Tools for Supporting the Software Engineering Laboratory of the SFB 501

Marco Habetz

SFB 501 Bericht 04/99

Tools for Supporting the Software Engineering Laboratory of the SFB 501

Marco Habetz
habetz@informatik.uni-kl.de

Sonderforschungsbereich 501
Technical Report 04/1999

Software Engineering Group

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

ABSTRACT

Using an experience factory is one possible concept for supporting and improving reuse in software development (i. e., reuse of products, processes, quality models, …). In the context of the Sonderforschungsbereich 501: "Development of Large Systems with Generic Methods" (SFB 501), the Software Engineering Laboratory (SE Lab) runs such an experience factory as part of the infrastructure services it offers. The SE Lab also provides several tools to support the planning, developing, measuring, and analyzing activities of software development processes. Among these tools, the SE Lab runs and maintains an experience base, the SFB-EB.

When an experience factory is utilized, support for experience base maintenance is an important issue. Furthermore, it might be interesting to evaluate experience base usage with regard to the number of accesses to certain experience elements stored in the database. The same holds for the usage of the tools provided by the SE Lab. This report presents a set of supporting tools that were designed to aid in these tasks. These supporting tools check the experience base´s consistency and gather information on the usage of SFB-EB and the tools installed in the SE Lab. The results are processed periodically and displayed as HTML result reports (consistency checking) or bar charts (usage profiles).

Keywords

experience factory, experience base, SE laboratory, tool support, SFB 501

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 The Experience Factory Concept

As described in [1], software development in a reuse-oriented approach should involve two major organizations. One organization develops the system and is responsible for its delivery (the so-called “development organization”). The other, the experience factory (EF), offers support by providing experience. This experience consists of knowledge that has either been gained in previous projects or previous development cycles of the same project. Experience that goes into the EF can also be provided by external sources. A typical example for a piece of experience is a lesson learned [2]. EFs also provide experience in the form of reusable artifacts, like software products (code, documentation, …), processes (process models, methods, …) and correlations (cost or resource models). The goal of providing experience is to enable the development organization to learn from this experience in order to create products of higher quality.

For the sake of proper project support, the experience factory is continuously supplied with information on the current projects. This includes descriptions of product development and environment characteristics, resource models and data, products, or processes that are used in the project. The experience factory processes this information, retrieves new experience and enters the newly gained experience in the experience base.

The *experience base* (EB) is the repository for storage and retrieval of experience. The EB makes the experience available for development teams. We use the term “experience element” to denote the actual representations of experience that is stored in an EB. An experience base usually also stores information that characterizes experience elements and relates them to other ones. The following list describes possible types of experience elements according to the software engineering glossary of the Special Research Project 501:

- *Product models:*
  A product model describes the static properties of a class of products (especially their structure).
- *Process models:*
  A process model is a description or replication of a class of basic processes. An important issue is the description of the processes’ input/output behavior in these classes. Control flow relationships to other processes are only implicitly represented. Additionally, a process model covers further static and possibly dynamic properties of a class of processes.
  A process model does not have to represent structure and behavior of these processes exactly; approximations or limitations are also permitted.
- *Quality models:*
  A quality model determines quality by considering specific attributes of the product or process in order to describe, evaluate or predict. A quality model uses mathematical functions to describe relations between independent variables (input parameters) and usually one dependent variable (output parameter). If a quality model is empirical, its validity must be shown in each new context.
  Example: A validation technique’s test effort can be described using a quality model that relates the effort to the complexity of the component which is to be tested.

- *Techniques:*
  A technique is an instruction how to perform a specific activity (e. g., black box testing technique, programming languages).
- *Methods:*
  A method is a systematic, justified technique which is used to achieve a given goal (e. g., a testing method).
- *Tools:*
  A tool is a computer-based technique or method.
- *Products:*
  A concrete description or implementation of a system or a part of a system. The product consists of all information that is generated during its development and construction, including information that is not delivered to the customer.
- *Project plans*:
  A project plan is a project-specific instantiation of a procedural model (which describes a potential sequence of activities during the project's execution). The project plan assigns project-specific pre-set values to the parameters (e. g., project duration and effort), which are described by the models used for the project. Additionally, the project plan assigns resources for the project.

1.2 The Software Engineering Laboratory of SFB 501

Special Research Project 501 is a research project at the University of Kaiserslautern. It deals with the development of large systems using generic methods (i. e., methods which support reuse). The project's initial field of applications are house automation systems. In the remainder of this report, Special Research Project 501 is abbreviated "SFB 501", which is derived from the German translation "Sonderforschungsbereich 501".
SFB 501 is divided into several subprojects, where each subproject covers a specific area of expertise, like networking or database systems. Subproject A1 is the software engineering laboratory of SFB 501 (SE Lab). It is responsible for "supporting the experimental testing of the other subprojects' generic description techniques and development methods"; its goal is to "edit the experience gained during the project's execution to allow reuse in future experiments, and to provide this experience via the SFB 501 experience base" [4]. Therefore, the SE Lab can be seen as SFB 501's experience factory provider.

The SE Lab offers a variety of tools to support project execution. There are tools for project planning, measurement, development and analysis. In addition to this, the SE Lab maintains the SFB 501 experience base, which stores experience elements and their characterizations. We use the abbreviation SFB-EB in order to denote the experience base of SFB 501. In the context of this paper, it is important to briefly describe the current and future design of the SFB-EB.

Currently, the SFB-EB is based on the UNIX file system. Experience elements are stored as files and are accessed using an HTML-based access interface, which also contains the characterizations [3]. The access interface is based on HTML-documents and is displayed using a standard web browser.
The main disadvantage of this approach is the fact that it can only offer limited search and retrieval operations. As

for now, the user searches for experience elements by navigating via hypertext-links. Finding a specific experience element, by providing the system with keywords and having it retrieve appropriate objects, is not possible in general.

In the near future, the SFB-EB will be based on an object-relational database management system (ORDBMS). Both experience elements and the HTML-based access interface will be stored in the database. The access interface will remain unchanged, and will still be provided by a web server. However, search and retrieval operations will be fully supported by utilizing the ORDBMS.

1.3 Tool Support for the SE Lab

The SE Lab supports other subprojects by providing tools for software development and by offering experience with the help of its experience base. Besides this support provided *by*, there is a need for support *for* the SE Lab: One the one hand, the SFB-EB's current design (based on an HTML-access interface) implies regular consistency checks (we will explain the meaning of "consistency" in Section 2.1). On the other hand, the usage of tools provided by the SE Lab and the accesses to experience elements stored in the EB should be monitored in order to …

a.) … find tools provided by the software engineering laboratory that are not or rarely used, so that they can be removed in order to save disk space and/or license fees

b.) … gather information on the overall usage of the experience base so that it can be updated to better meet the needs of the EB users.

This report describes three tools which were designed to meet these requirements. Chapters 2 to 4 describe each tool's requirements in detail, as well as their design and implementation.

2 CONSISTENCY CHECKING

2.1 Consistency in the Context of SFB-EB

First of all, we have to explain what consistency means in the context of the SFB-EB. To achieve this feat, we will first explain which *inconsistencies* can occur due to the technical realization of the experience base. Obviously, the experience base will be called "consistent" if there are no inconsistencies.

Technically, the SFB-EB uses the UNIX file system to store experience elements, and provides a user interface that is based on HTML documents [3]. Therefore, access to the experience elements is provided by means of HTML links which, either directly or via a sequence of other HTML documents, lead to the desired experiences. In this context, we will introduce the term "*target document*": A link's target document is the document which is meant to be referenced by this HTML link.

Inconsistencies occur if:

a.) links refer to non-existing documents or

b.) existing documents are not referred by links and are therefore not accessible.

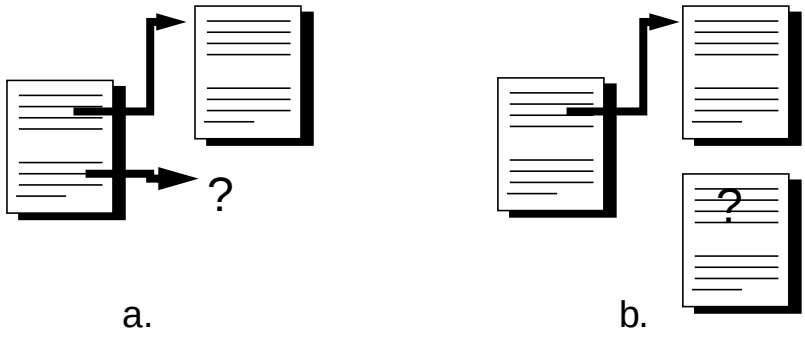


Fig. 2.1: Possible inconsistencies

There are several possible causes of inconsistencies. For example, links to non-existing documents occur if the target document's name is misspelled in the link definition, or the original target document is deleted or renamed. Existing but unreferenced documents obviously arise if they are (by mistake) not referenced at all. They can also occur if the documents were meant to be referenced, but there is a misspelling in the HTML link definition. In this case, the result would be a combination of a. and b.: One the one hand, there is a link to a non-existing document (the target document's name is misspelled in the link definition, and there is no document with this name); on the other hand, the target document is not referenced.

In addition, it is possible that the misspelled document name is equal to the name of another existing document. In this case, the latter would be referenced unintentionally. This should be considered as an inconsistency as well. But since the checking of consistency is meant to be performed by a tool, and because it is impossible for a tool to decide if the actually referenced document is the intended target document or not, this case will not be regarded any further.

2.2 Tool Support for Consistency Checking: Reuse of "HTMLCHECK"

Due to the large amount of experience elements stored in the experience base and the complexity of the HTML interface structure, a manual consistency check is not feasible. So, checking for consistency (i. e., the absence of HTML links to non-existing documents, and the absence of unreferenced documents) is intended to be done automatically. Since SFB-EB is maintained by a number of people who work independently, it is likely that inconsistencies occur. Therefore, the goal of consistency checking is not to prove that the EB is consistent, but to find inconsistencies in order to correct them.

The tool “htmlcheck” offers a functionality that is a generalization of the requirements of consistency checking for Special Research Project 501’s experience base. Htmlcheck was designed and developed by Oliver Flege [5]. It checks HTML-based information systems for inconsistencies of types a.) and b.) (see Section 2.1). The consistency check starts with the so-called “*homepage*”, a document that has been specified to be used as the starting point for the check. The tool then follows the homepage’s HTML links and checks the documents that are referenced by the links. After that, it follows the links found in these documents, and so on. The consistency check´s result is presented in the so-called “*result report*”, which is an HTML document generated by the tool. Htmlcheck is capable of identifying documents that have been checked already. Loops in the link structure will not lead to infinite loops in the program’s behavior.

In addition, htmlcheck is capable of checking if all documents of the information system are stored in a specific directory tree. This aims at discovering unwanted “scattering” of the documents in the directory structure. By restricting the storage of documents to a specific part of the directory structure, transferring the information system on a different computer is made easier, because all documents are stored in the same subtree of the file system. For SFB-EB this is important when the EB has to be transferred, e. g., onto a laptop for a presentation at a conference where remote access is not possible. Figure 2.2 depicts the desired and undesired distribution of experience base documents in the directory structure.

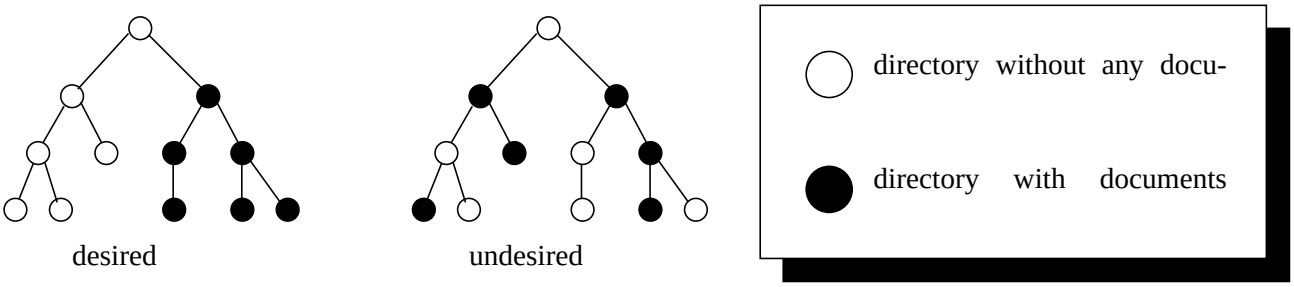


Fig. 2.2: Additional checking if all documents are part of the same directory tree

So, as htmlcheck is capable of the performing the required checks for HTML-based information systems, can it be appropriately modified for checking the experience base? Fortunately, htmlcheck allows adjustments to different environments by changing a configuration file. Therefore, adapting htmlcheck for checking the consistency of SFB-EB just means creating a suitable configuration file.

As far as consistency checking of the experience base is concerned, required entries in this file are:

- the name of the web server which provides access to the experience base (“sep1.informatik.uni-kl.de” in case of SFB-EB)
- definitions of alias names for directories that contain icons or scripts used by the web server

- the location (URL) of the homepage (the document which is the starting point for the consistency check)
- the result report's filename (this file will be created by the tool)
- the location (URL) of the result report (the report will be created by the tool)
- the absolute path to the root of the directory tree where all EB documents are supposed to be stored (this is for checking if all EB documents are part of this tree, the "desired" state in Fig. 2.2)
- files which are not to be checked for consistency: For example, version control of EB documents is done with RCS (Revision Control System) [6]. The RCS tool uses so-called rcs files to retain information on the changes made on specific EB documents. These rcs files are stored in the same directory tree where the EB documents reside. However, they are not documents that belong to the EB, so there are no HTML links which reference them. In order to prevent htmlcheck to wrongly report inconsistencies of type b.), rcs files have to be excluded from consistency checking.

Obviously, on the one hand, adapting htmlcheck for consistency checking is quite easy. On the other hand, there are a few deficiencies:

First, htmlcheck is not able to correctly check for consistency if the HTML-based information system uses *client-sided image maps.* An *image map* assigns an HTML link to a certain area of a graphical image (in contrast to assigning a link to the whole image). There are two possible kind of implementations for image maps: they can be server-sided or client-sided.
If a *server-sided image map* is used, and the user clicks the mouse button when the pointer is over the image, the client sends the mouse pointer´s pixel coordinates to the web server, and the server decides whether the coordinates are inside the area defined by an image map or not. In case of a *client-sided image map*, this calculation is done by the webclient (i. e., the browser).
Unfortunately, htmlcheck can only follow links that are defined via server-sided image maps. Thus, documents which are referenced via client-sided image maps will be reported as "not referenced" (because the tool thinks there is an inconsistency of type b.). But as image maps (which, in SFB-EB, are all client-sided in order to reduce net traffic) are only utilized in one place in SFB-EB, this does not prevent reuse of htmlcheck in this context: The limited usage of client-sided image maps allows manual checking without tool support.

Second, and more important, problems occur if htmlcheck encounters documents with HTML syntax errors. HTML uses so-called *tags* to control the overall appearance of a document and for the definition of HTML links. These tags have a special syntax. Often, there are tags that mark the beginning of a specific formatting statement ("start tag"), and there are corresponding tags which denote the end ("end tags"). For example, the "<A>"-tag marks the beginning of an HTML link definition, whereas "</A>" marks its end.

As web browser generally are rather forgiving in terms of incorrect syntax, the web pages displayed by them may be correctly displayed even if the syntax is flawed. For example, for some formatting statements it is common not to use end tags but rather implicitly mark the end of the previous formatting statement with the start tag of the new statement, and the web browser will still display which was intended.
Htmlcheck, on the other hand, is very restrictive in terms of HTML syntax. Fortunately, the tool provides assistance in locating syntax errors because it reports them (including the line number where they can be found) to the standard (error) output device if the configuration file contains a “show_activity” statement. Therefore, these syntax errors can be manually corrected.

2.3 Result Report

Because its deficiencies are only minor, htmlcheck is used for checking the consistency of the SFB-EB. As mentioned before, htmlcheck generates a result report in which the encountered inconsistencies are listed. The result report consists of several sections:

The first section is the “Configuration” section. It tells which configuration file was used and which document was used as the homepage (the starting point for the consistency checking). The second section is the “Table of Contents”. It is a list of HTML links that allow to directly jump to the individual sections which form the rest of the document:

- The “References to non-local Sites” section. This section lists HTML links which are references to external documents. For example, the EB contains a reference to the homepage of the SDL forum. This link to “www.sdl-forum.org” is a reference to a non-local site. As these links are not checked for consistency by htmlcheck, manual verification is recommended.
- The “Warnings” section which (among other warnings) reports links to non-existing documents (i. e., inconsistencies of type a.).
- The “Files outside Infostructure Directories” section lists documents that are referenced, but are not part of the directory tree specified in the configuration file. Therefore, this section reports undesired scattering of the EB (see Fig. 2.2).
- The “Files outside the Infostructure” section. This section lists the documents which are located in the specified directory tree but are not referenced. With the help of this section, inconsistencies of type b.) can be discovered.
- Four additional sections that provide further information.

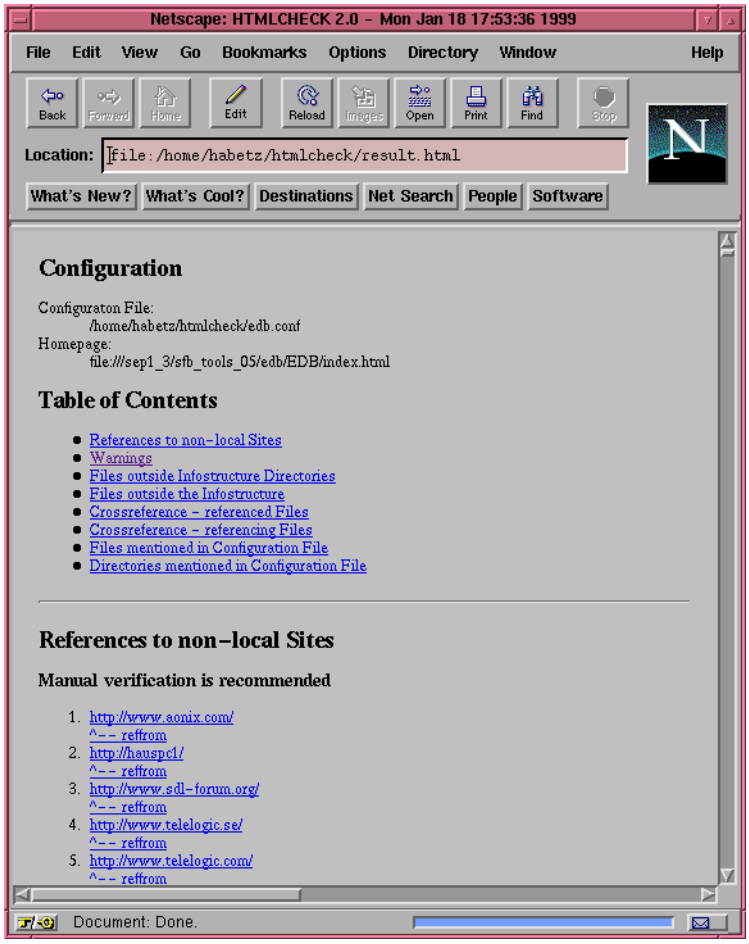


Fig. 2.3: The result report

Thus, with the help of the information provided by htmlcheck's result report, inconsistencies can be located. Using htmlcheck to check SFB-EB´s consistency has revealed a number of syntax errors in HTML documents as well as inconsistencies of type a.) and b.) which could be removed successfully.

3 USAGE PROFILES FOR SFB-EB

Besides checking the experience base' consistency, information on the overall acceptance of the experience base has to be gathered, i e., how frequently the experience base is used and which documents are most frequently accessed or, on the other hand, hardly accessed at all. This information on the EB's usage can give hints how to tailor the experience base to better meet the needs of the EB users. Again, there is a need for a tool which automatically gathers the appropriate information. Furthermore, the tool should display these information graphically.

3.1 Graphical Display of Usage Profiles

The number of accesses to the experience base is visualized by using bar charts to display the number of accesses to experience elements. There are three different types of graphs that are created:

- The "*Annual Overview*" (Fig. 3.1): Annual overview graphs display the number of accesses to experience elements in each month of a given year. For each year, exactly one annual overview is created.

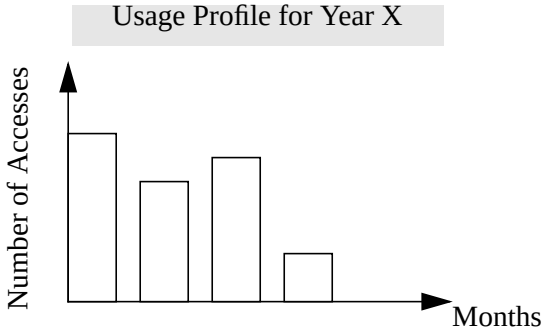


Fig. 3.1: Annual overview

- The "*Monthly Overview*" (Fig. 3.2): The monthly overview is a per-month comparison between the number of accesses to experience elements and the numbers of accesses to the individual tools of the Software Engineering Laboratory in a given year. For each year, this kind of graph is created for every month of this year.

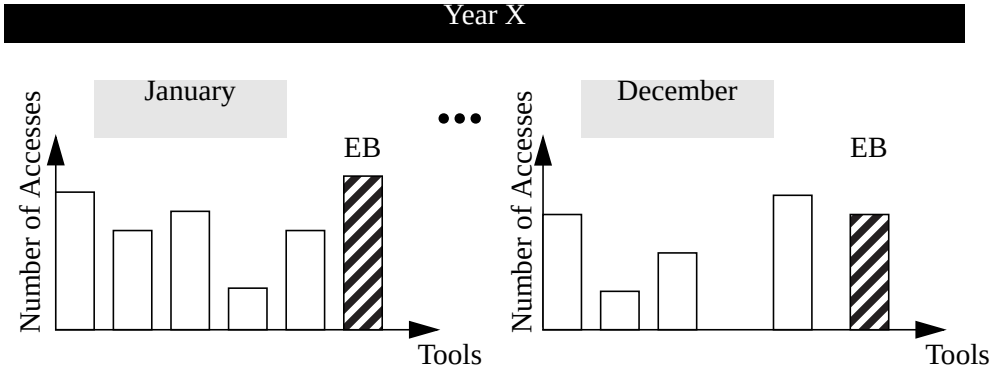


Fig. 3.2: Monthly overview

- The "*Top 20 Graphs*" (Fig. 3.3): These graphs display the number of accesses to those 20 experience elements that have the highest number of accesses. There are actually two subtypes which differ in the lapse of time used to determine the top 20 ranking. The ranking is done

  a.) per month ("*monthly Top20 graph*") and

b.) for the whole year (“*annual Top20 graph*”))

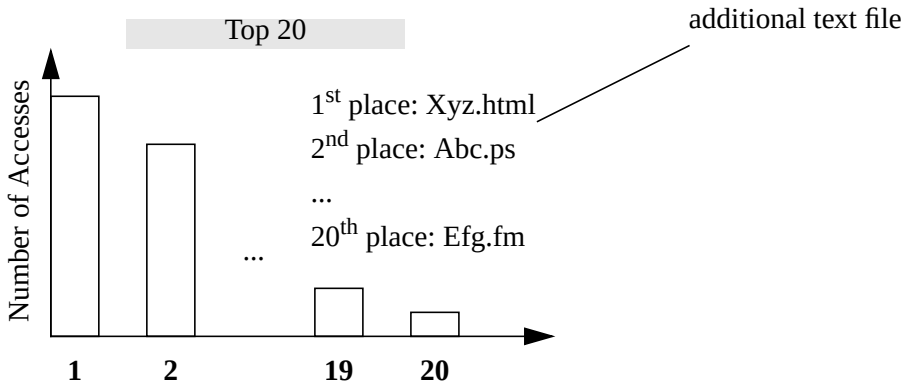


Fig. 3.3: Top 20 graph (annual / monthly)

For each year, there is exactly one annual Top 20 graph, and a monthly Top 20 graph for each month. Since the bar chart does not display the document names of the experience elements, an additional (automatically generated) text file is created, which relates the numbers in the Top20 ranking to the respective experience elements.

In addition to these mandatory text files for the Top20 graphs, the tool creates text files which list all experience elements that are accessed, together with the respective number of accesses for each experience element.

3.2 Data Gathering

Clients access the experience base by sending requests to a web server. The web server then fulfills these requests and transfers documents to the client. The client sends requests using the HTTP (“Hypertext Transfer Protocol”). Tab. 3.1 lists the HTTP methods defined in the current HTTP version. Requests that occur when accessing SFB-EB are shaded in grey.

Tab. 3.1: HTTP methods

HTTP-Method	Semantics
OPTIONS	get information on the server’s communication options
GET	request a document
HEAD	request meta information on a specific document
POST	submit data to the server
PUT	deposit documents on the server
DELETE	delete documents on the server
TRACE	send a special message to the server, which will be sent back to the client (for test and diagnosis purposes)

The GET method is used whenever the user demands a new document. The POST method is used when user interaction is required (e. g., for search and retrieval operations, where the user must state the search term).

After submitting its request, the client receives a *status code*. The status code tells the client about success or failure of the server to process the request. Status codes are three-digit numbers. The first digit is used to divide the status codes into different classes. Codes that start with the same digit belong to the same class. Tab. 3.2 gives a list of possible status codes. Only status codes starting with “2” indicate a successful completion of the client’s request. Nevertheless, status code 304 indicates that the request can be fulfilled by retrieving the requested document from the client’s cache, so even though the server transmitted no data, the access to the document was successful. Therefore, any status code of class “Successful” or status code 304 indicate accesses to experience base documents that must be taken into account for creating the usage profiles. To emphasize the importance of these classes for the usage profile, the corresponding entries in Tab. 3.2 are shaded in grey.

Tab. 3.2: HTTP server status code classes

Status codes	Class	Semantics
1xx	Informational	Preliminary replies by the web server that are sent before the request is (completely) fulfilled, e. g., telling the client that the request has not been rejected so far (Code 100 “Continue”)
2xx	Successful	Status codes that signalize successful reception, comprehension and acceptance of the client’s request
3xx	Redirection	Further actions by the client are necessary to fulfill the request. For example, this is the case if a document’s URL has changed (Code 301 “Moved Permanently). Then, the client has to rerequest the document via the new URL provided by the server (this rerequest is usually done automatically by the user’s web browser)
4xx	Client Error	Indicates user errors, e. g., bad request syntax (Code 400 “Bad Request”).
5xx	Server Error	Describes server errors, for example “Internal Server Error” (Code 501)

In order to create the usage profiles, the tool must know which experience elements have been accessed in a given lapse of time (in this context, this can be a certain month or year). In order for the tool to access this data, it obviously has to be gathered first. This data gathering is supported by the web server: the server logs every data transfer in a “*TransferLog*” text file using the format depicted in Fig. 3.4.

<host> <ident> <authuser> <date> <request> <status> <bytes>

Identifier	Semantics
<host>	Complete domain name of the client (or IP number if the name is not available)
<ident>	Client-sided login name of the person who sent the request to the server (only available under certain conditions)
<authuser>	User id (when requesting password-protected documents)
<date>	Date and time of the request In this context, the following format is used: *<date> = [DD/MMM/YYYY:hh:mm:ss zone]* DD: number of day (01..31) MMM: month name (Jan...Dec) YYYY: year (z.B. “1998”) hh hour (00..23) mm: minutes (00..59) ss: seconds (00..59) zone: time zone, in format *(‘+’|’-’) zzzz* Example: +0200
<request>	HTTP request by the client, enclosed in double quotes. In the context of SFB-EB GETs and POSTs are the only possible requests.
<status>	Status code reply by the server
<bytes>	Number of bytes sent to the client in reply to the request

Fig. 3.4: TransferLog entry format

The following line illustrates how an entry in the TransferLog will look like:

computer.domain.de - EB-login [01/Apr/2386:08:15:42 +200] “GET document.html HTTP/1.1” 200 79

The example is a log entry for a successful transmission (status code 200) of the file “document.html” of length 79 bytes to the client “computer.domain.de”. The request was issued by the EB user “EB-login” (which in reality is not a valid user name for the experience base) on April 1, 2386 at 8 hours, 15 minutes and 42 seconds. The symbol “-” in the second field of the entry denotes that the user name on the user´s computer (the user’s login name on this host, in contrast to the login name for the experience base in the third field) could not be retrieved.

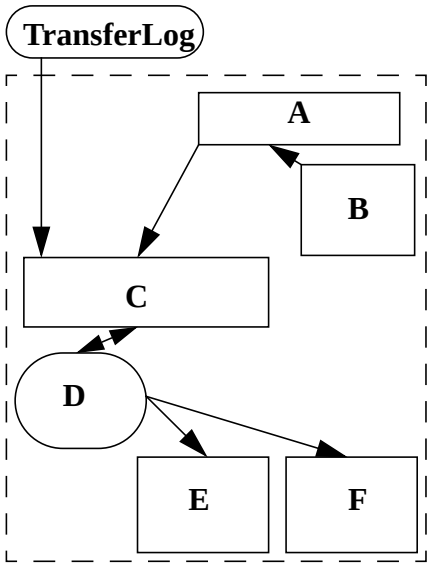
3.3 EBStats: A Tool for Generating SFB-EB´s Usage Profiles

As all relevant information for the usage profile is recorded and available, generating the profiles from this information is straightforward. All request that were fulfilled successfully (status code 2xx) or could be redirected to the client's web browser cache (status code 304) are being taken into account. And because the information includes the date of the request and the requested document name, the required annual and monthly overviews can be easily generated. However, due to the technical realization of the SFB-EB, experience elements and layout elements (e. g., background images) for the HTML interface are stored together. These layout documents must not be included into the analysis. Fortunately, layout elements are stored in specific directories so that all documents residing in these directories can be (and are) excluded from the usage profile.

The tool for creating the EB's usage profile is called EBStats (abbreviation for "Experience Base Statistics"). It is implemented using a combination of the unix C shell and the AWK script languages. These languages were chosen for implementation for two reasons:

- The tool identifies and processes patterns in text files. It also creates text files (in addition to the bar charts). Scripting languages like Perl or AWK are specialized on these tasks and provide suitable programming statements. This allowed developing the tool in a shorter time, and development was less prone to errors, since most of the required pattern extraction and manipulation functions were available and did not have to be developed from scratch.
- The C shell and AWK scripting languages were used for the implementation of an already existing prototype implementation for creating the SE Lab usage profiles. Because creating usage profiles for the experience base and for the software engineering laboratory are very similar tasks, part of the prototype implementation's code could either be directly reused, or used as a framework for new code.

EBStats is divided into several components, where each component has a distinctive role (see Fig. 3.5). Component A is responsible for command line parsing. The tool can be executed in two modes. The mode of execution is chosen by command line parameters, which are evaluated by component A. *Reconstructive mode* is used for reconstructing usage profiles for years that are already over. In this mode, the tool looks for data ranging from January to December of the specified year. In current mode, the tool determines the current date (via component B) and creates the usage profile from the beginning of the current year up to the last day of the previous month. The fact that data on EB usage is only gathered since November 1997 is taken into account when calling the tool in reconstructive mode. Component C extracts the relevant information from the TransferLog file and creates the usage profiles. The usage profile data is stored in the so-called "profile database", which is in fact a collection of (text) files that store the information retrieved from the TransferLog (i. e., the names of the experience elements and the respective number of accesses per month and year). So, when executing EBStats, information already gathered in a previous



Components of EBStats

- A: Command line parsing
- B: Determines the current date
- C: Generates usage profiles for SFB-EB
- D: Stores data of past usage profiles for reuse
- E: Creates text files
- F: Visualization

Fig. 3.5: Data flow between EBStats' components

analysis is directly available and does not have to be redetermined by retrieving it from the TransferLog file.

For example, if EBStats is invoked in current mode at the beginning of each month, it reuses information when it creates annual overviews. A new annual overview is basically the previous annual overview plus the respective number of accesses in the month that had not been over yet when the previous annual overview was done. So, EBStats reuses the data of the previous annual overview, which is stored in its profile database. Reuse of information is also utilized in reconstructive mode: If bar charts or text files created by EBStats were previously generated but somehow were lost (e. g., deleted by mistake), the tool recreates them using the information in its profile database (the relevant information was stored there in the previous tool call). If relevant information is missing in the profile database (but was supposed to be there), the tool first gathers this information from the TransferLog file and then stores it in the profile database.

Finally, the information in the profile database is used to create the required text files, which is done by component E. The usage profile is visualized by using the public domain tool "gnuplot" (component F).

3.4 SFB-EB Usage Profile Examples

While Fig. 3.1 to Fig. 3.3 only illustrate the general layout of the charts, Fig. 3.6 and Fig. 3.8 were created by EBStats. Fig. 3.7 was created by LABStats (see Section 4.4 for details).

Please note (see Fig. 3.7) that in monthly overviews, the names of SE Lab tools (x-axis of the bar chart) are abbreviated for the purpose of proper display. This is done automatically by LABStats, which also generates a text file relating abbreviations to the full names (e. g., "Fr1" stands for FrameMaker and "EB" for Experience Base).

Also, for the purpose of proper display, EBStats does not list document names in Top20 graphs. So, the x-axis displays only the ranking (1 - 20). As mentioned in Section 3.1, EBStats creates text files for Top 20 Graphs which relate ranking positions to EB documents.

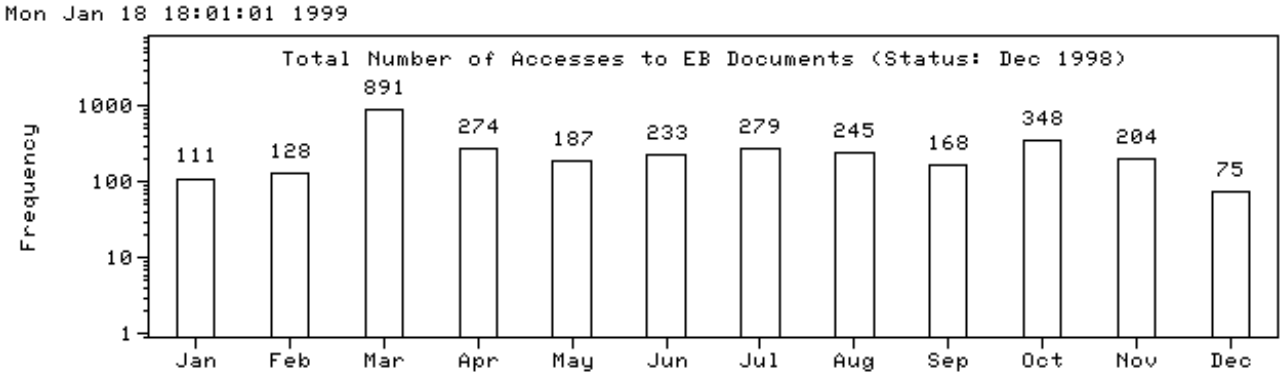


Fig. 3.6: Annual overview for 1998

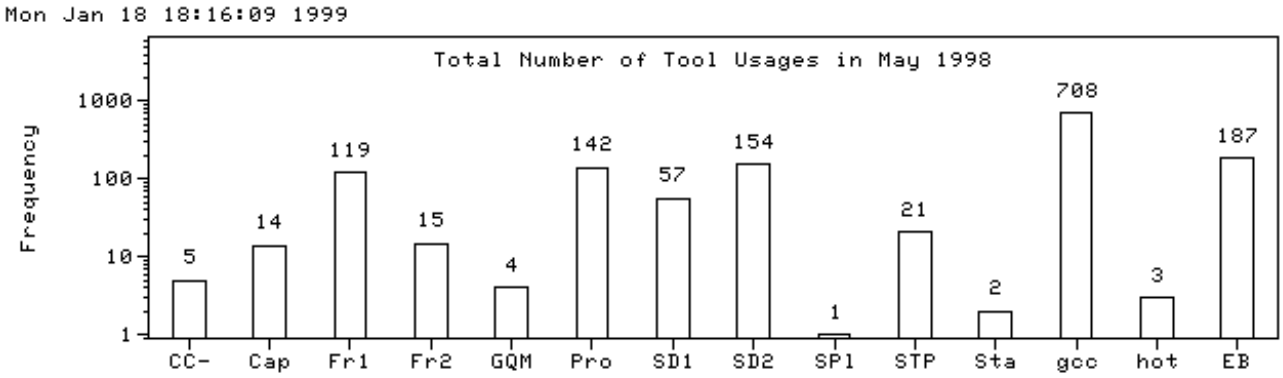


Fig. 3.7: Monthly overview for May 1998

Usage profiles created by EBStats are accessible on-line via
http://wwwagse.informatik.uni-kl.de/Projects/SFB_Subprojects/A1/EQUIPMENT/usageprofiles.html
in the “Experience Base” sections.

4 USAGE PROFILES FOR TOOLS PROVIDED BY THE SE LAB

Creating usage profiles for the SE Lab is closely related to the task of creating usage profiles for SFB-EB. However, the main focus of the SE Lab usage profiles is to find out if some tools are hardly used or not used at all. By

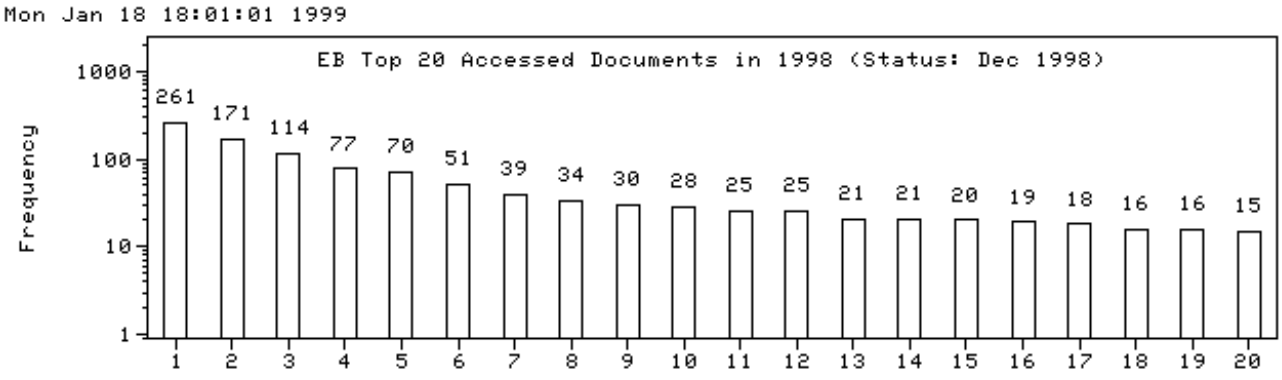


Fig. 3.8: Annual Top 20 graph for 1998

gathering this information, it is easier to decide if a tool should be installed on the SFB 501´s SE Lab computers any longer. Tool support for the creation of these usage profiles is required.

4.1 Graphical Display of Usage Profiles

Visualizing is done by creating bar charts that display the number of tool usages in the software engineering laboratory. There are two different types of graphics:

- The *"Tool Specific Annual Overview":* These bar charts display the number of usages of a specific tool in each month of a given year (see Fig. 4.1). For each year and each tool installed in this year, a tool specific overview is created. It is important to include all *installed* tools in the analysis rather than only taking the ones into account that are actually *used*, because one of the main goals is to detect unused tools.
- The "*Annual Overview*": The annual overview displays the number of tool usages in the SE laboratory, i. e., for each installed tool, there is a bar in the bar chart that displays the total number of tool usages in the given year (see Fig. 4.2). For each year, there is exactly one annual overview bar chart.

4.2 Data Gathering

Before a usage profile can be created, the appropriate data has to be gathered first. Whenever a tool is called in the software engineering laboratory, the tool usage is noted and written into a log file. In order to achieve this, tools are not called directly, but via UNIX shell scripts that create the appropriate log entries and then call the actual tool. These script files are called "*start-scripts*". For each tool, there is a related start-script.

There are two types of start-scripts: start-scripts of type "*run*" and start-scripts of type "*start/stop*". Start-scripts of type "run" create exactly one entry in the log file, whereas start-scripts of type "start/stop" create two entries.

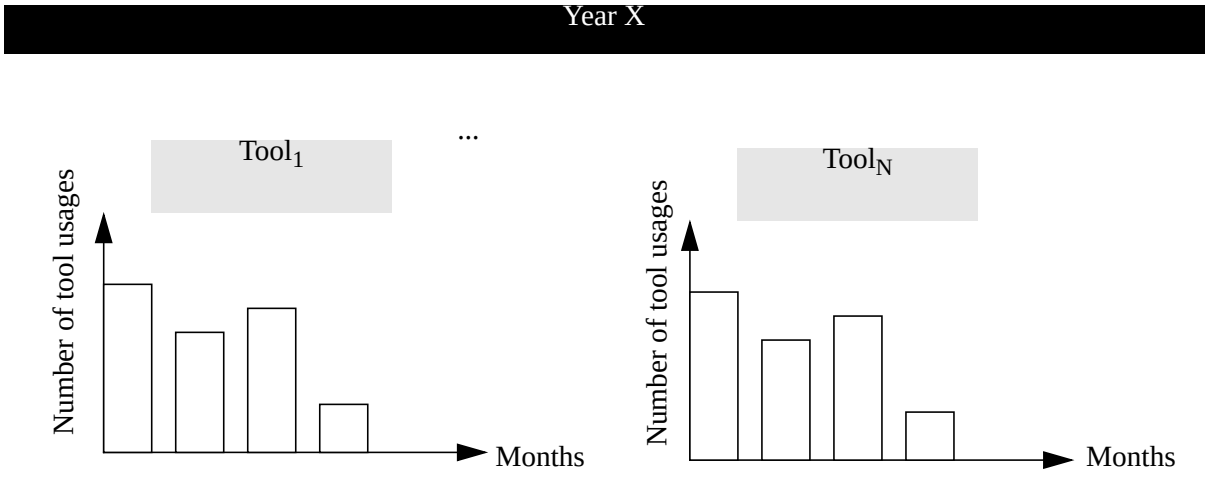


Fig. 4.1: Tool specific annual overview

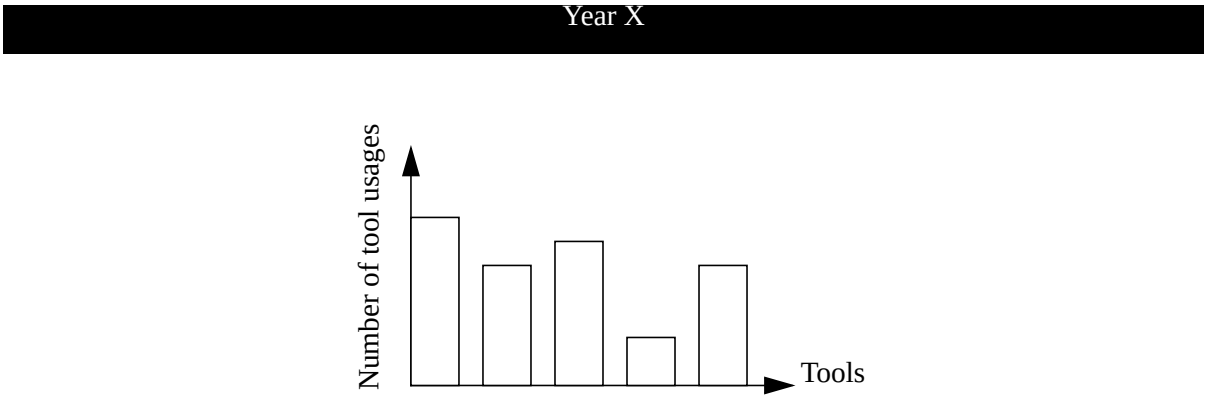


Fig. 4.2: Annual overview

Originally, when the logging mechanism was introduced, it was intended not only to log how *often* the tools were called, but also how *long* they were used. Therefore, the start-script would create two entries in the log file, one when the tool was started and one when it was closed. This is done by start-scripts of type “start/stop”.

However, for some of the tools installed in the SE Lab, it is not possible for the start-script to determine the point in time when the tool usage is finished (some of the tools use multiple threads, and the start-script can only supervise the thread that started the tool). So, in this case, only the starting point of the tool usage can be recorded, and start-scripts for these tools are of type “run”.

Start-scripts of type “run” create one entry in the log file. They use the following format:

YYYYMMDD:xxxxxx:hhmmss:<status>:<toolname>:<comment>

YYYY	*Year (1970...)*
MM	*Month (01..12)*
DD	*Day (01..31)*
xxxxxx	*Process-ID of the start-script (000000..999999)*
hh	*Hours (00..23)*
mm	*Minutes (00..59)*
ss	*Seconds (00..61)*
status	*“run”*
toolname	*Name of the tool called by the start-script*
comment	*Comment (optional)*

where the sequence “YYYYMMDD” denotes the date of the tool usage and the sequence “hhmmss” the time when the tool was called.

Start-scripts of type “start/stop” create two log entries. The format is the same as the format for entries of type “run”. The only difference is that <status> is “start” or “stop” instead of “run”. The start entry gives date and time when the tool was started. In case of the “stop” entry, “YYYYMMDD” and “hhmmss” denote date and time when the tool was stopped.

Fig. 4.3 shows an excerpt of the logfile.

4.3 LABStats: A Tool for Generating Usage Profiles for the SE Lab

All relevant information for generating usage profiles for the development tools provided by the SE Lab (i. e., which tool is called and when it is called) can be retrieved from the log file. However, since the log file is periodically split into smaller files (where each file only contains the data for a single month), the tool operates on these split log files.

Furthermore, there is an existing prototype of a tool that creates a usage profile for the SE Lab, so its design (and some of its code) could be used for the development of “LABStats” (which stands for Software Engineering Laboratory Statistics). And as generating usage profiles for SFB-EB and for its software engineering laboratory are similar tasks, a lot of the design and code of EBStats could be reused as well. Therefore, the design of LABStats resembles EBStats (see Fig. 4.4).

In general, the overall behavior of LABStats is very similar to that of EBStats. For this reason, we will only give a brief description of its components: Like EBStats, LABStats is divided into several components. It can be con-

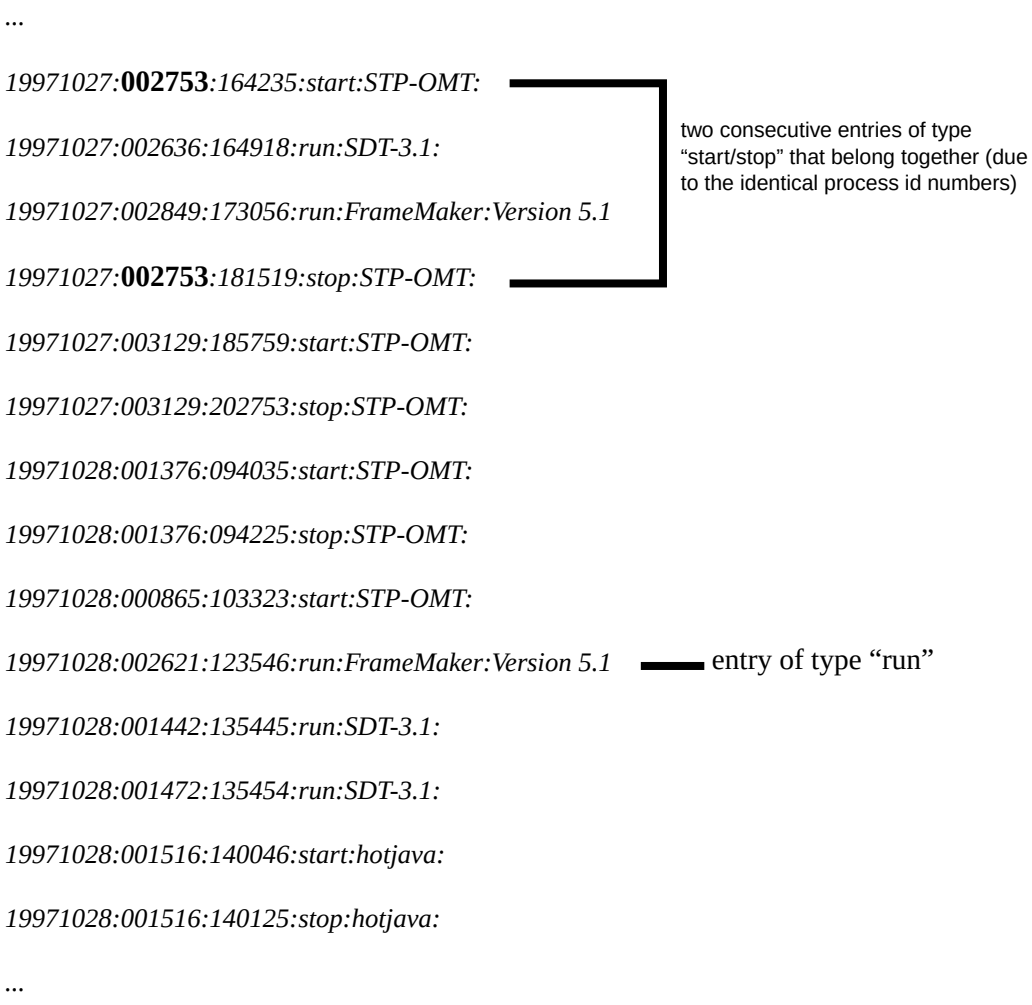


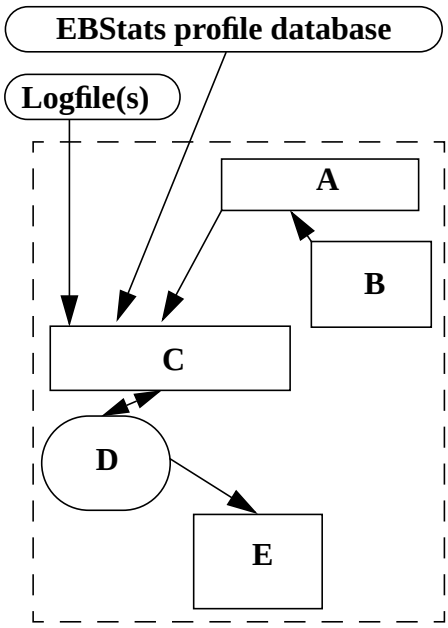
Fig. 4.3: Log file example

trolled via command line options, and can be executed in reconstructive and current mode. Component A of EBStats could be reused without any modification. The same is true for component B, which determines the current date.

Component C is responsible for generating the usage profiles. Besides extracting the tool calls from the log file, it has to determine which tools are currently installed, in order to detect tools that are installed but not called. This is achieved by extracting the names of the installed tools from the start-scripts (here is a start-script for each installed tool).

When LabStats creates the usage profiles, it only incorporates log file entries of type “run” or “start” into the analysis (entries of type “stop” only mark the end of the tool usage). This is major difference to the existing prototype, because the prototype did not distinguish between start and stop entries and counted them both.

Component D permanently stores past usage profiles in the file system. In addition to the number of tool usages for



Components of LABStats

- A: Command line parsing
- B: Determines the current date
- C: Generates usage profiles for the SE Lab
- D: Stores data of past usage profiles for reuse
- E: Visualization

Fig. 4.4: Data flow between LABStats' components

a specific month or year, component D also stores which tools were installed in this month or year.
Finally, component E performs the visualization. Again, the tool gnuplot is utilized for this task.

4.4 SE Lab Usage Profile Examples

LABStats generates the bar charts described in Section 4.1. In addition to this, it generates the monthly overviews mentioned in Section 3.1. The reason for this is simple: The monthly overview bar charts are comparisons between the tool usages in SFB 501's SE Lab and the number of accesses to SFB-EB. So, it is less effort for LABStats to determine the number of accesses to the EB from EBStats' profile database than for EBStats to determine the number of tool usages for all installed tools in the SE Lab. Therefore, LABStats accesses EBStats' profile database (as depicted in Fig. 4.4) to create the monthly overview bar charts.

Fig. 4.5 and Fig. 4.6 are examples of bar charts created by LABStats. An example for the monthly overview can be found in Section 3.4 (see Fig. 3.7). Again, the tool creates abbreviations for the tool names and generates text files which relate abbreviations to tool names.

Usage profiles for the software engineering laboratory of Special Research Project 501 are accessible on-line via
http://wwwagse.informatik.uni-kl.de/Projects/SFB_Subprojects/A1/EQUIPMENT/usageprofiles.html
in the "Software Engineering Laboratory" section.

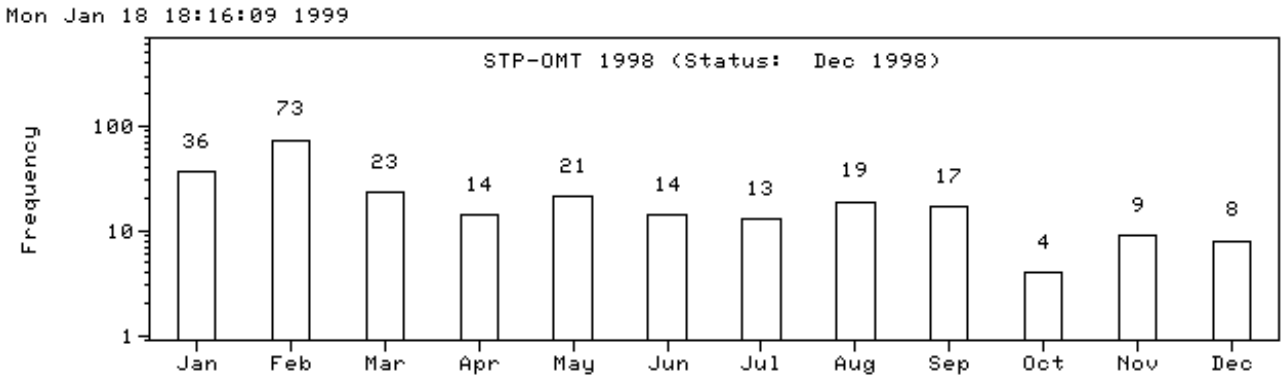


Fig. 4.5: Tool specific annual overview for STP-OMT in 1998

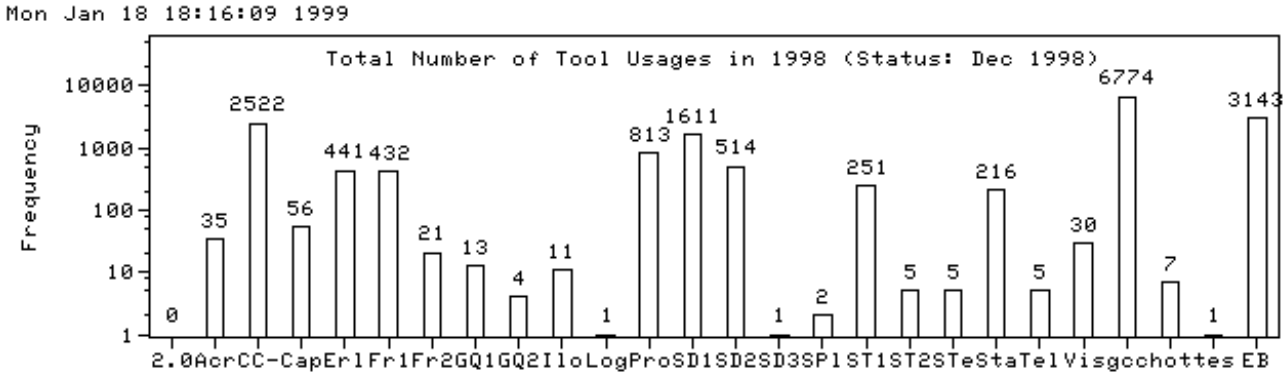


Fig. 4.6: Annual overview for 1998

5 CONCLUSION

In this chapter, we will give a summary and provide an outlook for future use of the described tools.

For *consistency checking* of SFB-EB, the tool htmlcheck was reused.
Htmlcheck's configuration file was adapted to the experience base environment. The tool generates a result report and lists the detected inconsistencies.

For generating *usage profiles for the SFB-EB*, the tool EBStats was developed. Part of the tool was developed by reusing a prototype implementation for a tool generating usage profiles for SFB 501´s SE Lab. EBStats consists of six components, which are responsible for command line parsing, determining the current date, creating usage profiles, permanent storing of usage profiles, creation of text files and visualization, respectively. The tool can be executed in two modes. A reconstructive mode is used for analysis of EB accesses of past years and utilized if data has

been lost or not created yet. Current mode is used for analysis of EB accesses in the current year. Mode of execution is chosen by command line parameters. Analysis of accesses in past years is performed for the whole year, where the fact that data on EB accesses is available since November 1997 only is notified by the tool. Analysis for the current year is performed from January 1st up to the last day of the previous month.
The usage profile data is extracted from a server log file, the so-called “TransferLog”. This file lists all accesses to the experience base. The usage profile is stored in the profile database, which consists of a collection of text files in the file system. This information is reused in future calls of EBStats to prevent unnecessary reanalysis of the TransferLog. Only access data for months that are not (yet) covered in the profile database is extracted from the TransferLog file and then incorporated into the profile database. Text files providing additional information are created, and the usage profile is visualized using the tool “gnuplot”.

LABStats, the tool for generating *usage profiles for the software engineering laboratory,* is very similar to EBStats, both in respects of design and functionality. It was developed by reusing design and code of EBStats as well as code from a prototype implementation for SE Lab usage profile generation. Because only bar charts and no text files providing additional information were required, the tool consists of five components, compared to six for EBStats’ realization. The five components are responsible for command line parsing, determining the current date, creating usage profiles, permanent storing of usage profiles and visualization. Like the tool for the experience base, LABStats can be executed in reconstructive and current mode. Analysis is based on a log file that lists tool calls, and information is extracted from start-scripts to determine which tools are currently installed (i. e., in case of a current analysis). Furthermore, the tool utilizes a profile database for permanently storing usage profiles and information about which tools were installed in a given month or year. The profile database is used in the same way as described for the tool EBStats. Again, the usage profile is visualized with gnuplot, and LABStats additionally creates bar charts required for the experience base usage profile (see Section 4.4 for details).

As mentioned in Section 1.2, the technical realization of SFB-EB will change in the near future. Currently, efforts are made to create a new implementation of the EB based on an object-relational database management system (ORDBMS). This leads to the question which impact this will have on the three tools presented in this report.

The tool for consistency checking of the experience base will presumably not be used in the future technical realization of the EB. As database management systems have their own mechanisms for consistency checking, there will be no need for an additional tool. However, it should be used until the ORDBMS implementation is completed in order to ensure consistency until then.

The tool for creating the usage profile for the experience base, EBStats, can still be used if the database system or the webserver is able to provide similar data on user accesses. If such information is available, the tool can be eas-

ily modified. Because EBStats uses its profile database for analysis and visualization, the main functionality of the tool can remain unchanged. Only the part which is responsible for extracting the information from the log file has to be changed if the logging mechanism does not remain the same.

In general, the tool for creating the usage profile for the software engineering laboratory, LABStats, is not affected. For the creation of the monthly overview, the tool utilizes EBStats' profile database, which does not have to be changed. Problems only occur if the ORDBMS version of the EB will be installed on a different computer than the rest of the tools of the SE Lab. This means that EBStats would also have to be installed on this new computer, and EBStats' profile database would be inaccessible by LABStats, which has to be on the same computer where the rest of the SE Lab tools are.

There are two solutions to this problem: On the one hand, it would be possible to refrain from comparing the number of tool usages to the number of accesses to the EB. LABStats can also generate the monthly overview if data on the number of accesses to EB documents is missing; in this case, it sets the number of accesses to the EB to zero. At last, on the other hand, it should be possible to (automatically) transfer the appropriate data files from EBStats' profile database to the computer where LABStats is located.

ACKNOWLEDGMENTS

This work has been conducted in the context of the Sonderforschungsbereich 501 ‘Development of Large Systems with Generic Methods’ (SFB 501), funded by the Deutsche Forschungsgemeinschaft (DFG). I would like to express my gratitude to my supervisor Raimund L. Feldmann and our project leader Prof. Dr. H. D. Rombach.

REFERENCES

[1] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach:
Experience Factory.
In J. J. Marciniak (ed.), Encyclopedia of Software Engineering, Volume 1, John Wiley & Sons, 1994, 469–476.

[2] Andreas Birk and Carsten Tautz:
Knowledge Management of Software Engineering Lessons Learned.
In Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE‘98), pages 116–119, San Francisco Bay, CA, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.

[3] Raimund L. Feldmann and Stefan Vorwieger:
Providing an Experience Base in a research Context via the Internet.
In Proceedings of the ICSE 98 Workshop on “Software Engineering over the Internet”, On-line at: http://sern.cpsc.ucalgary.ca/ maurer/ICSE98WS/ICSE98WS.html, April 1998

[4] Progress Report of Sonderforschungsbereich 501 (1995 - 1997), (*in german)*
http://sep2.informatik.uni-kl.de:8080/localdocs/Report1997.html

[5] Oliver Flege:
Consistency Checking of Hypermedia Systems (*in german*).
Projektarbeit, Dept. of Computer Science, University of Kaiserslautern, Germany, 67653 Kaiserslautern, Germany, August 1994

[6] Walter F. Tichy:
RCS - A System for Version Control.
Software–Practice and Experience 1985; 15(7):637–654