Information Sciences Letters

Volume 11
Issue 2 *Mar. 2022*

Article 40

2022

# A Software Evolution Process Model: Analysis of Software Failure Causes

Mohammad Issam Malkawi
*Software Engineering Department, Jordan University of Science and Technology, Irbid, Jordan*, mimalkawi@just.edu.jo

Ehab Mohammad Abidah
*Information Technology Department, Shamal International school, Dubai, UAE.*, mimalkawi@just.edu.jo

Ahmed S. Shatnawi
*Software Engineering Department, Jordan University of Science and Technology, Irbid, Jordan*, mimalkawi@just.edu.jo

Information Sciences Letters
*An International Journal*

http://dx.doi.org/10.18576/isl/110209

# A Software Evolution Process Model: Analysis of Software Failure Causes

*Mohammad Issam Malkawi*[1,*], *Ehab Mohammad Abidah*[2] *and Ahmed S. Shatnawi*[1]

[1]Software Engineering Department, Jordan University of Science and Technology, Irbid, Jordan
[2]Information Technology Department, Shamal International school, Dubai, UAE.

Received: 21 Dec. 2021, Revised: 22 Jan. 2022, Accepted: 3 Feb. 2022.
Published online:1 Mar. 2022.

**Abstract:** This paper presents a study on the degree of impact of several components on the evolvability of software systems. In particular, it focuses on failure rates, testing, and other factors which force the evolution of a software system. Also, it studies the evolution of software systems in the presence of various failure scenarios. Unlike previous studies based on the system dynamic (SD) model, this study is modeled on the basis of actor-network theory (ANT) of software evolution, using the system dynamic environment. The main index used in this study is the destabilization period after the recovery from any failure scenario. The results show that more testing and quick recovery after failure are keys to a fast system return to stability.
**Keywords:** software evolution process, system dynamic (SD), actor-network theory (ANT), agent-based simulation environment 'Repast', ANT model, stability.

## 1 Introduction

Contrary to software aging, software evolution addresses the ability of software to evolve in a manner to sustain its effectiveness and improve its overall cost benefits characteristics [1]. SW evolution highlights the sequence of changes that happen to a software system during its lifetime, involving both system development and maintenance [2]. The software evolution process is a very important issue in software-based systems. This topic has received high attention in the last decade. In particular the growth of using such systems in human life, e.g., healthcare, emergency, and safety has made it an important topic for researchers in software engineering and the research community in general. However, researchers have attempted to understand the reasons behind the process of software evolution in order to manage and control the factors that influence this process [3]. Software evolution takes several shapes, one of the commonly known as the development of new versions of the software. A new version is a natural evolution of the previous version. It has been observed [4] that major software systems such as operating systems experience longer stabilization periods in subsequent versions. A stabilization period is defined as the time required for fault rate drops below a certain level. Typically, a newly developed version is expected to have relatively large fault rates due to software faults, errors, and failures [4]. It is also expected that new releases of the software to take less time for the fault rates to drop to a reasonable level, where the system can be considered stable. The authors in [4] show that in real large software systems, where the stability period in subsequent releases is larger than the previous ones. In this paper, we will study the impact of several factors on the stability of a system so that developers and project managers can improve the evolution experience, in a manner where subsequent releases continue to have a better stability period [5]. In essence, it is aimed at making the evolution of software systems a method for improving the software systems over time, in defiance with the software aging phenomenon.

Liguo Yu1 and Alok Mishra described the base of the global software process as a set of humans and events that control the evolution of software-based systems. They presented the process as being driven by feedback, which demonstrated the 8th law in software evolution [6]. They note that "E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems" [7]. Based on the previous researchers, [8] [9]

developed many simulation models in software evolution, they aimed to understand and explain the factors that influence the software evolution process.

[*]Corresponding author e-mail: mimalkawi@just.edu.jo

## 2 Actor Network Model (ANT)

Wernick (2008) suggested applying ANT theory to the global software process in order to understand the reasons behind software system evolution and to observe software system behavior such as software system size growth over time.
The model described in Table 1 is structured as 16 entities, including 13 actors and 3 mediators.

**Table 1:** ANT entities based on Latour perspective.

| Model participants | Description | Role | ANT Model |
|---|---|---|---|
| **Actors** | Mostly people and they can be technological elements | Act but constrained to make choices by their situations. | 13 Actors |
| **Mediators** | Law, science, religion and economies | Receive and transmit messages | Mutable tools. Immutable tools. System Change IP Queue |
| **Intermediaries** | | Receive Messages without changing the message content | none |

## 3 Related Work

Wernick and a team of experts in software engineering [10] have developed many System Dynamics (SD) simulation models in the field of software evolution processes including models based on Actor-Network Theory (ANT) [11], which aimed to characterize the global software processes through SD environment.

in a realistic form that considers actual software evolution environments.
The model participants‘ actors and mediators were given as equations to provide the ability to quantify each participant changing support degree to the evolution process. The equation shown below is for the participants in the model, where *Ht* denotes the Health of the system evolution process, *Ho* denotes the health own weight; *D* denotes the Developers, *Imt* denotes the Immutable tools, *Mt* denotes the Mutable tools, *Pm* denotes the Project manager, *Sc* denotes the System change input queue, *Sd* denotes the System design, *A* denotes the Architecture, *So* denotes the System development owners, and *Ho* denotes the Health own weight.

The equation for the participants in the model shown in the Health of software evolution process (HSE) equation as follows:

$$HSE = \frac{(Ht \times Ho) + D + Imt + Mt + Pm + Sc + (Sd/A) + So}{7 * (1 - Ho)}$$

Wernick and his team [10] utilized a typical, abstract, global large-scale software process evolving a bespoke commercial software system to build ANT based model.

However, the challenge with this model is that it needs modifications to reflect other operational environments settings or process differences in a specific environment. For example, it may need modifications to incubate open-source software evolution processes or package software products evolution. The model structure and its participants and their connections are built based on Lehman and co-workers [7,9, 12] . This model is structured as 16 entities, including 13 actors and 3 mediators, as shown in Figure 1.

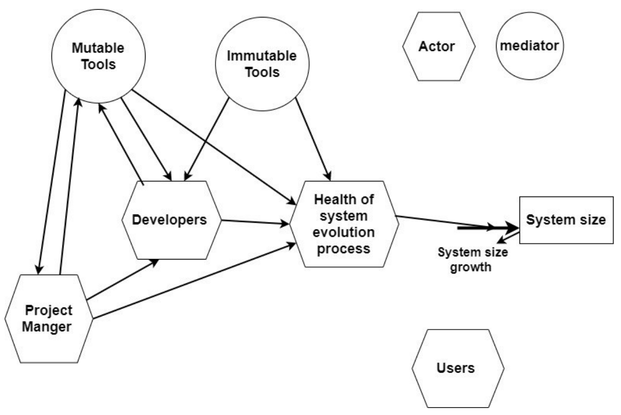

**Fig. 1:** General purpose ANT based model structure.

This model contains participants which are arranged in a hierarchy (a typical, abstract, global large-scale software process evolving a bespoke commercial software system). However, each one of these participants has behaviors and interactions with the other participants in the model. The interactions between these participants create social and technical situations over time which is reflected in system health. This study aims to refine and modify the structure

This equation shows how the actor re-computes its value at each time step based on the average of the values of those factors which influence it, weighted against its own value from the immediate past. Participants’ own health weighting represents the impact of participants on system health.
The output from the model equation is a value, which represents the expected evolutionary trend of this participant over time. All participants in the model are given a value of 1, which represents the participant’s behavior whereas no positive or negative impact on the system evolution

process. Any changes to participant's value above or below 1 are reflected on system health evolution process value because of changing support degree of actors on the network. This means that a value
¿1 represents the actor's positive attitude toward the system health and its evolution process and growing
trend, and a value of ¡1 shows a negative attitude toward the system and the process of software evolution. The output most commonly observed in software evolution processes, and therefore the most easily calibrated against and related to real-world software evolution, is the change in current physical system size over time.
In order to calibrate model inputs and parameters to numerical values, for each participant, 'nominal' (default) behavior is represented by a value of 1 as is the case for the 'Health of the system evolution processes'. This value represents the behavior of each participant in the SD simulation model that has no positive or negative effect on the system evolution process. Moreover, the inputs of the computation of each participant were given an equal weighting percentage, 50% (a value of 0.5 for all the participants), as a deliberate simplification to enable the model outputs to be computed in advance of actual values being available. The own health weighting for each participant in the simulation model (Repast) [13] refers to the percentage of these participants' effect on the health of system evolution.

## 4 Modifying and Based Model Structure

This model is considered an atypical, abstract, global large-scale software process evolving a bespoke commercial software system with clustering methodology [14,15]. This model would require changes to reflect the differences in processes for other environments such as the evolution of package software products or for open-source software evolution processes. Therefore, additional improvements to the structure of the current ANT model are added in this study. This is an essential step towards evolving the model into a realistic representation of actual software evolution environments which include new two agents (Testing and Failure Rate) to an actor-network formed of participants in the evolution of an abstracted large-scale long-term commercial software evolution process as represented in Figure 2.

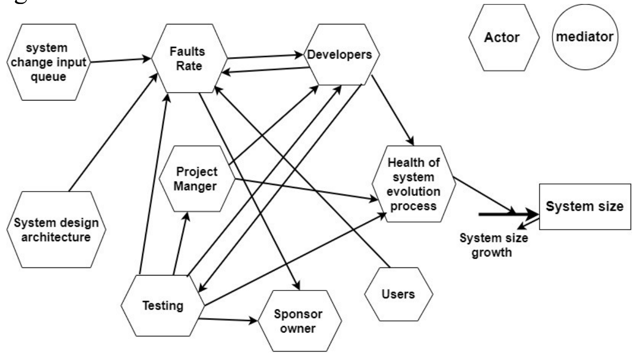

**Fig. 2:** ANT model with new participants (Faults rate, Testing).

## 5 Implementation

The implementation of the new agent-based model [16, 17, 18, 19, 20] of system evolution process is considered depending on the specifications of the existing SD model and the available researches and descriptions for it by Wernick and his team [10]. The model is structured as 16 participants, including 13 actors and 3 mediators with connections links between these participants. The new model implemented by using Repast Symphony [13] simulation by using Relogo with the new participants (testing , failure rate) [21]. So it consisted of 18 participants.

## 6 Experiments, Results and Analysis

### 6.1 First Experiment

The first experiment is conducted by setting model participants' default value to 1 and equal weighting value (0.5 to all the participants). This test has no positive or negative impact on the system evolution process. However, the result of this test refers to a stable behavior of the system health evolution process

### 6.2 Second Experiment

The second experiment took into account the impact of the reduction in Failure Rate value and associated support level to the following values (0.1 to 1.0) in tick time 50 while keeping all participants' own health weighting to 0.5. This experiment aimed to cover all project failure possibilities that may happen in order to observe system health and its behavior in the evolution process when it is impacted by negative support from one participant (failure rate in this case). The results are shown in Table 2.

**Table 2:** Failure rate attitude to (0.1 – 1.0)

| Failure Rate | Minimum Health | | Period |
|---|---|---|---|
| 0.1 | 0.998 | 53 | 145 |
| 0.2 | 0.987 | 53 | 150 |
| 0.3 | 0.986 | 53 | 155 |
| 0.4 | 0.985 | 53 | 160 |
| 0.5 | 0.984 | 53 | 165 |
| 0.6 | 0.983 | 53 | 170 |
| 0.7 | 0.982 | 53 | 175 |
| 0.8 | 0.981 | 53 | 180 |
| 0.9 | 0.980 | 53 | 185 |
| 1.0 | 0.979 | 53 | 190 |

These results show that the health of the system evolution process is reduced gradually by the negative impact from failure rate beginning from 0.1 to 1.0 as shown in table 2. It also shows that the time to return to stability increases as the recovery from failure rate is reduced.

### 6.3 Third Experiment

The third experiment is conducted by fixing the testing support degree value from 0.1 to 1 (10% - 100%), and fix own weighting value to three possible values which are:

0.1, 0.5, and 0.9. This experiment demonstrates the health of system evolution process negative impact from testing and then check model behavior. The results in Table 3 shows that the health of the system evolution process is reduced gradually by the negative impact from testing beginning from 0.1 to 1.0. It also shows that testing own weighting affects the health of the system whereas the higher degree of own weighting creates a higher effect on the health of the system.

**Table 3:** Testing attitude to (0.1 – 1.0)

| Testing Support Degree | Testing Own Weighting | Minimum Health Value | In tick Time | Period Needed to return to stability |
|---|---|---|---|---|
| 3*0.1 | 0.1 | 0.969 | 51 | 3*200 + |
| | 0.5 | 0.937 | 51 | |
| | 1 | 0646 | 56 | |
| 3*0.2 | 0.1 | 0.976 | 52 | 3*200 + |
| | 0.5 | 0.919 | 52 | |
| | 1 | 0.464 | 275+ | |
| 3*0.3 | 0.1 | | | 3*160 |
| | 0.5 | 0.914 | 52 | |
| | 1 | | | |
| 3*0.4 | 0.1 | | | 3*160 |
| | 0.5 | 0.908 | 52 | |
| | 1 | | | |
| 3*0.5 | 0.1 | | | 3*160 |
| | 0.5 | 0.903 | 52 | |
| | 1 | | | |
| 3*0.6 | 0.1 | | | 3*160 |
| | 0.5 | 0.897 | 52 | |
| | 1 | | | |
| 3*0.7 | 0.1 | | | 3*200 + |
| | 0.5 | 0.892 | 52 | |
| | 1 | | | |
| 3*0.8 | 0.1 | | | 3*200 + |
| | 0.5 | 0.886 | 52 | |
| | 1 | | | |
| 3*0.9 | 0.1 | | | 3*200 + |
| | 0.5 | 0.880 | 52 | |
| | 1 | | | |
| 3*1.0 | 0.1 | | | 3*200 + |
| | 0.5 | 0.875 | 52 | |
| | 1 | | | |

## 6.4 Forth Experiment

The fourth experiment is conducted by considering the median value from failure rate (support degree) which is 0.5 in order to give the health of the system evolution process a positive impact from testing. Default testing (support degree) value is 1. Therefore, any percentage above 1 can be considered a positive support degree and can be measured as the following: 1.1 = 10% , 1.2=20%, and so on as shown in Table 4.

**Table 4:** Testing positive attitude to (0.1 – 1.0).

| Testing support degree | Highest Health Value | In tick time | Minimum Health Value | In tick time | Period needed to return to stability |
|---|---|---|---|---|---|
| 1.1 | 1.004 | 51 | 0.988 | 54 | 160 |
| 1.2 | 1.004 | 51 | 0.992 | 54,55 | 160 |
| 1.3 | 1.009 | 51 | 0.996 | 55 56 | 160 |
| 1.4 | 1.015 | 51 | 0.999 | 56 to 70 | 160 |
| 1.5 | 1.020 | 51 | 1 | 72 | 160 + |
| 1.6 | 1.026 | 51 | 1 | 142 | 160 + |
| 1.7 | 1.032 | 51 | 1 | 180 | 160 + |
| 1.8 | 1.038 | 51 | 1 | 180 | 160 + |
| 1.9 | 1.043 | 51 | 1 | 180 | 200 + |
| 2.0 | 1.049 | 51 | 1 | 180 | 200 + |

## 6.5 Fifth Experiments

Due to the lack of available real-world data, experimental and hypothetical data are used to investigate whether the model is able to reflect real-world software evolution process or not based on software system professional viewpoints [22]. Table 4 below shows the results after running the repast model for 100 time ticks when the Sponsor's attitude is reduced by 40% for one-time tick in tick 45, and reset the own health weighting for the participant (sponsor owner) beginning from 0.1 and ending with 0.99 instead of the proposed arbitrary value 0.5 (default percentage of 50%) in order to check and measure the behavior of the simulation model by measuring the health of the system evolution when it is affected by varying degrees of effect by sponsor owner on the health of the system evolution as shown in Table 5.

**Table 5:** Sponsor owner impact on the health of the system evolution.

[H]

| Own health weighting percentage | Minimum health of system | In tick time | Period needed to return |
|---|---|---|---|
| 0.1 | 0.9950 | 47 | 63 |
| 0.2 | 0.9887 | 47 | 87 |
| 0.3 | 0.9815 | 47 | 211 |
| 0.4 | 0.9735 | 47 | 230 |
| 0.5 | 0.9635 | 48 | 280 |
| 0.6 | 0.9570 | 48 | 296 |
| 0.7 | 0.9338 | 49 | 320 |
| 0.8 | 0.9080 | 50 | 325 |
| 0.9 | 0.8555 | 53/54 | 345 |
| 0.99 | 0.6280 | 76/77 | 1173 |
| 1 | 0.47345 | 145 | Get stable to infinity but in |

The result shows that the health of the evolution process continues to decline until tick 48 when the health of evolution is equal to 0.963. According to the result, after tick 48 the evolution health starts to increase again to become 0.965 at tick 49 and 0.967 at tick 50. System evolution health continues to increase to 0.969 at tick 51 and to 0.970 at tick 52. However, software evolution health does not return to its stable health at the value of 1, even at tick 100. To indicate the tick step in which the health of the software evolution process returns to its previous stability at a value of 1 before applying the pulse, the model is re-run for 200 ticks. The numerical result shows that the software evolution process will not return to its stable health '0.999 1'until tick 152 as shown in Table 5.

many tests are conducted to check and measure the behavior of the simulation model by measuring the health of the system evolution when it is affected by a change in support of each sponsor owner. These tests are conducted by reducing the degree of support of the sponsor owner by an arbitrary 40%, while the other participants in the model retain their initial degree of support at a value of 1. In the real world, such temporary reductions in an individual's support could be due to causes such as financial or political pressures [10]. Furthermore, the results of these ten tests measure the degree of the decrease in the health of system evolution. This represents the probability of the software project to fail eventually. Therefore, building these criteria are conducted based on the effect proportion of the negative attitude of Sponsors owner to cause a failure of a software project in the real world. As the results show in Table 5, the criteria for investigating the behavior of the Repast model show that the health of the software evolution process is affected most by the negative support of the Sponsor owner project management team when the own health weighting is equal to 0.99 and the minimum health of system evolution was 0.6280 at tick time 76/77 , and the lowest degree of effect on the health of system evolution when it was 0.1 with minimum health of system evolution of 0.9950 at tick time 47. However, a stranger behavior has noticed in the repast model when the won health

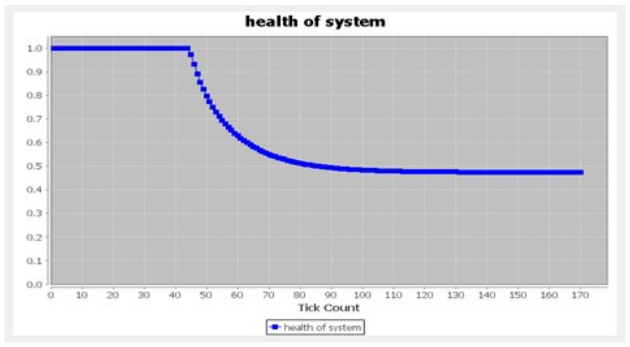

**Fig. 3:** Health of system evolution process with negative support when the won health weighting reset to 1).

weighting reset to 1 as shown in Figure 3, the minimum health of system evolution has decreased to 0.47345 in tick time 145 and health of system evolution remained stable on this value for infinity and it does not return to its normal health of value of 1 where (no positive or negative effect on the expected evolutionary trend). This kind of behavior could cause a failure of a software project in the real world. Furthermore, other participants in the repast model are affected badly by resetting the percentage of the own health weighting of sponsor owner to 1, whereas, no participant of them has returned to its normal behavior, on the contrary, they remained stable on several values of less than 1 for infinity, and some decreased to values less than zero. The ability to expect or to predict when a software system could fail is available through tracing one agent's behavior in a particular situation in the real world.

## 7 Discussions and Conclusions

Although project management methodologies and software have improved, project failures remain high. In real-world SW project testing play an essential role to protect a project from failure [23]. According to several published studies on project failure, various types of failure were collected and categorized. In [18], 26,595 served participants confirmed that 84% of projects fail due to incorrect assumptions in the schedule and budget or resource issues. Figure 4 shows a breakdown of the impact of various factors on project' failures.

In industries in the real world, factors of both project and sponsor are explicit that the most dominant factors in project failure are the Project Manager and the Sponsor [24, 25]. By calibrating the results of the ten tests conducted in this stud against real-world factors on project failures in industry, it was concluded that these results are compatible with the real-world criteria of software evolution. This compatibility shows that the Repast simulation model of software evolution is able to reflect real-world software evolution if an accurate own weighting for the participants 'actors and mediators are

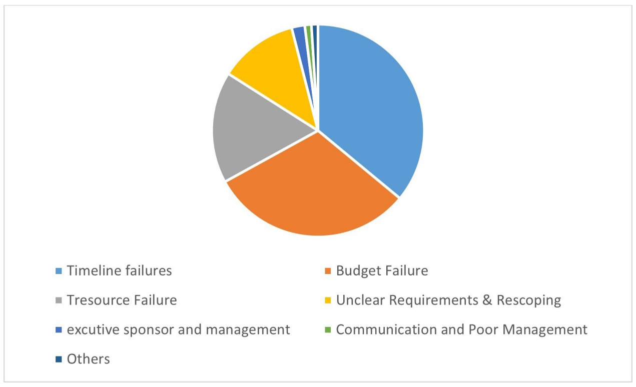

**Fig. 4:** Distribution of software system project failure causes.

set. This behavior of the Repast model supports and can be considered as an advanced work that Werneck and colleagues [7, 8, 10, 26, 27, 28] were intending to undertake. Another important conclusion of this study is the ability of the simulation model to test and measure the stabilization period of a system given a certain failure rate. Further studies are required to study the impact of certain common failure modes on the overall system stability.

## Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

## References

[1] Mohammad Isam Malkawi. The art of software systems development: Reliability, availability, maintainability, performance (ramp). Human-Centric Computing and Information Sciences., **3(1)**,1–17, 2013.

[2] Stephen Cook, He Ji, and Rachel Harrison. Software evolution and software evolvability. University of Reading, UK., 1–12, 2000.

[3] Bruno Latour. On actor-network theory: A few clarifications. Soziale welt., 369–381, 1996.

[4] Mohammad Malkawi. Empirical data and analysis of defects in operating systems kernels. In Proceedings of the 24th IBIMA conference. Milan, Italy., 6–7, 2014.

[5] E Death March et al. The complete software developer's guide to surviving "mission impossible" project [m]. 1999.

[6] Liguo Yu and Alok Mishra. An empirical study of lehman's law on software quality evolution. 2013.

[7] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. Metrics and laws of software evolution-the nineties view. In Proceedings Fourth International Software Metrics Symposium., 20–32. IEEE, 1997.

[8] Paul Wernick and Meir M Lehman. Software process white box modelling for feast/1. Journal of Systems and Software., **46(2-3)**,193–201, 1999.

[9] Goel Kahen, Meir M Lehman, Juan F Ramil, and Paul Wernick. System dynamics modelling of software evolution processes for policy investigation: Approach and example. Journal of Systems and Software., **59(3)**, 271–281, 2001.

[10] Paul Wernick, Tracy Hall, and Chrystopher L Nehaniv. Software evolutionary dynamics modelled as the activity of an actor-network. IET software., **2(4)**, 321–336, 2008.

[11] Bruno Latour et al. Reassembling the social: An introduction to actor-network-theory. Oxford university press, 2005.

[12] G Kahen, MM Lehman, and JF Ramil. Empirical studies of the global software process-the impact of feedback. In Proc. Workshop on Empirical Studies of Software Maintenance (WESS'99), 3–4. Citeseer, 1999.

[13] Nick Collier. Repast: An extensible framework for agent simulation. The University of Chicago's Social Science Research, 36:2003, 2003.

[14] Mark Shtern and Vassilios Tzerpos. Clustering methodologies for software engineering. Advances in Software Engineering, 2012, 2012.

[15] Chung-Horng Lung, Marzia Zaman, and Amit Nandi. Applications of clustering techniques to software partitioning, recovery and restructuring. Journal of Systems and Software., **73(2)**, 227–244, 2004.

[16] Nigel Gilbert. Agent-based models, volume 153. Sage Publications, 2019.

[17] Steven F Railsback, Steven L Lytinen, and Stephen K Jackson. Agent-based simulation platforms: Review and development recommendations. Simulation., **82(9)**, 609–623, 2006.

[18] Charles M Macal and Michael J North. Agent-based modeling and simulation. In Proceedings of the 2009 Winter Simulation Conference (WSC)., 86–98. IEEE, 2009.

[19] Robert John Allan et al. Survey of agent based modelling and simulation tools. Science & Technology Facilities Council New York, 2010.

[20] Dirk Helbing. Agent-based modeling. In Social self-organization., 25–70. Springer, 2012.

[21] Nuno Fachada, Vitor V Lopes, Rui C Martins, and Agostinho C Rosa. Towards a standard model for research in agent-based modeling and simulation. PeerJ Computer Science, 1:e36, 2015.

[22] Mohammed N Alenezi, Haneen Alabdulrazzaq, Abdullah A Alshaher, and Mubarak M Alkharang. Evolution of malware threats and techniques: A review. International Journal of Communication Networks and Information Security., **12(3)**, 326–337, 2020.

[23] May NIST. The economic impacts of inadequate infrastructure for software testing. Technical report, Technical Report., 2002.

[24] Robert N Charette. Why software fails [software failure].

IEEE spectrum., **42(9)**, 42–49, 2005.

[25] CHAOS Manifesto. Think big, act small. The Standish Group International Inc., 176, 2013.

[26] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. Metrics and laws of software evolution-the nineties view. In Proceedings Fourth International Software Metrics Symposium., 20–32. IEEE, 1997.

[27] Nabeel Tawalbeh, Mohammad Malkawi, Hanan Abusamaha, Sahban Alnaser, Demand Based Cost Optimization of Electric Bills for Household Users, International Journal of Communication Networks and Information Security 13(3):276-281, 2021

[28] Mohammad Malkawi, A. Shatnawi, K. Al-Zoubi L. Alawneh, Improving Network Entry Procedure in Broadband Wi-Fi Networks, International Journal on Communications Antenna and Propagation (IRECAP) 11(5):354, 2021, DOI: 10.15866/irecap.v11i5.20996