Empirical Software Engineering, 8, 59–81, 2003.

Designing and Conducting an Empirical Study on Test Management Automation

GRISELDA GIRAUDO griselda.giraudo@sodalia.it
Sodalia SpA, via V. Zambra, 1—38100 Trento, Italy

PAOLO TONELLA tonella@itc.it
ITC-irst, 38050 Povo, Trento, Italy

Editor: Victor Basili

Abstract. Test management aims at organizing, documenting and executing test cases, and at generating execution reports. The adoption of a support tool and of a standard process for such activities is expected to improve the current practice. ITALO is a European project devoted to the evaluation of the benefits coming from test management automation.

In this paper the experiences collected and the lessons learned during ITALO are summarized. A formal methodology was adopted for the selection of a support tool among those available from the market. A survey of the current practice in component testing was conducted to adapt the existing process model so as to obtain the greatest benefits from automation. An empirical study was then designed to measure the effects that are expected to be produced by the new test process complemented with the introduction of the support tool.

Three pilot projects were conducted to measure the benefits obtained from tool usage and process modification. Results are presented and discussed in this paper.

Keywords: Test management, process improvement, empirical study.

1. Introduction

The goal of ITALO (improvement of the testing activities for the development of object oriented software), the ESSI PIE (process improvement experiment) European project no. 27912, is to improve Sodalia[1] object oriented (OO) testing phases by adopting automatic tools to complement and help programmers during component testing activities. Components are the smallest units implementing and exporting a user recognizable functionality; component testing, with current practice, has no automatic support and requires almost the same effort as the coding phase.

The area of component test which was perceived in need of improvement and for which commercial tools are available is test management, which includes test case organization, execution and report generation. The improvement in the test case organization consists of a uniform hierarchical structuring of all test cases defined for a component. As regards test case execution, tools can support the automatic execution of a given sequence of test cases. Report generation should permit the automatic production of documents containing test execution results and statistics, giving a better control on the ongoing testing activities, and improving the availability and quality of the documentation.

The component test process model was revised so as to incorporate the automation available through tool adoption. The approach to the component testing process improvement involved the acquisition of an automatic tool, the identification of pilot projects and the set up of an empirical study. Three pilot projects were conducted to measure the effects of applying the new testing process and introducing the support tool. Qualitative and quantitative data were collected during the pilot projects, in accordance with the experimental design. This paper describes the activities performed within the main phases of ITALO, analyzes the results obtained by the three pilot projects, and reports some general lessons that could be learned from this experience. Preliminary results were presented in La Commare et al. (2000).

The paper is organized as follows: the experimental design developed to evaluate the effects of tool adoption on three pilot projects is detailed in Section 2, while the results of the study execution are given in Section 3. A discussion of such results can be found in Section 4, where a list of lessons learned is provided, along with several general considerations on the adoption of a software engineering support tool. Finally conclusions are drawn in Section 5. The following Appendixes deal with the tool selection procedure, the process improvement and the questionnaires used for the qualitative evaluations.

2. Experimental Design

The aim of the study is to determine if the adoption of tools that help programrs during component testing and the formalization of the component test process promote a higher software quality with decreasing or constant human resources. The selection procedure adopted for the identification of a support tool with the features needed for its successful introduction in Sodalia is presented in Appendix A. Appendix B describes how the component test process was revised and modified, so that an optimal use of the tool functions could be achieved.

Evaluation methods and experimental design issues related to conducting empirical studies in software engineering are discussed in Kitchenham (1996) and Pfleeger (1994, 1995). In Basili et al. (1999) the definition of a common framework for empirical studies in software engineering is proposed, with the purpose of building knowledge out of families of experiments based on shared principles and models. In the following, terms and concepts are borrowed from such works.

2.1. Experimental Hypothesis

Current component testing practice was investigated to determine the features that the support tool should provide and the new process to be adopted. A result of the survey conducted for such an investigation is a list of improvement areas in the component testing process. The main activities that are expected to receive benefits from tool adoption are:

- Test case organization and documentation.
- Test case execution.
- Report generation.

The improvement in the test case organization consists of a uniform hierarchical structuring of all test cases defined for a component. A common database is used to store all artifacts constituting and documenting the test cases, and a graphical user interface gives access to the test cases in a given project. In the current practice, the only opportunity for test case organization is the use of a proper structure of directories, while documentation is generally provided by means of comments in the test scripts and README files. No standard and uniform organization and documentation approach is followed by the developers. Thus the organization and documentation facilities offered by the adopted tool will ensure a uniform and standard test case management. As a consequence, the most beneficial effects are in the possibility to easily share information between different development and testing groups, and in the possibility to reuse available test cases, their organization and their documentation in the successive iterations[2] on the given component and across developers.

As regards test case execution, the selected tool supports the automatic execution of a given scenario, consisting of a sequence of test cases from a test suite. The scenario can be re-executed during regression check, and singular test cases can be selectively re-executed based on user defined criteria. Automation in the execution phase is achieved, according to the current practice, by defining ad hoc test scripts. While script definition is still required, with tool adoption it is possible to make them uniform across projects and developers, and thus reuse them. In addition, an automatic support is provided for their management and execution. During the successive iterations, the advantages are still more relevant, since regression check is automatically achieved and test cases and suites can be obtained from the previous iterations and reused even when the developers have changed. Finally, some of the test cases for component test can be passed to the next testing phases (integration and system test), thus avoiding the definition of redundant or repeated checks.

Report generation permits the automatic production of test execution log and summary, giving a better control on the ongoing testing activities. Information such as the test cases still in the FAIL state can be automatically retrieved and can drive the successive test steps. In addition, the final documentation on the component test activity is automatically filled in. According to the current practice the final report on the component test is manually written and detailed reports on the execution of each test case are not considered mandatory.

From the above considerations it is clear that some improvements associated with tool adoption might be anticipated. Standardization in the test case management is an example of such an improvement with respect to the current practice. However, the final objective of tool adoption is an improvement in the testing efficiency and effectiveness. Testing efficiency is the number of defects found per unit of effort, while testing effectiveness is the number of defects found per unit of size. They should increase thanks to the automatization of tedious management activities and of regression checks, so that a deeper test can be conducted. As a consequence, the successive test phases, namely integration and system test, are expected to find fewer defects that can be associated to a particular isolated component. Other direct benefits of tool adoption have been identified in the reuse of available test cases, of their organization and documentation, and in their re-execution during the successive development iterations on a given component.

The effects of tool introduction are of wide range, and it is difficult to have a single experimental hypothesis for them. Some benefits are expected on a time scale longer than this empirical study. The following hypothesis accounts for the quantifiable effects:

Component test support tool adoption and process formalization should result in a reduced testing effort. Moreover, defects detected during component testing are expected to increase, while those reaching integration and system test should decrease.

2.2. State Variables

The state variables are the factors that can influence the results of the study. The projects selected for the empirical study should exhibit a variety of characteristics typical for the organization, and possibly more than one value should be taken for each characteristic. It is important to determine the state variables during the experimental design, because they can affect the conclusion on the experimental hypothesis. In fact, if only boundary or non typical values occur in the pilot projects for the state variables, the results of the study are hardly extensible to other projects in the organization. The coverage of the most typical values for the state variables is a precondition to obtain general and valid results on the experimental hypothesis.

Based on our experience, we hypothesized that the most important state variables for the pilot projects are the following:

1. Programming and testing experience of the developers.

2. Application domain of the tested component.

3. Functional area of the classes involved in the tested component.

4. Familiarity of the developers with other tools.

5. Scale of the project.

6. Size of the project team.

7. Number of iterations previously completed.

The programming and testing experience may affect the learning curve associated to the adoption of the tool. Experienced developers may, for example, quickly build a mental model of the way things are organized in the tool.

The application domain may influence the degree to which the test cases can be automated. The tool can be better exploited when the test cases are executed unattended and the output of the execution is automatically checked for correctness.

The functional area of the classes composing the selected component may affect the kind of test to be executed. As a consequence, the tool support may vary and may have a percentage impact more or less relevant in the overall figure for the component test.

The familiarity of the developers with other software engineering support tools may reduce the psychological difficulty to accept the adoption of the new testing tool.

The scale of the selected projects can also influence the result of tool evaluation, since in small projects the burden associated to tool adoption may be dominant with respect to the benefits, while larger projects are likely to find high improvements in the management area.

The size of the project team has similar influences, since projects with many people involved have a remarkable need of ways to share information and adopt standards.

Finally, testing a component that was developed from scratch is different from testing a component coming from a previous development iteration and modified to augment its functionalities. In the first case all the effort is devoted to conceiving the test cases in the way that is supported by the tool, while in the second case a preliminary activity of migration may be required to import the existing test cases inside the tool database.

Other affecting factors, that are more difficult to check and control, are the stability and number of changes, and the defect impact and cost.

2.3. Pilot Project Selection Criteria

Internal validity (Basili et al., 1999) is a main concern of ITALO. Consequently, pilot projects had to be selected with features that are typical of Sodalia, so that the results can be applied to a wide range of similar projects. The state variables identified for this study were used to drive the definition of the criteria to be employed for pilot project selection. The pilot projects were required to cover the most typical values of the state variables:

1. The range from novice developers to senior programmers should be covered.

2. The most typical application domain is telecommunication services.

3. The tested classes should be distributed among graphical user interface, database access and network service use.

4. The typical spectrum of familiarity with other tools ranges from no familiarity to expert tool users.

5. Typically Sodalia projects are medium scale projects.

6. Typically a component development team consists of about four persons.

7. Both components developed from scratch and components coming from previous development iterations should be found in the pilot projects.

In order to check that all the specified criteria are met by the pilot projects selected for the empirical study, the subjects were asked to fill in a pre-questionnaire (provided in Appendix C). The criteria listed above are not particularly selective. However, the knowledge of the pre-questionnaire answers, which position each project with respect to such criteria, are useful when trying to explain and interpret the experimental results.

2.4. Experimental Measures

The response variables of this study are the component testing effort, the number of defects found during component testing, and the number of defects found during integration/system test, as stated in the experimental hypothesis. Tool adoption should result in a reduced testing effort and an increased defect removal (i.e., more defects found during component test and less defects during integration/system test). To evaluate the results of the study on the pilot projects, the following measures strictly related to the experimental hypothesis were collected:

1. Component construction effort.

2. Defects found in component test.

3. Defects found in integration/system test, that can be associated to an isolated component.

4. Component size in LOC (Lines of code).

The component construction effort includes the component test effort. Thus a reduction of this measure (per unit size) is expected from tool adoption. During integration/system test, defects are associated to different sources. The defects (per unit size) that can be associated to individual components are expected to decrease, if the component test is able to discover more defects.

Baseline comparison data were extracted from the historical data base from a previous iteration on the same component or from projects with similar characteristics as the pilot projects. In this way the comparison of collected and historical data is expected to be more meaningful. Some considerations have to be made about the statistical relevance of the collected data. The experimental data are a small number, and thus their position, with respect to the baseline, could be affected by many deviating factors. During the experimental design, such factors were analyzed, and selection criteria were defined to obtain pilot projects representing the most typical situations in Sodalia. Nevertheless, some further noise could have some impact on the collected data. The detailed analysis of the experimental results will also address this point. However, some corrective actions have been designed in advance. The idea is to complement the above objective measures with subjective evaluations on the usefulness of the test support tool, to be collected through a post-questionnaire (provided in Appendix C). The purpose of the questionnaire is also to evaluate the medium to long term effects of tool adoption, that cannot be otherwise captured given the short time scale of this study. By complementing objective measures and questionnaire outcomes, a complete picture of the effects of tool introduction can be formed.

Additional details on the experimental design are provided in Tonella et al. (1998).

3. Results

The first pilot project, REULIB (reuse user libraries) involved three project staff people. REULIB is a set of libraries of common use along Sodalia projects. Within the experimentation, a selected subset of components written in C++ and Java languages were tested under UNIX and Windows NT platforms. Test cases were built to exercise them in several different software configurations (e.g. different compilers, single and multi-thread versions, etc.).

The second pilot project was devoted to testing the NSI file interface, used to access network and service data stored in a file. The related component was implemented as a CORBA server, able to answer client queries on the collected data. It involved two people who developed this component from scratch. This pilot project differs from the other ones in that only the testing methodology defined by ITALO was adopted, while no automatic support was exploited. The need to limit the pilot project to the methodological prescriptions came from the development team, that could not afford the risk of both changing methodology and testing environment. On the other side, it was judged of interest for ITALO, since it allowed distinguishing the effects of process formalization from the effects of tool support. In fact, the other two pilot projects (differently from this one) were already characterized by a process formalization level similar to that defined by ITALO, and thus were mainly concerned with the effects of automation.

The main function of component WFM (workflow manager), tested within the third pilot project, is decoupling business process management from the application logics, so that processes can be explicitly handled as a whole or in their individual steps. WFM includes the enactment service, which handles every active instance of a process, and the workflow schema, which maintains the schemes defining the process instances. Six people participated in this pilot project. They exploited the support tool for test case organization, documentation and execution. Reports were also automatically generated by the support tool. This component was developed from scratch during the iteration under analysis. It includes the use of CORBA as middleware, and of the DBMS Oracle for data storage.

3.1. Pre-Questionnaire

The analysis of the pre-questionnaires for the pilot projects suggests that the collected data are meaningful and representative of Sodalia.

Subjects in the REULIB project team are spread between experts (1 person) and novices (2 persons), claiming some familiarity with other software engineering tools, and their number was judged typical. Application domain and functional area are horizontal with respect to Sodalia's core business. The application size (about 6 kLOC) is slightly below the average. All components come from previous development iterations.

The two people involved in the NSI project team are experts, and they were supported by a verification and validation (V&V) specialist for the methodological aspects. The application domain is typical of Sodalia's core business.

The WFM component has to be considered a whole subsystem, and due to its size the team was composed of six people. Among them, three are experts, while the other three have between 1 and 3 years of experience. The number of LOC considered here only accounts for the code not generated automatically, while the whole component size is about 20 kLOC. The application domain cannot be directly mapped onto telecommunications. Nevertheless, this component was developed using technologies that are common to the core business applications.

3.2. Quantitative Data

Since REULIB is an evolution of both ULIB and COMMONLIB, the latter were considered candidate baselines, against which response variables are evaluated. The considered iteration on ULIB was basically a maintenance intervention aimed at fixing some problems and porting the component. The development iteration on COMMONLIB had the purpose of adding eight new features, managing the date properly and allowing a better configurability. The iteration on REULIB was a deep restructuring aimed at handling nine different configurations, differing on platform, operating system, single vs. multi-thread and SC (standard component) vs. RW (rogue wave) commercial libraries. Both candidate baseline projects have features making them similar to REULIB, but the interventions performed for COMMONLIB make it a slightly better reference point, because they were mainly restructuring changes, as in REULIB, rather than corrective changes, as in ULIB. Consequently the version of COMMONLIB that has a higher similitude with the REULIB implementation was chosen as baseline.

Data were collected from *SAJ* (*Sodalia Activity Journal*) reports (effort) and from the defects database. The same procedures and tools were used for data collection, so as to obtain comparable data (effort is collected with a granularity of 1 hour; defects are collected using the TIR, test incident reports), while LOCs were re-calculated using a common tool. Then data were validated and normalized.

Table 1 gives the measures collected for REULIB and the reference values from the chosen baseline. The size of REULIB is greater than COMMONLIB's. Effort was measured in person-days (pd). Defects were counted both during component test (CT) and integration/system test (I/ST).

The collected measures were normalized with respect to the size of the component, so as to make them comparable. Since the proportion of changed LOC (lines of code) over the size is nearly the same in the two considered systems, normalizing on size instead of changed LOC is acceptable. Effort and defects per kLOC (thousands LOC) are provided at the bottom of Table 1. The last line of the table gives the number of user projects that included each component, and thus exercised it during their integration and system test. This datum is important to understand the extent to which each component was actually tested while used inside a real application.

The normalized effort is slightly decreased with respect to COMMONLIB. REULIB entered the integration and system test phase with a lower number of residual defects than COMMONLIB. Such improvement in the quality of this component may be due to a more effective component test phase. The number of user projects that exercised REULIB during integration and system tests is lower than COMMONLIB, which was tested within three different applications. Although the presence of 1 more test project for COMMONLIB may explain the higher number of defects detected (2.66 per kLOC), the difference between the two projects is remarkable (– 72.1%) and the additional test project seems to be only a partial explanation: a second reason for the defect decrease in integration/system test may be an increased quality of the component test.

Table 1. Measures collected for the REULIB component. COMMONLIB is used as baseline.

	COMMONLIB	REULIB	Δ
Size (LOC)	4504	6678	
Effort (pd)	79	109	
Defects (CT)	n/a	4	
Defects (I/ST)	12	5	
Effort per kLOC	17.5	16.3	– 6.8%
Defects (CT) per kLOC	n/a	0.59	n/a
Defects (I/ST) per kLOC	2.66	0.74	– 72.1%
User projects	3	2	

The NSI component was developed and tested following current practice, and after its delivery to the integration group, a new test phase was performed, this time following the ITALO methodology. The test management tool was used only in the design phase, in order to document test cases.

This experimentation allowed distinguishing the effects of process formalization from the effects of tool support. Main methodological modifications involved moving from an informal component test phase, conducted by each programmer according to personal practices, to a standard and uniform testing process, including strategy definition, test case design, test case implementation, environment setup and execution as main steps.

The historical baseline data consist of the effort and number of errors collected while performing development and testing as usual, with the old practice. Analysis of results is based on the comparison with the data collected while repeating the testing process following the new method proposed by ITALO.

The kLOCs and effort data were calculated and traced with the same tools and granularity as for the REULIB project.

The normalized effort is increased with respect to historical data. Such increase is due to the cost of changing the usual work organization. In this case there is no impact of the tool learning curve, but there is an impact of defining the strategy and the test design. The ITALO person supporting the project team needed some time to understand the component design and technology used, and the whole team had to work to set up the test environment and to create new test cases.

As shown in Table 2, the normalized defects found during component test are more than those detected with the current practice. This might be an indicator of a more effective defect removal ability of the component test phase, independently of the tool usage. Components delivered to the next development phase (integration and system test) are of higher quality and a lower number of residual defects is left to be revealed.

As regards the WFM pilot project, the data used as baseline were those collected for the bulk data transfer (BDT) component. This component was chosen because it was also developed from scratch, by almost the same group of people, and the methodology applied is comparable. The technology is slightly different, since CORBA was not used for BDT.

Table 2. Measures collected for the NSI component with the old informal practice, and applying the ITALO test method.

	Old practice	New test method	Δ
Size (LOC)	8000	8000	
Effort (pd)	16.25	25	
Defects (CT)	6	9	
Defects (I/ST)	10	8	
Effort per kLOC	2.03	3.12	+53.6%
Defects (CT) per kLOC	0.75	1.12	+49.3%
Defects (I/ST) per kLOC	1.25	1.00	− 20.0%

Table 3. Measures collected for the WFM component. BDT is used as baseline.

	BDT	WFM	Δ
Size (LOC)	7199	9100	
Effort (pd)	66	176	
Defects (CT)	4	10	
Defects (I/ST)	7	6	
Effort per kLOC	9.20	19.3	+109.7%
Defects (CT) per kLOC	0.56	1.09	+94.6%
Defects (I/ST) per kLOC	0.98	0.65	− 33.6%

Also in this case, the tool used to count the LOC is the same as for the previous projects, as well as the method and tool to collect the effort.

As shown in Table 3, the normalized effort is increased with respect to historical data. Such an increase is due to the high intrinsic complexity of the WFM component, and the relative high number of persons involved, and to the cost of learning the tool functions.

Also for the WFM, the normalized defects found during component test are more than those detected in the reference project. Defects per unit size detected during integration and system test are lower for WFM than for BDT (0.65 vs. 0.98). Both data (CT and I/ST) are in accordance with the experimental hypothesis.

3.3. Qualitative Data

Answers to the post-questionnaire contain subjective estimates of the tool usefulness. The scores were given by the REULIB and WFM teams, evaluating the support provided by the tool. The post-questionnaire was not filled by the NSI team, because the information requested was about tool adoption.

Tables 4 and 5 contain the subjective tool evaluations collected through the post-questionnaire and filled in by the REULIB and WFM project team. Since scores on the tool support were given on an ordinal scale, consisting of the five levels: not at

Table 4. Subjective scores given by the REULIB team. The adopted scale consists of five levels of support: not at all, little, somewhat, much, very much.

Tool support	Score (median)
Organization and documentation	somewhat—much
Execution	somewhat
Report generation	much
Effort reduction	little
Defect removal increase	little
Medium-long term documentation	much
Medium-long term regression check	much

Table 5. Subjective scores given by the WFM team. The adopted scale consists of five levels of support: not at all, little, somewhat, much, very much.

Tool support	Score (median)
Organization and documentation	much
Execution	somewhat
Report generation	much
Effort reduction	little-somewhat
Defect removal increase	not at all-little
Medium-long term documentation	somewhat
Medium-long term regression check	much

all, little, somewhat, much, very much, the median score is provided as a summary of the evaluations given by the different persons from the teams.

The support to effort reduction and defect removal increase was perceived to be little by the subjects involved in the study, but much support is expected in the medium to long term. The discrepancy on the short term effects with respect to the quantitative data may be due to the difficulties found when the tool was adopted for the first time. In fact, it required a substantial change in the organization of the testing activities and a lot of information had to be provided to the tool. Such activities were judged not much useful, having no direct and immediate impact on the discovered defects, and may have caused a sensation of waste of time. An initial resistance to tool adoption had to be overcome as well, which may also have contributed to the pessimistic evaluations on the short term effects.

The evaluations of the tool intrinsic features (support to test case organization and documentation, execution and to report generation) range from somewhat to much, thus showing that they are generally satisfactory, even if not particularly good. The lower score for the execution support is due to some peculiarities of both the REULIB and the WFM project. REULIB required the possibility of executing the test cases under nine different configurations, and this was not so easy to achieve with the facilities provided by the tool. Since WFM is based on CORBA, the execution of its test cases could be performed only after developing five proper driver modules, acting as CORBA clients. Such time-consuming activity is not supported at all by the selected tool, being focused on test management, but was perceived as an area where (partial) automation could help.

The medium to long term impact of tool adoption was judged positive, with regard to the possibility of obtaining good documentation and of effectively and efficiently performing regression check.

4. Discussion

The selected tool for test management automation was successfully introduced within three pilot projects. The experimental hypothesis was only partially confirmed by the empirical study, since an increased defect removal was obtained with an effort increase. According to the qualitative evaluations, some of the additional costs are expected to disappear in the next iterations of tool usage.

4.1. Short Term Effects

The normalized number of defects found during component test, when available, resulted increased, and the integration and system testing defect number was always lower than the historical data. The observed increase (per kLOC) was +49.3%, +94.6% for the second and third projects (the meaningful baseline datum for the first project is not available), while the decrease, for the three pilot projects, was respectively: −72.1%, −20.0%, −33.6%. This could be the result of an improved component test phase, achieved through the adoption of the test management support tool. On the contrary, the perceived short term effect on quality was obscured by the difficulties of tool and process adoption. The higher level of formalization and the different artifact organization produced an initial resistance. Consequently the estimated short term benefits were considered poor, while the medium to long term effects were judged more favorably.

The component testing effort measured during two out of three pilot projects resulted increased with respect to the historical data. Percentage effort variations for the three projects are respectively: −6.8%, +53.6%, +109.7%. Subjective estimates collected through the post-questionnaire confirmed such an outcome. The main reason for this result is that the effort spent during the iterations under analysis was devoted to importing existing test cases into the tool database, finding an appropriate organization that could match the characteristics of the component under test, customizing the tool to satisfy the specific needs of the project and solving problems in the tool functions. All such costs are expected to diminish in the next usages of the tool within the same project.

The existence of an initial cost related to the adoption of a software engineering support tool is a known and studied phenomenon (Kemerer, 1992; Kitchenham et al., 1995). It is in part explained by the need of performing start up activities, the cost of which is paid once for all, and of becoming familiar with the tool functions and organization. The latter is also known as the learning curve effect, and was quantitatively modeled in Wright (1936) for the first time.

Other ESSI PIE projects observed an initial cost of tool introduction, which apparently invalidates the experimental hypothesis and makes tool usage inefficient and ineffective. A deeper investigation suggests a longer term return on the investment and other less quantifiable benefits. The ESSI PIE project SIMTEST (Gaburro, 1996) worked on the test of man-machine interfaces, to be automated by adopting a support tool. They observed a 55% testing effort increase due to the following activities, required by the adopted tool: environment setting and test case production. However, benefits were obtained in regression checks, lower execution times, formalization of the test process and possibility to automatically generate test reports. The ESSI PIE project PHOENIX (Martin and Gallagher, 1996) objective was the re-engineering of a legacy system, to be achieved with the support of a CASE tool. The time necessary for an efficient and convenient tool usage was estimated about 3 to 6 months, because of the initial extra cost of tool introduction.

There are several positive effects of tool adoption, that are not apparent from the quantitative data.

The test case organization was judged very satisfactory. A new software engineer charged to test the component is expected to be able to understand it in a short time. On the contrary, in the previous setting such organization was implicit, and dependent on compilation flags and environment variables.

Test case documentation is another important achievement. Standard information can be retrieved for each test case, and the user is supported in its insertion and access. Future work will be devoted to some additional customization.

The component test process resulted improved in general, since test case management and execution is now a reproducible activity, independent of the personal choices of the programmers, even in presence of complex multi-version structures. Requirement coverage is explicitly supported by the tool, thus automating traceability and enforcing a deeper test of the different running conditions.

The tool was considered of interest also for the successive testing phases, namely integration and system test, thus extending the associated benefits to a wider portion of the software life cycle.

Finally, a result deriving from the training sessions on OO testing, from the tool adoption, and from the revision of the component test process was a general cultural improvement, that allowed to better exercise the component under test, with a wider coverage of the meaningful scenarios. In fact, component test, differently from integration and system test, is performed by the same engineers who developed the component, and who are more familiar with development techniques than test.

4.2. Long Term Effects

The answers to questions associated to medium and long term effects suggest that regression check, test case reuse, documentation and organization are expected to find substantial support from the tool, thus reducing the effort during the future testing iterations. The deeper regression check is also expected to increase the defect removal level.

Medium to long term usage of the tool is likely to become more effective and efficient. In fact the learning curve effect will be totally absorbed and the initial costs, to be paid the first time the tool is adopted by a project, will be absent.

4.3. Lessons Learned

Reduced effort and increased quality are goals that often drive the selection and adoption of support tools. Nevertheless such effects may be difficult to achieve, due to the unavoidable problems related to the culture of the people, the need to customise the tool, the learning phase, and the modification of existing practices. Therefore, managers and project leaders should be aware that even when the selected tool offers features extremely appealing for the automation of the process, an initial resistance has to be overcome. Cultural aspects should not be underestimated. The initial resistance to tool adoption cannot be addressed by simply showing the tool usefulness. Training sessions stressing the long term benefits, presenting the best practices and a high level view of the problem are crucial.

Ideally tool introduction and process formalisation should be conducted in parallel, so that the new activities can be directly coupled with the tool features. A common situation is that the current practice is already somewhat structured, and the migration requires the possibility to import old artifacts, together with a modification of the user's perspective on their organization and execution. For a successful tool introduction both activities need specific support.

A consequence of the above considerations is that the role of tool expert, possibly different from the tool administrator, is an essential one. A person has to be responsible for helping people to deal with practical problems, to make decisions on the overall organization of the artifacts and the overall approach to tool use, and to customise the tool according to specific needs. In this way the knowledge about tool and process can be spread company-wide and become part of the culture of each developer. The importance of facilitating the technology transfer process and of accumulating knowledge from successful experiences is a lesson that we share with several researchers in the software engineering community (Basili, 1993).

To import existing test cases into the tool database, a re-design activity may be necessary. In fact, the organization and operations provided by the tool could not always be directly mapped into the existing structure. The result in our experiment was a higher standardisation and quality of the inserted test cases, which justified the effort spent in such an activity.

5. Conclusion

An empirical study was conducted within the ITALO European project to measure the effects of test management automation. Three pilot projects were conducted in which a modified component test process was adopted and a support tool was introduced.

Results suggest that the short term effects are influenced by the initial cost due, e.g. to the need of properly reorganizing the test cases, and to the learning curve effect. The subjective estimates of the medium-long term effects allow predicting a progressive decay of such costs.

As regards the effects on the software quality, the collected data highlight an improvement with respect to historical data. Actually, the integration and system test teams which included the considered components detected a lower number of defects (per unit size). The experimental hypothesis of increased component test effectiveness cannot therefore be rejected and the qualitative data suggest that in the long term the positive effects should be amplified.

Moreover, other less quantifiable benefits are related to a formalisation and standardisation of the test cases hierarchical organization, documentation, execution procedure and to the possibility of automatically generating reports from the tool database.

This kind of study was an initial, necessary step in understanding and determining process improvement. It provided sufficient indications for the organization to continue adopting the tool and the new testing process. Further quantitative and qualitative studies will be conducted in the future to obtain data on a wider basis.

Appendix

A. Tool Selection

The DESMET method, described in Kitchenham (1996), was followed in the selection of the tool to be adopted within the ITALO project. The DESMET method is aimed at helping an evaluator in a particular organization in the design and execution of an evaluation exercise, in order to select the best tool in an unbiased and reliable way. Evaluations are context dependent, in that each specific tool is not expected to be the best in all circumstances.

A qualitative DESMET evaluation is also termed feature analysis and is based on identifying the requirements for a given set of activities and mapping them to features that the tool should support.

The feature analysis performed for the ITALO project was conducted in two sequential iterations. The first iteration was based on the screening of the tools through the examination of the documentation available. It involved 9 tools. The second iteration considered a subset of the initially selected tools and adopted an approach that is intermediate between the screening and the case study. A realistic example task was used to exercise an evaluation version of the tools and some of the future tool users were involved.

Scores obtained for the initial list of nine tools are not shown for space reasons. Two tools were selected after the first iteration of feature analysis. Table 6 gives the resulting aggregate scores as a percentage of the maximum achievable for each macro-feature, after the second iteration. TestExpert[3] is definitely better in its

Table 6. Aggregates scores for macro-features after the second iteration.

Feature	SMARTS (%)	TestExpert (%)
1. Test management and execution	38.2	71.7
2. Interoperability	18.0	81.0
3. Learnability	93.0	85.0
Overall	53.2	76.8

support to test management and execution (1) and to interoperability (2). It offers superior facilities for documenting and executing the test cases, and for their versioning. It records the information attached to test cases and to executions in a database which can be accessed by the user to produce customised reports. Explicit support to interoperability is also provided. On the contrary, SMARTS[4] has poor documentation functionalities. It does not use a database and it records only few fixed information, available through a rigid set of reports. Poor support to interoperability is given. SMARTS appears to be globally superior in learnability (3). In fact it is simpler to get started with it, but in part this is due to the lower number of functionalities and to their lower sophistication level.

In conclusion, the tool selected for the empirical study was TestExpert. The interested reader can find additional details about the considered features, the evaluation procedure and the partial scores in Tonella et al. (1998).

B. Process Improvement

The Sodalia V&V testing guidelines (Bolzan and Correrini, 1996), which are part of the SIMEP (Sodalia integrated management and engineering process) methodology, contain the description of the component test process, which was subject to an improvement within the ITALO project. The new process has incorporated the suggestions of the survey conducted to understand the process "as is" and its deviation from the former process definition, as well as the suggestions from experts, experienced people working in component construction.

The general approach to the component test process, described below, was enhanced so that it can be performed with the support of the selected tool. Main changes with respect to the previous guidelines are:

- More emphasis on object oriented methodology.
- Reorganization of the testing activities artifacts to allow their production by exploiting tool facilities (i.e., test cases documentation management, test cases execution reports).

Component test is performed on a software component, i.e., on a software artifact that coincides or is part of an architectural component or subsystem, which may be either an executable program or a library, implementing and exporting a user or system recognisable functionality. Component test outcomes have to be evaluated against the design specifications of the component.

Current component test process consists of the following activities:

1. Strategy definition
2. Design

3. Construction

4. Environment setup

5. Execution

The objective of the strategy definition activity is to provide directions for the successive testing steps. Entry and exit criteria must be clearly stated, and could be inherited from the project plan. The strategy must be approved by the V&V manager, who is responsible for the quality of the whole product, even though he/she is not directly involved in the component test execution, which is generally performed by the development team. The related sub-activities are: entry/exit criteria definition, feature identification, staff composition, test plan definition, test of reused modules, specific test techniques.

The objective of the second activity is to design the test suites and the test cases to be associated to the features (software characteristics specified or implied by the requirements documentation, e.g.: functionality, design constraints, attributes or quality requirements) to be tested. For each feature one or more test cases should set up a proper environment and exercise the specified functionalities. The related sub-activities are: feature traceability (requirement traceability), suite definition, test case design, scenarios definition (a scenario is a set of test cases to be executed sequentially).

In the construction phase, test cases are written and documented. Test suites are used to group them. Standard configuration management tools are used to handle successive versions. Test cases can be of two types: automatic and manual. Automatic test cases can be executed unattended; by collecting their output or their return code it is possible to assess the associated PASS or FAIL state.[5] Manual test cases are executed interactively and the user must perform a sequence of actions before being able to decide on the PASS or FAIL result. For both kinds of tests, instructions have to be defined in this phase. The associated sub-activities are: suite hierarchy construction, test case construction, documentation, manual instructions, baseline construction and comparison criteria definition.

The purpose of the environment setup activity is to configure the runtime environment for each test case, so that all the needed resources are available. Its only sub-activity is: code and script construction.

The test execution activity consists in the execution of a scenario or a test case. For each detected failure a component test incident report (CTIR) has to be filled in, so that appropriate actions be taken to solve the problem. Execution is also performed during regression check. Test execution involves the following sub-activities: interactive execution, unattended execution, CTIR compilation, regression check, execution report, summary report.

The use of TestExpert impacts both the way activities are performed and the way documentation is produced. The three central activities in the component test process can be grouped and can be performed by interactively accessing the tool facilities (see Figure 1). In fact, in the design phase the suites, the test cases and the

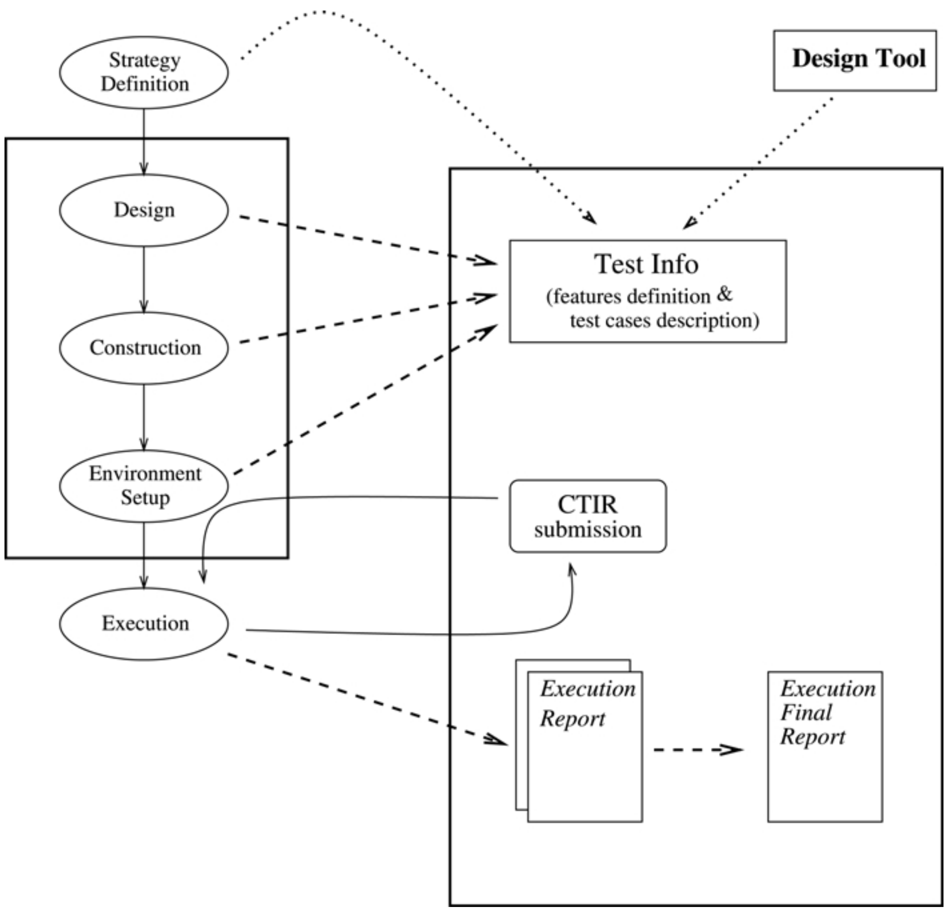

Figure 1. Component test process. Activities are shown on the left, while the associated documentation is depicted on the right.

scenarios that are defined can be inserted in the tool database. Then the next activities in the construction phase consist in the insertion of the instructions for each test case, including environment setup instructions. TestExpert supports such activities by automatically generating a prototype script that is customized by the programmer. Finally, the baseline can be constructed for the automatic test cases from TestExpert, and a pattern file with the comparison criteria can be defined, in the format required by the external file difference tool `exdiff`.[6] Thus the three central activities in the component test process are simplified and performed together with the support of TestExpert. The other two activities remain distinct. The strategy definition activity is not supported by TestExpert, but could receive its input from the design support tool. The execution activity is strongly supported by TestExpert, and most of its sub-activities can be performed automatically.

The documentation to be produced during component testing consists of two main documents, as shown in the right column of Figure 1. The first four activities produce as output a document with the features to be tested and with a description of the test cases. The execution activity produces as output a document with information on the results of the executions. The features to be tested, inserted in the first document, can be in part extracted from the design documentation. In fact, the requirements are features to be tested, together with the use cases and the other diagrams that correspond to the dynamic views. The information about the test cases, also in the first document, can be automatically extracted from the TestExpert database after completing the three central activities, which involve the insertion of such information. The second document for component test is produced after test execution, and contains an execution report for each test case and a summary report. It can be automatically obtained from TestExpert as a customized execution report.

Furthermore, as shown in Figure 1, it is possible to open a defect report with the aid of the tool. If an execution fails and a defect report (CTIR) is to be submitted, test data can be filled automatically into ClearDDTS[7] forms customized by Sodalia.

Additional details on the new component test process can be found in Giraudo et al. (1998).

C. Questionnaires

C.1. Pre-Questionnaire

1. What is your length of experience (years) in component development and testing?

2. What is the application domain of the components under test?

3. What is the functional area of the classes in the components under test (graphical user interface, database access, network services, etc.)?

4. Which software engineering support tools do you usually employ?

5. What is the size (function points or lines of code) of the components under test?

6. How many people are there in the project team?

7. How many components under test were developed from scratch and how many components come from previous development iterations?

C.2. Post-Questionnaire

Please answer the questions below. Use a score from 1 to 5 to indicate how useful/helpful the tool was (1 = "not at all", 2 = "little", 3 = "somewhat", 4 = "much", 5 = "very much").

1. How useful was the tool in supporting test case organization and documentation?

2. How useful was the tool in supporting test case execution?

3. How useful was the tool in supporting report generation?

4. How helpful was the tool in reducing the testing effort?

5. How helpful was the tool in increasing the defect removal rate?

6. In the medium to long term, how helpful is the tool going to be in providing useful documentation about the test cases and suites?

7. In the medium to long term, how helpful is the tool going to be in reusing existing test cases for regression check?

Notes

1. Sodalia is an Italian company, established in May 1992, which produces advanced software for the management of telecommunication services and networks.
2. The Sodalia development process is based on a spiral model with successive refinement iterations.
3. TestExpert is a test management tool by Silicon Valley Networks.
4. SMARTS is a test management tool commercialized by Software Research.
5. Additional states that can be associated with a test case include: ABORTED, UNKNOWN and UNTESTED.
6. Exdiff is a text, binary and bitmap file comparison tool commercialized by Software Research.
7. ClearDDTS is a work flow support tool comercialized by Rational Software Corporation.

References

Antoniol, G., La Commare, G., Giraudo, G., and Tonella, P. 1999. Effective feature analysis for tool selection. *Proceedings of Profes'99, International Conference on Product Focused Software Process Improvement*, Oulu, Finland, pp. 103–117.

Basili, V. R. 1993. The experience factory and its relationship to other improvement paradigms. *Lecture Notes in Computer Science* 717, Software Engineering ESEC'93, 4th European Software Engineering Conference, Garmish-Partenkirchen, Germany.

Basili, V., Shull, F., and Lanubile, F. 1999. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering* 25(4): 456–473.

Bolzan, E. and Correrini, F. 1996. *Verification and Validation Testing Activity Guidelines*. Sodalia Internal Report.

Gaburro, P. 1996. Automated testing for the man machine interface of a training simulator. *ESSI PIE 10824, SIMTEST, Final Report*.

Giraudo, G., Tonella, P., and La Commare, G. 1998. Component test guidelines. *ESSI PIE 27912, ITALO, Deliverable D2*.

Kemerer, C. F. 1992. How the learning curve affects CASE tool adoption. *IEEE Software* 23–28.

Kitchenham, B., Pickard, L., and Pfleeger, S. L., 1995. Case studies for method and tool evaluation. *IEEE Software* 52–62.

Kitchenham, B. 1996. A method for evaluating software engineering methods and tools. *Technical Report TR96-09*, DESMET project, UK DTI.

La Commare, G., Giraudo, G., and Tonella, P. 2000. Test management automation: lessons learned from a process improvement experiment. *Proceedings of EWSPT'2000, European Workshop on Software Process Technology*, Kaprun, Austria, pp. 156–160.

Martin, M., and Gallagher, J. 1996. Mature Software Application Adaptation Project. *ESSI PIE 10360, PHOENIX, Final Report*.

Pfleeger, S. L. *Experimental design and analysis in software engineering*. SIGSOFT Notes, Parts 1 to 5, 1994 and 1995.

Tonella, P., Giraudo, G., Antoniol, G., and Mambella, E. 1998. Sodalia. Tool selection feature analysis. *ESSI PIE 27912 – ITALO – Deliverable D1*.

Tonella, P., and Giraudo, G. 1998. Experimental design. *ESSI PIE 27912 – ITALO – Deliverable D3*.

Wright, T. P. 1936. Factors affecting the cost of airplanes. *Journal of Aeronautical Science* 3(2).

Griselda Giraudo has a degree in Mathematics. She has worked as software designer for 10 years in the avionics and factory automation fields. She joined Sodalia, a company that develops software for telecommunications, 7 years ago. In Sodalia, she first managed the testing teams for large software projects, then moved to the research and technology department responsible for verification and validation methodology and testing tools. She was the project manager for ITALO, an European Process Improvement Experiment where the present paper was developed. At present she performs analysis and architecture tasks in the service fulfillment line, that develops large software systems for telecommunications companies.

Paolo Tonella received his laurea degree cum laude in Electronic Engineering from the University of Padua, Italy, in 1992, and his PhD degree in Software Engineering from the same university, in 1999, with the thesis "Code Analysis in Support to Software Maintenance". Since 1994 he has been a full time researcher of the Software Engineering group at ITC-irst, Trento, Italy. He participated in several industrial and European Community projects on software analysis and testing. His current research interests include reverse engineering, object oriented programming, web applications and static code analysis.