

Measuring Goal-Oriented Requirements Language Actor Stability

Jameleddine Hassine*, Mohammad Alshayeb*

*Information and Computer Science Department, King Fahd University of Petroleum and Minerals

jhassine@kfupm.edu.sa, Alshayeb@kfupm.edu.sa

Abstract

Background: Goal models describe interests, preferences, intentions, desired goals and strategies of intervening stakeholders during the early requirements engineering stage. When capturing the requirements of real-world systems such as socio-technical systems, the produced goal models evolve quickly to become large and complex. Hence, gaining a sufficient level of understanding of such goal models, to perform maintenance tasks, becomes more challenging. Metric-based approaches have shown good potential in improving software designs and making them more understandable and easier to maintain.

Aim: In this paper, we propose a novel metric to measure GRL (Goal-oriented Requirements Language) “actor stability” that provides a quantitative indicator of the actor maintainability.

Method: We first, validate the proposed metric theoretically then empirically using a case study of a GRL model describing the fostering of the relationship between the university and its alumni.

Results: The proposed actor stability metric is found to have significant negative correlation with the maintenance effort of GRL models.

Conclusions: Our results show that the proposed metric is a good indicator of GRL actors’ stability.

Keywords: Goal models, Goal-oriented Requirements Language (GRL), stability, metrics, maintenance

1. Introduction

Software systems must evolve to meet customer needs, business environment, technologies and regulations. Several studies have shown that requirements evolution can significantly affect overall project costs and schedule [1, 2]. Requirements evolution management has emerged as one important topic in requirements engineering research [3]. ISO/IEC 25010 [4] specified eight characteristics for software product external quality, one of which is maintainability which contains modifiability is a sub-characteristic. Modifiability is a combination of changeability and stability. Stability is the ability of the software to remain stable when modified. Maintainable software tends to have a better estimation of the change cost and better prediction of the resulting quality [1].

Goal models are used in order to make sure that stakeholders’ interests and priorities are met in the early requirements engineering stages. Goal modeling is an effective approach to represent and reason about stakeholders’ goals using models. Over the past two decades, numerous goal-oriented modeling languages and approaches have been introduced (e.g., i^* [5], NFR Framework [6], Keep All Objects Satisfied (KAOS) [7], TROPOS [8] and the Goal-oriented Requirements Language (GRL) [9] part of the ITU-T standard User Requirements Notation (URN)). In addition, there were few attempts to propose domain-specific languages, such as DSML/GoalML [10], DSL/KAOS [11] and ARMOR/KAOS [12].

As goal models grow in size and complexity (e.g., large socio-technical systems having many interdependent stakeholders), they be-

come difficult to maintain. To address this challenge, numerous goal-oriented metrics-based techniques [13–18] have been proposed. These techniques vary in their targeted notation, their aim, their selected analysis (e.g., quantitative, qualitative, hybrid) and their targeted scope (e.g., global (targeting the entire goal model), local (focusing on one specific actor or path)).

Goals are known to be much more stable than requirements [19, 20]. More specifically, the higher level the goal is, the more stable it is [19]. However, in a fast-changing world, goal models are deemed to evolve to meet constant changes of business needs and stakeholders' intentions. A goal model requires maintenance when there is a shift in stakeholder's motivations, e.g., adoption of new goals or ceasing to support existing goals. Goals may become undesirable or infeasible to realize, e.g., goals might become too costly to realize or non-compliant with new regulations [21]. Hence, an outdated representation of stakeholders' intentions can easily lead to systems that do not fulfill their purpose. According to a recent survey by Horkoff et al. [22], interest in adaptation/variability/evolution of goal models has increased recently compared to other goal-oriented requirements engineering (GORE) topics. Although, there is a variety of empirical evaluations in the area of modeling languages in general (assessing different qualities, e.g., syntactic, semantic, pragmatic, completeness, comprehensibility, complexity, etc.), the majority of the studies in GORE [23] focus on providing empirical evidence of the applicability of goal-oriented notations for specific domains [24], such as collaborative systems [25], socio-technical systems [26] and knowledge transfer [27].

Most of the existing work that addresses the evolution of goal models, focuses mainly on handling inconsistencies (such as tolerating, diagnosing and tracking inconsistencies) [28–31] and modeling and analysis of evolution over time [32, 33]. However, both approaches introduced in [32] and [33] consider only the evolution of goals' satisfactions values (qualitative and quantitative) and do not discuss the evolution of the goal model structure.

Measuring stability provides better estimation of the cost and effort and better prediction of

the software quality [34]. Unstable software tends to increase maintenance cost; in some cases, the maintenance cost may reach up to 75% of the software total cost [1, 2]. Stable software, on the other hand, reduces maintenance cost. We believe that the design of a GRL actor stability metric would provide information about GRL actors and their evolution; which will provide control over actor-specific change amplification. Furthermore, measuring GRL actor stability gives an indicator of the GRL actor and model maintainability since stability is directly related to maintainability [4].

The main motivation of this paper is to propose a metric to support the maintainability of goal models during the requirements modeling and analysis phase. In particular, we focus on measuring quantitatively the stability of actors across many versions of the goal model. This paper provides the following contributions:

- Propose a novel metric to measure GRL actor stability. To the best of our knowledge, no goal-based stability metrics were introduced in the literature.
- Validate theoretically and empirically the proposed GRL-based actor stability metric.
- Provide a foundation for systematic assessment of GRL actor stability with respect to the many changes undergone by GRL models during the development life cycle, e.g., model refinement, validation, and maintenance.

The remainder of this paper is organized as follows. In Section 2, we review the current state of the art. Section 3 introduces briefly the GRL language. Our proposed actor stability metric is presented in Section 4. In Section 5, we provide theoretical and empirical validation of the proposed metric and we discuss the possible threats to validity. Section 6 discusses the interpretation and benefits of the proposed metrics. Finally, conclusions and future work are presented in Section 7.

2. Related Work

In this section, we review software stability metrics, goal models and requirements stability measurements.

2.1. Software stability

Researchers proposed stability metrics at system [35–39], model [40], architecture [41–44] and class levels [38, 45–47]. Classes in object-oriented (OO) systems form the basic building blocks and thus they are the most related stability metrics to the metric proposed in this paper as we propose a stability metric at actor level, which is also the basic building block for GRL models, hence, we discuss these metrics in details.

Li et al. [38] proposed the “Class Implementation Instability”(CII) metric to measure the evolutionary change in the implementation of a class. The authors in [38] measure class instability by measuring the lines of code added, deleted, or modified between two versions. CII metric is not normalized; therefore, its value has no upper or lower limit. Grosser et al. [45] proposed a metric to measure the class stability based on the method interface; the class is considered stable if the method interfaces are unchanged between versions. Hence, the metric measures the number of methods whose interface (signature) has not been changed between two versions regardless of the changes occurred to the method bodies. According to Grosser et al. [45], the class is fully stable when all method signatures available in one version are available in the other version. On the other hand, the class is considered fully instable, when none of the method signatures remain unchanged between the two versions. This metric is normalized and yields a value between 0 and 1. Ratiu et al. [46] proposed a class stability metric that uses the number of methods in a class between two versions. According to Ratiu et al. [46] the class can be either stable or instable i.e., the value of the metric can be either 0 for instable or 1 for stable classes. A class is stable when the number of methods between two versions remain is unchanged; the class is instable when there is a change in the number of methods between two versions. Alshayeb et al. [47] proposed a Class Stability Metric (CSM) to measure Object-Oriented (OO) class stability. The authors [47] analyzed the OO class properties and identified eight class properties that affect class stability. These properties are: method

access-level, method code, method signature, class variable access-level, class variable, class access-level, class interface name and inherited class name. For each property, the authors [47] measure the extent of change between two versions by measuring the unchanged properties. The class stability is measured by aggregating the individual stability values for all the properties. CSM is normalized and hence the value of the metric can be between 0 (fully instable) and 1 (fully stable).

There are three approaches to defining software systems’ stability for model or code levels. The first approach is that the software system is stable if no changes are made to the software artifact being measured. Thus, the software is fully stable when the original and the subsequent version are identical [48]. The second approach considers the software system as stable if it avoids addition of new artifacts or modification of existing ones, thus deletion is considered as modification [34]. The third approach allows additions to the existing software system. Thus, the software is considered fully stable when there are no changes to the existing artifacts regardless of the additions that might be made to the system [49]. In this paper, we adopt the third stability definition when defining the proposed metric.

2.2. Goal models measurement

There is a growing body of literature on goal-oriented metrics [5, 7, 13–16, 18, 50–52]. Kaiya et al. [17] proposed quality metrics (introduced as part of the AGORA (Attributed Goal-oriented Requirements Analysis) approach) to measure correctness, unambiguity, consistency, verifiability, modifiability, traceability, and completeness of AND-OR goal graphs according to a stakeholder preference matrix. The proposed metrics are global and do not consider dependencies between intervening actors (used to describe stakeholders and systems in goal models). Franch and Maiden [13] introduced metrics to quantify i* Strategic Dependencies (SD) [5] models, that can be used to help choosing the most appropriate Commercial Off-The-Shelf (COTS) components.

These quantitative metrics are based on a categorization of the different types of strategic dependencies into duplicated and non-duplicated, hidden and non-hidden, resource and non-resource, etc. The resulting metrics are then applied to measure six system properties, namely, Diversity, Vulnerability, Packaging, Self-Containment, Uniformity, and Connectivity. In Franch et al. [14], i^* SD actors and dependencies are categorized into sorts, e.g., human/computer, goal/ task, etc. The proposed framework [14] implements three structural metrics, aiming to assess system properties, such as privacy, accuracy and efficiency. The proposed framework supports both global and local metrics and takes into account actor and dependency weights (i.e., importance values). Later, the approaches introduced in [13] and [14], have been applied by Grau et al. [16, 18] to assess the effectiveness of alternative architectures. In order to evaluate an architectural property, the authors proposed a coupling metric over i^* SD models. Coupling is measured by the number of incoming and outgoing dependencies (multiplied by a weight factor relative to each actor) an actor is associated with. To measure model predictability, Franch [15] has proposed a framework that considers both i^* SD and SR (Strategic Rationale) models. The predictability metric, expressed in Object Constraint Language (OCL), can be local or global and may require expert judgment. More recently, Franch [51] has proposed a method based on system domain analysis for defining metrics in i^* using the iMDF framework. Espada et al. [52] proposed quantitative metrics for evaluating the complexity and the completeness of KAOS [7] goal models. The authors used the Goal-Question-Metric (GQM) approach [50]. However, their approach does not consider dependencies between KAOS agents. Gralha et al. [53] proposed a set of metrics to measure and analyze complexity and completeness of goal models. In the GRL [9] context, Hassine and Alshayeb [54], proposed a structural metric to measure actor external dependencies (AED). Furthermore, jUCMNav [55] tool (GRL modeling and analysis framework) captures many simple structural GRL metrics (e.g., number of

actors, goals, tasks, intentional elements, intentional links, etc.) and allows for the definition of additional metrics using OCL.

In this paper, we extend the set of existing GRL metrics [54, 55] by introducing a novel metric to measure GRL actor stability. Our proposed metric is (1) structural in the sense that it depends only on the connectivity of a given GRL model and not on its semantic, (2) local since it is applied at the actor level rather than the entire GRL model, and (3) quantitative since it measures the degree of actor stability with respect to changes a GRL model can undergo, e.g., model refinement and maintenance.

2.3. Requirements stability measurement

Many metrics have been proposed to understand the sources, frequencies and types of requirements evolution. Lam and Shankararaman [56] proposed a change volatility metric to measure the number or proportion of changes, within a specified period. Change volatility [56] helps assess the stability of a system. A high-risk requirement may be characterized by a high-level change volatility. Anderson and Felici [57] proposed the Requirements Maturity Index (*RMI*), a metric used to quantify the readiness of requirements. The *RMI* is computed as follows:

$$RMI = \frac{RT - RC}{RT}$$

where *RT* is the total number of software requirements in the current release and *RC* is the number of requirements changes, i.e., added, deleted or modified requirements, allocated to the current release. It is worth noting that *RMI* metric is sensitive to requirements change in successive releases, but it does not take into account historical information about change. To address this issue Anderson and Felici [58] refined *RMI* by introducing the Requirements Stability Index (*RSI*) (a metric used to measure the extent of requirements stability and the frequency of changes to requirements) and the Historical Requirements Maturity Index (*HRMI*). *RSI* takes into account *CRC* (the cumulative number of

requirements changes) and is defined as follows:

$$RSI = \frac{RT - CRC}{RT}$$

HRMI takes into consideration *ARC* (the average number of requirement changes) and is defined as follows:


$$HRMI = \frac{RT - ARC}{RT}$$

The authors [57] claimed that *HRMI* are less sensitive than *RMI* to changes over consecutive releases. Stark et al. [58] characterized requirements volatility as additions to the delivery content, deletions from the delivery content and changes in scope to an agreed-upon requirement. More recently, AbuHassan and Alshayeb [40] proposed a suite of stability metrics for UML use case models, UML sequence diagrams and UML class diagrams. However, in the context of goal-oriented languages and to the best of our knowledge, no stability metrics have been proposed. In this paper, we aim to fill this gap by proposing a novel metric to measure GRL “actor stability” that provides a quantitative indicator of the actor maintainability, allowing for a better estimation of GRL models change.

3. GRL in a nutshell

The Goal-oriented Requirements Language (GRL) [9] is an ITU-T standard visual goal modeling language used to model stakeholders’ intentions, business goals and non-functional requirements. GRL is based on *i** [5] and the NFR framework [6]. In what follows, we briefly introduce the different GRL constructs.


3.1. GRL actors

An actor (illustrated as , where the name of the actor reference is shown as a label next to a stickman icon on the top-left side of the dashed ellipse) represents an entity that has intentions and carries out actions to achieve its goals by

exercising its know-how. Actors are often used to represent stakeholders as well as systems.

3.2. GRL intentional elements and indicators

There are five different types of intentional elements:

1. *Goal* (illustrated as \square): A (hard) *Goal* (either a business goal or a system goal) is a condition or state of affairs in the world that the stakeholders would like to achieve.
2. *Softgoal* (illustrated as \sqsupset): is a condition or state of affairs in the world that the actor would like to achieve, but there are no clear-cut criteria for whether the condition can be entirely achieved. However, it can be sufficiently achieved. Softgoals are often used to describe non-functional aspects such as availability, security, etc.
3. *Task* (illustrated as \triangleleft): states a particular way of performing something. Tasks can be considered as the operations, processes, data representations, structuring and constraints used to meet the needs stated in the goals and softgoals of the target system.
4. *Resource* (illustrated as \square): is a physical or informational entity.
5. *Belief* (illustrated as \circ): used to represent design rationale. Intentional elements may be included in actor definitions and they can be linked to each other in different ways. In addition to intentional elements, GRL defines *indicators* (illustrated as ) to describe a qualitative or quantitative real-world measurement.

3.3. GRL links

There are five types of GRL links [9]:

1. *Contributions* (illustrated as \longrightarrow): describe how a source intentional element or source indicator contributes to the satisfaction of a destination intentional element. A contribution has a qualitative level and an optional quantitative value.
2. *Correlations* (illustrated as \dashrightarrow): express knowledge about interactions between inten-

tional elements. A correlation link is similar to a contribution link except that the correlation is not an explicit desire but is a side-effect and that correlations are only used with intentional elements and not with indicators.

3. *Dependencies* (illustrated as $\rightarrow\rightarrow$): enable reasoning about how actor definitions depend on each other to achieve their desired goals. It describes how a source actor (the *depend-der*) depends on a destination actor (the *de-pendee*).
4. *Decompositions* (illustrated as $\rightarrow\vdash$): provide the ability to define what source intentional elements need to be satisfied in order for a target intentional element to be satisfied. There is no ordering between the decomposing elements. A decomposition link can be one of the following:
 - *AND* decomposition: The satisfaction of each of the sub-intentional elements is necessary to achieve the target.
 - *XOR* decomposition: The satisfaction of one and only one of the sub-intentional elements is necessary to achieve the target.
 - *OR* decomposition: The satisfaction of one of the sub-intentional elements is sufficient to achieve the target, but many sub-intentional elements can be satisfied.
5. *Belief links* (illustrated as $-----$): used to connect beliefs to GRL intentional elements.

3.4. Qualitative contributions

The qualitative contribution of a source intentional element or indicator to a destination intentional element can be one of the following values based on the degree (positive or negative) and sufficiency of the contribution to the satisfaction of the destination intentional element:

1. *Make* (illustrated as \dagger): the contribution is positive and sufficient.
2. *Help* (illustrated as \ddagger): the contribution is positive but not sufficient.
3. *SomePositive* (illustrated as \oplus): the contribution is positive, but the extent of the contribution is unknown.
4. *Unknown* (no symbol on the link): there is some contribution, but the extent and the de-

gree (positive or negative) of the contribution is unknown.

5. *SomeNegative* (illustrated as \ominus): the contribution is negative, but the extent of the contribution is unknown.
6. *Hurt* (illustrated as \mp): the contribution is negative but not sufficient.
7. *Break* (illustrated as $\omin�$): the contribution of the contributing element is negative and sufficient.

It is worth noting that GRL is permissive in how intentional elements can be linked to each other, contrary to i^* [5] which imposes restrictive usage of relationships (e.g., a contribution link cannot have a task as a destination).

For a detailed description of the GRL language, the reader is invited to consult the URN (User Requirements Notation) ITU-T standard [9].

3.5. GRL example

Figure 1 illustrates a GRL model composed of one GRL actor, called *Commuter*, who wants to minimize the time lost during a commute (modeled as a GRL goal “Minimize time lost by commute”). Two goals “Work during commute” and “Minimize travel time” contribute positively (through two *Help* contributions) to the achievement of the upper goal. While taking public transportation (represented as a goal called “take public transportation”) contributes positively (through a *Help* contribution) to the achievement of the goal “Work during commute”, it contributes negatively (through a *Hurt* contribution) to the achievement of the goal “Minimize travel time”. Similarly, taking private transportation (represented as a goal called “take private transportation”) contributes positively (through a *Help* contribution) to the achievement of the goal “Minimize travel time”, it contributes negatively (through a *Hurt* contribution) to the achievement of the goal “Work during commute”. When it comes to use public transportation, the commuter has the choice (illustrated as an OR decomposition link) between taking the regular bus (i.e., illustrated as task “Take regular bus”) or taking the express bus (i.e., illustrated as task “Take express bus”). Furthermore,

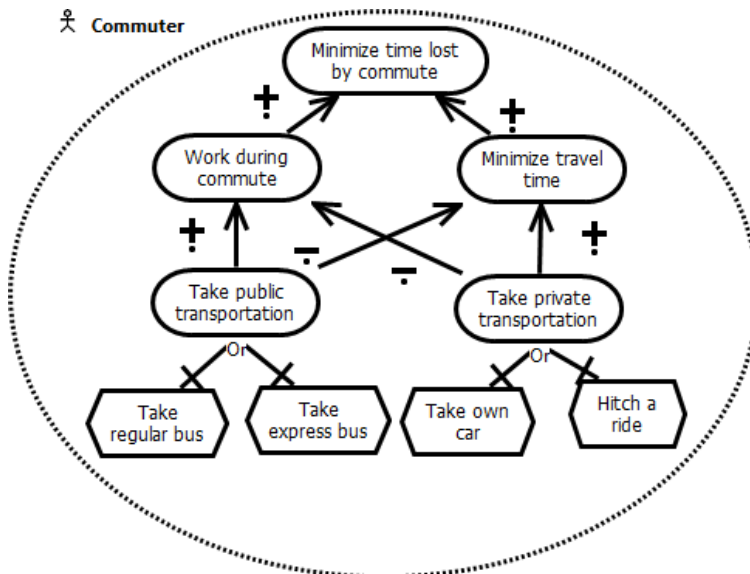


Figure 1. GRL commuter example

two options (illustrated as an OR decomposition link) are available to the commuter when it comes to the use of the private transportation: take his own car (represented as task “Take own car”) or hitch a ride (represented as task “Hitch a ride”).

4. Measuring GRL actor stability

In this section, we propose a novel metric to measure GRL actor stability and we provide an example to illustrate its calculation.

4.1. GRL Change Unit (GCU)

Conducting a maintenance task on a GRL model generates a new version (version i is the current version and version $i + 1$ is the modified version). To quantitatively assess the magnitude of a change, we should characterize the basic units (GRL sub-models) that are subject to change. We define the “GRL Change Unit (GCU)”, as being:

1. An intentional element combined with its outgoing link (with or without a quantitative value). It is worth noting that a GCU may have an outgoing link that crosses the boundary of the containing actor to reach another intentional element contained within another actor.

Or

2. An intentional element that does not have any outgoing links.

Links are not unique while intentional elements are unique within an actor, therefore, many similar links may exist in the actor model and hence we will not be able to identify which links remain unchanged. Therefore, we combine the intentional element with its outgoing link to form a GCU. Figure 2a illustrates a generic GRL example composed of two actors A and B. Actor A is composed of two GCUs: (1) GCU1 composed of task T1 and a *help* contribution and (2) GCU2 composed of goal G0 and the dependency link. Actor B is composed of three GCUs: (1) GCU1 composed of goal G1 and the AND decomposition, (2) GCU2 composed of goal G2 and the AND decomposition and (3) GCU3 composed of the softgoal SG1.

Compared with version i of a GRL model, version $i + 1$ may have added, deleted, changed, and unchanged GCUs. It is worth noting that we consider both syntactic and semantic changes. Examples of possible changes include: changing the type of an intentional element (e.g., from a goal to a task), changing the type of a decomposition (e.g., from OR to AND), changing the qualitative/quantitative values of a contribution (e.g., from *help* to *hurt*), changing the text of an intentional element with a different text not

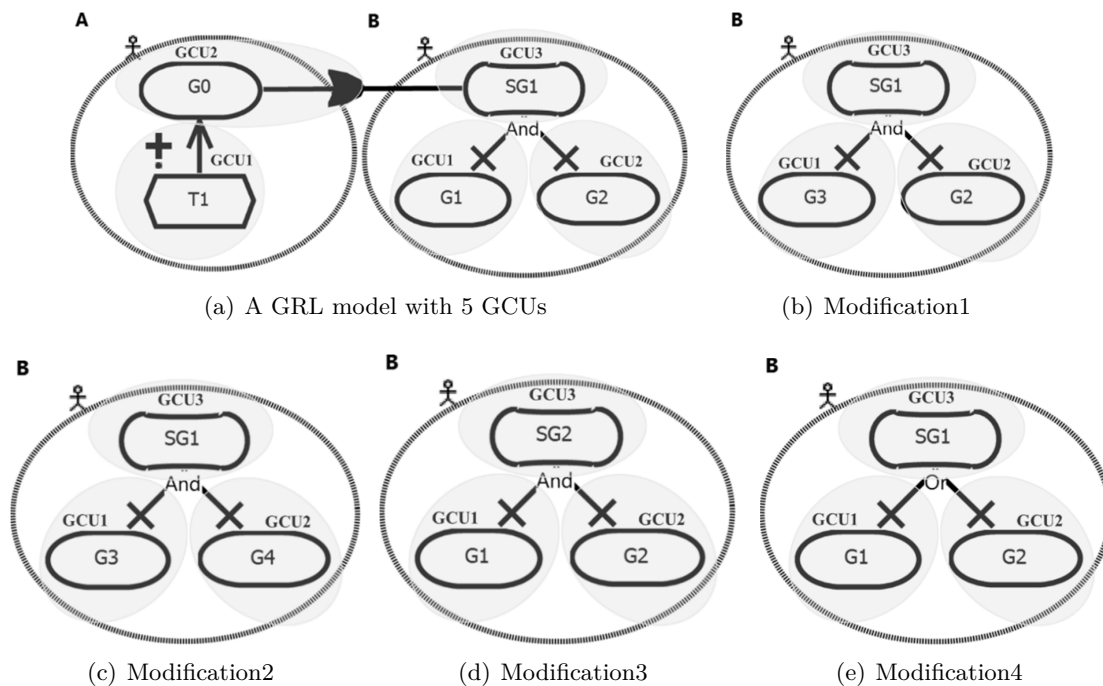


Figure 2. GRL Change Unit (GCU)

having the same meaning (semantic change), rearranging the text within an intentional element (although such a change seems to impact only the syntactic aspect, it may also impact the semantics if the text meaning changes). However, fixing typos in the text of an intentional element is not considered as a change. Furthermore, since the GCU represents the smallest change unit, we count for only one single change when either the intentional element changes or the outgoing link changes or both changes.

In what follows, we discuss and justify the GCU definition through some examples of possible changes. Assume that we perform the following changes to the GRL actor B of Figure 2a:

1. **Modification1:** Replace goal G1 by another goal G3 (see Figure 2b). Intuitively, it should be accounted as a single change, which is captured by the GCU definition as a change in GCU1 only (GCU2 and GCU3 remain unchanged). Indeed, since G1 is part of GCU1 only (within actor B), replacing G1 by G3 would only affect GCU1.
2. **Modification2:** Replace goal G1 by goal G3 and replace goal G2 by goal G4 (see Figure 2c). Intuitively, these two replacements should be

accounted as two changes, which is captured by the GCU definition as two changes in both GCU1 and GCU2, while GCU3 remains unchanged. Considering the AND-decomposition as a single unit (as opposed to our current definition of GCU) would not reflect the amount of applied change.

3. **Modification3:** Replace the softgoal SG1 by softgoal SG2 (see Figure 2d). Intuitively, it should be accounted as a single change, which is captured by the GCU definition as a change in GCU3 only (GCU1 and GCU2 remain unchanged). The enclosure of the target element into the GCU (as opposed to our current definition of enclosing source and link only), e.g., G1-AND-SG1 and G2-AND-SG1, would lead to double counting the change that impacts SG1.
4. **Modification4:** Replace the AND-decomposition by an OR-decomposition (see Figure 2e). This change has an impact on how goals G1 and G2 contribute to the achievement of SG1.

Hence, both links of the decomposition are modified from AND to OR, which is captured by the GCU definition as two changes in GCU1 and

GCU2 (GCU3 remains unchanged). Considering the AND decomposition as a single unit would not reflect the fact that many children have to contribute differently to their parent node.

GRL beliefs are connected to intentional elements through belief links, presenting no specific direction. We assume that a belief link terminates at a GRL belief. Since indicators are used only in converting real-world values into satisfaction levels, they are out of the scope of this research.

4.2. GRL actor stability metric

The objective of this paper is to propose a metric to measure GRL actor stability. We will follow a similar approach to the one by Alshayeb et al. [47]. We will reason about GCU as being the basic unit of change. We define possible changes between versions i and $i + 1$ as follows:

1. Added GCU means that the GCU was not present in GRL model version i and it has been added in version $i + 1$.
2. Deleted GCU means that the GCU was present in GRL model version i and has been deleted in version $i + 1$.
3. Changed GCU means that the GCU was present in GRL model version i and has been changed in version $i + 1$.
4. Unchanged GCU means that the GCU was present in GRL model version i and has neither been deleted nor changed in version $i + 1$.

All actors within a GRL model are tagged with a version number. When a change occurs

in any actor, the new GRL model is tagged with a different version number, even though the model may contain unchanged actor(s). Since we are measuring the stability of the GRL actor, we will measure the number of GCUs that have been unchanged.

We measure the stability between two consecutive versions, i.e. we measure the stability of version $i + 1$ with respect to its previous version i . Version $i + 1$ is considered fully stable when all GCUs in version i have not been changed or removed and is considered fully instable if all of its GCUs have been changed or removed as compared to version i .

To measure the GRL actor stability, we measure the stability of each GRL intentional element type for the actor and then sum the stabilities of all GRL Intentional element type for that actor. To calculate the stability of each GRL Intentional element type, we use the steps shown in Algorithm 1. Equations (1-5) show the metrics used to calculate each GRL actor intentional element stability.

If the intentional element has more than one outgoing link, each link is treated independently. The actor stability value ranges from 0 to 1, with 0 denoting completely instable and 1 denoting completely stable actor.

To calculate the GRL actor stability of version $i + 1$ with respect to version i , we then average the stabilities of all GRL actor intentional element stability as shown in equation 6.

$$\text{Goal Stability(GS)}_{i,i+1} = \frac{\text{number of unchanged Goals between model version } i \text{ and version } i + 1}{\text{number of Goals in model version } i} \quad (1)$$

$$\text{Softgoal Stability(SS)}_{i,i+1} = \frac{\text{number of unchanged Softgoals between model version } i \text{ and version } i + 1}{\text{number of Softgoals in model version } i} \quad (2)$$

$$\text{Task Stability(TS)}_{i,i+1} = \frac{\text{number of unchanged Tasks between model version } i \text{ and version } i + 1}{\text{number of Tasks in model version } i} \quad (3)$$

$$\text{Resource Stability(RS)}_{i,i+1} = \frac{\text{number of unchanged Resources between model version } i \text{ and version } i + 1}{\text{number of Resources in model version } i} \quad (4)$$

$$\text{Belief Stability(BS)}_{i,i+1} = \frac{\text{number of unchanged Beliefs between model version } i \text{ and version } i + 1}{\text{number of Beliefs in model version } i} \quad (5)$$

$$\text{Actor Stability}_{i,i+1} = \frac{GS + SS + TS + RS + BS}{\text{number of distinct types of intentional elements in model version } i} \quad (6)$$

Algorithm 1: Measuring GRL actor stability

Input : Actor A
Output : Stability of Actor A
Let T denote the set of distinct intentional element types {GS, SS, TS, RS, BS} within Actor A
foreach $t \in T$ **do**
 Compute *unchangedCount*;
 ▷ Count the number of unchanged GCUs (the number of unchanged occurrences of that Intentional element type between versions i and $i+1$)
 Compute *maximumPossibleChangeCount*;
 ▷ Count the maximum possible change of that GCU (the number of occurrences of that Intentional element that exists in version i)

$$extentOfChange[t]_{(i,i+1)} = \frac{UnchangedCount_{(i,i+1)}}{maximumPossibleChangeCount \text{ in model version } i};$$

end
▷ Compute Actor's A stability

$$ActorStability_{(i,i+1)} = \frac{\sum_t extentOfChange[t]_{(i,i+1)}}{\text{number of distinct intentional element types in model version } i};$$

return *ActorStability*;

Where 1 (at least one type) \leq number of distinct types of intentional elements ≤ 5 (goals, softgoals, tasks, resources and beliefs).

The proposed metric neither measures the percentage of change nor the number of changes, it rather measures the extent of change between two versions. The extent of change is measured by aggregating the individual stability values for all the intentional elements types. A GRL actor stability value of $x\%$ does not mean $x\%$ of the actor elements have unchanged, rather, it means that $x\%$ of the actor model structure remains unchanged.

To reduce the impact of having one change in simple actor that has few possible changes as compared to a single change for a complex actor, we do not just calculate the number of possible changes for all GCUs in the actor together; rather, we consider the actor to have 5 main intentional element types and we calculate the extent of change for each intentional element type separately. Thus, the change will be with respect to the GCUs for that particular intentional element type.

4.3. Measuring GRL actor stability example

In this section, we apply the proposed actor stability metric to a generic GRL example. Figure 3a

illustrates an AND decomposition within actor A. In Figure 3b, the AND decomposition is converted to an OR decomposition, the goal *Goal1* is changed to *Softgoal1* and *Task3* is deleted.

To calculate the extent of change, we first calculate the *maximumPossibleChangeCount* as proposed in [59]. The *maximumPossibleChangeCount* counts the maximum possible change that can happen to each property with respect to version i , thus it measures the number of GCUs for each property that exists in the model, as shown in Table 1. For example, in Figure 3a, we have two GCUs with goals as intentional elements: (1) *Goal1* with the *help* link and (2) *Goal2*. In addition, we have three GCUs with tasks as intentional elements: (1) *Task1* with the AND decomposition link, (2) *Task2* with the AND decomposition link and (3) *Task3* with the AND decomposition link. The remaining types of intentional elements (i.e., Softgoal, Resource, Belief) are not present in the model. Hence, their maximum possible change is zero.

Then, calculate the number of GCUs that have not been changed between the two versions as given in Table 2. In Figure 3b, the GCU composed of *Goal2* has not been changed, while the GCU composed of *Goal1* and the *help* contribution link has been changed,

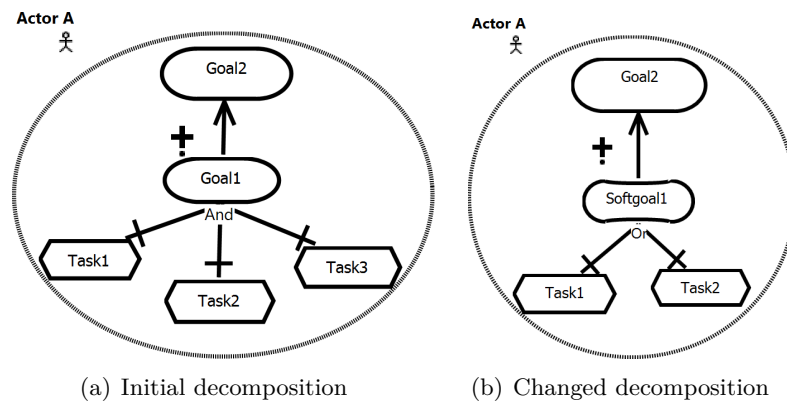


Figure 3. GRL example

Table 1. Calculate maximum-possible-change for each GCU' intentional element

GCU intentional element	Maximum possible change
Goals	2
Softgoals	0
Tasks	3
Resources	0
Beliefs	0

resulting into *Softgoal1* and a *help* contribution link. Hence, the number of unchanged GCUs with a goal as intentional element is 1. Since, the AND decomposition is changed to an OR decomposition, the three GCUs involving tasks as intentional elements are considered as changed. Consequently, their number of unchanged GCUs is 0.

Finally, calculate stability value for each property and the whole GRL actor stability as shown in Table 3. This is done by dividing number of unchanged GCU by maximum possible change for each property. The GRL actor stability is computed as the total stability divided by the number of distinct intentional element types involved in the model, since not all intentional element types might be present in the model. In this example, it is 2 (Goals and Tasks).

5. GRL actor stability metric validation

Theoretical and empirical validation of software metrics are usually performed before they can be used with confidence. In this section, we vali-

date the proposed GRL actors stability metric theoretically and we evaluate it empirically.

5.1. Theoretical validation

Theoretical validation refers to the process of certifying that the metric confirms to the principles of measurement theory. Different frameworks have been proposed to validate software metrics [60–63]. However, no framework has been found to specifically validate stability metrics. Therefore, we use Kitchenham et al. framework [64] which is a generic metric validation framework, to theoretically validate the proposed GRL actor stability metric. They proposed a framework for validating software measurement; the framework contains four properties the metric should satisfy to be theoretically valid. These properties are: (1) different entities must be distinguished from each other, (2) the valid measure must satisfy the representation condition, (3) units that contribute to the valid measure must be equivalent and (4) different entities can have the same attribute value. In addition to Kitchenham's et al. framework [64], we show the validation through an example. Figure 4 shows an

Table 2. Calculation of the number of Unchanged GCU' intentional elements

GCU intentional element	Number of unchanged GCUs
Goals	1
Softgoals	0
Tasks	0
Resources	0
Beliefs	0

Table 3. Stability calculation for each property and the overall actor stability

GCU intentional element	Maximum possible change	Number of unchanged GCUs	Stability
Goals	2	1	0.5
Softgoals	–	–	–
Tasks	3	0	0
Resources	–	–	–
Beliefs	–	–	–
Total stability = 0.5 + 0 = 0.5			
GRL actor stability = 0.5/2 = 0.25			

Note: “–” means the intentional element is not present in the actor.

example of four versions of a GRL actor enclosed elements, the base version (version 0) and three other versions (versions 1 to 3). The examples are used to demonstrate the validity of the proposed metric against Kitchenham's et al. theoretical validation properties. For simplicity, we only use goals in these examples; however, what applies to goals applies to the other intentional elements.

Property 1: “For an attribute to be measurable, it must allow different entities to be distinguished from one another” [64]. That is, different entities should have different measurement values, thus, the stability value for two actors will be different if they have different number of unchanged entities between the two versions.

To validate this property, consider the GRL models in Figure 4a, Figure 4b and Figure 4c. As compared to the base version of Figure 4a, in Figure 4b three goals remained unchanged (*Goal1*, *Goal2*, and *Goal4*) while in Figure 4c four goals remained unchanged. Since the number of unchanged goals in Figure 4b and Figure 4c with respect to the base version Figure 4a are different, stability values should also be different. This will always be true as the denominator will be the same when calculating the stability for both versions (number of goals in the base version)

while the numerator value (unchanged goals) will be different; therefore, the overall stability value will also be different. By calculating the stability value for these versions, we notice that the stability value for the GRL model shown in Figure 4b is 0.6 while the stability value for the GRL model shown in Figure 4c is 0.8. Thus, this property is shown to be true.

Property 2: “A valid measure must obey the Representation Condition” [64]. That is, if more entities have been unchanged between the two versions in two GRL actors, then the stability value of the GRL actor that has less unchanged entities should be higher.

To validate this property through the example, consider the same scenario used to prove property 1. The GRL model shown in Figure 4b has less unchanged properties as compared to the GRL model shown in Figure 4c; this is reflected in their stability values being 0.6 and 0.8 respectively. This property will always be true as the value in the denominator will be the same when calculating the stability for both versions while the numerator value will be different and when more unchanged properties exists, the numerator value will be higher and thus the overall stability is higher.

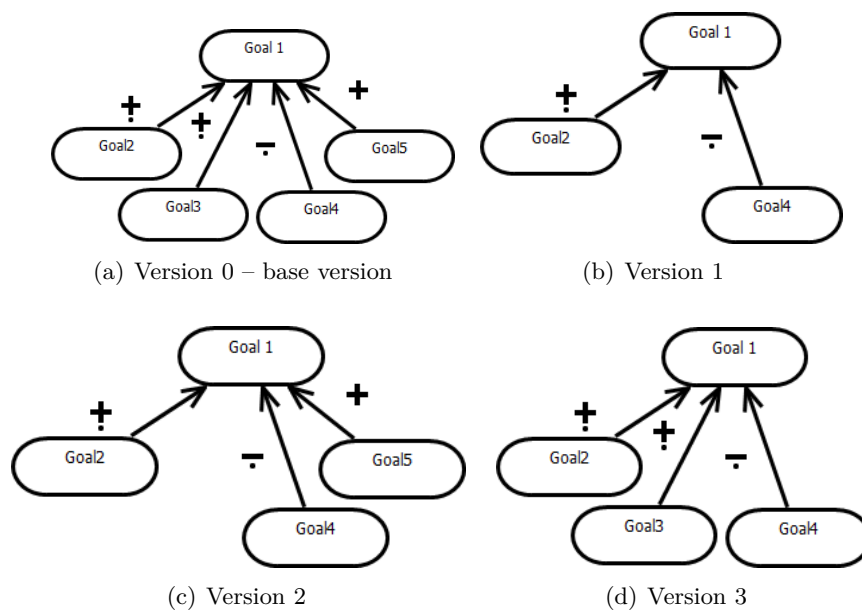


Figure 4. Different versions of GRL models

Property 3: “Each unit of an attribute contributing to a valid measure is equivalent” [64]. Consider Figure 4b and Figure 4c, in the same line as property 1 and property 2, the denominator will be the same when calculating the stability for both versions (five in the example shown in Figure 4). Each change in the goal will have the same weight since it has impact of 0.2 (which is $1/5$); thus, each change contributes by the same weight. In Figure 4b, three goals are unchanged, which makes its stability 0.6; there are four unchanged goals in Figure 4c, which makes its stability 0.8. Since the three goals still exist in Figure 4c and one more goal has unchanged, the stability of the model in Figure 4c should be equal to the value of actor stability in Figure 4b (0.6) plus the stability of the individual goal that has unchanged (0.2), which is shown to be true as the total stability of the model in Figure 4c is 0.8.

Property 4: “Different entities can have the same attribute value” [64]. That is, two GRL actors can have the same stability value if the same number of GCUs have unchanged when they have the same number of GCUs in each distinct intentional element.

Finally, to show the validity of property 4, consider the GRL models shown in Figure 4c

and Figure 4d. Figure 4c has four goals that are unchanged (goal 1, 2, 4 and 5), while Figure 4d has four unchanged goals (goal 1, 2, 3 and 4). We notice that in Figure 4c goal 3 has been deleted, while in Figure 4d goal 5 has been deleted as compared to the base version shown in Figure 4a. Therefore, the two GRL models are different, yet, they both have the same stability value (0.8) when compared to the base version (shown in Figure 4a). This is true because the denominator (number of goals in the base version) is the same in both cases and the numerator (number of unchanged goals) is also the same as the count of unchanged goals is equal regardless of which goals have changed. Therefore, the four properties proposed by Kitchenham et al. [64] are satisfied, thus, the proposed metric is theoretically valid.

5.2. Empirical validation

The empirical validation of a metric helps in assessing its usefulness and relevance. This section describes the experiments carried out to provide empirical evidence with respect to the usefulness and relevance of the proposed actor stability metric. This is achieved by following the templates and recommendations presented in Wohlin et al. [65].

5.2.1. Experiment goals

The main goal of our empirical study is to investigate the relationship between maintainability and the proposed stability metric. If such relationship is revealed by the experiment, then it can be shown that the proposed stability metric can be used as an indicator of the maintenance effort for the GRL actor model. Previous studies used time and effort to measure maintainability. Time is measured by the number of hours spent on maintenance activities [66, 67] and effort is measured by the number of lines added, deleted, or changed [68, 69]. The proposed metric is at the model level; thus, we measure maintainability effort using the time spent on performing the maintenance task. Since the proposed stability metric is measured between two consecutive versions ($\text{Stability}(i, i + 1)$), we measure the effort ($\text{Effort}(i, i + 1)$) to produce version $i + 1$ using version i as base version.

5.2.2. Experimental design

Empirical studies are conducted to test a theory to provide further evidence to support or reject it [70]. Since software stability is directly related to maintainability [4], we expect that a decrease in software stability will translate to more time spent on maintainability. To empirically validate the proposed metric, we designed and conducted a controlled experiment to test this assumption. In the experiment, we correlate the proposed metric values with the time spent on performing four maintenance tasks. We expect that the more stable an actor is, the less the time required for its maintenance will be. If such relation is observed, we can conclude that the proposed metric is empirically valid. Figure 5 illustrates the main steps of our experimental plan.

1. **Subjects.** Our subjects are 28 undergraduate software engineering students (randomly assigned to 7 groups of 4 members each) and 9 individual software engineering undergraduate students, enrolled in requirements engineering course. This would allow for more variability to gain more confidence in the experiment results. All participants received

around 9 hours of training on GRL including hands-on using the jUCMNav tool [55].

2. **Material.** The material given to the subjects consists of printouts of a GRL model that describes how to foster the relationship between a university and its alumni. Figure 5 shows the initial version of the designed GRL model that has been adapted from [71]. The four maintenance tasks to be executed on the GRL model are detailed in Section 5. To address the variability in the experiment, we considered four different actors with different sizes, performed the maintenance tasks on different actors and model constructs and applied all types of changes (modification, addition, and deletion).
3. **Variables:** We measure maintainability by means of the following dependent variables: (1) the time spent by the subjects in performing the four maintenance tasks (in seconds) and (2) the stability values of the four actors computed by the authors after each maintenance task. The independent variable is the performed maintenance tasks.
4. **Hypothesis:** The experiment was planned with the purpose of testing the following hypothesis:

Null hypothesis (H0): there is no correlation between actor stability and maintainability measured by time spend on performing the maintenance task.

Alternative hypothesis (H1): there is a correlation between actor stability and maintainability measured by time spend on performing the maintenance task.
5. **Experimental tasks:** The subjects were asked to conduct 4 corrective maintenance tasks. We have considered the following aspects when designing the maintenance tasks: (1) Tasks are small enough so they can be performed by students within a short period of time, (2) Maintenance tasks are not trivial and require careful analysis (to mimic real maintenance tasks), (3) Tasks are not too restrictive. Hence, more than one solution may be retained. The four maintenance tasks are as follows:

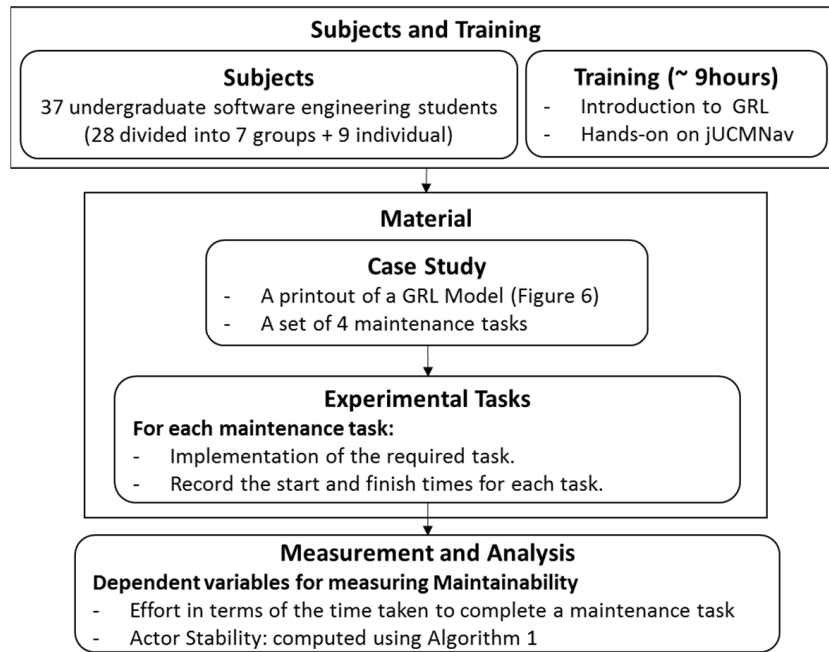


Figure 5. Experimental design

- **Maintenance Task 1:** The “Alumni Department” investigated ways to assess precisely how the department can serve alumni. It turned out that the goal “Serving alumni through University commitment” has no clear-cut satisfaction criteria. In addition, the alumnus found that the “Alumni Department” is breaching their privacy by sending too many SMSs. Please fix the GRL model to resolve these two issues.
- **Maintenance Task 2:** Please use the GRL model that you have already modified in maintenance Task 1. The university allocated funds to the alumni department have been reduced. The alumni department has to cope with this constraint by reducing their expenses without affecting the offered activities. Please modify the GRL model to implement these constraints.
- **Maintenance Task 3:** Please use the GRL model that you have already modified in maintenance Task 2. Each semester, the “Alumni Department” has been asking their alumnus to mentor an increasing number of undergraduate students,

in their respective fields. However, based on our undergraduate students’ feedback, this experience has many shortcomings and was not that positive. According to the alumni, mentoring a large number of students is not sustainable. Please modify the GRL model to reflect this fact.

- **Maintenance Task 4:** Please use the GRL model that you have already modified in maintenance Task 3. Alumni are willing to contribute to the university activities, but they are reluctant to donate money. Please modify the GRL model to reflect this fact.

It is worth noting that for tasks 2, 3, and 4, subjects were asked to not count the time taken to copy changes made in the previous task.

5.2.3. Experiment execution and data collection

In this section, we present samples of the executions of the maintenance tasks along with their corresponding actor stability computation. It is worth noting that some data was excluded as some subjects did not complete some/all tasks, did not produce correct responses for the required

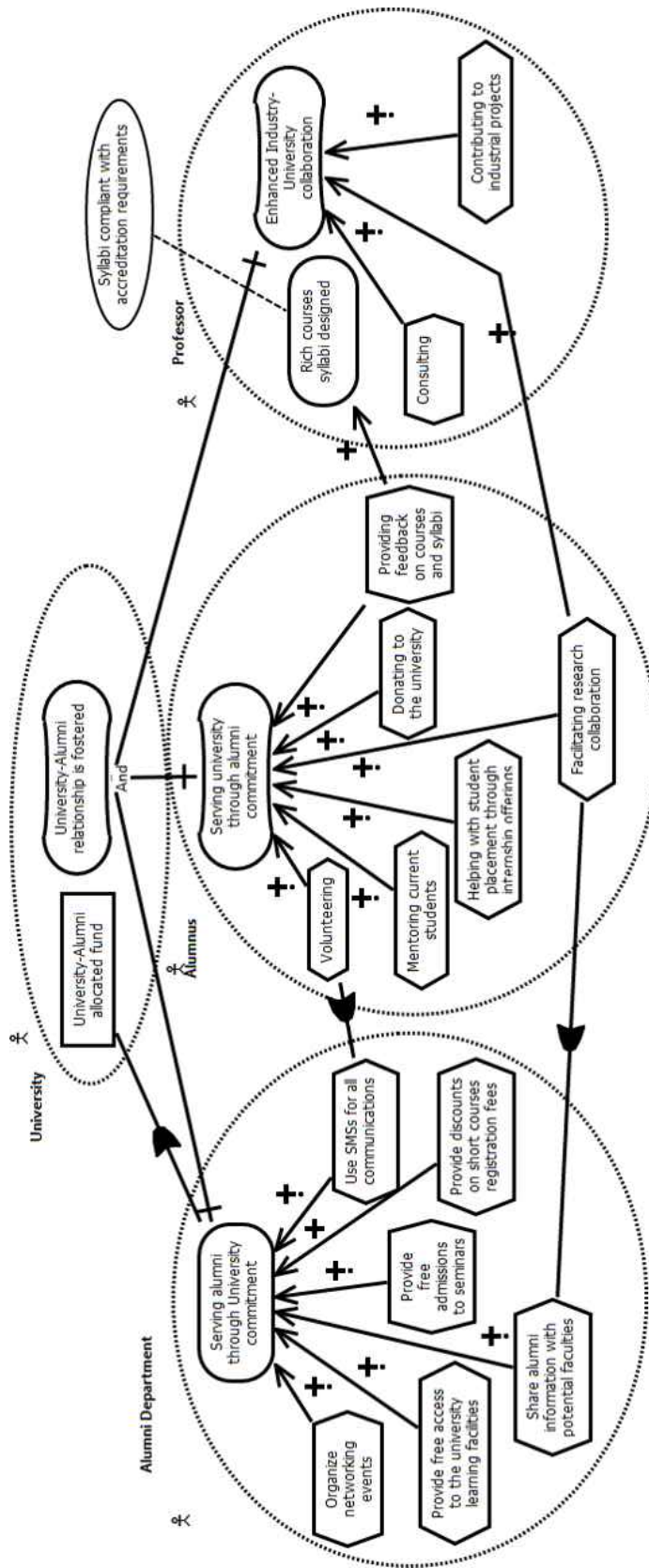


Figure 6: Initial GRL model describing the fostering of the university – alumni relationship

tasks, did not record the start and end time, and/or did provide an unrealistic time. Solutions containing minor syntactic errors or typos are retained as long as they make sense semantically. Furthermore, since actor stability is computed between two GRL consecutive versions, the output of the current task, is considered as the base to the next task. We discard the output of a task that results in an invalid GRL model. However, if a task output is syntactically valid but represents an incorrect solution then we consider it as a base model for the subsequent task since the tasks are independent.

Figure 7 illustrates an example of the resulting “Alumni Department” actor performed by one of the groups as part of maintenance Task 1 and Table 4 shows its corresponding stability computation. In order to address the first issue, the goal “Serving alumni through University commitment” is converted to a softgoal. To address the second issue, the contribution type between task “Use SMSs for all communications” and softgoal “Serving alumni through University commitment” is changed from “help” to “hurt”.

In the original model, the Alumni department has 8 GCUs (i.e., 6 GCUs with tasks and contributions, one GCU composed of a goal and a dependency and one GCU composed of one goal and one AND decomposition link). By converting the goal to a softgoal both GCUs involving the goal are considered as changed (i.e., number of unchanged goal GCUs is zero). Among the 6 GCUs involving tasks, only “use SMS for

all communications” task and the *help* contribution (converted to a *hurt*) has changed, i.e., the number of unchanged task GCUs is 5. The number of distinct types of intentional elements is equal to 2 for the Alumni Department (goals and tasks). Hence, the stability of the “Alumni Department” actor is 0.416.

Figure 8 illustrates an example of the resulting “Alumnus” actor performed by one of the groups as part of maintenance Task 3 and Table 5 shows its corresponding stability computation. To address the issue of mentoring a large number of students, the task “mentoring current students” is changed to “Mentoring a maximum of 2 students per year”. Another possible change would be to keep the task as is and change the contribution type from *help* to *hurt* (not shown in Figure 8).

In the original model, the Alumnus actor has 11 GCUs (i.e., 8 GCUs with tasks and contributions, 2 GCUs composed of a task and a dependency and one GCU composed of one softgoal and one AND decomposition link). Only one GCU, having a task as intentional element, is changed leaving the 9 task GCUs unchanged. The number of distinct types of intentional elements is equal to 2 for the Alumnus actor (softgoals and tasks). Hence, the stability of the Alumnus actor is 0.95.

The University and Professor actors are fully stable (i.e., stability equal to 1) since they have not been changed as part of the four maintenance tasks, while Alumnus and Alumni Department actors are partially stable.

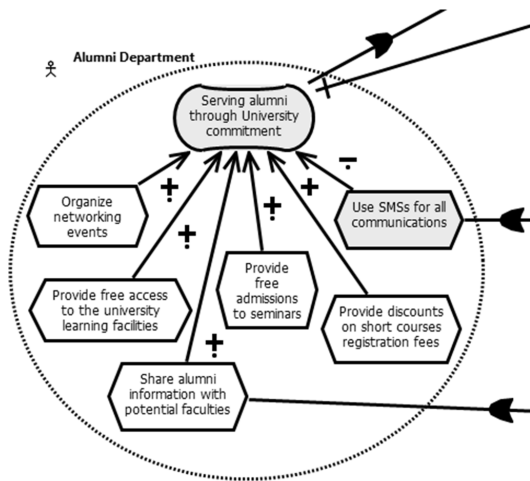


Figure 7. Example of maintenance Task 1 solution

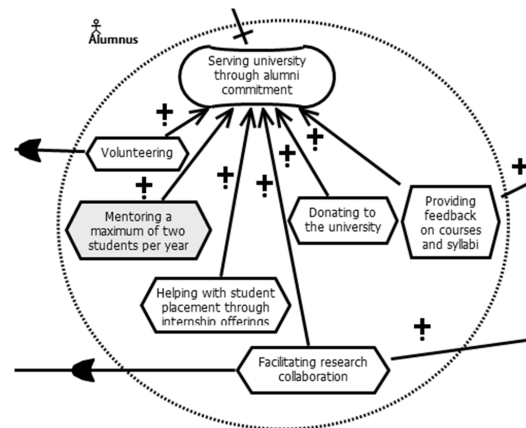


Figure 8. Example of maintenance Task 3 solution

Table 4. Maintenance Task 1 stability computation

Actor: Alumni department		
GCU intentional element	Maximum possible change	Number of unchanged GCUs
Goals	2	0
Softgoals	—	—
Tasks	6	5
Resources	—	—
Beliefs	—	—
ExtentOfChange: (0/2) + (5/6) = (5/6)		
Distinct types of intentional elements: 2		
GRL actor stability: (5/6) / 2 = 0.416		
Note: “—” means the intentional element is not present in the Actor.		

Table 5. Maintenance Task 3 stability computation

Actor: Alumnus		
GCU intentional element	Maximum possible change	Number of unchanged GCUs
Goals	—	0
Softgoals	1	1
Tasks	10	9
Resources	—	—
Beliefs	—	—
ExtentOfChange: (1/1) + (9/10) = (19/10)		
Distinct types of intentional elements: 2		
GRL actor stability: (19/10) / 2 = 0.95		
Note: “—” means the intentional element is not present in the Actor.		

5.2.4. Experimental analysis

To test the hypothesis, we performed Spearman correlation between the actor stability value and maintainability effort measured by the time spent on each task (in seconds). We considered each task as a different experiment and thus the data was combined to perform the analysis. Results of performing a maintenance task can vary between subjects as their solutions might be different. Therefore, the time recorded to perform the maintenance task can also vary by different subjects. However, the stability measurement of the solution can be equal. The results of the experiment, shown in Figure 9, show that the maintainability effort has a significant strong negative correlation as the correlation value is -0.713 and the P-value is <0.05 [72]. Thus, we reject the null hypothesis and accept the alter-

native hypothesis, that there is a correlation between actor stability and maintainability measured by time spend on performing the maintenance task.

The results confirm that there is a negative relationship between GRL actor stability and maintenance effort, hence, the less stable the actor, the more maintenance effort it requires. Therefore, actor stability value can be used as an indicator of maintenance effort. Requirements engineers need to give the GRL model special attention when the stability value is low as this will yield to high maintenance cost.

5.2.5. Threats to validity

The proposed metric and its empirical validation are subject to some limitations and threats to validity that we categorize as follows:

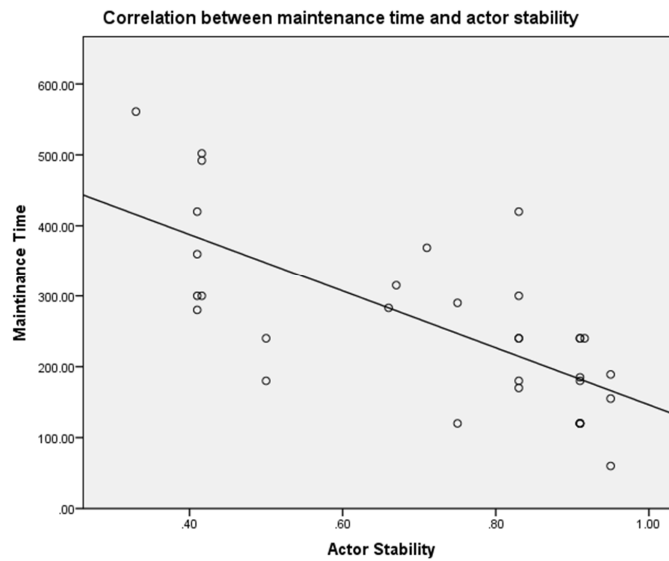


Figure 9. Correlation between maintenance time and actor stability

Conclusion validity: a possible threat is the small sample size used in the validation. More samples would have provided more confidence in the evaluation. Another possible conclusion threat is that we used a simple case study; this should not affect the validity of the results, as the metric will be measured in the same way regardless the systems' size. However, in our future work, our plans include conducting an experiment using bigger real-world case studies to further support our findings. A third possible conclusion threat is the reliability of the time measurement recorded by the subject. Although how to measure the time span was clarified to the subjects, variation in time measurement may have occurred. The last possible conclusion validity threat is that we correlated the actor stability values with maintainability effort measured by the time it takes to perform a task. However, we assumed that this relation exists based on the relationship identified by ISO 25010.

Internal validity: a possible internal threat is that some subjects did not perform the task correctly and thus produced a wrong GRL model that does not satisfy the task requirement. The data of such tasks was excluded from the empirical validation as they produced invalid GRL model. However, to mitigate this threat, we provided all subjects with similar training in GRL language including hands-on using the jUCMNav tool. Another possible internal threat is related to

the variables used in the empirical validation; we used the time spent by the subjects in performing the four maintenance tasks and the stability values of the actors as dependent variables. However, the selection of these variables is done based on the existing relationship reported in ISO 25010. ISO 25010 indicates that maintainability is related to modifiability which in turns is related to stability. Thus, we expect maintainability to be correlated with stability. Furthermore, there is a threat that the time for solving latter tasks might be affected by the learning time spent on earlier tasks. However, this practice effect is experienced by all participants as we followed the same sequence of tasks in all experiment tasks with all subjects.

Construct validity: a possible construct validity threat is that we consider all intentional elements to have the same weight regardless of the number of instances each element has; however, this is done intentionally to make sure that we treat all intentional elements equally as some intentional elements might be used more than others in GRL models. In fact, this is the reason why we did not consider a simpler metric that measures the number of unchanged GCUs divided by the total number of GCUs.

External validity: the subjects who performed the experiments are undergraduate software engineering students. The subjects executed

the experiment tasks and recorded the time taken to perform each task. This presents an external threat as the experiments were not performed by professionals, however, a recent study by Falessi et al. [73] shows that using students as subjects is acceptable and provides simplification of the actual context. Another possible external threat is related to the tasks used in the empirical evaluation as they are not industry tasks. To mitigate this threat, we planned and designed the maintenance tasks carefully so that they are not trivial and require careful analysis to mimic real maintenance tasks in addition to involving changes on different intentional elements.

6. Discussion

In the following subsections, we discuss the benefits of our proposed approach and how to interpret the metric.

6.1. Metric benefits

In early requirements engineering process, goal models are used to capture interests, intentions and strategies of different stakeholders. They go through many modifications that are necessary to accommodate changing user requirements, evolving business goals and objectives or even induced by changes in implementation technologies. The proposed GRL actor stability metric brings the following benefits:

1. It offers a systematic way to measure the extent of modifications across many versions of a goal model.
2. The computation of the metric is based on easily countable parameters, such as the number of unchanged GCUs, that does not require individual attention or time-consuming processing.
3. It allows for reasoning about which actor is less/more stable. In case, an actor represents a human stakeholder, an instable actor may be an indication that your stakeholders do not understand the problem they are trying to address, as they have changed their minds drastically. Some sort of visioning session with

them may be necessary. In addition, it allows for an early assessment of the risk of a major project reset as a result of several new stakeholder input.

4. The proposed metric takes into consideration all GRL constructs. We believe that the computation of the stability metric can be fully automated in this context.
5. It can be generalized to cover other goal-oriented languages, such as i^* [5], KAOS [7] and TROPOS [8]. Indeed, our approach is based on the notion of GCU, which is present in i^* Strategic Rationale (SR) diagrams. The KAOS approach covers goals of many types but is less concerned with the intentionality of actors. However, our stability metric may be applied at the goal model level to assess the extent of changes between different versions of the model. Similarly, we may tweak the metric to cover TROPOS and i^* Strategic Dependency (SD) models. Indeed, the relationships between actors and other constructs in i^* SD models and in TROPOS can be considered as a GCU, which is the basic concept in our proposed metric.

6.2. Metric practical implication

The proposed GRL actor stability metric can be used as a proxy of maintenance effort. Our empirical validation of the proposed metric has shown a direct negative relationship between actor stability and maintenance effort. Hence, requirements engineers may have an indicator of the maintenance effort required to maintain the GRL model. A low stability value indicates that the model will require more maintenance effort, therefore, the requirements engineers can make appropriate actions to refactor the current GRL model in order to reduce the expected maintenance effort.

7. Conclusion and future work

Requirements evolution is a main driver for systems evolution. Many metrics have been proposed to understand the sources, frequencies and types of requirements evolution. More specifically,

many metrics have been introduced to measure requirements stability at different abstraction and granularity levels. Goal models are used to capture interests, intentions and strategies of different stakeholders in early requirements engineering. In this paper, we presented a novel metric to measure GRL actor stability. The proposed metric provides a quantitative indicator of GRL actor maintainability to have a better estimation of the change cost. We have validated theoretically and empirically our proposed stability metric using a case study.

As a future work, we plan to automate and apply the proposed metric to real-world large-size case studies to assess whether our metric is a good indicator of the stability of GRL actors. We also plan to investigate which type of maintenance effort has the highest impact on stability. In addition, we plan to build prediction models to predict GRL actor stability. Furthermore, we plan to conduct an empirical experiment to study if stability measures converge over time and have a consistent trend. Moreover, we are currently working on proposing a metric suite for goal-oriented languages, which includes model stability. In this paper, we used time to measure maintainability, in future studies, we plan to use other measures such as effort.

Acknowledgment

The authors would like to acknowledge the support provided by King Fahd University of Petroleum & Minerals (KFUPM) for funding this work.

References

- [1] J.C. Chen and S.J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *Journal of Systems and Software*, Vol. 82, No. 6, 2009, pp. 981–992.
- [2] D. Galorath, "Software total ownership costs: development is only job one," *Software Tech News*, Vol. 11, No. 3, 2008.
- [3] J. Li, H. Zhang, L. Zhu, R. Jeffery, Q. Wang, and M. Li, "Preliminary results of a systematic review on requirements evolution," *IET Conference Proceedings*, 2012, pp. 12–21.
- [4] ISO/IEC, "25010:2011: Systems and software engineering – systems and software quality requirements and evaluation," 2011.
- [5] E.S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *International Symposium on Requirements Engineering*. IEEE, 1997, pp. 226–235.
- [6] L. Chung and J. Leite, *On Non-Functional Requirements in Software Engineering*. Springer-Verlag, 2009, pp. 363–379.
- [7] A. van Lamsweerde, "Requirements engineering: from craft to discipline," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 238–249.
- [8] P. Giorgini, J. Mylopoulos, and R. Sebastiani, "Goal-oriented requirements analysis and reasoning in the tropos methodology," *Engineering Applications of Artificial Intelligence*, Vol. 18, No. 2, 2005, pp. 159–171.
- [9] ITU-T, "Recommendation Z.151 (10/18), User Requirements Notation (URN) language definition, Geneva, Switzerland," Geneva, Switzerland, 2018. [Online]. <http://www.itu.int/rec/T-REC-Z.151/en>
- [10] S. Overbeek, U. Frank, and C. Köhling, "A language for multi-perspective goal modelling: Challenges, requirements and solutions," *Computer Standards & Interfaces*, Vol. 38, 2015, pp. 1–16. [Online]. <http://www.sciencedirect.com/science/article/pii/S0920548914000798>
- [11] A. Dias, V. Amaral, and J. Araujo, "Towards a domain specific language for a goal-oriented approach based on KAOS," in *Third International Conference on Research Challenges in Information Science*. IEEE, 2009, pp. 409–420.
- [12] D. Quartel, W. Engelsman, H. Jonkers, and M. van Sinderen, "A goal-oriented requirements modelling language for enterprise architecture," in *International Enterprise Distributed Object Computing Conference*. IEEE, 2009, pp. 3–13.
- [13] X. Franch and N. Maiden, "Modelling component dependencies to inform their selection," *COTS-Based Software Systems*, Vol. 2580 of LNCS, 2003, pp. 81–91.
- [14] X. Franch, G. Grau, and C. Quer, "A framework for the definition of metrics for actor-dependency models," in *12th International Requirements Engineering Conference*. IEEE, 2004, pp. 348–349.
- [15] X. Franch, "On the quantitative analysis of agent-oriented models," *Advanced Information*

- Systems Engineering*, Vol. 4001 of LNCS, 2006, pp. 495–509.
- [16] G. Grau, X. Franch, and N. Maiden, “Prim: An i*-based process reengineering method for information systems specification,” *Information and Software Technology*, Vol. 50, No. 1-2, 2008, pp. 76–100.
- [17] H. Kaiya, H. Horai, and M. Saeki, “Agora: attributed goal-oriented requirements analysis method,” *Joint International Conference on Requirements Engineering*, 2002, pp. 13–22.
- [18] G. Grau and X. Franch, “A goal-oriented approach for the generation and evaluation of alternative architectures,” in *European Conference on Software Architecture*. Springer, 2007, pp. 139–155.
- [19] A. van Lamsweerde, “Goal-oriented requirements engineering: A guided tour,” in *Proceedings fifth International Symposium on Requirements Engineering*. IEEE, 2001, pp. 249–262.
- [20] A.I. Antón, W.M. McCracken, and C. Potts, “Goal decomposition and scenario analysis in business process reengineering,” in *International Conference on Advanced Information Systems Engineering*. Springer, 1994, pp. 94–104.
- [21] C.M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, “Multi-objective reasoning with constrained goal models,” *Requirements Engineering*, 2016, pp. 1–37.
- [22] J. Horkoff, F.B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, J. Mylopoulos, and P. Giorgini, “Goal-oriented requirements engineering: A systematic literature map,” in *24th International Requirements Engineering Conference (RE)*. IEEE, 2016, pp. 106–115.
- [23] T. Ambreen, N. Ikram, M. Usman, and M. Niazi, “Empirical research in requirements engineering: trends and opportunities,” *Requirements Engineering*, 2016, pp. 1–33.
- [24] L. López, F.B. Aydemir, F. Dalpiaz, and J. Horkoff, “An empirical evaluation roadmap for iStar 2.0,” in *Proceedings of the Ninth International i* Workshop (istar’16)*, Vol. 1674, 2016, pp. 55–60.
- [25] M.A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, “Comparing goal-oriented approaches to model requirements for CSCW,” in *International Conference on Evaluation of Novel Approaches to Software Engineering*. Springer, 2011, pp. 169–184.
- [26] J.P. Carvallo and X. Franch, “On the use of i* for architecting hybrid systems: A method and an evaluation report,” in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2009, pp. 38–53.
- [27] G. Elahi, E. Yu, and M.C. Annosi, “Modeling knowledge transfer in a software maintenance organization – an experience report and critical analysis,” in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2008, pp. 15–29.
- [28] K. Hoesch-Klohe, *A Framework to support the Maintenance of Formal Goal Models*, PhD. Dissertation, University of Wollongong, 2013. [Online]. <http://ro.uow.edu.au/theses/4214>
- [29] N.A. Ernst, J. Mylopoulos, and Y. Wang, *Requirements Evolution and What (Research) to Do about It*. Berlin, Heidelberg: Springer, 2009, pp. 186–214.
- [30] A.K. Ghose, “Formal tools for managing inconsistency and change in RE,” in *Proceedings of the 10th International Workshop on Software Specification and Design*. IEEE Computer Society, 2000, p. 171.
- [31] N.A. Ernst, A. Borgida, J. Mylopoulos, and I.J. Jureta, *Agile Requirements Evolution via Paraconsistent Reasoning*. Berlin, Heidelberg: Springer, 2012, pp. 382–397.
- [32] A.M. Grubb and M. Chechik, “Looking into the crystal ball: requirements evolution over time,” in *24th International Requirements Engineering Conference (RE)*. IEEE, 2016, pp. 86–95.
- [33] Aprajita and G. Mussbacher, “TimedGRL: Specifying goal models over time,” in *24th International Requirements Engineering Conference Workshops (REW)*, 2016, pp. 125–134.
- [34] M.O. Elish and D. Rine, “Investigation of metrics for object-oriented design logical stability,” in *Seventh European Conference on Software Maintenance and Reengineering*. IEEE, 2003, pp. 193–200.
- [35] N.L. Soong, “A program stability measure,” in *Proceedings of the 1977 Annual Conference*. ACM, 1977, pp. 163–173.
- [36] M. Fayad, “Accomplishing software stability,” *Communications of the ACM*, Vol. 45, No. 1, 2002, pp. 111–115.
- [37] S.S. Yau and J.S. Collofello, “Some stability measures for software maintenance,” *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 6, 1980, pp. 545–552.
- [38] W. Li, L. Etzkorn, C. Davis, and J. Talburt, “An empirical study of object-oriented system evolution,” *Information and Software Technology*, Vol. 42, No. 6, 2000, pp. 373–381.

- [39] M. Alshayeb and W. Li, "An empirical study of system design instability metric and design evolution in an agile software process," *Journal of Systems and Software*, Vol. 74, No. 3, 2005, pp. 269–274.
- [40] A. AbuHassan and M. Alshayeb, "A metrics suite for UML model stability," *Software Systems Modeling*, Vol. 18, No. 1, 2019, pp. 557–583.
- [41] Y. Hassan, *Measuring software architectural stability using retrospective analysis*, Ph.D. dissertation, King Fahd University of Petroleum & Minerals, 2007.
- [42] J. Bansiya, "Evaluating framework architecture structural stability," *ACM Computing Surveys*, Vol. 32, No. 1, 2000.
- [43] M. Mattsson and J. Bosch, "Characterizing stability in evolving frameworks," in *Technology of Object-Oriented Languages and Systems*, 1999, pp. 118–130.
- [44] S.A. Tonu, A. Ashkan, and L. Tahvildari, "Evaluating architectural stability using a metric-based approach," in *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 2006, pp. 261–270.
- [45] D. Grosser, H.A. Sahraoui, and P. Valtchev, "An analogy-based approach for predicting design stability of Java classes," in *5th International Workshop on Enterprise Networking and Computing in Healthcare Industry*. IEEE, 2004, pp. 252–262.
- [46] D. Rapu, S. Ducasse, T. Gîrba, and R. Marinescu, "Using history information to improve design flaws detection," in *Eighth European Conference on Software Maintenance and Reengineering*. IEEE, 2004, pp. 223–232.
- [47] M. Alshayeb, M. Najji, M. Elish, and J. Al-Ghamdi, "Towards measuring object-oriented class stability," *Software, IET*, Vol. 5, No. 4, 2011, pp. 415–424.
- [48] R.C. Martin, "Large scale stability," *C++ Report*, Vol. 9, No. 2, 1997, pp. 54–60.
- [49] D. Grosser, H.A. Sahraoui, and P. Valtchev, "Predicting software stability using case-based reasoning," in *Proceedings 17th International Conference on Automated Software Engineering*. IEEE, 2002, pp. 295–298.
- [50] V. Basili, G. Caldiera, and H.D. Rombach, "The goal question metric approach," *Encyclopedia of Software Engineering*, 1994.
- [51] X. Franch, *A method for the definition of metrics over i* models*. Berlin Heidelberg: Springer, 2009, Vol. 5565, pp. 201–215.
- [52] P. Espada, M. Goulão, and J. Araújo, "A framework to evaluate complexity and completeness of KAOS goal models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2013, pp. 562–577.
- [53] C. Gralha, J. Araújo, and M. Goulão, "Metrics for measuring complexity and completeness for social goal models," *Information Systems*, Vol. 53, 2015, pp. 346–362.
- [54] J. Hassine and M. Alshayeb, "Measurement of actor external dependencies in GRL models," in *Proceedings of the Seventh International i* Workshop co-located with the 26th International Conference on Advanced Information Systems Engineering*, 2014. [Online]. <http://ceur-ws.org/Vol-1157/paper22.pdf>
- [55] *jUCMNav – Eclipse plugin for Use Case Maps*, University of Ottawa, Canada, 2016. [Online]. <http://softwareengineering.ca/jucmnav> Last Accessed Jan 2019.
- [56] W. Lam and V. Shankararaman, "Requirements change: A dissection of management issues," in *25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, Vol. 2. IEEE, 1999, pp. 244–251.
- [57] S. Anderson and M. Felici, "Quantitative aspects of requirements evolution," in *Proceedings 26th Annual International Computer Software and Applications*. IEEE, 2002, pp. 27–32.
- [58] G. Stark, A. Skillicorn, and R. Smeele, "A micro and macro based examination of the effects of requirements changes on aerospace software maintenance," in *Aerospace Conference*, Vol. 4. IEEE, 1998, pp. 165–172.
- [59] M. Mattsson and J. Bosch, "Stability assessment of evolving industrial object-oriented frameworks," *Journal of Software Maintenance: Research and Practice*, Vol. 12, No. 2, 2000, pp. 79–101.
- [60] L.C. Briand, J.W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," *Empirical Software Engineering*, Vol. 3, No. 1, 1998, pp. 65–117.
- [61] L. Briand, S. Morasca, and V. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, 1996, pp. 68–86.
- [62] E.J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, 1988, pp. 1357–1365.
- [63] G. Poels and G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures," *Information & Software Technology*, Vol. 42, No. 1, 2000, pp. 35–46.

- [64] B. Kitchenham, S.L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Transactions on Software Engineering*, Vol. 21, No. 12, 1995, pp. 929–944.
- [65] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.
- [66] A.B. Binkley and S.R. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures," in *Proceedings of the 20th International Conference on Software Engineering*. IEEE, 1998, pp. 452–455.
- [67] D. Darcy, C. Kemerer, S. Slaughter, and J. Tomayko, "The structural complexity of software: An experimental test," *IEEE Transactions on Software Engineering*, Vol. 31, No. 11, 2005, pp. 982–995.
- [68] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, Vol. 23, 1993, pp. 111–122.
- [69] M. Alshayeb and W. Li, "An empirical validation of object-oriented metrics in two iterative processes," *IEEE Transactions on Software Engineering*, Vol. 29, No. 11, 2003, pp. 1043–1049.
- [70] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*. London: CRC Press, 2014.
- [71] J. Hassine and D. Amyot, "A questionnaire-based survey methodology for systematically validating goal-oriented models," *Requirements Engineering*, Vol. 21, No. 2, 2016, pp. 285–308.
- [72] J. Evans, *Straightforward Statistics for the Behavioral Sciences*. Brooks/Cole Publishing, 1996.
- [73] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Software Engineering*, Vol. 23, No. 1, 2018, pp. 452–489.