BIO-INSPIRED MODELLING OF SOFTWARE VERIFICATION BY MODIFIED MORAN PROCESSES

Sven Söhnlein

Method Park Engineering GmbH, Wetterkreuz 19a, Erlangen, Germany

ABSTRACT

A new approach for the control and prediction of verification activities for large safety-relevant software systems will be presented in this paper. The model is applied on a macroscopic system level and based on so-called Moran processes, which originate from mathematical biology and allow for the description ofphenomena as, for instance, genetic drift. Beside the theoretical foundations of this novel approach, its application on a real-world example from the medical engineering domain will be discussed.

KEYWORDS

Modelling,Simulation, Dependability, Reliability, Software Engineering

1. INTRODUCTION

The development of safety-relevant software systems usually underlies very strict regulations prescribed by corresponding standards, like the IEC 62304 for medical device software [1], for example. In order to provide the necessary control and prediction instruments for the required verification activities of such applications, the use of software reliability models seems to be reasonable. Here, a huge spectrum of different theoreticalapproaches is available in the literature (see [2, 3, 4] for an overview). But the problems in the practical implementation of such models in a real-world software lifecycle process are manifold:

First of all, the usually very strict (and non-verifiable model assumptions [2]) are not flexible enough to map also in cases of continuous integration paradigms [5] or post-development phases, where patches or add-ons are integrated [6]. Moreover, these assumptions are usually not implied from the relevant standards and regulations, but are frequently model-intrinsic [2]. In addition to that, implications that come from typical management necessities in those areas are predominantly ignored [7].

With regard to these determining factors, we propose a practical model that applies on a macroscopic level of large systems and takes into account regulative prescriptions regarding the lifecycle process, software architecture, as well as planning and management demands. The introduced model is inspired by mathematical concepts, that were originally applied to describe biological processes in finite populations.

DOI : 10.5121/ijcsea.2015.5301

1.1. Paper Structure

The paper is organized as follows: In section 2, the relevant regulative and organisational factors are determined, which will be used in the following to derive the theoretical basis for the model. The approach itself will be introduced in section 3. Section 4 illustrates the application of the model on a real-world system from the medical engineering domain, followed by a conclusion in section 5.

2. DETERMINING REGULATIVE AND ORGANISATIONAL FACTORS

In order to derive an adequate context-specific model, one has to analyse the implications that come from the corresponding standards in the particular application domain. In case of medical device software, the IEC 62304 [1] represents the relevant norm (where it should be stated, that similar standards exist for other safety-relevant applications, like the ISO 26262 [8] for the automotive domain, for instance).

In the following, the key-aspects carvedout from the regulative and organisational prescriptions will be highlighted, and referenced in subsequent sections as a basis for the provided model.

Regulative Factors:

R1. Software Lifecycle Process: The development underlies a strict plan-driven software lifecycle process (like the V-Model [9] or the Waterfall-Model [9]). This implies particularly that at least every requirement has to be verified by one (or more) corresponding test casesor by another adequate verification technique [1].

R2. Software Architecture: The subdivision of the software system into interacting components and units must be described and documented. With regard to this modularization, software units represent the smallest atomic parts in the software architecture [1], whereas components in turn consist of a finite number of units [1].

R3. Quality Management System: The IEC 62304 [1] prescribes a quality management system (as defined by the ISO 13485 [10], for instance). Thus, it is required to define quality goals and verify to which extend they are fulfilled.

Organisational Factors:

O1. Verification and Correction Phases: The typical management procedure [11] for the verification process in the considered domain consists of a timely subdivided organization of verification and correction phases, which consist of a certain subset of the overall number of planned test cases.

O2. Impact Analysis: In advance to every correction, an impact analysis [12] is performed in order to reveal the number of units that will be “touched” in the subsequent correction phase.

O3. Statistical Process Control: Statistical process control [13] is performed with the intentto derive measures (considering the progress of verification and correction activities)from past projectswith regard to the current or upcoming one.

Delimitation of Consideration:

Further, the scope of consideration will be delimited as follows:

D1. Classification of Software Units: Software units represent the smallest parts of consideration and will be classified as 'correct' XOR 'faulty' (with no further distinction regarding the involved code parts).

D2. Correction of Faults: Faulty software units which are corrected during a verification and correction phase, change their classification status from 'faulty' to 'correct'.

D3. Insertion of Faults: The correction process is not perfect, i.e. it also has the potential to inject new faults into the system, which is represented by a change of the classification status of a software unit from 'correct' to 'faulty'.

Taking all these aspects into account, the following relation between the relevant elements of the verification and correction process can be established (see figure 1), whererq_i(with$i = 1, \dots, M$) denotes a requirement, tc_i(with$i = 1, \dots, N$) a test case, u_i(with$i = 1, \dots, K$) a software unit and c_i(with$i = 1, \dots, R$) a component:

Each requirement is at least verified by one or more test cases (with regard to assumption R1), where test cases "spot" the 'faulty' (or 'correct') units within certain components of the system (with regard to assumptions R2 and D1). The software units to be "touched" in the subsequent correction phase are revealed by the performed impact analysis (with regard to assumption O2). These software units might thereby change its classification status from 'faulty' to 'correct' (which is the more probable case) but possibly also from 'correct' to 'faulty' (with regard to assumptions D2 and D3).

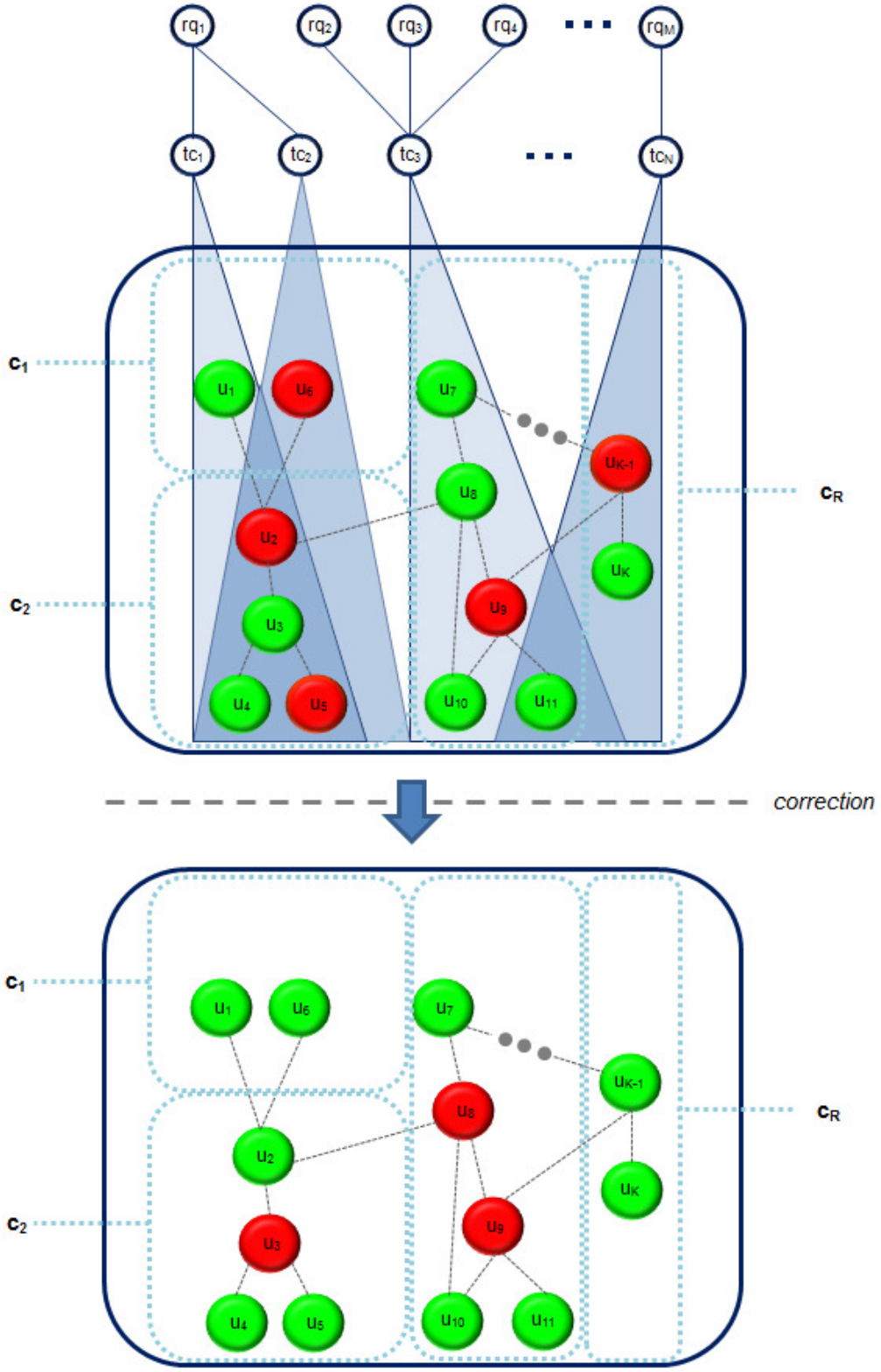

Figure 1. Relation between verification and correction elements

3. MODELLING SOFTWARE VERIFICATION VIA MORAN PROCESSES

Moran processes are stochastic models that originate from mathematical biology and are used to describe, for instance, mutations in finite populations (see [14, 15] for an introduction). In the basic model, a finite population of size $S \in \mathbb{N}$ consists of two alleles (let's say 'green' and 'red'), which are competing for dominance. In each time step, a random individual is chosen for reproduction and another one is chosen for death, thus ensuring a constant population size. The "fitness" of the alleles hereby determines how likely they are to be chosen for reproduction and thereforeaffects the time for fixation (i. e. the time for taking over the whole population).

In order to map this biological model to theconsidered software context, the discussed aspects from section 2 are addressed as follows: The whole software system (which can be interpreted as the DNA [16]) consists of components (DNA Segments) that consist of a finite population of units (genes [16]). Those units (genes) can be classified into two categories (alleles [16]) marked 'correct' (green) XOR 'faulty' (red). A single unit can shift its classification (allel) from 'correct' to 'faulty' or from 'faulty' to 'correct' in one time step (which represents the mutation process [16]). This means that the whole verification and correction process can be considered as the genetic drift [16] in the software system, where the goodness of the process is affected by the fitness of the alleles. Table 2 shows an overview of the corresponding elements from both worlds.

Table 1. Mapping oftechnical and biological elements

Software World	**Biological World**
Software System	DNA
Component	DNA Segment
Unit	Gene
Classification of a unit {'correct', 'faulty'}	Allel {'green', 'red'}
Correction of a fault / Insertion of a fault	Mutation
Verification and correction process	Genetic drift
Goodness of the verification and correction activities	Fitness

In accordance to this mapping, the verification and correction process can be described by an irreducible ergodic discrete-time Markov chain (DTMC [15])

$$(X_t) \text{ with } t \in \mathbb{N}$$

where (X_t) denotes the family of random variables (indexed by the discrete time t). The process underlies a finite state space $S \in \mathbb{N}$, where

$$|S| = K + 1$$

and $K \in \mathbb{N}$ represents the number of software units in the system. Every state $i \in S$ (with $i = 0, \dots, K$) is hereby associated with a software system consisting of i correct (verified) software units (and $K - i$ faulty units).

With regard to assumption O1 (see section 2), it is assumed that in verification and correction phase vcp_q (with $q \in \mathbb{N}$ and $q = 1, \ldots, Q$) where Q represents the overall number of verification and correction phases, we reach a certain state i.Then, the required impact analysis (see assumption O2) will reveal the number of software units, that is "touched" in the next verification and correction phase $vcp_{q+1}$and therefore implies the expected number of time steps for the Moran process in the subsequent phase (see figure 2 for an illustration).

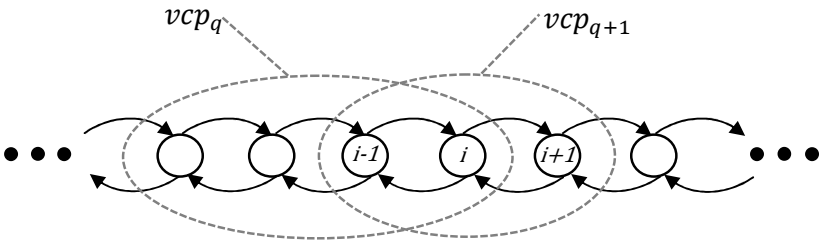

Figure 2. Verification and correction phases (vcp_q) in the Moran process

Therefore, the DTMC for the Moran Process described above can be defined by the $|S| \times |S|$ transition matrix**T**, where the entries of**T** are specified as

$$\mathbf{T}_{i,i+1} = \frac{\phi_q \cdot i}{\phi_q \cdot i + K - i} \cdot \frac{K - i}{K} \quad \text{for} \quad 0 \leq i < K \tag{1}$$

$$\mathbf{T}_{i,i-1} = \frac{K - i}{\phi_q \cdot i + K - i} \cdot \frac{i}{K} \quad \text{for} \quad 0 < i \leq K \tag{2}$$

$$\mathbf{T}_{i,i} = 1 - \mathbf{T}_{i,i-1} - \mathbf{T}_{i,i+1} \quad \text{for} \quad 0 < i < K \tag{3}$$

$$\mathbf{T}_{i,i} = 1 \text{ for } \quad i = 0 \vee i = K \tag{4}$$

and all other entries of **T**are zero, which results in a triangular matrix. Here, ϕ_q represents the mentioned phase-specific "fitness" of the verification and correction activities and can be derived by statistical process control techniques (see assumption O3 in section 2). A coarse approximation of ϕ_q might be estimated by the fraction of successfully corrected components in relation to the inserted faults. Note that in contrast to the original Moran process model [14], the fitness is not fixed here, but changes in accordance to the phase of the whole verification and correction process, which is reasonable with regard to the different preconditions in each phase. In general, ϕ_q can be categorized as follows:

$\phi_q > 1$: This is the usual and expected case, where (significantly) more faults are detected and corrected than injected.

$\phi_q = 1$: In this case, we have a "neutral" drift (and an unsystematic verification and correction process).

$\phi_q < 1$: This is the unusual and unexpected case, where more new faults are injected in the system than detected and corrected.

The defined Moran process is initialized by the start vector $\boldsymbol{\pi}^{(0)}$ with entries

$$\boldsymbol{\pi}_j{}^{(0)} = 1 \text{ for } j = 1 \tag{5}$$

$$\boldsymbol{\pi}_j{}^{(0)} = 0 \text{ for } j \neq 1 \tag{6}$$

which means that at least one correct unit is available at the beginning. Further, we denote by δ_q (with $\delta_q \in \mathbb{N}$ and $\delta_q = 0, \ldots, K$)the number of software units to be touched in a certainvcp_q(see the presumed impact analysis O2). Then, if the first verification and correction phase vcp_1 reveals, that the number of software units to be touched in this phase is δ_1, than the state vector of the Moran process at this stage is computed by

$$\boldsymbol{\pi}^{(1)} = \boldsymbol{\pi}^{(0)} \cdot \mathbf{T}^{(\delta_1)} \tag{7}$$

More generally, if in phase vcp_q, we reach a certain state i (which is associated with a software system of already i verified software units),than the state vector for phase vcp_{q+1} is computed by

$$\boldsymbol{\pi}^{(q+1)} = \boldsymbol{\pi}^{(i)} \cdot \mathbf{T}^{(\delta_{q+1})} \tag{8}$$

with

$$\boldsymbol{\pi}_j{}^{(i)} = 1 \text{ for } j = i \tag{9}$$

$$\boldsymbol{\pi}_j{}^{(i)} = 0 \text{ for } j \neq i \tag{10}$$

Moreover, by

$$\sigma_q = \max_{j=0,\ldots,K} \boldsymbol{\pi}_j{}^{(q)} \tag{11}$$

the most probable state $j_{max}{}^{(q)}$ at vcp_qcan be determined with

$$\boldsymbol{\pi}_{j_{max}}{}^{(q)} = \sigma_q \tag{12}$$

In order to estimate, if predefined reliability targets (see assumption R3) are met (in terms of the minimum number of components that have to be correct after a certain verification and correction phase), we denote by $\theta_{min}(\rho_q)$ the probability, that in phase vcp_q we have at least ρ_q correct components, which can be computed by

$$\theta_{min}(\rho_q) = \sum_{j=\rho_q}^{K} \boldsymbol{\pi}_j{}^{(q)} \tag{13}$$

4. EXAMPLE APPLICATION

In this section, the application of the model on a real-world system (from the medical engineering domain) will be discussed.The model was applied in order to assess the progress of the verification and correction activities, with a specific focus on the reachability of the predefined reliability targets. The system consisted of an overall number of $K = 363$ software units. The verification and correction process was subdivided into $Q = 5$ phases. Table 2 shows the number of software units δ_q that were touched in each phase (estimated by the corresponding impact analysis), the verification fitness $\phi_q$for each phase (estimated by the application of the previously mentioned statistical process control techniques), the predefined reliability targetρ_q for each phase (as an outcome from the project and risk management activities)as well as the computed measures according to equations (1) – (13) from section 4.

Table 2. Computed measures for the example system

q	δ_q	$\frac{\delta_q}{K} \cdot 100\%$	$\Sigma\delta_q$	$\Sigma\frac{\delta_q}{K} \cdot 100\%$	ϕ_q	ρ_q	$\frac{\rho_q}{K} \cdot 100\%$	$j_{max}{}^{(q)}$	$\theta_{min}(\rho_q)$
1	114	31,40	114	31,40	37,96	72,60	20,00	65	0.14
2	95	26,17	209	57,57	18,79	163,35	45,00	168	0.69
3	74	20,39	283	77,96	13,67	235,95	65,00	234	0.40
4	53	14,60	336	92,56	11,32	290,40	80,00	293	0.87
5	27	7,44	363	100,00	9,41	326,70	90,00	337	0.99

If we look at the predefined reliability target ρ_q and the computed most probable state $j_{max}{}^{(q)}$ for each phase, we can see how the predefinition differs from the prediction according to the varying fitness and number of touched components in each phase. While for phases vcp_2, vcp_3 and vcp_4, the most probable states are pretty close to the predefined targets, the discrepancies in phases vcp_1 and vcp_5 are comparatively high. And apart from $vcp_1$and vcp_3, the predefined targets underestimated the most probable states in this case, which is also illustrated in figure 3. But this might be a little bit misleading with regard to the computed probabilities $\theta_{min}(\rho_q)$ of reaching the predefined goals. Here, only vcp_4 and vcp_5 establish a substantial confidence in the reachability of the predefined quality goals, which is also shown in figure 4.

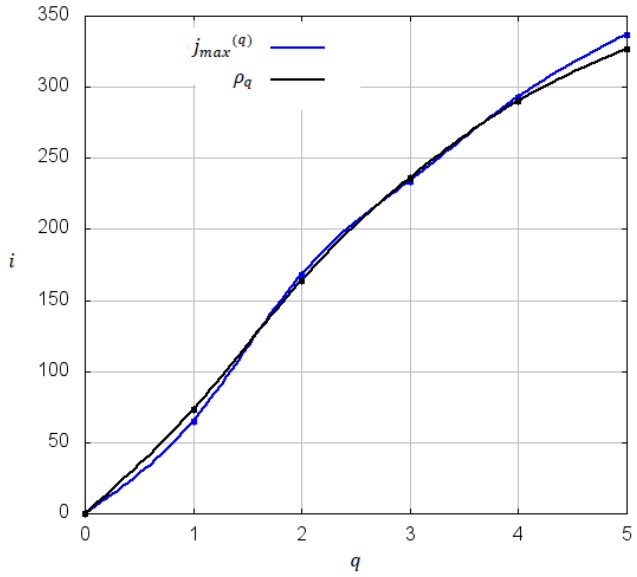

Figure 3. Comparison of predefined and predicted measures

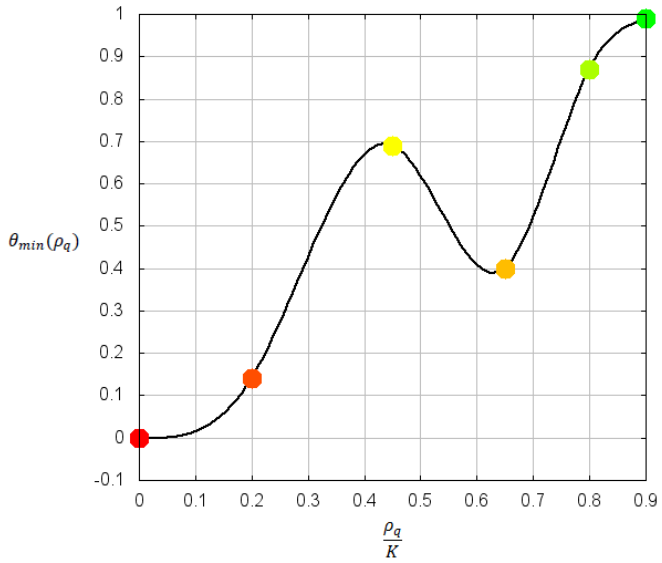

Figure 4. Evolving probability of meeting the predefined targets

The computed measures illustrate how the introduced model can be utilized to adjustpredefined targets for the verification an correction phases.

5. CONCLUSIONS

In this paper, a novel approach for the support of correction processes of large safety-relevant software systems was introduced. Beside the derivation of the theoretical foundations of this model, its application on a real-world example was also shown. Thereby it could be

demonstrated, how this technique can serve as an instrument for the planning and control of the verification activities in such an environment.

REFERENCES

[1] International Electrotechnical Commission: Medical device software - Software life-cycle processes, IEC62304:2006 (2006).
[2] M. R. Lyu (Editor), Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw-Hill, 1996.
[3] J. D. Musa, Software Reliability Engineering, McGraw-Hill, 1999.
[4] D. P. Siewiorek and R. S. Swarz, The Theory and Practice of Reliable System Design,Digital Press, 1982.
[5] P. M. Duvall et al., Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007.
[6] S. S. Yau and J. S. Collofello, “Design Stability Measures for Software Maintenance”, IEEE Transactions on Software Engineering, Vol. 11 (9), pp. 84-856, 1985.
[7] S. R. Rakitin, Software Verification and Validation for Practitioners and Managers, 2nd ed., Artech House, Inc., 2001.
[8] ISO 26262-1:2011(en) Road vehicles - Functional safety, International Standardization Organization (2011).
[9] I. Sommerville, Software Engineering, 9th ed., Pearson, 2012.
[10] ISO 13485:2003 Medical devices - Quality management systems - Requirements for regulatory purposes (2003).
[11] M. Pol et al., Software Testing: A Guide to the TMap Approach, Addison-Wesley Professional, 2001.
[12] K. Fisler et al.,"Verification and change-impact analysis of access-control policies", Proceedings of the 27th international conference on Software engineering, ACM, 2005.
[13] J. S. Oakland, Statistical process control, Routledge, 2008.
[14] P. A. P. Moran, "Random processes in genetics", Mathematical Proceedings of the Cambridge Philosophical Society, Vol. 54. (1), Cambridge University Press, 1958.
[15] M. A. Nowak, Evolutionary dynamics, Harvard University Press, 2006.
[16] K. S. Trivedi, Probability & Statistics with Reliability, Queuing and Computer Science Applications, PHI Learning Pvt. Limited, 2011.

AUTHOR

Dr. Sven Söhnlein received a PhD in Engineering and a MSc in Computer Science from the University of Erlangen-Nürnberg (Germany). Until 2014 he was a Senior Researcher at the University of Erlangen-Nürnberg and is currently working for the company Method Park Engineering GmbH.