A Software Defect Report and Tracking System in an Intranet

A. Monteiro, A.B. Almeida
DAMAG - Portuguese Navy
DAMAG, Praça do Município,
1118 Lisboa Codex, Portugal
bigotte.almeida@mail.damag.pt

M. Goulão, F. B. Abreu, P. Sousa
INESC
R. Alves Redol, 9,
1017 Lisboa Codex, Portugal
{miguel.goulao | fba | pedro.sousa}@inesc.pt

Abstract

This paper describes a case study where SofTrack - a Software Defect Report and Tracking System - was implemented using internet technology in a geographically distributed organization. Four medium to large size information systems with different levels of maturity are being analyzed within the scope of this project. They belong to the Portuguese Navy's Information Systems Infrastructure and were developed using typical legacy systems technology: COBOL with embedded SQL for queries in a Relational Database environment.

This pilot project of Empirical Software Engineering has allowed the development of techniques to help software managers to better understand, control and ultimately improve the software process. Among them are the introduction of automatic system documentation, module's complexity assessment and effort estimation for maintenance activities in the organization.

1. Introduction

Software process improvement (SPI) is currently viewed as a means to achieve a higher quality in the developed products at a lower cost and within schedule [10]. An investment in the achievement of quality may lead to an effective cost reduction due to the decrease of costs associated with lack of quality [6].

At DAMAG, a department responsible for the information systems infrastructure and also for analyzing, developing and integrating management support methodologies within the Portuguese Navy, a SPI initiative is currently being carried out.

Before the implementation of the actions described in this paper, most of the defects found either in-house (by black box testing) or by final users, were simply handled on the telephone and no record was made. The systems under study were built with typical legacy systems technology. They exhibit common fragilities such as unstructuredness, unmanaged complexity, insufficient documentation and unexpected impact of changes. Forecasts of short to medium term effort required to support the software systems were not available. This was particularly problematic since the members of the teams in charge of the systems' evolution often have regular military assignments (e.g. naval exercises at sea) and those must be planned with some advance. On the other hand, without on-line information of pending actions, the heads of the Software Division could not have control over the ongoing projects. Ultimately, the final users had no mechanism of feedback on their submitted requests.

To overcome this *status-quo* a DAMAG/INESC (a R&D institute) joint action was set up to define the architecture and develop a defect and reporting system that could be used in the Navy Intranet. The SofTrack tool was then developed.

Simply put, SofTrack is a system that combines the features of a software defect report tracking system with the collection, analysis and presentation of software process and product metrics. Moreover, it also embeds a system responsible for automatically generating documentation on a software product based on the analysis of its source code, thus contributing to the existence of updated product documentation.

The complexity metrics are combined in a proposed software complexity model. The metric-based quantitative analysis has allowed identifying error-prone modules due to their extreme complexity. A restructuring activity is currently underway to deal with this problem.

Effort estimation based on the complexity of the evolution actions is another topic that will be addressed in this paper.

SofTrack has a web interface that allows users spread over distinct buildings to have a view of the evolution of a software system tailored to their activity.

Although there are some other web based software evolution tracking systems that provide a strong coverage of evolution tracking[2], there is still some work to be done in what concerns the integration of these systems with tools that deal with the automatic documentation of the

maintained systems implementation and with the issue of software complexity assessment. We feel the combination of these features may strengthen the SPI initiatives.

SofTrack is being tested with four applications developed by different DAMAG teams with COBOL and SQL/DS in a proprietary system during the last decade. The data collected is expected to give a reasonable overview of the main problems detected in this kind of systems within the organization.

2. Global architecture

The conception and acceptance of SofTrack was not an easy nut to crack. It raised technological, methodological and cultural problems. The SofTrack architecture (Figure 1) combines the usage of adapted commercial of the shelf software (COTS) with software developed by INESC and DAMAG.

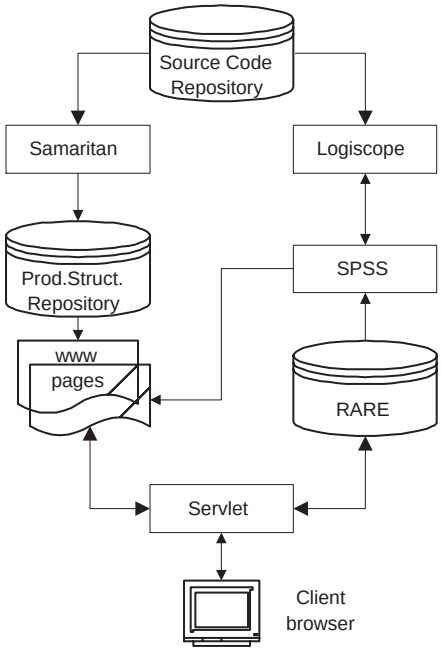

Figure 1. SofTrack Architecture

The physical support of the SofTrack in the Portuguese Navy Intranet, lies on a web server running Windows NT with the Internet Information Server (IIS) and on a Unix workstation. The IIS is enhanced with the usage of a commercial add-on (ServletExec for Windows) to enable the support of Java servlets.

The source code files stored in the *Source Code Repository* provide the data for the product analysis supported by SofTrack. They are submitted to *Logiscope* [13] and *Samaritan* [3] (System for Aided MAintenance and Reengineering of Information Technology ApplicatioNs) tools, both running on the workstation.

Logiscope is a COTS tool responsible for extracting information about the overall structure of an application (call graph), the logical structure of its components (control graph) and measures of its complexity (product metrics). *Logiscope*'s outputs are the inputs for both *Samaritan* and *SPSS* [12] (Statistical Package for Social Sciences).

Samaritan is a tool responsible for the generation of updated documentation of the systems under analysis, based on their source code. The produced documentation is organized in the *Product Structure Repository* and presented in *HTML pages* to allow an easy browsing through them.

The process data is collected with *RARE* (Register and Analysis of Requests of Evolution [4]), a framework for evolution actions tracking. The collected data allows the computation of process metrics.

The *SPSS* tool is used in the statistical analysis of the product and process metrics. This is achieved with SPSS scripts that are run with a defined periodicity, generating HTML pages containing reports with project control information.

The client-server structure of SofTrack is implemented with the usage of a *SessionServlet*, which is responsible for dealing with the interaction performed between clients and the system, on the server side. It implements the security policy of SofTrack, granting customized privileges to each user.

3. Product structure - SAMARITAN

The Samaritan is a tool developed at INESC to document the implementation of information systems. It analyses the system components and produces a dependency graph between physical artifacts such as executable images, source files, functions, databases, data base tables, and attributes of database tables.

Each node of the graph describes a single physical artifact and each edge establishes a dependency between two artifacts.

The graph is stored in the Samaritan internal repository and can be accessed for different purposes, such as complexity computation and impact analysis. In large information systems, the graph is often in the scale of hundreds of thousands of nodes and edges.

To manage the complexity and fully explore the dependency graphs, Samaritan allows users to generate customized sub graphs in a hypertext format, where nodes are HTML pages and edges are hypertext links between pages.

The current version of Samaritan is able to parse source code in C and Cobol languages (in SUN, HP and IBM

systems) with embedded SQL statements for ORACLE, INFORMIX, SQL/DS and DB2 databases. It also parses Oracle stored procedures, forms and triggers in PL/SQL language.

4. Report disclosure problem

SofTrack's report generation activity is automated, making updated information available to its users.

SofTrack maintains information about access rights of its users. At a given point in time, the user will have access to the reports contained in the groups of reports on his access list and also to the reports accessible through the groups of users to which he belongs. These access lists are defined according to a data disclosure policy that filters the relevant information for each user profile. To support the accessibility restrictions, the web server is extended so that it grants or denies access to pages according to the user's access list.

SofTrack has a java servlet (SessionServlet) that is responsible for controlling data access. It enhances the HTTP server by enabling request/response services. When a client sends a request to the web server, the web server redirects the request information to the servlet and have the servlet construct the response. Because all user information is passed to the servlet as part of the HTTP request, it implements additional authorization as part of its service method.

Once the user logs in the SofTrack, a cookie is set to carry his information inside the system. The servlet uses this cookie to dynamically generate the available links on the pages it serves back to the client. From here on, the whole environment is configured for that particular user, only showing him the available resources for his profile.

When the servlet receives a serve page request, it checks if the user has access to that URL and serves it back, providing he does have access. This double check on the accessibility of the page is performed to prevent users from composing an unauthorized page request.

With the described approach, the SofTrack is able to fulfill its requirement of restricted access to data.

5. Process analysis - RARE

The purpose of collecting process metrics is the detection of eventual process flaws, pitfalls or improvement opportunities and the monitoring of the product's quality, to support the SPI.

Measuring the process requires the help of the people involved in it, normally by registering how much effort they spent doing some maintenance activity and when it was done. The normalization and classification scheme of the evolution requests used in the realm of this experiment was developed with the support of the maintenance team in order to keep it simple to use. It includes a taxonomy for the evolution action itself, along with process data such as the effort spent in an evolution action and links to track the changes back to the source code.

RARE is the basis for the tracking of the evolution of the software. The managers use it to control the process and quality of the product, through the analysis of reports such as control charts, medium time between failures and medium time to repair. The structure of the RARE database makes it easy to get more specific reports by filtering the actions through some characteristic feature.

6. Module complexity assessment

6.1. Definition of an assessment model

Assessing the complexity of a module based on a set of complexity measures is a somewhat similar problem to multi-criteria decision making. Although we have several driving factors to make a judgment, in the end, we want to get an overall value for complexity or a rank between the modules.

The core of the collected metrics set consists on textual complexity metrics [5] and structural complexity metrics [8]. Each metric measures an aspect of software complexity, but some of these aspects have a high correlation with others. The collected metrics can be combined using the Principal Components Analysis technique [7, 9], so that we can get a smaller number of metrics without a significant information loss. This method allows us to derive complexity metrics with a low correlation between them (factors).

Using the metrics collected with Logiscope [13], we got three factors condensing the information conveyed by several complexity metrics: F1 (program complexity, maximum nesting level, direct calls, exit nodes, entry nodes, maximum number of degrees, intelligent content, cyclomatic complexity, program level, size, maximum number of nodes, maximum number of statements, nodes, edges, operators and estimated number of errors), F2 (mental effort, unconditional jumps and essential complexity) and F3 (pending nodes).

Using the RAPE records on effective effort, we were able to compute a linear model with the effort as a dependent variable and the complexity factors as the estimators. It turned out that F3 had a neglectible effect on the effort required to build the modules($\beta_3 \approx 0$), so it was removed from our model.

$$Effort_i = \beta_0 + \beta_1 * F1_i + \beta_2 * F2_i + \epsilon_i \qquad (1)$$

The β_j coefficients were computed by the least square method and ϵ_i represents the residual error for each

case. This model may be instantiated with the parameters presented in Table 1.

$Coeff.$	$Est.$	$StdError$	$t(5\%)$	$Sign.$
β_0	6.682	0.470	14.221	0.000
β_1	3.104	1.060	2.928	0.006
β_2	8.522	0.587	14.508	0.000

Table 1. Effort model parameters.

The F statistic value (108.561) with a significance of 0.05 is higher than the critical value for a sample of this dimension (F(2, 38) = 3.23). This means we can reject the null hypothesis ($H_0 : \beta_1 = \beta_2 = 0$).

6.2. Model discussion

The positive coefficients show that an increase in any of the complexity factors results in the increase of the expected effort. The adjusted determination coefficient of the model (R^2_{adj}) is 84.3%. This means that only 15.7% of the effort variation is not explained by the model. For instance, the complexity of the database accesses is not accounted for. Therefore, if a change involves modifying an embedded SQL statement or change the definition of a table, the model will underestimate the necessary effort needed to fulfill that task.

Another limitation is due to the sources of data used by this approach. There are factors external to the software's intrinsic complexity that have an impact on the total effort required to perform a maintenance activity, such as the different programmer's productivity [1, 11]. A programmer with twice the productivity of another one is likely to need only half the effort to perform the same evolution action. There is not enough statistically relevant data to include this *human factor* on the model, for now, due to the relatively low total number of registered evolution actions (41).

These aspects help explaining the current level of accuracy of effort estimation provided by the model, which is expected to grow with the re-calibration that new evolution actions will enable and the inclusion of new factors to cover the earlier stated limitations.

However, the model gives a good framework for assessing a module's complexity. This information is quite useful as it allows the detection of the most fault prone modules and thus can help planning re-engineering activities [4].

7. Conclusions

SofTrack is a system that combines several different technologies, from the information retrieval techniques used for extracting data from the source code to the usage of web technology for making that information available to users in a distributed environment. It was designed to support a SPI initiative.

Its implementation has generated awareness in the organization to the need of producing software in a more controlled environment. It has led to a significant change in people's behaviors, in what concerns the making of evolution requests by the introduction of a formalism in that activity.

Instead of the traditional usage of human expertise by itself to make decisions, the decisions can now be taken with the help of updated documentation and software metrics. This makes software development a much more controllable and predictable activity.

SofTrack provides an important help both in the understanding of the systems (with the documentation generated by Samaritan) and the impact analysis of eventual evolution actions. The effort estimation model for evolution actions shows the expected dependency between the complexity of an evolution action and the required effort. The complexity assessment enables the detection of error prone modules.

References

[1] F. Brooks. The mythical man-month. *Addison-Wesley*, 1975.

[2] J. Callahan, R. Katsuriya, and R. Hefner. Real-time verification and validation for large-scale software projects via the web. *http://hopper.cs.wvu.edu/projects/IEEE.html*, 1998.

[3] ESW. Samaritan - manual tecnico. *Internal Report*, 1996.

[4] M. Goulao, A. Monteiro, F. Abreu, A. Almeida, and P. Sousa. A software evolution experiment. In *Proceedings of the ESCOM ENCRESS 98*, 1998.

[5] M. Halstead. Elements of software science. *Elsevier North-Holland, New York*, 1977.

[6] D. Houston and J. Keats. Cost of software quality: A means of promoting software process improvement. 1996.

[7] H. Kaiser. The varimax criterion for analytic rotation in factor analysis. *Psychometrica*, 1958.

[8] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4), 1976.

[9] E. Reis. Analise factorial das componentes principais: Um metodo de reduzir a complexidade sem perder informacao. *Giesta ISCTE*, 1993.

[10] M. Roberts. Experiences in analizing software inspection data. In *Proceedings of the Software Engineering Process Group Conference*, 1996.

[11] H. Sackman, J. Erikson, and E. Grant. Exploratory studies comparing online and offline programming performance. *CACM*, 11(1), 1968.

[12] SPSS. Spss user's guide. *SPSS user manual package*, 1997.

[13] Verilog. Logiscope viewer - basic concepts. *Logiscope user manual package*, 1993.