Experimenting Traditional and Modern Reliability Models in a 3-Years European Software Project

Davide Tosi[1], Valentina Lenarduzzi[2], Sandro Morasca[1] and Davide Taibi[2]
[1]Dipartimento Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, Varese, Italy
[2]Faculty of Computer Science, Free University of Bozen, Bolzano, Italy
davide.tosi@uninsubria.it
valentina.lenarduzzi@unibz.it
sandro.morasca@uninsubria.it
davide.taibi@unibz.it

Abstract: Reliability is a very important non-functional aspect for software systems and artefacts. In literature, several definitions of software reliability exist and several methods and approaches exist to measure reliability of a software project. However, in the literature no works focus on the applicability of these methods in all the development phases of real software projects.

In this paper, we describe the methodology we adopted during the S-CASE FP7 European Project to predict reliability for both the S-CASE platform as well as for the software artefacts automatically generated by using the S-CASE platform. Two approaches have been adopted to compute reliability: the first one is the ROME Lab Model, a well adopted traditional approach in industry; the second one is an empirical approach defined by the authors in a previous work. An extensive dataset of results has been collected during all the phases of the project.

The two approaches can complement each other, to support to prediction of reliability during all the development phases of a software system in order to facilitate the project management from a non-functional point-of-view.

Keywords: Early Reliability, Reliability by design, Perceived Reliability, ROME Lab Model, Static Analysis, Predictive Models

1. Introduction

Reliability is considered by the community as one of the most important qualitative aspect for software systems and artefacts. In a survey conducted by the authors (Lavazza et al. 2011), 151 open source software (OSS) stakeholders were asked to rank 37 functional and qualitative factors for their importance during the selection process of OSS solutions. Reliability was indicated as the most important factor to take into consideration when selecting and adopting OSS.

Several definition of software reliability exist: software reliability can be seen as the "*probability that a system, product or component performs specified functions under specified conditions for a specified period of time*" (Musa et al, 1987). The OASIS standard "*Web Services Quality Factors*" defines the Reliability wrt input/output messaging only (OASIS, 2010). ISO/IEC25010 defines Reliability as the composition of several sub-characteristics: Maturity, Availability, Fault Tolerance, Recoverability (ISO/IEC, 2011), where:

- Maturity: degree to which a system meets needs for reliability under normal operation;
- Availability: degree to which a system, product or component is operational and accessible when required for use;
- Fault Tolerance: degree to which a system, product or component operates as intended despite the presence of hardware or software faults;
- Recoverability: degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.

In this paper and in the S-CASE project, we adopted the definition of software reliability as the probability of the S-CASE platform and tools to succeed in performing its functionalities and facilities.

In literature, several methodologies and approaches exist to measure reliability of a software project. Traditional approaches take into account the assessment of some specific factors to early predict the fault density of the software application and, as a consequence, its reliability (Software Reliability, 1992). More modern approaches try to observe from a statistical point-of-view the temporal distribution of failures to compute reliability, such as the Duane Model (Duane, 1964). For instance, in the Duane model and in its following extensions – such as in (Cheung, 1980) – mathematical and statistical models exist to compute the total number of cumulative test and development hours that are required to meet a target cumulative or instantaneous mean time between failures (MTBF) goal for a software artefact. Finally, other approaches try to correlate static and dynamic internal measures (Lavazza et al, 2012) of the software artefact to predict reliability (Lavazza et al, 2012). Moreover, several tools support the process of computing reliability indexes, such as the Reliability extension for R (R Rel Package, 2015) or the Frestimate platform (SoftRel, 2012).

However, it is quite hard to find in the literature papers that describe the process of applying these approaches to real software projects. Hence, it is important to report experiences that describe: (1) how the software system is analysed to collect reliability data; (2) how reliability indicators are computed during all the phases of software development to provide early indications on the software quality; (3) how to act to the software system to improve reliability overtime. To this end, in this paper, we describe step-by-step the methodology we adopted during the S-CASE FP7 European Project to predict reliability for both the S-CASE platform as well as for the software artefacts automatically generated by using the S-CASE platform. Two approaches have been adopted to compute reliability: the first one is the ROME Lab Model (Software Reliability, 1992), a well adopted traditional approach in industry; the second one is an empirical approach defined by the authors in a previous work (Lavazza et al, 2012). An extensive dataset of results has been collected during all the phases of the project.

The paper is structured as follows: Section 2 describes the methodology adopted during the S-CASE project to evaluate software reliability; Section 3 summarizes the reliability results collected during the project; We conclude in Section 4.

2. The S-CASE Reliability Methodology

The S-CASE European project [http://www.scasefp7.eu] is about semi-automatically creating RESTful Web Services through multi-modal requirements using a Model Driven Engineering methodology. The world of web services is moving towards REST, and S-CASE aims at facilitating developers implement such web services by focusing mainly on requirements engineering.

With S-CASE you have at your fingertips everything you need to build RESTful web services using a model-driven approach, including built-in artefact mapping, that minimises cost and saves you time.

In S-CASE, the quality of the automatically generated code through S-CASE tools is one of the most important aspect we tackled during the project. To this end, we developed a two-fold methodology to constantly monitor the reliability level of each tool developed in the S-CASE project, during the whole life-cycle of the project. The first approach exploits the Rome Lab Model to predict the early reliability (Software Reliability, 1992), while the second approach uses logistic regression models to predict the perceived reliability that end-users may have when adopting S-CASE tools (Software Reliability, 1992). The Rome Lab Model computes the early reliability of each S-CASE tool and pilot, starting from the observation of design and development criteria followed during the engineering process of S-CASE. The Rome Lab Model consists of a set of factors that are used to early predict the fault density of the software application. These factors are organized in macro sectors and worksheets that developers of a software component should use as checklists to collect data and parameters of their software under development. The output of the Rome Lab model is a fault density estimation in terms of faults per KSLOC (thousands of Source Lines Of Code). This output can be used to compute the total estimated number of inherent defects by simply multiplying the Rome output by the total number of KSLOC of the target project.

Predictive models aim at detecting correlations between reliability and observable qualities of software source code. These models aim at predicting the level of reliability perception that final users may have when adopting a software product. These models can be used by 1) end-users and developers that would like to reuse existing software products and components, to evaluate the perceived level of reliability of these software products that can be expected based on the characteristics of code; 2) the developers of software products, who can set code quality targets based on the level of perceived reliability they want to achieve.

We applied this two-fold methodology to compute the reliability indexes both for the tools developed in the context of the S-CASE project and also for the code developed by using the S-CASE tools and facilities. In total, we analysed the reliability of ten S-CASE tools and of three industrial projects developed by using S-CASE.

In this paper, we report the results of the reliability approach we followed in S-CASE, and we compare the results coming from traditional reliability models (such as the Rome Lab Model (Software Reliability, 1992)) with more recent reliability approaches (such as the regression models defined in Lavazza et al, 2012).

2.1 Adoption of the Rome Lab Model

In S-CASE, we applied the Rome Lab Model to compute the early reliability of each S-CASE tool, starting from the observation of design and development criteria followed during the engineering process of S-CASE.

The document *Software Reliability Measurement and Test Integration Techniques Method RL-TR-92-52* (Software Reliability, 1992) contains empirical data that were collected from a variety of sources, including the Software Engineering Laboratory and the Rome Laboratory (Lakey et al, 1997). The Rome Lab Model consists of a set of factors that are used to early predict the fault density of the software application. These factors are organized in macro sectors and worksheets (as summarized in Table 1) that developers of a software component should use as checklists to collect data and parameters of their software under development. Some parameters in this reliability model have trade-off capabilities (i.e., there is a large difference between the maximum and minimum predicted values for that particular factor.) Performing a trade-off means that the analyst determines where some changes can be made in the software engineering process or product to experience an improved fault density prediction.

The output of the Rome Lab model is a fault density estimation in terms of faults per KSLOC (thousands of Source Lines Of Code). This output can be used to compute the total estimated number of inherent defects by simply multiplying the Rome output by the total number of KSLOC of the target project.
Specifically, for each software component of the S-CASE platform, the Reliability Prediction Figure of Merit (RPFOM) is calculated according to the formula:

$$RPFOM = A * D * S \qquad \text{(eq.1)}$$

where *RPFOM* is the predicted fault density, *A* the application type metric, *D* the software development metric, and *S* the software characteristic metric. *A* is expressed in (fractional) faults per line of code determined prior to initiation of software development and encompasses the average fault density, which is used in the Rome Lab Model Worksheet 0. *D* encompasses information that should be available during the software pre-development phase of the life cycle. There are three development environment categories: embedded (i.e., the software team is unfamiliar with the application served by the program), semi-detached (i.e., the software team is experienced in the application but not affiliated with the user), and organic (i.e., the software team is part of the organization that is served by the program). This information is collected in the Rome Lab Model Worksheet 1 or 1A. *S* metrics are split into *S1* and *S2* metrics and can be derived by Rome Lab Model Worksheets 2 to 11 and are measures of the development processes and practices. *S1* is a metric of software characteristics during software Requirements and Design Specification; *S2* is a metric of Software Characteristics during software Implementation. The *S1* metric is derived from SA (Anomaly Management), ST (Traceability), and SQ (Quality Review Results) metrics. The *S2* Metric is derived from SL (Language Type), SM (Modularity), SX (Complexity), and SR (Standards Review) metrics. Hence:

$$S = S1 * S2 = SA * ST * SQ * SL * SM * SX * SR \qquad \text{(eq.2)}$$

Summarizing, the methodology adopted in S-CASE to early prediction of reliability can be schematized as shown in Table 1.

We created a set of spreadsheets to simplify both the collection process and also the automatic computation of RPMOF for each S-CASE component. We iterated the whole collection and elaboration process in two rounds: The first one on March 2016, while the second on October 2016. The design and development phases undertook from March to October were conducted with the Rome Lab Model indications in mind in order to improve the reliability index.

Section III details the results obtained during the two iterations.

2.2 Adoption of the Perceived Reliability Models

As described in (Software Reliability, 1992), statistically significant quantitative models exist to describe the correlation between Reliability and observable qualities of Java code. These models aim at predicting the level of reliability perception final users may have when adopting the Open Source Software (OSS) product. These models can be used by 1) end-users and developers that would like to reuse existing OSS products and components, to evaluate the perceived level of reliability of these OSS products that can be expected based on the characteristics of code; 2) the developers of OSS products, who can set code quality targets based on the level of perceived reliability they want to achieve. Hence, we applied these models to each S-CASE tool – since we have the availability of the source code of each S-CASE component – to estimate the reliability perception that end-users of the S-CASE platform may have when interacting with the S-CASE tools.

Table 1: Factors composing the Rome Lab Model

Task Id	Data	Input	Data Worksheet	Procedure Id
101	Application (A)		0	0
102	Development Env. (D)		1	1
103	System Level Char. (S)	S1		
	Requirements and Design Representation Metric (S1)	SA * ST * SQ		
	Anomaly Management (SA)			
	Traceability (ST)	All system documentation and source code	2	2
		Requirements and design documents with a cross-reference matrix	3	3
	Quality Review Results (SQ)	Requirements document; preliminary design specification; detailed design specification	4	4
104	Software Implementation Metric (S2)	SL * SM * SX * SR		
	Language Type (SL) (Lavazza et al, 2016)	Requirements specification	8	6 and 8
	Modularity (SM)	Module size estimates and source code	9D	9
	Complexity (SX)	Source code	10D	10
	Standard Review (SR)	Source code	11D	11

Two predictive models (logistic regression models) have been adopted to compute the Perceived Reliability of each S-CASE tool (for more details on the adopted models, the reader can refer to Lavazza et al, 2012). The first model correlates the Perceived Reliability with the two metrics: average eLOC (effective Lines Of Code) per Class and the average McCabe index (eq.3). The second model correlates the Perceived Reliability with the metric: average number of methods class (eq.4). Respectively:

$$\mathrm{REL}(\%) = \frac{1}{1+e^{-(0.36-0.0126\,\mathrm{eloc\_per\_class\_avg}+0.533\mathrm{McCabe\_index\_avg})}} \qquad \text{(eq.3)}$$

$$\mathrm{REL}(\%) = \frac{1}{1+e^{-(1.67-0.01114\,\mathrm{num\_methods\_per\_class\_avg})}} \qquad \text{(eq.4)}$$

We asked each S-CASE team leader to collect these internal software metrics (by means of the Metrics2 Eclipse Plugin [http://metrics2.sourceforge.net]) and we executed the two models against the collected data.

Also in this case, we iterated the whole collection and elaboration process in two rounds: The first one on March 2016, while the second on October 2016.
Section III details the results obtained during the two iterations.

3. Results

The first data collection process and the subsequent computation of the Rome indexes were done during March 2016. We completed the collection of all data for each S-CASE tool at the beginning of March and we computed, for each S-CASE tool the Predicted Fault Density RPMOF by following the metrics, values, and formulas suggested the ROME Lab Model.

At the end of the data collection phase, which involved all the S-CASE partners with development responsibilities, we collected 10 S-CASE plugin evaluations, for a total of 90 filled-in worksheets and 90 computed metrics. In Rome Lab Model (2016) the reader can find all the collected data for the worksheets of each S-CASE plugin.

At the end of the data elaboration phase, we computed in an automatic way, the 10 reliability indicators (RPMOF), one for each S-CASE plugin evaluated by means of the Rome Lab Model. As for this first iteration of data collection, the number of expected faults per KSLOC spans from a maximum of 14,3 (S-CASE Web Service Composition Tool) to a minimum of 5,3 (S-CASE NLP Server), with an average value equals to 8,7 faults per KSLOC. This suggests that the expected quality of the first release of the S-CASE tools is quite high. It is important to observe that the Rome Lab Model considers intrinsically a minimum number of faults per KSLOC of 4 (e.g., this is due to the average fault density introduced intrinsically by the application type and also to the average fault density introduced intrinsically by the development environment.)

We conducted a second iteration of data collection with reference to the most up to date release of each S-CASE tool in order to understand whether the guidelines, which we derived during the first iteration, actually improved the development process of each S-CASE tool and, as a consequence, the level of reliability. Specifically, S-CASE partners and developers were asked to:

- Increase the number of small software modules / packages (improve modularity)
- Reduce the number of software modules / packages with McCabe Complexity > 7
- Minimize the number of lines of code in a non-high order language.

At the end of the data collection phase, which involved all the S-CASE partners with development responsibilities, we collected 13 S-CASE plugin evaluations and also 3 S-CASE applications developed in three real pilots, for a total of 117 filled-in worksheets and 117 computed metrics. The three pilots were implemented by three partners of the project (Delphis for the Wapo.IO application, Ericcson Nikola Tesla for the GiftCase application, and Engineering spa for the ETICS BTE framework).

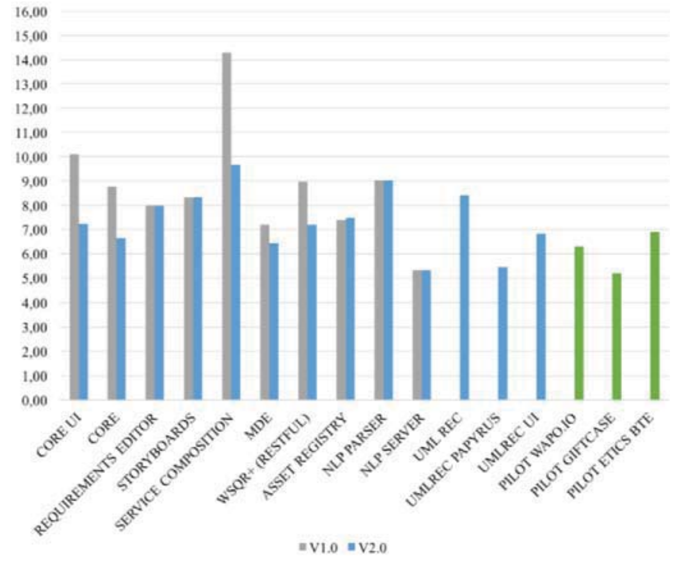

Figure 1: Graphical representation of estimated faults per kloc by means of the Rome Lab Model

for S-CASE Tools, Plugins, and Pilots V1.0 vs. V2.0

At the end of the data elaboration phase, we computed in an automatic way, the 16 reliability indicators (RPMOF), one for each S-CASE plugin and one for each pilot, evaluated by means of the Rome Lab Model. As for this second iteration of data collection, the number of expected faults per KSLOC spans from a maximum of 9,7 (S-CASE Web Service Composition Tool) to a minimum of 5,2 (GIFTCASE Pilot), with an average value equals to 7,1 faults per KSLOC. This suggests that the expected quality of this second release of the S-CASE tools is quite high, with a considerable improvement, if compared to the first iteration. Table 2 and Figure 1 summarize the results (i.e., Predicted Fault Density RPMOF) by applying the Rome Lab Model to each S-CASE tool under development and also to each pilot, and it compares RPMOF computed for the two rounds of data collection (histogram bars in grey are for release v1.0, in blue for release v2.0, and in green for pilots).

For all the tools we had an improvement on the ROME reliability index, except for the Assets registry plugin (where we had a small negative impact for the new release) The reason behind this small reduction in predicted reliability is the increase in number of modules with LOC > 500 (from 3 to 4) and this is not considered as a good practice for modularity. As for NLP, we do not have new indexes, since the two plugins (NLP parser and NLP server) did not change in the last six months. As for UML tools, we only have data concerning the 2nd round, since the UML tools were in a very early development phase, in march 2016.
We also collected the number of failures detected by the S-CASE partners in charge to develop the three Pilots when generating code automatically by using S-CASE tools. For the three pilots we also computed the ROME REL indexes.

One failure has been detected by the three pilots for the auto-generated code through S-CASE tools, thus demonstrating the good quality of the auto-generated code. Table 2 summarizes the results collected in this phase. Specifically, the table reports on:

1. the ROME RPMOF: all the three pilots have very low fault density estimations;
2. the lines of code generated automatically by using the S-CASE tools during the development of each pilot;
3. the lines of code generated manually by the developers of each pilot;
4. the number of real failures experienced when testing the pilots and referring to the code auto-generated by the S-CASE tools;
5. the number of real failures experienced when testing the pilots and referring to the code manually generated by pilots' developers

It is interesting to observe that we detected, in total for the three pilots, only 1 failure for the auto-generated code (on a total of 25.385 LOCs) against 15 failures for the manually generated code (on a total of 5.470 LOCs), thus confirming the reliability of the code generated by using the S-CASE tools and plugins.

Table 2(a): Predicted fault density for S-CASE toolset comparison (round 1 & round 2)

Tool/plugin	**RPMOF/KLOC round 1 (2/3/2016)**	**RPMOF/KLOC round 2 (17/10/2016)**	**Diff (%)**
Web service composition	14,30	9,66	+48,0
Core UI	10,09	7,23	+39,6
NLP parser	9,02	na	-
WSQR	8,97	7,20	+24,6
Core	8,75	6,65	+31,6
Storyboard creator	8,32	8,32	+0,0
Requirements Editor	7,98	7,98	+0,0
Assets registry	7,40	7,49	**-1,2**
MDE	7,20	6,43	+12,0
NLP server	5,33	na	-
UMLREC	na	8,41	-
UMLREC PAPYRUS	na	5,44	-
UMLREC UI	na	6,83	-

Table 2(b): Predicted fault density for the pilot prototypes (16/9/2016)

	DELPHIS PILOT WAPO.IO	ERICSSON PILOT GIFTCASE	ENGINEERING PILOT ETICS BTE
ROME RPMOF/KLOC	6,3	5,2	6,9
LOC auto-generated via S-CASE tools	12.791	4.021	8.573
LOC manually generated by pilots	1.777	3.140	553
Failures for auto-generated code	0	0	1
Failures for manually generated code	6	7	2

As for the perceived reliability, the predictive model 1 (explained in eq.3) leads to the reliability indexes of Figure 2 for the S-CASE tools, plugins, and pilots. Both release v1.0 and also v2.0 are reported in the histogram to show how perceived reliability changed during the two major releases.

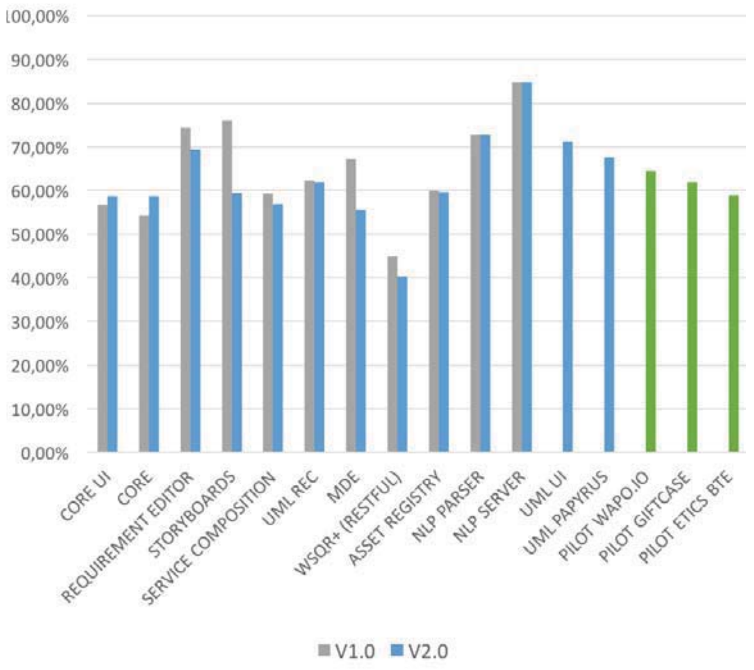

Figure 2: Graphical representation of perceived reliability computed by means of predictive model 1 (eq.3) for S-CASE Tools, Plugins, and Pilots V1.0 vs. V2.0

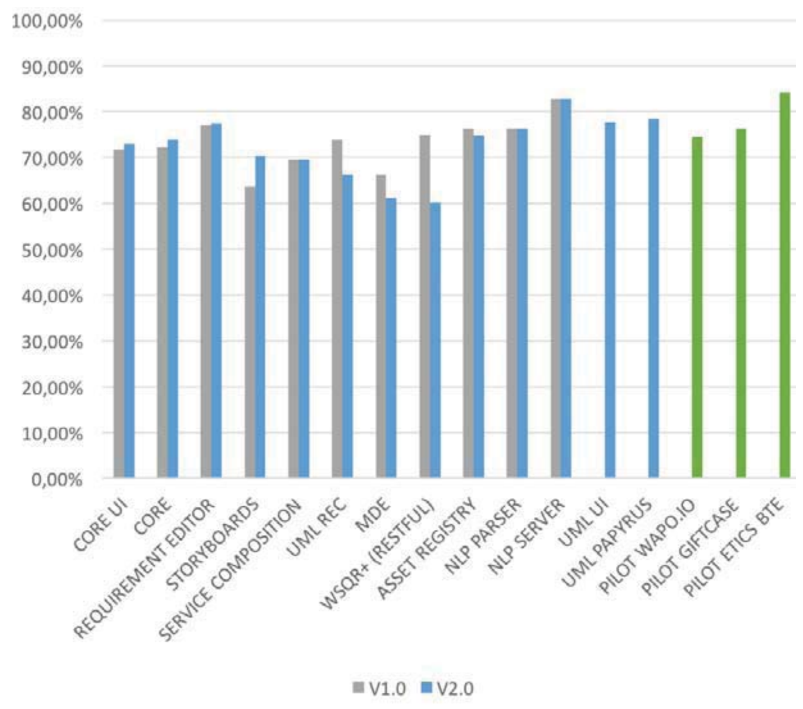

Figure 3: Graphical representation of perceived reliability computed by means of predictive model 2 (eq.4) for S-CASE Tools, Plugins, and Pilots V1.0 vs. V2.0

The perceived reliability for release v1.0 spans from a minimum value of 45% for the WSQR tool (Tosi, 2015) (Thair et al, 2013) (Lavazza et al, 2015) (i.e., 45% of the end-users of the S-CASE WSQR tool will be very satisfied by the reliability of the tool) to a maximum of 85% for the NLP SERVER tool (i.e., 85% of the end-users of the S-CASE NLP SERVER tool will be very satisfied by the reliability of the tool). The perceived reliability for release v2.0 spans from a minimum value of 40% for the WSQR tool to a maximum of 85% for the NLP SERVER tool. It is important to highlight that, for the NLP PARSER and SERVER tools, the releases did not change. As for WSQR, it is important to note that the tool drastically changed from v1.0 to v2.0 moving from a desktop-based tool to a RESTful implementation of the WSQR service. All the three pilots have a quite high perceived reliability index (in average 62%).

The predictive model 2 explained in eq.4 leads to the reliability indexes of Figure 3 for the S-CASE tools, plugins, and pilots. Also in that case, both release v1.0 and also v2.0 are reported in the histogram to show how perceived reliability changed during the two major releases.

The perceived reliability for release v1.0 spans from a minimum value of 64% for the STORY BOARD tool to a maximum of 83% for the NLP SERVER tool. The perceived reliability for release v2.0 spans from a minimum value of 60% for the WSQR tool to a maximum of 83% for the NLP SERVER tool. All the three pilots have a high perceived reliability index (in average 78%).

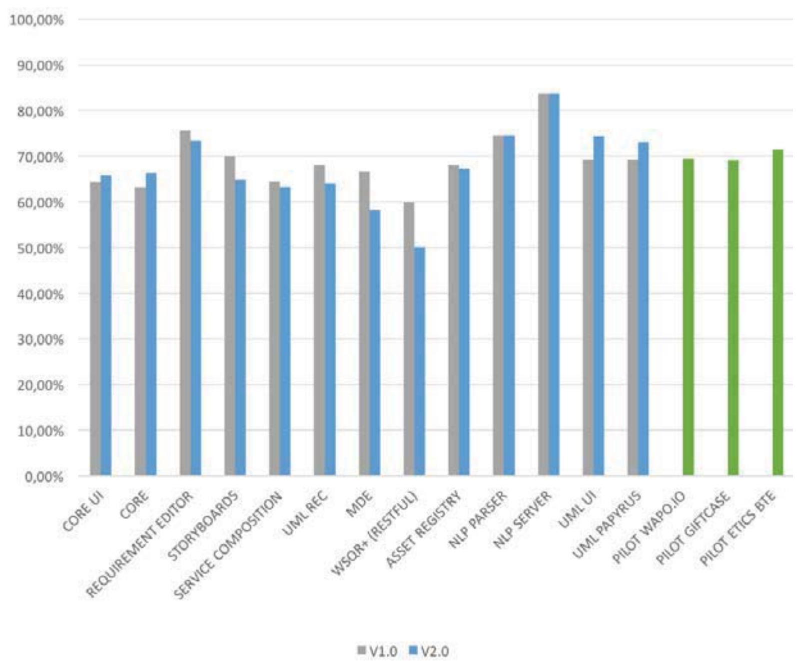

Figure 4: Graphical representation of the average perceived reliability computed by means of predictive models 1, 2 (eq.3, eq.4) for S-CASE Tools, Plugins, and Pilots V1.0 vs. V2.0

The average results for the two predictive models (eq.3 and eq.4) are reported in Figure 4. The perceived reliability of each S-CASE tool is very high and it spans from a minimum of 60% for the WSQR tool to a maximum of 84% for the S-CASE NLP SERVER tool. It is also important to highlight that also the perceived reliability of the three pilots, which has been developed with the support of S-CASE tools, is very high (in average 70%).

We also applied the two models to a set of (five) well-known and adopted OSS tools (Lavazza et al, 2012) to compare the reliability level of the S-CASE tools and pilots with "standard" tools. Specifically, we computed the Perceived Reliability for: Jackarta Commons IO, HTTPUnit, PMD, Log4J and JUnit. They have a Perceived Reliability that spans from a minimum of 64% to a maximum of 80%, perfectly in line with the S-CASE values.
If we compare Figure 1 and Figure 3, we can observe that the results obtained by using the two approaches (i.e., the Early Reliability ROME model and the Perceived Reliability) are comparable. For example, the S-CASE NLP SERVER is the most reliable service for both the two approaches. Figure 5 compares the ROME indexes (inverted values in the graph – blue line) and the average of the perceived reliability indexes (orange line) for S-CASE Tools, Plugins, and Pilots release V2.0.

Summarizing, both the two approaches to compute reliability demonstrate the high reliability level reached by each of the S-CASE component (see Figure 5).

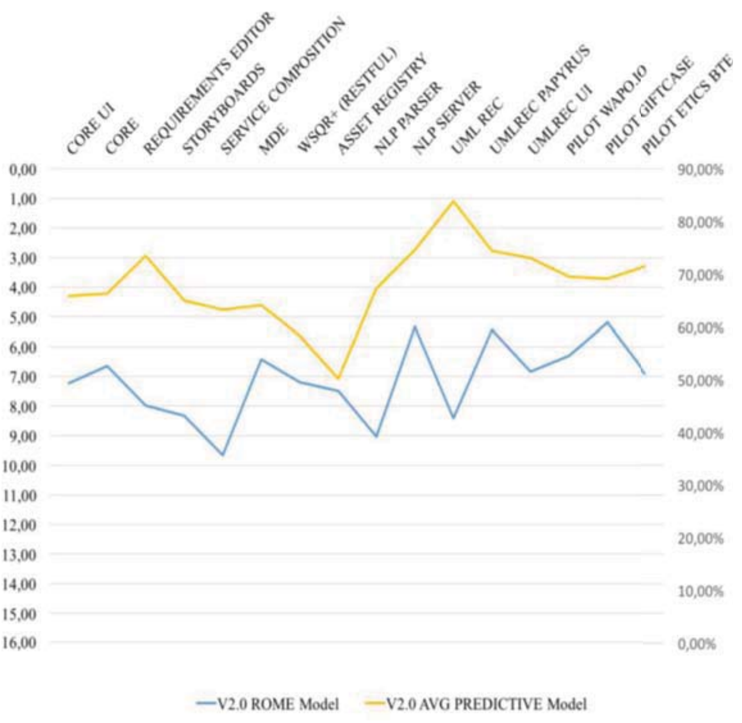

Figure 5: Graphical comparison between the ROME indexes (inverted values) and the average of the perceived reliability indexes for S-CASE Tools, Plugins, and Pilots V2.0

4. Conclusion

In this paper, we described the methodology we adopted during the S-CASE FP7 European Project to predict both early reliability by design as well as perceived reliability for the S-CASE platform and for all the software artefacts automatically generated by using the S-CASE platform. Two different approaches have been integrated in the methodology and an extensive dataset of results has been collected during all the phases of the project. In general, the two approaches had the same behaviour and they provided complementary indicators for the S-CASE reliability. With the methodology, all the partners of the S-CASE project had the chance to monitor the level of reliability continuously, and then act promptly to solve bugs and design issues that can compromise the reliability level of the overall platform. At the end of the project, a very high level of reliability has been reached, demonstrating that the designed methodology was able to support partners in all the development phases.

Acknowledgments

This work has been partially supported by (blind for review), funded by the European Commission, and by the project (blind for review) funded by (blind for review).

References

Cheung, R.C., (1980) "A User-Oriented Software Reliability Model," Software Engineering, IEEE Transactions on, vol.SE-6, no.2, pp.118,125, March 1980

Duane, J.T., (1964) "Learning Curve Approach To Reliability Monitoring," IEEE Transactions on Aerospace, Vol. 2, pp. 563-566, 1964.

ISO/IEC 25010 (2011). “Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models”. 2011

Lakey, P. and Neufelder, A. (1997) “System and Software Reliability Assurance Notebook”, Rome Laboratory

Software Reliability (1992) “Measurement, and Testing Guidebook for Software Reliability Measurement and Testing”. RL-RT-92-52 Vol.II. April 1992 http://www.dtic.mil/dtic/tr/fulltext/u2/a256164.pdf

Lavazza, L. Morasca, S. Taibi, D. and Tosi, D. (2012) “An empirical investigation of perceived reliability of Open Source Java Programs”, In Proceedings of the 27th ACM Symposium on Applied Computing (SAC), Riva del Garda, Italy. March 2012

Lavazza, L. Morasca, S. and Tosi, D. (2016) “An Empirical Study on the Effect of Programming Languages on Productivity”. In Proceedings of the 31st ACM Symposium on Applied Computing (SAC), Pisa, Italy. April 2016.

Lavazza, L. Morasca, S. and Tosi, D. (2015) “Enriching Specifications to Represent Quality in Web Services in a Comprehensive Way”. In Proceeding of the 9th IEEE International Symposium on Service-Oriented System Engineering. Redwood City, CA, USA.

Lavazza, L. Morasca, S. Taibi, D. and Tosi, D. (2012) “On the Definition of Dynamic Software Measures”, In Proceedings of the 6th International ACM Symposium on Empirical Software Engineering and Measurement (ESEM), Sweden. Sept. 2012.

Musa, D., Iannino, A. and Okumoto, K. (1987) “Software Reliability: Measurement, Prediction, Application”. McGraw-Hill. 1987.

OASIS (2010) “Web Services Quality Factors Version 1.0”. 2010

Putnam, L. and Myers, W. (1992) “Measures for Excellence: Quantitative Software Management”, Yourdon Press, Englewood Cliffs, NJ.

R Package for Reliability (2015). Web: http://cran.r-project.org/web/packages/Reliability/Reliability.pdf. Accessed: Feb. 2017

Rome Lab Model Collected Data set (2016): http://s-case.github.io/reliability/S-CASE_ROME_Lab_Model_Results_2.zip

SoftRel Frestimate (2012). Web: http://www.softrel.com/1About_Frestimate.html. Accessed Feb. 2017

Tahir, A. Tosi, D. and Morasca, S. (2013) “A Systematic Review on the Functional Testing of Semantic Web Services”. Elsevier Journal of Systems and Software (JSS)

Tosi, D. and Morasca, S. (2015) “Supporting the semi-automatic semantic annotation of web services: A systematic literature review”. Elsevier Journal of Information and Software Technology (IST). Volume 61, Pages 16–32. May 2015.