A Novel Taxonomy of Deployment Patterns for Cloud-hosted Applications: A Case Study of Global Software Development (GSD) Tools and Processes

Laud Charles Ochei, Andrei Petrovski

School of Computing Science and Digital Media
Robert Gordon University
Aberdeen, United Kingdom
Emails: {l.c.ochei,a.petrovski}@rgu.ac.uk

Julian M. Bass

School of Computing, Science and Engineering
University of Salford
Manchester, United Kingdom
Email: J.Bass@salford.ac.uk

Abstract—**Cloud patterns describe deployment and use of various cloud-hosted applications. There is little research that focuses on applying these patterns to cloud-hosted Global Software Development (GSD) tools. As a result, it is difficult to know the applicable deployment patterns, supporting technologies and trade-offs to consider for specific software development processes. This paper presents a taxonomy of deployment patterns for cloud-hosted applications. The taxonomy is composed of 24 sub-categories, which were systematically integrated and structured into 8 high-level categories. The taxonomy is applied to a selected set of software tools: JIRA, VersionOne, Hudson, Subversion and Bugzilla. The study confirms that most deployment patterns are related and cannot be fully implemented without being combined with others. The taxonomy revealed that (i) the functionality provided by most deployment patterns can often be accessed through an API or plugin integrated with the GSD tool, and (ii) RESTful web services and messaging are the dominant strategies used by GSD tools to maintain state and exchange information asynchronously, respectively. This paper also describes CLIP (CLoud-based Identification process for deployment Patterns), to guide software architects in selecting applicable cloud deployment patterns for GSD tools using the taxonomy and thereafter applies it to a motivating cloud deployment problem. Recommendations for guiding architects in selecting applicable deployment patterns for cloud deployment of GSD tools are also provided.**

Keywords*—*Taxonomy; Deployment Pattern; Cloud-hosted Application; GSD Tool; Plugin; Continuous Integration

I. INTRODUCTION

Collaboration tools that support Global Software Development (GSD) processes are increasingly being deployed on the cloud [1][2][3]. The architectures/patterns used to deploy these tools to the cloud are of great importance to software architects, because they determine whether or not the system's required quality attributes (e.g., performance) will be exhibited [4][5][6].

Collections of cloud patterns exist for describing the cloud, and how to deploy and use various cloud offerings [7][8]. However, there is little or no research in applying these patterns to describe the cloud-specific properties of applications in the software engineering domain (e.g., collaboration tools for GSD, hereafter referred to as GSD tools) and the trade-offs to consider during cloud deployment. This makes it very challenging to know the deployment patterns (together with the technologies) required for deploying GSD tools to the cloud to support specific software development processes (e.g., continuous integration (CI) of code files with Hudson).

Motivated by this problem, we propose a taxonomy of deployment patterns for cloud-hosted applications to help software architects in selecting applicable deployment patterns for deploying GSD tools to the cloud. The taxonomy will also help to reduce the time and risk associated with large-scale software development projects. We are inspired by the work of Fehling et al. [7], who catalogued a collection of patterns that will help architects to build and manage cloud applications. However, these patterns were not applied to any specific application domain, such as cloud-hosted GSD tools.

The research question this paper addresses is: **"How can we create and use a taxonomy for selecting applicable deployment patterns for cloud deployment of GSD tools."** It is becoming a common practice for distributed enterprises to hire cloud deployment architects or "application deployers" to deploy and manage cloud-hosted GSD tools [9]. For example, the CI systems used by Saleforce.com (a major player in the cloud computing industry), runs 150000 + test in parallel across many servers and if it fails it automatically opens a bug report for software architects and developers responsible for that *checkin* [10].

We created and applied the taxonomy against a selected set of GSD tools derived from an empirical study [11] of geographically distributed enterprise software development projects. The overarching result of the study is that most deployment patterns are related and have to be combined with others during implementation, for example, to address hybrid deployment scenarios, which usually involves integrating processes and data in multiple clouds.

This article is an extension of the previous work by Ochei et al. [1]. The main contributions of this article are:
1. Creating a novel taxonomy of deployment patterns for cloud-hosted applications.
2. Demonstrating the practicality of the taxonomy by: (i)

applying it to position a set of GSD tools; and (ii) comparing the different cloud deployment requirements of GSD tools.
3. Describing CLIP, a novel approach for guiding architects in selecting applicable cloud deployment patterns for GSD tools using the taxonomy, and thereafter applying it to a motivating cloud deployment problem.
4. Presenting recommendations and best practice guidelines for identifying applicable deployment patterns together with the technologies for supporting cloud deployment of GSD tools.

The rest of the paper is organized as follows: Section II gives an overview of the basic concepts related to deployment patterns for Cloud-hosted GSD tools. In Section III, we discuss the research methodology including taxonomy development, tools selection, application and validation. Section IV presents the findings of the study focusing on positioning a set of GSD tools within the taxonomy. In Section V, we discuss the lessons learned from applying the taxonomy. The recommendations and limitations of the study are in Sections VI and VII, respectively. Section VIII reports the conclusion and future work.

II. Deployment Patterns for Cloud-hosted GSD Tools

A. Global Software Development

In recent times, Global Software Development has emerged as the dominant methodology used is developing software for geographically distributed enterprises. The number of large scale geographically distributed enterprise software development projects involving Governments and large multi-national companies is on the increase [12][13][14].

Definition 1: **Global Software Development.** GSD is defined by Lanubile [15] as the splitting of the development of the same software product or service among globally distributed sites. Global Software Development involves several partners or sites of a company working together to reach a common goal, often to make a product (in this case, software) [15, 16].

In geographically distributed enterprise software development, there are not only software developers, but many stakeholders such as database administrators, test analysts, project managers, etc. Therefore, there is a need to have software tools that support collaboration and integration among members of the team involved in the software development project. As long as a software project involves more than one person, there has to be some form of collaboration [17] [16][11][18].

B. Cloud-hosted GSD Tools and Processes

Software tools used for Global Software Development projects are increasingly being moved to the cloud [3]. This is in response to the widespread adoption of Global Software Development practices and collaboration tools that support geographically distributed enterprises software projects [19]. This trend will continue because the cloud offers a flexible and scalable platform for hosting a broad range of software services including, APIs and developments tools [2][3].

Definition 2: **Cloud-hosted GSD Tool.** "Cloud-hosted GSD Tool" refers to collaboration tools used to support GSD processes in a cloud environment. We adopt the: (i) NIST Definition of Cloud Computing to define properties of cloud-hosted GSD tools; and (ii) ISO/IEC 12207 standard as a frame of reference for defining the scope of a GSD tool. Portillo et al. [20] identified three groups of GSD tools for supporting ISO/IEC 12207 processes:
(i) Tools to support Project Processes- These tools are used to support the management of the overall activities of the project. Examples of these processes include project planning, assessment and control of the various processes involved in the project. There are several GSD tools that fit into this group. For instance, JIRA and Bugzilla are software tools widely used in large software development projects issue and bug tracking.
(ii) Tools to support Implementation Processes such as requirements analysis and integration process. For example, Hudson, is a widely used tool for continuously integrating different source code builds and components into a single unit.
(iii) Tools for Support Processes - Software tools that fall into this group are used to support documentation management processes and configuration management processes involved in the software development project. For example, Subversion is a software tool used to track how the different versions of a software evolves over time.

These GSD tools, also referred to as Collaboration tools for GSD [20], are increasingly being deployed to the cloud for Global Software Development by large distributed enterprises. The work of Portillo et al. [20] presents the requirements and features of GSD tools and also categorizes various software tools used for collaboration and coordination in Global Software Development.

C. Architectures for Cloud-hosted Applications

Definition 3: **Architectural Pattern.** Architectural patterns are compositions of architectural elements that provide packaged strategies for solving recurring problems facing a system [5]. Architectural patterns can be broadly classified into 3 groups based on the nature of the architectural elements they use [5]:
(i) module type patterns - which show how systems are organized as a set of codes or data units in the form of classes, layers, or divisions of functionality.
(ii) component-and-connector (C&C) type patterns - which show how the system is organized as a set of components (i.e., runtime elements used as units of computation, filters, services, clients, servers etc.) and connectors (e.g., communication channels such as protocols, shared messages, pipes, etc.).
(iii) allocation patterns - which show how software elements (typically processes associated with C&C and modules) relate to non-software elements (e.g., CPUs, file system, networks etc.) in its environment. In other words, this pattern shows how the software elements are allocated to elements in one or more external environment on which the software is executed.

Architectural and design patterns have long been used to provide known solutions to a number of common problems facing a distributed system [5][21]. The architecture of a system/application determines whether or not its required quality attributes (e.g., performance, availability and security) will be exhibited [4][5].

D. Cloud Deployment Patterns

In cloud computing environment, a cloud pattern represents a well-defined format for describing a suitable solution to a cloud-related problem. Several cloud problems exist such as how to: (i) select a suitable type of cloud for hosting applications; (ii) select an approach for delivering a cloud service; (iii) deploy a multitenant application that guarantees isolation of tenants. Cloud deployment architects use cloud patterns as a reference guide that documents best practices on how design, build and deploy applications to the cloud.

Definition 4: **Cloud Deployment Pattern.** We define a "Cloud deployment pattern" as a type of architectural pattern, which embodies decisions as to how elements of the cloud application will be assigned to the cloud environment where the application is executed.

Our definition of cloud deployment pattern is similar to the concept of design patterns [21], (architectural) deployment patterns [5], collaboration architectures [4], cloud computing patterns [7], cloud architecture patterns [22], and cloud design patterns [8]. These concepts serve the same purpose in the cloud (as in many other distributed environments). For example, the generic architectural patterns- client-server, peer-to-peer, and hybrid [5] - relate to the following: (i) the 3 main collaboration architectures, i.e., centralized, replicated and hybrid [4]; and (ii) cloud deployment patterns, i.e., 2-tier, content distribution network and hybrid data [7].

One of the key responsibilities of a cloud deployment architect is to allocate elements of the cloud-application to the hardware processing (e.g., processor, files systems) and communication elements (e.g., protocols, message queues) on the cloud environment, so that the required quality attributes can be achieved.

Figure 2 shows how the elements of Hudson (a typical of GSD tool) is mapped to the elements of the cloud environment. Hudson runs on an Amazon EC2 instance while the data it generates is regularly extracted and archived on a separate cloud storage (e.g., Amazon S3).

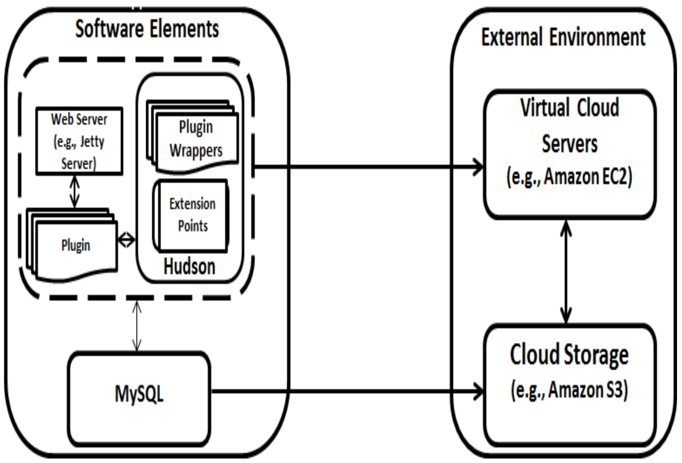
Software Elements
Web Server (e.g., Jetty Server)
Plugin Wrappers
Extension Points
Plugin
Hudson
MySQL
External Environment
Virtual Cloud Servers (e.g., Amazon EC2)
Cloud Storage (e.g., Amazon S3)

Fig. 1. Mapping elements of a GSD tool to External Environment

E. Taxonomy of Cloud Computing Patterns

What is a Taxonomy and its Purpose? The IEEE Software & Systems Engineering Standards Committee defines a Taxonomy as "a scheme that partitions a body of knowledge into taxonomic units and defines the relationship among these units. It aims for classifying and understanding the body of knowledge [23]." As understanding in the area of cloud patterns and cloud-hosted software tools for distributed enterprise software development evolves, important concepts and relationships between them emerge that warrant a structured representation of these concepts. Being able to communicate that knowledge provides the prospects to advance research [24].

Taxonomies and classifications facilitate systematic structuring of complex information. Taxonomies are mechanisms that can be used to structure, advance understanding and to communicate this knowledge [25]. According to Sjoberg [26], the development of taxonomies is crucial to documenting the theories that accumulate knowledge on software engineering. In software engineering, they are used for comparative studies involving tools and methods, for example, software evolution [27] and Global Software Engineering [28]. The work of Glass and Vessey [25] and Bourque and Dupuis [29] laid down the foundation for developing various taxonomies for software development methods and tools in software engineering. In this paper, we focus on using a taxonomy to structure cloud deployment patterns for cloud-hosted applications, in particular in the area of GSD tools.

Existing Taxonomies and Classifications of Deployment Patterns for Cloud-hosted Applications Several attempts have been made by researchers to create classifications of cloud patterns to build and deploy cloud-based applications. Wilder [22] describes eleven patterns: Horizontally Scaling Compute, Queue-Centric Workflow, Auto-Scaling, MapReduce, Database Sharding, Busy Signal, Node Failure, Colocate and Valet Key. The authors then illustrate how each pattern can be used to build cloud-native applications using the Page of Photos web application and Windows Azure. Each pattern is preceded by what the authors refer to as "primers" to provide a background of why the pattern is need. A description is provided about how each pattern is used to address specific architectural challenges that are likely to be encountered during cloud deployment.

A collection of over 75 patterns for building and managing a cloud-native application are provided by Fehling et al. [7]. The "known uses" of the implementation of each pattern is provided with examples of cloud providers offering products that exhibit the properties described in the pattern. This helps to further give a better understanding of the core properties of each pattern. We find the examples of known uses of patterns under "storage offering" category (e.g., blob storage, key-value storage) very useful in understanding how to modify a GSD tool in order to access a cloud storage. For example, Amazon S3 and Google cloud storage are products offered by Amazon and Google, respectively, for use as blob storage on the cloud. Blob storage is based on an object storage architecture, and so the GSD tool has to be modified to allow access using a REST API.

Homer et al. [8] describe: (i) twenty-four patterns that are useful in developing cloud-hosted applications; (ii) two primers and eight guidance topics that provide basic information and good practice techniques for developing cloud-hosted applications; and (iii) ten sample applications that illustrate how to implement the design patterns using features of Windows Azure. The sample code (written in C#) for these sample applications is provided, thus making it easy for architects who intend to use similar cloud patterns to convert the codes to other web programming languages (e.g., Java, Python) for use in other cloud platforms.

Moyer [30] discusses a collection of patterns under the following categories: image (e.g., prepackaged images), architecture (e.g., adapters), data (e.g, queuing, iterator), and clustering (e.g., n-tier) and then use a simple Weblog application written using Amazon Web Services (AWS) with Python to illustrate the use of these patterns. For example, one of the architectural patterns- Adapters, is similar to "Provider Adapter" pattern described by Fehling et al [7], which can be used for interacting with external systems not provided by the cloud provider. The weblog application uses a custom cloud-centric framework created by the author called *Marajo*, instead of contributing extensions to existing Python frameworks (e.g., pylons). Apart from Marajo's tight integration with AWS, it may be difficult for it to be widely used by software architects since it does not offer the rich ecosystem and large public appeal which other Python-based web frameworks currently offer.

Sawant and Shah discussed patterns for handling "Big Data" on the cloud [31]. These include patterns for big data ingestion, storage, access, discovery and visualization. For example, it describes how the "Federation Pattern" can be used to pull together data from multiple sources and then process the data. Doddavula et al. [32] present several cloud computing solution patterns for handling application and platform solutions. For instance, it discusses cloud deployment patterns for: (i) handling applications with highly variable workloads in public clouds; and (ii) handling workload spikes with cloud burst.

Erl et al. [33] present a catalogue of over 100 cloud design patterns for developing, maintaining and evolving cloud-hosted applications. The cloud patterns, which are divided into eight groups cover several aspects of cloud computing, such as scaling and elasticity, reliability and resiliency, data management, and network security and management. For example, patterns such as shared resources, workload distribution and dynamic scalability (which are listed under the "sharing, scaling and elasticity" category) are generally used for workload management and overall optimization of the cloud environment. The major strength of Erl et al.'s catalogue of cloud patterns is in its extensive coverage of techniques for handling the security challenges of cloud-hosted applications. It describes various strategies covering areas such as hypervisor attack vectors, threat mitigation and mobile device management. Other documentation of cloud deployment patterns can be found in [34][35][36][37][38][39].

Existing classifications of cloud patterns do not organize the individual patterns into a clean hierarchy or taxonomy. This is because most of the patterns tend to handle multiple architectural concerns [22]. This makes it difficult for an architect to decide whether the implementation of the cloud can be done by modifying the cloud-application itself or the components of the cloud environment where the application is running.

Cloud patterns in existing classifications are applied to simple web-based applications (e.g., Weblog application [30]) without considering the different application processes they support. Moreover, these patterns have not been applied against a set of applications in software engineering domain, such as cloud-hosted GSD tools. GSD tools may have similar architectural structure but they (i) support different software development processes, and (ii) impose varying workloads on the cloud infrastructure, which would influence the choice of a deployment pattern. For example, Hudson being a compiler/build tool, would consume more memory than subversion when exposed to high intensive workload.

Motivated by these shortcomings, we extend the current research by developing a taxonomy of deployment patterns for cloud-hosted applications that reflects the two main components of an architectural deployment structure: the cloud-application and cloud environment. Thereafter, we apply the taxonomy to position a set of GSD tools.

III. Methodology

A. Development of a Taxonomy of Deployment Patterns for Cloud-hosted Applications

1) Developing the Taxonomy: We develop the taxonomy by using a modified form of the approach used by Lilien [40] in his work for building a taxonomy of specialized ad hoc networks and systems for a given target application class. The approach is summarized in the following steps:

Step 1: Select the target class of Software Tool- The target class is based on the ISO/IEC 12207 taxonomy for the software life cycle process (see Definition 3 for details). The following class of tools are excluded: (i) tools not deployed in a cloud environment (even if they are deployed on a dedicated server to perform the same function); and (ii) general collaboration tools and development environments (e.g., MS Word, Eclipse).

Step 2: Determine the requirements for the Taxonomy- The first requirement is that the taxonomy should incorporate features that restricts it to GSD tools and Cloud Computing. In this case, we adopt the ISO/IEC 12207 framework [20] and NIST cloud computing definition [41]. Secondly, it should capture the components of an (architectural) deployment structure [5] - software elements (i.e., GSD tool to be deployed) and external environment (i.e., cloud environment). Therefore, our proposed taxonomy is a combination of two taxonomies - Taxonomy A, which relates to the components of the cloud environment [41], and Taxonomy B, which relates to the components of the cloud application architecture [7].

Step 3: Determine and prioritize the set of all acceptable categories and sub-categories of the Taxonomy- We prioritized the categories of the taxonomy to reflect the structure of a cloud stack from physical infrastructure to the software process of the deployed GSD tool. The categories and sub-categories of the two taxonomies are described as follows:

(1) Application Process: the sub-categories (i.e., project processes, implementation processes and support processes) represent patterns for handling the workload imposed on the cloud infrastructure by the ISO/IEC 12207 software processes supported by GSD tools [20]. For example, the unpredictable workload pattern described by Fehling et al. [7] can be used to handle random and sudden increase in the workload of an application or consumption rate the IT resources.

(2) Core cloud properties: the sub-categories (i.e., rapid elasticity, resource pooling and measured service) contain patterns used to mitigate the core cloud computing properties of the GSD tools [7].

(3) Service Model: the sub-categories reflect cloud service models- SaaS, PaaS, IaaS [41].

(4) Deployment Model: the sub-categories reflect cloud deployment models- private, community, public and hybrid [41].

(5) Application Architecture: the sub-categories represent the architectural components that support a cloud-application such as application components (e.g., presentation, processing, and data access), multitenancy, and integration. The multitenancy patterns are used to deploy a multitenant application to the cloud in such a way that guarantees varying degrees of isolation of the users. The three patterns that reflect these degrees of isolation are shared component, tenantisolated component and dedicated component [7].

(6) Cloud Offerings: the sub-categories reflect the major infrastructure cloud offerings that can be accessed- cloud environment, processing, storage and communication offering [7]. Patterns that fall under "communication patterns" are probably the best documented in this group. Examples include Priority Queue [8], Queue-Centric workflow, message-oriented middleware, which are used to ensure the reliability of messages exchanged between users.

(7) Cloud Management: contains patterns used to manage both the components and processes/runtime challenges) of GSD tools. The 2 sub-categories are - management components, which are used for managing hardware components (e.g., servers) and management processes, which are used for managing processes (e.g., database transactions) [7]. The node failure pattern described by Wilder [22] can be used to handle sudden hardware failures. The "Health Endpoint Monitoring" pattern [8] and the "resiliency management" pattern can be used to handle runtime failures or unexpected software failures.

(8) Composite Cloud: contains compound patterns (i.e., patterns that can be formed by combining other patterns or can be decomposed into separate components). The sub-categories are: decomposition style and hybrid cloud application [7]. The patterns under the decomposition style describe how the software and hardware elements of the cloud environment are composed (or can be decomposed) into separate components. A well-known example is the two-tier (or client/server) pattern, in which each component or process on the cloud environment is either a client or a server. Another example is the multisite deployment pattern [22], where users form clusters around multiple data centres or are located in globally distributed sites. Hybrid cloud application patterns are integrations of other patterns and environments. For example, the "hybrid development environment" pattern can be used to integrate various clouds patterns to handle different stages of software development- compilation, testing, and production.

Step 4: Determine the space of the Taxonomy- The selected categories and their associated sub-categories define the space of the taxonomy. The taxonomy (Table I) is composed of 24 sub-categories, which were systematically integrated and structured into 8 high-level categories. The information that the taxonomy conveys has been arranged into four columns: deployment components, main categories, sub-categories, and related patterns.

2) Description of the Taxonomy of Deployment Patterns for Cloud-hosted Applications: Table I shows the taxonomy captured in one piece. In the following, we describe the key sections of the taxonomy.

Deployment Components of the Taxonomy: There are two sections of the taxonomy: the upper-half represents Taxonomy A, which is based on NIST Cloud Computing Definition, while the lower-half represents Taxonomy B, which is based on the components of a typical cloud application architecture. The taxonomy has 24 sub-categories, which are structured into 8 high-level categories: four categories each for Taxonomy, A and B.

Hybrid Deployment Requirements: The thick lines (Table I) show the space occupied by patterns used for hybrid-deployment scenarios. There are two groups of hybrid-related patterns: one related to the cloud environment and the other related to the cloud-hosted application. For example, the hybrid cloud pattern (i.e., under "hybrid clouds" sub-category of Taxonomy A) is used to integrate different clouds into a homogenous environment while the hybrid data pattern (i.e., under "hybrid cloud applications" sub-category of Taxonomy B) is used to distribute the functionality of a data handling component among different clouds.

Examples of Related Patterns: Entries in the "Related Pattern" column show examples of patterns drawn from well-known collections of cloud patterns such as [7][8][22]. The cloud patterns found in these collections may have different names but they share the same underlying implementation principle. For example, message-oriented middle-ware pattern [7] is captured in Homer et al. [8] and Wilder [22] as a Queue-centric workflow pattern and competing consumers pattern, respectively.

B. GSD Tool Selection

We carried out an empirical study to find out: (1) the type of GSD tools used in large-scale distributed enterprise software development projects; and (2) what tasks they utilize the GSD tools for.

TABLE I. TAXONOMY OF DEPLOYMENT PATTERNS FOR CLOUD-HOSTED APPLICATIONS

Deployment Components	**Categories of Deployment Patterns**		**Related Patterns**
	Main Categories	*Sub-Categories*	
Cloud-hosted Environment (Taxonomy A)	Application Process	Project processes	Static workload
		Implementation processes	Continuously changing workload
		Support processes	Continuously changing workload
	Core Cloud Properties	Rapid Elasticity	Elastic platform,Autoscaling[22]
		Resource Pooling	Shared component, Private cloud
		Measured Service	Elastic Platform, Throttling[8]
	Cloud Service Model	Software resources	SaaS
		Platform resources	PaaS
		Infrastructure resources	IaaS
	Cloud Deployment Model	Private clouds	Private cloud
		Community clouds	Community cloud
		Public clouds	Public cloud
		Hybrid clouds	Hybrid cloud
	Composite Cloud Application	Hybrid cloud applications	Hybrid Processing, Hybrid Data,Multisite Deployment [22]
Cloud-hosted Application (Taxonomy B)		Decomposition style	2-tier/3-tier application, Content Delivery Network [22]
	Cloud Management	Management Processes	Update Transition Process, Scheduler Agent [8]
		Management Components	Elastic Manager, Provider Adapter, External Configuration Store [8]
	Cloud Offerings	Communication Offering	Virtual Networking, Message-Oriented Middleware
		Storage Offering	Block Storage, Database Sharding [22], Valet Key [8]
		Processing Offerings	Hypervisor, Map Reduce [22]
		Cloud Environment Offerings	Elastic Infrastructure, Elastic Platform, Runtime Reconfiguration [8]
	Cloud Application Architecture	Integration	Integration Provider, Restricted Data Access Component
		Multi-tenancy	Shared Component, Tenant-Isolated Component
		Application components	Stateless Component, User Interface Component

1) *Research Site:* The study involved 8 international companies, and interviews were conducted with 46 practitioners. The study was conducted between January, 2010 and May, 2012; and then updated between December, 2013 and April, 2014. The companies were selected from a population of large enterprises involved in both on-shore and off-shore software development projects. The companies had head offices in countries spread across three continents: Europe (UK), Asia (India), and North America (USA). Data collection involved document examination/reviews, site visits, and interviews. Further details of the data collection and data analysis procedure used in the empirical study can be seen in Bass [11].

2) *Derived Dataset of GSD Tools:* The selected set of GSD tools are: JIRA [42], VersionOne [43], Hudson [44], Subversion [45] and Bugzilla [46]. We selected these tools for two main reasons: (i) Practitioners confirmed the use of these tools in large scale geographically distributed enterprise software development projects [11]; (ii) The tools represent a mixture of open-source and commercial tools that support different software development processes; and are associated with stable developer communities (e.g., Mozilla Foundation) and publicly available records (e.g., developer's websites, whitepapers, manuals). Table II (another view of the one in [11]) shows the participating companies, projects and the GSD tools they used.

TABLE II. PARTICIPATING COMPANIES, SOFTWARE PROJECTS, SOFTWARE-SPECIFIC PROCESS AND GSD TOOLS USED

Companies	Projects	Software process	GSD tool
Company A, Bangalore	Web Mail Web Calendar	Issue tracking Code integration	JIRA Hudson
Company B, Bangalore	Web Mail Web Calendar	Issue tracking Version control	JIRA Subversion
Company H, Delhi	Customer service Airline	Agile tailoring Issue tracking	VersionOne JIRA
Company D, Bangalore (Offshore Provider to Company E)	Marketing CRM	version control Error tracking	Subversion Bugzilla
Company E, London	Banking Marketing CRM	Issue tracking Agile tailoring Code Building	JIRA VersionOne Hudson

JIRA: JIRA is a bug tracking, issue tracking and project management software tool. JIRA products (e.g., JIRA Agile, JIRA Capture) are available as a hosted solution through Atlassian OnDemand, which is a SaaS cloud offering. JIRA is built as a web application with support for plugin/API architecture that allows developers to integrate JIRA with third-party applications such as Eclipse, IntelliJ IDEA and Subversion [42].

Hudson: Hudson is a Continuous Integration (CI) tool, written in Java, for deployment in a cross-platform environment. Hudson is hosted partly as an Eclipse Foundation project and partly as a Java.NET project. It has a rich set of plugins making it easy to integrate with other software tools [47]. Organizations such as Apple and Oracle use Hudson for setting up production deployments and automating the management of cloud-based infrastructure [44].

VersionOne: VersionOne is an all-in one agile management tool built to support agile development methodologies such as Scrum, Kanban, Lean, and XP [43]. It has features that support the handling of vast amounts of reports and globally distributed teams in complex projects covering all aspects of teams, backlog and sprint planning. VersionOne can be deployed as a SaaS (on-demand) or On-Premises (local) [48].

Subversion: Subversion is a free, open source version control system used in managing files and directories, and the changes made to them over time [45]. Subversion implements a centralized repository architecture whereby a single central server hosts all project metadata. This facilitates distributed file sharing [19].

BugZilla: Bugzilla is a web-based general-purpose bug tracker and testing tool originally developed and used for the Mozilla project [46]. Several organizations use BugZilla as a bug tracking system for both open source(Apache, Linux, Open Office) and proprietary projects(NASA, IBM) [49].

C. Applying the Taxonomy

In this section, we demonstrate the practicality of the taxonomy in two ways: (1) Positioning the selected GSD tools against the taxonomy; and (2) Presenting a process for identifying applicable deployment patterns for cloud deployment of GSD tools. This framework may be used for other similar GSD tools not listed in our dataset.

1) Positioning GSD Tools on the Taxonomy: We demonstrate the practicality of the taxonomy by applying it to position a selected set of GSD tools. We used the collection of patterns from [7] as our reference point, and then complemented the process with patterns from [8][22].

The structure of the positioned deployment pattern, in its textual form, is specified as a string consisting of three sections-(i) Applicable deployment patterns; (ii) Technologies required to support such implementation; and (iii) Known uses of how the GSD tool (or one of its products) implements or supports the implementation of the pattern. In a more general sense, the string can be represented as: [PATTERN; TECHNOLOGY; KNOWN USE]. When more than one pattern or technology is applicable, we separate them with commas. Each sub-category of the taxonomy represents a unique class of reoccurring cloud deployment problem, while the applicable deployment pattern represents the solution.

2) How to Identify Applicable Deployment Patterns using the Taxonomy: Based on the experience gathered from positioning the selected GSD tools on the taxonomy, we describe CLIP (CLoud-based Identification process for deployment Patterns), a general process for guiding software architects in selecting applicable cloud deployment patterns for GSD tools using the taxonomy. The development of CLIP (shown in Figure 2 in Business Process Model and Notation (BPMN)) was inspired by IDAPO. Stol et al. [6] used IDAPO to describe a process for identifying architectural patterns embedded in the design of open-source software tools.

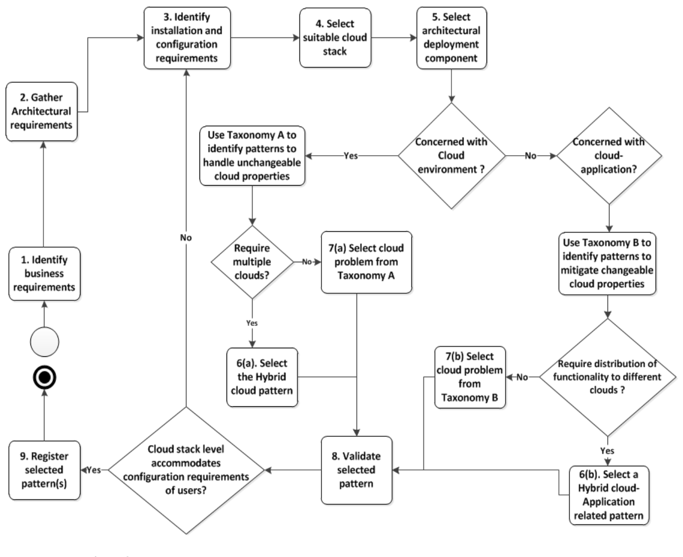
3. Identify installation and configuration requirements
4. Select suitable cloud stack
5. Select architectural deployment component
2. Gather Architectural requirements
Use Taxonomy A to identify patterns to handle unchangeable cloud properties
Yes
Concerned with Cloud environment ?
No
Concerned with cloud-application?
No
Require multiple clouds?
No
7(a) Select cloud problem from Taxonomy A
Use Taxonomy B to identify patterns to mitigate changeable cloud properties
1. Identify business requirements
Yes
6(a). Select the Hybrid cloud pattern
7(b) Select cloud problem from Taxonomy B
No
Require distribution of functionality to different clouds ?
Yes
9. Register selected pattern(s)
Yes
Cloud stack level accommodates configuration requirements of users?
8. Validate selected pattern
6(b). Select a Hybrid cloud-Application related pattern

Fig. 2. CLIP Framework for Identifying Deploying Patterns

The process of selecting the cloud deployment pattern(s) is an iterative process. The first step is to (1) **find out the main business requirements of the organization**. An example of a business requirement is fast feedback time, secure access to the shared component, and even the requirement of limited resource. The next step is to (2) **gather information about the architectural structure of the GSD tool**. We recommend the use of a IDAPO, a process designed by Stol el al[6] for identifying architectural patterns in an open-source software. At the end of that process, the architect would be able to identify among other things, the type of software and its domain, the technologies used, components and connectors, data source requirements (e.g., database type, data access method, file system etc.), and the default architectural pattern used in the design of the software.

After gathering information about the architectural structure of the GSD tool, the next step is to (3) **identify all the installation and configuration requirements of the GSD tool**. This information can be obtained directly from the documentation of the GSD tool or by creating a test application with the GSD tool. Based on the information gathered in the previous steps, the architect would be able to (4) **from the given cloud infrastructure, select a suitable level of the cloud-application stack that will accommodate all the installation and configuration requirements of the user**. If in doubt, we recommend that the architect should start with the first cloud stack level, which is the application level (i.e., GSD tool together with the software process it supports).

At this stage, the architect has to (5) **choose the architectural deployment component of interest**. In the cloud (as in other distributed environments), a cloud deployment pattern targets either the cloud environment or the cloud-application. If the architect is concerned with the cloud environment, then Taxonomy A should be used to select patterns for mapping business requirements to the unchangeable cloud properties, such as the location of the cloud infrastructure. However, if the architect is concerned with the cloud-hosted application, then Taxonomy B should be used to select deployment patterns for mitigating cloud properties, for example, performance and availability of the cloud-application.

The architect should then (6) **check for hybrid deployment requirements**. Usually, there are three main requirements that motivate the use of a hybrid-related cloud pattern. These include: (i) elasticity where there is need to increase or decrease the availability of cloud resources; (ii) accessibility; and (iii) combined assurance of privacy, security and trust [7]. For Taxonomy A, a typical requirement would be the need for integration of multiple clouds into a homogenous environment (e.g., using the hybrid cloud pattern), while that of Taxonomy B would be the need for distribution of the functionality/components of the GSD tool among different clouds (e.g., using the hybrid processing pattern). In either case, the respective hybrid related sub-category should be referenced to identify applicable patterns. otherwise the architect has to (7) **select a cloud deployment problem that corresponds to the sub-category of the chosen Taxonomy**. We have arranged the cloud deployment patterns into 8 high-level categories and 24 sub-categories that represent a recurring cloud deployment problem.

At this point, the process of selecting suitable deployment patterns involves referencing many sources of information several times. The architect can map the component/process of the GSD tool with the resources of the cloud infrastructure. We also recommend that the architect should revisit steps 1, 2, and 3. Assuming an architect wants Hudson to communicate with other external components/applications, then a better deployment pattern of choice would be Virtual Networking (via self service interface) to allow different users to be isolated from each other, to improve security and shield users form performance influence. However, if the communication is required internally to exchange messages between application components, then a message-oriented middleware would be the obvious choice.

After selection, the (8) **patterns have to be validated** to ensure that the chosen cloud stack level can accommodate all the installation and configuration requirements of the GSD tool. This can be done by mapping the components/process of the GSD tool identified from the previous steps to the available cloud resources. Another option would be to create a test application with the GSD tool to check if deploying to the cloud is workable. If validation fails, the architect may move one level lower in the cloud stack and repeat the process form step 4. Once confirmed, the (9) **selected pattern(s) (together with the use case that gave rise to the selection) should be registered** in a repository for later use by other architects.

D. Validation of the Taxonomy

We validate the taxonomy in theory by adopting the approach used by Smite et al. [28] to validate his proposed taxonomy for terminologies in global software engineering. A taxonomy can be validated with respect to completeness by benchmarking against existing classifications and demonstrating its utility to classify existing knowledge [28].

We have benchmarked Taxonomy A to existing classifications: the ISO/IEC 12207 taxonomy of software life cycle processes and the components of a cloud model based on NIST cloud computing definition, NIST SP 800-145. Taxonomy B is benchmarked to components of a cloud application architecture such as cloud offering and cloud management, as proposed by Fehling et al. [7]. The collection of patterns in [7] captures all the major components and processes required to support a typical cloud-based application, such as cloud management and integration.

We demonstrate the utility of our taxonomy by: (i) positioning the 5 selected GSD tools within the taxonomy; and (ii) applying CLIP to guide an architect in identifying applicable deployment patterns together with the supporting technologies for deploying GSD tools to the cloud. Tables III and IV show that several deployment patterns (chosen from 4 studies) can be placed in the sub-categories of our taxonomy. In Section III C, we describe CLIP and then demonstrate its practicality with a motivating cloud deployment problem.

E. Case Study: Selecting Applicable Patterns for Deploying Components for Automated Build Verification Process

In this section, we present a simple case study of a cloud deployment problem to illustrate how to use the process described in this paper (i.e., CLIP) given our taxonomy to guide in the selection of applicable pattern.

Motivating Problem: A cloud deployment architect intends to deploy a data-handling component to the cloud so that its functionality can be integrated into a cloud-hosted Continuous Integration System (e.g., Hudson). The laws and regulations of the company make it liable to archive builds of source code once every week and keep it accessible for auditing purposes. Access to the repository containing the archived source code shall be provided solely to certain groups of users. How can we deploy a single instance of this application to the cloud to serve multiple users, so that the performance and security of a particular user does not affect other users when there is a change in the workload?

Proposed Solution: In the following, we will go through the steps outlined in Section III C in order to select an appropriate cloud deployment pattern for handling the above cloud deployment problem.

Step 1: The key business requirements of this company are: (i) the shared repository that archives the source code cannot be shared; (ii) a single instance of this application should be deployed to the cloud to serve multiple users, and (iii) isolation among individual users should be guaranteed.

Step 2: Hudson is a web-based application and so it can easily be modified to support a 3-tier architectural pattern. An important component of this architectural pattern is the shared repository containing the archived data.

Step 3: Information obtained from Hudson documentation suggests that Hudson needs a fast and reliable communication channel to ensure that data is archived simultaneously between different environments/clouds.

Step 4: A review of the hardware and software requirements from Hudson documents suggests that having access to the application level and middle-level of the application stack will be sufficient to provide the configuration requirements for deploying and running Hudson on the given cloud infrastructure. A self-service interface can be provided as a PaaS (e.g., Amazon's Elastic Beanstalk) for configuring the hardware and software requirements of Hudson.

Step 5: The architectural deployment component of interest is the cloud-application itself, since the user has no direct access to the cloud IaaS. Therefore, the architect has to select a deployment pattern that can be implemented at the application level to handle the business requirements of the company. Based on this information, we turn to Taxonomy B, which contain cloud patterns used to mitigate cloud properties such as performance on the application level. Also, the fact that we are not attempting to integrate two cloud environments further strengthens the choice of our architectural deployment component of interest.

Step 6: After a careful review of the requirements, we conclude that a hybrid-related deployment pattern is the most suitable cloud deployment pattern for addressing the requirements of the customer. We assume that the data archived by Hudson contains the source code that drives a critical function of an application used by the company. Any unauthorized access to it can be disastrous to the company. The hybrid backup deployment pattern seems to be the most appropriate in this circumstance. This pattern can be used to extract and archive data to the cloud environment. Fehling et al. [7] discussed several types of hybrid -related patterns that can be used at the application level.

Step 7: As we have selected the hybrid backup pattern in the previous step, carrying out step 7 to select a deployment problem that corresponds to a particular sub-category of the taxonomy is no longer relevant. However, there are other patterns that can be selected from Taxonomy B for complementary purposes. For example, in a situation where the performance of the communication channel is an issue, the message-oriented pattern can be used to assure the reliability of messages sent from several users to access the component that is shared.

Step 8: The selected deployment pattern was validated by carefully reviewing its implementation to ensure that it can accommodate the user's configuration requirements and ultimately address the cloud deployment problem. We mapped Hudson and its supporting components to the available cloud resources.

In Figure 6, we show the architecture of the hybrid backup that is proposed for solving the cloud deployment problem. The architecture consists of two environments: one is a static environment that hosts Hudson and the other is an elastic cloud environment where the cloud storage offering (e.g., Amazon's S3) resides. This static environment represents the company's Local Area Network (LAN) that runs Hudson. During Hudson's configuration on the "Post-build Action" section, the location of the files to archive should point to the storage offering that resides on the cloud environment.

The cloud storage (accessed via a REST API) has to be configured in such a way that guarantees isolation among the different users. We assume that the data handling component is initially available as a shared component for all users. To ensure that the archived data is not shared by every user, the same instance of the shared component can be instantiated and deployed exclusively for a certain number of users.

From implementation standpoint, all user-id's associated with each request to Hudson are captured and those requests with exclusive access rights are then routed to the cloud storage. We discuss an approach named "COMITRE (Cloud-based approach to Multitenancy Isolation Through request RE-routing) in Ochei et al. [50] for deploying a single instance of Hudson to the cloud for serving multiple users (i.e., multi-tenancy) in such a way that guarantees different degrees of isolation among the users.

The different degrees of isolation between users accessing an application component that is shared is captured in three multitenancy patterns: shared component, tenant-isolated component and dedicated component [7].

Step 9: Finally, the cloud deployment scenario, the selected patterns together with the implemented architecture is documented for reference and reuse by other architects.

IV. Findings

In this section, we present the findings obtained by applying the taxonomy against a selected set of GSD tools: JIRA, VersionOne, Hudson, Subversion and Bugzilla. Refer to Section III- B for details of the processes supported by these tools.

A. Comparing the two Taxonomies

The cloud deployment patterns featured in Taxonomy A (i.e., upper part of Table I) relate to the cloud environment

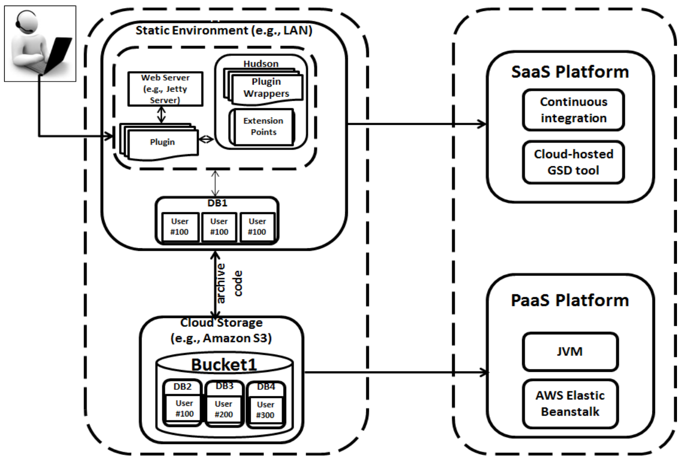

Fig. 3. Mapping Hudson to Cloud Stack based on Hybrid Backup pattern

hosting the application, while the cloud deployment patterns in Taxonomy B (i.e., lower part of Table I) relates to the cloud-hosted application itself. For example, the PaaS pattern is used to provide an execution environment to customers on the provider-supplied cloud environment. The Elastic platform pattern can be used in the form of a middleware integrated into a cloud-hosted application to provide an execution environment.

B. Hybrid-related deployment Patterns

Both taxonomies contain patterns for addressing hybrid deployment scenarios (i.e., the space demarcated with thick lines). For example, a hybrid cloud (Taxonomy A) integrates different clouds and static data centres to form a homogeneous hosting environment, while hybrid data (Taxonomy B) can be used in a scenario where data of varying sizes generated from a GSD tool resides in an elastic cloud and the remainder of the application resides in a static environment.

C. Patterns for Implementing Elasticity

We have observed that there are patterns that can be used by GSD tools to address rapid elasticity at all levels of the cloud stack. For example, an Elastic manager can be used at the application level to monitor the workload experienced by the GSD tool and its components (based on resource utilization, number of messages exchanged between the components, etc.) in order to determine how and when to provision or de-provision resources. Elastic platform and Elastic infrastructure can be used at the platform and infrastructure resources level, respectively.

D. Accessing Cloud Storage

The data handling components of most GSD tools are built on block storage architectures (e.g., relational databases such Oracle and MySQL used within Hudson and Bugzilla) for storing data, which are directly accessible by the operating system. However, a vast majority of storage offerings available on the cloud are based on object storage architecture. For example, Amazon S3, Google Cloud Storage and Windows Azure Blob provide cloud storage to cloud applications according to blob storage pattern [7]. Blob storage can be very useful for archiving large data elements (e.g., video, installers, and ISO images arising from Hudson builds and test jobs.

E. Positioning of GSD tools on the Taxonomy

Tables II and III show the findings obtained by positioning the cloud-hosted GSD tools on each sub-category of the taxonomy. In the following, we present a shortlist of these findings to show that we can identify applicable deployment patterns to address a wide variety of deployment problems.

(i) All the GSD tools considered in this study are based on web-based architecture. For example, Bugzilla and JIRA are designed as a web-based application, which allows for separation of the user interface, and processing layers from the database that stores details of bugs/issues being tracked.

(ii) All the GSD tools support API/Plugin architecture. For example, JIRA supports several APIs that allows it to be integrated with other GSD tools. The *Bugzilla:Web services*, a standard API for external programs to interact with Bugzilla, implies support for stateless pattern. These APIs represent known uses of how these deployment patterns are implemented.

(iii) Virtualization is a key supporting technology used in combination with other patterns to achieve elasticity at all levels of the cloud stack, particularly in ensuring fast provisioning and de-provisioning of infrastructure resources.

(iv) The GSD tools use Web services (through a REST API in patterns such as integration provider [7]) to hold external state information, while messaging technology (through message queues in patterns such as Queue-centric workflow [22] and Queue-based load leveling [8]) is used to exchange information asynchronously between GSD tools/components.

(vi) Newer commercial GSD tools (JIRA and VersionOne) are directly offered as SaaS on the public cloud. On the other hand, older open-source GSD tools (Hudson, Subversion and Bugzilla) are the preferred for private cloud deployment. They are also available on the public cloud, but by third party cloud providers.

We summarize our findings as follows: Although there are a few patterns that are mutually exclusive (e.g., stateless versus stateful components, and strict versus eventual consistency [7]), most patterns still have to be combined with others (e.g, combining PaaS with Elastic platform). These deployment patterns may also use similar technologies such as REST, messaging and virtualization to facilitate their implementation.

V. Discussion

The findings clearly suggest that by positioning a set of GSD tools on our proposed taxonomy, the purpose of the study has been achieved. The overarching result of the study is that most deployment patterns have to be combined with others during implementation. The findings presented here support previous research suggesting that most patterns are related and so two or more patterns can be used together [5][21].

A. Combining Related Deployment Patterns

Many deployment patterns are related and cannot be fully implemented without being combined with other ones, especially to address hybrid deployment scenarios. This scenario is very common in collaborative GSD projects, where a GSD tool either requires multiple cloud deployment environments or components, each with its own set of requirements. Our taxonomy, unlike others [22][8], clearly shows where to look for hybrid-related deployment patterns (i.e., the space demarcated by thick lines in Table I) to address this challenge. For example, when using Hudson there is usually a need to periodically extract the data it generates to store in an external storage during continuous integration of files. This implies the implementation of a hybrid data pattern. Hudson can be used in combination with other GSD tools, such as Subversion (for version control) and Bugzilla (for error tracking) within a particular software development project, each of which may also have their own deployment requirements.

B. GSD Tool Comparison

The taxonomy gives us a better understanding of various GSD tools and their cloud specific features. While other taxonomies and classifications use simple web applications [22] to exemplify their patterns, we use a mixture of commercial and open-source GSD tools. For example, commercial GSD tools (i.e., JIRA and VersionOne) are offered as a SaaS on the public cloud and also have a better chance of reflecting the essential cloud characteristic. Their development almost coincides with the emergence of cloud computing, allowing new features to be introduced into revised versions. The downside is that they offer less flexibility in terms of customization [52].

On the other hand, open-source GSD tools (i.e., Hudson, Subversion) are provided on the public cloud by third party providers and they rely on API/plugins to incorporate support for most cloud features. The downside is that many of the plugins available for integration are not maintained by the developer's community and so consumers use them at their own risk. The taxonomy also revealed that open-source GSD tools (e.g., Hudson, Subversion) are used at a later stage of a software life-cycle process in contrast to commercial tools, which are used at the early stages.

C. Support for API/Plugin Architecture

Another interesting feature of our taxonomy is that by positioning the selected GSD tools on it, it was discovered that the support for the implementation of most deployment patterns is practically achieved through API/Plugin integration [36]. This is no coincidence, as a typical cloud application is composed of various web-based related technologies such as web services, SOA and n-tier architectures. Therefore, a GSD tool with little or no support for APIs/Plugins is unlikely to attract interest from software developers. For example, JIRA's Elastic Bamboo support for Blob storage on Windows Azure is through an API [42]. JIRA has a plugin for integrating with Hudson, Subversion and Bugzilla [42] and vice versa.

D. Components and Connectors for maintaining state information and exchanging information asynchronously

Our taxonomy also highlights the technologies used to support the software processes of GSD tools, unlike others, which

TABLE III. POSITIONING GSD TOOLS ON THE PROPOSED TAXONOMY (TAXONOMY A)

Category	Sub-Category	*JIRA*	*VersionOne*	*Hudson*	*Subversion*	*Bugzilla*
Application Process	Project processes	**Static workload, Continuously changing workload**; *SaaS*; JIRA used by small no. of users, issues tracked reduces over time[42]	**Static workload**; *SaaS*; VersionOne is installed for a small number of users[43]	Process not supported	Process not supported	Process not supported
	Implementation processes	Process not supported	Process not supported	**Continuously changing workload**; *PaaS*; Hudson builds reduces gradually as project stabilizes[44]	Process not supported	Process not supported
	Support processes	Process not supported	Process not supported	Process not supported	**Static workload, Continuously changing workload**;*PaaS, Hypervisor*; rate of code files checked into Subversion repository is nearly constant or reduces over time[45]	**Continuously changing workload**; *PaaS,Hypervisor*; Errors tracked using Bugzilla reduces over time[46]
Core Cloud Properties	Rapid Elasticity	**Stateless pattern, Elastic platform**; *REST API*; JIRA is installed in cloud as SaaS[42]	**Stateless pattern, Elastic platform**; *REST API*; VersionOne is installed in cloud as SaaS[43]	**Elastic infrastructure, shared component**; *hypervisor*; Hudson server is supported by hypervisor in a private cloud[44]	**Elastic infrastructure, tenant-isolation component**; *hypervisor*; Subversion repository is supported by Elastic infrastructure[45]	**Stateless pattern**; *REST API*; Bugzilla is installed in cloud as SaaS in private cloud[46]
	Resource Pooling	**Hypervisor, Public Cloud,** ; *Virtualization*; JIRA deployed on the public cloud as SaaS[42]	**Hypervisor, Public cloud**; *Virtualization*; VersionOne deployed on public cloud as SaaS[43]	**Hypervisor, Tenant-isolated component**; *Virtualization*; Hudson is deployed on a hypervisor[44]	**Hypervisor, Tenant-isolated component**; *Virtualization*; Subversion is deployed on a hypervisor[42]	**Hypervisor, Public cloud**; *Virtualization*; Bugzilla deployed on the public cloud[46]
	Measured Service	**Static workload, Elastic Infrastructure,Throttling**[8]; *Virtualization*; Small number JIRA users generates a nearly constant workload[42]	**Static workload, Elastic Infrastructure,Throttling**[8]; *Virtualization*; Small number of VersionOne users generates small workload[43]	**Static workload, Elastic Infrastructure,Throttling**[8]; *Virtualization*; Hudson can be supported on public cloud by elastic infrastructure[44]	**Static workload, Elastic Infrastructure,Throttling**[8]; *Virtualization*; Subversion can be supported on public cloud by elastic infrastructure[45]	**Static workload, Elastic Infrastructure,Throttling**[8]; *Virtualization*; Bugzilla can be supported on third party public cloud by elastic infrastructure[46]
Cloud Service Model	Software resources	**SaaS**; *Web Services, REST*; JIRA OnDemand[42]	**SaaS**; *Web Services, REST*; VersionOne OnDemand[43]	**SaaS**; *Web Services, REST*; Hudson is offered by 3^{rd} party cloud providers like CollabNet[51]	**SaaS**; *Web Services, REST*; Subversion is offered by 3^{rd} party cloud providers like CollabNet[51]	**SaaS**; *Web Services, REST*; Bugzilla is offered by 3^{rd} party cloud providers like CollabNet[51]
	Platform resources	**PaaS**; *Elastic platform, Message Queuing*; JIRA Elastic Bamboo[42]	**PaaS**; *Elastic platform, Message Queuing*; No known use	**PaaS**; *Elastic platform, Message Queuing*; Build Doctor and Amazon EC2 for Hudson	**PaaS**; *Elastic platform, Message Queuing*; Flow Engine powered by Jelastic for Subversion	**PaaS**; *Elastic platform, Message Queuing*; No known use
	Infrastructure resources	Not applicable	Not applicable	**IaaS**; *Hypervisor*; Hudson is a distributed execution system comprising master/slave servers[44]	**IaaS**; *Hypervisor*; Subversion can be deployed on a hypervisor	Not applicable
Cloud Deployment Model	Private usage	**Private cloud**; *Hypervisor*; JIRA can be deployed on private cloud using private cloud software like OpenStack	**Private cloud**; *Hypervisor*; VersionOne On-premises[43]	**Private cloud**; *Hypervisor*; Hudson can be deployed on private cloud using private cloud software	**Private cloud**; *Hypervisor*; Subversion can be deployed on private cloud using private cloud software	**Private cloud**; *Hypervisor*; Bugzilla can be deployed on private cloud using private cloud software
	Community usage	**Community cloud**; *SaaS*; Bugzilla can be deployed on private cloud	**Community cloud**; *SaaS*; Bugzilla can be deployed on community cloud	**Community cloud**; *SaaS,Paas, IaaS*; Bugzilla can be deployed on community cloud	**Community cloud**; *SaaS,IaaS*; Bugzilla can be deployed on community cloud	**Community cloud**; *SaaS, PaaS*; Bugzilla can be deployed on community cloud
	Public usage	**Public cloud**; *SaaS*; JIRA OnDemand is hosted on public cloud[42]	**Public cloud**; *SaaS*; VersionOne is hosted on public cloud[43]	**Public cloud**; *SaaS,PaaS,IaaS*; Hudson is hosted on public cloud(via 3^{rd} party providers)[51]	**Public cloud**; *SaaS, IaaS*; Subversion is hosted on public cloud(via 3^{rd} party providers)[51]	**Public cloud**; *SaaS, PaaS*; Bugzilla is hosted on public cloud(via 3^{rd} party providers)[51]
	Hybrid usage	**Hybrid cloud**; *SaaS*; JIRA used to track issues on multiple clouds	**Hybrid cloud**; *SaaS*; Agile projects are stored in different clouds[45]	**Hybrid cloud**; *SaaS,PaaS, IaaS*; Hudson builds done in separate cloud	**Hybrid cloud**; *SaaS, IaaS*; Subversion repository resides in multiple clouds	**Hybrid cloud**; *SaaS, PaaS*;Bugzilla DB can be stored in different clouds

focus mostly on the design of cloud-native applications [7]. Web services (via REST) and messaging (via message queues) are the preferred technologies used by cloud deployment patterns (e.g., stateless pattern, message-oriented middleware) to interconnect GSD tools and other components. REST style is favoured by public cloud platforms. For example, JIRA's support for SOAP and XML-RPC is depreciated in favour of REST [42]. This trend is also reported in [22][36].

E. Accessing Data stored in Cloud Storage

Some GSD tools (e.g., Subversion) handle large amounts of data (images, music, video, documents/log files) depending on the nature of the software development project. This data can be stored on a cloud storage to take advantage of its ability to scale almost infinitely and store large volumes of unstructured data. The downside is that the application code of the GSD tool has to be modified to enable direct HTTP-based REST API calls. Cloud storage's object architecture requires REST API to be either integrated as a plugin into the GSD tool or coded separately. Storing data on the cloud is invaluable in a case where the GSD tool runs on a static environment and the data it generates is to be archived on an elastic cloud.

F. Patterns for Cloud-application Versus Cloud-environment

Our taxonomy can be used to guide an architect in focusing on a particular architectural deployment component of interest - that is, either a cloud-hosted application or cloud-hosted environment. Other taxonomies [8][22] are concerned with

TABLE IV. POSITIONING GSD TOOLS ON THE PROPOSED TAXONOMY (TAXONOMY B).

Category	**Sub-Category**	*JIRA*	*VersionOne*	*Hudson*	*Subversion*	*Bugzilla*
Application Architecture	Application Components	**User interface component,Stateless**; *REST API, AJAX*; State information in JIRA thru REST API[42]	**User-interface component,Stateless**; *jQuery AJAX, REST/Web Service*; VersionOne REST API[43]	**User-interface component,Stateless**; *REST API, AJAX*; Hudson Dashboard pages via REST[44]	**User-interface component,Stateless**;*REST API, AJAX*; ReSTful Web Services used to interact with Subversion Repositories [45]	**Stateless**; *Bugzilla:WebService API*; Bugzilla::WebService API[46]
	Multitenancy	**Shared component**; *Elastic Platform, Hypervisor*; JIRA login system[42]	**Shared component**; *Hypervisor*; VersionOne supports re-useable configuration schemes[43]	**Shared component**; *Hypervisor*; Hudson 3.2.0 supports multi-tenancy with Job Group View and Slave isolation[44]	**Tenant Isolated component**; *Hypervisor*; Global search/replace operations are shielded from corrupting subversion repository.[45]	**Shared component**; *Hypervisor*; Different users are virtually isolated within Bugzilla DB[46]
	Cloud Integration	**Restricted Data Access component, Integration provider**; *REST API*; JIRA REST API is used to integrate JIRA with other applications[42]	**Integration provider**; *REST, Web Services*; VersionOne OpenAgile Integrations platform, REST Data API for user stories[43]	**Integration provider**; *REST, Web Services*; Stapler component of Hudson's architecture uses REST[44]	**Integration provider**; *REST, Web Services*; Subversion API[45]	**Integration provider**; *REST, Web Services*; Bugzilla::WebService API[46]
Cloud Offering	Cloud environment Offering	**Elastic platform**; *PaaS*; JIRA Elastic Bamboo runs builds to create instances of remote agents in the Amazon EC2[42]	**Integration provider**; *REST, Web Services*; Versionone's Project Management tools are used with TestComplete for automated testing environment [43]	**Elastic Infrastructure/Platform, Node-based Availability**; *PaaS, IaaS*; Hudson is a distributed build platform with ”master/slave” configuration [44]	**Elastic platform**; *PaaS*; Subversion repository can be accessed by a self-service interface hosted on a shared middleware	**Elastic Platform**; *PaaS*; Bugzilla s hosted on a middleware offered by providers[46]
	Processing Offering	**Hypervisor**; *Virtualization*; JIRA is deployed on virtualized hardware	**Hypervisor**; *Virtualization*; VersionOne can be deployed on virtualized hardware	**Hypervisor**; *Virtualization*; Hudson is deployed on virtualized hardware	**Hypervisor**; *Virtualization*; Subversion is deployed on virtualized hardware	**Hypervisor**; *Virtualization*; Bugzilla is deployed on virtualized hardware
	Storage Offering	**Block**; *Virtualization*; Elastic Bamboo can access centralized block storage thru an API integrated into an operating system running on virtual server[42]	**Block storage**; *Virtualization*; VersionOne can access centralized block storage thru an API integrated into an operating system running on virtual server[43]	**Block, Blob storage**; *Virtualization*; Azure Blob service used as a repository of build artifacts created by a Hudson	**Hypervisor**; *Virtualization*; Subversion can access centralized block storage thru an API integrated into an operating system running on virtual server	**Hypervisor**; *Virtualization*; Bugzilla can access centralized block storage thru an API integrated into an operating system running on virtual server
	Communication Offering	**Message-Oriented Middleware**; *Message Queuing*; JIRA Mail Queue[42]	**Message-Oriented Middleware**; *Message Queuing*; VersionOne's Defect Work Queues[43]	**Message-Oriented Middleware, Virtual networking**;*Message Queuing,Hypervisor*;Hudson's Execution System Queuing component	**Message-Oriented Middleware**;*Message Queuing*;Subversion's Repository layer[45]	**Message-Oriented Middleware**;*Message Queuing*; Bugzilla's Mail Transfer Agent[46]
Cloud Management	Management Components	**Provider Adapter, Managed Configuration, Elastic manager**;*RPC, API*; JIRA Connect Framework[42], JIRA Advanced configuration	**Managed Configuration**;*RPC, API*; VersionOne segregation and appl. configuration	**Elastic load balancer, watchdog**;*Elastic platform*; Hudson execution system's Load Balancer component)	**Managed Configuration**;*RPC, API*; configuration file is used to configure how/when builds are done	**Managed Configuration**;*RPC, API*; Bugzilla can use configuration file for tracking and correcting errors
	Management Processes	**Elastic management process**;*Elasticity Manager*; JIRA Elastic Bamboo, and Time Tracking feature[42]	**Elastic management process**;*Elasticity Manager*; VersionOne's OnDemand security platform[43]	**Update Transition process**;*Message Queuing*; continuous integration of codes by Hudson's CI server[44]	**Update Transition process**;*Message Queuing*; continuous updates of production versions of the appl. by Subversion[45]	**Resiliency management process**;*Elasticity platform*; Bugzilla Bug monitoring/reporting feature[46]
Composite Application	Decomposition Style	**3-tier**;*stateless, processing and data access components*; JIRA is web-based application[42]	**3-tier**;*stateless, processing and data access components*; VersionOne is a web application[43]	**3-tier, Content Dist. Network**;*user interface, processing, data access components, replica distr.*; Hudson is an extensible web application, code file replicated on multiple clouds[44]	**3-tier**;*stateless, processing and data access components*; Subversion is a web-based application [45]	**3-tier**;*stateless, processing and data access components*; Bugzilla is a web application[46]
	Hybrid Cloud Application	**Hybrid processing**; *processing component*; JIRA Agile used to track daily progress work[42]	**Hybrid Development Environment**;*processing component*; VersionOne's OpenAgile Integration[43]	**Hybrid Data, Hybrid Development Environment**; *data access component*;Separate environment for code verification and testing	**Hybrid Data, Hybrid Backup**; *data access component,stateless*;Code files extracted for external storage	**Hybrid Processing**; *processing component*; DB resides in data center, processing done in elastic cloud

the design of cloud-native applications. Assuming an architect is either interested in providing the right cloud resources, or mapping the business requirement to cloud properties that cannot be changed (e.g., location and ownership of the cloud infrastructure), then Taxonomy A would be more relevant.

However, if the interest is in mitigating certain cloud properties that can be compensated at an application level (e.g., improving the availability of the cloud-hosted GSD tool), then Taxonomy B should be considered. Fehling et al. describe other cloud properties that are either unchangeable or compensatable for deploying cloud applications [7].

VI. Recommendations

In this section, we present a set of recommendations in the form of selection criteria in Table V to guide an architect in choosing applicable deployment patterns for deploying any GSD tool.

To further assist the architect in making a good choice, we describe CLIP (CLoud-based Identification process for deployment Patterns), a general process for guiding architects in selecting applicable cloud deployment patterns for GSD tools using our taxonomy. The development of CLIP was

TABLE V. RECOMMENDATIONS FOR SELECTING APPLICABLE DEPLOYMENT PATTERNS FOR CLOUD DEPLOYMENT OF GSD TOOLS.

Category	**Sub-Category**	***Selection Criteria***	***Applicable Patterns***
Application Process	Project Processes	Elasticity of the cloud environment is not required	Static workload
	Implementation Processes	Expects continuous growth or decline in workload over time	Continuously changing workload
	Support Processes	Resources required is nearly constant;continuous decline in workload	Static workload, Continuously changing workload
Core Cloud Properties	Rapid Elasticity	Explicit requirement for adding or removing cloud resources	Elastic platform, Elastic Infrastructure
	Resource Pooling	Sharing of resources on specific cloud stack level-IaaS, PaaS, SaaS	Hypervisor, Standby Pooling Process
	Measured Service	Prevent monopolization of resources	Elastic Infrastructure, Platform, Throttling/Service Metering[8]
Cloud Service Model	Software Resources	No requirement to deploy and configure GSD tool	Software as a Service
	Platform Resources	Requirement to develop and deploy GSD tool and/or components	Platform as a Service
	Infrastructure as a Service	Requires control of infrastructure resources (e.g., storage, memory) to accommodate configuration requirements of the GSD tool	Infrastructure as a Service
Cloud Deployment Model	Private Usage	Combined assurance of privacy, security and trust	Private cloud
	Community Usage	Exclusive access by a community of trusted collaborative users	Community cloud
	Public Usage	Accessible to a large group of users/developers	Public cloud
	Hybrid Usage	Integration of different clouds and static data centres to form a homogenous deployment environment	Hybrid cloud
Application Architecture	Application Components	Maintains no internal state information	User Interface component, Stateless pattern
	Multitenancy	A single instance of an application component is used to serve multiple users, depending on the required degree of tenant isolation	Shared component, tenant-isolated component, dedicated component
	Integration	Integrate GSD tool with different components residing in multiple clouds	Integration provider, Restricted Data Access component
Cloud Offering	Cloud environment	Requires a cloud environment configured to suit PaaS or IaaS offering	Elastic platform, elastic infrastructure
	Processing Offering	Requires functionality to execute workload on the cloud	Hypervisor
	Storage Offering	Requires storage of data in cloud	Block storage, relational database
	Communication Offering	(1) Require exchange of messages internally between appl. components; (2) Require communication with external components	(1) Message-oriented middleware; (2) Virtual Networking
Cloud Management	Management Components	(1) Pattern supports Asynchronous access; (2) State information is kept externally in a central storage	(1) Provider Adapter; Elastic manager; Managed Configuration
	Management Processes	(1)Application component requires continuous update; (2) Automatic detection and correction of errors	(1) Update Transition process; (2) Resiliency management process
Composite Application	Decomposition Style	Replication or decomposition of application functionality/components	(1) 3-tier; (2) Content Distribution Network
	Hybrid Cloud Application	Require the distribution of functionality and/or components of the GSD tool among different clouds	(1) Hybrid processing; (2) Hybrid Data; (3) Hybrid Backup; (4) Hybrid Development Environment

inspired by IDAPO, a similar process proposed by Stol et al. for identifying architectural patterns in open source software [6]. The key for the successful use of CLIP is selecting a suitable level of cloud stack that will accommodate all the configuration requirements of the GSD tool to be selected. The architect has more flexibility to implement or support the implementation of a deployment pattern when there is greater "scope of control" of the cloud stack according to either the SaaS, PaaS or IaaS service delivery model [9]. For example, to implement the hybrid data pattern [7] for deploying Hudson to an elastic cloud, the architect would require control of the infrastructure level of the cloud stack to allow for provisioning and de-provisioning of resources (e.g., storage, memory, CPU).

VII. Limitations of the study

There are multiple taxonomies developed by researchers to categorize the cloud computing space into various aspects such as, cloud resources provisioned to customers, features of cloud environment for research and scientific experimentation, cloud usage scenarios [53] and cloud architectural patterns. This study considered cloud deployment patterns that could be used to design and deploy applications to the cloud.

The findings of this study should not be generalized to:
(i) all cloud hosted software services. We focused on cloud-hosted GSD tools (e.g., Hudson) used for large-scale distributed enterprise software development projects.
(ii) small and medium size software development projects. Large projects are usually executed with stable and reliable GSD tools. For small projects (with few developers and short duration), high performance and low cost may be the main consideration in tool selection.

The small number of GSD tools in the selected dataset is appropriate because we are not carrying out a feature-analysis based study of GSD tools, but only using it to apply against our proposed taxonomy. Future research can be done to re-evaluate how new GSD tools can be positioned within the taxonomy.

VIII. Conclusion

In this paper, we have created and used a taxonomy of deployment patterns for cloud-hosted applications to contribute to the literature on cloud deployment of Global Software Engineering tools.

Eight categories that form the taxonomy have been described: Application process, Cloud properties, Service model, Deployment model, Application architecture, Cloud offerings, Cloud management, and Composite applications. *Application process* contains patterns that handles the workload imposed on the cloud infrastructure by the ISO/IEC 12207 software processes. *Cloud properties* contains patterns for mitigating the core cloud computing properties of the tools. Patterns in *Service model* and *Deployment model* reflect the NIST cloud definition of service models and deployment models, respectively. *Application architectures* contains patterns that support the architectural components of a cloud-application. Patterns in *Cloud offerings* reflect the main offerings that can be provided to users on the cloud infrastructure. *Cloud management* contains patterns used to manage both the components and processes of software tools. *Composite cloud* contains patterns that can be formed by combining other patterns or can be decomposed into separate components.

These categories were further partitioned into 24 subcategories, which were mapped to the components of an (architectural) deployment structure. This mapping reveals two components classes: cloud-hosted environment and cloud-hosted application. Cloud-hosted environment and cloud-hosted application classes capture patterns that can be used to address deployment challenges at the infrastructure level and application level, respectively.

By positioning a selected set of software tools, JIRA, VersionOne, Hudson, Subversion and Bugzilla, on the taxonomy, we were able to identify applicable deployment patterns together with the supporting technologies for deploying cloud-hosted GSD tools. We observed that most deployment patterns are related and can be implemented by combining with others, for example, in hybrid deployment scenarios to integrate data residing in multiple clouds.

We have described CLIP, a novel approach for selecting applicable cloud deployment patterns, and thereafter applied it to a motivating deployment problem involving the cloud deployment of a GSD tool to serve multiple users in such a way that guarantees isolation among different users. We have also provided recommendations in tabular form, which shows the selection criteria to guide an architect in choosing applicable deployment patterns. Examples of deployment patterns derived from applying these selection criteria have been presented.

We plan to carry out several Case Studies involving the deployment of cloud-hosted GSD tools to compare how well different deployment patterns perform under different deployment conditions with respect to specific Global Software Development tools and processes (e.g., continuous integration with Hudson). Thereafter, we will carryout a cross-case analysis to synthesize the findings of the different case studies. In the future, we will develop a Decision Support Model based on the CLIP framework to help software architects in automating the process of selecting applicable cloud deployment patterns for GSD tools. This will speed up the cloud deployment process by providing proven patterns with the supporting technologies.

Acknowledgment

This research was supported by the Tertiary Education Trust Fund (TETFUND), Nigeria, and Robert Gordon University, UK.

References

[1] L. C. Ochei, J. M. Bass, and A. Petrovski, "Taxonomy of deployment patterns for cloud-hosted applications: A case study of global software development (gsd) tools," in *The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2015).* IARIA, 2015, pp. 86–93.

[2] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms.* John Wiley & Sons, Inc., 2011.

[3] M. A. Chauhan and M. A. Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," in *Proceedings of the WICSA/ECSA 2012 Companion Volume.* ACM, 2012, pp. 5–13.

[4] S. Junuzovic and P. Dewan, "Response times in n-user replicated, centralized, and proximity-based hybrid collaboration architectures," in *Proceedings of the 2006 20th anniversary conference on Computer supported co-operative work.* ACM, 2006, pp. 129–138.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 3/E.* Pearson Education India, 2013.

[6] K.-J. Stol, P. Avgeriou, and M. A. Babar, "Design and evaluation of a process for identifying architecture patterns in open source software," in *Software Architecture.* Springer, 2011, pp. 147–163.

[7] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns.* Springer, 2014.

[8] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson, *Cloud Design Patterns*, R. Corbisier, Ed. Microsoft, 2014.

[9] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, "Cloud computing synopsis and recommendations," *NIST special publication*, vol. 800, p. 146, 2012.

[10] S. Hansma, "Go fast and don't break things: Ensuring quality in the cloud." in *Workshop on High Performance Transaction Systems(HPTS 2011), Asilomar, CA, October 2011. Summarized in Conference Reports column of USENIX; login 37(1)*, February 2012.

[11] J. Bass, "How product owner teams scale agile methods to large distributed enterprises," *Empirical Software Engineering*, pp. 1–33, 2014.

[12] W. Aspray, F. Mayadas, M. Y. Vardi *et al.*, "Globalization and offshoring of software," *Report of the ACM Job Migration Task Force, Association for Computing Machinery*, 2006.

[13] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *2007 Future of Software Engineering.* IEEE Computer Society, 2007, pp. 188–198.

[14] C. Larman and B. Vodde, *Practices for scaling lean and agile development: large, multisite, and offshore product development with large-scale Scrum.* Pearson Education, 2010.

[15] F. Lanubile, "Collaboration in distributed software development," in *Software Engineering.* Springer, 2009, pp. 174–193.

[16] J.-P. Pesola, H. Tanner, J. Eskeli, P. Parviainen, and D. Bendas, "Integrating early v and v support to a gse tool integration platform," in *Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on.* IEEE, 2011, pp. 95–101.

[17] M. A. Babar and M. Zahedi, "Global software development: A review of the state-of-the-art (2007–2011)," IT University of Copenhagen, Tech. Rep., 2012.

[18] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *Software Engineering, IEEE Transactions on*, vol. 29, no. 6, pp. 481–494, 2003.

[19] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaíno,

"Collaboration tools for global software engineering," *Software, IEEE*, vol. 27, no. 2, pp. 52–55, 2010.

[20] J. Portillo-Rodriguez, A. Vizcaino, C. Ebert, and M. Piattini, "Tools to support global software development processes: a survey," in *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*. IEEE, 2010, pp. 13–22.

[21] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Addison-Wesley*, vol. 49, p. 120, 1995.

[22] B. Wilder, *Cloud Architecture Patterns*, 1st ed., R. Roumeliotis, Ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly Media, Inc., 2012.

[23] IEEE, "Ieee standard glossary of software engineering terminology," *Office*, vol. 121990, no. 1, p. 1, 1990.

[24] M. Unterkalmsteiner, R. Feldt, and T. Gorschek, "A taxonomy for requirements engineering and software test alignment," *ACM Transactions on Software Engineering and Methodology*, vol. Vol. V, No. N, Article A, 2013. [Online]. Available: http://doi.acm.org/10.1145/0000000.0000000

[25] R. L. Glass and I. Vessey, "Contemporary application-domain taxonomies," *Software, IEEE*, vol. 12, no. 4, pp. 63–76, 1995.

[26] D. I. Sjoberg, T. Dyba, and M. Jorgensen, "The future of empirical methods in software engineering research," in *Future of Software Engineering, 2007. FOSE'07.* IEEE, 2007, pp. 358–378.

[27] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 5, pp. 309–332, 2005.

[28] D. Smite, C. Wohlin, Z. Galvina, and R. Prikladnicki, "An empirically based terminology and taxonomy for global software engineering," *Empirical Software Engineering*, pp. 1–49, 2012.

[29] R. Dupuis, "Software engineering body of knowledge," 2004.

[30] C. Moyer, *Building Applications for the Cloud: Concepts, Patterns and Projects*. Pearson Education, Inc, Rights and Contracts Department, 501 Boylston Street, Suite 900, Boston, MA 02116, USA: Addison-Wesley Publishing Company, 2012.

[31] N. Sawant and H. Shah, *Big Data Application Architecture - A problem Solution Approach*. Apress, 2013.

[32] Z. Mahmood, *Cloud Computing: Methods and Practical Approaches*. Springer-Verlag London, 2013.

[33] T. Erl and A. Naserpour, *Cloud Computing Design Patterns*. Prentice Hall, 2014.

[34] S. Strauch, U. Breitenbuecher, O. Kopp, F. Leymann, and T. Unger, "Cloud data patterns for confidentiality," in *Proceedings of the 2nd International Conference on Cloud Computing and Service Science, CLOSER 2012, 18-21 April 2012, Porto, Portugal.* SciTePress, 2012, pp. 387–394.

[35] J. Varia, "Migrating your existing applications to the cloud: a phase-driven approach to cloud migration." Amazon Web Services (AWS), [Online: accessed in November, 2014 from http://aws.amazon.com/whitepapers/].

[36] J. Musser. (2012) Enterprise-class api patterns for cloud and mobile. CITO Research.

[37] Arista.com. Cloud networking: Design patterns for cloud-centric application environments. ARISTA Networks, Inc. Online: accessed in November, 2015 from http://www.arista.com/assets/data/pdf/CloudCentric...pdf.

[38] C. Brandle, V. Grose, M. Young Hong, J. Imholz, P. Kaggali, and M. Mantegazza. (2014, June) Cloud computing patterns of expertise. IBM. International Technical Support Organization. [Online]. Available: ibm.com/redbooks

[39] J. Varia. (2014) Architecting for the cloud: best practices. Amazon Web Services (AWS). Online: accessed in November, 2015 from http://aws.amazon.com/whitepapers/.

[40] L. Lilien, "A taxonomy of specialized ad hoc networks and systems for emergency applications," in *Mobile and Ubiquitous Systems: Networking & Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on*. IEEE, 2007, pp. 1–8.

[41] P. Mell and T. Grance, "The nist definition of cloud computing," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[42] Atlassian.com. Atlassian documentation for jira 6.1. Atlassian, Inc. Online: accessed in November, 2015 from https://www.atlassian.com/software/jira/.

[43] VersionOne, "Versionone-agile project management and scrum," VersionOne Inc., [Online: accessed in November, 2015 from http://www.versionone.com].

[44] M. Moser and T. O'Brien. The hudson book. Oracle, Inc., USA. Online: accessed in November, 2015 from http://www.eclipse.org/hudson/the-hudson-book/book-hudson.pdf.

[45] B. Collins-Sussman, B. Fitzpatrick, and M. Pilato, *Version control with subversion*. O'Reilly, 2004.

[46] Bugzilla.org. The bugzilla guide. The Mozilla Foundation. [Online: accessed in November, 2015 from http://www.bugzilla.org/docs/.

[47] Hudson. Hudson extensible continuous integration server. Oracle Inc., USA. [Online: accessed in November, 2015 from http://www.hudson-ci.org].

[48] Versionone.com. Top 10 reasons to choose versionone. VersionOne Inc. [Online: accessed in November, 2015 from http://www.versionone.com/whyversionone.pdf].

[49] N. Serrano and I. Ciordia, "Bugzilla, itracker, and other bug trackers," *Software, IEEE*, vol. 22, no. 2, pp. 11–13, 2005.

[50] L. C. Ochei, J. M. Bass, and A. Petrovski, "Evaluating degrees of multitenancy isolation: A case study of cloud-hosted gsd tools," in *2015 International Conference on Cloud and Autonomic Computing (ICCAC)*. IEEE, 2015, pp. 101–112.

[51] CollabNet. Subversionedge for the enterprise. CollabNet, Inc. [Online: accessed in November, 2015 from http://www.collab.net/products/subversion].

[52] I. Sommerville, *Software Engineering*. Pearson Education, Inc. and Addison-Wesley, 2011.

[53] A. Milenkoski, A. Iosup, S. Kounev, K. Sachs, P. Rygielski, J. Ding, W. Cirne, and F. Rosenberg, "Cloud usage patterns: A formalism for description of cloud usage scenarios," Standard Performance Evaluation Corporation(SPEC) Research Cloud Working Group, Tech. Rep., 2013.