Cloud-Based Software Engineering Framework

Ifeanyi Ubaechu, Michael Esiefarienrhe and Naison Gasela  
Department of Computer Science  
North West University  
Mafikeng, South Africa  
ifybech@yahoo.co.za, michael.esiefarienrhe@nwu.ac.za, naison.gasela@nwu.ac.za

Abstract—In today's computing world, cloud computing (CC) has redefined the consumption of information technology (IT) resources in more convenient and affordable ways and is gaining momentum. However, challenges emanating from pricing, architectural and legal aspect are not ruled out. Moreover, CC applications are developed without appropriate requirement engineering (RE) process. With the criticality of software and the cost of software failure in today's fast-paced technological and industrial world, it is important that appropriate RE processes are followed to create high quality CC software products. This is based on the fact that software bugs not found during the requirements phase can be very costly to find and fix at the later stage of development. Therefore, this paper examines seven software development process RE phases, to assess their suitability for CC development. We achieved this by employing a comparison framework from existing development methods. The objective is to develop a RE framework for CC to improve the quality of CC software products. Based on the analysis conducted, we found that the Rational Unified Process (RUP) is best method to deal with RE issues during CC application development.

Keywords— Cloud computing, software engineering, Software development methodology.

I. INTRODUCTION

The unprecedented IT needs today of storage, data processing elastic and limitless scale of computing infrastructure has led to a new computing paradigm – Cloud computing. In this cloud computing, massively scalable IT-enabled capabilities are delivered “as a service” to consumers of internet technologies [1]. Services provided here are readily available, relatively cheap and easy to use. Computer has undeniable become indispensable in our world today. The diversity and efficiency of programs manufactured to run on these computers cover a retinue of life activities. These programs or more technically put, software, basically define the functionality of the computer. Software is everywhere in computing. Thus the process of its development – software engineering, is of paramount importance.

Software engineering (SE) is an engineering discipline that is concerned with all aspects of software production, from the early stages of system specification through to maintaining the system after it has gone into use. SE has been defined as a discipline whose aim is the production of fault-free software, delivered on time and within budget that satisfies the client’s needs [3]. There are over a dozen known software development processes in existence by which software can be developed; each comprising of sequential stages of development. Examples are Waterfall model, V-shaped model, incremental, extreme programming, spiral, etc., just to name a few.

In an attempt to establish a suitable process model for CC development, this paper is designed to compare seven of the existing software development methodologies or models with emphasis on their RE phases. The aim is to identify one or more models which can be positioned for cloud-based application development or simply modified to adapt to cloud-based application development. Ultimately, we desire that the attractive benefits of CC be sustained by ensuring the software engineering activities such as RE are carefully performed.

The remainder of this research paper is structured in this manner: in section 2, we outline and discuss different requirement engineering processes. We present the comparison framework applied in this work in section 3, while applying the same framework in section 4 to critically compare the requirement models. Our conclusion of the paper is done in section 5; here also we discuss way-forward in future research work.

II. BACKGROUND INFORMATION

A. Cloud Computing

Cloud Computing is a model whereby IT resources of hardware, software and services are rented over the internet. Depending on the services that cloud users subscribe to, the cloud services can be divided into three platforms:

- IaaS: Cloud customers have access to hardware such as storage space, processors and network; also to virtual machines as well as operating systems.
- PaaS: this is an enabling platform environment for users to develop applications using API's. This can well be implemented and remotely operated.
- SaaS: under this cloud service platform, a (remote) third party provider makes software or applications available on demand.

Nevertheless, these services reach consumers in one of, or combination of either private or public cloud types. In a private cloud, just on enterprise has absolute control over the supplied infrastructure; whereas public cloud entails the extension of these over the supplied infrastructure; whereas public cloud entails the extension of these infrastructure to the general public by this enterprise. The combination of these two clouds is called the hybrid cloud. As the term implies, part of the cloud resources are owned and controlled by an organisation while renting some other resources from a third party.
B. Requirement-based Software Process Models

Waterfall model is the springboard of all other methods of software development; the oldest paradigm for software engineering with clearly spelt out steps in software application development [13]. The processes involved include requirement analysis, design, coding, testing and implementation, where by each successive phase is fully completed before the next one begins, without repetition. Waterfall is thus not suitable to be used for projects of dynamic nature [14]. Requirement gathering is most fundamental and crucial. It falls in the analytical stage of the model, divided into verification and validation [15]. Verification keeps the system in check with the model’s requirement standard, ascertaining the correctness of transition to the next phase. Validation on the other hand maintains that the user’s needs remain in focus.

The v-shaped model as the name connotes cuts the whole bulk of the software project progressively, and resizes it in steps, down to a simplified testable unit. This development model in some ways is similar to the waterfall model. Due to its sequential nature, each phase must be completed before the next one is kick-started. Testing, however, is a key component of this model. Testing is applied in parallel with corresponding phase of development to both verify and validate the product [12]. With the v shaped model, the project at hand undergoes about a dozen stages of course including requirement engineering, namely: outlining project objective, determining the functional and non-functional requirements, determining the risk acceptance, drafting project life cycle and system architecture, requirement quality [17]. Requirement gathering kick-starts the life cycle of the v shaped model just like the waterfall model. While functionality of the project is duly specified at this stage of development, a test plan is usually developed which focuses on satisfying these functionalities [16].

The incremental process model combines basic features from the waterfall model but in an iterative manner, and rather stretched out in progressive sequences [18]. Each fragment process is called an increment. However, unlike the waterfall process, a fragment or sequence in the incremental model requires less upfront requirements but yields a deliverable sequence facilitating the next phase of iteration [19]. Requirements in the incremental process are gathered and implemented in “dozes”, with regard to each increment. Generally, priority is given to the most required functionalities. The customer evaluates the system early to ascertain that prescribed functions are met. Otherwise, existing increment is changed or adjusted, while new requirements are defined for subsequent increments [17].

In extreme programming (XP), the customer is regarded as part of the team [20]. Extreme programming demands routing releases or increments based on customer requirements, but these increments must be daily. To ensure such regular increments, extreme programming adopts strict practices. For instance, two programmers work on a particular code at the same time. The one does the actual coding while the other proof-reads or cross-checks the codes. This is referred to as pair programming [20].

Rational Unified Process (RUP) is a development process focused on delivering high quality software within the prescribed schedule and budget, and that satisfies the end user. RUP is intrinsically a risk-driven process. It incorporates risk management in the development processes laying strong emphasis on carefully planned iterations. Another aspect of RUP is that it employs use-cases in expressing requirements on the functionality of the system. It is equally architecture-centric in its design activities. The use cases serve as prototypes for the main process [19]. RUP mainly divides into two disciplines, namely time and technical. With the time discipline, RUP lays emphasis roughly on steps to reaching the goal for a particular project. At the same time, there is refinement with regards to structured iterations. The technical dimension primarily divides into six disciplines which include requirement engineering as well as infrastructure disciplines with their respective workflow.

The requirement engineering of RUP seeks to achieve reliable specifications development and needful modifications to the system in question. It consists of the following activities: problem analysis, clear knowledge of end user’s needs, system management, requirement adjustment and fine-tuning the system definition [17].

However, none of the works done on RUP so far links it to the cloud. This is the main goal of this research work. Hence we strongly consider RUP development process as our choice software method for modification in its requirement engineering phase. We aim at eventually having a cloud-based requirement method.

The spiral model places more emphasis on risk analysis. Its success depends largely on efficient risk analysis, otherwise it is a lot like the incremental method [21]. Increments here are instead referred to as spirals. Each spiral constitutes of five development stages: Communication, Planning, Risk Analysis, Engineering and Evaluation [22], in a spiral formation. Subsequent phases build on the baseline phases; communication and planning. These initial phases provide fundamental requirements as well as plans for the application in question. First iteration may just result in the development of product specification, not necessarily the real thing. But subsequently, a prototype may follow, to more sophisticated versions of the software product. From the basic spiral, the project team starts off with small doze of requirements, going through the stages of development (excluding installation and maintenance). In subsequent spirals, the team adds functionality as for additional requirements; this continues until the application is ready for the installation and maintenance phases [12]. The spiral model is best applied when users are unsure of what they actually need; in essence, when requirements are complex and hazy. It is equally ideal for high-risk projects, projects needing significant changes or for new product line.

The volere model, in terms of RE, is relatively unique when compared to other software development methods. It ensures that every requirement is uniquely identified for the purpose of development, and at the same time distinguishes between requirement types. Volere also prescribes how requirements can be documented [23, 24]. In essence, this method presents a very comprehensive, structured requirement engineering template to its users. The volere technique structures requirements using the following approach: motivation (the driving force behind the project), limit of the project (in
III. RELATED WORKS

Schrodl [17] in their comparison work compared and contrasted between selected software development processes for requirements engineering in terms of their suitability to the specific requirements of cloud-based solutions. They came up with a classification system focused on cloud in order to develop a comparison framework for the requirements engineering models. However, a noteworthy limitation is that the number of models evaluated is just a small fraction of existing software developments. The framework employed for comparison is not quite encompassing, considering that cloud computing is component based. In as much as our research takes its cue from this work in terms of the comparison framework used, we go farther than just comparing four software development models; we compared seven. We applied similar framework. However, we isolate RUP arguing that its requirement engineering method is relatively more poised for the cloud. RUP is a modern software development process as well as product, one that is based on best practices [19]. It is designed and documented using UML (the Unified Modelling Language), and it is delivered online via Web technology [25].

Mishra [12] did a comparative study on some software development methods, namely waterfall, rad, spiral, v-shaped and incremental methods. According to [12], SDLC is a method by which software can be developed in a systematic manner and which will increase the probability of completing the software project within the time deadline while maintaining the quality of the software product as per the standard. The comparison yardsticks used in this work include requirement specification, cost, simplicity, risk involvement, expertise, flexibility to change, user involvement, flexibility, maintenance and duration. The choice of software development model is of utmost importance, depending on the scale of project at hand; the reason being that the software must be completed as well as delivered on or before the set time, with the desired quality. This work is intended to make for easy selection of an SDLC model, or a combination, for particular software project type. However, the yardsticks used for comparison in their work all stemmed from traditional software development methods. It is important to consider the trend of events in the software world; such things as reusability, and newer technologies as cloud computing.

In the research work [26] Munassar et al. aimed at creating a new way of software development based on certain already existing software development models. This novel method, Hybrid model was to meet the needs of different systems while eliminating the defects presented in previous models. Hybrid model combines the features of the following five development models: waterfall, iteration spiral, v-shaped and extreme programming. By combining five different software development processes, Hybrid model offers a novel model that integrates many good software development practices such as ease of use and application, needful risk analysis, testing, periodic iterations, identifiable phase deliverables, applicability in varying project sizes etc. This work is commendable, in that it galvanises multiple development models into one; harnessing each one’s strengths into an integrated unit. However, the number of development processes represented is very limited. Plus, the crucial importance of cloud computing to software engineering in general was totally left out.

Liu et al. [10] proposed to combine modelling with object oriented technologies for requirement engineering. Their model: Model-based Object-oriented Approach to Requirement Engineering (MORE) is to capture the domain knowledge of the requirement phase of software development, in order to cost effectively improve its completeness, consistency, traceability and reusability, as well as integrating it with artefacts of the other phases. The aim is to have software requirement development being model-driven as opposed to being document-centred. The proposed model is a blueprint which instantiates software requirement. It consists of targeted requirement content, corresponding attributes and operation behaviors. Software requirements are rather packaged in models, specifically MOORMs, which relate by association, generalization or importation, and are arranged in a hierarchy. There are nevertheless, various more software development methods or processes. This work does not pin-point a particular one that can adopt its proposed model.

IV. CHARACTERISTIC FOR ASSESSMENT IN CLOUD-BASED REQUIREMENTS ENGINEERING

TABLE I through 4 below summarizes the criteria for the comparison framework adopted in this research work.

<table>
<thead>
<tr>
<th>Name</th>
<th>Cloud Customer’s Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>In interviews, scenarios or workshops with the cloud provider, the customer unveils all expectations of the application in question. The customer’s said expectations which are interpreted as requirements are fine-tuned all together to reach a common ground; and hence documented in the form of essays, use-cases or style guides.</td>
</tr>
<tr>
<td>Example</td>
<td>Customer expresses all the desired attributes for the application to be developed, which become requirements.</td>
</tr>
<tr>
<td>When Used</td>
<td>These initial expectations form the core of the requirements for the application.</td>
</tr>
<tr>
<td>Advantage</td>
<td>1. There is a common ground between the user and provider in terms of expressed needs turned requirements according to SLA. 2. Documentation retains the original intent for the software; forms basis for subsequent changes.</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>Failure to well-interpret both well and ill expressed requirements of the customer results to full end product dissatisfaction and possibly costly changes.</td>
</tr>
</tbody>
</table>
Cloud Provider’s Persp
ible costly corrections
By this perspective, all stakeholders are
The advantages of adhering to this perspектив
This perspective applies to every phase of
This step if ignored may hamper the advantage
1. This perspective makes for a
On the other hand, the absence of the

A. Applying Comparison

The properties from the above development models are summarized in the table below. Although the comparison is not holistic, but from the scope that this research covers, RUP stands out from the sample of seven being researched in this work, as potentially more applicable as well as most effective.

a) RUP: RUP attempts to gather well-coordinated requirements by employing high level requirement model; at the same time ascertains them by the stakeholders [28]. It further considers applicable guidelines and constraints to analyze chosen requirements, as well as design implementable solution.

Unlike volere and Extreme Programming models, RUP is architecture-centric in design. It depends on resilient architecture for its conceptualization, construction, management and evolution [19]. In addition, RUP applies appropriate modeling techniques to explore usage, business rules, user interface and technical (non-functional) requirements. There is ample provision for carrying stakeholders along through and beyond the development process. After final delivery, further requirements bring about newer versions of the product. RUP engages proper change management program to both track and control these resulting changes [19, 29].

<table>
<thead>
<tr>
<th>Name</th>
<th>Cloud Provider’s Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>In view of the available cloud services and components, the cloud provider is to represent the customer’s wants into technical-readable requirements decode-able by the developers. Necessary adjustments must be made to accommodate both ill-expressed and important but unexpressed requirements, such as mode of security, ease of openness and other non-functional requirements. The cloud supplier equally makes provision for unforeseen requirements.</td>
</tr>
<tr>
<td>Example</td>
<td>View the consumer’s needs under the lenses of the applicable SLA.</td>
</tr>
<tr>
<td>When Used</td>
<td>This is applicable after the consumer’s initial requirements. It is important to view these early requirements though the cloud provider’s eyes.</td>
</tr>
<tr>
<td>Advantage</td>
<td>The advantages of adhering to this perspective when developing cloud application are as follows: 1. It brings clarity to the picture of application in the mind of the customers-said and unsaid, and places all stakeholders on common ground. 2. The availability of a change management system is vital for avoiding possible costly corrections in the long run.</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>This step if ignored may hamper the advantage of solidifying on the vital first meeting of the stakeholders (consumer and supplier).</td>
</tr>
</tbody>
</table>

TABLE III. SOLUTION ARCHITECTURE’S PERSP

<table>
<thead>
<tr>
<th>Name</th>
<th>The Solution Architecture Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Firstly in orchestration, the model’s capacity to draw requirements of the entire architecture of the system; very handy. Furthermore, in orchestration, a much needed model criteria is a summary layout of collective infrastructure of the system, especially in non-functional requirement areas of security and system quality.</td>
</tr>
<tr>
<td>Example</td>
<td>In requirement elicitation of project, the system architecture is projected at the same time.</td>
</tr>
<tr>
<td>When Used</td>
<td>This perspective applies after agreed-upon requirements by all stakeholders have been documented.</td>
</tr>
<tr>
<td>Advantage</td>
<td>1. This perspective makes for a seamless transition from documented requirement to actual implementation, in that it paints technical pictures, so to say, of the requirements in the mind of the developer.</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>On the other hand, the absence of the solution architecture perspective will impede the speed of implementation of already-drawn requirements.</td>
</tr>
</tbody>
</table>

TABLE IV. APPLICATIONS AND SAA SPERSPECTIVE

<table>
<thead>
<tr>
<th>Name</th>
<th>The Application and SaaS Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The stakeholders must keep in touch constantly and in good communication every step of the way. The customer is not at all exempted, but must be involved in every phase, challenges to do so not withstanding. Application components such as software and services are inter-dependent. The criteria of well integrated requirement engineering process which connects these components is vital. A working post-delivery change management system must be put in place to cater for modifications in the service area. The requirements for these changes must have a way to be monitored well.</td>
</tr>
<tr>
<td>Example</td>
<td>Make the customer take part in every stage of development. Adopt effective and traceable change management process.</td>
</tr>
<tr>
<td>When Used</td>
<td>This perspective applies to every phase of software development, including the post-delivery stage.</td>
</tr>
<tr>
<td>Advantage</td>
<td>By this perspective, all stakeholders are carried along simultaneously in the project, with change management both well implemented and documented; thus there is fluidity in both pre- and post-application delivery.</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>Without a proper change management and attendant requirement up-keep, discontinuity of the software application is inevitable.</td>
</tr>
</tbody>
</table>

The use of RUP as a software development model promises quite a few benefits relative to the other models. However, it is not cut out for optimum support for SaaS [17]. It is meant for software development by design. In terms of established principles of the various established elicitation techniques, the choice of method employed is crucial for proper traceability. RUP falls short in this aspect. It does not put the source of elicitation fully into consideration during requirement elicitation. Moreover, in the area of infrastructure requirement, RUP is not crystal about a well-structured elicitation as it is about iteration, change management or customer integration. And with this model, elicitation and validation of non-functional requirements is partial. They are only considered as supplementary [23].

b) V-Model: The v-model uniquely supports determining non-functional requirements; and being documentation-based, it ensures good requirement
documentation. It however falls on the contracting client to summarize the set of requirement specifications; thus

<table>
<thead>
<tr>
<th>Area</th>
<th>Characteristic</th>
<th>V-model</th>
<th>RUP</th>
<th>XP</th>
<th>Vol</th>
<th>Waterfall</th>
<th>Incremental</th>
<th>Spiral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Offering (Customer's Perspective)</td>
<td>Support with ascertaining requirements</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Support with analysing and agreeing requirements</td>
<td>n</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>m</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Validating requirements</td>
<td>M</td>
<td>M</td>
<td>n</td>
<td>M</td>
<td>M</td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>Taking into account of non-functional requirements</td>
<td>M</td>
<td>m</td>
<td>n</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>Management methods and change management</td>
<td>m</td>
<td>M</td>
<td>n</td>
<td>M</td>
<td>m</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Solution Architecture Perspective</td>
<td>Architectural Capability</td>
<td>M</td>
<td>M</td>
<td>n</td>
<td>n</td>
<td>M</td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>Structured elicitation of infrastructure requirements</td>
<td>M</td>
<td>m</td>
<td>n</td>
<td>m</td>
<td>n</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Application/SaaS Perspective</td>
<td>Coordinated and integrated RE for all single components of SaaS</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Better customer integration into the RE process</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>m</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Consideration of changes of requirements after/during delivery</td>
<td>n</td>
<td>M</td>
<td>m</td>
<td>m</td>
<td>n</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Consideration of the sources of requirements during elicitation</td>
<td>M</td>
<td>n</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>Consideration of the sources of requirements during change management</td>
<td>n</td>
<td>M</td>
<td>m</td>
<td>m</td>
<td>M</td>
<td>m</td>
<td>m</td>
</tr>
</tbody>
</table>

*M* = met  
**m** = partially met  
***n*** = not met
establishing the requirement appears to be a one-sided affair [17]. In the wake of this shortcoming, the model does not ensure adequate support in SaaS due to lack of extensive requirement coordination in the software and service areas. Furthermore, post-delivery change management is not applicable as it only takes place during the project development. The v-model also distances the customer from the development process. It limits the customer’s role in the document issue which marks the end of a particular phase.

c) Extreme programming: The XP model supports SaaS as well as customer’s participation in the development process [2]. A change management plan in place in this model only covers the duration of the development process. There is therefore no consideration of changes of requirements after delivery of the software product, only before. More so, at post delivery period, the source of XP requirement is only partially described while there is no definite change management plan or requirement validation. Once the customer is satisfied, the product can be given the green light. In general, this model is considered unfit for implementing requirement engineering for cloud computing because of its shortcoming in eliciting agile architecture requirements as well as a structured elicitation of requirements of infrastructure.

d) Volere: Volere is a typical requirement engineering development model by original design. It both provides templates used in specifying requirement structure and as well determines the techniques. However, it holds no support for SaaS, because the stipulated template or domain limits the support for coordinated and integrated requirement engineering system. The volere model equally has shortcomings in its architecture capacity as well as its requirement agility. Owing to active feedback feature after delivery, it provides support for a change management system, but only partially. Otherwise, it is efficient in areas such as post-delivery requirement management and documentation, and support of non-functional requirements.

e) Waterfall: Waterfall is a one-directional phased model whose phases do not overlap. It is restrictive to projects whose requirements – functional and nonfunctional, are definitely spelt out afore hand; unambiguous projects, with stable product definition, and whose technology is well understood [22, 30]. A strong point of waterfall model is that it completely establishes system and software requirements before other subsequent phases or steps. The system users are consulted to define in detail the system’s services, constraints and goals; with eventual precise documentation of these requirements [31]. Additionally, this model establishes an overall systemarchitecture.

V. CONCLUSION AND FUTURE WORKS

This paper aimed at consolidating on some already existing software development models, in view of the suitability of their requirement engineering phases for the cloud computing paradigm. We took our cue from [17], Requirement Engineering for Cloud Computing, whereby a comparison framework based on a broad literature review was developed. Applying this framework, development models could be relatively analyzed in a structured manner. It consists of four categories which cover twelve characteristics in all.

We went further to compare seven development models, as opposed to just four; namely: The V-model, Rational Unified Process (RUP), Extreme Programming (XP), Volere model; then additionally, Waterfall, Incremental and Spiral development models. From the results obtained, although it is evident that none of the seven models is perfect for cloud-based requirement engineering, yet we recommend RUP for requirement engineering in the cloud.

Based on the innate properties, RUP meets up to two-thirds of the framework criteria applied. These include support with ascertaining requirements, support with analyzing and agreeing requirements, validating requirements, change management, architectural capability, customer process integration, post-delivery changes as well as consideration of the source requirements during change management.

In conclusion, RUP comes closest to the whole objective relative to the other software engineering models. By the virtue of the framework comparison of software engineering models done in this work we pointed out the RUP to be relatively most accommodative of the cloud computing model, emphatically in its requirement engineering phase. Subsequently in future, we extend this work by proposing an RUP requirement engineering model by which cloud computing with its attendant technological advantages is harnessed into software engineering.

ACKNOWLEDGMENT

This research was supported by FRC and the Department of Computer Science at the NWU-Mafikeng and CSIR, South Africa.

REFERENCES


[16] P. Agarwal and M. Stan, "SOFTWARE MODELS."


