A New Approach to the Quantitative Measurement of Software Reliability

Abbas Rasoolzadegan*
Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
rasoolzadegan@um.ac.ir

Received: 02/May/2015 Revised: 07/Jun/2015 Accepted: 04/Jul/2015

Abstract

Nowadays software systems have very important role in a lot of sensitive and critical applications. Sometimes a small error in software could cause financial or even health loss in critical applications. So reliability assurance as a nun-functional requirement, is very vital. One of the key tasks to ensure error-free operation of the software, is to have a quantitative measurement of the software reliability. Software reliability engineering is defined as the quantitative study of the operational behavior of software systems with respect to user requirements concerning reliability. Software Reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Quantifying software reliability is increasingly becoming necessary.

We have recently proposed a new approach (referred to as $SDA_{Flex\&Rel}$) to the development of «reliable yet flexible» software. In this paper, we first present the definitions of a set of key terms that are necessary to communicate with the scope and contributions of this work. Based on the fact that software reliability is directly proportional to the reliability of the development approach used, in this paper, a new approach is proposed to quantitatively measure the reliability of the software developed using $SDA_{Flex\&Rel}$, thereby making precise informal claims on the reliability improvement. The quantitative results confirm the reliability improvement that is informally promised by $SDA_{Flex\&Rel}$.

Keywords: Reliability; Quantitative Measurement; Reliability Assessment; Fault Prevention; Formal Methods.

1. Introduction

The demand for complex software-hardware systems has increased more rapidly than the ability to develop them with highly desired quality [2], [6], [9], [12]. When the requirements for and dependencies on such systems increase, the possibility of crises from software failures also increases. The impact of these failures ranges from inconvenience (e.g., malfunctions of home appliances) to economic damage (e.g., interruptions of banking systems) to loss of life (e.g., failures of flight systems or medical software).

Software reliability engineering (SRE) is defined as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability. SRE is centered around a very important facet of dependability, i.e., reliability. Software Reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1]. Software reliability has to be a probabilistic measure because the failure process, i.e. the way faults become active and cause failures, depends on the input sequence and operation conditions, and those cannot be predicted with absolute certainty [37-39]. Human behavior introduces uncertainty and hence probability into software reliability, although software usually fails in the same way for same operational conditions and same parameters. An additional reason to claim a probabilistic measure is that it is usually only possible to approximate the number of faults of complex software system.

Many concepts of software reliability can be adapted from the older and successful techniques of hardware reliability [40-41]. However, this must be done with care, since there are some fundamental differences in the nature of hardware and software, and their failure processes. The largest part of hardware failures is considered as result from physical deterioration. Sooner or later, these natural faults will introduce faults into hardware components and hence lead to failures. Experience has shown, that these physical effects are well-described by exponential equations in the relation to time. Usage commonly accelerates the reliability decrease, but even unused hardware deteriorates. Software does not wear outor deteriorate, i.e., its reliability does not decrease with time. Moreover, software generally enjoys reliability growth during testing and operation since software faults can be detected and removed when software failures occur. On the other hand, Software may experience reliability decrease due to abrupt changes of its operational usage or incorrect modifications to the software. Software is also continuously modified throughout its life cycle. The malleability of software makes it inevitable for us to consider variable failure rates.

Design faults are a different source for failures. They result mainly from human error in the development process or maintenance. Design faults will cause a failure under certain circumstances. The probability of the

* Corresponding Author

activation of a design fault is typically only usage dependent and time independent. Unlike hardware faults which are mostly physical faults, software failures are caused by design faults, which are harder to visualize, detect, and correct. In the context of software reliability, the term *design* refers to all software development steps from the requirements to implementation [2-3].

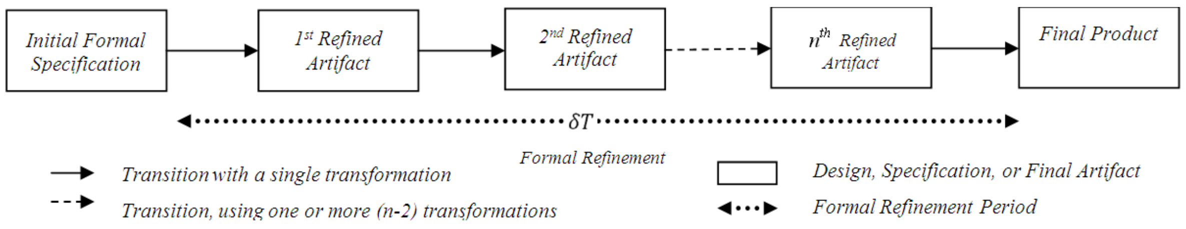

Initial Formal Specification
1st Refined Artifact
2nd Refined Artifact
nth Refined Artifact
Final Product
δT
Formal Refinement
Transition with a single transformation
Transition, using one or more (n-2) transformations
Design, Specification, or Final Artifact
Formal Refinement Period

Fig. 1. Formal Software Development Process

In contrast to hardware, software can be perfect (i.e. fault-free). Formal modeling methods (FMMs) are broadly defined as notations with accurate and unambiguous semantics. They are supported by various tools. FMMs mathematically prove the consistency and completeness of activities during software development. Such proofs help detect all faults before they turn into failures. In addition, the correctness insured by proof is more comprehensive and reliable than the correctness guaranteed by test. These advantages facilitate the development of correct and reliable software [3-5].

Fig. 1 illustrates the formal software development process using FMMs. This process starts with an *initial formal specification*, which abstractly states the stakeholders' requirements. Then, the details of design are added to the initial specification through a gradual process (δT), using *formal refinement*. This process contains several intermediate *artifacts* refined by *transformations* and continues until producing the *final product* [6].

FMMs, along with formal refinement and formal verification techniques prove the correctness of software throughout the formal software development process. As a result, the absence of faults is guaranteed. However, lack of knowledge and high cost restrict their use to the development of critical and high integrity software. Critical systems such as spacecraft, aircraft, nuclear power plant and pacemakers require a high level of reliability in their operation. Software failures can lead to fatal consequences in safety-critical systems [4], thereby making it more important than ever to ensure the reliability of such systems. The term 'safety-critical' refers to those software systems whose failure may lead to loss of life or severe injury. In other words, safety-critical systems include software whose failure can lead to a hazardous state.

We have recently proposed a Software Development Approach (SDA). This approach, referred to as $\text{SDA}_{\text{Flex\&Rel}}$ in this paper, promises to develop reliable yet flexible software [7]. In this approach, Object-Z, as a dominant formal specification language, is used to formally specify and refine requirements – which, in turn, prevent and remove probable faults. Formal modeling and refinement in Object-Z ensure the reliability of software.

So far, many models have been proposed for quantification of the software reliability. Each of these models has its advantages and limitations [11-41]. In [43] we classify different approaches of software reliability modeling and finally, based on the analysis of the advantages and limitations, compare different approaches and mention some challenges and issues.

In this paper, we quantitatively measure the reliability improvement promised by $\text{SDA}_{\text{Flex\&Rel}}$. Indeed, the contribution of this paper is to measure the reliability of the software developed using $\text{SDA}_{\text{Flex\&Rel}}$ by measuring the reliability of $\text{SDA}_{\text{Flex\&Rel}}$ because there is a direct relation between the reliability of software and the reliability of the corresponding development approach. The idea behind this work has been inspired by an existing technique for reliability assessment, i.e., software metric based reliability analysis, as well as a typical type of reliability measurement, i.e., prediction when failure data are not available.

The rest of this paper is organized as follows: Section two presents the definitions of a set of key terms that are necessary to communicate with the scope and contributions of this work. These terms are *dependability*, *failure*, *fault*, and *error*. A brief description of the main approaches to the achievement of reliability, the major classes of reliability assessment, and the main activities of reliability measurement are also presented in section two. The reliability of the software development approach $\text{SDA}_{\text{Flex\&Rel}}$ is quantitatively measured in section three. Finally, section four discusses the conclusions.

2. Background

2.1 Dependability

Dependability is defined as the trustworthiness of a software-hardware system such that reliance can justifiably be placed on the service it delivers [1-3], [8-9]. The service delivered by a system is its behavior as it is perceptible by its user(s); a user is another system (human or physical) interacting with the former. Depending on the application(s) intended for the system, a different emphasis may be put on the various facets of dependability, that is, dependability may be viewed according to different, but complementary, properties, which enable the attributes of dependability to be defined:

- The readiness for usage leads to *availability*.
- The continuity of service leads to *reliability*.
- The nonoccurrence of catastrophic consequences on the environment leads to *safety*.
- The nonoccurrence of the unauthorized disclosure of information leads to *confidentiality*.
- The nonoccurrence of improper alterations of information leads to *integrity*.
- The ability to undergo repairs and evolutions leads to *maintainability*.

2.2 Failure

A failure occurs when the user perceives that the system ceases to deliver the expected service [1]. The user may choose to identify several severity levels of failures, such as: catastrophic, major, and minor, depending on their impacts to the system service. The definitions of these severity levels vary from system to system [3].

Failure behavior directly depends on the environment and the number of faults present in the software during execution. Let T denotes a random variable representing the system failure time. Failure density $f(T)$ corresponds to the probability distribution function of T. Failure probability $F(t)$ is the probability that the failure time is less or equal to time t [2], [10]:

$$F(t) = Prob(T \leq t) = \int_0^t f(u).du \qquad (1)$$

Reliability $R(t)$ is the probability that the system delivers the expected services in the time interval:

$$R(t) = 1 - F(t) = Prob(T \geq t) = \int_t^\infty f(u).du \qquad (2)$$

With respect to the type of hardware faults, hardware reliability metrics are usually time dependent. Although the failure behavior of software (design) faults depends on usage and not directly on time, software reliability is usually expressed in relation to time, as well. However, it is possible to define software reliability with respect to other bases such as software runs. A major advantage of time dependent software reliability metrics is that they can be combined with hardware reliability metrics to estimate the system reliability. Only as intermediate results, some reliability models use time-independent metrics.

2.3 Fault

A fault is uncovered when either a failure of the software occurs or an internal error (e.g., an incorrect state) is detected within the software. The cause of the failure or the internal error is said to be a fault. It is also referred as a *bug*. Software faults arise mostly from design issues. The source of software faults include:

- Incorrect requirements, even though the implementation may match them.
- Implementation (software design and coding) deviating from (correct) requirements.
- Uncontrolled or unexpected changes in operational usage or incorrect modifications.

In summary, a software failure is an incorrect result with respect to the specification or an unexpected software behavior perceived by the user at the boundary of the software system, while a software fault is the identified or hypothesized cause of the software failure. When the distinction between fault and failure is not critical, *defect* can be used as a generic term to refer to either a fault (cause) or a failure (effect).

2.4 Error

The term error has two different meanings [3], [10]:

1. A discrepancy between a computed, observed, or measured value, or condition and the true, specified, or theoretically correct value or condition. Errors occur when some part of the software produces an undesired state. Examples include exceptional conditions raised by the activation of existing software faults and an incorrect system status due to an unexpected external interference. This term is especially useful in fault-tolerant computing to describe an intermediate stage in-between faults and failures.
2. A human action that results in software containing a fault. Examples include omission or misinterpretation of user requirements in a software specification, and incorrect translation or omission of a requirement in a software design. However, this is not a preferred usage, and the term *mistake* is used instead to avoid the confusion.

2.5 Approaches to the Achievement of Reliability

The development of a reliable software system calls for the combined utilization of a set of methods and techniques which can be classed into [3], [16], [25-30], [32], [34], [36]:

- *Fault prevention:* how to prevent fault occurrence or introduction. The interactive refinement of the user's system requirement, requirements engineering (RE), the use of sound design methods, and the encouragement of writing clear code are the general approaches to prevent faults in the software. Formal methods develop and refine requirement specifications correctly using languages and tools with sound mathematical bases in order to achieve the following goals: 1) executable specifications for systematic and precise evaluation, 2) proof mechanisms for step-by-step verification using incremental refinement, and 3) every intermediate artifact is a subject to mathematical verification for correctness and appropriateness.
- *Fault removal:* how to reduce the presence (number and seriousness) of faults. Fault removal uses techniques such as testing, inspection, verification, and validation to track and remove faults in software. Formal inspection is a practical fault removal

scheme which is widely implemented in industry. Formal inspection is a rigorous process focused on finding faults, correcting faults, and verifying the corrections.

- *Fault tolerance:* how to ensure a service capable of fulfilling the system's function in the presence of faults. Software fault tolerance is achieved by design diversity in which multiple versions of software are developed. These multiple versions, which are functionally equivalent yet independent, are applied in the system to provide ultimate tolerance to software design faults.
- *Fault forecasting:* how to estimate the present number, future incidence, and consequences of faults. Fault forecasting involves formulation of the fault/failure relationship, an understanding of the operational environment, the establishment of reliability models, the collection of failure data, the application of reliability models by tools, the selection of appropriate models, and the analysis and interpretation of results.

2.6 Reliability assessment

The three major classes of software reliability assessment are [8-9], [14], [24]:

- *Black box reliability analysis:* Estimation of the software reliability based on failure observations from testing or operation. These approaches are called black boxapproaches because internal details of the software are not considered.
- *Software metric based reliability analysis:* Reliability evaluation based on the static analysis of the software (e.g., lines of code, number of statements, complexity) or its development process and conditions (e.g., developer experience, applied testing methods).
- *Architecture-based reliability analysis:* Evaluation of the software system reliability from software component reliabilities and the system architecture (the way the system is composed out of the components). These approaches are sometimes called *component-based reliability estimation* (CBRE), or *grey* or *white box* approaches.

2.7 Reliability measurement

Measurement of software reliability includes two types of activities: reliability *estimation* and reliability *prediction* [11], [13]. Estimation determines current software reliability by applying statistical inference techniques to failure data obtained during system test or during system operation. This is a measure regarding the achieved reliability from the past until the current point. Its main purpose is to assess the current reliability and determine whether a reliability model is a good fit in retrospect. Prediction determines future software reliability based upon available software metrics and measures [15]. Depending on the software development stage, prediction involves different techniques [17-23]:

1. When failure data are available (e.g., software is in system test or operation stage), the estimation techniques can be used to parameterize and verify software reliability models, which can perform future reliability prediction.
2. When failure data are not available (e.g., software is in design or implementation stages), the metrics obtained from the software development process and the characteristics of the resulting product can be used to predict reliability of the software.

Data collected during the test phase is often used to estimate the number of software faults remaining in a system which in turn often is used as input for reliability prediction. This estimation can either be done by looking at the numbers (and the rate) of faults found during testing or just by looking at the effort that was spent on testing. The underlying assumption when looking at testing effort is "more testing leads to higher reliability" [31], [33], [35].

3. Quantifying the reliability of the software developed using $\mathrm{SDA}_{\mathrm{Flex\&Rel}}$

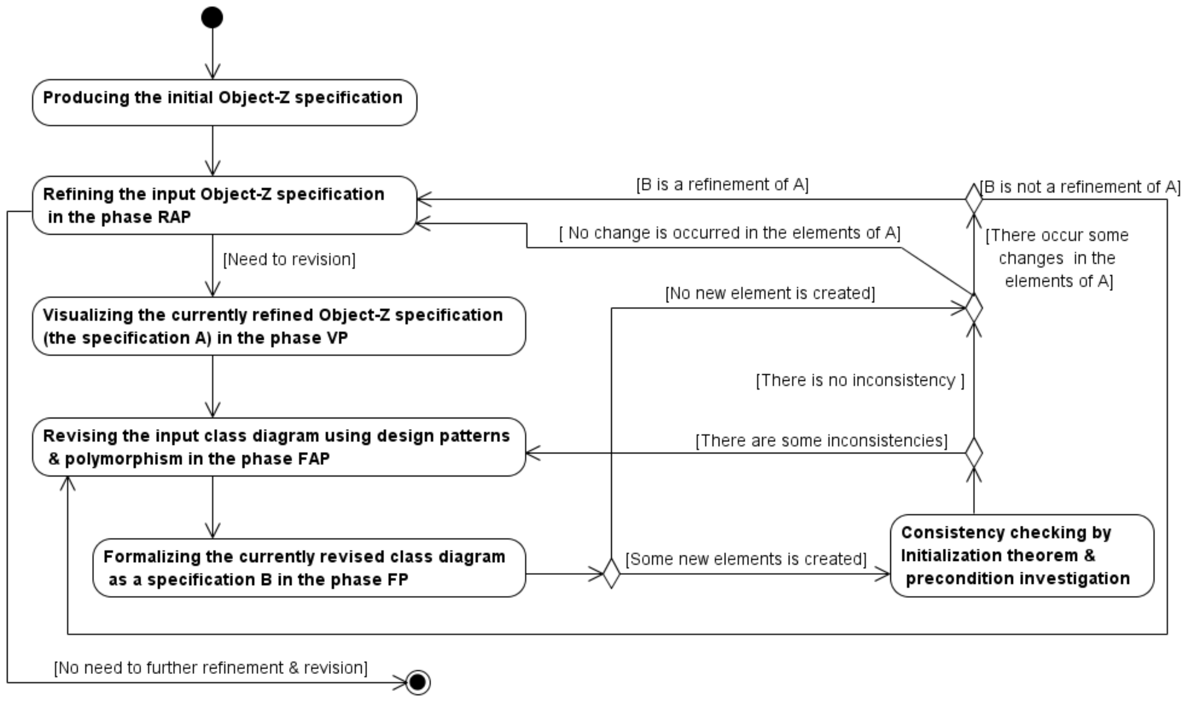
The software development approach $\mathrm{SDA}_{\mathrm{Flex\&Rel}}$ has recently been proposed to develop reliable yet flexible software [7]. In $\mathrm{SDA}_{\mathrm{Flex\&Rel}}$, formal (Object-Z) and semi-formal (UML) models are transformed into each other using a set of bidirectional formal rules. In this approach, Object-Z, as a dominant formal specification language, is used to formally specify, verify, and refine requirements to prevent and remove probable faults. As previously mentioned, fault prevention and fault removal are two main approaches to the development of reliable software systems. Therefore, formal modeling, verification, and refinement in Object-Z ensure the reliability of software. Visual models (UML diagrams) facilitate the interactions among stakeholders who are not familiar enough with the complex mathematical concepts of formal modeling methods. Applying design patterns to visual models improves the flexibility of software. The transformation of formal and visual models into each other through the iterative and evolutionary process, proposed in [7], helps develop the software applications that need to be highly reliable yet flexible. The workflow of $\mathrm{SDA}_{\mathrm{Flex\&Rel}}$ is illustrated in Fig. 2.

The iterative and evolutionary process illustrated in Fig. 2 continues until a final product with a desired quality (in terms of reliability and flexibility) is achieved. Fig. 3 illustrates the details of an iteration of $\mathrm{SDA}_{\mathrm{Flex\&Rel}}$ which consists of the following phases:

- Reliability Assurance Phase (RAP) which supports formal specification and refinement in Object-Z.
- Visualization Phase (VP) which transforms Object-Z models into UML ones.
- Flexibility Assurance Phase (FAP) which revises UML models from the viewpoints of design patterns and polymorphism.

- Formalization Phase (FP) which transforms UML models into Object-Z ones.

In order to assess/measure the reliability of the software developed using SDA$_{\text{Flex\&Rel}}$, the reliability of SDA$_{\text{Flex\&Rel}}$ is evaluated because there is a direct relation between the reliability of software and the reliability of the corresponding development approach [2-3], [6]. In other words, software reliability is directly proportional to the reliability of the development approach used. As previously mentioned, from the view point of assessment, such reliability assessment is categorized as software metric based reliability analysis, and from the viewpoint of measurement, such reliability measurement is categorized as prediction when failure data are not available. According to the details of each iteration in the proposed approach, the total reliability of SDA$_{\text{Flex\&Rel}}$ is calculated as:



Producing the initial Object-Z specification
Refining the input Object-Z specification in the phase RAP
[Need to revision]
Visualizing the currently refined Object-Z specification (the specification A) in the phase VP
Revising the input class diagram using design patterns & polymorphism in the phase FAP
Formalizing the currently revised class diagram as a specification B in the phase FP
[Some new elements is created]
Consistency checking by Initialization theorem & precondition investigation
[There are some inconsistencies]
[There is no inconsistency]
[No new element is created]
[No change is occurred in the elements of A]
[There occur some changes in the elements of A]
[B is a refinement of A]
[B is not a refinement of A]
[No need to further refinement & revision]

Fig. 2. The workflow of SDA$_{\text{Flex\&Rel}}$

$$R_{SDA_{Flex\&Rel}} = \prod_{i=1}^{k}\prod_{j=1}^{n_i} R_{RAP}(i,j) * \prod_{i=1}^{k} R_{VP}(i) * \prod_{i=1}^{k}\prod_{j=1}^{m_i} R_{FAP}(i,j) * \prod_{i=1}^{k} R_{FP}(i) \quad (3)$$

k	Number of iterations in the development process proposed by SDA$_{\text{Flex\&Rel}}$.
n_i	Number of formal refinement steps during RAP in the iteration i of SDA$_{\text{Flex\&Rel}}$.
m_i	Number of revision steps during FAP in the iteration i of SDA$_{\text{Flex\&Rel}}$.
$R_{RAP}(i,j)$	Reliability of the jth formal refinement step in RAP during the ith iteration.
$R_{VP}(i)$	Reliability of the ith formal transformation from Object-Z into UML (formalization) in VP.
$R_{FAP}(i,j)$	The reliability of the jth revision step in FAP during the ith iteration of SDA$_{\text{Flex\&Rel}}$.
$R_{FP}(i)$	Reliability of the ith formal transformation from UML into Object-Z (visualization) in FP.
$R_{SDA_{Flex\&Rel}}$	Total reliability of SDA$_{\text{Flex\&Rel}}$.

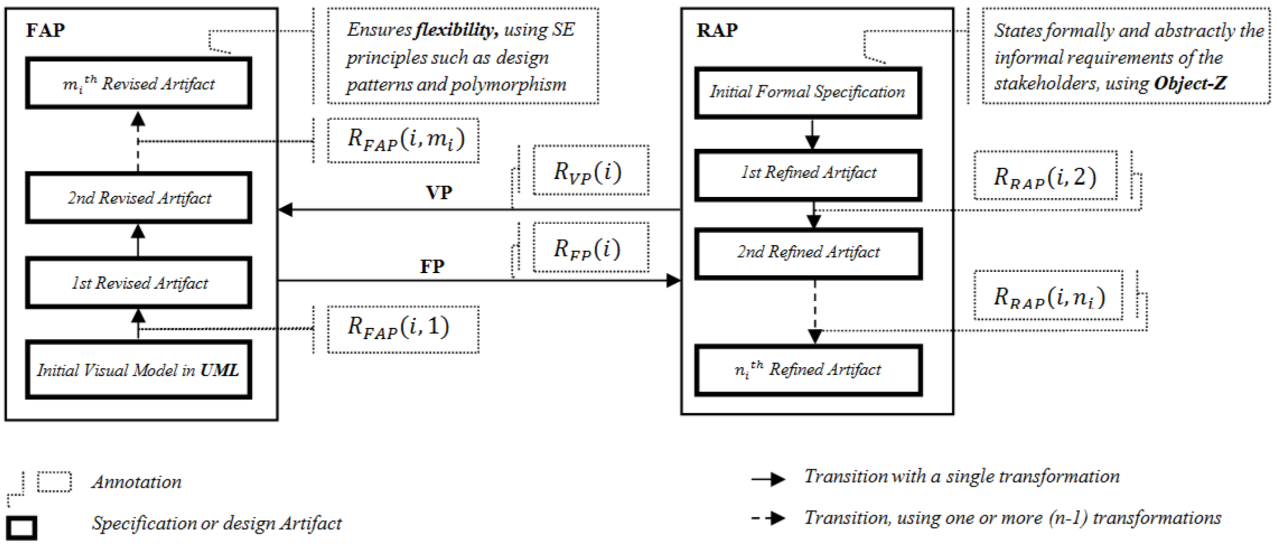
As previously mentioned, contrary to hardware, software does not wear out or deteriorate, i.e., its reliability does not decrease with time due to physical depreciation. However, Software may experience reliability decrease due to abrupt changes of its operational usage or incorrect modifications to the software. Therefore, the reliability of a flexible software or a flexible software development approach (such as SDA$_{\text{Flex\&Rel}}$) does not decrease with time because “flexibility” is defined as the ability of a system to respond to potential internal or external changes affecting its value delivery, in a timely and cost-effective manner. In other word, the reliability of software or a software development approach in the presence of flexibility is equivalent to R (reliability in the absence of time) instead of $R(t)$ (general definition of reliability presented in subsection 2.2). According to the fact that the flexibility of SDA$_{\text{Flex\&Rel}}$ has been demonstrated in [42], the reliability of SDA$_{\text{Flex\&Rel}}$ is calculated regardless of time as $R_{SDA_{Flex\&Rel}}$ instead of $R_{SDA_{Flex\&Rel}}(\mathrm{t})$.

In the current version of SDA$_{\text{Flex\&Rel}}$, all activities of the phases RAP, FP, and VP are performed formally with sound mathematical bases. The proposed formal transformation rules make it possible to transform UML class diagrams and Object-Z specifications into each other without any fault during the phases FP and VP. Moreover, formal refinement, along with formal verification guarantees the correctness of the activities performed during the phase RAP. As previously mentioned, formalism ensures the absence of faults. Therefore, the reliability of each of those activities performed during the phases RAP, FP, and VP equal 1 $(\forall i,j,\ R_{RAP}(i,j) = R_{VP}(i) = R_{FP}(i,j) = 1)$.

However, in the current version of SDA$_{\text{Flex\&Rel}}$, during the phase FAP, designers apply the required design patterns to the class diagram of the system being developed without any formal systematic control. This may cause the syntactic or the semantic structure of the class diagram to become inconsistent. Therefore, the reliability of every activity in the phase FAP does not equal 1 $(\forall\, i,j,\ R_{FAP}(i,j) \neq 1)$. Relation (3) is then simplified as:

$$R_{SDA_{Flex\&Rel}} = (1) * (1) * \prod_{i=1}^{k}\prod_{j=1}^{m_i} R_{FAP}(i,j) * (1) \Rightarrow$$

$$R_{SDA_{Flex\&Rel}} = \prod_{i=1}^{k}\prod_{j=1}^{m_i} R_{FAP}(i,j) \qquad (4)$$
$$\forall i,j,\ \ 0 < R_{FAP}(i,j) < 1$$



FAP
m_i th Revised Artifact
2nd Revised Artifact
1st Revised Artifact
Initial Visual Model in UML
Ensures flexibility, using SE principles such as design patterns and polymorphism
$R_{FAP}(i,m_i)$
$R_{FAP}(i,1)$
$R_{VP}(i)$
VP
$R_{FP}(i)$
FP
RAP
Initial Formal Specification
1st Refined Artifact
2nd Refined Artifact
n_i th Refined Artifact
States formally and abstractly the informal requirements of the stakeholders, using Object-Z
$R_{RAP}(i,2)$
$R_{RAP}(i,n_i)$
Annotation
Specification or design Artifact
Transition with a single transformation
Transition, using one or more (n-1) transformations

Fig. 3. A schematic view of an iteration i of SDA*Flex&Rel*

A formal mechanism can be proposed to make the class diagram of the software being developed be formally revised when a design pattern is applied to it in FAP. As a result, applying design patterns to the class diagram of software not only improves the flexibility of the software but also preserves the syntactic and the semantic structure of the class diagram – which, in turn, leads to consistency preservation during the revision process of the class diagram in FAP. To do so, a set of formal rules can be defined using model refactoring based on graph transformation at the meta-level of the UML class diagram to make it possible to add/remove/change a modeling element to/from/in a class diagram without making its syntax and semantics become inconsistent. Designers are then allowed to change a class diagram just using the defined rules in order to revise it based on a design pattern. Therefore, the reliability of every activity in the phase FAP will equal 1 $(\forall\, i,j,\ R_{FAP}(i,j) = 1)$. Relation (3) is then simplified further as:

$$R_{SDA_{Flex\&Rel}} = \prod_{i=1}^{k}\prod_{j=1}^{m_i} 1 = 1 \qquad (5)$$

As previously mentioned, the reliability of software developed using a development approach is directly proportional to the reliability of the development approach. Therefore, the reliability of the software developed using the current version of SDA$_{\text{Flex\&Rel}}$ is obtained according to relation (4), but in the future, by proposing a formal mechanism for supporting the revision process of the phase FAP, the reliability of the software increases to 1 according to relation (5).

Generally, a software development process includes several (n) activities [6], [44-45]. Based on the assumption that these activities do not enjoy a sound mathematical (formal) basis, the reliability of each of them dose not equal to 1 $(\forall\, i, R(i) \neq 1)$. As a result, the process reliability can be formulated as:

$$R_{GenericProcess} = \prod_{i=1}^{n} R(i)\ \forall i,\ \ 0 < R(i) < 1 \qquad (6)$$

In order to simply compare $R_{GenericProcess}$ with $R_{SDA_{Flex\&Rel}}$, relation (4) is reformulated as follows:

$$R_{SDA_{Flex\&Rel}} = \prod_{i=1}^{k}\prod_{j=1}^{m_i} R_{FAP}(i,j) = \prod_{p=1}^{k'} R_{FAP}(i',j') \quad (7)$$

such that:

$$\forall\, i', j',\ \ 0 < R_{FAP}(i', j') < 1,$$

$$k' = \sum_{i=1}^{k} m_i,$$

$$\forall p, 0 \le n' < k \cdot \sum_{j=1}^{n'} m_j \le p \le \sum_{j=1}^{n'+1} m_j$$

$$\Rightarrow i' = n' + 1 \wedge j' = i - \sum_{j=1}^{n'} m_j$$

The following assumptions are made according to relations (6) and (7):

1. In $R_{GenericProcess}, \forall i,\ 0 < R(i) < 1,$
2. In $R_{SDA_{Flex\&Rel}}, \forall\, p, i', j',\ 0 < R_{FAP}(i', j') < 1,$
3. $\forall\, i', j', i,\ R_{FAP}(i', j') \nless R(i)$, because: 1) the input materials of FAP are correct and fault-free and 2) During FAP, the input materials are just revised using design patterns and polymorphism with low possibility of fault occurrence, and 3) the lack of semantic inconsistency between the input and the output of the phase FAP is guaranteed by the existing formal analysis techniques (such as initialization theorem and precondition investigation) and the various formal verification mechanisms that support Object-Z (as illustrated in Fig. 2).
4. $k' \ll n,$

With respect to these assumptions, we can conclude that:

$$\prod_{p=1}^{k'} R_{FAP}(i', j') \prod_{i=1}^{n} R(i) \Rightarrow \qquad (8)$$

$$R_{SDA_{Flex\&Rel}} \gg R_{GenericProcess}$$

The above-mentioned analysis shows that the reliability of SDA$_{\text{Flex\&Rel}}$ is greater than the reliability of a generic software development process. The main conclusion is that the more widespread the use of formalism including formal specifications, refinement, and verification throughout a software development process, the more reliable the software development process will be. Therefore, supposing that some (m) of the activities of the software development process $GenericProcessare$ performed formally, the reliability of $GenericProcess$, previously formulated as relation (6), is reformulated as relation (9):

$$R_{GenericProcess} = \prod_{i=1}^{n} R(i) = \prod_{i=1}^{m} R(i) * \prod_{i=1}^{n-m} R(i)$$

$$\xRightarrow{\forall i, 1 \le i \le m,\ R(i)=1} R_{GenericProcess} = 1 * \prod_{i=1}^{n-m} R(i) \Rightarrow$$

$$R_{GenericProcess} = \prod_{i=1}^{n-m} R(i)\ \forall i,\ \ 0 < R(i) < 1 \qquad (9)$$

Conclusion: $m \to n\ \Rightarrow\ R_{GenericProcess} \to 1$

With respect to the fact that the reliability of a software product is directly proportional to the reliability of the development approach used, the more reliable the software development approach, the more reliable the software product. According to the aforementioned analyses, the reliability of software developed using SDA$_{\text{Flex\&Rel}}$ is greater than the reliability of software developed using a generic software development process.

4. Conclusions

In this paper, we quantify the reliability improvement promised by the software development approach SDA$_{\text{Flex\&Rel}}$, which has recently been proposed to develop reliable yet flexible software. This approach improves software reliability through preparing the ground for formal modeling, refinement, and verification– which, in turn, prevent and remove probable faults. In order to quantify the reliability of the software developed using SDA$_{\text{Flex\&Rel}}$, the reliability of SDA$_{\text{Flex\&Rel}}$ is quantitatively measured because there is a direct relation between the reliability of software and the reliability of the corresponding development approach. In other words, software reliability is directly proportional to the reliability of the development approach used. Such reliability assessment is categorized as software metric based reliability analysis. The results confirm the promised reliability improvement

References

[1] ISO/IEC/IEEE, *Systems and software engineering – Vocabulary*, ISO/IEC/IEEE 24765:2010.

[2] H. Pham, *System Software Reliability*, Springer, 1st ed., 2007.

[3] M. R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

[4] A. Pandit, "A Framework-Based Approach for Reliability & Quality Assurance of Safety-Critical Software," *Int. Journal on Computer Science and Eng.*, vol. 2 (9), pp. 2874-2879, 2010.

[5] H. B. Christensen, *Flexible, Reliable Software: Using Patterns and Agile Development*, Chapman and Hall/CRC; 1st ed., 2010.

[6] D. Bjørner, *Software Engineering III: Domains, Requirements, and Software Design*, Springer, 2006.

[7] A. Rasoolzadegan, A. Abdollahzadeh, "Reliable yet Flexible Software through Formal Model Transformation (Rule Definition)," *Journal of Knowledge and Information Systems (KAIS)*, vol. 40 (1), 2014.

[8] W. Ecker, W. Müller, Rainer Dömer, *Hardware Dependent Software Principles and Practice*, Springer, 2009.

[9] L. I. Millett, *Software for Dependable Systems: Sufficient Evidence?*, The National Academies Press, 2007.

[10] International Organization for Standardization, *ISO Standard 9126: Software Engineering – Product Quality, parts 1, 2 and 3*, Geneve, Switzerland, 2001 (part 1), 2003 (parts 2 and 3).

[11] M. Rahmani, A. Azadmanesh, "Exploitation of Quantitative Approaches to Software Reliability," *Tech. Rep. cst-2011-002*, Computer Science, University of Nebraska, Omaha, Dec. 2011.

[12] P.H. Seong, *Reliability and Risk Issues in Large Scale Safety-critical Digital Control Systems*, Springer, 1st ed., Berlin, Germany, pp. 85–87, 2009.

[13] X. Li, "Software reliability measurement: a survey," *MSc. Thesis,* Dept. Computer Science & Software Engineering, Concordia University, 2002.

[14] P. C. J. P. K. Kapur, H. Pham, A. *Gupta, Software Reliability Assessment with OR Applications*, 1st ed., London, England, Springer, 2011.

[15] A. K. Pandey, N.K. Goyal, *Early Software Reliability Prediction: a Fuzzy Logic Approach*, Springer, 2013.

[16] S. Yamada, *Software reliability modeling Fundamental and Applications*, Japan, Springer, 2014.

[17] Q. P. Hu, Y.-S. Dai, M. Xie, S. H. Ng, "Early software reliability prediction with extended ANN model," *in30th Annual International Conference on Computer Software and Applications*, 2006, vol. 2, pp. 234–239, 2006.

[18] S. Mohanta, G. Vinod, A. K. Ghosh, R. Mall, "An approach for early prediction of software reliability", *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 6, pp. 1–9, 2010.

[19] A. Immonen, E. Niemela, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software & System Modeling*, Springer, vol. 7, no. 1, pp. 49-65, Jan 2007.

[20] S. S. Gokhale, K. S. Trivedi, "Analytical models for architecture-based software reliability prediction: A unification framework," *Reliab. IEEE Trans*., vol. 55, no. 4, pp. 578–590, 2006.

[21] R.H. Reussner, H.W. Schimidt, I.H. Poernomo, "Reliability prediction for component-based software architectures," *J. System Softw*., vol. 66, no. 3, pp. 241-252, 2003.

[22] G.N. Rodrigues, D.S. Rosenblum, S. Uchitel, "Using scenarios to predict the reliability of concurrent component-based software systems," *in Proceedings of the8th international conference on Fundamental Approaches to Software Engineering*, pp. 111-126, 2005.

[23] S. S. Gokhale, K. S. Trivedi, "reliability prediction and sensitivity analysis based on software architecture," *in Proceedings ofthe 3rd international symposiumon Software Reliability Engineering*, pp. 64-75, 2002.

[24] K. Goševa-Popstojanova, K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Journal of Performance Evaluation*, Elsevier, vol. 45, no. 2, pp. 179–204, 2001.

[25] H. A. Stiber, "A family of software reliability growth models," *in Proceeding of 31th Annual International Computer Software and Applications Conference*, IEEE, vol. 2, pp. 217-224, July 2007.

[26] H. Pham, "Software reliability and cost models: Perspectives, comparison, and practice," *Eur. J. Oper. Res*., vol. 149, no. 3, pp. 475–489, 2003.

[27] A. L. Goel, K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, pp. 206 – 211, 2009.

[28] V. Volovoi, "Modeling of System Reliability Using Petri Nets with Aging Tokens," *J. Reliab. Eng. Syst. Saf*., vol. 84, pp. 149–161, 2004.

[29] M. Xie, K.-L. Poh, Y.-S. Dai, *Computing System Reliability: Models and Analysis*, 1st ed., New York, USA: Springer, 2004.

[30] M. Ohba, "Software reliability analysis models," *IBM J. Res. Dev*., vol. 28, no. 4, pp. 428–443, 1984.

[31] L. K. Singh, A. K. Tripathi, G. Vinod, "Software reliability early prediction in architectural design phase: Overview and Limitations," *J. Softw. Eng. Appl*., vol. 4, p. 181, 2011.

[32] W.L. Wang, M.H. Chen, "Heterogeneous software reliability modeling," *in Proceedings of 13th International Symposium on Software Reliability Engineering*, pp. 41 – 52, 2002.

[33] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Torner, W. Meding, C. Hoglund, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," *System and Software*, Elsevier, vol. 98, pp. 59–78, 2014.

[34] R. Lai, M. Garg, "A Detailed Study of NHPP Software Reliability Models (Invited Paper)," *J. Softw*., vol. 7, no. 6, pp. 1296–1306, Jun. 2012.

[35] K. Goševa-Popstojanova, K. S. Trivedi, "Architecture-based approaches to software reliability prediction," *International Journal of Computer Mathematics with Applications*, vol. 46, no. 7, pp. 1023–1036, 2003.

[36] S. S. Gokhale, M.-T. Lyu, "A simulation approach to structure-based software reliability analysis," *Softw. Eng. IEEE Trans*., vol. 31, no. 8, pp. 643–656, 2005.

[37] K.C. Chiu, Y.S. Huang, T.Z. Lee, "A study of software reliability growth from the perspective of learning effects," *International Journal of Reliability Engineering & System Safety*, vol. 93, no. 10, pp. 1410–1421, 2008.

[38] K. M. Cheol, J. S. Cheol, J. J. Ha, "Possibilities and Limitations of Applying Software Reliability Growth Models to Safety- Critical Software," *Journal of Nuclear Engineering and Technology*, vol. 39, no. 2, pp. 129–132, 2007.

[39] V. Almering, M. Van Genuchten, G. Cloudt, P. J. M. Sonnemans, "Using software reliability growth models in practice," *Software*, IEEE, vol. 24, no. 6, pp. 82–88, 2007.

[40] B. Cukic, E. Gunel, H. Singh, G. U. O. Lan, "The theory of software reliability corroboration," *IEICE Trans. Inf. Syst*., vol. 86, no. 10, pp. 2121–2129, 2003.

[41] K. Sharma, R. Garg, C. K. Nagpal, R. K. Garg, "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach," *IEEE Transactions on Reliability*, pp. 266–276, 2010.

[42] A. Rasoolzadegan, "A New Approach to the Quantitative Measurement of Software Flexibility," *Journal of Soft Computing and Information Technology*, submitted, to be evaluated.

[43] M. Hashemi, Z. Ghavidel, A. Rasoolzadegan, "A Systematic Literature Review on Software Reliability Modeling," *Journal of Modeling in Engineering*, submitted, to be evaluated.

[44] R. S. Pressman, *Software Engineering-A Practitioner's Approach-Required*, 7th ed. McGraw Hill, 2009.

[45] I. Sommerville, *Software Engineering*, 9th ed. Addison Wesley, 2011.

Abbas Rasoolzadegan has received his B.Sc. degree in Software Engineering from Air Force University in 2004, Tehran, Iran. He has also received M.Sc. and Ph.D. degrees in Software Engineering from Amirkabir University of Technology, Tehran, Iran, respectively in 2007 and 2013. During his Ph.D., he has worked on formal software engineering and model transformation. He is currently an assistant professor in the Computer Engineering Department of Ferdowsi University of Mashhad. His main research focus is on software quality engineering, model transformation, testing, and design patterns.