Calibrating a Customizable System Dynamics Simulation Model of Generic Software Development Processes

Keyvan Khosrovian[1], Dietmar Pfahl[1, 2, 3], Vahid Garousi[1]

[1]Schulich School of Engineering, University of Calgary, Canada
[2]Simula Research Laboratory, Norway,
[3]Department of Informatics, University of Oslo, Norway
{kkhosrov, dpfahl, vgarousi}@ucalgary.ca

Abstract. GENSIM 2.0 is a customizable system dynamics simulation model of generic software development processes which takes as input the specifics of software development projects (e.g. size of the code document, headcount of the developer team and their skills) and generates as output a broad range of distinct variables (e.g. software product quality, total project effort) of interest to different users of the model. This technical report is dedicated to elaborate on current calibration of GENSIM 2.0, its calibration parameters and the source from which they can be potentially calibrated.

Keywords. Model Calibration, Customizable Simulation Model, System Dynamics, Software Development Process.

TABLE OF CONTENTS

1 INTRODUCTION

Managerial aspects of software development projects including planning, resource allocation, workforce training, etc. have a significant impact on their performance, i.e., duration, effort and the quality of the final product. However, these areas have not received enough attention by the software engineering research community and still most problems involving these issues are often dealt with relying on expert knowledge and intuition only.

Empirical research is essential for developing theories of software development, transforming the art of software development into an engineering discipline and hence overcoming the aforementioned problem. However, empirical theories and knowledge require evidence of the efficiency and the effectiveness of the tools, techniques or practices in various application contexts.

Controlled experiments and surveys are methods commonly used for empirical research, however, they are costly. Hence, support for making decisions on which experiments and case studies are more worthwhile to spend effort and time on, would be helpful. GENSIM 2.0 [1] is a customizable and reusable software development process simulation model developed to address this issue. Inspired by the idea of frameworks in software systems, GENSIM 2.0 consists of a small set of generic reusable components which can be composed to model a wide range of different software development processes.

These components capture key attributes of the entities involved in different building blocks of the development processes affecting the project performance. What makes the model results interesting and hard to predict, are the numerous complex relationships and influences between attributes. The current implementation of GENSIM 2.0 simulates the well-known V-Model software development process. It consists of three development phases (requirements specification, design and code), each consisting of a document development activity and a verification activity (e.g. Inspection) carried out on the developed artifacts (requirements specification, design, code), and three validation activities (unit, integration and system test).

GENSIM 2.0 has many parameters. Parameters can represent model inputs, outputs, or they are used to calibrate the model to expert knowledge and empirical data specific to an organization, process, technique or tool. Input parameters represent project specific information such as estimated product sizes, developer skills and project policies. Project policies may, for example, define how verification and validation activities should be combined, or whether design artifacts should be reworked if defects originating in the design phase are only detected in the code by subsequent verification and validation activities. Calibration parameters represent organization specific information that typically is retrieved from measurement programs and empirical studies. Input and calibration parameters are similar in the sense that they both remain constant during simulation time. Output parameters represent values that are calculated by the simulation engine .

Besides their usage in the simulation model, i.e., input, calibration and output, GENSIM 2.0 parameters can also be categorized according to the entity which they represent an attribute of, i.e., process, product, resource and project. Additionally, they can be classified according to the view they are associated with. Which type of view, i.e., product, defect, resource or state flow view, a parameter is associated with depends on the primary effect of the attribute it represents. A complete description of all parameters and their defining equations can be found in [1].

Model calibration refers to the adjustment of the simulation model calibration parameters until, for a certain input, the model's generated output matches a dataset observed in a real-world environment. Software process simulation model calibration can be done based on expert estimates or through parameter fitting based on historic data collected from organization-specific or public repositories. In

cases when such data is not available, calibration can also be done using data published in the software engineering literature, which is in fact a mix of different sources. For the current version of GENSIM 2.0, data published in the literature was used. Calibration is an important step in the development of simulation models and is intended to ensure sound behavior of the model. Calibration is also required to build confidence in the simulation results.

GENSIM 2.0 is designed to be reused, customized and applied to tackle emerging software development related problems of any kind. This technical report is not only intended to show the current calibration values of GENSIM 2.0, but also to elaborate more on its calibration parameters themselves which in turn allows for easy re-calibration of the model. The rest of this technical report is structured as follows: Sections 2 discusses the previous works in the area of simulation model calibration. Section 3 presents GENSIM 2.0 calibration parameters with details. Section 4 discusses sources that can potentially be used to calibrate GENSIM 2.0 and similar models. Section 5 illustrates the values currently used for calibration of GENSIM 2.0. Finally, section 6 discusses conclusions of the presented material and suggests future steps.

2 RELATED WORK

The calibration of SD (System Dynamics) models and the associated difficulties have been discussed in the literature for many years. The approach used for calibrating GENSIM 2.0 can be characterized as calibration "by hand". Calibration "by hand" is an iterative process in which the modeler "examines differences between simulated output and data, identifies possible reasons for those differences, adjusts model parameters in an effort to correct the discrepancy, and re-simulates the model, looping back to the first step." [2]. A discussion of the problems associated with this approach, i.e., limited reliability of the calibration process which strongly relies on the expertise of the modeler, can be found in [2]. The alternative to calibration "by hand" is automated calibration. Automated calibration, while improving reliability, is not an easy endeavor either. A thorough discussion of the problems associated with automatic calibration of SD models, and ways to mitigate these problems, can be found in [3].

In [4], the authors argue that in order to truly realize the benefits of the use of software process simulation models, i.e., as virtual software engineering laboratories, process simulation has to be combined with empirical studies. They propose that this combination should be done in such a ways that firstly, empirical knowledge is used for development and calibration of simulation models and secondly, results from process simulation are used for supporting real experiments. They provide guidelines on how to achieve these objectives as well. The idea of combining simulation models with empirical studies is not limited to [4]. Pervious studies such as [5] had proposed the combination of process simulation models and empirical data in support of decision analysis in software development. In [5] the authors argue that since for decision-making, in practice, there is a need to apply the available empirical knowledge, expensive empirical work should be systematically extended with simulation to fill the gaps in the variable space of the decision-making context.

[6] is an example of the studies that used the concepts provided in [5] for developing and calibrating a simulator in support of decision-making for planning the effort to be allocated to inspections in different software engineering development phases. In [6], the authors elaborate on how isolated pieces of empirical data could be combined and used for developing and calibrating simulation models, in order to provide project managers with comprehensive data that allows meaningful comparison of different process alternatives. The approach taken in the study represented in this technical report is no different from the work discussed above from a methodological point of view. The only difference is that the use of empirical data in calibrating the model is explained for the entire set of model parameters which enables easier re-calibration and extended experimentation with the model as suggested in [4].

3 GENSIM 2.0 CALIBRATION PARAMETERS

GENSIM 2.0 parameters including its calibration parameters can be classified in different ways [1]. Besides their usage in the simulation model, i.e., input, calibration and output, the GENSIM 2.0 parameters can also be categorized according to the entity which they represent an attribute of. In GENSIM 2.0 it is assumed that parameters can represent attributes of four different entity types namely process, product, resource and project. Attributes of the process category define the structure of the development process or the specifics of how different activities are carried out. Attributes of the product category define the specifics of the software product that is being developed. Attributes of the resource category capture the specifics of the available resources for the project including tools/techniques and the workforce. Finally, attributes of the project mostly capture the software development context and managerial policies.

GENSIM 2.0 parameters can also be classified according to the view that they are associated with. In order to capture the main dimensions of project performance, i.e., project duration, project effort, and product quality, and to explicitly represent the states of activities, GENSIM 2.0 is implemented in separate views as follows:

1. **Product Flow View** models the specifics of how software artifacts are sent back and forth during and between different activities of the development project.
2. **Defect Flow View** models the specifics of how defects are moved around (generated, propagated, detected and corrected) as different software artifacts are processed
3. **Resource Flow View** models the specifics of how different resources (workforce, techniques/tools) are allocated to different activities of the development project.
4. **State Flow View** models the specifics of how the states of different entities as explained in Section 4.1 change during the development project.

Which type of view, i.e., product, defect, resource or state flow a parameter is associated with depends on the primary effect of the attribute it represents. For example, *Required skill level for code dev* is an input parameter representing a managerial policy that primarily affects the resource flow of the code development activity. For a more detailed description of GENSIM 2.0 parameters, their classifications and the views refer to [1]. In the following we exclusively discuss the calibration parameters.

Since all three development phases of GENSIM 2.0 are modeled using one single pattern (see [1] for details), corresponding sets of parameters (including calibration parameters) have been defined for each development phase. To give an example, in the code phase model, the parameter *Code ver effectiveness* is used to represent the effectiveness of the code verification technique in detecting defects of the code artifacts. Meanwhile, the parameter *Design ver effectiveness* is defined in the design phase model to specify the effectiveness of the design verification technique with regards to detecting defects of the design artifacts. The same rule applies for the group of parameters used in modeling different validation phases. Table 1 and Table 2 list and describe the calibration parameters used in modeling the code development and system test phases of GENSIM 2.0, respectively. As mentioned, corresponding tables could be defined for other development or validation phases.

Table 1: Calibration parameters of the code development phase

	Parameter Name	Unit	Attribute	View	Description
1	Code rework effort for code faults detected in CI	PD/Defect	Process	C-D	The amount of effort that should be spent for fixing a defect in the code artifacts if detected during code verification
2	Code rework effort for code faults detected in UT	PD/Defect	Process	C-D	The amount of effort that should be spent for fixing a defect in the code artifacts if detected during unit test
3	Code rework effort for code faults detected in IT	PD/Defect	Process	C-D	The amount of effort that should be spent for fixing a defect in the code artifacts if detected during integration test
4	Code rework effort for code faults detected in ST	PD/Defect	Process	C-D	The amount of effort that should be spent for fixing a defect in the code artifacts if detected during system test
5	Average design to code conversion factor	KLOC/Page	Product	C-P	The multiplier specifying how many kilo lines of code has to be developed per each page of the design artifacts.
6	Average # of UT test cases per code size unit	Test case/ KLOC	Product	C-P	The number of test cases that has to be developed in order to unit test the code artifacts
7	Average design to code fault multiplier	N/A	Product	C-D	The multiplier specifying how many faults will be committed in the code artifacts because of an undetected design defect.
8	Maximum code ver. effectiveness	N/A	Resource	C-D	The effectiveness of the code verification technique with regards to defection of code defects if applied by an optimally skilled verifier
9	Maximum code ver. rate per person per day	KLOC/ Person-Day	Resource	C-P	The amount of code artifacts that can be verified in one day by an optimally skilled verifier.
10	Initial code dev. rate per person per day	KLOC/ Person-Day	Resource	C-R	The amount of code artifacts that can be developed (not reworked) in one day by one optimally skilled developer
11	Minimum code fault injection rate per size unit	Defect/KLOC	Resource	C-D	The number of defects that an optimally skilled developers in a size unit of the code artifacts.

CI = Code Inspection, UT = Unit Test, IT = Integration Test, ST = System Test

Table 2: Calibration parameters of the system test phases

	Parameter Name	Unit	Attribute	View	Description
1	Maximum ST effectiveness	N/A	Resource	S-D	The effectiveness of the system testing technique with regards to defection of code defects if applied by an optimally skilled tester
2	Average ST productivity per person per day	KLOC/ Person-Day	Resource	S-P	The amount of code artifacts that can be system tested (including test case development) in one day by a tester. Since the test case development and execution activities are considered together in this parameter, It is assumed that the skill level of the testers does not have any effect on this rate.
3	Maximum # of ST test cases developed per person per day	Test case/ Person-Day	Resource	S-P	The number of system test cases that can be developed in one day by an optimally skilled tester.
4	Average # of ST test cases executed per person per day	Test case/ Person-Day	Resource	S-P	The number of system test cases that can be executed in one day by a tester. It is assumed that the skill level of the testers does not have any effect on the test case execution rate.

4 SOURCES OF CALIBRATION

Many different sources could potentially be used for calibration of software process simulation models, expert estimates are one of them. In [7] the authors discuss how expert knowledge and data from project data bases was used to develop and calibrate PSIM, a software process simulation model developed in order to show the usefulness of SD modeling in tackling software project management issues in a development department of Siemens. Despite being a good source in many situations, expert knowledge is considered to be subjective and is often complemented with real-world empirical data collected through measurement programs. Whenever simulation models are developed for specific purposes within specific organizations, this data could be gathered from different repositories within the organization. These repositories often contain data collected from many different projects carried out in the organization and are specific to its environment.

Whenever, expert knowledge or organization-specific data is not available or not applicable due to their relevance only in a specific context, online repositories (such as [8], [9] and [10]) which often contain cross-organizational data could be used to collect the necessary information for calibration of simulation models. The Software Information Repository [8], is maintained by an online community. Its members contribute and exchange information and data regarding process improvement activities around the world in order to build a knowledge base of this information which can be used by any interested individual. The PROMISE Software Engineering Repository presented in [9] is a collection of datasets and tools is made publicly available to support researchers in building predictive software models. The Software-artifact Infrastructure Repository presented in [10] has been designed and constructed in

support of controlled experimentation with software testing techniques. It contains many Java and C software systems, in multiple versions, along with their supporting artifacts such as test suites, fault data, scripts and manuals on how to experiment with the provided material. Results obtained from experimentation with the provided systems could be used to obtain estimates of different calibration parameters used in simulation models.

Another source of data that can be used for calibration of software process simulation models is the software engineering literature. The software engineering literature now contains many publications reporting data collected from different kinds of sources. Some of these publications such as [11] report data collected from an experiment carried out with groups of students or professionals. In [11] an experiment is carried out with one group of 42 advanced students and another group of 32 professionals to compare the effectiveness of three different testing strategies. Another group of these publications are the ones that report coarse-grained data gathered from real-world projects in industrial settings and mostly discuss the lessons learnt and the experiences. [12], is an example of these publications. In [12], the authors present high-level information regarding couples of projects in Motorola and discuss how CMM [13] helped them to improve the performance of their projects. The last group of publications comprises the results from surveys carried out on the existing literature itself. In [14] and [15] for example the authors have gathered and compared data reported in many other existing publications. For the current version of GENSIM 2.0 the SE literature has been chosen as the source for calibration.

5 CURRENT PARAMETER CALIBRATION OF GENSIM 2.0

In this section current values of GENSIM 2.0 calibration parameters together with how and from what source they were obtained are presented. The next section discusses the calibration of the development phases (requirements specification, design and code) and after that calibration of the validation phases are explained.

5.1 CALIBRATION OF THE DEVELOPMENT PHASES

This subsection describes how and from which sources calibration parameters used in the simulation modeling of the development phases of GENSIM 2.0 were calibrated for its current version. To make the understanding of the overall material easier, the parameters are divided into different groups and then the calibration is presented for each group.

Initial development parameters: These parameters specify the productivity of the developers in developing software artifacts for the first time. For example, *Initial code dev. rate per person per day* specifies the speed with which the developers develop software code artifacts for the first time. Three parameters in the model belong to this category as shown in Table 3. To calibrate these parameters, the COCOMO II [16] post architecture model was used. COCOMO II is a model that generates estimations of the cost, effort, and schedule of a new software development activity across 4 life-cycle phases of software development, i.e., Plans and Requirements, Product Design, Programming and Integration and test. Additionally, within each life-cycle phase, it provides effort and time estimations for 8 different activities, i.e., Requirements Analysis, Product Design, Programming, Test Planning, Verification and Validation, Project Office, CM/QA and manuals.

In order to calculate the initial development parameters, specifications of a hypothetical system was inputted into the COCOMO II model and the estimations were generated. Following explains the details on how the *Initial code dev. rate per person per day* parameter was calculated:

1. The effort spent on the programming activity during the plans and requirements, product design and programming phases were summed up to obtain E. The schedule time spent on the programming activities of the three aforementioned phases were also summed up to obtain S.

2. The resultant effort (E) was divided by the resultant schedule time (S) to obtain the average number of developers allocated for the programming activity (D) during S.

3. It was assumed that S includes the time spent for the initial code development activity, rework due to faults detected in the code verification activity and rework due to defects detected in the unit test activity. Since, as explained in [1], the effect of learning is incremented by 1 every time an artifact is processed (developed, reworked or verified), the learning effect is 2 right after it is verified completely for the first time and is 3 after it is completely reworked due the faults detected during verification. Hence, it is assumed as an approximation that the effect of learning equals an average of 2.5 during the period that it is reworked due to faults detected in code verification. As a result, rework due to code faults detected in the code verification activity is 2.5 times faster than the initial development. For the same reason, rework due to defects detected in unit testing is assumed to be 3.5 times faster than the initial development.

4. With the above assumptions, solving the equation below for x will determine the overall schedule time spent on the initial code development activity (T).

$$S = x + (1/2.5)\,x + (1/3.5)\,x$$

5. Having calculated T, the productivity of each of the developers in the initial code artifact development activity could be obtained using the following formula assuming the size of the code artifacts:

$$p = \frac{Size\,of\ the\ \ code\ artifacts}{D \times T}$$

Corresponding calculations were applied to calculate the Initial *design dev. rate per person per day* and the *Initial requ spec dev. rate per person per day* parameters. In the hypothetical system specified for the current calibration of GENSIM 2.0, the size of the requirements specification artifact was assumed to be 50 pages. Considering the conversion factor between the requirements specification and the design artifacts, size of the overall design artifacts is calculated as 1550 pages by the simulation model. To calculate the size of the module code artifacts, their average size is calculated initially using the conversion factor between the design and code artifacts. After calculating the average module code size, in order to avoid having the same size for all the modules, their average size is multiplied by a uniformly generated random number between 0.5 and 1.5. With these calculations, the size of the overall system is calculated as 307.15 KLOC.

Table 3: Calibration values of the initial development parameters

Parameter Name	Unit	Value
Initial requ. dev. rate per person per day	Page/Person-Day	0.07
Initial design dev. rate per person per day	Page/Person-Day	0.829
Initial code dev. rate per person per day	KLOC/Person-Day	0.048

Correction effort parameters: These parameters specify the amount of effort that needs to be spent for correcting the detected defects. The values of these parameters depend greatly on the time of the

detection of the defect. For the code artifacts, to capture this difference, four distinct parameters where specified in GENSIM 2.0 as shown in the first four rows of Table 4. To calculate values of these four parameters two different sources were used ([14] and [17]). In [14], the author presents averages of many reported values in the literature for each of these four parameters as absolute numbers. In [17] the author presents these values as relative numbers. He claims that if the cost of fixing a defect if detected during the implementation is 1, then the cost of fixing it if detected during integration test is 2.5 and if detected during system test is 13. For the defects detected during code verification and unit test, values reported in [14] were used. For the defects detected in integration test and system test, we assumed that in [17], by defects detected during implementation, the author means, the defects detected during unit test. Hence, the value reported in [14] for the defects detected during unit test, was multiplied by 2.5 and 13 to calculate the parameters for integration and system test respectively.

For the requirements specification and design artifacts, since no data was found regarding the difference of the correction effort of the detected defects depending on the time of detection, the values reported in [14] were used to specify the amount of the effort required for correction of the defects of these artifacts regardless of the time of detection.

Table 4: Calibration values of the correction effort parameters

Parameter Name	Unit	Value
Code rework effort for code faults detected in CI	Person-Day/Defect	0.3387
Code rework effort for code faults detected in UT	Person-Day/Defect	0.4325
Code rework effort for code faults detected in IT	Person-Day/Defect	1.0815
Code rework effort for code faults detected in ST	Person-Day/Defect	5.6225
Design rework effort per fault	Person-Day/Defect	0.29
Requ. spec. rework effort per fault	Person-Day/Defect	0.125

Artifact Conversion Parameters: These parameters including *Average requ. spec. to design conversion factor* and *Average design to code conversion factor*, determine the relationship between the sizes of the artifacts developed in up-stream phases and the sizes of the artifacts developed in the down-stream phases. These parameters should be calibrated by collecting information from multiple projects for a certain context. Since no such data was found published in the software engineering literature, for the current calibration of GENSIM 2.0, these values where assumed hypothetically as shown in Table 5.

Table 5: Calibration values of the artifact conversion parameters

Parameter Name	Unit	Value
Average requ. spec. to design conversion factor	Page/Page	31
Average design to code conversion factor	KLOC/Page	0.2

Fault Conversion Parameters: These parameters determine the number of faults that an undetected defect in the artifacts of an up-stream phase causes in the artifacts of the down-stream phase. Similar to artifact conversion parameters, these parameters should be calibrated using information collected from multiple projects for a certain context. Since no such data was found published in the software engineering literature, for the current calibration of GENSIM 2.0, these values where assumed hypothetically as shown in Table 6.

Table 6: Calibration values of the fault conversion parameters

Parameter Name	Unit	Value
Average requ. spec. to design fault multiplier	N/A	3
Average design to code fault multiplier	N/A	3

Verification Rate Parameters: These parameters determine the speed with which the verification activities are carried out. For calibration of these parameters the values presented in [14] were used as shown in Table 7.

Table 7: Calibration values of verification rates

Parameter Name	Unit	Value
Maximum requ. spec. ver. rate per person per day	Page/Person-Day	8
Maximum design ver. rate per person per day	Page/Person-Day	30
Maximum code ver. rate per person per day	KLOC/Person-Day	0.6

Defect Injection Parameters: In GENSIM 2.0, defects in artifacts are caused by two different sources, firstly, the defects propagated from the up-stream phases and, secondly, the faults that the developers commit themselves. Defect injection parameters determine the rate with which the developers themselves inject defects in the artifacts. These parameters were calibrated using a Defect Containment Matrix [18]. A Defect Containment Matrix maps the phase in which a defect originated to the phase in which the defect was detected. For the current calibration of GENSIM 2.0, a hypothetical matrix, as illustrated in Table 8, was assumed and the injection rates were calculated according to this matrix. Rows represent the phases in which the defects are detected. Columns represent the origin of the detected defects. For example, it can be seen that a total number of 2736 design defects were detected in the design phase, 1181 of which were originated in the requirements phase, i.e., were injected in the design artifacts due to undetected requirements specification defects.

Table 8: Hypothetical Defect Containment Matrix

	Stage Originated							
Stage Detected	Requirements	Design	Code	UT	IT	ST	Field	Total
Requirements	1515							1515
Design	1181	1555						2736
Code	402	912	2421					3735
Unit test	200	420	1525	37				2182
Integration test	191	223	370	7	1			792
System test	89	114	114	5	0	10		332
Field	5	8	23	0	0	0	0	36
Total	3583	3232	4453	49	1	10	0	11328

Using Table 8 we calculate the total number of defects that the developers have injected in different artifacts, i.e., requirements specification, design and code. In the following the calculation process is explained for the requirements specification artifact:

1. The number of requirements specifications defects that were detected in the design phase is calculated. From Table 8 it can be seen that 1181 design defects were detected in the design phase that originate in the requirement specification phase. Considering the fault multiplier between the requirements specification defects and the design defects (*Average requ. spec. to design fault multiplier*) which is assumed to be 3, it results that 394, i.e., 1181 divided by 3 requirements specification defects have been detected in the design phase.

2. The above calculation is done for the code defects that were found in the code phase and originated in the requirements specification phase. However, in this step, the applied fault multiplier is 9 because every requirements specification causes 3 design defects and every design defect causes 3 code defects. Hence, 45 (402 divided by 9) requirements specification defects were detected during the code phase.

3. The previous step with the same fault multiplier is carried out for the unit test, integration test, system test and the field phases.

4. The numbers of the requirements specification defects from all the previous steps are summed up together with the number of the requirements specification defects detected during the requirements specification phase itself to give us the total number of defects that the developers have injected in the requirements specification artifact.

A process corresponding to the above is followed for the design and code artifacts. The results of these calculations are shown in Table 9.

Table 9: Total number of defects injected in different artifacts

Artifact	Unit	Value
Requirements Specification	Defect	2007=1515+394+45+22+21+10
Design	Defect	2114=1555+304+140+74+38+3
Code	Defect	4453=2421+1525+370+114+23

Having the total number of defects that the developers have injected in different artifacts, the defect injection rates could be calculated by dividing the total number of defects injected in the artifacts by their size. The results of the calculations are shown in Table 10. The sizes of the artifacts were assumed as explained in the calibration of the initial development parameters.

Table 10: Calibration values of defect injection parameters

Parameter Name	Unit	Value
Minimum requ. spec. fault injection rate per size unit	Defect/Page	40.14
Minimum design fault injection rate per size unit	Defect/Page	1.362
Minimum code fault injection rate per size unit	Defect/KLOC	14.52

Verification Effectiveness Parameters: These parameters specify the effectiveness of the artifact verification techniques with regards to defect detection. Effectiveness is expressed as the percentage of the artifact defects that are detected by the artifact verification technique. There exist many sources that can be used for calibration of these parameters in the software engineering literature, e.g., [14] and [15]. However, for the current calibration of GENSIM 2.0, we used the Defect Containment Matrix shown in Table 8 and the total number of defects injected in different artifacts shown in Table 9. In the following, the calibration is presented for the design verification technique.

To calculate the effectiveness of the design verification technique, we have to consider both the total number of defects that are injected in the design artifacts due to developer errors and the number of defects injected due to undetected requirements specification defects. Comparing Table 8 and Table 9, it can be seen that 492 (2007 minus 1515) requirements specifications defects propagate to the design phases. Considering the fault multiplier between requirements specification and design defects, it is concluded that 1477 (492 multiplied by 3) defects are injected in the design artifacts due to undetected requirements specification defects. Adding these 1477 defects to the 2114 design defects that are injected due to developer errors determines that 3591 defects have been injected in the design artifacts. According to Table 8, 2736 of these defects are detected in the design phase. Hence the effectiveness of the design verification technique is calculated as 0.76 (2736 divided 3591). A corresponding process is carried out for the requirements specification and code verification technique and the results are presented in Table 11.

Table 11: Calibration values of verification effectiveness parameters

Parameter Name	Unit	Value
Maximum requ. spec. ver. effectiveness	N/A	0.75
Maximum design ver. effectiveness	N/A	0.76
Maximum code ver. effectiveness	N/A	0.53

Number of Test Cases Parameters: These parameters such as *Average # of UT test cases per code size unit* specify the number of test cases that is required to test the developed artifacts. Since no source for calibrating these parameters was found in the literature, no values have been assigned to these variables for the current calibration of GENSIM 2.0. Refer to [1] for explanation on how the model works when test case data is not available.

5.2 CALIBRATION OF VALIDATION PHASES

This subsection describes how and from which sources calibration parameters used in the simulation modeling of the validation phases of GENSIM 2.0 were calibrated for its current version. To make the understanding of the overall material easier, the parameters are divided into different groups and then the calibration is presented for each group.

Validation Rate Parameters: These parameters specify the speed with which the testers test the artifacts. For calibration of these parameters the values represented in [14] as shown in Table 12 were used.

Table 12: Calibration values of the validation rate parameters

Parameter Name	Unit	Value
Average UT productivity per person per day	KLOC/Person-Day	0.3093
Average IT productivity per person per day	KLOC/Person-Day	0.1856
Average ST productivity per person per day	KLOC/Person-Day	0.1546

Validation Effectiveness Parameters: These parameters specify the effectiveness of the validation techniques with regards to defect detection. Similar to the verification effectiveness parameters, many sources are available in the software engineering literature to calibrate these parameters such as [14]. However, for the current calibration of GENSIM 2.0, similar to the verification effectiveness parameters, these parameters were calibrated using the Defect Containment Matrix shown in Table 8. Hence the effectiveness of each of these techniques was calculated by dividing the total number of code defects detected by the validation technique divided by the number of code defects propagated to the corresponding validation phase. The results of the calculations are shown in Table 13.

Table 13: Calibration values of the validation effectiveness parameters

Parameter Name	Unit	Value
Maximum UT effectiveness	N/A	0.66
Maximum IT effectiveness	N/A	0.69
Maximum ST effectiveness	N/A	0.93

Test case Development and Execution Rate Parameters: These parameters such as *Average # of UT test cases developed per person per day* specify the number of test cases that are developed or executed by one tester in one day. Since no source for calibrating these parameters was found in the literature, no values have been assigned to these variables for the current calibration of GENSIM 2.0. Refer to [1] for explanation on how the model works when test case data is not available.

6 CONCLUSION

This technical report elaborated on how GENSIM 2.0 is calibrated currently. GENSIM 2.0 is a software process simulation model that is intended to be reused, customized and applied to tackle emerging software engineering issues of any kind. One of the essential characteristics of a reusable simulation model is that it can be re-calibrated. That is why, besides showing the current values that are used for calibration of GENSIM 2.0, this report is also intended to be used as a guideline on how GENSIM 2.0 could be re-calibrated.

Future work regarding the calibration of GENSIM 2.0 includes further investigation of the software engineering literature to find more sources that could potentially be used for calibration of GENSIM 2.0 and its re-calibration in order to add to the reliability of its generated estimations.

REFERENCES

[1] K. Khosrovian, D. Pfahl, and V. Garousi, "GENSIM 2.0: A Customizable Process Simulation Model for Software Process Evaluation," Schulich School of Engineering, University of Calgary SERG-2007-07 and Simula TR 2008-01, 2007.

[2] J. M. Lyneis and A. L. Pugh, 1996, "Automated vs. "hand" calibration of system dynamics models: An experiment with a simple project model," in: G. P. Richardson, J. D. Sterman (Eds.), in *Proceedings of the 1996 International System Dynamics Conference*, System Dynamics Society, Cambridge, MA, pp. 317–320.

[3] R. Oliva, "Model calibration as a testing strategy for system dynamics models," in *European Journal of Operational Research,* vol. 151, pp. 552-568, 2003.

[4] J. Munch, D. Rombach, and I. Rus, "Creating an Advanced Software Engineering Laboratory by Combining Empirical Studies with Process Simulation," in *Software Process Simulation Modeling (ProSim03)*, 2003.

[5] I. Rus, S. Biffl, and M. Halling, "Systematically combining process simulation and empirical data in support of decision analysis in software development," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering* Ischia, Italy: ACM, 2002.

[6] I. Rus, F. Shull, and P. Donzelli, "Decision support for using software inspections," in *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*, 2003, pp. 3-11.

[7] D. Pfahl and K. Lebsanft, "Knowledge Acquisition and Process Guidance for Building System Dynamics Simulation Models. An Experience Report from Software Industry," *International Journal of Software Engineering and Knowledge Engineering,* vol. 10, pp. 487-510, 2000.

[8] "Software Engineering Information Repository (SEIR)", https://seir.sei.cmu.edu/seir/seir-home.html, last viewed 5 June 2008

[9] "PROMISE Software Engineering Repository", http://promise.site.uottawa.ca/SERepository/, last viewed 5 June 2008

[10] "Software-artifact Infrastructure Repository", http://sir.unl.edu/portal/index.html, last viewed 5 June 2008

[11] V. R. Basili and R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *Software Engineering, IEEE Transactions on,* vol. SE-13, pp. 1278-1296, 1987.

[12] M. Diaz and J. Sligo, "How software process improvement helped Motorola," *Software, IEEE,* vol. 14, pp. 75-81, 1997.

[13] "Capability Maturity Model for Software (CMM)", http://www.sei.cmu.edu/cmm/, last viewed 05 June 2008

[14] W. Stefan, "A literature survey of the quality economics of defect-detection techniques," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering* Rio de Janeiro, Brazil: ACM, 2006.

[15] O. Laitenberger, "A Survey of Software Inspection Technologies," *Handbook on Software Engineering and Knowledge Engineering,* vol. 2, pp. 517-555, 2002.

[16] "COCOMO II", http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html, last viewed 06 June 2008

[17] L. L. C. W. Lars-Ola Damm, "Faults-slip-through - a concept for measuring the efficiency of the test process," *Software Process: Improvement and Practice,* vol. 11, pp. 47-59, 2006.

[18] A. A. Frost and M. J. Campo, "Advancing Defect Containment to Quantitative Defect Management," *The Journal of Defense Software Engineering,* vol. 20, pp. 24-28, 2007.