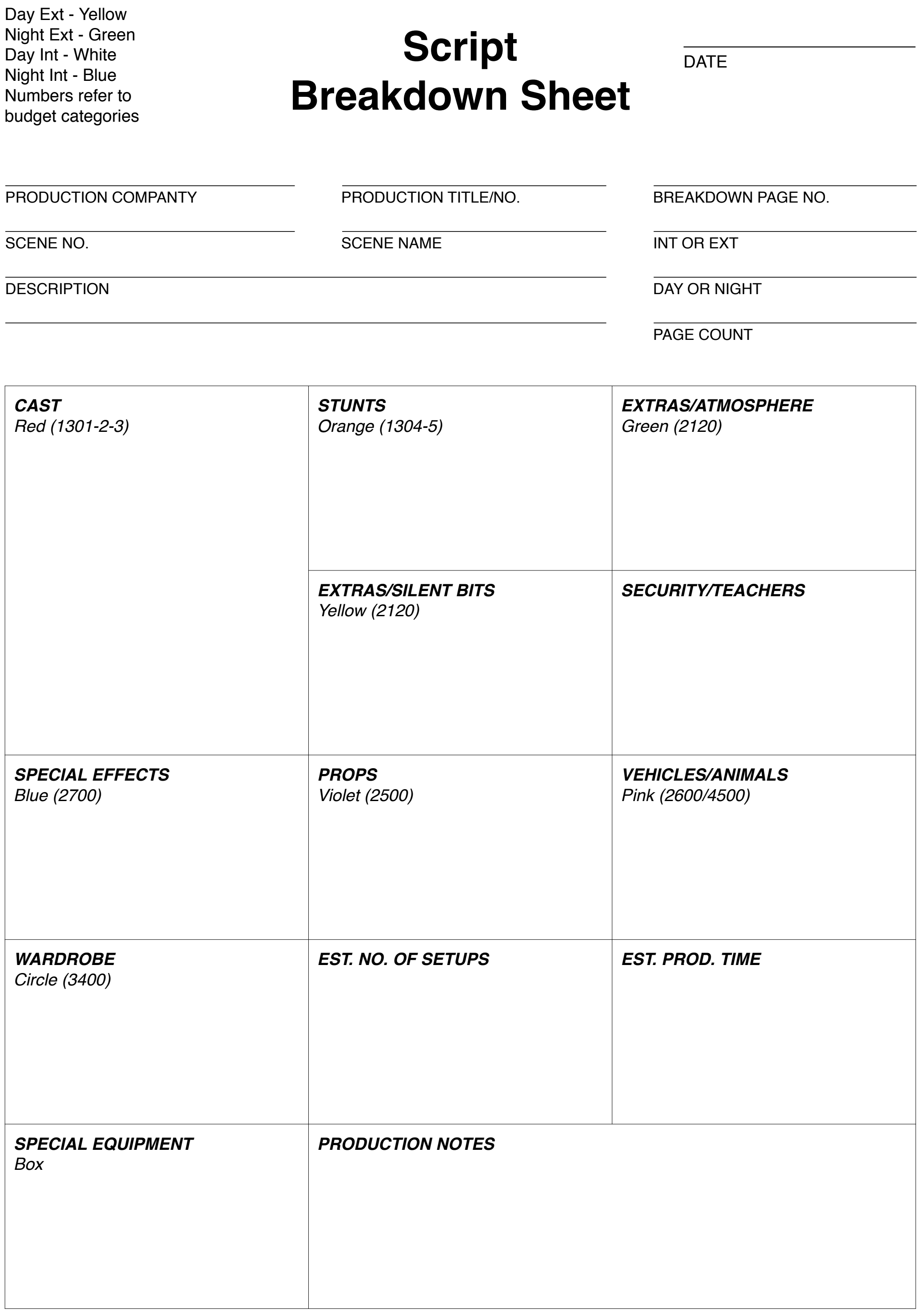
2. Found scenes are then numbered consecutively and measured by their length. The length indicator used is the relative height of the scene on one page, expressed by so-called *1/8s* which equal approximately 1 inch of height on the page. Hence, a scene with a length of 4/8 is about half a page or 4 inches.

3. For every scene, all relevant elements (such as cast, props and equipment) are highlighted, usually in the script, using different colors and other ways of formatting. Additionally, any scene and shooting information (including dates, 1/8s, day/night, interior/exterior shooting) is noted.

4. The information gathered this way is then reviewed and, if necessary, complemented, for instance speaking roles are ordered by the number of scenes they appear in, teachers for minors may be added and other production details might be reconsidered and noted.

5. One breakdown sheet per scene is produced, listing all the information obtained in the previous steps. Usually, a predefined form or template is used for breakdown sheets, to ensure uniformity and completeness. An example for a typical breakdown sheet template is shown in figure 2.2.

   The PM often creates additional breakdown sheets for aid in his own organizational tasks, for example a breakdown of all scenes which a certain actor is in or in a given shooting location.



Day Ext - Yellow
Night Ext - Green
Day Int - White
Night Int - Blue
Numbers refer to
budget categories

Script Breakdown Sheet

DATE

PRODUCTION COMPANY

PRODUCTION TITLE/NO.

BREAKDOWN PAGE NO.

SCENE NO.

SCENE NAME

INT OR EXT

DESCRIPTION

DAY OR NIGHT

PAGE COUNT

CAST *Red (1301-2-3)*	***STUNTS*** *Orange (1304-5)*	***EXTRAS/ATMOSPHERE*** *Green (2120)*
	EXTRAS/SILENT BITS *Yellow (2120)*	***SECURITY/TEACHERS***
SPECIAL EFFECTS *Blue (2700)*	***PROPS*** *Violet (2500)*	***VEHICLES/ANIMALS*** *Pink (2600/4500)*
WARDROBE *Circle (3400)*	***EST. NO. OF SETUPS***	***EST. PROD. TIME***
SPECIAL EQUIPMENT *Box*	***PRODUCTION NOTES***	

Figure 2.2: Template of a typical script breakdown sheet, showing elements for a scene grouped by categories (adapted from (Clevé, 2005)).

Singleton (1991) and Clevé (2005) provide comprehensive discussion on the most commonly used element categories, their special characteristics and possible legal and other requirements that are relevant in breakdown and scheduling.

2.4 Scheduling

In the film production process, scheduling produces two of the most important documents concerning the actual shooting and, later on, the budgeting of the film: the production strip board and the eventual shooting schedule.

2.4.1 Production Strip Board

Singleton (1991) notes that in film history, production strip boards were embodied using multitudes of paper or cardboard cutouts, each representing its corresponding scene's breakdown sheet. Since shooting costs greatly depend on a variety of scheduling factors (including travel costs, holding days for actors and the number of setups needed for shooting locations), these so-called production strips were then arranged on the production strip board and continuously regrouped and reordered so as to find more cost-efficient shooting schedules.

In recent years, an increasing number of software applications providing scheduling features include dynamic and interactive creation of production strip boards. See sections 3.1 and 4.1 for more information on scheduling software for film production.

2.4.2 Shooting Schedule

According to Singleton (1991), the shooting schedule represents the most current version of the production strip board in a more concise and structured form. It is usually printed out for all staff members for reference and communication purposes.

As with the production strip board, a shooting scheduling feature is a main component of every modern film scheduling software.

2.5 Budgeting

Once an initial shooting schedule has been produced and agreed upon by the responsible personnel, a planning budget is created by the production manager. Cost estimations required prior to having a set shooting schedule and thus relying on incomplete data, as is needed in development and preproduction, are not budgets in this sense. Therefore, until all details and contractual variables are set in stone, this budget is considered to be preliminary.

Film production budgets, like their counterparts in other business areas, are complex and demand high degrees of knowledge, research and experience. The budget must cover all factors attributing to production costs as well as considering any ways of covering and minimizing them.

Therefore, a film production budget usually also accounts for fringe costs for personnel, buyout costs for the intellectual rights of creative staff, insurance of equipment, grants given by public and private institutes including possible requirements for fulfillment as well as numerous other factors. Whether or not some of these are relevant for a production strongly depends on the type of production. Literature on the topic covers these factors in great detail; see, for example (Singleton, 1996; Clevé, 2005).

Similar to other areas of economics, a film budget makes use of an account system and budget forms, grouping together similar and related cost positions and assigning them numbers and titles. Due to differing demands between different types of productions, production countries and their laws, among others, there is no fully standardized account system, but there are some established guidelines and common rules. In recent years, some film budgeting software applications (see section 4.1) have garnered enough popularity in certain regions and areas to be able to influence how accounts are set up.

2.5.1 The Planning Budget

In order to set up a budget, the production manager accounts for all elements pointed out in the breakdown data (after it has been consolidated with unit production managers, creatives and other involved personnel), creating all necessary calculation positions for them and assigning estimated costs (allowances). To assist in the process of finding prices, the PM can make use of price lists and so-called rate books, provided by institutes supplying equipment, staff and services to film production companies (for example camera and film manufactures, actor guilds and other unions).

2.5.2 The Working Budget

As the production transcends from preproduction to the actual shooting, the planning budget is also transitioned into the working budget. Over the course of production, all incurred costs are considered and constantly evaluated in terms of allowed costs with fixed periods for intermediate settlement of accounts (e.g. per month) and often using additional means for analyzing the current financial situation (such as cash flow charts). Once all effective costs have been determined, the budget is finalized and signed off by all responsible staff, which at least includes the PM and the producer.

3. Related Work

This chapter will first establish a simple taxonomy of software solutions in the area of film production, which will serve as a means of classification of evaluated applications in chapter 4. An overview of text recognition techniques for natural language will then be provided in order to support scientific and technical decisions and implementation details of the prototype, as will be discussed in chapter 6.1. A complementary observation of relevant literature on usability engineering will also be conducted, with consideration of remote and asynchronous methods.

3.1 Classification of software solutions in film production

It has been established that the production of a motion picture is a complex, cascaded process with a number of different points of view and demands on both the project members and the associated software applications. It seems, therefore, advisable to exploit this inherent quality of the observed software products, which is the affiliation with certain aspects of film production.

Following this principle, the discussion and evaluation of existing software solutions in the next sections will categorize the samples (i.e., applications) by their belonging to one or more of the following groups:

Screenwriting software is mainly used by authors for writing and editing scripts for different genres of motion picture. Frequently provided features include the creation of certain printouts for genre-specific kinds of script and summary sheets. Applications in this group may support exporting of data for scheduling software and sometimes provide rudimentary tools for conducting script breakdown.

Scheduling software provides both general and film-proprietary project management functions, usually including interfaces for planning the shooting sequences for scenes, depending on various factors like the location and the required availability of staff (for instance the main actor). Therefore, scheduling applications benefit strongly from being able to directly read such script-related breakdown information and sometimes provide the means to import and breakdown script data.

Budgeting software represents the accounting side of film projects. It may incorporate modules for planning the funding of a project, the calculation of both budgeted and incurred costs, as well as other more in-detail features such as, for example, a human resources module. Budgeting software is thus typically required to incorporate information directly from script breakdown, as well as the scheduling software and possibly external sources, such as catalogs containing staff wages and equipment rates.

This list is not exhaustive. Script authors, for example, may also use standard software word processors for writing, similar to the way in which certain PDF applications may be used to mark elements within scripts. For the context of this work, however, these are exceptions and are to be considered as influenced by, rather than affecting, the status quo of professional expert software in this field.

It is also to note that some of the optional, more specialized features mentioned in the list above may be implemented by self-contained applications as well. Still, they usually share the same characteristics as the respective categories they are listed in above and will not be considered separately from here on.

3.2 Model-View-Controller (MVC) in research and business applications

Model-View-Controller (MVC) is a software design pattern which was created by Trygve Reenskaug in 1979 (Reenskaug, 1979a,b) while working for Xerox PARC.

It is considered to be an architectural design pattern and it divides an interactive application into three separate components, as described in Buschmann et al. (1996):

The Model represents the underlying data representation as well as the core functionality of the system. The Model collaborates with View and Controller through a change-propagation mechanism operating on a registry of dependencies, i.e., the Model informs dependent views about its own state changes.

The View is responsible for displaying information to the user. The change-propagation mechanism is meant for keeping the View in sync with the Model. There is usually a 1:1 relationship between View and Controller and a View can provide additional functions to the Controller that are independent of the Model, for example for scrolling lists or similar.

The Controller handles user input as events which can include invoking requests and changes on the Model and/or the associated View. The Controller, similarly to the View, can request to be notified about updates to the Model. View and Controller together form the actual user interface and implement interactivity.

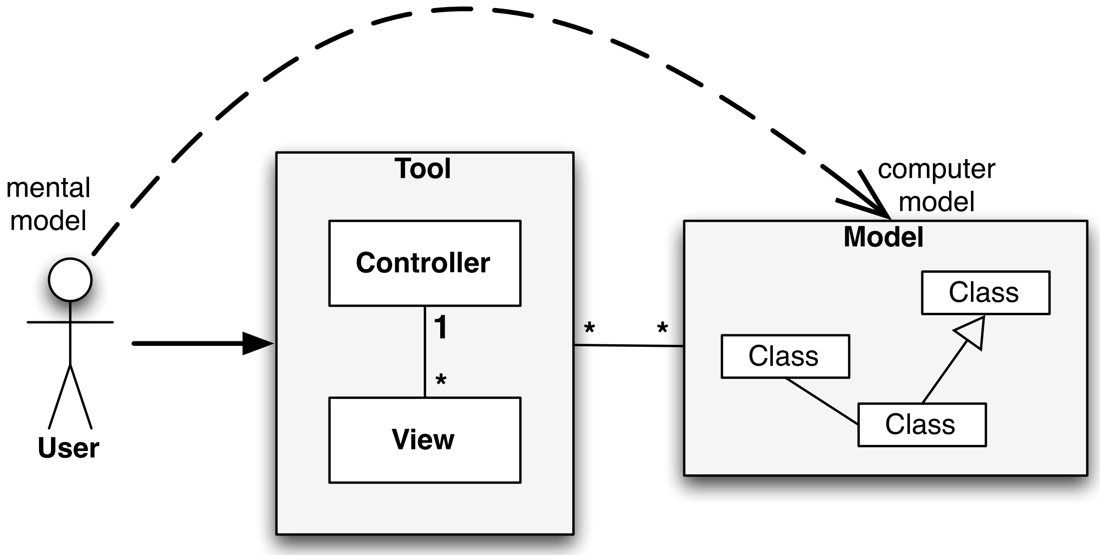


Figure 3.1: Basic design of a MVC-based architecture according to Reenskaug (2009).

MVC has become a widely known and used pattern for varieties of software systems, from interactive graphical user interface (GUI) frameworks like Apple's Cocoa[1] and Cocoa Touch[2] for Mac OSX and iOS Devices (iPhone, iPod Touch, iPad) respectively, to web presentation and interface frameworks like Ruby on Rails[3]. Figure 3.1 shows the basic concept of the MVC architecture, as devised by Reenskaug (2009).

3.2.1 Advantages of MVC

Fowler (2002) states that MVC's most fundamental benefit is the strong separation of the presentation layer from the underlying domain-specific Model. The author also adds that restricting the Model to a collection of nonvisual objects makes it easier to cover the software's functionality with automated tests, thus allowing for efficient unit testing, as discussed by Zhu et al. (1997). The author, in accordance with Buschmann et al. (1996), therefore states that MVC is mostly recommendable in systems where the separation of presentation and model is an important concern, and where nonvisual logic and workflows would otherwise clutter the Model and make it less reusable.

MVC has been adopted in a wide range of application domains, apart from web presentation and GUI applications. For example, the Medical Imaging Interaction Toolkit (MITK, Wolf et al. (2005)) aims at providing frameworks and tools to efficiently develop highly interactive software for medical imaging and visualization. The toolkit was designed with a model in mind that places medical need at the beginning of the development of tools for medical imaging and interaction. The need invokes development of a data model and of algorithms able to fulfill this need, leading to the design and implementation of the actual visualization and interaction. Practical use in clinical processes triggers refinement of the created components, which often results in iterative improvement. In this development model, the data and algorithms relate to MVC's Model, while the visualization and the interaction components match View and Controller respectively. Consequently, the implementation of MITK is strongly concerned with graphical representation of data and user

[1]`http://developer.apple.com/technologies/mac/cocoa.html`, last access 11/2011
[2]`http://developer.apple.com/technologies/ios/cocoa-touch.html`, last access 11/2011
[3]`http://rubyonrails.org/documentation`, last access 11/2011

interaction and was therefore based on MVC.

Another example of how diversely MVC has been applied throughout the last few decades is the patent invented by Flores et al. (1998). It describes a system component called Workflow Application Builder as part of a complex, interactive system made to analyze, design and document business processes and their inherent workflows as well as create workflow-oriented applications based on these definitions. The system architecture is separated into a presentation (GUI) tool set providing graphical tools and a definition part allowing the specification of business processes and their attributes and characteristics. The workflow application builder allows business process designers to specify business processes along with their network of workflows and to automatically or semi-automatically generate an application with graphical interfaces (forms and views) to create, manipulate and ensure persistence of underlying data. Concerning the software design and implementation, the inventors therefore based the system's architecture on MVC. The reasoning behind this decision is based on two main advantages: First, to ensure separation the application logic (i.e., Controller) from the Model and therefore making *"the application more portable, the design more understandable and the implementation extendible"* (Flores et al., 1998). Second, the isolation of the View from the Controller was deemed beneficial for the ability to port the application's back-end to other GUI frameworks.

3.2.2 Disadvantages of MVC and alternatives

MVC is often criticized for being inefficient in system designs where the Model itself is required to implement large amounts of functionality, or where separation between View and Controller is either difficult to implement (for instance due to limitations of underlying frameworks or similar) or unnecessary (Buschmann et al., 1996; Fowler, 2002).

Another of MVC's characteristics that is attracting increasing amounts of criticism is its strong adherence to principles of classical object oriented programming (OOP). This is, as pointed out in numerous recent works such as (Schärli et al., 2003, 2002; Reenskaug and Coplien, 2009; Reenskaug, 2009; Coplien and Bjørnvig, 2010), due to OOP's limited ability to reflect and model system behavior, which results from its strong focus on creating a static model of object classes and their relationships (Booch et al., 2007). In terms of designing a software system in concordance with MVC, this means that business processes and system-global workflows need to be either the responsibility of a Controller, and therefore designed separately from the actual Model, or be a part of the Model objects they pertain to (for example a bank transaction could be part of an `Account` class).

In the first case, a lot of functionality will not be reusable, as Controller implementations are typically very application-dependent. The alternative means that logic will generally be reusable, but at the cost of the Model becoming bigger and increasingly filled with functionality, which will be both harder to maintain and port, and more difficult to extend due to limitations of OOP's static class inheritance, as pointed out by Schärli et al. (2002).

The inheritance problem is a result of the static nature of how OOP is used to model systems - a Model class (and therefore its instances, or objects) either implements a certain functionality (self or through inheritance), or it does not, there is no dynamic model. Combined with well-known disadvantages of single and multi-inheritance, as well as mix-ins, this often results in compromises to the system design incurred by copying and pasting code or introducing deeply nested class hierarchies (Schärli et al., 2003, 2002; Coplien and Bjørnvig, 2010).

An alternative way of reusing functionality in OOP systems is object composition. Gamma et al. (1994) define it as a technique of assembling (or composing) functionality to create more powerful objects. Composition requires interfaces to be designed for the composed objects resulting in dynamically created, concrete objects implementing sets of interfaces. While this requires more effort in creating interfaces, it also results in less dependency between objects, flat hierarchies and better reusability. Gamma et al. (1994) clearly state that object composition should be favored to class inheritance if possible. However, inheritance has become a much more widely used mechanism for class reuse, which is likely attributable to the way OOP captures functionality, as reasoned above.

In recent years, the concept of object composition has repeatedly been adopted in publications proposing new software design architecture and patterns, like Traits (Schärli et al., 2003; Schärli, 2005) and DCI (Data-Context-Interactions or Data-Collaborations-Interactions, Reenskaug and Coplien (2009)).

The traits formal model, as defined by Schärli et al. (2003), defines traits as a means of adding implemented behavior to classes without changing the state or the meaning of the class. Traits define both a set of implemented (provided) behavior and a set of required behavior, allow composition and nesting, and inheritance which, contrary to class inheritance, do not result in trait hierarchies but remain *flattened*. This ability to nest and combine traits is what sets this model apart from aspect-oriented approaches, where specific functionality without clearly defined interfaces is superimposed onto existing classes to reduce the need to duplicate code and ease adding specific parts of behavior to objects (Elrad et al., 2001). A class is thus composed by its own state (i.e., its instance variables), a set of traits the class features and additional code to fulfill the interface requirements of the provided traits, for example by granting access to instance variables (Schärli, 2005).

Native support for traits is highly limited in current programming languages and few implementations of the Traits model have been published as yet[4].

DCI's development is based on the assumption that modeling system state in conjunction with system behavior is beneficial in creating a system design that better reflects the user's mental model of software (Reenskaug and Coplien, 2009).

Similar to traits, DCI introduces the concept of roles as a means of defining sets of object behaviors that are logically coherent. This design was motivated by the observation that object boundaries are seldom congruent with behavioral boundaries, as most typical workflows and business processes involve more than one type of object. Roles are subdivided into methodless roles, also called role interfaces, and methodful roles, which provide generic implementation of such interfaces. An important difference of DCI compared to traits is the inclusion of a central Context object. The Context is responsible for knowing which object classes are able to take on certain roles, and implementing the according object composition by applying such a methodful role to a class object (Coplien and Bjørnvig, 2010).

DCI roles can be implemented by incorporating traits, or by using runtime method injection techniques provided by a number of programming languages and environments.

[4]`http://scg.unibe.ch/research/traits?_n&17` A list of traits implementations. Last access 11/2011.

3.2.3 Conclusion

MVC offers a clearly defined and widely known and accepted way of designing the architecture of interactive and GUI-driven software by separating the concerns of Model, View and Controller where possible. It also describes how these three isolated components should interact with each other and how change propagation can be implemented.

It has been proven that MVC exhibits weaknesses in terms of modeling system behavior, which is directly related to the static class model of OOP. In this respect, alternative models of architecture and patterns follow promising ideas of integrating object composition into system design and implementation.

However, in the current state, most popular programming languages are largely incapable of supporting design-driven object composition techniques, such as traits. What is more, current application frameworks are very often based on MVC or other architectural design principles and do not adequately support superimposing other complex design paradigms onto them.

It seems, therefore, reasonable to adhere to MVC principles in systems and development environments that are already founded on MVC. On the other hand, design patterns incorporating object composition are recommendable for further research and can prove valuable for future programming language implementations and the understanding of how to model dynamic behavior in software design.

3.3 Text recognition in natural language

As shown previously, script breakdown requires the producer to read through the script, understand the linguistic syntax as well as the semantics of the text, and categorize its elements into fitting groups while considering numerous factors. These factors include, for example, the part-of-speech the text fragments belong to, which is indirectly related to another factor of whether certain elements are important for the scheduling, setup or shooting of the script's scenes.

This chapter is dedicated to research and work concerned with automatic and semi-automatic recognition of the said factors with the use of a software application. It will further try to assess whether existing techniques can be adopted in the implementation of a script breakdown software that will help a user in carrying out tagging.

Different fields of research are concerned with the recognition of texts and their constituent parts, their semantics and syntactical structure. In general, the contributed solutions and systems are devised in order to provide (semi-)automatic extraction of information on a given text, which partly correlates to the act of script breakdown, where script-specific features are to be discovered and correctly classified into categories. However, there are no rules defining which kinds of words or word groups are relevant for script breakdown, which makes automated recognition difficult and requires flexible solutions for text recognition.

The fact that recognition of parts of natural language is not trivial, but rather demanding, is expressed aptly by the following statement found in Wilks and Stevenson (1998):

> *"It is no answer to the question 'What is a noun in German?' to answer that it is the part-of-speech that is regularly capitalised!"*

It efficiently describes the main problem of lexical ambiguity that text recognition in natural speech encounters. A prominent example of an English sentence that can be difficult to analyze even for native speakers, as mentioned by Church (1988), is the famous:

> *"The horse raced past the barn fell."*

In order to establish a context for the topic, the following sections examine related work in various areas of text recognition. They will also provide reasoning about whether or not the discussed techniques can and should be adopted for the implementation of script breakdown prototype presented later herein.

3.3.1 Part-of-speech tagging

Part-of-speech (POS) is dependent on context, because lexical analysis of words in a text is context sensitive and therefore often ambiguous, as noted before. Part-of-speech tagging is a technique and an area of scientific research concerned with this problem in order to provide methods to improve numerous real-world applications, such as spell checkers, screen readers and others.

Early work on automated POS tagging brought forth systems and prototypes based on statistical measures, assigning words and word groups their most likely tags. Church (1988), for example, proposed a POS tagger that parses natural language text and uses statistical heuristics describing which word is most likely (or often) a certain part-of-speech, e.g. a proper noun. The parser then reiterates different possible combinations to find the best solution. The statistical model is built on tagged natural language text corpora and additional dictionary data. The parser also takes into account probabilities of certain part-of-speech sequences starting and ending noun phrases.

Apart from stochastic methods, rule-based parsers have been implemented, like, for instance, described by Brill (1992). In this work, a rule based POS tagger is conceived that is able to improve and extend its set of rules from a very basic set, by dynamically creating *patch templates*. These patch templates represent rules inferred by errors the parser detects by testing its learned model on a tagged text corpus. A similar concept for grammar induction is proposed by Brill (1993), where a naive implementation is iteratively improved by applying a set of eight possible transformations on tagged and bracketed sentences and determining the best fitting one.

Brill and Marcus (1992) describe a method designed to build a part-of-speech tagger for a completely unknown language, using untagged text corpora and minimal supervision by an informant knowledgeable of the language. A text parser first scans the corpus for the 300 most frequent words and tries to find similarity pairs, as determined by their distributional divergence. The pairs are then merged into a limited set of tag classes and presented to the informant who then points out the most important classes (the so called *tag set*). The informant is then asked to enumerate example words that represent the classes well, starting with those classes where the informant can most easily name all examples. Using this *distributional fingerprint*, the system then tags words into their most likely classes, again as per their distributional divergence.

More recent research in this field is presented, for example, by Smith et al. (2004), who apply a stochastic part-of-speech tagging on a database containing publications in health sciences and related fields. Another even more recent publication (Snyder et al., 2009) investigates the possibilities of improving unsupervised part-of-speech tagging through the use of unannotated multilingual text corpora. The concept is based on the idea that lingual ambiguities vary between languages and that having access to parallel text data in other languages can mitigate the ambiguity problem. This work is especially interesting as it may provide multilingual part-of-speech tagging, which has been a difficult issue with POS tagging since the beginning. The authors show that results improve with the number of languages available.

Relevance for this work

While the promise of being able to recognize nouns and names (the parts of speech most likely to be relevant for tagging in script breakdown) seems most interesting, its use for this work is, at of the time of writing, difficult to evaluate.

This is partly due to the fact that no efficient language-independent solution has been presented yet, and while the system proposed by Snyder et al. (2009) is promising, the building and employment of POS taggers with reasonably low error-rates is highly complex. Given the fact that part-of-speech tagging by itself does not solve the task of tagging script elements into semantic, structural or organizational categories, the possible benefits do not justify the effort and compromises of building a specialized POS tagger at this time.

On the other hand, various concepts and approaches adopted for or developed in the process of engineering POS taggers, such as lookup in dictionaries or rule-based heuristics, present sound ideas and valuable input for script breakdown. What is more, incorporating pre-built multilingual POS taggers into a breakdown software at a later time seems a reasonable and promising outlook.

3.3.2 Lexical databases

Lexical databases are human-maintained, language-specific reference systems allowing external systems to look up information on given words. WordNet[5] (Princeton University, 2010) is an example of a lexical database for the English language. It is described and summarized by Miller et al. (1993) as:

[5]copyright Princeton University

“[. . .] an on-line lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, and adjectives are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets.”

In lexical databases, these relations between words are most commonly defined by the following types (Miller et al., 1993):

Synonymy means, as the name implies, similar meaning, thus one expression (word) in a context can be replaced by the other without changing the semantics of the context. Synonymy is a lexical relation between word forms.

Antonymy means that one word expresses the opposite of another word. As synonymy, it describes a lexical relation between word forms.

Hyponymy is a semantic relationship between word meanings, contrary to synonymy and antonymy. It expresses a connection between word meanings where one word superordinates another, like *tree* is a hyponym of *plant*. Consequentially, hyponymy relationships put word meanings into a hierarchical structure. The inverse of hyponymy is hypernymy. In technical terms, hyponymy describes an *is-a* relation between entities.

Meronymy is a more abstract concept describing part-whole relations between words, like *finger* being a meronym of *hand*. The inverse of meronymy is called holonymy. In technical terms, meronymy is expressed by *has-a* relations between entities.

Exploiting semantic and lexical relationships is widely used in research and practical implementations. Budanitsky and Hirst (2006) provide a survey of techniques and implementations of similarity measures based on WordNet. Based upon this foundation, frameworks for measuring similarities between concepts (such as word groups, parts of speech) have been created, for example WordNet::Similarity (Pedersen et al., 2004), a freely available perl implementation.

As with WordNet, lexical databases have been created for various other languages, with different licensing models that usually distinguish between commercial and non-commercial use. Sophisticated and well-developed lexical databases provide extensive application programming interfaces (APIs) with sets of functions to access and generate data. Most databases are available via online queries as well as downloadable text repositories and databases.

There are, and have been, projects concerned with providing multilingual and interlingual lexical database access:

EuroWordNet [6] contains seven European languages (Dutch, Italian, Spanish, German, French, Czech and Estonian) and is aligned in structure with Princeton's WordNet.

MultiWordNet [7] is a lexical database for Italian and provides WordNet-aligned access to lexical databases in five more languages (Spanish, Portuguese, Hebrew, Romanian and Latin) (Pianta et al., 2002).

Global WordNet [8] connects Spanish and Catalan to the Princeton WordNet.

At the time of writing, the core language definitions of these databases have reached completion. Through common interfaces (for example by following the WordNet structure), these databases can be connected and interlingual queries are subject to being made available (Vossen, 2002).

Relevance for this work

Lexical databases have been founded to provide fundamental information on text segments, which is strongly in line with the context of this work. Due to the size of the respective databases, many of the systems currently working with lexical databases are not distributed software suites, but rather online services providing web interfaces to said databases, such as synonym search engines.

A number of scientific projects operating in the field of computational linguistics and natural language processing use lexical databases, which are often stored and accessed locally. This is a practical solution only in research projects where distribution to end users is negligible.

[6]`http://www.illc.uva.nl/EuroWordNet/`, last access 11/2010
[7]`http://multiwordnet.fbk.eu`, last access 07/2010
[8]`http://www.globalwordnet.org`, last access 07/2010

In the context of a commercial, end-user oriented software application, distribution of such databases is impractical, especially if multilingual processing is required. Querying interlingual lexical databases online using an internet connection can provide a solution to this. However, online-querying of information on hundreds of words is subject to data bandwidth and transmission latencies.

3.3.3 Text Mining

Text mining, as noted by Tan (1999), is the act of extracting relevant and non-trivial patterns, knowledge or other information from text documents. A survey of text mining techniques presented by Hotho et al. (2005) also notes that text mining can be described as a new form of data mining that applies methods of text recognition and refining methods (from research areas concerned with parsing and recognition of text and its elements) on unstructured text documents.

Similarly, Hearst (2003) states that the key to text mining is the combination of higher-level, extracted information in order to discover new knowledge, whereas scientific advances in computational linguistics (or natural language processing, NLP), information retrieval (IR) and other research areas provide new means to segment text and analyze its compounding elements.

According to Tan (1999), systems implementing text mining can be defined by an abstract system architecture that separates the actual process of text mining into two parts:

1. **Text refining** translates arbitrary text documents into an intermediate format (IF) that is usable by the components of the system. Current systems usually use document-based or concept-based IF structures.

2. **Knowledge distillation** derives concepts and knowledge from the now structured data in the IF, for instance by classifying or clustering documents into two- or multi-dimensional spaces. Afterwards, these spaces are frequently visualized using methods of information visualization.

By using other documents' IFs as input for the text refining of a corpus of documents, the resulting IFs will convey less document-based and more concept-based information. The results from carrying out knowledge distillation on IFs depend on the nature of the IFs, as well as the used text mining functions, and can range from information extraction for document space visualization to textual summarizations of the underlying documents.

Tan (1999) states that multilingual text refining is an open issue in text mining that will considerable potential in the future due to the possibility of applying text mining on whole new information collections written in languages other than English. Cross-language text classification, as examined, for example, by Shi et al. (2010), is subject to current research projects.

Another important research topic is applying text mining in the bio-medical context. Cohen and Hersh (2005) expect high potential in making huge amounts of existing unstructured text from medical documents available for researchers and practitioners in this field. The work surveys a number of publications and concepts regarding different aspects of bio-medical text mining. They conclude by stating that the major challenge in bio-medical text mining for the next 5-10 years will be creating tools for medical researchers and practitioners to use.

This notion is shared by Holzinger et al. (2008), who propose a system implementing statistical text mining to analyze expert comments on magnetic resonance images (MRI). The application is able to calculate co-occurences of mentions of anatomic structures and pathologic expressions found in the diagnoses' reports. This method relates to the technique of *relationship extraction* found in (Cohen and Hersh, 2005). Furthermore, *synonym and abbreviation extraction*, as suggested by Cohen and Hersh (2005), has been applied, to some extent, with the recommendation of creating more thorough synonym databases for pathological terms.

While no attempt at (automatic) *hypothesis generation* (Cohen and Hersh, 2005) was made, the interface provided by the system in (Holzinger et al., 2008) allows querying of the database of indexed documents, as well as the visualization of the findings so as to allow researchers to find and prove correlations and deduce other information, such as new hypotheses.

Relevance for this work

The focus and main point of application of text mining lies, beyond the recognition of text parts and individual semantics, on the categorization and clustering of documents within an information domain (or across several), as described, for example, by Larsen and Aone (1999).

Text mining applies various text recognition techniques to extract information on the topic and other representative data from text documents. Its focus lies on documents and their places in information space, rather than text components. Hence its text recognition and concept extraction operate on a more abstract level than is required in script breakdown.

3.3.4 Stop word filtering

Stop word filtering is a technique used in computational linguistics, natural language processing (NLP) and information retrieval (IR), among others. It describes the usage of lists of so-called stop words that are to be filtered out of text in pre or post-processing steps in order to improve results of text recognition and language processing tasks (Manning et al., 2008).

Depending on the context, stop word lists can be limited mainly to articles of speech and link words, such as *the*, *a* or *and*, which most likely convey no factual information on the topic, but are used to build intelligible sentences.

Other applications, for example in information retrieval, where domain-specific information can be relevant, may incorporate lists of stop words that are automatically constructed by the use of predefined heuristics, such as when querying a certain information domain for specific search terms, where most documents share some common keywords relevant for the domain. In such a case, these keywords may be ignored using stop word filtering.

Relevance for this work

Stop word filtering is becoming less important in certain areas of research and application, for instance in IR where more sophisticated retrieval methods implicitly incorporate statistical heuristics to filter out extremely common words.

For script breakdown, however, it presents an efficient approach to provide a basic means of predicting relevance of certain parts of (multilingual) text. It is also flexibly applicable in many situations and can be well combined with other techniques of text recognition. The main drawback of the method is the dependency of a stop word list on the particular language it is constructed for. This is simply remedied by providing separate lists per supported language, which are far less complex and demanding to create than POS taggers, for example. This strategy, however, brings along the requirement of being able to automatically recognize the language of the text it is to be used on.

3.3.5 Conclusion

Table 3.1 summarizes the findings presented above. It shows that stop word filtering provides an easily and efficiently applicable method for providing basic assistance in script break down by conducting simple text recognition. It is also easily customizable, enabling script breakdown software to provide the user with means to configure its behavior.

Technique	Scope	Relevance	Complexity
Part-of-speech tagging	words, word groups	medium	medium
Lexical databases	words, word groups	medium	medium
Text mining	documents, document collections	low	high
Stop word filtering	words	medium-high	low

Table 3.1: Overview of observed text recognition techniques and their relevance for this work.

Part-of-speech tagging and lexical databases are more sophisticated and more complex to employ in an end-user compatible and distributable software. Both methods operate on the same scope and, while using different concepts, give similar results. Multilingual systems are topics of recent and current research. Text mining, as established above, operates on a higher, more abstract level and therefore turned out to be of less value to script breakdown than the text recognition methods it is based upon.

3.4 Usability engineering (UE)

Usability Engineering is the branch within the process of technological development that aims at ensuring ease of use and acceptability of the resulting solutions, prototypes and products, as pointed out by Bevan and Macleod (1994). Ease of use in this context relates, according to Holzinger (2005), to the (positive) degree of achievable user performance and satisfaction, while acceptability, as the name implies, is the deciding factor of whether the system is actually used.

One of the prime findings of human-computer-interaction (HCI), the corresponding field of research, is that usability can not (or only with tremendous effort) be applied to existing systems or finished designs. Rather it must be considered before and throughout the complete software development cycle. Besides being aware of commonly established guidelines and recommendation concerning usability and user-centered design (see, for example, Nielsen (1994a); Andrews (2010)), there are a number of usability evaluation methods that can be applied in order to assess the state of usability of the system at a certain point in development.

These usability evaluation methods are generally divided into the groups of Inspection methods and Test methods, as shown in table 3.2. The main difference between these groups is that inspection methods do not involve the participation of end users, whereas test methods do, to varying degrees (Nielsen, 1994b).

Other distinguishing features, independent of which group the respective techniques belong to, include the development phase(s) they can be applied in and the amount of required personnel, equipment and the evaluators' experience. Some of these methods are well suited to being combined (for instance conducting a Field Observation with test users and having them fill in a questionnaire afterwards) and usually a combination of techniques is able do yield improved results (Holzinger, 2005).

From another point of view, usability evaluation methods can be distinguished by considering whether the results or answers are related directly to the system (direct methods) or not (indirect methods, such as questionnaires), and how frequently they need to be conducted and reiterated. In this context, questionnaires are commonly considered to be suited to frequent usability testing and monitoring of results, in

Inspection Methods	**Heuristic Evaluation**	**Cognitive Walkthrough**	**Action Analysis**
Phases	all	all	design
Intrusive	-	-	-
Requirements			
Time	low	medium	high
Users	-	-	-
Evaluators	3+	3+	1-2
Expertise	medium	high	high
Equipment	low	low	low

(a) Inspection Methods in Usability Evaluation

Test Methods	**Thinking Aloud**	**Field Observation**	**Questionnaire**
Phases	design	final testing	all
Intrusive	yes	yes	no
Requirements			
Time	high	medium	low
Users	3+	20+	30+
Evaluators	1	1+	1
Expertise	medium	high	low
Equipment	high	medium	low

(b) Test Methods in Usability Evaluation

Table 3.2: Comparison of Usability Evaluation techniques: Inspection Methods (a) and Test Methods (b) (adapted from (Holzinger, 2005)).

order to be able to quickly estimate the outcome and possible other implications of usability measures invoked in the development process.

3.4.1 Location and synchronicity in usability evaluation

In recent years, usability engineering has adopted means to expand usability evaluation to de-centralized and asynchronous environments and a number of articles have been published, examining the feasibility and suitability of such techniques for the evaluation of systems ranging from websites to desktop software.

Andreasen et al. (2007) provide an empirical study of three remote usability evaluation and inspection methods employed in both synchronous and asynchronous environments. They present various measures, like average number of severe/unique usability problems found per methodology, and conclude that, while synchronous methods show promising results, asynchronous methods seem to be unreliable, which is attributed to the low numbers of test users (six per methodology). At the same time, asynchronous methods are deemed worthwhile due to their flexibility in distributing and evaluating test runs, making it feasible to collect larger amounts of data than with, for example, classical local thinking aloud tests.

Bruun et al. (2009) present a more in-depth analysis of asynchronous remote usability test methods, stating that, while they tend to discover less usability problems than a classical local lab-approach, they also take far less time to carry out and analyze. As with Andreasen et al. (2007), the number of test users in this work is also rather limited (ten per methodology).

Other interesting concepts in this field concern (semi-)automatic analysis of usage and usability data through various means and sources, such as a user's interaction with the software (Hilbert and Redmiles, 2000). Moreover, particular types of interaction are interesting, like backtracking/undo actions, which, in the context of software usability tests, can be interpreted as indicators of errors or other problems (Akers, 2009).

3.4.2 System usability scale (SUS)

With the knowledge of how important frequent and iterative usability assessments are in the software development process, it becomes more desirable to establish efficient, low-cost evaluation methods that can be repeated quickly and as necessary. SUS (Brooke, 1996) is a usability measurement that was conceived to suit this purpose by defining a simple standardized questionnaire and a heuristic to generate a usability score from the answers given.

The questionnaire (see A) consists of ten questions that are to be answered on a Likert-scale with coded values of 1..5. The resulting score is determined by a heuristic in the following way:

1. Calculate the sum of the decoded scores of all questions:
    - For every odd question (1, 3, 5, 7, 9) the decoded score is $score - 1$ (*positive* questions)
    - For every even question (2, 4, 6, 8, 10) the decoded score is $5 - score$ (*negative* questions)

2. Multiply the sum with 2.5 to make it reflect a percentage such that $0 \leqslant score \leqslant 100$ with 0 being the lowest and 100 being the highest (best) achievable score.

3. Repeat these steps for every test person and aggregate an average value that will be considered as a reference value in the next test run.

Following this heuristic, a concrete SUS usability score can be identified as per the corresponding development state of the system. By repeating the test and monitoring and comparing both the aggregate scores and all sub-scores of particular questions, SUS represents a meaningful indicator of how recent usability-related measures and other changes to the system have improved (or deteriorated) the system's usability rating.

4. Current System

In order to better understand why there is need for improvement, one has to evaluate and analyze the deficiencies of the current processes and tools. The following sections will therefore describe the findings of observations conducted on film production management and the adoption of standard and expert software. Strengths, weaknesses and opportunities for improvement will be pointed out.

4.1 State of the art

As has already been stated, a great number of film production professionals still resort to using highlighters and script printouts or, slightly more software aided, highlighting text manually using annotation tools in certain PDF (Portable Document Format, Adobe Systems Incorporated (2010)) applications.

This consequently creates the need to manually transfer the marked elements into a number of places within scheduling and budgeting, which themselves may or may not be covered by software aided processes, depending on the project, the production company or individual preferences and experience. In addition to the obvious increase in manual work, such steps of transition are much more error-prone than automated or semi-automated software aided processes.

In the context of script breakdown, there are some reasons why production managers or the directors of certain scenes can benefit from working through the scripts thoroughly and iteratively. As Clevé (2005) states, such direct interaction with the script may create a more pronounced understanding of the scenes and their requirements. It is, however, to be questioned whether the repeated manual breaking down of script elements is the actual benefiting factor, or whether such deeper involvement with the script can be gained in other ways, for example during inter-team conferences and rehearsal sessions, aided, for example, by printouts of automated breakdown sheets as a basis for discussion.

Leaving this thought aside in order to concentrate on the focus of this work, another question at hand demands an answer: it is essential to know if the software which is currently available is adequate to complete the tasks inherent in the process observed herein, in terms of both functionality and usability.

The following sections will present a number of popular, relevant software applications. Depending on which platforms the respective software is available for, all observations and tests were conducted on Apple Mac OS X 10.6+, Microsoft Windows XP with Service Pack 3, and Microsoft Windows 7, where applicable.

4.1.1 CeltX

CeltX[1] is a freely available, open source software application with a focus on media pre-production. It is intended to serve as an all in one solution for pre-production tasks, providing interfaces for authoring scripts and storyboards, breaking down elements, scheduling and generating reports.

The software is based on Mozilla[2] and released under the CeltX Public License[3]. It is available on multiple platforms and extensible through downloadable add-on programs. It is also continuously enhanced and updated by the developers.

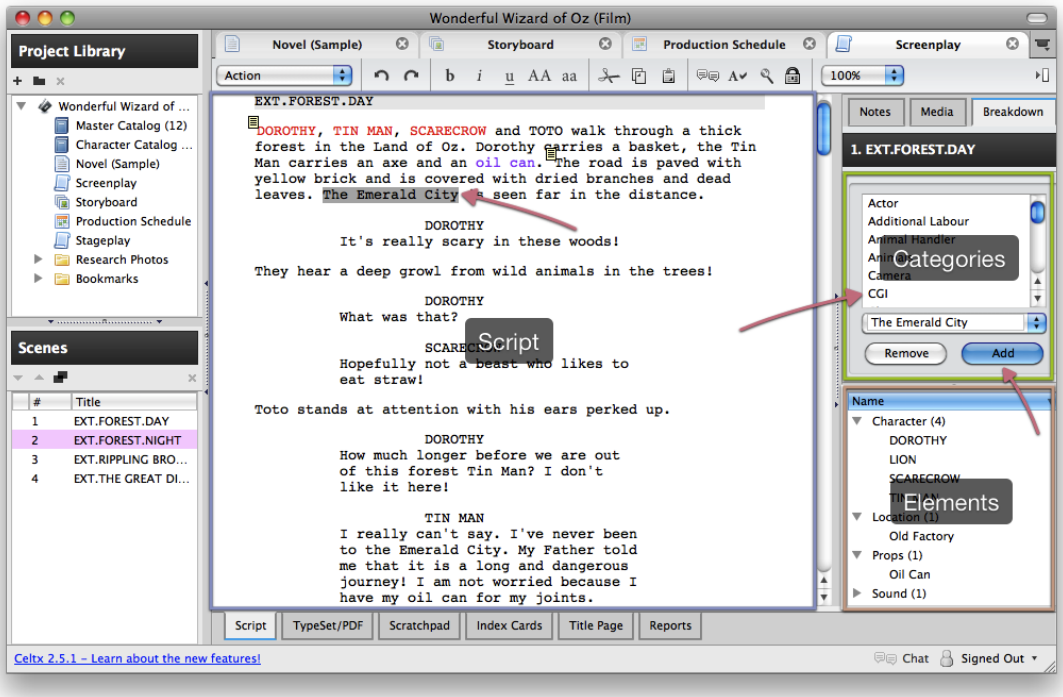
In addition to the freely distributed client software, CeltX supports the forming of collaboration teams, so called *CeltX Studios*, with prices depending on numbers of team members, known as users. Studios can be accessed from the CeltX home page and are also integrated directly into the client software. They provide shared access to project files created with CeltX, as well as a set of collaboration functions like team-chats and optional mutually exclusive access for writing and deleting documents.

At the time of writing, version 2.5.1 was available on Windows, Mac OS X 10.4+ and Linux. CeltX 2.5.1 was tested on Windows 7 and Mac OS X 10.6.

[1]`http://www.CeltX.com` CeltX home page, last access 09/2010
[2]`http://www.mozilla.org` Mozilla software project, last access 09/2010
[3]`http://CeltX.com/CePL/` CeltX Public License Version 1.3, last access 09/2010

Script breakdown

CeltX features script breakdown by tagging elements and includes the generation of reports of data. The breakdown interface only supports one way of tagging elements, which is shown in figure 4.1.



```
EXT.FOREST.DAY
DOROTHY, TIN MAN, SCARECROW and TOTO walk through a thick
forest in the Land of Oz. Dorothy carries a basket, the Tin
Man carries an axe and an oil can. The road is paved with
yellow brick and is covered with dried branches and dead
leaves. The Emerald City is seen far in the distance.

                    DOROTHY
          It's really scary in these woods!

They hear a deep growl from wild animals in the trees!

                    DOROTHY
          What was that?

                    SCARECROW
          Hopefully not a beast who likes to
          eat straw!

Toto stands at attention with his ears perked up.

                    DOROTHY
          How much longer before we are out
          of this forest Tin Man? I don't
          like it here!

                    TIN MAN
          I really can't say. I've never been
          to the Emerald City. My Father told
          me that it is a long and dangerous
          journey! I am not worried because I
          have my oil can for my joints.
```

Figure 4.1: Script breakdown interface in CeltX.

In order to tag an element, the user first marks the corresponding text within the script (by mouse or keyboard), then selects the appropriate category from the list on the upper right hand side and confirms by pressing the *Add*-button below.

The set of tagged elements in the currently selected scene is outlined in the lower right box, which can also be used to un-tag elements. Tagged elements are visualized in the script by certain font colors associated with the category they are tagged in.

Conclusion

While CeltX offers an extensive set of features for pre-production, its breakdown functionality shows a number of deficiencies, such as the lack of simpler alternative ways to tag elements, like using keyboard shortcuts or context menus.

Moreover, some aspects of the software lack flexibility. For example, the list of breakdown categories can not be customized per project, but only for the whole application, which can result in confusion and errors if a project that has elements tagged in categories that are were set unavailable from within another project is opened. What is more, the colors associated with those categories are neither customizable, nor visible anywhere in the software.

The reports dialogue is able to filter generated scene breakdown sheets per category and element. It does not, however, support generating location breakdown sheets or cast breakdown sheets. In addition, the layout of the generated reports is not customizable.

One positive aspect of the breakdown function is the direct integration with the project catalog and the scheduling part of the software.

4.1.2 Final Draft and Final Draft Tagger

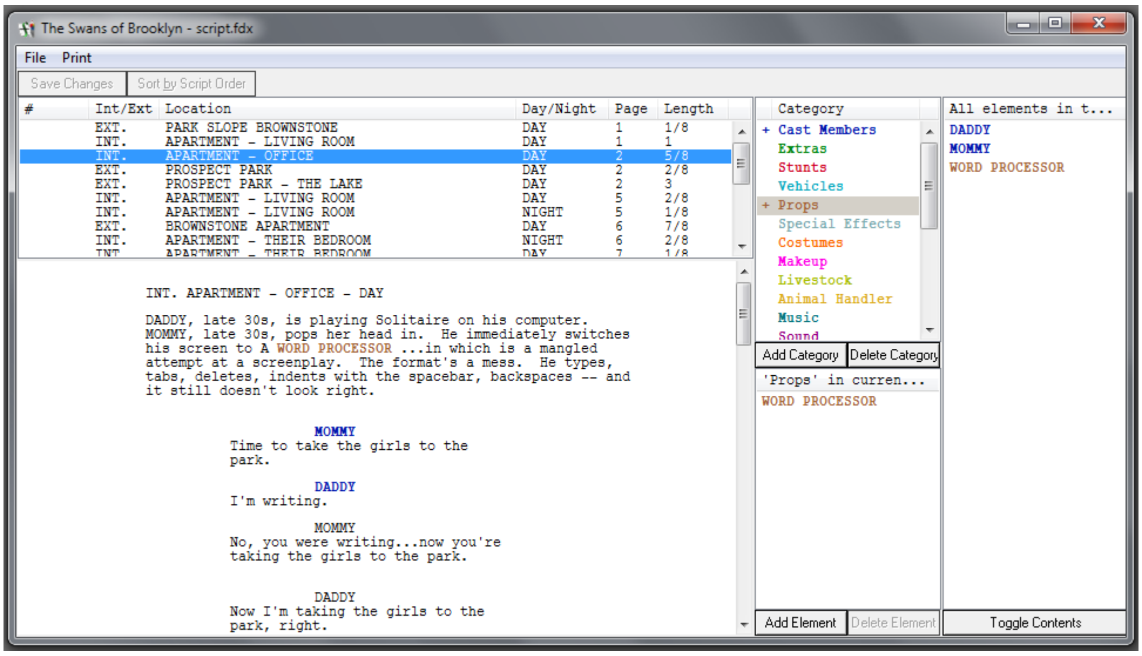
Final Draft[4] is commercial screenwriting software by Final Draft, Inc.. Besides a set of functions and interfaces for authoring scripts, it also provides a standalone application for script breakdown, called Final Draft Tagger. The software is well known and is one of the most important screenwriting products in the US.

At the time of writing, Final Draft 8 and Final Draft Tagger 2 were available for Microsoft Windows and Apple Mac OS X and were tested on Windows 7 and Mac OS X 10.6.

Script breakdown

Tagger allows the opening of Final Draft document files for breakdown. Its main window, as depicted in figure 4.2, consists of a list of scenes, a text view of the current script portion, and a breakdown interface containing lists of categories, elements and contents. Tagging of elements is accomplished by selecting the respective text in the script view and either hitting the right mouse button, or by pressing the Add Element button. In the following dialog, the element text and the category can be changed and upon hitting one of the OK buttons, the element is tagged for the

[4] `http://www.finaldraft.com/products/final-draft/` Final Draft product page, last access 08/2010



#	Int/Ext	Location	Day/Night	Page	Length
	EXT.	PARK SLOPE BROWNSTONE	DAY	1	1/8
	INT.	APARTMENT - LIVING ROOM	DAY	1	1
	INT.	APARTMENT - OFFICE	DAY	2	5/8
	EXT.	PROSPECT PARK	DAY	2	2/8
	EXT.	PROSPECT PARK - THE LAKE	DAY	2	3
	INT.	APARTMENT - LIVING ROOM	DAY	5	2/8
	INT.	APARTMENT - LIVING ROOM	NIGHT	5	1/8
	EXT.	BROWNSTONE APARTMENT	DAY	6	7/8
	INT.	APARTMENT - THEIR BEDROOM	NIGHT	6	2/8
	INT.	APARTMENT - THEIR BEDROOM	DAY	7	1/8

```
INT. APARTMENT - OFFICE - DAY

DADDY, late 30s, is playing Solitaire on his computer.
MOMMY, late 30s, pops her head in.  He immediately switches
his screen to A WORD PROCESSOR ...in which is a mangled
attempt at a screenplay.  The format's a mess.  He types,
tabs, deletes, indents with the spacebar, backspaces -- and
it still doesn't look right.


                    MOMMY
          Time to take the girls to the
          park.

                    DADDY
          I'm writing.

                    MOMMY
          No, you were writing...now you're
          taking the girls to the park.


                    DADDY
          Now I'm taking the girls to the
          park, right.
```

Category
+ Cast Members
Extras
Stunts
Vehicles
+ Props
Special Effects
Costumes
Makeup
Livestock
Animal Handler
Music
Sound

'Props' in curren...
WORD PROCESSOR

All elements in t...
DADDY
MOMMY
WORD PROCESSOR

Figure 4.2: Script breakdown interface in Final Draft Tagger.

current scene. The resulting breakdown data can be printed out (scene and element report) and exported into a scheduling file format.

Conclusion

Serving the sole purpose of providing breakdown functionality for existing Final Draft documents, the importing capabilities are limited to this file format.

The main interface shows some specialization for the task, for instance by providing a useful list of recognized scenes. Tagging itself is cumbersome due to the imperative step through the Add Element dialog and the lack of hotkeys or similar instruments. In addition, no global (script-wide) tagging is available. Category management is sufficient, but editing categories is only accessible through double clicking the right mouse button. This feature was discovered in testing simply by accident.

The software's printing capabilities are limited due to a lack of customization features and breakdown page layouts differing from industry standards. Data export is only available in a single, proprietary format and is thus very inflexible.

4.1.3 Mindstar Cinergy

Cinergy 2000 Motion Picture Production System (MPPS) Version 5[5] is a commercial suite of software by Mindstar Productions that covers several aspects of the film production process, including budgeting, scheduling and labor rate integration.

According to the product homepage, there has not been a new version of the software since 2004. Furthermore, the software is only available for Windows operating systems. Cinergy 2000 MPPS 5 was tested on Windows XP and Windows 7.

Script breakdown

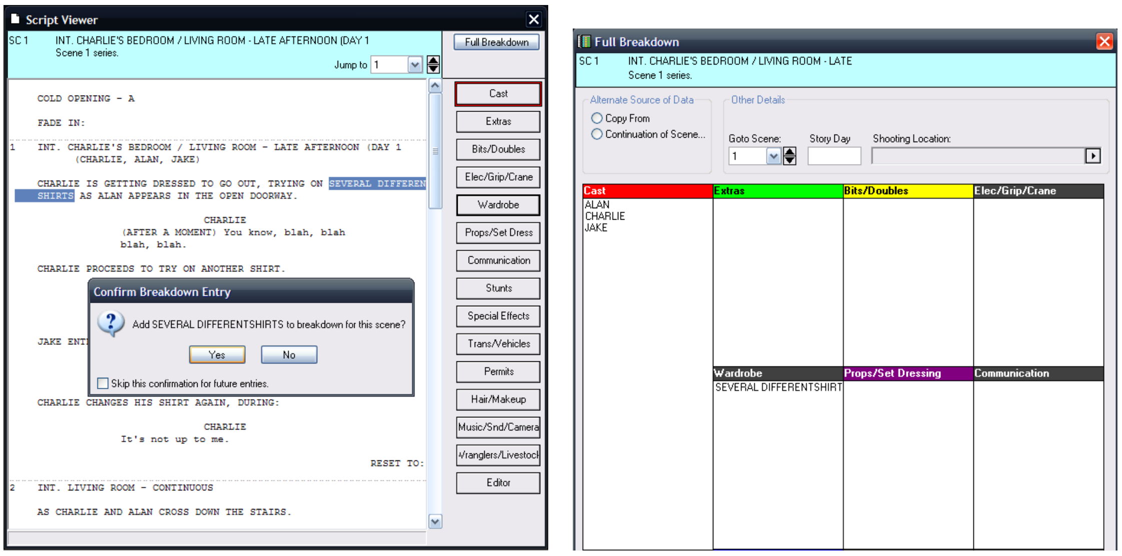
The suite's scheduling module supports the breaking down of scripts for the creation of script breakdown sheets as well as using the data for scheduling and budgeting tasks. The script breakdown editor also supports to import scripts from a number of file types, including proprietary formats, and providing various options, for example custom category mappings and formatting.

Tagging of elements can be done by means of two different interfaces. The first (shown in figure 4.3a) contains a view of the script and supports tagging by several means, including: selecting text and pressing the category button, selecting text and dragging/dropping it onto the category button, selecting text and tagging per context menu. The second interface (depicted in figure 4.3b) is laid out like a breakdown sheet and supports the direct editing of elements from within the table's category cells.

Data integration

Breakdown information is automatically integrated and synchronized with the on-the-set module for the purpose of keeping scheduling consistent and informing responsible personnel.

[5]`http://www.mindstarprods.com/cinergy/overview.html` Mindstar Productions Cinergy 2000 MPPS Product Information, last access 08/2010



SC 1 INT. CHARLIE'S BEDROOM / LIVING ROOM - LATE AFTERNOON (DAY 1
Scene 1 series.

```
    COLD OPENING - A

    FADE IN:

1   INT. CHARLIE'S BEDROOM / LIVING ROOM - LATE AFTERNOON (DAY 1
          (CHARLIE, ALAN, JAKE)

    CHARLIE IS GETTING DRESSED TO GO OUT, TRYING ON SEVERAL DIFFEREN
    SHIRTS AS ALAN APPEARS IN THE OPEN DOORWAY.

                          CHARLIE
              (AFTER A MOMENT) You know, blah, blah
              blah, blah.

    CHARLIE PROCEEDS TO TRY ON ANOTHER SHIRT.

    JAKE ENT

    CHARLIE CHANGES HIS SHIRT AGAIN, DURING:

                          CHARLIE
              It's not up to me.

                                                  RESET TO:

2   INT. LIVING ROOM - CONTINUOUS

    AS CHARLIE AND ALAN CROSS DOWN THE STAIRS.
```

Confirm Breakdown Entry

Add SEVERAL DIFFERENTSHIRTS to breakdown for this scene?

☐ Skip this confirmation for future entries.

SC 1 INT. CHARLIE'S BEDROOM / LIVING ROOM - LATE
Scene 1 series.

Cast	Extras	Bits/Doubles	Elec/Grip/Crane
ALAN CHARLIE JAKE			
	Wardrobe	**Props/Set Dressing**	**Communication**
	SEVERAL DIFFERENTSHIRT		

(a) Script View, showing tagging per dialog **(b)** Full Breakdown View (truncated)

Figure 4.3: Two breakdown interfaces in Mindstar Cinergy: Script View (a) and Full Breakdown View (b).

Conclusion

The script breakdown module provides a surprisingly extensive set of tagging abilities that are suitably implemented, but show some usability issues. The fact that the full breakdown interface, once its window is visible, renders the script view interface unable to interact with, diminishes its value greatly.

Customizable importing of various script formats is done well, whereas export and printing functions lack flexibility, particularly in terms of customizable layouts. Data integration is limited with no ability to directly use breakdown data for creating calculation positions with the aid of the built-in rate labor browser.

4.1.4 Movie Magic Scheduling and Budgeting

Movie Magic Scheduling (MMS) and Movie Magic Budgeting (MMB) are two modules of a film production software suite by Entertainment Partners (EP)[6] which also includes specialized applications for accounting, labor rates and online collaboration. The suite does not include screenwriting software.

MMS and MMB are rebranded and functionally extended versions of the well-

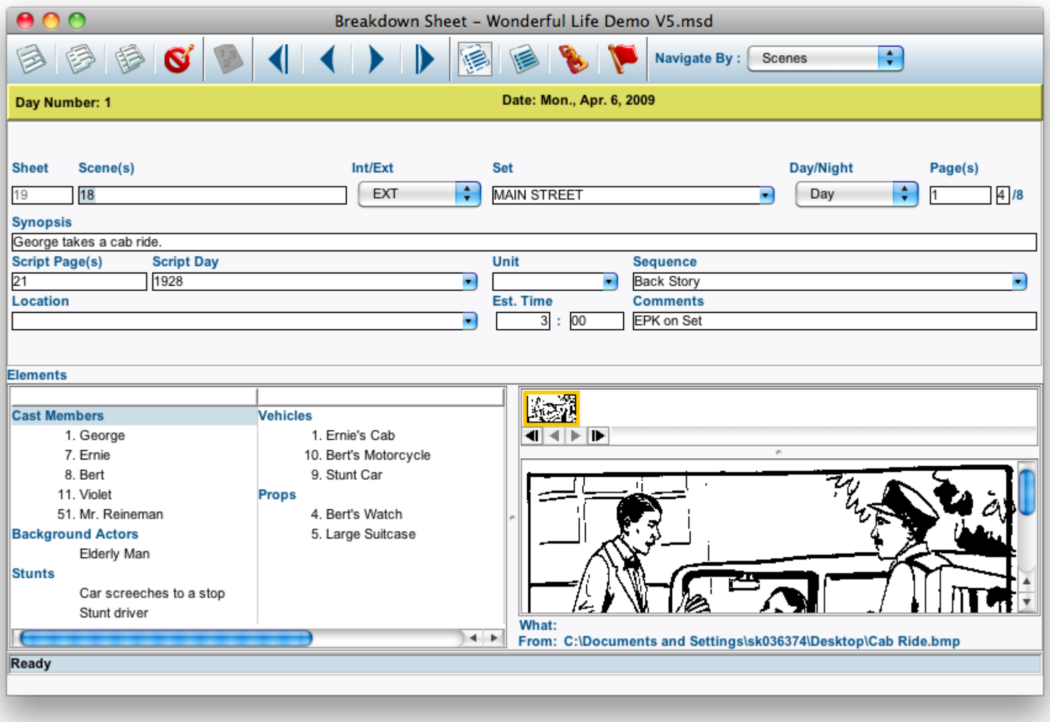
[6]`http://www.entertainmentpartners.com` Entertainment Partners home page, last access 08/2010

known EP Scheduling and EP Budgeting applications, which themselves are rooted in a Movie Magic Scheduling/Budgeting suite that EP bought in 1999. The Movie Magic Screenwriter application presented in section 4.1.5 was part of this original Movie Magic suite too, but was not sold with the rest of the suite and remained with the inventors, Write Brothers Inc., as discussed below.

MMS 5 and MMB 7 are available both for Windows and Mac OS X systems and were tested on Windows XP, Windows 7 and Mac OS X 10.6.

Script breakdown

While there is no support for screenwriting from within MMS, MMB or any other part of the suite, existing scripts can be imported from a limited number of supported formats. Additionally, MMS supports the manual entering of breakdown elements for scenes either via a quick entry dialog or by entering elements directly in the breakdown outline (shown in figure 4.4). Both features only work locally. Breakdown


Breakdown Sheet – Wonderful Life Demo V5.msd

Navigate By : Scenes

Day Number: 1 Date: Mon., Apr. 6, 2009

Sheet: 19
Scene(s): 18
Int/Ext: EXT
Set: MAIN STREET
Day/Night: Day
Page(s): 1 4/8
Synopsis: George takes a cab ride.
Script Page(s): 21
Script Day: 1928
Unit:
Sequence: Back Story
Location:
Est. Time: 3 : 00
Comments: EPK on Set

Elements

Cast Members
1. George
7. Ernie
8. Bert
11. Violet
51. Mr. Reineman

Background Actors
Elderly Man

Stunts
Car screeches to a stop
Stunt driver

Vehicles
1. Ernie's Cab
10. Bert's Motorcycle
9. Stunt Car

Props
4. Bert's Watch
5. Large Suitcase

What:
From: C:\Documents and Settings\sk036374\Desktop\Cab Ride.bmp

Ready

Figure 4.4: Script breakdown interface in Movie Magic Scheduling.

data can be exported in a number of sheet formats that can also be customized by the user.

Data integration

Elements created in an MMS 5 project can be exported into an MMB 7 library file. Users of MMB 7 can import and use these library files as catalogs for their budgeting tasks.

Conclusion

The Scheduling and Budgeting applications of the suite offer a wide range of functionality with limited capabilities in script breakdown and data integration. Export functions are extensive with good customization and layouting options, whereas the recently refreshed user interfaces still lack polish (mainly in the Mac version) and suffer from unintuitive workflows.

4.1.5 Movie Magic Screenwriter

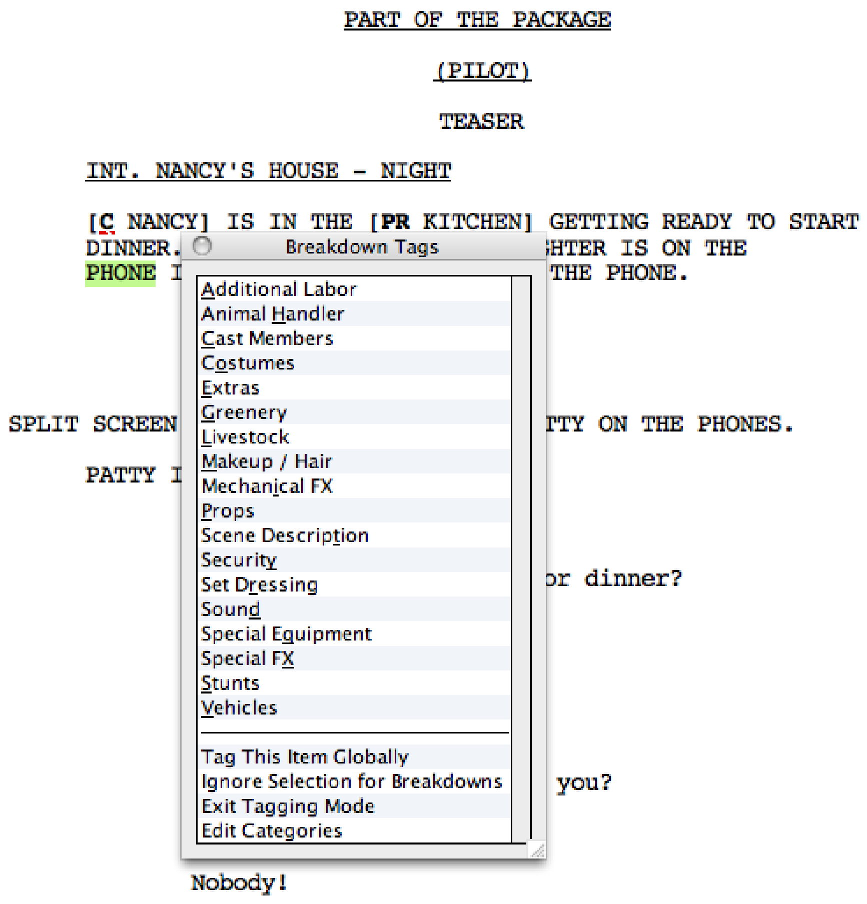
Movie Magic Screenwriter by Write Brothers Inc.[7] is a screenwriting software for Windows and Mac OS systems supporting script breakdown and the generating and printing of reports. Screenwriter 6 was tested on Windows 7 and Mac OS X 10.6.

Script breakdown

The software provides a tagging mode which, when activated, allows the tagging of selected text into breakdown categories by using a small category list window that opens whenever text is selected in the script view (see figure 4.5). Categories in this window can be selected by clicking on them or by pressing the associated short-key on the keyboard (keys from *a* to *z*).

This window also allows un-tagging, as well as local (only for the current scene) and global (for all scenes) tagging. In addition to this, elements can be tagged globally via a dedicated dialog window. The list and names of categories can be managed and the page layout of generated breakdown sheets can be customized to some extent. Breakdown data can also be exported for scheduling in one proprietary file format.

[7] `http://www.write-bros.com` Write Brothers Inc. home page, last access 08/2010



PART OF THE PACKAGE

(PILOT)

TEASER

INT. NANCY'S HOUSE - NIGHT

[C NANCY] IS IN THE [PR KITCHEN] GETTING READY TO START DINNER. HTER IS ON THE PHONE I THE PHONE.

SPLIT SCREEN TTY ON THE PHONES.

PATTY I

or dinner?

you?

Nobody!

Figure 4.5: Script breakdown interface in Movie Magic Screenwriter.

Conclusion

Tagging works effectively, especially when using shortcuts for specifying categories. The highlighting of tagged elements, on the other hand, is too low-key, with only a textual indicator as a prefix for the respective element. Category list management lacks flexibility, as does the data export for scheduling. Printing is solid with a number of low-impact options.

4.1.6 Conclusion

The previous sections intended to give an impression of the qualities and possible deficiencies of state of the art film production management software. Table 4.1 presents a summary of the findings, broken down into the most important features relevant for this work.

The most extensive set of features in terms of breakdown and integration is pro-

Software	W	S	B	Breakdown	Integration
CeltX	Yes	Yes		Limited (no data export)	Limited (central master catalog, no budgeting)
Final Draft + Tagger	Yes			Yes (Tagger)	Limited (only with scheduling)
Cinergy MPPS	Yes	Yes	Yes	Yes	Limited (only with scheduling)
MMS / MMB		Yes	Yes	Limited (import, manual input with MMS)	Limited (element library, not automated)
MM Screenwriter	Yes			Yes	

Table 4.1: Comparison of film production software by key features.

vided by Mindstar Cinergy. However, the software is not maintained anymore and is only available on Microsoft Windows platforms. All other applications support interfaces for tagging, importing or manually entering breakdown data, most of which lack task-specific features, as discussed.

This indirectly relates to the fact that none of the observed applications provide any assistance or indication to the user as to what parts of the text represent likely important/unimportant elements of the script. For example, there are no features which use text recognition to aid the user.

A number of the observed software suites make an effort to provide features for importing and exporting data, but again all those efforts only provide limited use with little to no automation or direct integration with budgeting tasks.

The second important aspect of this chapter, besides discussing sophistication and feature extensiveness, was to make a statement about the softwares' usability in terms of the factors described and established in 3.4. Again, the observed software products fall short in various aspects, most importantly in the intuitiveness of workflows and simplicity of interfaces. This is reflected commonly by tagging mechanisms that do not follow the natural and intuitive flow of actions, but rather present separate tagging dialogs and other superfluous prompts.

4.2 Making a business case for smart script breakdown software

Chapter 2 provided an overview of the organizational context the film production process is embedded in. It also noted how the script breakdown acts as a basis for cost estimations and budget negotiations, provides essential data for creating and optimizing production schedules and represents a central communication device through breakdown sheet reports.

The findings of chapter 4 show that current software solutions for script breakdown do not satisfy high standards in software usability and often do not implement specialized or smart interfaces which are adequate for the task. While some of the studied software suites allow exporting and importing of proprietary file formats, the data exchange is often limited to specific applications and certain parts of data.

Therefore, a smart script breakdown software (like will be described in more detail in the following chapters) has to offer benefits in a number of areas to film production companies using currently available software products for script breakdown and even more so to companies where producers and production assistants still carry out breakdown manually using a pen and paper or standard software like word processors or PDF annotation tools.

4.2.1 Potential process improvements

Using script breakdown applications can be beneficial for a number of independent, essential tasks in film productions. What is more, well-integrated and smart script breakdown software can enable the improvement of whole processes, both along the four film production phases and across organizational processes.

Tables 4.2 and 4.3 demonstrate the potential for the budgeting and scheduling processes, respectively, with a number of concrete examples. Table 4.4 adds to this how various organizational processes across and outside of current film production can be improved.

Process Improvement	Effect
Budgeting - Phase: Preparation	
Assets and Results: initial breakdown, cost estimation	
Better interface for faster (and more enjoyable) task completion	More cost-efficient, earlier cost estimations, earlier decisions (*green light*), less change latency
Smart features (global tagging, scene selection, rate book integration) for more accurate data	Better cost estimations, more accurate data for preliminary budget, more accurate data for scheduling
Budgeting - Phase: Pre-Production	
Assets and Results: preliminary budget	
Import of breakdown data for quick and error-free data exchange	More cost-efficient, less errors, more accurate/complete budget data

Table 4.2: Potential improvements of the budgeting processes in the context of filmproduction by employing smart script breakdown software.

4.2.2 Cost and risk assessment

The total costs associated with introducing new software into existing business processes and workflows comprise numerous factors, such as licensing fees, internal/external development costs, hardware expenses, training and education and integration into the process environment.

In the case of acquisition of an existing solution, the development costs are negligible, but integration demands thorough risk assessment. The commission of a system to an external developer or assigning internal resources to development brings along higher development costs, but allows for continuous integration of the solution into the running system. This can greatly reduce adoption risk and delays in productivity.

Software Process Improvement (SPI)

An important aspect of internal software development is continuous improvement of the software development process, also referred to as SPI (software process improvement). Dybå (2005) notes that SPI's origins lie in quality management and that it is closely related to organizational learning.

Harrison et al. (1999) state that conducting SPI is expected to benefit a company in various ways, including improved cost efficiency through reduced development

Process Improvement	Effect
Scheduling - Phase: Pre-Production	
Assets and Results: initial shooting schedule	
Import of breakdown data for quick and error-free data exchange	More cost-efficient, less errors, more accurate shooting schedule
Collaborative work on schedule with creatives, location managers and other team members	Better communication across divisions, integrate team in scheduling process, find errors early
Scheduling - Phase: Production	
Assets and Results: shooting (not immanent)	
Automatic/standardized reporting to keep staff informed	Standardized breakdown sheets improve intuitiveness, automatic change propagation and task delegation, less communication overhead, common understanding

Table 4.3: Potential improvements of scheduling processes in the context of filmproduction by employing smart script breakdown software.

times and lower error and bug rates, higher customer satisfaction, less expenses on support, and many more. The authors show how economically sound estimations of investment into SPI can be made by employing proven financial measures, and consequentially how investments can be justified economically. The authors recommend Net Present Value (NPV) as a key indicator for SPI investments and provide reasoning on how implementation time, costs and risks can be considered in NPV calculations and comparative investment evaluation.

As a result of an empirical investigation of success factors in SPI, Dybå (2005) finds that the two most important key factors are *business orientation*, i.e., how thoroughly SPI goals are set in accordance with the company's business orientation, and *employee participation*, which is defined by considering how extensively employees use their knowledge and experience to act for the good of SPI.

These findings, expressing the importance of business decisions being aligned with internal software development processes, are in concordance with the reasoning provided by Abrahamsson et al. (2005), who suggest agile project management and test driven development (TDD) as potential safeguards against development risks.

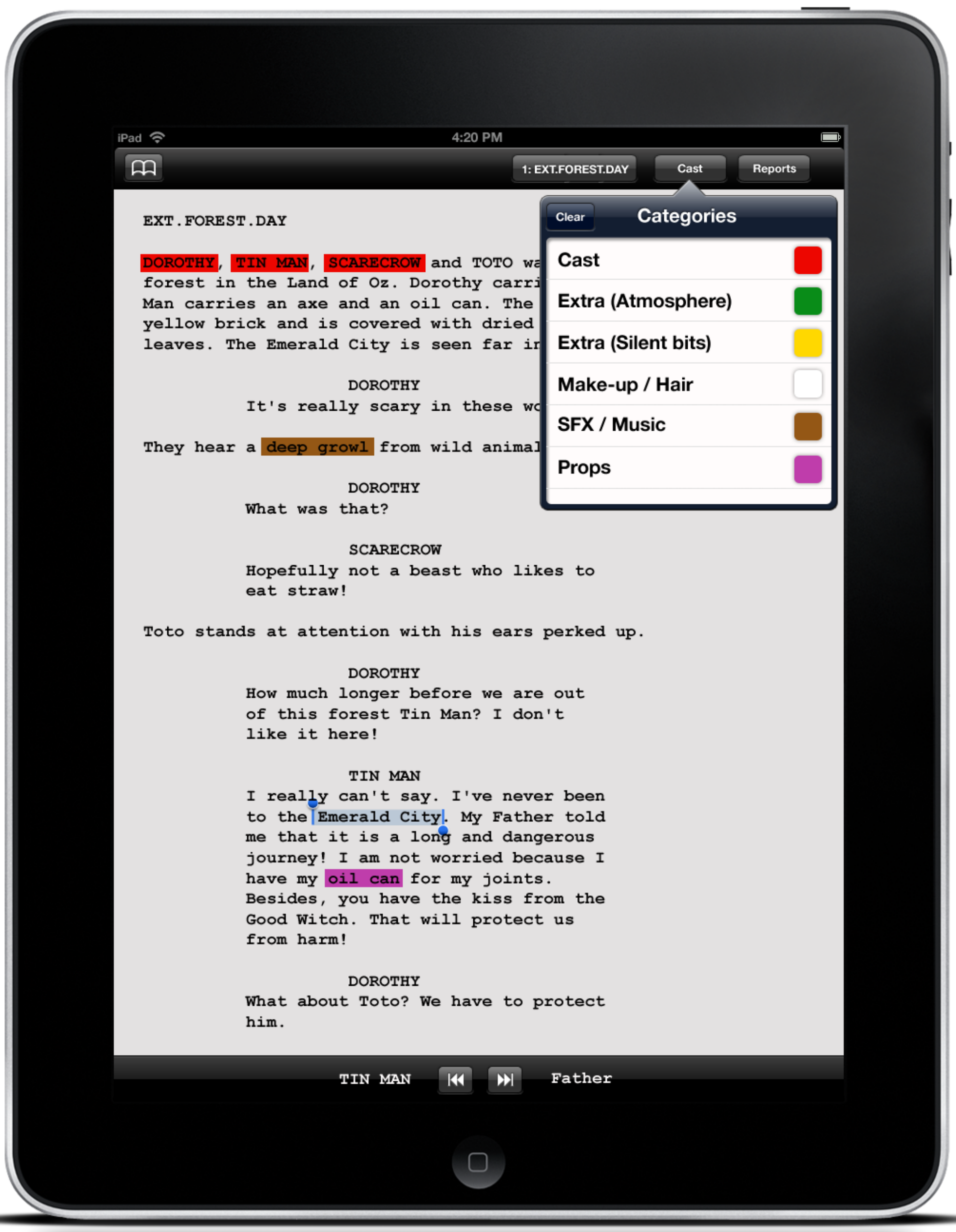
Process Improvement	Effect
Documentation - Phases: All	
Assets and Results: version protocols, other documentation	
Versioning and reporting mechanisms create automatic project history	Protocol of project (script, scheduling) changes, keep documentation after project was finished, compare project histories
Communication - Phases: All	
Assets and Results: reporting, information distribution	
Ensure and improve common understanding through reporting	Less communication overhead across divisions and processes
Escalation - Phases: Pre-Production, Production	
Assets and Results: adapted breakdown version, reporting	
Allow fast re-assessment of breakdown to react to budgeting or scheduling changes	Re-establish budget or scheduling quickly, automatically inform team members
Controlling - Phases: Production, Post-Production	
Assets and Results: version protocols, change documentation	
Breakdown data provides documented data source for cost estimations and budgets	Better knowledge of cost structures, improved traceability of irregularities and errors

Table 4.4: Potential improvements of organizational processes in the context of filmproduction by employing smart script breakdown software.

4.2.3 Outlook and application spectrum

Once an integrated software solution for script breakdown has been put into place, further development to automate and improve processes, as well as ensure quality, can be attempted. For example, direct integration with scheduling and budgeting solutions to enable automated change propagation and data updates, as well as additional controlling mechanisms, and distributed collaboration features via internet connection and across hardware platforms, as shown in figure 4.6, are recommendable.

Besides the main area of application, the concepts and interfaces devised for script breakdown can also be transferred to other areas. One example is the work of stage and set designers, who, similarly to a film production manager, need to analyze scripts for stage or film productions in order to create cost estimations and



Cast	🟥
Extra (Atmosphere)	🟩
Extra (Silent bits)	🟨
Make-up / Hair	⬜
SFX / Music	🟫
Props	🟪

```
EXT.FOREST.DAY

DOROTHY, TIN MAN, SCARECROW and TOTO wa
forest in the Land of Oz. Dorothy carri
Man carries an axe and an oil can. The
yellow brick and is covered with dried
leaves. The Emerald City is seen far in

                    DOROTHY
          It's really scary in these wo

They hear a deep growl from wild animal

                    DOROTHY
          What was that?

                    SCARECROW
          Hopefully not a beast who likes to
          eat straw!

Toto stands at attention with his ears perked up.

                    DOROTHY
          How much longer before we are out
          of this forest Tin Man? I don't
          like it here!

                    TIN MAN
          I really can't say. I've never been
          to the Emerald City. My Father told
          me that it is a long and dangerous
          journey! I am not worried because I
          have my oil can for my joints.
          Besides, you have the kiss from the
          Good Witch. That will protect us
          from harm!

                    DOROTHY
          What about Toto? We have to protect
          him.
```

Figure 4.6: Mock-up showing a possible interface of a script breakdown application running on an Apple iPad.

budgets. In later phases, breakdown data is analogously required to provide the basis for negotiations and to design and compose adequate, persuasive backdrops and other stage props.

Another interesting idea is integration as part of a personal learning system, where the application would provide tools and mechanisms to mark important definitions, key dates, names and similar. The application could then be used to create automated reports with summarizations of the tagged context, possibly enriched with data and information from online resources selected by the user.

In the scientific field, the tagging of free text is also often a requirement, for example to create annotated free text corpuses for the training of learning algorithms.

4.2.4 Conclusion

Adopting and integrating new software into existing business processes bears certain risks and costs, especially if existing solutions are to be replaced. Careful evaluation and consideration of quality-ensuring measures, such as continuous integration, can help mitigate negative influence factors.

On the other hand, task and process improvements can account for saved costs, employee motivation and increased process knowledge drawn from better communication and documentation. These are important factors for learning organizations.

5. Materials and Methods

The discussion of film production software in the previous chapters has shown that most of the applications providing mechanisms for script breakdown lack suitable tagging interfaces and provide little means of data pre-processing, export and interchangeability.

This chapter aims at creating a concept for a more efficient and satisfying script breakdown application supporting suitable, specialized tagging interfaces while strongly considering particular requirements such as script pre-processing, category management, global/local tagging and export of breakdown data.

5.1 Use cases

Use case diagrams have been constructed in order to summarize and depict the main use case scenarios of an application for script breakdown. Figure 5.1 describes the most general, abstract use case scenarios like script breakdown, export to other software and generating (printed) reports. In most cases, the Production Manager (PM) or Assistant Production Manager (PMA) is responsible for carrying out script breakdown and providing according reports. The Producer is also an important user who requires the software for assistance in calculating cost estimations and transferring the data into scheduling and budgeting applications.

Other members of production staff requiring data from script breakdown include Unit Production Managers (UPM), Location Managers (LM), Set Designers and many more. This corresponds, to a certain degree, to the fact that the script (and therefore its broken down elements) is a strong and important asset in communication between project members.

The use cases shown in figure 5.2 pertain to the breakdown subsystem of the

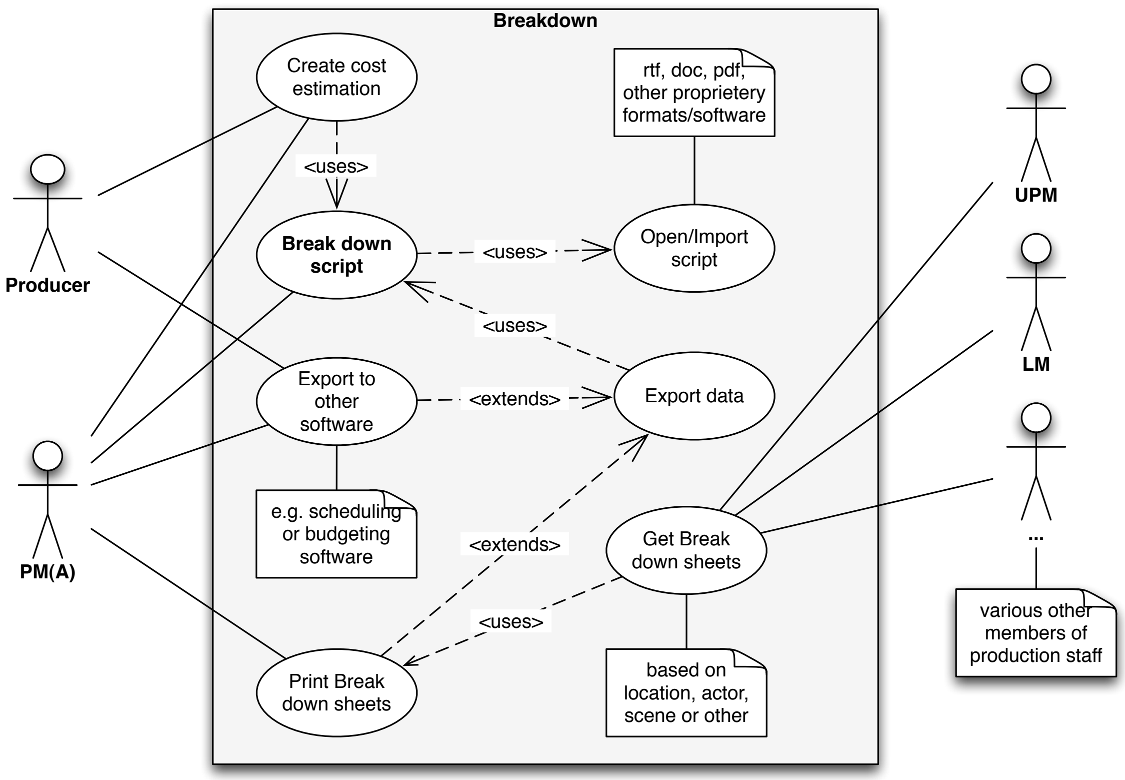


Figure 5.1: General use cases of the breakdown application.

software. It includes the tagging of elements, as well as category management and the ability to navigate and/or filter by scene, which is also required for scene-local tagging of elements.

The `Tag elements` use case also acts as a basis and common descriptor of a variety of tagging interfaces, such as `Shortcut tagging`, `Button tagging` and `Mouse tagging`. It also defines that tagging interfaces can incorporate pre-selection heuristics.

5.2 Tagging interfaces

For easier reference, the act of identifying relevant elements of a scene and assigning them to a breakdown category is called tagging (sometimes also referred to as highlighting or marking). In order to present such tagging functionality to the user, there are numerous possible graphical interfaces that can be employed, some of which were already shown in 4.1.

Table 5.1 presents the main types of tagging interface identified from the obser-

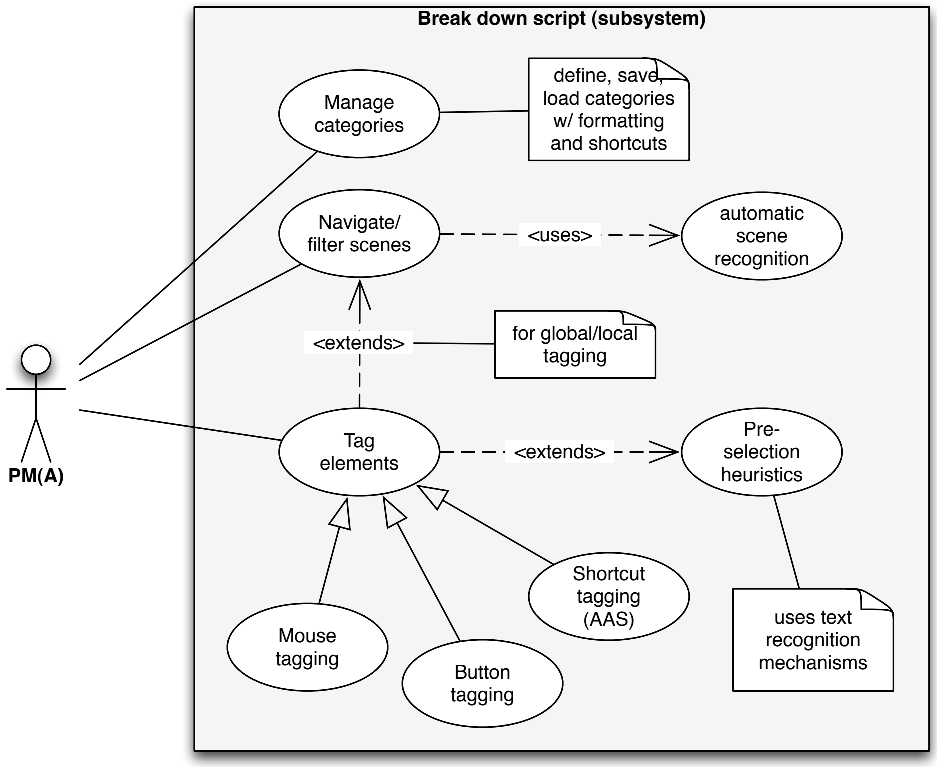


Figure 5.2: Specific use cases of the script breakdown subsystem of the breakdown application.

vations in 4.1. It explains the typical tagging sequences and gives an estimation on their respective complexity (lower values meaning less complexity). The complexity estimation used in table 5.1 takes into consideration the number of actions required from the user to tag one element as well as particular circumstances that greatly affect user experience and performance. Examples include missing script context or the necessity to switch between contexts (for instance from script view to a separate dialog window), as well as other complex actions like manually filling in text fields.

One factor that all these methods have in common is that they require the user to select or enter the element text they want to tag. While this does not pose problems concerning intuitiveness of the interface, the act of repeatedly highlighting text using the mouse or keyboard can become very tedious and error-prone, especially when the script is broken down for the first time and therefore worked through sequentially in its entirety.

The discrepancy between time and effort required for selecting the text and the actual tagging becomes especially evident in the case of shortkey tagging, where

Type	Sequence	Complexity	Remarks
Manual tagging 4.1	type or select text, select category, confirm	3-4	standard interface, can provide options for local/global tagging etc., little context
Manual tagging with dialog 4.2 4.3a 4.4 4.5	type or select text and open dialog, select category, confirm	4-5	as above, but less context
Add-to-category button 4.3a	select category, select text, press button	3	very tedious in initial breakdown, more suitable for corrections
Context menu tagging 4.3a 4.5	select text, open context menu, confirm category	2	good context, intuitive for many user groups
Drag and drop tagging 4.3a	select text, drag onto category	2	good context, intuitive for many user groups
Category shortkeys 4.5	select text, press shortkey	2	good context, fast but requires memorizing of keys
Direct manipulation of BDS 4.3b 4.4	start editing, type text, confirm	2-3	provides less context than script interaction, only scene-local changes

Table 5.1: Classification of tagging interfaces by action sequence and complexity.

the tagging action takes just a fraction of the time necessary to select the text. This observation also recommends the usage of shortkey tagging in combination with assisted text selecting in order to create a specialized, more effective tagging interface for script breakdown.

5.3 Auto-advancing shortkey (AAS) tagging

This proposed tagging method aims at providing a streamlined tagging interface using shortkeys for tagging, and requiring no mouse interaction at all. It is based on the assumption that the user knows about breakdown categories and has memorized their corresponding shortcut keys.

To support this, flexible category management including free assignment of shortcut keys is required. It was deemed recommendable to define ranges of keys usable for categories and to pre-assign control keys for navigation and manual text selection, similar to the scheme in table 5.2.

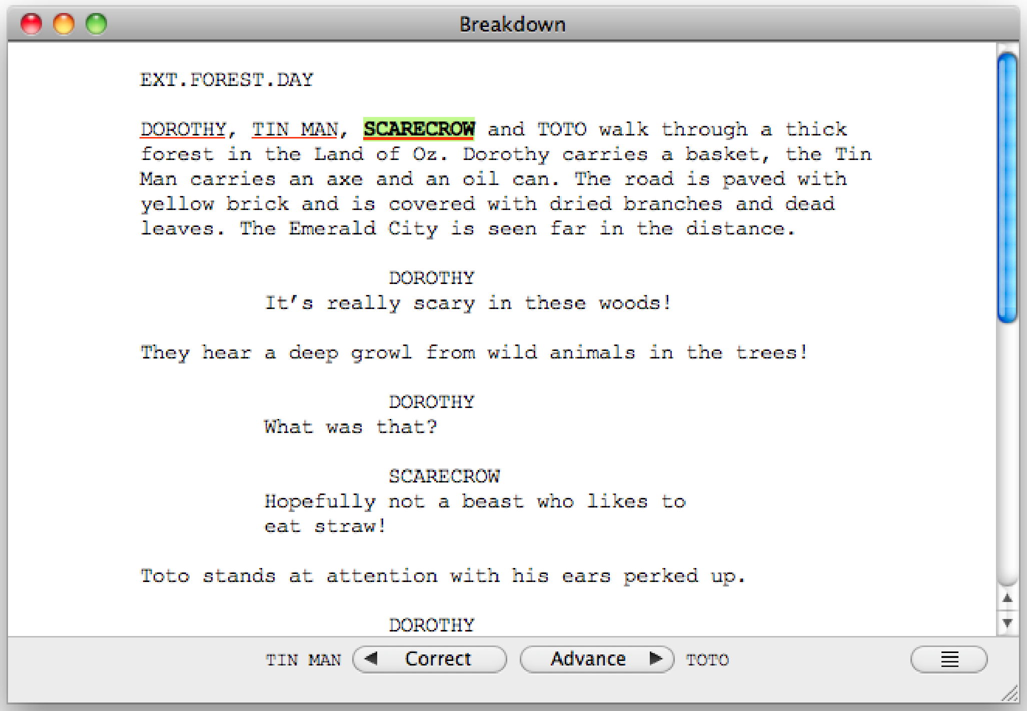
Keys	**Usage**
`a-z`, `A-Z`	shortcut keys, tag selected text and advance (freely assignable, for instance `c` for cast and `p` for props; case-sensitivity optional)
`<left>`, `<right>`	go to previous or advance to next word or word group
`<shift+left>`, `<shift+right>`	add previous/next word to selection (such as for multiword-elements)
`<up>`, `<down>`	jump one line up or down
`<backspace>`, `<delete>`	un-tag text
`<space>`, `<tab>`	un-tag text and advance

Table 5.2: AAS shortcut assignment schema for categories and control keys.

As discussed previously, the act of repeatedly selecting text for tagging is tedious work for the user, especially when the script is being broken down for the first time, from start to finish, in sequential order. AAS tagging mitigates this negative effect by auto-advancing after an element has been tagged, and pre-selecting the range of text that is likely to be tagged next. Using AAS does, however, not limit the application to this tagging interface in any way. Providing additional, mouse-controlled tagging mechanisms to complement it, for example for punctual corrections or similar, is encouraged.

AAS benefits strongly from the smartness of the pre-selection heuristics, which can be improved by implementing general text recognition and language processing mechanisms (such word boundary finding, skipping of stop words or looking up lexical databases for word types, as discussed in 3.3), as well as application and context-sensitive features like the pre-processing of sluglines, recognizing compound tokens and skipping tagged elements.

Figure 5.3 shows the first mock-up of a possible script breakdown application prototype using AAS tagging. The mock-up shows a small excerpt of the script for a screenplay of Frank L. Baum's *The Wizard of Oz* (Langley et al., 1939).



```
EXT.FOREST.DAY

DOROTHY, TIN MAN, SCARECROW and TOTO walk through a thick
forest in the Land of Oz. Dorothy carries a basket, the Tin
Man carries an axe and an oil can. The road is paved with
yellow brick and is covered with dried branches and dead
leaves. The Emerald City is seen far in the distance.

                    DOROTHY
          It's really scary in these woods!

They hear a deep growl from wild animals in the trees!

                    DOROTHY
          What was that?

                    SCARECROW
          Hopefully not a beast who likes to
          eat straw!

Toto stands at attention with his ears perked up.

                    DOROTHY
```

Figure 5.3: First mock-up screenshot of a script breakdown interface using AAS tagging.

The mock-up in figure 5.3 shows the state in which the elements `DOROTHY` and `TIN MAN` have already been tagged (and are therefore rendered underlined) and

`SCARECROW` is currently being tagged. At this point, pre-processing will already have recognized the scene's slugline `EXT.FOREST.DAY` and the pre-selection heuristic has determined the pre-selection ranges of text to the left and right of the currently selected text, as reflected by the two text labels at the bottom of the screen.

Note that even though `TIN MAN` consists of two words, it will have been recognized as a previously tagged compound token. Moreover, the word `and` to the right of the currently selected text will have been considered a stop-word resulting in it being skipped and `TOTO` being shown in the preview label.

5.4 Category management

The simple and flexible management of breakdown categories is essential in order to be able to accommodate for the needs of differing types of scripts and productions, but often neglected by the software applications observed in section 4.1. With the introduction of AAS, it is even more important due to the need to easily and freely assign shortcut keys to categories. The minimum set of features required includes the following:

- add, remove and rename categories
- change the highlighting format of categories (color, underlining) and automatically reflect changes in the script view; provide a formatting preview
- assign/remove shortcut keys to/from categories
- remove assigned elements from categories and automatically reflect changes in the script view

Additionally, means to save and load category presets (including formats and assigned keys) are suggestible and recommended for real-world use of a script breakdown software.

5.5 Scene and scope management

Scenes should be recognized as per their sluglines in the pre-processing phase when the opened script is first analyzed by the application. Scene management itself should be limited to controlling the tagging scope, i.e., limiting tagging actions to certain or all scenes. Changing names or other attributes of scenes is not desired in the script breakdown phase.

Tagging elements globally prevents the user from having to re-tag recurring elements in all scenes they appear in, which means greatly reduced effort in the case of the main actor, for example.

Breakdown sheets usually also incorporate additional information about scenes, including scene length in 1/8s (as discussed in section 2), estimations on required numbers of shooting days and setups, as well as further production notes. Where possible, this data should be calculated automatically (for instance, scene lengths can be derived from the number of lines they take), else the PM should be provided with an interface to add such information.

5.6 Breakdown data management

Collected breakdown data is used for various communication and scheduling purposes throughout production, thus printing and exporting functions are essential.

For data export, and more generally data transfer, XML (Extensible Markup Language, World Wide Web Consortium (W3C) (2010b)) provides a platform-independent, universally known and supported document format for data exchange. An equally well accepted file format for transferring printable documents is the aforementioned PDF.

5.7 Conclusion

AAS tries to improve the script breakdown process by emphasizing streamlined tagging actions and keeping the user's attention on the tagged elements and their context within the script. Flexible category management and unobtrusive scene handling, as well as means to transfer and print breakdown data in suitable formats, support the efficient application of AAS.

6. Results

This chapter will first provide an insight into the conception, design and implementation of the prototype software developed for this thesis, based on the observations and results of the previous chapters. It will also account for problems encountered during the work on the prototype.

The second part of this chapter is dedicated to presenting the methodology, execution and analysis of the usability evaluation that was conducted with the goal of assessing the acceptance and efficiency of various tagging interfaces provided by the prototype.

6.1 Prototype Application

The main concepts described in chapter 2.3 have been implemented into a prototype application. The following chapters will explain how the prototype itself was designed and developed in accordance with established requirements and provide an account of problems, solutions and lessons learned in the process.

6.1.1 Target system

Throughout the research conducted for this thesis, a number of goals and formal requirements were noted and established. These included:

- Focus on usability in terms of issues discussed in sections 3.4 and 4.1.
- Streamline the process from script breakdown to the first cost estimation, especially by the smart tagging of elements and semi-automatic management of scenes and categories.

- Ensure re-usability of data by providing data exports in a universally usable format. Moreover, support printing of breakdown data, for instance breakdown sheets for scenes, actors and locations.

In addition to these demands, the following technical requirements were added to the specification:

- The target operating system is Mac OS X 10.5 and higher.
- Interfaces must be designed and implemented to be suitable for resolutions of 1280x800 pixels and higher.
- Tagging must be efficient enough to be capable of global-tagging multiword elements in reasonably long scripts (`>10,000 words`) in a timely fashion (`<1 second`).
- Saving/loading of project files (including script with formattings, elements, categories and scenes) and category configurations must be supported.

6.1.2 Software architecture and design

The prototype's architecture was designed around three principles of software engineering: object-orientation, modularity and the model-view-controller design pattern.

Object-oriented programming (OOP) is a software engineering paradigm that abstracts the real-world context of a software into a model of objects which are instances of classes representing said real-world entities. Booch et al. (2007) define OOP as

> *"[. . .] a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships."*

The prototype application will, in accordance with OOP principles, create an abstraction model around classes that represent real-world entities, such as the script, breakdown categories, elements, scenes and more.

Modularity means that the separation of concerns (one of the main OOP principles) is ensured throughout the whole scope of the software design in this context. Classes that are logically related and adhesive are coupled within modules, and communication interfaces between self-contained modules are defined.

Model-view-controller as described in chapter 3.2, is an architectural software design pattern dividing the implementation in three separate layers of model, view and controller. As the native application programming interfaces (API) on current Apple Mac OSX systems are thoroughly MVC-oriented, the prototype's implementation can be built seamlessly onto this basis.

System architecture

In the context of this work, the aforementioned principles meant that the prototype's system architecture was to be constructed of internally adhesive modules consisting of intrinsically tied controller and model classes. Initial analyses of real-world circumstances yielded a likely reasonable segmentation of responsibilities in a breakdown module, a printing and export module and, later on, a module that handles integration into calculation.

In addition to these modules, a text parsing module consisting of service classes for the breakdown module, such as a text tokenizer, and a module responsible for handling usability engineering interfaces and workflows were considered in the design phase as well.

Based on these responsibilities, modules were then analyzed more thoroughly in their context, in order to find a suitable data model for the representation of real-world entities and interfaces between the controllers and the modules. Figure 6.1 shows how the finalized prototype-architecture was set up.

Breakdown The breakdown module provides and implements interfaces for category and element management, scene management and the script breakdown. Therefore, it includes two controllers pertaining to each of the views (document or script view, category view).

This module also defines the data model representing the real-world entities of

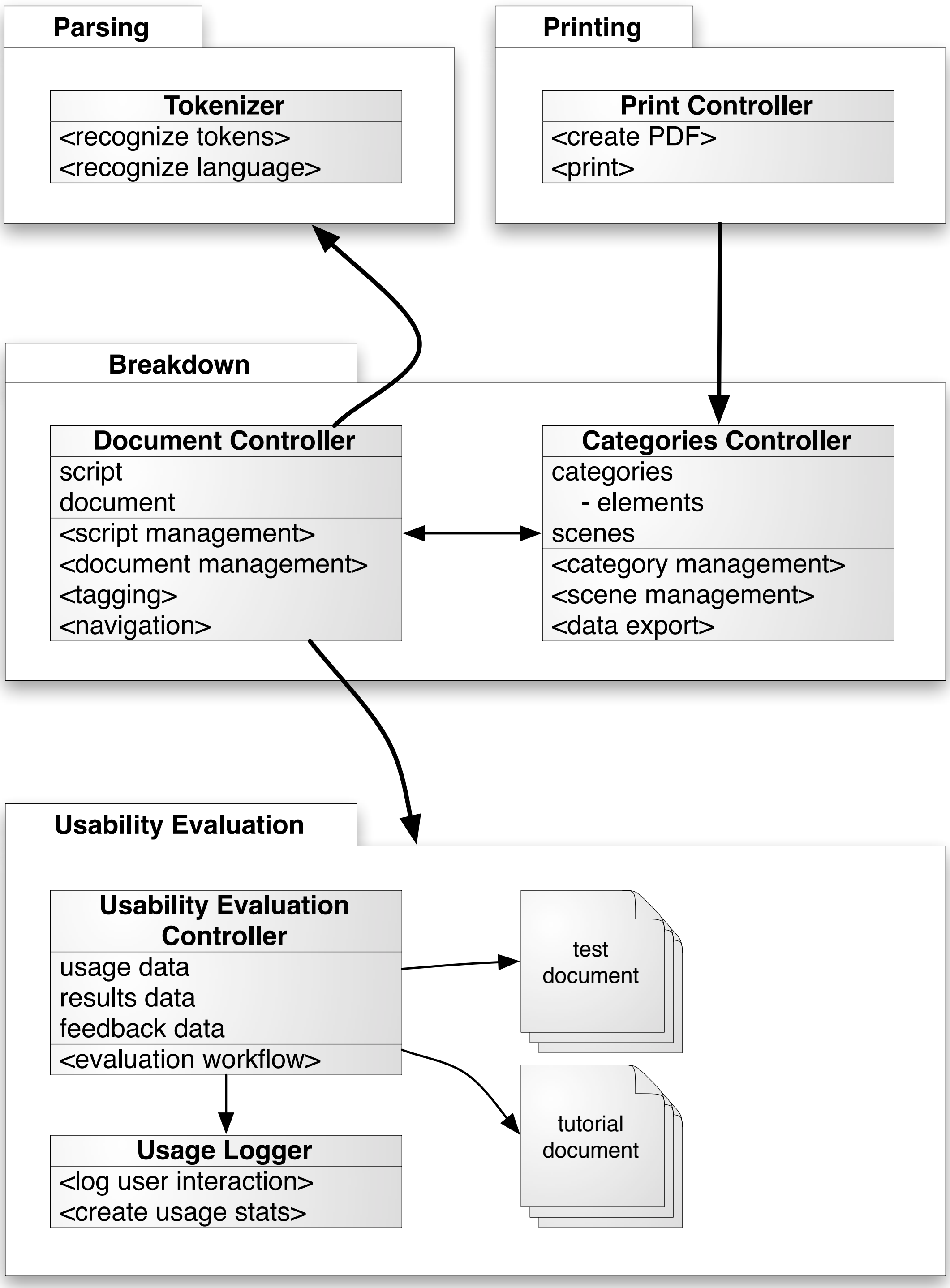


Figure 6.1: Simplified module/controller architecture of the breakdown prototype including responsibilities.

document (including the script), category, entry and scene. The controllers use and apply this data model to implement script breakdown.

The breakdown module also implements methods preparing the export of generated data into a structured, application-independent data format that can be used for tasks like printing or importing into other applications.

Parsing This module provides a multilanguage-compatible tokenizer capable of performing a wide range of atomic tasks concerning text recognition and parsing, for example word-boundary scanning or recognition of the most likely language of a given text.

This module is meant to be extended with more service routines and text recognition features as required by the main breakdown module.

Printing Printing implements methods for laying out and printing data provided by the main breakdown modules. Using a layout and template engine, this module renders given data to legible formats in order to create PDF report printouts from templates that are written in HTML (Hypertext Markup Language (World Wide Web Consortium (W3C), 2010c)).

The breakdown module is responsible for providing templates for script breakdown sheets per scene, location and actor. In future development, the integration module can provide templates for reports about created calculation entries or similar, but such reports are not relevant for this work and will therefore not be implemented.

Usability Evaluation This module provides mechanisms and interfaces required for usability evaluations. Most importantly, it implements methods to start and end the usability evaluation workflow, which includes showing and validating forms and feedback questionnaires and sending collected results to a receiving script that is hosted on a server reachable through a normal internet connection.

A second aspect of this module is the automatic collection of usage data and creation of simple usage statistics. This will be discussed thoroughly in section 6.2.

Data model

With the exception of the printing module, which is unaware of the exact structure of the data it processes and applies to the given templates, all controllers must have knowledge about the underlying data model. This is especially true for the breakdown module, as it directly applies its class hierarchies and interfaces to model and store script breakdown data.

Figure 6.2 depicts a diagram of the data model as defined by the breakdown module. `Category` and `Entry` share a common super-class from which they inherit

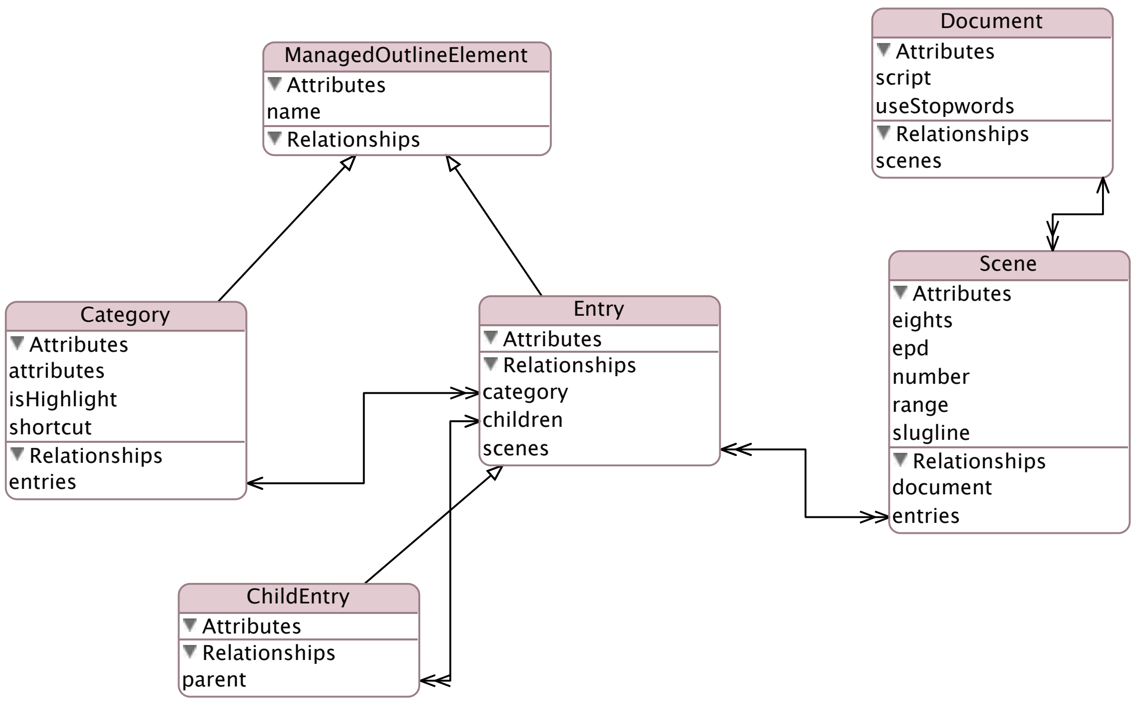


Figure 6.2: Data model of the breakdown prototype.

attributes and methods they both need to implement, for example a name property. Among other things, `Category` additionally administers a relationship to its elements (for instance `Entry` objects representing text added to this category) and on how they are formatted in the script view. `Entry`, on the other hand, knows which `Category` it belongs to, and in which `Scene`s its specific text is tagged.

In order to model a recursive, hierarchical relation between entries, `ChildEntry` serves as an intermediate model entity specifically representing entries that belong to a parent entry. This is used in the prototype to enable the user to define entries that are derived from others, and can not exist without them; for instance, animal trainers must be on set for shooting scenes with animals.

In the script pre-processing phase, objects of the `Scene` class are created for each recognized slugline in the script. Therefore, the `Document` encapsulates both the script as text, and the list of scenes found therein.

Further details about the employment of MVC in the design and implementation of this prototype, along with a discussion of results and lessons learned are presented by Holzinger et al. (2010).

6.1.3 Implementation

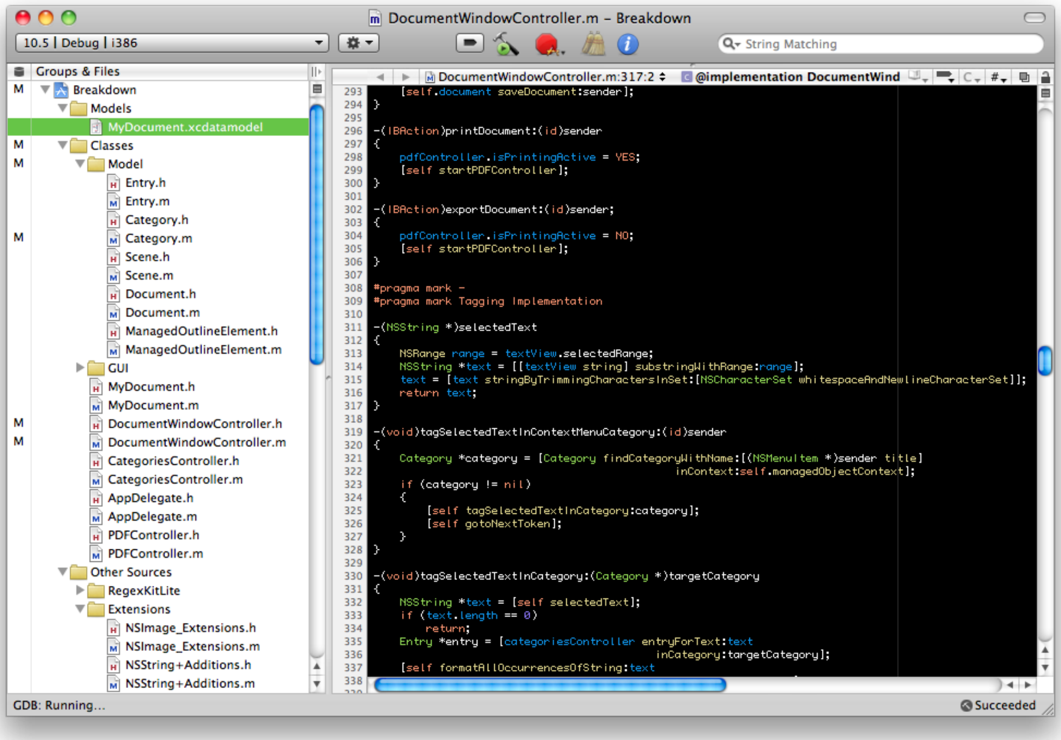
Tools and materials

The prototype was developed mainly on a 13.3" Apple MacBook (for details, see technical specifications in table 6.1) using Apple Xcode 3.2[1] as an integrated development environment (IDE). Of the extensive set of tools provided by Xcode, mainly the source code editor and debugger, the interface designer (called Interface Builder, see figure 6.4) and the data model editor (figure 6.2 is a screenshot of the tool's data model view) were used. Since platform-independence was not included in the prototype's requirements, all code has been written in Objective-C 2.0 using application programming interfaces (APIs) natively provided by Mac OSX and Xcode. Analogously, all user interfaces were therefore constructed of graphical user interface elements native to Mac OSX.

Hardware

The development system's specifications (see table 6.1) in terms of computing power and display size and resolution are ranked at the lower end of the specifications established in section 6.1.1. The prototype's performance running on the development system was therefore considered an indicator for whether the prototype meets the set requirements.

[1]`http://developer.apple.com/tools/xcode/` Xcode development environment, last access 08/2010



```
    [self.document saveDocument:sender];
}

-(IBAction)printDocument:(id)sender
{
    pdfController.isPrintingActive = YES;
    [self startPDFController];
}

-(IBAction)exportDocument:(id)sender;
{
    pdfController.isPrintingActive = NO;
    [self startPDFController];
}

#pragma mark -
#pragma mark Tagging Implementation

-(NSString *)selectedText
{
    NSRange range = textView.selectedRange;
    NSString *text = [[textView string] substringWithRange:range];
    text = [text stringByTrimmingCharactersInSet:[NSCharacterSet whitespaceAndNewlineCharacterSet]];
    return text;
}

-(void)tagSelectedTextInContextMenuCategory:(id)sender
{
    Category *category = [Category findCategoryWithName:[(NSMenuItem *)sender title]
                                              inContext:self.managedObjectContext];
    if (category != nil)
    {
        [self tagSelectedTextInCategory:category];
        [self gotoNextToken];
    }
}

-(void)tagSelectedTextInCategory:(Category *)targetCategory
{
    NSString *text = [self selectedText];
    if (text.length == 0)
        return;
    Entry *entry = [categoriesController entryForText:text
                                           inCategory:targetCategory];
    [self formatAllOccurrencesOfString:text
```

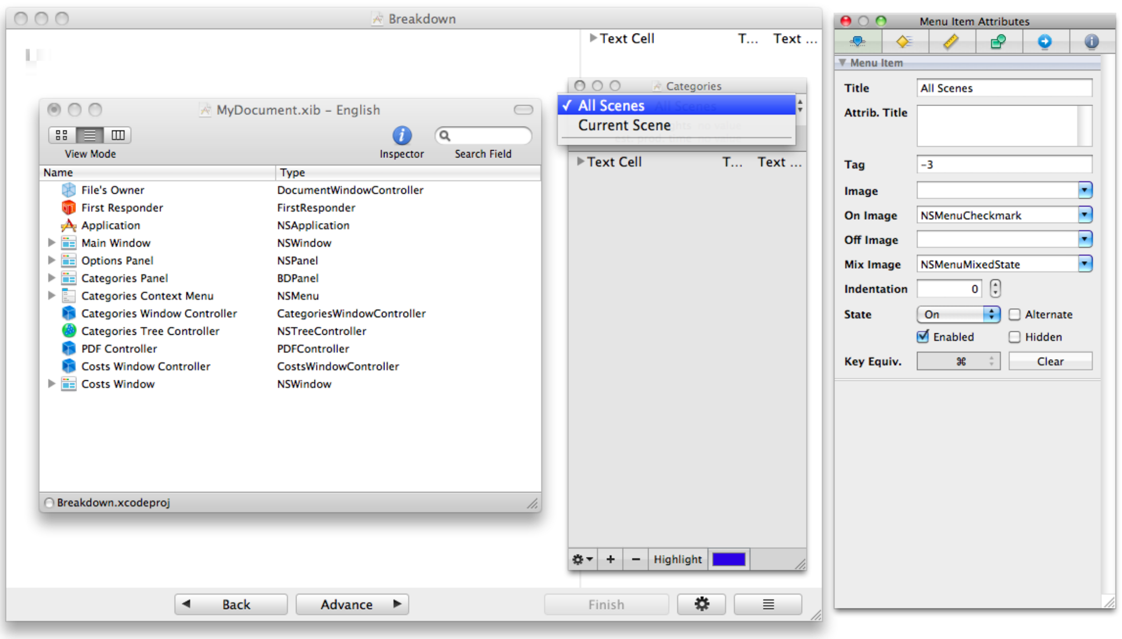
Figure 6.3: Main window of Apple Xcode 3.2 showing the project outline on the left and the code editing view.

Graphical user interfaces

Interface Builder was used to design the prototype's user interfaces and to set up connections (so called outlets) between the controller classes and the interface. The combination of views and controllers represents the implementation of the interactivity of the software. The user interfaces were created with general usability and interface design considerations in mind, including the usage of known controls and icons for known purposes, the grouping of elements that belong together, and similar.

Component	Specification
Operating system	Mac OS X 10.6.2
CPU	Intel Core 2 Duo T7300 2 Ghz
Memory	4 GB DDR2
Display	13.3" widescreen LCD
Native resolution	1280x800
Hard disk	120 GB 5400 RPM SATA

Table 6.1: Technical specifications of the development system.



Name	Type
File's Owner	DocumentWindowController
First Responder	FirstResponder
Application	NSApplication
Main Window	NSWindow
Options Panel	NSPanel
Categories Panel	BDPanel
Categories Context Menu	NSMenu
Categories Window Controller	CategoriesWindowController
Categories Tree Controller	NSTreeController
PDF Controller	PDFController
Costs Window Controller	CostsWindowController
Costs Window	NSWindow

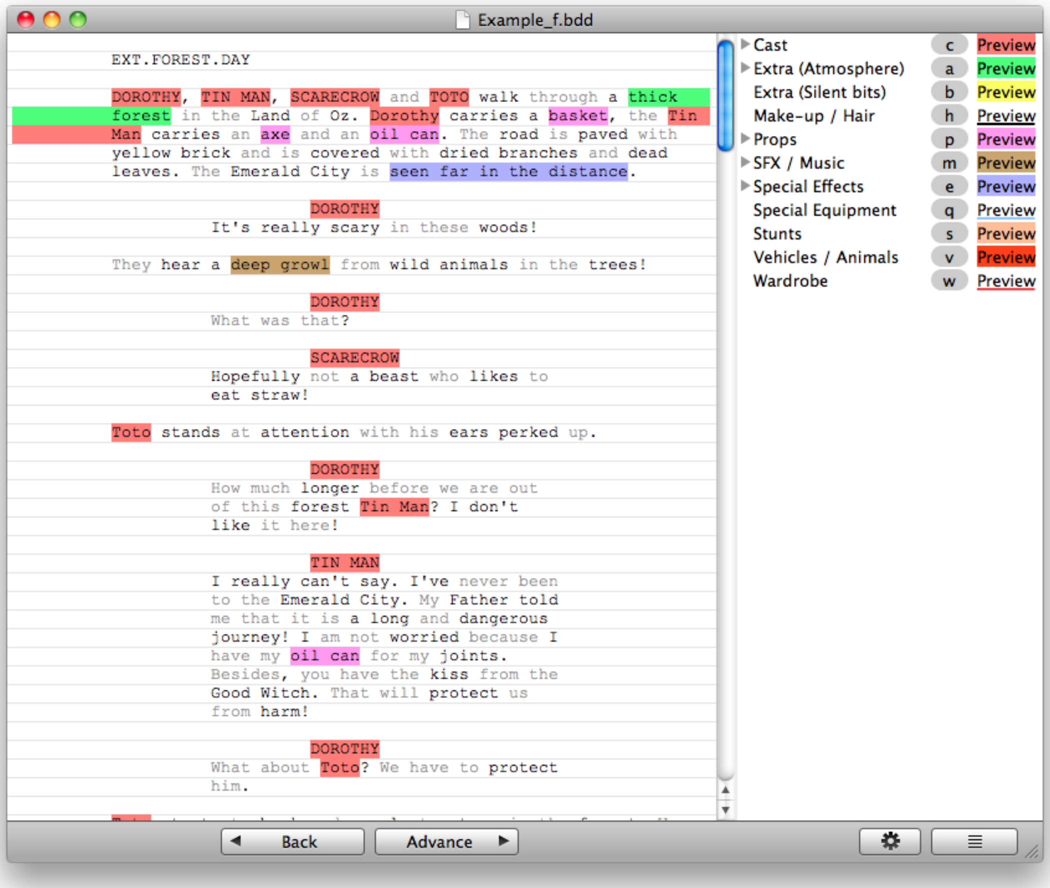
Title	All Scenes
Attrib. Title	
Tag	-3
Image	
On Image	NSMenuCheckmark
Off Image	
Mix Image	NSMenuMixedState
Indentation	0
State	On ☐ Alternate
	☑ Enabled ☐ Hidden
Key Equiv.	⌘ Clear

Figure 6.4: Designing the interface of the breakdown prototype with Interface Builder.

Breakdown module

Main window The breakdown module consists of two windows. The first one is the main window, including the script view and navigation/tag buttons and previews. In addition, there is a list view of all defined categories and entries. The second window is the category management panel which is dependent on, but detached from, the main window and can be shown on demand using a button in the main view. Figure 6.5 shows the main window of the prototype. This window mainly consists of the script view showing the current part of the script with highlighted or underlined tagged elements. Below the script view, the most important functions are available at the press of a button and grouped logically. Less frequently used functions are available from the application's main menu and from context menus to keep the interface unobtrusive.

The right hand side of the main view shows all defined categories, along with the entries already tagged into them. It also incorporates a shortkey indicator and a textual preview of the respective category's formatting, similar to the category management view, as will be described below.



```
          EXT.FOREST.DAY

          DOROTHY, TIN MAN, SCARECROW and TOTO walk through a thick
          forest in the Land of Oz. Dorothy carries a basket, the Tin
          Man carries an axe and an oil can. The road is paved with
          yellow brick and is covered with dried branches and dead
          leaves. The Emerald City is seen far in the distance.

                              DOROTHY
                    It's really scary in these woods!

          They hear a deep growl from wild animals in the trees!

                              DOROTHY
                    What was that?

                              SCARECROW
                    Hopefully not a beast who likes to
                    eat straw!

          Toto stands at attention with his ears perked up.

                              DOROTHY
                    How much longer before we are out
                    of this forest Tin Man? I don't
                    like it here!

                              TIN MAN
                    I really can't say. I've never been
                    to the Emerald City. My Father told
                    me that it is a long and dangerous
                    journey! I am not worried because I
                    have my oil can for my joints.
                    Besides, you have the kiss from the
                    Good Witch. That will protect us
                    from harm!

                              DOROTHY
                    What about Toto? We have to protect
                    him.
```

Cast	c	Preview
Extra (Atmosphere)	a	Preview
Extra (Silent bits)	b	Preview
Make-up / Hair	h	Preview
Props	p	Preview
SFX / Music	m	Preview
Special Effects	e	Preview
Special Equipment	q	Preview
Stunts	s	Preview
Vehicles / Animals	v	Preview
Wardrobe	w	Preview

Figure 6.5: Main window of the breakdown prototype application.

Category management view Breakdown categories, elements and scenes are managed in this window, which is toggled by a button in the main view. Figure 6.6 shows its layout. At the top of this view, a scene selection menu (which is generated automatically from the scenes recognized in the script while pre-processing) allows global tagging (`All Scenes`), local context-sensitive tagging (`Current Scene`) and scene-restricted tagging by clicking on a scene name.

Selecting a particular scene also scrolls the script view in the main window and positions the cursor to the start of the scene. Additionally, AAS pre-selection is restricted to the selected scene, thus it does not traverse to the next scene but wraps around to the current scene's start.

The scene selection panel also displays the length (in eighths) of the currently selected scene or scenes, as well as an estimation of the production time required to shoot the scenes.

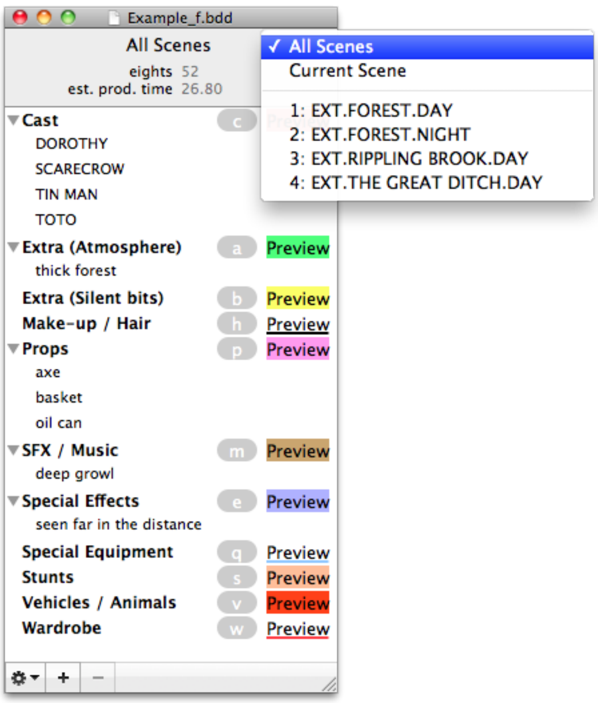


Figure 6.6: Category management window of the breakdown prototype application.

Below the scene selection, an outline showing the configured breakdown categories (including assigned hotkeys and a preview of their elements' formatting in the script view) is presented. Any tagged elements are listed directly within the groups they belong to. This outline view can be manipulated using the buttons at the bottom of the window.

The category configuration (again including hotkeys and formatting) can be stored into a file and loaded later on, allowing to the definition and reusal of default templates for certain production types.

Script pre-processing The prototype supports the pasting of text directly into the script view. Once text has been pasted, the script view is immediately write-protected and pre-preocessing is started, scanning the script for scenes.

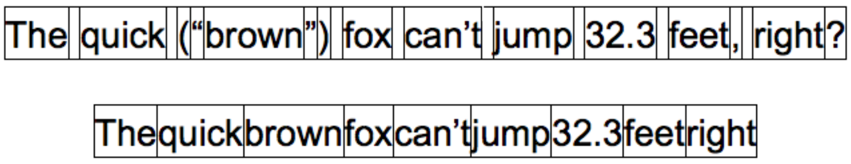
As previously noted, scene beginnings are indicated by sluglines which commonly follow a general pattern. For the purpose of efficient and flexible scanning for such sluglines, the following basic regular expression is used:

```
(?i)(EXT|INT).*(DAY|NIGHT)
```

Using this regular expression, the pre-processing algorithm searches case-insensitively (indicated by `?i`) for any text enclosed by `EXT` or `INT` on the left and `DAY` or `NIGHT` on the right. Every found scene is stored along with its slugline and the range (location and length) of the scene within the script.

This is advantageous compared to normal text-searching because it is far more efficient to compute, and enables the software to easily incorporate options set by the user, for example for making the operation case-sensitive or adding daytime-elements like `AFTERNOON` via a list interface.

Word boundaries scanning In section 5.3, it was stated that AAS requires robust and smart pre-selection heuristics to work properly and be beneficial for the user. The main responsibility of these heuristics is finding word boundaries. Figure 6.7 shows the segmentation of an English sentence by its words (or tokens).



The quick (“brown”) fox can’t jump 32.3 feet, right?
Thequickbrownfoxcan’tjump32.3feetright

Figure 6.7: Segmentation of an English sentence by word boundaries. Original sentence at the top, extracted version at the bottom (Davis, 2009).

There are several trivial and non-trivial solutions for this, including manually implementing a text segmentation by a defined set of delimiter-characters (such as space, tabulator, line break, punctuation marks, quotation marks), but they suffer from drawbacks concerning computing efficiency and flexibility. What is more, the solution implemented should be able not only to work with English or German scripts, but also with text written in other languages, which sometimes use very different means of separating words, or do not separate them at all. Davis (2009) provides an impression of how complex and diverse demands for language-independent word boundary finding are.

Based on these observations, using an existing library, be it an external suite or integrated into the Mac OSX API, was found to be the most suitable solution. After thorough research on this topic, a class of the Mac OSX Core Foundation library[2]

[2]`http://developer.apple.com/corefoundation/` Apple Core Foundation documentation,

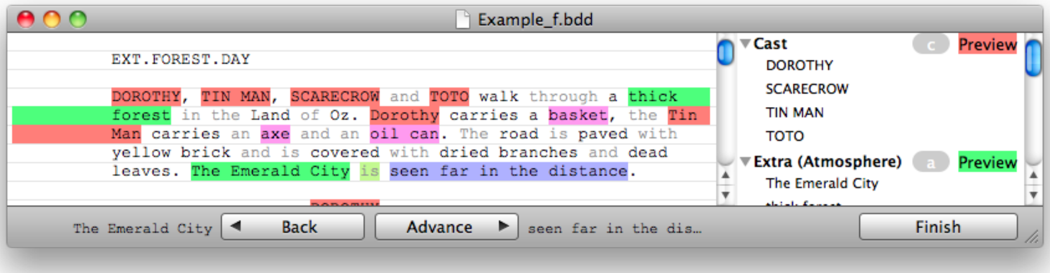
for interlingual string management was found and incorporated into the prototype due to recommendations by both independent developers and Apple.

This class, called `CFStringTokenizer`, supports not only the tokenizing of strings, which means segmenting text by its atomic words, but also recognizing types of tokens, de-compounding of German compound words and finding Latin transcriptions for tokens. It can also identify the language a given text was most likely written in, which can be used to great effect for incorporating language-specific processing like stop word-filtering and the use of syntactical databases.

In order to make use of the low-level `CFStringTokenizer` implementation, a string tokenizer class was created encapsulating the management of a `CFStringTokenizerReference` and providing an efficient interface for the breakdown controller classes.

Pre-selection heuristics By using the aforementioned text tokenization module, a set of methods for navigating by token and word boundaries was implemented. These methods are used by the pre-selection heuristics to determine which word or word groups to select after tagging via mouse and in accordance with the shortcut-key assignment schema defined in section 5.3.

Additionally, a number of extensions to token-based pre-selection have been implemented. One of them is **compound token**-based pre-selection, which means that groups of words that have already been tagged together as a breakdown element are pre-selected as a whole. In figure 6.8, the word `is` is currently selected. To


```
EXT.FOREST.DAY

DOROTHY, TIN MAN, SCARECROW and TOTO walk through a thick
forest in the Land of Oz. Dorothy carries a basket, the Tin
Man carries an axe and an oil can. The road is paved with
yellow brick and is covered with dried branches and dead
leaves. The Emerald City is seen far in the distance.
```

Figure 6.8: Compound token-based pre-selection of text.

last access 10/2010. Core Foundation is a low-level framework implementing base functionality which high-level classes of the Mac OSX Cocoa API, such as interface elements and controllers, are built upon.

its left, a tagged compound token `The Emerald City` has already been identified. This is reflected at the bottom of the breakdown view, where the pre-selection preview shows `The Emerald City` as the text that will be selected if the user navigates left.

This functionality is implemented in one atomic method that is used by both the pre-selection heuristics to determine the range of text to be selected, and the tagging mechanism to check whether a given occurrence of the tagged text is part of an already tagged *super*-tag. A simplified illustration of the underlying algorithm in pseudo-code is shown in algorithm 1.

Algorithm 1 Pattern-matching of supertags for pre-selection heuristics and smart tagging.

```
 1 $t \leftarrow selectedText()$
 2 $s \leftarrow supertags(t)$
 3 for all $s$ ordered by $s.name.length$ descending do
 4   for all occurrences of $t$ in $s.name$ do
 5     if pattern $p$ of $s.name$ around $t$ in script matches then
 6       return $p.range$
 7     end if
 8   end for
 9 end for
10 return $NotFoundRange$
```

While this algorithm is implemented with two cascading for-loops, it has to be considered that both loops are traversed very infrequently. The outer loop is only passed through for every tagged element that has the selected text in it (like for `emerald` as part of `The Emerald City`). The inner loop is then passed through for every occurrence of the selected text in the super-tag, which is highly unlikely to be more than one or two times, but has to be considered.

An alternative implementation might store the range of every occurrence of a tagged element in the data model and use this information instead of pattern-matching in the script. While this might improve performance slightly with very little additional memory cost, it renders the system less flexible to changes of text and tagging, and requires more frequent updates to the data model.

Due to the fact that this method presents a central point for checking compound token ranges and is invoked by both the pre-selection and the tagging algorithms, it can be used to easily employ additional language processing. This has already been done in the prototype to incorporate language-dependent stop word-filtering, for example.

Tagging algorithm Tagging itself builds upon the token- and compound token-based interfaces provided by the string tokenizer and the pre-selection heuristics to find and tag occurrences of given text in particular categories. Therefore, it is a connector and controller between the category module, the data model and the string based mechanisms.

The process of tagging via shortcut key representing a given category can be illustrated in simplified pseudo-code as shown in algorithm 2. Since the script is

Algorithm 2 Tagging algorithm using supertag pattern-matching and compound token ranges.

```
1  $t \leftarrow selectedText()$
2  $e \leftarrow newEntry(t)$
3  $c \leftarrow category(shortkey)$
4  for all occurrences $o$ of $t$ in script do
5    $r \leftarrow compoundTokenRange(o)$
6    if $supertags(r) == NotFoundRange$ then
7      $merge(r.format, c.format)$
8      $s \leftarrow sceneForRange(r)$
9      $e.scenes.add(s)$
10   end if
11 end for
```

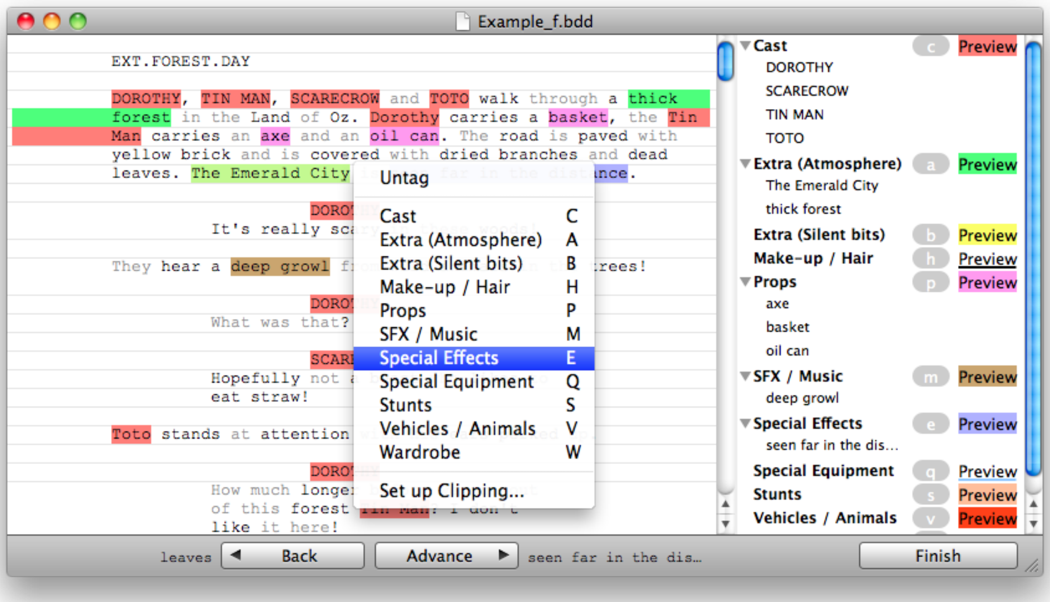
scanned for `t` using text-based methods, the algorithm has to expand all found occurrences to whole words, that is, their compound token range, by using methods that are aware of word boundaries. This range is defined as the range from the start of the token enclosing the start of the occurrence-range, to the end of the token enclosing the end of the occurrence-range.

For example, assume the user selected `e Emerald Ci` and an occurrence of this exact text was found in the script due to the existence of the phrase `The Emerald City`. In this case, the token enclosing the start of the range would be `The` and its starting location would be the index of the character `T`. Analogously, the token enclosing the end of the range would be `City` and its last index the position of the character `y` in the text. Therefore, the resulting compound token range would span the complete string `The Emerald City`.

For every occurrence found that is not masked by a supertag (which is checked using algorithm 1), the corresponding text in the script is formatted accordingly, merging the category's layout set by the user with existing formatting. Moreover, the scene enclosing the occurrence is stored in the entry's bi-directional scene relation.

Other tagging interfaces To complement AAS tagging, three other means of tagging (as discussed in section 5.2) were implemented:

- Drag-and-drop tagging
- Manual tagging by double-clicking on the desired category
- Context menu-based tagging both in the script view, as shown in figure 6.9, and directly in the category outline view.



```
EXT.FOREST.DAY

DOROTHY, TIN MAN, SCARECROW and TOTO walk through a thick
forest in the Land of Oz. Dorothy carries a basket, the Tin
Man carries an axe and an oil can. The road is paved with
yellow brick and is covered with dried branches and dead
leaves. The Emerald City                              nce.

                         DORO
          It's really sc

They hear a deep growl f                        trees!

                         DORO
          What was that?

                         SCAR
          Hopefully not
          eat straw!

Toto stands at attention

                         DORO
          How much longe
          of this forest
          like it here!
```

Figure 6.9: Context menu tagging in the prototype's main window.

The context menu contains all available categories ordered alphabetically by their name, and lists them along with their shortcut-key. Note that the `Untag` menu item is only shown if precisely one tagged element is selected in the text in order to reduce user confusion. The supertag pattern-matching method was used for this check.

It is also to note that both the tag-button and the context menu incorporate auto-advancing, therefore tagging using these interfaces also triggers pre-selection of the next suitable range of text.

Stop word filtering In order to improve pre-selection heuristics, stop word filtering was implemented to mark and skip elements of natural language that are, in most cases, of no significance to the factual content of the script and therefore its broken down elements.

Using the method of language detection mentioned in 6.1.3, the most likely language of the loaded script is determined in the pre-processing phase and stop word filtering is activated if an appropriate stop word list for the language is found. Basic stop word lists for English and German are included in the prototype. The lists are provided as XML/Plist compatible text files which are loaded by the document controller and applied on demand by the pre-selection heuristics.

The English stop word list was derived and adapted from the built-in list of stop words for the English language used by the popular database software MySQL (Oracle Corporation, 2010). The list of German stop words was compiled from various sources, including guidelines for optimizing the performance of search algorithms and search engine optimization (SEO), and adapted for use in this context.

The prototype currently also provides the option to enable or disable stop word filtering. If it is activated by the software and enabled by the user, all found stop words are indicated in the script view by rendering them in a lighter gray text color, so as to help the user anticipate and intuitively understand when and why certain words are skipped due to them being stop words. Figure 6.10 illustrates the effect of this visualization.

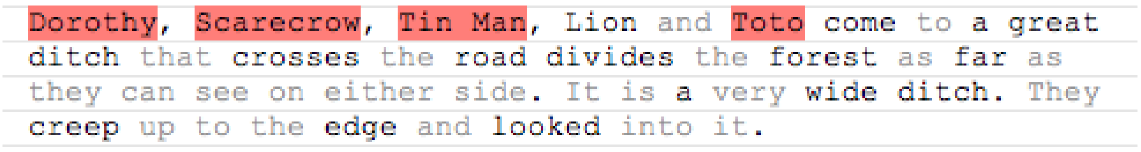

Dorothy, Scarecrow, Tin Man, Lion and Toto come to a great ditch that crosses the road divides the forest as far as they can see on either side. It is a very wide ditch. They creep up to the edge and looked into it.

Figure 6.10: Preview of stop words in the breakdown script view. Stop words are rendered in gray text color.

Data export Exporting breakdown data via XML as a data transaction file format is implemented using means of the Mac OSX API. First, all required data (scenes, categories, elements) is retrieved using queries on the underlying data model or using existing relations of the document-entity and the model object hierarchy. Afterwards, an `NSDictionary`, a dictionary-like data structure containing keys and values (which can be objects of any kind, including arrays of data and other, nested dictionaries), is created and continuously filled with breakdown data.

Depending on the purpose and destination of the export data, the dictionary may be written to an XML-compatible file and stored using safe file management functions of the API, or for example, for printing, directly forwarded to other software components and modules. Listing 6.1 shows a short excerpt of an example XML file resulting from writing an export dictionary to file. Note that in this case,

```
...
<key>scenes</key>
<array>
  <dict>
    <key>categories</key>
    <array>
      <dict>
        <key>name</key>
        <string>Actors</string>
        <key>entries</key>
        <array>
          <string>DOROTHY</string>
          <string>SCARECROW</string>
          <string>TIN MAN</string>
        </array>
        ...
      </dict>
      ...
    </array>
    <key>number</key>
    <string>1</string>
    <key>slugline</key>
    <string>EXT.FOREST.DAY</string>
  </dict>
</array>
...
```

Listing 6.1: Shortened example of an XML-compatible breakdown data export using PLIST-syntax.

the data is grouped by scene and then by categories, in order to simplify data access for the generation of breakdown sheets, which itself groups data by scenes. Other ways of structuring, for instance a completely flat list of all data objects, are possible and easily implemented using this method.

Printing module

The printing module accepts structured data (such as a breakdown data dictionary or XML file) and applies a layout template to it to create printable PDF data. A layout template is comprised of three components:

- **A base template definition** stored in a property list file (`template.plist`). It contains layout information such as paper dimension and orientation, as well as information on how the corresponding data should be structured and interpreted in terms of sheets (collections of pages with similar layout) and boxes (layout elements that are used to compose pages).

- **HTML-template-files** following a predefined naming convention for each sheet-box-combination. The format is `<sheet>_<box>.html`, therefore `cover_footer.html` defines the layout of a footer box on the cover sheet. These files define the overall layout of the boxes using markers and filters provided by the underlying template engine (see below). Listing 6.2 is an excerpt of a possible HTML template for printing categories and their entries into an HTML table.

- **A style definition** expressed in CSS (Cascading Style Sheet, (World Wide Web Consortium (W3C), 2010a)) file. The styles defined in this file (`style.css`) are applied upon rendering the processed HTML code into PDF.

As noted, the printing module uses a text-based template engine which is able to evaluate special expressions within the HTML templates to create more sophisticated and flexible templates. The main purpose of such expressions is to define placeholders in the template which, upon processing, are replaced by their corresponding actual values from the data dictionary. See, for example, line 5 in listing 6.2, where `{{category.name}}` defines a placeholder for category names. The template engine also supports conditional expressions (line 12) and loops (line 3).

```
...
<tr>
{% for category in categories %}
  <td style="border:1px solid gray;">
    <h3>{{category.name}}</h3>
    <ul>
    {% for entryName in category.entries %}
      <li>{{entryName}}</li>
    {% /for %}
    </ul>
  </td>
  {% if category.index % 3 %}
  {% else %}
  </tr>
  <tr>
  {% /if %}
{% /for %}
</tr>
...
```

Listing 6.2: Shortened example of an HTML-based export template file using template engine markups to layout tables of breakdown categories and their respective elements.

The template engine implemented for the prototype is adapted from a template engine for Cocoa that is freely available and usable for commercial and other use, and was written by Gemmell (2008).

6.1.4 Conclusion

The prototype software presented in this chapter is a working proof of the concept and the design established in previous sections. It is implemented in accordance with guidelines for clean software design and architecture and with strong consideration of general usability recommendations. Required functionality, including the loading and automatic pre-processing of text, script breakdown, category management, export and printing, was implemented.

The prototype implements different mechanisms for tagging elements into categories, allowing users to mix and match mouse (drag and drop, context menus) and keyboard (AAS) input as preferred. While pre-selection heuristics were originally implemented for AAS, they can also be combined with traditional input interfaces.

Recommended topics for further research and development are the possible improvement of pre-selection heuristics by adopting more sophisticated text recognition techniques and direct integration with scheduling and budgeting solutions.

The following chapter presents the methodology, technical background and findings of a usability evaluation conducted in order to assess user acceptance and satisfaction towards the various tagging mechanisms.

6.2 Usability Evaluation

In order to assess whether the system and its various aspects satisfies usability requirements such as joy of use and efficiency, a thorough usability evaluation was conducted. This section will clearly present the methods and methodologies incorporated in the evaluation, describe how tests were set up and carried out, and provide and discuss results gathered from the analyses of various test runs.

6.2.1 Methodology

When deciding about the methodology, a number of factors were considered. First of all, due to the user-centered nature of the prototype and script breakdown being a rather atomic task, it was deemed suitable to employ usability testing with end users, rather than using inspection methods (as described in section 3.4). Furthermore, a focus was set on finding out whether certain tagging interfaces provided by the software were usable and deemed efficient by the test users. This led to several preliminary assumptions and decisions about both the test methodology and setup:

- User-centered usability testing
- Simple test protocol ("break down this script" or "tag the words in their respective categories" instead of task-lists) with automated workflow
- Unintrusive test-environment
- Gather usage-statistics and other context information in background
- Combination of direct and indirect methods

In order to fulfill these requirements and findings, a combination of various approaches was devised:

- **Remote testing**
  Several of the above factors benefit remote usability testing, like keeping the test protocol simple and enabling the user to work in a comfortable, natural environment.

- **Asynchronous testing**
  In combination with remote testing, asynchronous testing, as was established before, increases the possibility to acquire larger amounts of data (in terms of completed result sets) that are efficient to analyze and evaluate.

- **Usage data**
  Data about the user's interaction with the software (mouse-clicks, key presses) can be collected automatically, also allowing for various degrees of automated data analyzation and creation of usage statistics.

- **Results data**
  The data the user creates in the test procedure can be logged and compared, for example, to sample solutions in order to assess task completion and error rates.

- **Feedback data**
  The user's feedback about various aspects of the tested system provides meaningful information about subjective software quality. This can greatly improve the overall quality of findings of a usability evaluation while providing easily comparable usability measures on its own.

In summary, a remote/asynchronous, user-centered and self-conducted usability evaluation based on a simple test protocol was conducted. Besides general demographic data (age, sex, native language, education, among others), three types of results data were collected, analyzed and included in evaluation to cover a wide range of possible problem areas and aspects.

6.2.2 Test setup

In order to ensure problem-free user-conducted tests, the created prototype was extended in a number of ways. Some functions were made invisible or inaccessible in the usability prototype (UP), such as category management, pre-selection and stop word options and other features inappropriate for the test. On the other hand, a number of interfaces and forms for entering demographic data, giving ratings on the SUS scale as well as a tutorial project were added.

Test process

As discussed in chapter 2, the target end user group of a script breakdown software consists of film students and producers. From personal experience of talking with professionals working in this field, as well as a number of standard books on the topic, it was deemed safe to assume that such users at the very least look back on a few years of experience working with computers and have a solid understanding of office software and frequently of expert software for film calculation.

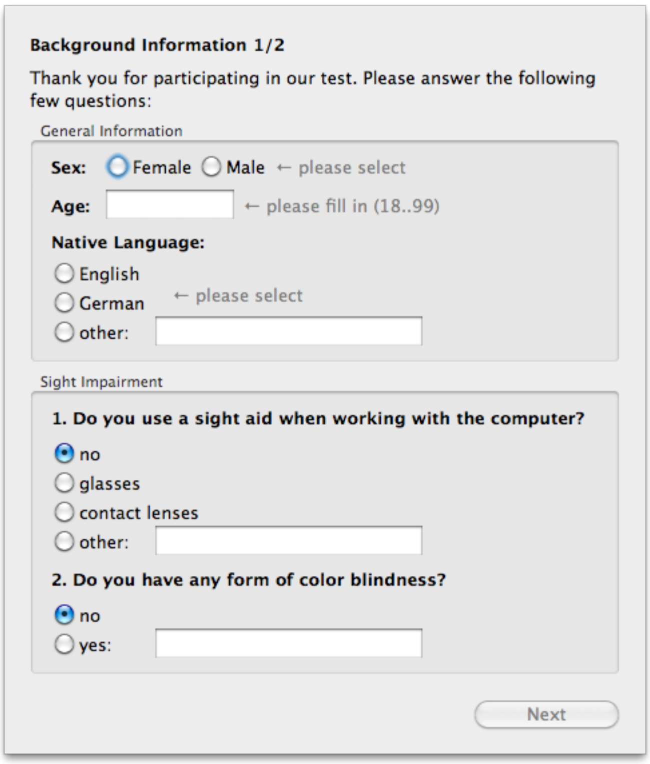
However, in order to ensure a large enough number of test results available, and to broaden the range of user types, the usability test was extended to test users without any film-specific educational or professional background. In the following considerations, these groups will be called *filmers* (F) and *non-filmers* (NF), respectively. To give an impression of how the tests worked and looked, a short walkthrough will now be provided.

1. **Background**
   The first dialog that is presented to the user is the first of two background information forms. These forms contain 4-5 questions each about the user's demographics, possible sight impairments, educational level and area as well as experience working with computers and operating systems.

   As shown in figure 6.11, these forms are implemented as application windows, rather than separate documents like PDF forms. This simplifies the test workflow for the user and at the same time allows for automated collecting of data.

2. **Tutorial**
   After filling in answers to the background questions, a tutorial document (as shown in figure 6.12) is opened, which was designed to give the user a quick introduction to the most important parts of the test.

   Two different tutorial documents were created to reflect the different knowledge backgrounds of the two groups. First, a short tutorial for filmers who are assumed to know about script breakdown, merely explaining the different ways to tag elements. Second, a more general tutorial for the group of non-filmers, explaining what tagging in the context of the test means and showing concisely the available ways to actually do tagging. Note that no time limit is defined


Background Information 1/2

Thank you for participating in our test. Please answer the following few questions:

General Information

Sex: ○ Female ○ Male ← please select

Age: ← please fill in (18..99)

Native Language:

○ English
○ German ← please select
○ other:

Sight Impairment

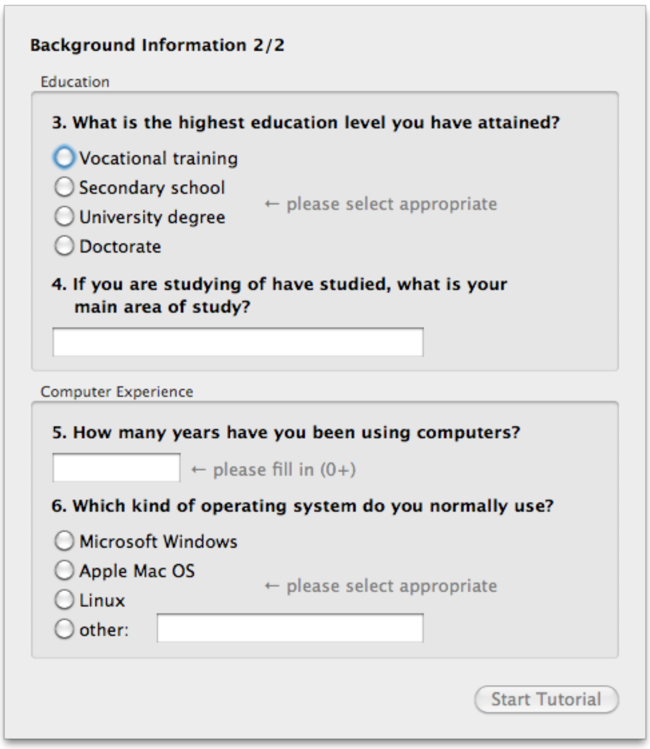
1. Do you use a sight aid when working with the computer?

[✓ no]
○ glasses
○ contact lenses
○ other:

2. Do you have any form of color blindness?

[✓ no]
○ yes:

(a) Background information form, page one


Background Information 2/2

Education

3. What is the highest education level you have attained?

○ Vocational training
○ Secondary school
○ University degree ← please select appropriate
○ Doctorate

4. If you are studying of have studied, what is your main area of study?

Computer Experience

5. How many years have you been using computers?

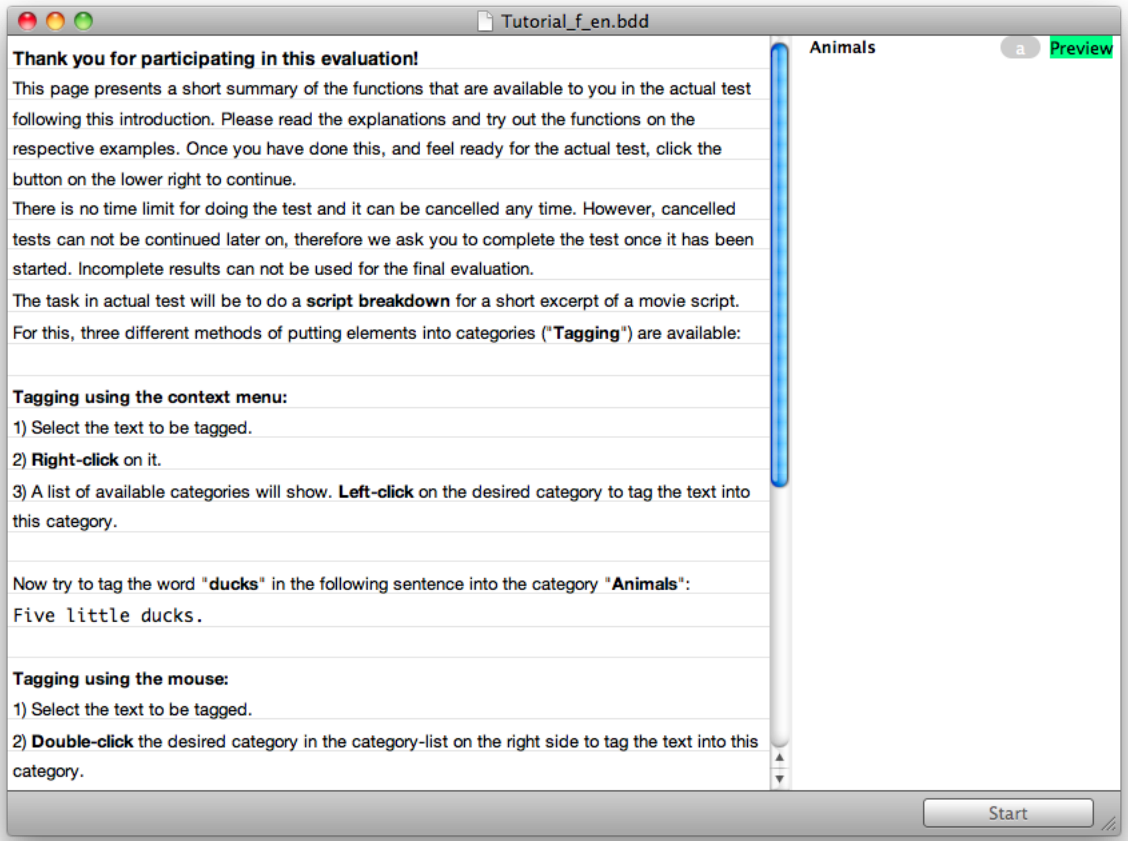
← please fill in (0+)

6. Which kind of operating system do you normally use?

○ Microsoft Windows
○ Apple Mac OS
○ Linux ← please select appropriate
○ other:

(b) Background information form, page two

Figure 6.11: Background information form, pages one (6.11a) and two (6.11b), as used in the usability evaluation.



Thank you for participating in this evaluation!

This page presents a short summary of the functions that are available to you in the actual test following this introduction. Please read the explanations and try out the functions on the respective examples. Once you have done this, and feel ready for the actual test, click the button on the lower right to continue.

There is no time limit for doing the test and it can be cancelled any time. However, cancelled tests can not be continued later on, therefore we ask you to complete the test once it has been started. Incomplete results can not be used for the final evaluation.

The task in actual test will be to do a **script breakdown** for a short excerpt of a movie script.

For this, three different methods of putting elements into categories ("**Tagging**") are available:

Tagging using the context menu:

1) Select the text to be tagged.

2) **Right-click** on it.

3) A list of available categories will show. **Left-click** on the desired category to tag the text into this category.

Now try to tag the word "**ducks**" in the following sentence into the category "**Animals**":

```
Five little ducks.
```

Tagging using the mouse:

1) Select the text to be tagged.

2) **Double-click** the desired category in the category-list on the right side to tag the text into this category.

Figure 6.12: Tutorial document for filmers, as used in the usability evaluation.

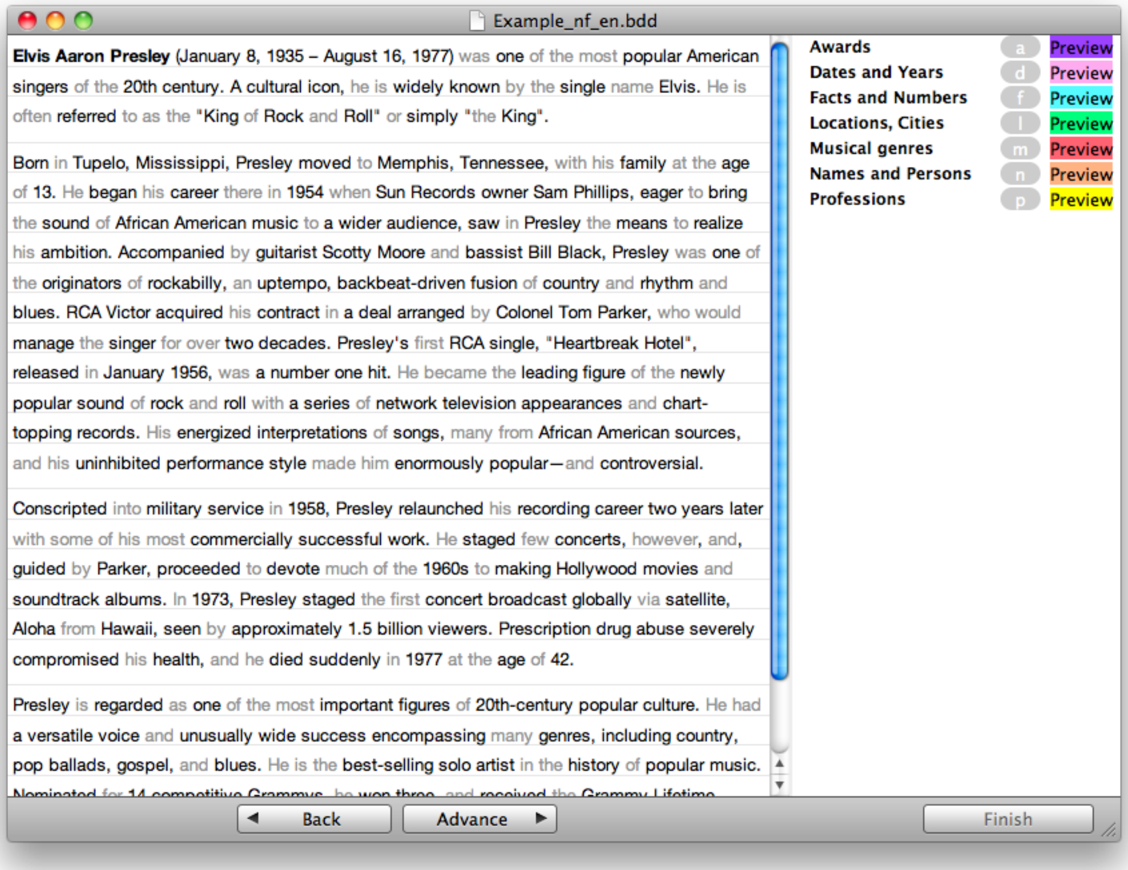
for the completion of the tutorial screen. However, the button for continuing to the next step of the test remains disabled until certain requirements are fulfilled. In the non-filmers version, for example, the user is supposed to tag at least three words into the only available category, so as to make sure the tutorial screen is not just skipped altogether.

3. **Test**

   The test document is shown after the user has completed the tutorial. As with the tutorial document, two test documents pertaining two each of the test groups were created.

   In the filmers-version, a short excerpt of the script to the popular movie adaption of "The Wizard of Oz" (Langley et al., 1939) is used. Additionally, a typical list of script breakdown categories (adapted from Singleton (1991); Clevé (2005)) is provided alongside the script.

The test-document for non-filmers, on the other hand, is based on the Wikipedia article for Elvis Presley[3] and adapted slightly for the test. The list of categories for tagging consisted of more general items like *Names and Persons*, *Locations*, *Musical genres* and similar. The test-document for non-filmers is shown in figure 6.13.



Elvis Aaron Presley (January 8, 1935 – August 16, 1977) was one of the most popular American singers of the 20th century. A cultural icon, he is widely known by the single name Elvis. He is often referred to as the "King of Rock and Roll" or simply "the King".

Born in Tupelo, Mississippi, Presley moved to Memphis, Tennessee, with his family at the age of 13. He began his career there in 1954 when Sun Records owner Sam Phillips, eager to bring the sound of African American music to a wider audience, saw in Presley the means to realize his ambition. Accompanied by guitarist Scotty Moore and bassist Bill Black, Presley was one of the originators of rockabilly, an uptempo, backbeat-driven fusion of country and rhythm and blues. RCA Victor acquired his contract in a deal arranged by Colonel Tom Parker, who would manage the singer for over two decades. Presley's first RCA single, "Heartbreak Hotel", released in January 1956, was a number one hit. He became the leading figure of the newly popular sound of rock and roll with a series of network television appearances and chart-topping records. His energized interpretations of songs, many from African American sources, and his uninhibited performance style made him enormously popular—and controversial.

Conscripted into military service in 1958, Presley relaunched his recording career two years later with some of his most commercially successful work. He staged few concerts, however, and, guided by Parker, proceeded to devote much of the 1960s to making Hollywood movies and soundtrack albums. In 1973, Presley staged the first concert broadcast globally via satellite, Aloha from Hawaii, seen by approximately 1.5 billion viewers. Prescription drug abuse severely compromised his health, and he died suddenly in 1977 at the age of 42.

Presley is regarded as one of the most important figures of 20th-century popular culture. He had a versatile voice and unusually wide success encompassing many genres, including country, pop ballads, gospel, and blues. He is the best-selling solo artist in the history of popular music. Nominated for 14 competitive Grammys, he won three, and received the Grammy Lifetime

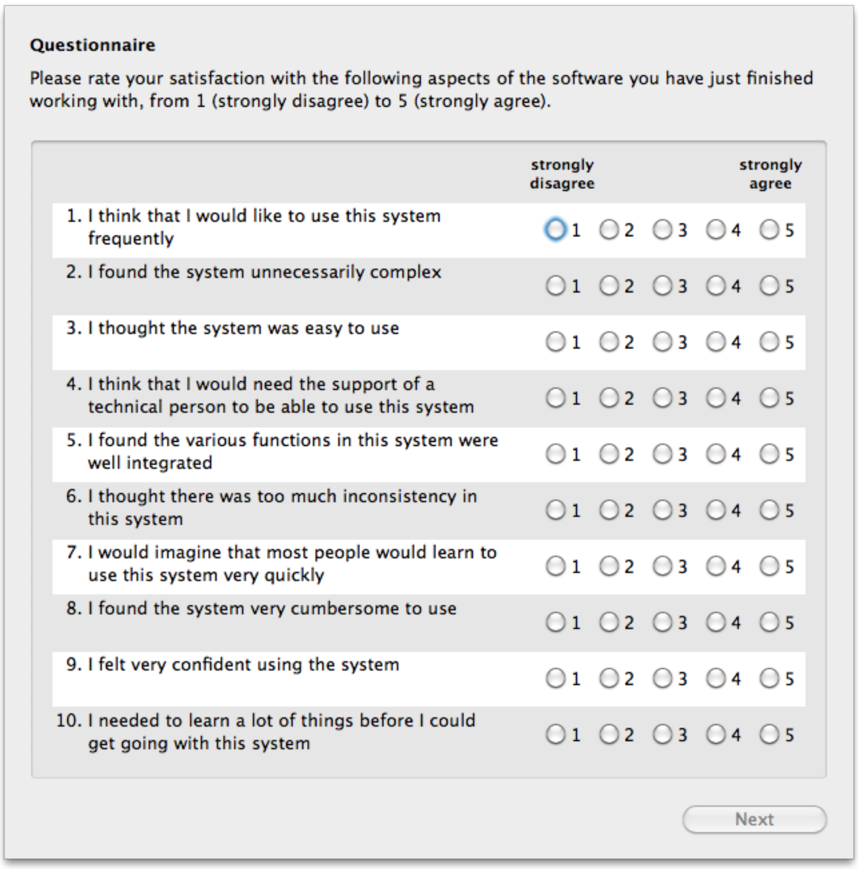
Awards	a	Preview
Dates and Years	d	Preview
Facts and Numbers	f	Preview
Locations, Cities	l	Preview
Musical genres	m	Preview
Names and Persons	n	Preview
Professions	p	Preview

Figure 6.13: Test document for non-filmers, as used in the usability evaluation.

4. **Feedback**

   A page incorporating a SUS feedback form concludes the test workflow. The answers given by the users are, as with the background forms and all other data, collected automatically and included in the results set. Figure 6.14 shows the SUS feedback form.

[3]`http://en.wikipedia.org/wiki/Elvis_presley` English Wikipedia article on Elvis Presley, last access 11/2010



Questionnaire

Please rate your satisfaction with the following aspects of the software you have just finished working with, from 1 (strongly disagree) to 5 (strongly agree).

	strongly disagree				strongly agree
1. I think that I would like to use this system frequently	○ 1	○ 2	○ 3	○ 4	○ 5
2. I found the system unnecessarily complex	○ 1	○ 2	○ 3	○ 4	○ 5
3. I thought the system was easy to use	○ 1	○ 2	○ 3	○ 4	○ 5
4. I think that I would need the support of a technical person to be able to use this system	○ 1	○ 2	○ 3	○ 4	○ 5
5. I found the various functions in this system were well integrated	○ 1	○ 2	○ 3	○ 4	○ 5
6. I thought there was too much inconsistency in this system	○ 1	○ 2	○ 3	○ 4	○ 5
7. I would imagine that most people would learn to use this system very quickly	○ 1	○ 2	○ 3	○ 4	○ 5
8. I found the system very cumbersome to use	○ 1	○ 2	○ 3	○ 4	○ 5
9. I felt very confident using the system	○ 1	○ 2	○ 3	○ 4	○ 5
10. I needed to learn a lot of things before I could get going with this system	○ 1	○ 2	○ 3	○ 4	○ 5

Figure 6.14: SUS feedback form, as used in the usability evaluation.

Language support

Language support was limited to English and German, and divided into the general interface language of the prototype, and the language for the tutorial and test documents used within the test.

Language selection for the interface (including background forms and SUS feedback questionnaire) was based upon the user's locale settings, which is the default behavior for all software on Mac OS computers. This meant that the software itself was in either English or German, depending on which language was ranked higher in the list of the user's personally preferred languages, as set in the Mac OS system preferences. This was also one of the benefits from allowing the users to run the test on their own computers.

On the other hand, language selection for the documents presented to the user throughout the test was based primarily on the user's native language, which was queried by a question on the first page of the background forms. The available answers were English, German, and other (with a suggestion to note which other language). If the user selected either English or German, the respective language was used for the documents, thus allowing users on an English operating system to work on the German documents if it was their native language, for example. Otherwise, the user's locale setting was again used to determine the document language, analogous to the user interface language. An exception to this was the test document for the filmers group, which was only provided in English.

Result sets - data generation and statistics

It has already been mentioned that throughout the evaluation process, a number of different types of information and data were collected from various sources. These data were stored in memory for the duration of the test and sent to a script on a server which was written to receive the results and store them appropriately. Table 6.2 provides a general definition of the structure these result sets followed.

Section	**Notes**	**Type**
comment	comments the user added to the results	text
project data	breakdown data created throughout the test	hierarchy
e.g. `Locations (Tupelo, Memphis), Musical genres (Blues, Rock)`		
usability data	data from background and SUS feedback forms	key-value pairs
e.g. `age = 23, language = English, SUS question 1 = 3`		
usage protocol	chronological list of interactions with timestamps	list of text
e.g. `2010-09-03 13:23:17 - tagging "Elvis" into "Names"`		
usage statistics	statistics automatically generated from interaction	key-value pairs
e.g. `AAS Tagging = 17, Untag = 3, BACKSPACE = 7`		

Table 6.2: The structure of a result set received from a completed usability test with example values.

While some of this data was gathered directly from values the user entered (comment, project and usability data), the usage data was generated indirectly from the user's interaction with the software. Since these interactions mostly consisted of mouse clicks and keyboard presses, and logging and evaluating these interactions should not require any alterations to the software design and implementation, an aspect-oriented approach (Elrad et al., 2001) was used.

Using several mechanisms of the *Objective-C*[4] runtime environment, specific methods responsible for handling user interaction were replaced by slightly altered implementations during runtime. These altered implementations added some functionality to log user interaction and, if applicable, calculate and update statistical values. After this, they invoked the original implementations in order to guarantee absolute transparency and seamless integration of usage logging.

Example If the user presses a key associated with a category to tag the currently selected text, the method responsible for handling key presses is `keyDown:`, implemented by the `DocumentController`. This method will have been replaced by an aspect method logging this specific event textually into the usage log and delegating the actual handling of the event to the original method.

This original method will at some point trigger tagging of the selected text into the category by invoking `tagSelectedTextIntoCategory:` on `DocumentController`. Again, this method will have been replaced by an aspect method which will first increase the number of tagging actions issued via keyboard shortcut (i.e. AAS-tagging) and then delegate tagging to the original implementation.

6.2.3 User demographics

As was noted, all test users were asked to enter some basic demographic information. Tables 6.3 and 6.4 present these data for both test groups. The columns correspond to sex, age, experience with computers, most used operating system, visual impairments (VI) and highest attained education.

In the filmers group, 20% of the testers were female and all participants most commonly used the Apple OSX operating system. In the non-filmers group, 45% were female and most commonly used operating systems were Apple OSX (two thirds) and Microsoft Windows (one third).

[4]`http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ObjectiveC/` Apple's Introduction to the Objective-C Programming Language, last access 11/2010

	Sex	Age	Exp	OS	VI	Edu
TP1F	M	25	10	OSX	-	University
TP2F	M	22	13	OSX	-	Secondary school
TP3F	M	24	12	OSX	-	Secondary school
TP4F	F	25	13	OSX	Contact lenses	Secondary school
TP5F	M	30	17	OSX	Glasses	University
Average	-	25.20	13.00	-	-	-
Minimum	-	22	10	-	-	-
Maximum	-	30	17	-	-	-
Median	-	25	13	-	-	-
Std.Dev.	-	2.95	2.55	-	-	-

Table 6.3: Demographic data of the test participants in the filmers group.

For the filmers group, invitations to participate in the study were issued in three mailing lists of German film academies with several hundred recipients each, as well as in a Facebook group of Austrian and German film producers with more than 1000 members. Additionally, invitations were sent directly to several film producers and lectors in Germany.

After separate pre-tests, a total of 21 people participated in the usability evaluation. All participants were asked to add verbal comments before sending the corresponding results. Three participants in the non-filmers group were briefly interviewed after the test.

6.2.4 Results

Tables 6.5 and 6.6 present statistical results and calculated SUS results of both test groups.

Adoption rate is the ratio between the number of users who adopted a particular tagging mechanism as their main tagging interface against the total number of users in the group. A user was counted towards a particular tagging interface if it was used to tag more entries than all other interfaces combined. It is to note that this approach yielded an absolute preferred method for all users in the study.

The aspect that backtracking events can indicate usability problems, as mentioned in chapter 3.4.1, was considered by calculating the ratio between the average number of tagged entries versus the average number of untagged entries for each user. This ratio is reflected by the untagging rate and calculated towards the respective

	Sex	**Age**	**Exp**	**OS**	**VI**	**Edu**
TP1NF	M	27	20	OSX	-	University
TP2NF	F	25	10	OSX	-	University
TP3NF	M	30	20	OSX	Glasses	Secondary school
TP4NF	F	27	12	OSX	-	Secondary school
TP5NF	F	50	12	OSX	Contact lenses	Secondary school
TP6NF	M	27	15	OSX	-	University
TP7NF	F	25	15	OSX	-	Secondary school
TP8NF	M	30	17	Win	-	Secondary school
TP9NF	M	29	13	Win	Glasses	University
TP10NF	M	33	25	OSX	Glasses	Secondary school
TP11NF	M	30	18	OSX	-	Secondary school
TP12NF	M	28	17	OSX	Glasses	University
TP13NF	M	29	20	OSX	Glasses	University
TP14NF	F	30	20	OSX	Other *	University
TP15NF	M	27	16	Win	Contact lenses	University
TP16NF	M	29	17	Win	Glasses	Secondary school
Average	-	29.75	13.00	-	-	-
Minimum	-	25	10	-	-	-
Maximum	-	50	17	-	-	-
Median	-	29	13	-	-	-
Std.Dev.	-	5.78	2.55	-	-	-

Table 6.4: Demographic data of the test participants in the non-filmers group.
* Red-green color blindness

user's preferred tagging method.

Similar to the untagging rate, the navigation rates describe which methods of input were used mainly for scrolling through and selecting text. The tree methods were keyboard navigation using cursor keys, button navigation using the next and previous buttons and mouse navigation using the mouse for scrolling and selecting text. All these methods inherently use pre-selection heuristics. For cross-examining the data with the corresponding preferred tagging mechanisms, these values have been produced for each of the tagging methods separately.

The SUS scores were calculated according to section 3.4.2. Table 6.7 shows detailed results of the SUS score evaluation, including individual scores per question, for both test groups. Individual SUS scores for questions range from 0.00 (lowest rating) to 4.00 (highest rating). Values below or equal 2.50 are in bold. The overall score is a percent value, thus its possible range is 0.00 (lowest score) to 100.00 (highest score). The overall SUS score of all users for this evaluation was **71.55**.

	AAS	Context Menu	Button	Overall
Adoption rate	.750	.250	.000	1.000
Untagging rate	.113	.000	-	-
Keyboard nav.	.500	.000	-	-
Button nav.	.000	1.000	-	-
Mouse nav.	.500	.000	-	-
SUS score	70.00	87.50	-	73.50
Minimum	7.50*	87.50	-	7.50
Maximum	97.50	87.50	-	97.50
Median	87.50	87.50	-	87.50
Std.Dev.	42.38	-	-	37.52

Table 6.5: Statistics concerning the test results of the filmers group by tagging interface.
* Irregularity in results potentially caused by the user misunderstanding the SUS scale. Average SUS score of non-filmers group excluding this score is 90.

	AAS	Context Menu	Button	Overall
Adoption rate	.625	.250	.125	1.000
Untagging rate	.020	.028	.053	-
Keyboard rate	.600	1.000	.000	-
Button rate	.100	.000	.500	-
Mouse rate	.300	.000	.500	-
SUS score	71.50	73.75	62.50	70.94
Minimum	30.00	60.00	57.50	30.00
Maximum	90.00	85.00	67.50	90.00
Median	78.75	75.00	62.50	75.00
Std.Dev.	19.97	10.50	7.07	16.63

Table 6.6: Statistics concerning the test results of the non-filmers group by tagging interface.

6.2.5 Discussion

Tables 6.5 and 6.6 show high adoption rates for AAS tagging with 75% and 62.5% of users mainly using AAS for breakdown. This is surprising because it uses a concept that is less popular and well-known than the other tagging interfaces. More than half of the users preferring AAS for tagging used the keyboard to navigate and select text as well. This makes AAS + keyboard navigation by far the most used combination of input methods in the test.

Untagging rates in the non-filmers group indicate that users preferring button tagging made about twice as many corrections as other test participants. Untagging

	Filmers (5 results)			Non-Filmers (16 results)		
	Average	Median	Std.Dev.	Average	Median	Std.Dev.
1	3.40	4.00	1.34	**2.00**	2.50	1.46
2	3.20	4.00	1.79	3.13	3.00	1.02
3	3.20	4.00	1.79	**2.38**	3.00	1.31
4	3.20	4.00	1.79	3.56	4.00	1.03
5	**2.00**	**1.00**	1.41	2.81	3.00	1.05
6	3.00	4.00	1.73	3.00	3.00	1.10
7	**2.40**	3.00	1.34	3.38	3.50	0.81
8	3.20	4.00	1.79	**2.38**	3.00	1.15
9	3.20	4.00	1.30	**2.25**	**2.00**	1.29
10	2.60	3.00	1.67	3.50	4.00	0.73
Overall	73.50	87.50	37.52	70.94	75.00	16.63

Table 6.7: SUS score details with individual SUS scores per question for the filmers and non-filmers groups.

rates in the filmers group bear no meaningful information due to the low adoption rate of tagging methods other than AAS. Comparing both groups shows that filmers corrected many more entries than non-filmers, which is likely to be related to the different types of test documents and a more professional approach from participants in the filmers group.

A comparison of SUS scores reveals that users favoring AAS and context menu tagging were generally more satisfied with the software than those mainly using interface buttons for tagging. While average AAS SUS scores are lower than context menu scores, the standard deviation is also much higher, which is related to a higher adoption rate. Accordingly, median values for AAS are better than those for context menu, as it smoothes out extreme values such as those shown in the minimum SUS score for AAS tagging in table 6.5.

Overall SUS scores of 73.50 in the filmers group and 70.94 in the non-filmers group are satisfying for an initial usability test of the prototype. Context menu tagging received slightly better ratings than AAS tagging, whereas button tagging suffered from low adoption and worse ratings.

Aspects related to the lowest ranked SUS questions, which are shown in tables 6.8 and 6.9, are recommended for improvement prior to the next usability evaluation. All questions with a score (average rating) or a median rating of 2.50 or below are considered.

Ranked	Question	Average	Median	Std.Dev.
F.1	I found the various functions in this system were well integrated.	**2.00**	**1.00**	1.41
F.2	I would imagine that most people would learn to use this system very quickly.	**2.40**	3.00	1.34

Table 6.8: SUS questions for the filmers group ranked by SUS score (average rating), median rating and standard deviation.

Ranked	Question	Average	Median	Std.Dev.
NF.1	I think that I would like to use this system frequently.	**2.00**	**2.50**	1.46
NF.2	I felt very confident using the system.	**2.25**	**2.00**	1.29
NF.3	I found the system very cumbersome to use.	**2.38**	3.00	1.15
NF.4	I thought the system was easy to use.	**2.38**	3.00	1.31

Table 6.9: SUS questions for the non-filmers group ranked by SUS score (average rating), median rating and standard deviation.

It was noted that all test participants were also asked to add feedback before sending in the result data. The following list presents the most frequently issued **negative** comments:

C.1 *I had problems selecting the right words.* (6 NF)

C.2 *It was unclear which words to tag into which categories.* (3 NF)

C.3 *The list of categories or keywords does not fit on screen.* (2 NF)

C.4 *The mixing of English GUI and German documents was irritating.* (1 NF)

These results show that participants in the filmers group were generally more satisfied with the software, which is reflected by the higher overall SUS scores as discussed above. **F.1** can be interpreted to mean that the feature-reduced usability prototype provided too little functionality for professionals in film production. In the pre-test phase, this notion was verbalized by one film producer and therefore according information was included in the correspondence to potential test participants of this group. A more general interpretation of **F.1** is that filmers were unsure which of the tagging methods to use. Further investigation into this problem is recommended.

F.2, **NF.1** and **NF.2** are closely related to the fact that the software is tailored to the task of script breakdown. While the evaluation documents for the non-filmers group were adapted thoroughly, some irritation remained, as can also be seen by some of the non-filmers' negative comments. For future tests it is recommendable to re-assess the test documents or, if possible, intensify tests with filmers. **NF.3** and **NF.4** are problems that can also be associated with the issues above, but suggest further improvement of the user interface and tweaking of parameters in the preselection heuristics, as indicated by **C.1** and **C.3**.

6.2.6 Conclusion

The usability evaluation of the breakdown prototype showed that test participants with a professional background in film production could work well with the software and were generally satisfied by how usable it was. Further work is recommended to investigate **F.1**.

Members of the other test group without education or experience in film production adapted well to the inherent task of tagging elements of free text into categories, but some irritation remained which is partly rooted in the prototype implementation and interface and subject for further work to make the software more accessible for users without film production education or experience.

Further work on the prototype must therefore involve improvements of the mentioned problems with continuous integration of film production professionals and usability evaluations. In this regard, the results of this first rather extensive development iteration are very promising with a good overall SUS score of 71.55 and highly valuable user feedback.

7. Discussion and Lessons Learned

At the start of this project, currently available software for script breakdown was neither enjoyable nor efficient to use, lacked intuitive workflows and basic features and was hardly integrated into film production processes. In investigating these issues, a number of research topics in different scientific fields have been covered. For instance, four important techniques and methods in natural language processing (NLP) have been researched and evaluated according to key factors for this work.

The outcome was that, for the time being, stop word filtering provides a simple but efficient means of conducting basic text recognition in movie scripts and similar types of natural language text. According to the findings of the usability evaluation, the resulting filtering technique was well accepted by the test participants. The corresponding implementation of pre-selection heuristics (PSH) was also well received by the filmers test group, whereas the non-filmers encountered some problems with the Wikipedia article. This suggests that some of the mechanisms used in PSH and AAS work differently for film scripts and other free texts and that future usability evaluations must take this into account even more.

Another research and design topic in this work was concerned with software design patterns for architectural software design. The well-known MVC pattern was thoroughly discussed and compared to new patterns and paradigms like DCI and Traits. It was discussed that MVC has deficiencies in modeling system behavior that are mitigated by other techniques, but that MVC is widely used and therefore recommendable for systems interacting with APIs and frameworks that are based on MVC, as is the case in this work.

Basing the prototype's software design on MVC proved most reasonable as it allowed the clear separation of the concerns of data modeling and persistence, presentation and interface as well as user interaction and workflows. For instance, the algorithms and methods used for PSH are invoked by all three tagging mechanisms transparently on a controller. The prototype can therefore easily be extended with new tagging interfaces or differently parameterized PSH algorithms. This also greatly widens the possibilities of adapting the user interface to future requirements or suggestions derived from future usability evaluations.

One of the most important aspects of this work is that the prototype is fully prepared for direct integration into various business workflows. The prototype is currently able to import a number of text formats directly via copy and paste, while adding specialized import mechanisms for proprietary formats can easily be done. For data export, the prototype already implements the generation of XML data of the project, also with the possibility of easily adding other ways of structuring the data. This allows for direct integration into scheduling or budgeting software, which offers great potential for faster and earlier cost estimations, overall work time saving, improved correctness of schedules and budgets, and much more.

8. Conclusions

As was pointed out in the previous chapter, current software solutions for script breakdown do not meet the demands of professional film productions. A result of this is that many producers and production assistants still carry out script breakdown manually on printouts and with highlighters. These methods are clearly not well integrated into the production processes.

To improve on this situation, the prototype developed for and presented in this work was expected to meet a number of technical and formal requirements stated in section 6.1. An observation of how well and to what extent these goals were met will now follow:

- **The target operating system is Mac OS X 10.5 and higher.**
  The prototype was partly developed and thoroughly tested on Mac OS X 10.5.

- **Interfaces must be designed and implemented to be suitable for resolutions of 1280x800 pixels and higher.**
  The prototype was mainly developed on a MacBook with a display resolution of 1280x800 (see table 6.1).

- **Tagging must be efficient enough to be capable of global-tagging multiword elements in reasonably long scripts (`>10,000 words`) in a timely fashion (`<1 second`).**
  On the development machine (see table 6.1), tagging of differently structured entries (single words, word groups, sub-tokens) in a script with 21,840 words with enabled stop word filtering and usability data gathering takes at the most 0.7015s. Note that these are the results of a standardized benchmark with typical data.

- **Saving/loading of project files (including script with formattings, elements, categories and scenes) and category configurations must be supported.**

  This feature complex has been fully implemented and was used in the usability evaluation to distribute and automatically load appropriate test documents for both test groups and different language versions.

- **Focus on usability in terms of issues discussed in sections 3.4 and 4.1.**

  Where not enforced by the development tools used to create the user interfaces, this requirement was adhered to by careful design and evaluated in the usability study. The study showed that test participants were pleased with the GUI and that testers with a background in film production quickly adopted keyboard tagging and navigation.

- **Streamline the process from script breakdown to the first cost estimation, especially by the smart tagging of elements and semi-automatic management of scenes and categories.**

  Auto-advancing shortkey tagging (AAS) in combination with pre-selection heuristics (PSH) was implemented in order to improve efficiency and ease of conducting script breakdown. Automatic scene recognition and flexible category management complement the streamlined process.

- **Ensure re-usability of data by providing data exports in a universally usable format. Moreover, support printing of breakdown data, for instance breakdown sheets for scenes, actors and locations.**

  An XML-based data interface for exporting and printing data was implemented. Printing is further improved by employing a text-based template engine allowing to easily extend the set of provided reports and breakdown sheets by creating new templates which require no client programming.

This work thus shows how issues identified in existing software solutions for script breakdown were remedied or avoided altogether, and what was done to further ensure joy of use, overall software quality and better integration into existing business processes.

9. Future Work

There are some concrete recommendations for future work on the prototype to further increase its value in practical application. For example, more templates for breakdown sheet prints and more customization options are suggestible.

An investigation of the possibilities for adopting more sophisticated pre-processing and text recognition techniques is also recommended. For instance, automatic recognition of dialogue text in the script can significantly reduce workload for the user as it often conveys no factual information about the scene not already present outside the dialog. Moreover, pre-selection heuristics could still be improved by using part-of-speech tagging or lexical databases, for example. In this regard, further research on interlingual techniques are recommended or language-specific limitations could be introduced as a trade-off for improved functionality.

The aspect promising most potential for further work is the ability to integrate script breakdown more directly into the information processes in film productions. This includes direct interaction with scriptwriting software to receive script data, possibly enriched or annotated by the authors, that can be utilized directly, as well as immediate communication with scheduling and budgeting software, for instance to update scene structure in schedules or amend budgeting data with newly tagged elements. The potential of such integration is enormous and the range of possible benefits includes overall workload reduction, time saving, better time and cost estimations, improved correctness of budgets and schedules and much more.

In addition to these concepts, a widening of the application spectrum is also suggestible, which can include the adaption of the system for annotating text for a number of other purposes. For instance, adaptations could be used in creating annotated text corpuses for scientific research in natural language processing, or as part of a personal learning suite to annotate important elements of a text and generate automatic summarizations of the highlighted passages. On the other hand, the concept can also be translated to work on other platforms with different input hardware, such as the Apple iPad using a touch screen. In this context, the metaphor of tagging words on printouts with text highlighters could be used to great effect.

A. SUS Feedback Sheet

	strongly disagree				strongly agree
	1	2	3	4	5
I think that I would like to use this system frequently					
I found the system unnecessarily complex					
I thought the system was easy to use					
I think that I would need the support of a technical person to be able to use this system					
I found the various functions in this system were well integrated					
I thought there was too much inconsistency in this system					
I would imagine that most people would learn to use this system very quickly					
I found the system very cumbersome to use					
I felt very confident using the system					
I needed to learn a lot of things before I could get going with this system					

Figure A.1: SUS evaluation questionnaire consisting of ten questions on Likert-scale (adapted from (Brooke, 1996)).

List of Figures

List of Tables

References

Abrahamsson, Pekka, Antti Hanhineva, and Juho Jäälinoja [2005]. *Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development.* In Baskerville, Richard, Lars Mathiassen, Jan Pries-Heje, and Janice DeGross (Editors), *Business Agility and Information Technology Diffusion*, *IFIP International Federation for Information Processing*, volume 180, pages 227–243. Springer Boston. doi:10.1007/0-387-25590-7_14. `http://www.springerlink.com/content/w27pj02827064404/`.

Adobe Systems Incorporated [2010]. *PDF Reference and Adobe Extensions to the PDF Specification.* `http://www.adobe.com/devnet/pdf/pdf_reference.html`. Last access 09/2010.

Akers, David [2009]. *Backtracking Events as Indicators of Software Usability Problems.* PHD dissertation, Stanford University. `http://www.math.ups.edu/~dakers/papers/dakers_dissertation_1s.pdf`. Last access 09/2010.

Andreasen, Morten Sieker, Henrik Villemann Nielsen, Simon Ormholt Schrøder, and Jan Stage [2007]. *What happened to remote usability testing?: an empirical study of three methods.* In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1405–1414. ACM, New York, NY, USA. ISBN 9781595935939. doi:10.1145/1240624.1240838. `http://portal.acm.org/citation.cfm?doid=1240624.1240838`.

Andrews, Keith [2010]. *Lecture Notes on Human-Computer Interaction.* Graz University of Technology, Austria. `http://courses.iicm.tugraz.at/hci/hci.pdf`. Last access 09/2010.

Bevan, Nigel and Miles Macleod [1994]. *Usability measurement in context. Behaviour*

& Information Technology, 13(1&2), pages 132–145. ISSN 13623001. doi:10.1080/01449299408914592. `http://www.informaworld.com/index/773461629.pdf`.

Booch, Grady, Robert Maksimchuk, Michael Engle, Bobbi J. Young, Jim Conallen, and Kelli A. Houston [2007]. *Object-oriented analysis and design with applications*. Third Edition. Addison-Wesley Professional. ISBN 020189551X. `http://portal.acm.org/citation.cfm?id=1407387`.

Brill, Eric [1992]. *A simple rule-based part of speech tagger*. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics (ACL), Morristown, NJ, USA. doi:10.3115/974499.974526. `http://portal.acm.org/citation.cfm?id=974526`.

Brill, Eric [1993]. *Automatic grammar induction and parsing free text: a transformation-based approach*. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 237–242. Association for Computational Linguistics, Morristown, NJ, USA. ISBN 1558603247. doi:10.3115/1075671.1075726. `http://portal.acm.org/citation.cfm?doid=1075671.1075726`.

Brill, Eric and Mitch Marcus [1992]. *Tagging an Unfamiliar Text With Minimal Human Supervision*. In *AAAI Fall Symposium Series: Probabilistic Approaches to Natural Language (Working Notes)*, pages 10–16. Press. `http://www.aaai.org/Papers/Symposia/Fall/1992/FS-92-04/FS92-04-002.pdf`.

Brooke, John [1996]. *SUS: A quick and dirty usability scale*. `http://www.usabilitynet.org/trump/documents/Suschapt.doc`. Last access 09/2010.

Bruun, Anders, Peter Gull, Lene Hofmeister, and Jan Stage [2009]. *Let your users do the testing: a comparison of three remote asynchronous usability testing methods*. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1619–1628. ACM, New York, NY, USA. ISBN 9781605582467. doi:10.1145/1518701.1518948. `http://portal.acm.org/citation.cfm?doid=1518701.1518948`.

Budanitsky, Alexander and Graeme Hirst [2006]. *Evaluating WordNet-based Measures of Lexical Semantic Relatedness*. *Computational Linguistics*, 32(1), pages

13–47. doi:10.1162/coli.2006.32.1.13. http://www.mitpressjournals.org/doi/abs/10.1162/coli.2006.32.1.13.

Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal [1996]. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*. John Wiley & Sons, Inc. ISBN 0471958697. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471958697.html. Last access 09/2010.

Church, Kenneth Ward [1988]. *A stochastic parts program and noun phrase parser for unrestricted text*. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143. Association for Computational Linguistics, Morristown, NJ, USA. doi:10.3115/974235.974260. http://www.aclweb.org/anthology-new/A/A88/A88-1019.pdf#url.dl&CFID=96615029&CFTOKEN=74268930.

Clevé, Bastian [2005]. *Film Production Management*. Third Edition. Focal Press. ISBN 0240806956. http://focalpress.com/Book.aspx?id=6558. Last access 09/2010.

Cohen, Aaron M. and William R. Hersh [2005]. *A survey of current work in biomedical text mining. Brief Bioinform*, 6(1), pages 57–71. ISSN 14675463. doi:10.1093/bib/6.1.57. http://bib.oxfordjournals.org/cgi/content/abstract/6/1/57.

Coplien, James O. and Gertrud Bjørnvig [2010]. *Lean Architecture: for Agile Software Development*. John Wiley & Sons, Inc. ISBN 9780470684207. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470684208.html. Last access 11/2010.

Davis, Mark [2009]. *Unicode Technical Reports: UAX 29 Unicode Text Segmentation*. http://www.unicode.org/reports/tr29/. Last access 09/2010.

Directors Guild of America, Inc [2005]. *Basic Agreement of 2005, Article 1: Recognition and Guild Shop*. http://www.dga.org/contracts/ba2005-finalpdfs/03-ba2005-1.pdf. Last access 09/2010.

Dybå, Tore [2005]. *An Empirical Investigation of the Key Factors for Success in Software Process Improvement. IEEE Transactions on Software Engineering*, 31, pages 410–424. ISSN 00985589. doi:10.1109/TSE.2005.53. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1438376&tag=1`.

Elrad, Tzilla, Robert E. Filman, and Atef Bader [2001]. *Aspect-oriented programming: Introduction. Communications of the ACM*, 44(10), pages 29–32. ISSN 00010782. doi:10.1145/383845.383853. `http://portal.acm.org/citation.cfm?doid=383845.383853`.

Flores, Pablo A., Rodrigo F. Flores, Raul Medina-Mora Icaza, Jaime Garza Vasquez, John A. McAfee, Manoj Kumar, Manuel Jasso Nunez, Terry Allen Winograd, Harry K. T. Wong, and Roy I. Gift [1998]. *Method and apparatus for building business process applications in terms of its workflows.* `http://v3.espacenet.com/publicationDetails/biblio?CC=US&NR=5734837A&FT=D`.

Fowler, Martin [2002]. *Patterns of Enterprise Application Architecture.* First Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0321127420. `http://portal.acm.org/citation.cfm?id=579257`.

Gamma, Erich, Richard Helm, Ralph Johnson, and John M. Vlissides [1994]. *Design Patterns: Elements of Reusable Object-Oriented Software.* First Edition. Addison-Wesley Professional. ISBN 0201633612. `http://c2.com/cgi/wiki?DesignPatternsBook`. Last access 11/2010.

Gemmell, Matt [2008]. *MGTemplateEngine - Templates with Cocoa.* `http://mattgemmell.com/2008/05/20/mgtemplateengine-templates-with-cocoa`. Last access 09/2010.

Harrison, Warren, David Raffo, John Settle, and Nancy Eickelmann [1999]. *Technology Review: Adapting Financial Measures: Making a Business Case for Software Process Improvement. Software Quality Control*, 8, pages 211–231. ISSN 09639314. doi:10.1023/A:1008971726703. `http://portal.acm.org/citation.cfm?id=599120.599184`.

Hearst, Marti [2003]. *What Is Text Mining?* `http://people.ischool.berkeley.edu/~hearst/text-mining.html`. Last access 09/2010.

Hilbert, David M. and David F. Redmiles [2000]. *Extracting usability information from user interface events. ACM Computing Surveys (CSUR)*, 32(4), pages

384–421. ISSN 03600300. doi:10.1145/371578.371593. http://portal.acm.org/citation.cfm?doid=371578.371593.

Holzinger, Andreas [2005]. *Usability engineering methods for software developers. Communications of the ACM*, 48(1), pages 71–74. ISSN 00010782. doi:10.1145/1039539.1039541. http://portal.acm.org/citation.cfm?doid=1039539.1039541.

Holzinger, Andreas, Regina Geierhofer, Felix Mödritscher, and Roland Tatzl [2008]. *Semantic Information in Medical Information Systems: Utilization of Text Mining Techniques to Analyze Medical Diagnoses. Journal of Universal Computer Science*, 14(22), pages 3781–3795. http://www.jucs.org/jucs_14_22/semantic_information_in_medical.

Holzinger, Andreas, Karl Heinz Struggl, and Matjaz Debevc [2010]. *Applying Model-View-Controller (MVC) in Design and Development of Information Systems: An example of smart assistive script breakdown in an e-Business Application.* In *ICE-B 2010 - Proceedings of the International Conference on e-Business, Athens, Greece*, pages 63–68. ISBN 978989674.

Hotho, Andreas, Andreas Nürnberger, and Gerhard Paaß [2005]. *A brief survey of text mining. LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20, pages 19–62. http://www.kde.cs.uni-kassel.de/hotho/pub/2005/hotho05TextMining.pdf.

Langley, Noel, Florence Ryerson, and Edgar Allen Woolf [1939]. *The Wizard of Oz, movie script, based on the book by L. Frank Baum.* http://sfy.ru/?script=wizard_of_oz_1939. Last access 09/2010.

Larsen, Bjornar and Chinatsu Aone [1999]. *Fast and effective text mining using linear-time document clustering.* In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–22. ACM, New York, NY, USA. ISBN 1581131437. doi:10.1145/312129.312186. http://portal.acm.org/citation.cfm?doid=312129.312186.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze [2008]. *Introduction to Information Retrieval.* Cambridge University Press, New

York, NY, USA. ISBN 0521865719. `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`. Last access 09/2010.

Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller [1993]. *Introduction to WordNet: An On-line Lexical Database.* `http://wordnetcode.princeton.edu/5papers.pdf`. Last access 09/2010.

Nielsen, Jakob [1994a]. *Usability Engineering.* Morgan Kaufmann. ISBN 0125184069. `http://www.useit.com/jakob/useengbook.html`. Last access 09/2010.

Nielsen, Jakob [1994b]. *Usability inspection methods.* In *CHI '94: Conference companion on Human factors in computing systems*, pages 413–414. ACM, New York, NY, USA. ISBN 0897916514. doi:10.1145/259963.260531. `http://portal.acm.org/citation.cfm?doid=259963.260531`.

Oracle Corporation [2010]. *MySQL 5.6 Manual: Full-Text Stopwords.* `http://dev.mysql.com/doc/refman/5.6/en/fulltext-stopwords.html`. Last access 09/2010.

Pedersen, Ted, Siddharth Patwardhan, and Jason Michelizzi [2004]. *WordNet::Similarity: measuring the relatedness of concepts.* In *Demonstration Papers at HLT-NAACL 2004*, pages 38–41. HLT-NAACL '04, Association for Computational Linguistics, Morristown, NJ, USA. `http://portal.acm.org/citation.cfm?id=1614025.1614037`.

Pianta, Emanuele, Luisa Bentivogli, and Christian Girardi [2002]. *MultiWordNet: developing an aligned multilingual database.* In *Proceedings of the First International Conference on Global WordNet.* `http://multiwordnet.fbk.eu/paper/MWN-India-published.pdf`.

Princeton University [2010]. *WordNet.* Department of Computer Science, Princeton, New Jersey, United States. `http://wordnet.princeton.edu`. Last access 09/2010.

Reenskaug, Trygve [1979a]. *Models-Views-Controllers. Xerox PARC.* `http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf`. Last access 11/2010.

Reenskaug, Trygve [1979b]. *THING-MODEL-VIEW-EDITOR - an Example from a planningsystem. Xerox PARC.* `http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf`. Last access 11/2010.

Reenskaug, Trygve [2009]. *The Common Sense of Object Orientated Programming. Department of Informatics, University of Oslo, Norway.* `http://folk.uio.no/trygver/2009/commonsense.pdf`. Last access 09/2010.

Reenskaug, Trygve and James O. Coplien [2009]. *The DCI Architecture: A New Vision of Object-Oriented Programming.* `http://www.artima.com/articles/dci_vision.html`. Last access 11/2010.

Schärli, Nathanael [2005]. *Traits - Composing Classes from Behavioral Building Blocks.* PHD dissertation, Universität Bern. `http://scg.unibe.ch/archive/phd/schaerli-phd.pdf`. Last access 11/2010.

Schärli, Nathanael, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black [2002]. *Traits: The Formal Model.* Technical Report CSE-02-013, OGI School of Science & Engineering, Oregon Health & Science University. doi:10.1.1.13.7151. `http://scg.unibe.ch/archive/papers/Scha02cTraitsModel.pdf`.

Schärli, Nathanael, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black [2003]. *Traits: Composable units of behaviour.* In *ECOOP 2003 – Object-Oriented Programming*, *Lecture Notes in Computer Science (LNCS)*, volume 2743, pages 327–339. Springer Berlin / Heidelberg. ISBN 9783540405313. doi:10.1007/978-3-540-45070-2_12. `http://www.springerlink.com/content/169mbraepn4gmyd2/`.

Shi, Lei, Rada Mihalcea, and Mingjun Tian [2010]. *Cross language text classification by model translation and semi-supervised learning.* In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1057–1067. EMNLP '10, Association for Computational Linguistics, Morristown, NJ, USA. `http://portal.acm.org/citation.cfm?id=1870658.1870761`.

Singleton, Ralph S. [1991]. *Film Scheduling, or, How Long Will It Take To Shoot Your Movie?* Second Edition. Lone Eagle Publishing. ISBN 0943728398.

Singleton, Ralph S. [1996]. *Film Budgeting, or, How Much Will It Cost To Shoot Your Movie?* Lone Eagle Publishing. ISBN 0943728657.

Smith, Larry, Thomas C. Rindflesch, and W. John Wilbur [2004]. *MedPost: a part-of-speech tagger for bioMedical text*. *Bioinformatics*, 20(14), pages 2320–2321. doi:10.1093/bioinformatics/bth227. `http://bioinformatics.oxfordjournals.org/content/20/14/2320.abstract`.

Snyder, Benjamin, Tahira Naseem, Jacob Eisenstein, and Regina Barzilay [2009]. *Adding more languages improves unsupervised multilingual part-of-speech tagging: a Bayesian non-parametric approach*. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 83–91. NAACL '09, Association for Computational Linguistics, Morristown, NJ, USA. ISBN 9781932432411. `http://portal.acm.org/citation.cfm?id=1620754.1620767`.

Tan, Ah-hwee [1999]. *Text Mining: The state of the art and the challenges*. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, pages 65–70. doi:10.1.1.132.6973. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.6973`.

Vossen, Piek [2002]. *WordNet, EuroWordNet and Global WordNet*. *Revue française de linguistique appliquée*, 7, pages 27–38. `www.cairn.info/revue-francaise-de-linguistique-appliquee-2002-1-page-27.htm`.

Wilks, Yorick and Mark Stevenson [1998]. *The grammar of sense: Using part-of-speech tags as a first step in semantic disambiguation*. *Natural Language Engineering*, 4(2), pages 135–143. ISSN 13513249. doi:10.1017/S1351324998001946. `http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=48443`.

Wolf, Ivo, Marcus Vetter, Ingmar Wegner, Thomas Böttger, Marco Nolden, Max Schöbinger, Mark Hastenteufel, Tobias Kunert, and Hans-Peter Meinzer [2005]. *The Medical Imaging Interaction Toolkit*. *Medical Image Analysis*, 9(6), pages 594–604. ISSN 13618415. doi:10.1016/j.media.2005.04.005. `http://www.sciencedirect.com/science/article/B6W6Y-4G65CGY-2/2/55782e220f18fa8603eae0a33194371f`.

World Wide Web Consortium (W3C) [2010a]. *Cascading Style Sheets (CSS)*. `http://www.w3.org/Style/CSS/`. Last access 09/2010.

World Wide Web Consortium (W3C) [2010b]. *Extensible Markup Language (XML)*. `http://www.w3.org/XML/`. Last access 09/2010.

World Wide Web Consortium (W3C) [2010c]. *Hypertext Markup Language (HTML)*. `http://www.w3.org/HTML/`. Last access 09/2010.

Zhu, Hong, Patrick A. V. Hall, and John H. R. May [1997]. *Software unit test coverage and adequacy. ACM Computing Surveys (CSUR)*, 29, pages 366–427. ISSN 03600300. doi:10.1145/267580.267590. `http://doi.acm.org/10.1145/267580.267590`.