2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

STRAIT: A Tool for Automated Software Reliability Growth Analysis

Stanislav Chren, Radoslav Micko, Barbora Buhnova, Bruno Rossi
Faculty of Informatics, *Masaryk University*
Brno, Czech Republic
{chren, r.micko, buhnova, brossi}@mail.muni.cz

***Abstract*—Reliability is an essential attribute of mission- and safety-critical systems. Software Reliability Growth Models (SRGMs) are regression-based models that use historical failure data to predict the reliability-related parameters. At the moment, there is no dedicated tool available that would be able to cover the whole process of SRGMs data preparation and application from issue repositories, discouraging replications and reuse in other projects. In this paper, we introduce STRAIT, a free and open-source tool for automatic software reliability growth analysis which utilizes data from issue repositories. STRAIT features downloading, filtering and processing of data from provided issue repositories for use in multiple SRGMs, suggesting the best fitting SRGM with multiple data snapshots to consider software evolution. The tool is designed to be highly extensible, in terms of additional issue repositories, SRGMs, and new data filtering and processing options. Quality engineers can use STRAIT for the evaluation of their software systems. The research community can use STRAIT for empirical studies which involve evaluation of new SRGMs or comparison of multiple SRGMs.**

***Index Terms*—Software Reliability Growth Model, Reliability Analysis, Issue Repository, Issue Report Filtering, Automated Tool**

I. Introduction

Various critical infrastructures, such as power grids, telecommunication, transportation or financial services are controlled by software systems. For such systems, reliability is an essential qualitative attribute. Software Reliability Growth Models (SRGMs) are one of the most well-established approaches for software reliability analysis [1]–[3].

SRGMs can be used to predict reliability of a system in terms of the number of faults, probability of failure or effort needed to reach a required reliability level, which can help with release planning.

Nowadays, SRGMs face several challenges. First, the main models supported by existing tools were developed 20-30 years ago when software development followed mostly the waterfall lifecycle. Therefore, their applicability to modern projects is questionable since they do not account for situations that occur in agile project lifecycles. For example, the software reliability growth curve cannot describe the situation where developers stop testing multiple times during the iterative development. As a result, SRGMs provide output which either under- or overestimates the expected reliability values [4].

Furthermore, the current SRGM tooling support can be decades old and has limited support for extensions with new models developed to tackle the issues of modern projects.

Lack of proper tools affects also the research community. Many studies which propose new SRGMs [4]–[6] use both custom-made scripts and private datasets for the evaluation of the models. The unavailability of the scripts and the datasets decreases the possibility of reuse and replication of the results.

The unavailability of datasets for model evaluation could be solved by leveraging existing public issue repositories, such as Github. However, the data from issue repositories needs to be filtered and processed first, to become usable in SRGM analysis [7]. Ideally, the whole procedure of dataset preparation and application of SRGMs should be automated. Unfortunately, to the best of our knowledge, no existing tool can support the whole process.

In this paper, we propose a new open source tool that aims at addressing the challenges of SRGM analysis of modern software projects by utilizing data available in open issue repositories. The tool is intended to be fully automated and highly extensible. It supports downloading of the issue reports from repositories, the filtration and processing of the data, the application of SRGMs as well as the evaluation of results.

The paper is structured as follows: Section II gives an overview of related tools. Section III details the context of software reliability analysis with SRGMs. In Section IV, we present the STRAIT tool, its architecture and features. Section V outlines the main usage scenarios for our tool.

II. Related Work

Over the years, there have been several dedicated tools developed which provide software reliability analysis via SRGMs (Table I).

TABLE I
Overview or SRGM analysis tools

Name	Language	#SRGMs	Year	Licence
SMERFS [8]	Fortran	12	1988	Free/Open-source
SRMP [9]	Fortran	9	1988	Proprietary
SOFTREL [10]	C	2	1991	Free
CASRE [11]	Fortran	16	1993	Free
SOREL [12]	Pascal	4	1993	Free
SREPT [13]	Java	1	2000	Free
RGA [14]	-	3	2006	Proprietary
SRT [15]	Java	11	2007	Free/Open-source

Many of the existing tools started to emerge some 30 years ago, with only minor evolution in the past decades. This causes issues with their execution in modern operating systems, and their aged user interface. In the past, the lack of interest in supporting modern tools might have been caused by difficult access to suitable datasets. Nowadays, various open source code and issue repositories exist, such as Github, Bitbucket, and Sourceforge. The issue reports in these repositories can be filled by both developers and the community of users, providing a potentially rich data source for the SRGMs.

Furthermore, although most of the tools are free, only two of them (SMERFS and SRT) have source code available. Still, they do not provide any documentation of the possibilities of the tools' extension. Tools such as SMERFS, CASRE or SRT offer a large selection of SRGMs. However, many more models were introduced after the tools were released [16] and there is no tool available that could support them (besides generic mathematical software). Additionally, all of the tools require raw-processed data about fault or failure occurrences for input. They do not offer any automated way of extracting the input data directly from projects issue repositories. Many of the existing tools offer additional functionality related to reliability analysis, such as support for Markov model analysis, and simulation or metrics-based approaches. With STRAIT, we aim at offering a more specialized tool that focuses only on the SRGMs analysis and convenient processing of input data. Instead of providing a complete software reliability analysis suite, we want to offer a tool that is simple to use and can be easily extended in order to adapt to future trends in the application of SRGMs.

III. Software Reliability Growth Models

The reliability analysis models can be broadly classified into black-box and white-box models [17]. The former models analyse the reliability of the system as a whole or at the level of its main components, while the latter consider the internal structure of the system and also focus on behavioural aspects of its components [18].

In this paper, we are addressing SRGMs which are among the oldest and most widely applied black box reliability prediction approaches. The first SRGM was proposed by Jelinsky and Moranda in 1972 [1], and since then more than 100 models have been introduced [16]. Table II shows an overview of basic SRGMs that are also supported by STRAIT.

During the testing and early operation phases of software life cycle, failure events are encountered. They are recorded and the underlying faults that caused them are removed, which results in a process called reliability growth. SRGMs are regression-based models whose purpose is to estimate the parameters of a mean value function $m(t)$ based on the input data: $m(t)$ represents cumulative number of faults detected by the given time t.

Based on the shape of $m(t)$, we can classify SRGMs into three main categories [2] (see Fig. 1):

- *Concave models* – they assume that the total number of faults in software is finite and that it is possible to achieve fault-free software in finite time.
- *S-shaped models* – they also assume that the total number of faults is finite. Furthermore, they assume that early testing is not as effective in fault discovery as the testing in the later stages. Therefore, there is a period in which the number of faults is increasing.
- *Infinite models* – they assume that it is not possible to develop fault-free software, because during faults removal we can introduce new ones.

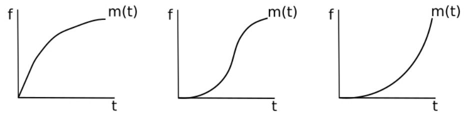
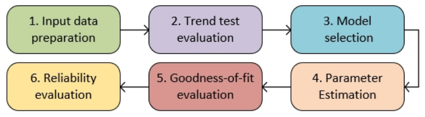


Fig. 1. Basic SRGMs shapes: Concave (left), S-shaped (middle) and Infinite (right)

The process of SRGMs application consists of several steps (as shown in Fig. 2).


1. Input data preparation
2. Trend test evaluation
3. Model selection
4. Parameter Estimation
5. Goodness-of-fit evaluation
6. Reliability evaluation
Fig. 2. Process of SRGM application

The first step is to prepare the input data. The bug reports obtained during testing or operation serve as the main source of input data. One problem that arises is that bug reports usually contain information about failures and not about underlying faults. Ideally, the reports should be processed first to identify the faults which caused the failures and only these faults should be considered in SRGM analysis. However, this process can be very time consuming. Therefore, it is possible to process the bug report just to filter out duplicate entries. Afterwards, reports are considered as substitutes for faults. The technical report of Wood [3] shows that this procedure can still provide credible results. Issue reports can be further filtered so that they fit into the SRGMs assumptions as closely as possible.

The second step is to perform a statistical test such as *Laplace's trend test* to determine whether a trend of decreasing number of failures can be observed. If such a trend cannot be observed, it means there is no reliability growth in the data and the SRGMs would not provide reasonable results.

Next, the SRGM is selected for the application. Since there is no single model that would fit all projects, it is recommended that multiple models are used and the one that describes the data most accurately is selected. To get results from SRGMs, the model's parameters need to be estimated based on input data. For parameter estimation, either the *Maximum likelihood method* or the *Minimum least squares method* are typically used.

Once the parameters are estimated, it is necessary to perform a *goodness-of-fit test* to evaluate how well the model fits the data. It can be performed by common statistical methods, such as computing the correlation coefficient, a Kolmogorov-Smirnov test, a Chi-square test and others. Based on the results, we can select the most appropriate SRGM.

With the model selected, SRGMs can be used to predict a variety of failure data. These include future failure intensity, number of remaining faults, or testing effort required to achieve a given reliability level.

TABLE II
OVERVIEW OF BASIC SRGMS

Model	Type	$m(t)$
Goel-Okumoto [19]	Concave	$a(1 - e^{-bt})$
Goel-Okumoto S-Shaped [20]	S-Shaped	$a(1- (1 + bt)e^{-bt})$
Hossain-Dahiya [21]	Concave	$a\frac{(1-e^{-bt})}{(1+ce^{-bt})}$
Musa-Okumoto [22]	Infinite	$\alpha ln(\beta t + 1)$
Duane [23]	Infinite	αt^{β}

IV. SOFTWARE RELIABILITY ANALYSIS TOOL

STRAIT is a command-line tool written in Java. For the parts requiring mathematical computations, an interface for R-Project [24] is used. The tool is multi-platform, capable of running on MS Windows and Unix-based systems. It is provided as open source under the MIT license. In this section we present an overall architecture and features of the tool.

A. Architecture

STRAIT consists of 10 main components as shown in Fig. 3. The *Core* component handles orchestration of other components. The remaining components are responsible for specific stages of the SRGM application process (as shown in Fig. 2).

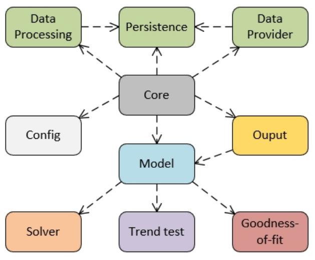


Fig. 3. STRAIT's main components

The *Data Provider* component is responsible for getting issue reports from specified data sources. Currently, STRAIT is able to download issue reports from projects in Github repositories. The raw issue reports are then mapped to *GeneralIssue* objects, which are stored in the local storage.

The storage management is done by the *Persistence* component, which uses an embedded Derby database. If required, the stored data can be exported to CSV files so that they can be reused also by other tools. STRAIT allows the creation of snapshots. Snapshots represent named collections of issue reports for a particular software project and the SRGM analysis is conducted on a specified snapshot. An initial snapshot can be created from the set of issues obtained from the *Data Provider*. This way, it is not necessary to download the issue reports from repositories every time the tool is executed. Further snapshots can be generated by the application of different data filtration and processing options. The snapshot feature simplifies reuse of the same input data across multiple experiments. The *Data Processing* component contains various filters and data processors which can filter or modify stored data based on the selected criteria (see Section IV-B for more details).

The *Model* component holds the implementations of the SRGMs. Currently implemented models are shown in Table II. The Model receives input data for SRGM analysis from the *Core* component (which in turn retrieves them from the *Persistence* component after being processed by the *Data Processor*). The Model component then uses the *Trend test* component to perform trend test on the data—at this moment, Laplace's trend test is implemented. Afterwards, the *Solver* component is used to estimate the model parameters. We support the *Minimum least squares* method for parameter estimation. With the parameters estimated, the *Goodness-of-fit* component evaluates the fit of the data and the model with the *Chi-square* test.

Finally, the *Output* component takes care of generating output with results of SRGM analysis. STRAIT supports either generation of plain text files which are more suitable for automated processing or HTML files which are more human-readable. Both text and HTML files contain information about the models used, the estimated parameters, the results of the *trend test* and the *goodnes-of-fit test* with the list of applied data filters and processors together with their parameters. Moreover, the HTML output also contains an interactive visual representation of the mean value functions of SRGMs. A sample output is available at STRAIT.

The whole execution of STRAIT is highly customizable. The execution options are handled by the *Config* component. The options can be specified as command-line arguments, or they can be stored in multiple configuration files enabling reusability of the configurations.

B. Data Filtering and Processing

STRAIT offers two mechanisms for manipulation of the data downloaded from issue repositories. First, the *Data Filters* are used to reduce the number of issues retrieved from a repository. Some issues might for example represent feature requests or problems not resulting in failures, which are not relevant for the SRGM analysis. Furthermore, some issues might need to be filtered out so that the input data follow the assumptions of the individual SRGMs. For example, we might require only issues representing a critical problem, issues that have been fixed or issues that appeared only after the software release [5]. Additionally, filters can be used to limit the scope of the analysis by focusing only on issues for particular software component or during a specified time period. Presently, the STRAIT tool allows filtering based on the general issue attributes which are shown in Table III.

Second, STRAIT uses *Data Processors* for modifying the issues. This can be used for transforming issue reports into data representations used by the SRGMs. Moreover, they can be used for changing the values of issue attributes, for

TABLE III
OVERVIEW OF ATTRIBUTES FOR ISSUE FILTERING

Attribute	Type	Attribute	Type
createdAt	Date	htmlUrl	String
closedAt	Date	labels	String[]
updatedAt	Date	userName	String
comments	String[]	userEmail	String
number	int	milestoneCreatedAt	Date
body	String	milestoneDueOn	Date
state	String	milestoneDescription	String
title	String	milestoneState	String
url	String	milestoneTitle	String

example by enriching the list of issue's labels based on some textual analysis or changing the structure of the issue bodies or anonymizing the authors of issue reports.

C. Extensibility

The possibility to extend or provide new implementations of the main components is one of the main features of STRAIT. The extensibility is achieved via using interfaces as dependencies between the components following the Dependency Inversion (DI) design principle [25]. This way, the addition of new features is possible by providing new implementation of the interfaces either directly or by extending the default classes implementing the interfaces. In STRAIT, the following extensions are available:

- *Data source* – by implementing the *DataProvider* interface, it is possible to add support for different data sources, such as different issue repositories, or even databases or text files.
- *Filters* and *Processors* – new filtering and data processing methods can be added by implementing the *DataProcessor* and *DataFilter* interfaces. This could include for example a complex filtering of issues based on parsing issues' bodies and comments via textual analysis methods or methods utilising machine-learning techniques.
- *Output generators* – implementation of the OutputWriter interface enables generation of output into additional file formats.
- *Models* – new SRGMs can be added by implementing the *Model* interface.
- *Computation methods* - additional methods for trend testing, parameter estimation and goodness-of-fit testing can be implemented via their respective interfaces.

V. USAGE SCENARIOS

In this section, we briefly outline several possible usage scenarios of STRAIT.

A. Reliability analysis of software projects

The tool can be used for reliability analysis of selected software projects by providing predictions of future failure occurrences, about total faults in the software and the time required to reach the desired reliability level. Since the tool can be configured to run autonomously, it can be integrated with existing continuous integration tools to provide reliability estimates throughout the whole development cycle. The developers or quality engineers could configure it to target the analysis of specific components or project versions. Due to its extensibility, the tool can be customized to meet any project-specific needs. For example, the integration can lead to software reliability growth models integrated in the software debugging process [26].

B. Empirical evaluation of new SRGMs

STRAIT's architecture allows the extension with additional models. Since STRAIT enables access to a large number of projects and their issue repositories, the new models can be evaluated on a variety of input data from different types of projects (for example, by providing a large-scale evaluation of a model in the context of open source software projects [27]—such as the one described in Aggarwal et al. [28]). Furthermore, with the ability to create data snapshots and configuration files, multiple models can be compared on the same datasets and with the same parameters which increases replicability of the experiments.

C. Investigation of applicability of SRGMs in modern projects

Many of the commonly used SRGMs were developed decades ago and target larger-scale projects developed with a waterfall development lifecycle. However, nowadays the agile development methodologies are prevalent. It is unclear to what extent the SRGMs can be used to analyse such projects—as discussed, for example, in Rawat et al. [29]. The STRAIT tool can help with the investigation of the models' applicability due to the access to many datasets from modern projects.

D. Supporting textual analysis and issue classification

SRGMs models can integrate knowledge derived from text mining research for the classification of bug reports. For example, STRAIT and SRGMs can be useful to determine the priorities of the bug-fixing process by complementing features derived from text analysis—as presented in Kumari et al. [30].

VI. CONCLUSION

In this paper, we presented STRAIT, a new tool for reliability analysis based on SRGMs. The main features of STRAIT are the utilization of online issue repositories as sources for input data, the automated execution and support for the whole process of SRGM analysis, including the data filtration and processing as well as the evaluation of results. Moreover, the tool is designed with high extensibility in mind to allow future enhancement of all its components. The tool is applicable to a variety of scenarios for software developers, quality engineers and the research community.

In our future work, we will focus on providing support for additional issue repositories and improve the data processing capabilities of the tool.

ACKNOWLEDGMENT

This research was supported by ERDF ”CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence” (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

References

[1] Z. Jelinski and P. Moranda, "Software reliability research," in *Statistical computer performance evaluation*. Elsevier, 1972, pp. 465–484.

[2] M. R. Lyu, *Handbook of software reliability engineering.* IEEE Computer Society Press, 1996.

[3] A.Wood, "Software reliability growth models," Tandem Computers Inc., Cupertino, CA 95014, Tech. Rep. 130056, 1996.

[4] H. Washizaki, K. Honda, and Y. Fukazawa, "Predicting release time for open source software based on the generalized software reliability model," in *2015 Agile Conference*, Aug 2015, pp. 76–81.

[5] H. Koziolek, B. Schlich, and C. Bilich, "A large-scale industrial case study on architecture-based software reliability analysis," in *2010 IEEE 21st international symposium on software reliability engineering*. IEEE, 2010, pp. 279–288.

[6] K. Honda, H. Washizaki, Y. Fukazawa, K. Munakata, S. Morita, T. Uehara, and R. Yamamoto, "Detection of unexpected situations by applying software reliability growth models to test phases," in *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Nov 2015, pp. 2–5.

[7] B. Buhnova, S. Chren, and L. Fabriková, "Failure data collection for reliability prediction models: A survey," in *Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures*. ACM, 2014, pp. 83–92.

[8] W. H. Farr and O. D. Smith, "A tool for statistical modeling and estimation of reliability functions for software: Smerfs," *Journal of Systems and Software*, pp. 47–55, 1988.

[9] G. Stark, "A survey of software reliability measurement tools," in *International Symposium on Software Reliability Engineering(ISSRE)*, 1991, pp. 90–97.

[10] T. Chu, M. Yue, M. Martinez-Guridi, and J. Lehner, "Review of quantitative software reliability methods," *International Topical Meeting on Probabilistic Safety Assessment and Analysis 2011, PSA 2011*, 2011.

[11] M. R. Lyu and A. Nikora, "Casre: a computer-aided software reliability estimation tool," in *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, 1992, pp. 264–275.

[12] K. Kanoun, M. Kaaniche, J. Laprie, and S. Metge, "Sorel: A tool for reliability growth analysis and prediction from statistical failure data," in *The Twenty-Third International Symposium on Fault-Tolerant Computing*, 1993, pp. 654–659.

[13] S. Ramani, S. S. Gokhale, and K. S. Trivedi, "Srept: software reliability estimation and prediction tool," *Performance Evaluation*, pp. 37–60, 2000.

[14] Rga: Reliability growth analysis and repairable system analysis. [Online]. Available: https://www.reliasoft.com/products/reliability-analysis/rga

[15] A. D. B. M. A. A. Boon, I. C. Ramos, "A new statistical software reliability tool," in *Proceedings of VVSS2007-verification and validation of software systems*, 2007, pp. 125–139.

[16] M. R. Lyu, "Software reliability engineering: A roadmap," in *Future of Software Engineering, 2007. FOSE'07*. IEEE, 2007, pp. 153–170.

[17] W.-L. Wang, D. Pan, and M.-H. Chen, "Architecture-based software reliability modeling," *Journal of Systems and Software*, vol. 79, no. 1, pp. 132–146, 2006.

[18] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software & Systems Modeling*, vol. 7, no. 1, p. 49, 2008.

[19] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, no. 3, pp. 206–211, 1979.

[20] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, pp. 475–484, 1983.

[21] S. A. Hossain and R. C. Dahiya, "Estimating the parameters of a non-homogeneous poisson-process model for software reliability," *IEEE Transactions on Reliability*, pp. 604–612, 1993.

[22] K. O. J. D. Musa, A. Iannino, *Software reliability: measurement, prediction, application*. McGraw-Hill, Inc., 1987.

[23] J. T. Duane, "Learning curve approach to reliability monitoring," *IEEE Transactions on Aerospace*, pp. 563–566, 1964.

[24] R. C. Team *et al.*, "R: A language and environment for statistical computing," 2013.

[25] R. C. Martin, "Design principles and design patterns," *Object Mentor*, vol. 1, no. 34, p. 597, 2000.

[26] M. Cinque, D. Cotroneo, A. Pecchia, R. Pietrantuono, and S. Russo, "Debugging-workflow-aware software reliability growth analysis," *Software Testing, Verification and Reliability*, vol. 27, no. 7, p. e1638, 2017.

[27] B. Rossi, B. Russo, and G. Succi, "Modelling failures occurrences of open source software with reliability growth," in *IFIP International Conference on Open Source Systems*. Springer, 2010, pp. 268–280.

[28] A. G. Aggarwal, V. Dhaka, N. Nijhawan, and A. Tandon, "Reliability growth analysis for multi-release open source software systems with change point," in *System Performance and Management Analytics*. Springer, 2019, pp. 125–137.

[29] S. Rawat, N. Goyal, and M. Ram, "Software reliability growth modeling for agile software development," *International Journal of Applied Mathematics and Computer Science*, vol. 27, no. 4, pp. 777–783, 2017.

[30] M. Kumari, A. Misra, S. Misra, L. Fernandez Sanz, R. Damasevicius, and V. Singh, "Quantitative quality evaluation of software products by considering summary and comments entropy of a reported bug," *Entropy*, vol. 21, no. 1, p. 91, 2019.

Appendix A
Tool Installation and Availability

A. Tool Availability

The tool is available as open-source under the MIT license for free. The project website is at: https://lasaris.fi.muni.cz/STRAIT. The website contains the source code, compiled binaries, user manual and sample examples with showcase of output.

B. Tool Requirements

- Java, version 8+: https://www.java.com/en/
- R Project, version 3.5.0+: https://cloud.r-project.org/

C. Installation for MS Windows

1) If Java is not installed, run the Java installer.
2) Append Java `bin` directory to the environment PATH variable, e.g. `C:\Program Files\Java\jdk1.8.0_171\bin`
3) If R Project is not installed, run the installer.
4) Run the `R.exe` and in the console, install the `rJava` package: `install.packages("rJava")`
5) Set the environment variables for R Project:
   - `R_HOME= Path-to-R-install-directory` e.g. `R_HOME = C:\Program Files\R-3.5.1`
   - `path = \%R_HOME\%\bin\x64`
   - `path = \%R_HOME\%\library\rJava\libs\x64`
   - `path = \%R_HOME\%\library\rJava\jri\x64`

   If 32-bit operating system is used the `\x64` part should be replaced with `\i386`

D. Installation for Unix

The Unix installation assumes that the Aptitude package manager is available. The following commands should be executed via terminal.

1)
```
sudo apt-get install default-jdk.
```
2)
```
sudo apt-key adv keyserver
keyserver.ubuntu.com recv-keys
E298A3A825C0D65DFD57CBB651716619E084DAB9
```

3) `sudo add-apt-repository deb https://cloud.r-project.org/bin /linux/ubuntu bionic-cran35/`
4) `sudo apt-get install r-base`
5) `sudo -i R`
6) `install.packages(rJava)`
7) Set the `R_HOME` variable

E. Tool usage

The tool can be executed from command-line by running:

- `java -jar strait.jar [OPTIONS]`

For options, multiple arguments can be used. An overview of command-line options is in Table IV. The tool also prints a list of all options if no argument is provided. The help for each option can be accessed by running:

- `java -jar strait.jar --help OptionName`

A simple execution of the tool to evaluate the *spring-boot* project hosted at Github may look like:

- `java -jar STRAIT.jar --url https://github.com/spring-projects/spring-boot --evaluate --filterLabel bug error fault issue fail defect --filterTime 2018-01-01 2018-12-01 --snapshotName springBoot --solver ls`

With the `--url` option, we specify the location of the project. The option `--evaluate` states the intention to execute the SRGM analysis. We do not specify any particular models, therefore, all of the available SRGMs will be applied. The `--filterLabel` option is followed by a list of labels. Issue reports which do not contain any of the labels will be filtered out. Furthermore, with `--filterTime` we limit the time period for which Issue reports will be considered. With the `--snapshotName` option, we specify the name of the snapshot for storing the gathered issues. At the second run, the snapshot name can be provided instead of the URL: the local data stored in the snapshot will be used for the analysis. The `--solver` option specifies that the minimum least squares solver will be used for parameter estimation.

TABLE IV
OVERVIEW OF THE COMMAND-LINE OPTIONS

Short option	**Long option**	**Arguments**
-h	--help	—
-url	—	[Repository URL]
-asl	--allSnapshotsList	—
-sn	--snapshotName	—
-cf	—	[Path to config file]
-sl	--snapshotsList	—
-s	--save	[Data format]
-e	--evaluate	—
-p	--predict	[Number of time units for prediction]
-ns	--newSnapshot	[Name of the new snapshot]
-fl	--filterLabel	[Label names]
-fc	--filterClosed	—
-ft	--filterTime	[From] [To]
-ms	--models	[Models]
-pt	--periodOfTesting	[Testing period time unit]
-tb	--timBetweenIssues	[Time unit for TBF]
-gm	--graphMultiple	—
-so	--solver	[Solver]
-out	—	[Output type]