# On Cloud-Based Software Processes

Sami Alajrami, Alexander Romanovsky and Paul Watson

**TECHNICAL REPORT SERIES**

**No. CS-TR-1458          March 2015**

**TECHNICAL REPORT SERIES**

---

**No. CS-TR-1458 March, 2015**

# On Cloud-Based Software Processes

S. Alajrami, A. Romanovsky, P. Watson

## Abstract

Development of modern critical software requires stringent processes that are scalable, automated and support traceability between various artefacts. It is important that the processes ensure that all the development steps are fully documented. This paper proposes a software engineering architecture that supports the development of critical software by enacting the development activities from a centralized cloud-based service, by storing all the development artefacts produced at all steps in a dedicated repository and by providing a general mechanism for supporting a repository of typical development steps (including, the ones supported by verification and analysis tools packaged as services). This extendable architecture makes use of all benefits the cloud technologies provide, including elasticity and per-demand cost, to help all system developers work together on a project and share processes, tools and artefacts. The paper introduces the general architecture, describes all its components, presents a proof-of-concept implementation, and briefly outlines a simple delegation example in which two organisations are involved in development to demonstrate how the implementation operates.

## Bibliographical details

ALAJRAMI, S., ROMANOVSKY, A., WATSON, P.
On Cloud-Based Software Processes
[By] S. Alajrami, A. Romanovsky, P. Watson
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1458)

### Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series.  CS-TR-1458

### Abstract

Development of modern critical software requires stringent processes that are scalable, automated and support traceability between various artefacts. It is important that the processes ensure that all the development steps are fully documented. This paper proposes a software engineering architecture that supports the development of critical software by enacting the development activities from a centralized cloud-based service, by storing all the development artefacts produced at all steps in a dedicated repository and by providing a general mechanism for supporting a repository of typical development steps (including, the ones supported by verification and analysis tools packaged as services). This extendable architecture makes use of all benefits the cloud technologies provide, including elasticity and per-demand cost, to help all system developers work together on a project and share processes, tools and artefacts. The paper introduces the general architecture, describes all its components, presents a proof-of-concept implementation, and briefly outlines a simple delegation example in which two organisations are involved in development to demonstrate how the implementation operates.

### About the authors

Sami Alajrami is a PhD student at the school of ComputiSami Alajrami is a PhD student at the school of Computing Science at Newcastle University and his work focuses on novel ways of delivering software engineering in the cloud. He received his BSc. degree in Computer Systems Engineering from Al-Azhar University - Gaza, Palestine in 2010, and MSc. in Internet Technologies and Enterprise Computing from Newcastle University in 2012. Sami has worked at the Cloud & Security Lab in Hewlett Packard Labs, Bristol as a software intern where he has been involved in the Security Intelligence as a Service (SILAS) and the Situational Awareness as a Service (SAaaS) projects.
ng Science at Newcastle University and his work focuses on novel ways of delivering software engineering in the cloud. He received his BSc. degree in Computer Systems Engineering from Al-Azhar University - Gaza, Palestine in 2010, and MSc. in Internet Technologies and Enterprise Computing from Newcastle University in 2012. Sami has worked at the Cloud & Security Lab in Hewlett Packard Labs, Bristol as a software intern where he has been involved in the Security Intelligence as a Service (SILAS) and the Situational Awareness as a Service (SAaaS) projects.

Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance. He received a PhD degree in Computer Science from St. Petersburg State Technical University and has worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland and at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993 he became a postdoctoral fellow in Newcastle University, and worked on the ESPRIT projects on Predictable Dependable Computing Systems (PDCS), Design for Validation (DeVa) and on UK-funded projects on the Diversity, both in Safety Critical Software using Off-the-Shelf components. He was a member of the executive board of EU Dependable Systems of Systems (DSoS) Project, and between 2004 and 2012 headed projects on the development of a Rigorous Open Development Environment for Complex Systems (RODIN), and latterly was coordinator of the major FP7 Integrated Project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). He now leads work on fault tolerance in Systems of Systems within the COMPASS project and is Principal Investigator of Newcastle's Platform Grant on Trustworthy Ambient Systems.

Paul Watson is Professor of Computer Science and Director of the Digital Institute. He also directs the £12M RCUK-funded Digital Economy Hub on Social Inclusion through the Digital Economy. He graduated in 1983 with a BSc in Computer Engineering from Manchester University, followed by a PhD on parallel graph reduction in 1986. In the 80s, as a Lecturer at Manchester University, he was a designer of the Alvey Flagship and Esprit EDS systems. From 1990-5 he worked for ICL as a system designer of the Goldrush MegaServer parallel database server, which was released as a product in 1994. In August 1995 he moved to Newcastle University, where he has been an investigator on research projects worth over £30M. His research interest is in scalable information management with a current focus on Cloud Computing. Professor Watson is a Chartered Engineer, a Fellow of the British Computer Society, and a member of the UK Computing Research Committee.

### Suggested keywords

SOFTWARE PROCESS MODELLING
PROCESS ENACTMENT
DEVELOPMENT ARTEFACTS
SERVICE-ORIENTED ARCHITECTURE
CLOUD COMPUTING

# On Cloud-based Software Processes

Sami Alajrami
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
s.h.alajrami@ncl.ac.uk

Alexander Romanovsky
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
alexander.romanovsky@ncl.ac.uk

Paul Watson
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
paul.watson@ncl.ac.uk

## ABSTRACT

Development of modern critical software requires stringent processes that are scalable, automated and support traceability between various artefacts. It is important that the processes ensure that all the development steps are fully documented. This paper proposes a software engineering architecture that supports the development of critical software by enacting the development activities from a centralised cloud-based service, by storing all the development artefacts produced at all steps in a dedicated repository and by providing a general mechanism for supporting a repository of typical development steps (including, the ones supported by verification and analysis tools packaged as services). This extendible architecture makes use of all benefits the cloud technologies provide, including elasticity and per-demand cost, to help all system developers work together on a project and share processes, tools and artefacts. The paper introduces the general architecture, describes all its components, presents a proof-of-concept implementation, and briefly outlines a simple delegation example in which two organisations are involved in development to demonstrate how the implementation operates.

## Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management —*Software process models*

## Keywords

Software process modelling, process enactment, development artefacts, service-oriented architecture, cloud computing

## 1. INTRODUCTION

As software systems are becoming more pervasive and our society is becoming more dependent on them, the complexity of these systems has been dramatically increasing. This complexity makes producing software systems a challenging task. When software is used in business, mission- and safety-critical applications its production needs to follow stringent development processes that ensure that well-defined steps are followed and traceability of the development artefacts is supported, and allow various stakeholders (e.g. certifiers, customers, managers) to have access to the development history. Due to the complexity of this task there is a constant pressure to improve the processes and to automate them as much as possible (while ensuring their usability).

There are various techniques that support capturing software processes in a useful workflow notation. Their use allows the developers not only to document, model and analyse processes but, which is more important, to capture, enact and, even, enforce the required practical steps.

During software development various artefacts are produced and shared between stakeholders (requirement documents, specifications, architectures, models, code, test cases, process descriptions, and many more). Every development step inputs a set of artefacts and produces a set of new artefacts. When the development steps are repeated (e.g. compile-debug-edit) modified sets of artefacts are routinely produced. These artefacts play a crucial role not only during system development but also for ensuring the system quality, system assessment, certification, documentation, delivering the system to the customer, deployment, etc.

The cloud computing paradigm has evolved to simplify organisational IT management and maintenance, and cut both operational and expenditure costs. Cloud offers computing resources on demand using different service models (infrastructures, platforms, and software) and different deployment models (public, private, community, and hybrid) [20].

In general, there are two perspectives to realize the potential collaboration between cloud and software engineering: (a) the use of cloud to support the software development process, (b) advancing software development methodologies to suite developing software for the cloud. The work presented in this paper fits in the first perspective. As the cloud is being widely adopted by both research and industry, researchers have started investigating the potential of using it to support some software development phases (especially the computing intensive ones e.g. testing) [23, 8, 24]. There are various cloud/web-based tools already used in practice (such as Github and Jenkins) that support particular software development tasks.

This paper proposes a cloud-based software engineering architecture that supports the development of critical software by enacting the development processes from a centralised cloud-based service, by storing all the artefacts produced at all development steps in a dedicated cloud repository and by providing a general mechanism for supporting a cloud repository of typical development steps and activities (that are viewed as a key software artefact). The architecture is open, extendable and allows the developers to quickly create, modify and enact processes that capture the development steps of different scales and time-spans. This, for example, allows automation of the non-interactive or repetitive development tasks.

The rest of the paper is structured as follows: a background discussing software processes and their modelling techniques is established in section 2. Section 3 describes the general architecture for our cloud-based software development platform. Section 4 discusses proof-of-concept implementation of our architecture. A small case study demonstrating how the architecture works is discussed in Section 5. The paper concludes with a brief summary of future work.

## 2. MOTIVATION AND BACKGROUND

### 2.1 Software Development Landscape

Since the introduction of the waterfall software development model [29], many variations of software process models came out to meet the challenging needs of software production. These models have moved from the slow and restrictive models to the fast and flexible ones [15]. Software-powered systems have become a vital part of everyday modern life which has increased the complexity and scale of software systems. The rise of agile software processes came as a response to this growing complexity and scale and aimed to provide a cheaper and faster process model. Today, Global Software Development (GSD) is a popular development model where teams are distributed (sometimes across continents) and collaborate in the development process. This popularity was driven by some factors; including: globalization, financial considerations (where companies employ cheap labour to reduce costs), proximity to local customers and markets, and faster delivery times (due to teams working around the clock in different countries).

Despite that the available software development processes are documented and described in literature, they are still abstract road maps and every company would implement them differently

### 2.2 Software Processes

Paulk et al. [25] describe software process as "a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals). As an organization matures, the software process becomes better defined and more consistently implemented throughout the organization". Software processes provide a guidance on the order of the stages involved in a project and the transitions between them [6]. Over the years, several process models have emerged, such as: the waterfall model [29], the spiral model [6], extreme programming model (XP), and the agile models to name a few. The agile models have gained popularity within the last decade as they focus on agility and speed of software development processes. The agile software development manifesto [1] explains that it gives higher priority to: individuals and interactions, working software, customer collaboration, and responding to change. Each of the models have its strengths and weaknesses and may or may not be suitable for particular projects. Rodriguez et al. [28] have compared 13 process models and software development life cycles, while Munassar et al. [21] evaluated 5 models of software engineering and when they are/are not suitable to apply. In general, the software process models tend to be generic and abstract.

#### *2.2.1 Software Process Modelling*

Models are used in most engineering domains to provide abstraction from the real world. In software systems, models are used for different purposes such as: documentation, testing, static analysis, and code generation. Modelling software processes has been investigated since late 80s and it was driven by multiple motivations including: a) improving the understanding for different perspectives, by visualizing the relevant components for each perspective. b) facilitating communication among team members, and c) supporting project management through reasoning in order to improve the process. Furthermore, the models can be partially automated (e.g. repetitive and non-interactive tasks). Several approaches for software process modelling have been introduced over time, they are categorized into four categories [5]:

1. Rules based (e.g. MARVEL [19])
2. Petri net based (e.g. SPADE [2])
3. Programming languages based (e.g. SPELL [12])
4. UML based (e.g. SPEM [2])

The first three did not receive industrial take up due to their complexity and inflexibility [17]. The UML approach was based on utilizing the wide adoption and acceptance of the Unified Modelling Language (UML) for modelling software processes. Several implementations of this approach have been proposed each with different strengths and weaknesses. The authors in [5], compare six UML-based modelling approaches based on a set of software process modelling requirements. The authors also admit that executability and formality are major weaknesses of UML in the context of software process modelling.

Among the previous approaches, SPEM (Software Process Engineering Meta-model) has became an OMG standard for

[1]http://agilemanifesto.org/
[2]http://www.omg.org/spec/SPEM/2.0/

software process modelling. SPEM is based on the concept of interaction between *Roles* that perform *Activities* which consume (and produce) *Work Products* [11]. However, a major criticism of SPEM in literature is its lack of support for process enactment. As a result, several researchers have proposed different approaches and extensions to support process enactment in SPEM. In [31], the authors propose mapping rules to map SPEM models into XML Process Description Language (XPDL) which then can be enacted. In [26], authors propose xSPIDER_ML (a software process enactment language based on SPEM 2.0 concepts). Although xSPIDER_ML is supported with a modelling tool and an enactment environment, the notion of enactment is limited to process monitoring since developers are supposed to perform their tasks off-line and report their progress to the enactment environment. The authors in [13] introduce eSPEM which is a SPEM extension to allow describing fine-grained behaviour models that facilitate process enactment. They implement a distributed process execution environment [14] based on the Foundational subset for Executable UML Models (FUML [3]) standard with emphasis on supporting the ability to share process state on different nodes, suspend and resume process execution, interact with humans, and adapt to different organizations. However, the notion of process enactment in that execution environment also assumes that developers carry out their tasks outside the execution environment and return control back to it once they finish.

Enforcing strict detailed processes can be useful in some cases (e.g. for certifying safety-critical systems). However, in practice, it can be restrictive for the creativity of team members. Organizations have been moving to agile methods to gain more dynamicity and to increase productivity.

### 2.2.2 Challenges

A decade ago, Boehm identified eight trends that will have an impact on software engineering and processes [7]. The fifth trend was *Globalization and Interoperability* where he anticipated that global development and collaboration will be a common place in the software industry and that there would be a need to overcome challenges associated with global software development (GSD). These challenges are caused due temporal, cultural and geographical distances [27]. In addition, Boehm emphasizes on the need for groups support in software packages rather than supporting individual use as they do now. Another challenge that software processes face as mentioned earlier is the lack of executability and autonomy support for software process models.

## 3. PROPOSED ARCHITECTURE FOR CLOUD-BASED SOFTWARE PROCESS ENACTMENT

As highlighted in the previous section, software engineering community is adopting the agile globally distributed approach for development and is facing several challenges such as: the need for efficient collaboration, groups oriented tools, process executability and autonomy, and utilizing the computing plenty. In this paper we address some of these challenges by proposing a cloud-based architecture for software process enactment.

[3]http://www.omg.org/spec/FUML/

Modern software processes need to be dynamic and flexible in order to respond to a rapidly and continuously changing world. Furthermore, these processes need to incorporate multiple stakeholders (i.e. developers, architects, quality assurance engineers, project managers, customers, and sub-contractors). Since most of the existing process models are generic and act as guidelines on how to perform the process, companies tend to have their own tailored versions of those generic ones that team members know mostly by heart (except in some domains where the processes might be very stringent and documented, e.g. safety critical systems). As explained in the previous section, the current state-of-the-art software process modelling languages and notations lack support for executing/enacting processes. To the best of our knowledge, the research attempts to support process enactment in SPEM for example, do not have any publicly available tool support. These developments typically have a narrow vision of enactment that is limited to process monitoring and management (with an assumption that the activities in the process will be undertaken outside of the enactment environment). We go one step further and envision that process enactment should encapsulate both process management and monitoring as well as executing the process activities within the same enactment environment. Therefore, we also propose a simple and informal software process modelling notation with supporting exeutability as the main objective of modelling a process.

Fig 1 illustrates the general architecture we are proposing for the cloud-based software process enactment. The system consists of three logical layers. The top layer is for process modelling where a project manager or a software developer can create/edit models for either higher level abstract processes (e.g project plan) or for daily tasks processes (e.g. implementation). The workflow management layer is where the enactment of the processes takes place while the cloud management layer handles the underlying cloud infrastructure issues (e.g. QoS and multiple cloud providers/models).

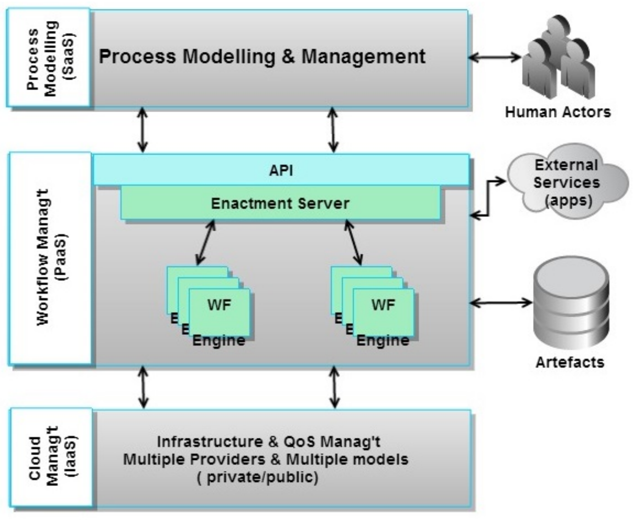


**Figure 1: Proposed architecture to support SW process enactment in the cloud**

The key features of this architecture and how they relate to some of the challenges mentioned earlier are detailed below:

- **Automation:** Often, software processes include some repetitive and non-interactive activities. These activities can be automated. Moreover, the flow control of the process, which is usually driven by results achieved or by change, can also be automated in a predefined way. If we add *automation* to the definition of software processes in section 2.2, it will comply with The Workflow Management Coalition (WfMC) definition of workflow [30] as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". Therefore software process can naturally be seen as a workflow. The idea of using workflow technologies for software processes is not new, several researchers have investigated it [4, 22, 9].

- **Tools and Artefacts:** In a company's tailored process, stakeholders perform certain activities in a predefined order with the aid of certain tools endorsed and provided by the company. Although these tools are still oriented for individual use, the architecture supports integrating tools within the enactment environment either by deploying them directly or by wrapping them as a service. This creates a repository of tools that can be used within the process. Unlike the current practice where tools are deployed on servers or stakeholders' workstations, our approach is that tools will be offered as a service [10] and will be orchestrated together within the process by the enactment environment. Therefore, different stakeholders can contribute their parts and oversee how the entire process is being executed. Similarly, the artefacts provided to/produced from the process are all saved and traced. Software Processes themselves are considered key artefacts which are stored in the artefact repository. Changes made to artefacts (including the processes) are recorded in the repository for traceability.

- **Collaboration:** Collaboration is supported by assigning activities to actors who can then execute them within a process that involves multiple stakeholders. The interactions between the stakeholders (actors) and the system are all recorded which also supports accountability and traceability. The architecture also supports collaboration across companies to perform a software process. This is illustrated in detail in section 5. The stakeholders will use their individual computers/devices to access the system and become part of the software development using a standard browser interface and run tools in the cloud. This makes participation location/tools/device -independent.

- **Computational plenty:** Cloud computing have been evolving over the last decade and have driven the transformation of computing resources (software and hardware) into services. The cloud offers computational plenty that has an appealing cost model. This can be utilized for some computing intensive activities (e.g. testing and verification). The availability and accessibility of the cloud supports collaboration as mentioned above. the process model can contain any cloud-related configuration that might be specific to a particular activity (e.g. executing an activity on private cloud for confidentiality reasons, utilizing cloud's elasticity to execute a computing intensive activity). These configurations are then considered when the process is executed.

## 3.1 Process Modelling and Definition

Software processes need to be dynamic and flexible to capture the everyday work of the developers. And at the same time, the organization should be able to tailor them to support general process models (such as spiral or waterfall) to meet their needs. In addition, organizations tailor process models such as waterfall or spiral differently to meet their needs. Hence, a flexible modelling notation is required. This notation needs to be (a) expressive (to express the process and its cloud execution settings), (b) executable, and (c) understandable and easy to use. The notation needs to support a combination of on the fly creation/modification of activities for the purposes of capturing the short term/everyday development and of the longer term activities at the organizational level. Process models consist of multiple entities that need to be captured by the modelling notation. Fig 2 illustrates those entities.

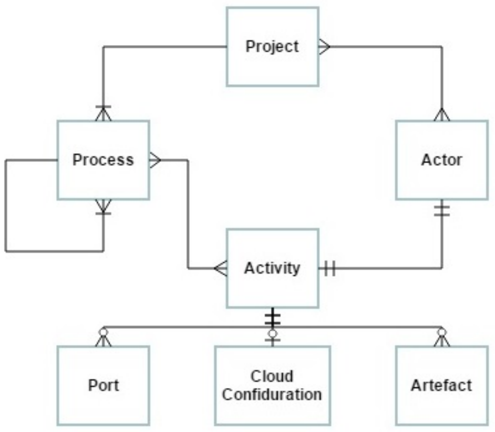


**Figure 2: Entity Relationship diagram**

- **Project:** it represents the project that processes belong to. A project can have multiple number of processes.

- **Process (Workflow):** this is the software development cycle. A project might have multiple processes which when combined together can form the entire software development life cycle. A process is usually created by an actor but might be performed by multiple actors.

- **Actor:** people who are involved in the process such as: process managers, software engineers, testers, etc. A project will involve multiple actors. Although a team of actors might collaborate off-line on performing an activity, the activity will be assigned to a single actor who takes the responsibility for this activity.

- **Artefacts:** items produced or needed by the activities of the software development process (e.g. code, executables, models, documents, etc.).
- **Activities:** Activities represent the smallest unit of execution. They represent the different steps in a software production life cycle. Those steps usually involve the use of tools and/or actor interaction to be completed. The platform can support different types of activities:
  - **Abstract activities:** an activity can represent a complete sub-process, in this case, the activity will not be executable itself.
  - **Concrete activities:**
    - *Local activities:* are executable blocks of code. They are built using the provided software development kit (SDK). These activities are written by users and stored in the tools repository within the platform. Local activities can also be wrappers to external services.
    - *Decision points:* a type of activities which allows actors to guide the execution of the process in one of multiple defined directions. This allows for supporting loops, if conditions, and forks.
- **Cloud configuration:** represents cloud-related configurations, the configurations are: number of machines to be used for executing an activity, time out for execution, preferred cloud provider, and whether the activity should be executed on a public or private cloud.
- **Port:** Each activity can have zero or more input ports and zero or more output ports. Ports provide the means to connect activities and direct the process execution flow. They define both the consumed and produced artefacts by an activity. In addition, input ports act as preconditions that need to be satisfied so that the activity can start executing.

Based on this specification, we have designed a modelling notation and created an XML schema meta-model for this notation (not present here due to space limitation). This allows processes to be expressed in XML format (as well as a graphical notation that can be mapped to XML). Ideally, processes can be defined graphically using a graphical notation. The graphically defined process can then be interpreted into XML. Although we do not implement the graphical editing of software processes in this work, we defined the graphical notation for the main process components in table 1. These will help in understanding the process examples below.

Each of the activities can be configured to specify how it will be executed; parameters include: the responsible actor, the cloud execution configurations, and the accepted and expected artefacts.

**Process Example:** Agile methods are widely adopted in industry as they increase the throughput. SCRUM is one of the agile methods which defines a project management framework. This framework defines a set of roles and a set of meetings with different purposes, attendees, and frequencies. Figure 3 illustrates the high level representation of a scrum sprint. This abstraction can be useful from a management perspective. However, it does not specify any details of how developers are going to implement the process. In reality, most software developers use an IDE, an issue tracking tool (e.g. Jira), a continuous integration framework (e.g. Jenkins), and a version control system (e.g. Github). These tools are used on daily basis to write, test, store, and integrate code. A model of the daily development process (representing the implementation sub-process) is illustrated in Figure 4. While the high level abstraction of the SCRUM process consists of abstract activities, the implementation sub-process consists of a set of concrete activities (i.e. tools).

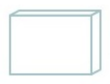
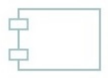
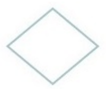
| *Component* | *Graphical Symbol* |
|---|---|
| Abstract Activity |  |
| Concrete Activity |  |
| Decision Point |  |

**Table 1: Graphical representation of software process components**

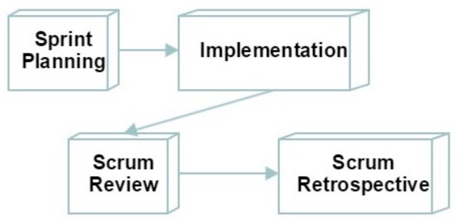


**Figure 3: Scrum high level abstraction**

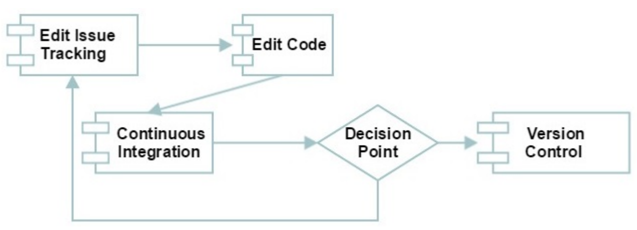


**Figure 4: Daily technical task by a scrum developer**

## 3.2 Artefacts Handling

Software processes include many activities which produce large number of software artefacts such as (code, models, test cases and reports, documentation, etc). These artefacts capture invaluable information about both the software process and product evolution. Researchers have already identified the importance of mining this treasure of information to gain insights about the software process evolution and how it can be improved. Kagdi et al. have surveyed several approaches to artefact repositories mining [18]. This shows the potential and importance of mining software artefacts repositories. At this stage, we focus on building a repository that can be mined and we consider the mining process a future work.

The current practice in industry is to use a set of tools to perform different activities within the software process. These tools produce and maintain software artefacts which are either kept on developers workstations or in a repository. Some commonly used tools include: version control systems (e.g. Github, CVS), issue tracking systems (e.g. Jira), and communication systems (e.g. email). These tools are usually deployed on servers and store different forms of artefacts (code, communications, communication logs, etc), however, these artefacts are stored and maintained in different ways and formats depending on the tools producing and managing them which makes traceability of artefacts harder to achieve.

As part of the proposed architecture, we propose to have a central artefacts repository to store and maintain all the artefacts produced by the different activities involved during the software process. The artefact repository needs to store different types of artefacts (software processes, activities, models, documents, code, tests, etc.). Artefacts would consist of two parts:

- **Artefact meta-data:** contains data about the artefact such as: the actor who created it, date, version, name, type, which activity produced it, which workflow engine was used to produce it, etc.
- **Artefact file:** which is the actual file.

The repository is going to be used for storing and retrieving artefacts files during a process execution. In addition, the artefacts meta-data can be mined for tracing a particular artefact or process. In order to satisfy its purpose, the repository needs to be implemented using a database that: a) is scalable (to cope with large number of artefacts), b) supports fast and reliable storage and retrieval of files as well as the meta-data, and c) provides fast and easy o use query system. Therefore, we decided to use a NoSQL database since it would be scalable, flexible and support fast manipulation of files.

### 3.3 Exposure to Developers

In order to support extensibility, the architecture is exposed to software and process developers who can create custom local activities that can be integrated into the repository and reused in software processes. The architecture will incorporate a Software Development Kit (SDK) to allow developers to define new activities and specify what artefacts they consume and produce, as well as the internal logic of the activity.

## 4. PROTOTYPE IMPLEMENTATION

In order to demonstrate the architecture highlighted in the previous section, we have implemented a prototype of the software processes workflow enactment system, a workflow engine application, and a couple of dummy activities. The implementation has been done in Java for portability reasons. At this stage, the prototype supports only command line activities (tools) which are implemented in Java.

### 4.1 Enactment Service

The enactment service acts as an orchestrator for executing software processes where the execution plan, selection of candidate workflow engines, and monitoring the distributed execution takes place. It is implemented as a REST web service which allows different enactment services (e.g. belonging to different companies) to collaborate on executing a particular process. The enactment service consists of the following components as illustrated in Fig 5.

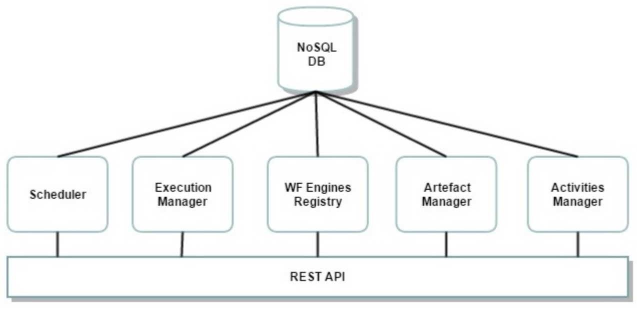


**Figure 5: Enactment Service Componenets**

- **REST API:** the enactment service interacts with the process-modelling layer and with other enactment services through the REST API. This allows the modelling layer to be implemented in any technology and to work on any chosen platform. The REST API provides endpoints for all the underlying services.
- **Scheduler:** handles the planning of a process execution. This involves checking the needed resources (based on the cloud configurations expressed in the process model). The current scheduling mechanism is simple; it allocates activities to the least-busy workflow engine.
- **Execution Manager:** once the scheduler has allocated activities to workflow engines where they will be executed, the distributed execution needs to be managed. The execution manager keeps track of the progress of the execution, and logs execution outcomes.
- **Workflow Engines Registry**: keeps track of all workflow engines that are in service and their status.
- **Artefact Manager:** manages the artefacts that are stored in the NoSQL database.
- **Activities Manager:** similar to the artefact manager, it manages the metadata about activities and the activities' executables (jar files) which are all stored in the NoSQL database.

- **NoSQL DB:** the data about the processes, activities, and the artefacts is stored in MongoDB which is a NoSQL document database. NoSQL databases support scalability and have no predefined schema which gives flexibility in the way data is stored and queried.

## 4.2 Workflow Engines

Workflow engines act as execution containers for executing activities and they can be deployed on any public or private cloud. Workflow engines register themselves with the enactment service when they start, which allows adding more workflow engines dynamically. Activities are allocated to a particular workflow engine by the scheduler of the enactment service and the selected engine is not aware of the process that this activity belongs to. Once a job has been received, the workflow engine requests the resources (artefacts and executables) required to execute this activity from the enactment service through the API. The workflow engine updates the enactment service with the execution progress throughout. When the execution is finished, the workflow engine uploads any produced artefacts to the enactment service and performs a clean up which leaves no traces of this execution on the workflow engine.

## 4.3 Message Oriented Communication

In order to decouple the enactment service from the workflow engines, asynchronous communication between them is achieved through message oriented middleware; namely, Java Messaging Service (JMS). The enactment service pushes jobs to workflow engines by placing the job into their designated jobs queue. The workflow engines place progress updates into the enactment service responses queue. Fig 6 illustrates the communication model.

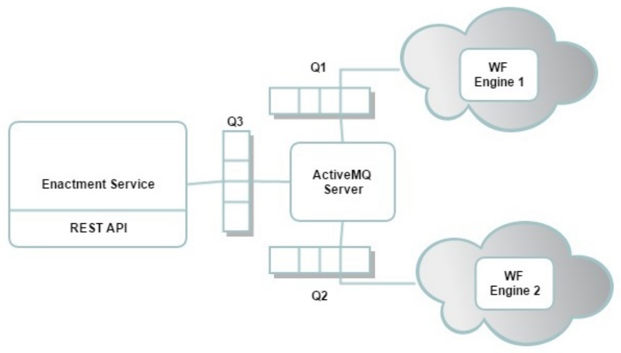


**Figure 6: Message Oriented Communication**

## 4.4 Artefacts Repository

The artefacts repository is implemented in MongoDB which is a scalable open source NoSQL database. MongoDB is a document database which has good support for storing meta-data and good file storage system. Storing data in documents makes the implementation easier since it avoids the complexity of mapping data into tables in the traditional relational database systems. The artefacts are composed of two components: the file representing the artefact, and the meta-data about it. The meta-data is stored as documents in MongoDB while the files are stored in GridFS which is MongoDB specification for storing files. GridFS divides files into chunks and store each them as a single document. It also provides facilities to support having multiple replicas and reading parts of a file without loading the entire file into memory, hence, providing high performance. Since a process can be run several times, it is essential that the repository keeps track of changes made every time to artefacts. This is achieved by adding a version tag to each instance of an artefact. The implementation of the artefact repository allows to trace the artefacts and their evolution throughout the software process evolution. The repository is supported with a REST API to add, retrieve, and query artefacts.

## 4.5 Tools in the Cloud

As mentioned in the previous section, our approach considers offering tools as services that can be orchestrated in our platform. Prior to this work, we have investigated creating services from existing verification tools and using them within a workflow platform. We have experimented with two model checkers (SPIN [16], Divine [3]) where we created web services for these two desktop/cluster-based tools. The services were deployed on the cloud and were configured to utilize the elasticity of the cloud. The details of this investigation are reported in [1]. The two services make use of cloud elasticity by supporting multiple users, even though we could not achieve a substantial performance improvement for Divine by adding new virtual nodes to help in verifying complex models (due to some problems with running local MPI-based networks over the cloud). These two tools are typical activities that can be used as parts of software processes supported by our architecture. This experiment has demonstrated how the activities can be created and stored in our repository both as meta-data and executable files. Overall, the idea of transferring traditional tools into cloud-based services is now a very active area of research and practice with many tools joining the cloud-based arena. Examples include IDEs such as: Eclipse Orion [4] and Codenvy [5].

## 5. CASE STUDY

In order to demonstrate how to use this architecture for a simple example, we have developed and implemented a case study of a company outsourcing part of its software process to another company.

## 5.1 Business Scenario

Company A is a contractor that runs large industrial projects for designing/redesigning railway networks. Among various tools the company uses a number of simulation tools to visualise and analysis the systems it is building, to debug them, to check their characteristics (such as throughput, energy consumption, performance, capacity). During such projects company A develops a wide range of models, diagrams, documents and blueprints that will be used for building the network. As part of this work, company A needs to develop a safe signalling software to operate the network by following a stringent software process.

To ensure the system safety the company would like to use industry-strength formal technologies. Company A does not have expertise in conducting large-scale formal verification

[4]http://eclipse.org/orion/
[5]https://codenvy.com/

of complex systems. This work can be outsourced to small independent company B that has the right skill set. Conducting this type of verification is the main business of company B. The artefacts to be used by company B include layouts, infrastructure data, service patterns, timetables and control tables. As part of its work company B transforms this information into a form that can be used by provers, solvers or model-checker, and that ensures scalable verification. Due to the confidential nature of these artefacts, company B signs a non disclosure agreement with company A and as a precaution, it undertakes all its process in a private cloud. The companies (A and B) only exchange relevant artefacts and do not know each other's internal processes.

### 5.2 Implementation

Our platform can be used to facilitate outsourcing of the verification part of the system development process of company A. Fig 7 illustrates how company A uses their own private deployment to enact its process. Company A's process consists of four activities; namely *T1*, *T2*, *T3*, and *T4*. For the sake of this example, dummy activities were used. Activity *T3* in A's process calls another enactment service (which is deployed on company B private cloud) to enact B's internal process to perform the outsourced verification job (B's process forms a transparent part of A's process). Company B has its own private cloud and it installs a private instance of the platform where the process will take place. The following sequence of events corresponds to the numbers on Fig 7.

1. Company A requests the enactment service to execute its main process.
2. Activities **T1**, **T2** are allocated to workflow engine 1 **WE1** which is deployed on cloud provider 1.
3. Activities **T3**, **T4** are allocated to workflow engine 2 **WE2** which is deployed on cloud provider 2. **T3** is a wrapper activity developed by company B. It makes the necessary calls and passes the required artefacts to B's enactment service to execute company B's process.
4. When **T3** is executed, it triggers the execution of company B's process on company B's private cloud.

This case study has been implemented using dummy activities to create the processes, however, it would work in the same way with real activities. This simple process demonstrates how the core functionality of outsourcing could be supported by our architecture.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we proposed our vision on how cloud can be used to support software development processes. The vision is to model software processes into executable workflows consisting of a set of activities. These activities can then be enacted in the cloud and utilize its elasticity (to perform computing intensive tasks, e.g. testing), availability and accessibility (to support collaboration and facilitate global development), and the different cloud deployment models (to support performing activities with special requirements, e.g. security and privacy).

We have developed a simple process modelling notation (and its meta-model) which avoids the complexity and formality of the current process modelling techniques and focuses on the executability of models. We have designed an architecture for the process enactment platform and implemented a first version of it as a simple case study. Furthermore, we have designed and implemented a case study demonstrating the potential of our platform in facilitating the collaboration not only within the same company, but also across companies. The main future work is to apply and evaluate it on several realistic case studies. The areas we are now looking at include development of business information and safety-critical systems.

Empirical studies are still needed to evaluate the impact this approach would make on software development practices in terms of: performance, quality, productivity, and cost. The process enactment platform represents a core that can be extended in the future. Currently, we are focusing on extending our implementation to support more types of activities; namely: interactive activities, control points, and sub-processes. Once those types of activities are supported, the platform will be able to execute more complex processes. An SDK will be developed to give users (individuals and companies) the ability to systematically create their specific activities (tools) and integrate them into the platform. The SDK will also support wrapping existing command-line tools and making them available as activities. These are some of the other areas we will be addressing in the longer term: mining the software artefact repository to automatically produce traces and evidences that can be used for safety certification processes, adopting a smart scheduling mechanism which allocates cloud resources and selects candidate workflow engines with consideration of increasing performance and reducing cost, and reviewing the current non-cloud tools (which support software development activities) pricing and licensing models and investigate the potential of pay-as-you-go models for software processes on the cloud.

## 7. ACKNOWLEDGEMENTS

We are grateful to Andreas Roth, Henry Muccini, Barbara Gallina and Jacek Cala for the useful feedback on the earlier versions of this paper.

## 8. REFERENCES

[1] S. Alajrami. On cloud-based engineering of dependable systems. *arXiv preprint arXiv:1404.7509*, 2014.

[2] S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the spade environment. *Software Engineering, IEEE Transactions on*, 19(12):1128–1144, 1993.

[3] J. Barnat, L. Brim, M. Češka, and P. Ročkai. DiVinE: Parallel Distributed Model Checker (Tool paper). In *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2010)*, pages 4–7. IEEE, 2010.

[4] A. Barnes and J. Gray. Cots, workflow, and software process management: an exploration of software engineering tool development. In *Software Engineering Conference, 2000. Proceedings. 2000 Australian*, pages 221–232, 2000.

[5] R. Bendraou, J. Jezequel, M.-P. Gervais, and

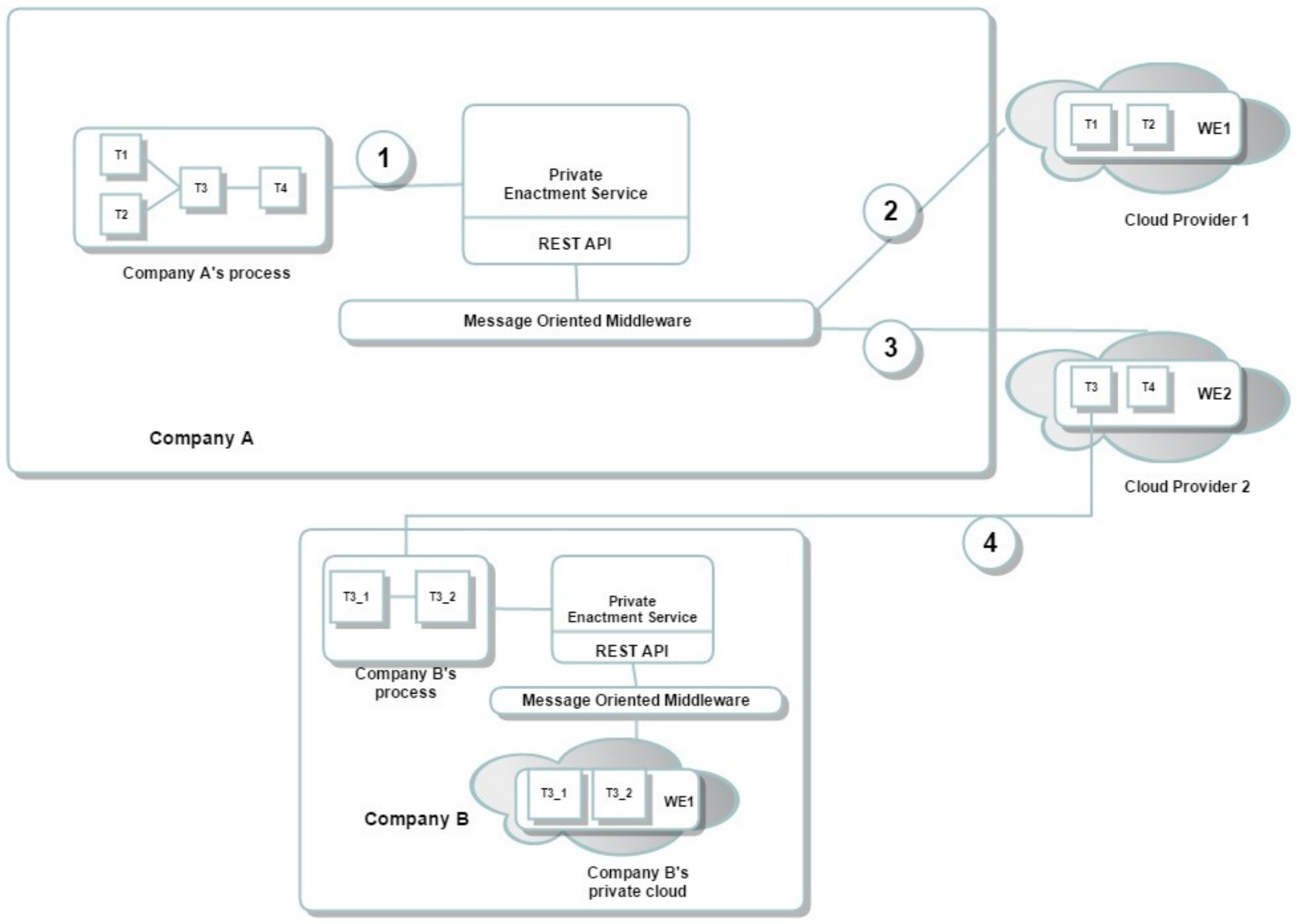


**Figure 7: Outsourcing case study**

X. Blanc. A comparison of six uml-based languages for software process modeling. *Software Engineering, IEEE Transactions on*, 36(5):662–675, Sept 2010.

[6] B. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[7] B. Boehm. Some future trends and implications for systems and software engineering processes. *Systems Engineering*, 9(1):1–19, 2006.

[8] S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel symbolic execution for automated real-world software testing. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 183–198. ACM, 2011.

[9] D. Chan and K. Leung. Software development as a workflow process. In *Software Engineering Conference, 1997. Asia Pacific ... and International Computer Science Conference 1997. APSEC '97 and ICSC '97. Proceedings*, pages 282–291, 1997.

[10] 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*, WICSA/ECSA '12, pages 5–13, 2012.

[11] B. Combemale, X. Crégut, A. Caplain, and B. Coulette. Towards a rigorous process modeling with SPEM. In *ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006*, pages 530–533, 2006.

[12] R. Conradi, M. L. Jaccheri, C. Mazzi, M. N. Nguyen, and A. Aarsten. Design, use and implementation of spell, a language for software process modelling and evolution. In *Proceedings of the Second European Workshop on Software Process Technology*, EWSPT '92, pages 167–177, 1992.

[13] R. Ellner, S. Al-Hilank, J. Drexler, M. Jung, D. Kips, and M. Philippsen. espem - a spem extension for enactable behavior modeling. In *Modelling Foundations and Applications*, volume 6138 of *Lecture Notes in Computer Science*, pages 116–131. 2010.

[14] R. Ellner, S. Al-Hilank, J. Drexler, M. Jung, D. Kips, and M. Philippsen. A fuml-based distributed execution machine for enacting software process models. In *Modelling Foundations and Applications*, volume 6698 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2011.

[15] Z. Galviņa and D. Šmite. Software development processes in globally distributed environment. *Baltic Journal of Modern computing*, 770:7–14, 2011.

[16] J. H. Gerard. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[17] B. Henderson-Sellers and C. Gonzalez-Perez. A comparison of four process metamodels and the creation of a new generic standard. *Information and Software Technology*, 47(1):49 – 65, 2005.

[18] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.

[19] G. Kaiser, N. Barghouti, and M. Sokolsky. Preliminary experience with process modeling in the marvel software development environment kernel. In *System Sciences, 1990, Proceedings of the Twenty-Third Annual Hawaii International Conference on*, volume ii, pages 131–140, 1990.

[20] P. M. Mell and T. Grance. Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[21] N. M. A. Munassar and A. Govardhan. A comparison between five models of software engineering. *IJCSI*, 5:95–101, 2010.

[22] A. Oberweis. Workflow management in software engineering projects. In *Proceedings of the 2nd International Conference on Concurrent Engineering and Electronic Design Automation*, pages 55–60, 1994.

[23] M. Oriol and F. Ullah. Yeti on the cloud. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 434–437.

[24] A. Pakhira and P. Andras. Leveraging the cloud for large-scale software testing - a case study: Google chrome on amazon. In *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, pages 252–279. IGI Global, Hershey, PA, USA, 2013.

[25] M. Paulk, W. Curtis, M. B. Chrissis, and C. Weber. Capability maturity model for software (version 1.1). Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, 1993.

[26] C. Portela, A. Vasconcelos, A. Silva, E. Silva, M. Gomes, M. Ronny, W. Lira, and S. Oliveira. xspider_ml: Proposal of a software processes enactment language compliant with spem 2.0. *Journal of Software Engineering and Applications*, 5(6):375 – 384, 2012.

[27] I. Richardson, V. Casey, J. Burton, and F. McCaffery. Global software engineering: A software process approach. In *Collaborative Software Engineering*, pages 35–56. Springer, 2010.

[28] L. Rodriguez-Martinez, M. Mora, and F. Alvarez. A descriptive/comparative study of the evolution of process models of software development life cycles (pm-sdlcs). In *Computer Science (ENC), 2009 Mexican International Conference on*, pages 298–303, 2009.

[29] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, 1987.

[30] W. M. C. Specification. *Workflow Management Coalition, Terminology & Glossary (Document No. WFMC-TC-1011).* Workflow Management Coalition Specification, 1999.

[31] F. Yuan, M. Li, and Z. Wan. SEM2XPDL: towards SPEM model enactment. In *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1*, pages 240–245, 2006.