F

Rational decomposition and orchestration for serverless computing

Deliverable D2.2

Final requirements

Version: 1.0

Publication Date: 30-June-2020

Deliverable Card

Deliverable	D2.2
Title:	Final requirements
Editor(s):	Alessandra Russo (IMP)
Contributor(s):	Matej Artač (XLB), Matija Cankar (XLB), Anže Luzar (XLB), Neža Vehovar (XLB), Giuliano Casale (IMP), Stefania D'Agostini (ENG), Stefano Dalla Palma (TJD), Chinmaya Dehury (UTR), Martin Garriga (TJD), Pelle Jakovits (UTR), Hans Georg Næsheim (PRQ), Dušan Okanović (UST), Satish Srirama (UTR), Damian A. Tamburri (TJD), Vasilis Tountopoulos (ATC), Andre van Hoorn (UST), Michael Wurster (UST), Vladimir Yussupov (UST), Lulai Zhu (IMP).
Reviewers:	Matija Cankar (XLB), Vasilis Tountopoulos (ATC)
Type:	R
Version:	1.0
Date:	30-June-2020
Status:	Final
Dissemination level:	Public
Download page:	http://radon-h2020.eu/public-deliverables/
Copyright:	RADON consortium

The RADON project partners

IMP	IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE
TJD	STICHTING KATHOLIEKE UNIVERSITEIT BRABANT
UTR	TARTU ULIKOOL
XLB	XLAB RAZVOJ PROGRAMSKE OPREME IN SVETOVANJE DOO
ATC	ATHENS TECHNOLOGY CENTER ANONYMI BIOMICHANIKI EMPORIKI KAI TECHNIKI ETAIREIA EFARMOGON YPSILIS TECHNOLOGIAS
ENG	ENGINEERING - INGEGNERIA INFORMATICA SPA
UST	UNIVERSITAET STUTTGART
PRQ	PRQ A/S

The RADON project (January 2019 - June 2021) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825040

Executive summary

The main objective of the RADON project is to develop an open source DevOps framework and tools that facilitate the adoption of serverless function-as-a-service (FaaS) technology in industrial software development. In the first year of the project, the consortium has identified the main features that the RADON framework needs to provide, and has developed, with the industrial partners, workflows on how RADON tools and users can interact during the development lifecycle to realise these features (cf. Section 3.2). Tool requirements have been specified together with usage scenarios and analysed through the tool application to the three RADON use case scenarios. This evaluation has identified and suggested through the course of the first year of tool development some changes to be made to the requirements as well as new requirements and functionality to be implemented in the tools. Evolution and changes have been managed through a flexible DevOps methodology that takes into account the typical characteristics of a DevOps approach to software development.

Interactions and interviews with third-party practitioners have also helped members of the RADON consortiums to identify gaps in the current DevOps state-of-the-art and reinforce the appropriateness and benefit that RADON features can bring to the industrial sector. Based on this work, this deliverable presents a comprehensive summary of the outcomes of these interviews and the relevance of RADON's features in the context of DevOps state-of-the-art from a practitioner perspective. The RADON features are then mapped into RADON workflows to provide a means for conceptualising and structuring tools requirements in terms of high-level features and users' needs. Updated and new tool requirements have also been presented together with evidence of the requirement management strategy adopted to handle these changes.

Glossary

AR	Application Repository
ASP	Answer Set Programming
AWS	Amazon Web Services
BAN	Body Area Network
CAPEX	CApital EXpense
CDL	Constraint Definition Language
CSAR	Cloud Service ARchive
DA	Deployment Artifact
FaaS	Function-as-a-Service
GMT	Graphical Modelling Tool
IDE	Integrated Development Environment
ILP	Inductive Logic Programming
ISV	Independent Software Vendor
IaC	Infrastructure as Code
KPI	Key Performance Indicator
NFV	Network Function Virtualization
OCL	Object Constraint Language
OPEX	OPerating EXpense
POI	Places Of Interest
QT	Quality Testing
RR	Robotic Rollator
SFC	Service Function Chaining
SaaS	Software as a Service
TOSCA	Topology and Orchestration Specification for Cloud Applications

Table of contents

1. Introduction

The main objective of the RADON project is to define an open source DevOps framework to facilitate the adoption of serverless function-as-a-service (FaaS) technology in industrial software development. In this deliverable, we specify the RADON methodology for managing requirements' evolution during open source software development and present a conceptualisation of the final set of RADON tools requirements by mapping them to the features and user requirements through the RADON workflow. This mapping provides also a means for maintaining traceability between high-level user requirements and tool technical requirements within the context of the different workflows' deployment of the RADON tools. The application of the RADON Requirements Evolution Management process is also documented for each tool, by identifying the requirements updates that have been informed by the lessons learned during the development of the tool prototypes. A full summary of the final tools and business requirements is also presented.

1.1. Deliverable objectives

The present deliverable has four main objectives:

1. Describe the RADON methodology for managing requirements evolution.
2. Identify the main high-level features of RADON and how they intersection with the value proposition of the framework.
3. Document the business workflows within the context of three RADON use cases and the way these are supported by the technical requirements of the developed RADON tools.
4. Present the final RADON tools technical requirements and document the application of the RADON requirement evolution process that has led to their final formalisation, starting from the initial draft of requirements given in deliverable D2.1[1].

1.2. Overview of main achievements

With the work done as part of this deliverable, the consortium makes several contributions:

1. The RADON process for managing requirement evolution has been formalised and its application throughout the development phase of RADON tools has been documented.

2. Lessons learned during the development phase of RADON tools have been identified, together with the changes in the initial tool requirements that they have informed. A companion document is supplied with a summary of tool-based requirements at M18, which are now tracked publicly on Github.

3. Conceptualisation of the tools technical requirements by linking them to the main RADON features and user's needs elicited and analysed through interactions and interviews with practitioners and external stakeholders. Traceability between high-level features and low-level tools technical requirements has been captured by mapping high-level features to business

[1] https://radon-h2020.eu/wp-content/uploads/2019/07/D2.1-Initial-requirements-and-baselines.pdf

workflow and orchestration of the tools needed to release such workflows.

1.3. Structure of the document

The rest of the deliverable is structured as follows.

- Section 2 provides a premise to the RADON value proposition by discussing the outcomes of interview and discussion activities conducted together with external practitioners and stakeholders in the 2nd project year and summarises the feedbacks of practitioners over the current DevOps state-of-the-art technology.
- Section 3 presents the RADON value proposition and positions it with respect to the practitioners' feedbacks by identifying the key RADON features that address the current gaps in the technologies and users' needs. Traceability between these key features is presented in terms of a mapping between RADON features and business workflows.
- Section 4 introduces the RADON requirements evolution methodology by positioning it within the context of existing literature in requirements engineering (RE) for cloud-based software development.
- Section 5 presents for each tool lesson learned during the development and evaluation of the tools within the context of the three use cases, and identification of new requirements and/or changes of existing requirements informed by these lessons learned.
- Section 6 summarises for each tool the full set of final requirements together with a diagrammatical view of how these requirements relate to the RADON features through the business workflows.
- Appendix documents the application of the RADON requirements engineering management process during the evolution of requirements.

2. User Involvement

2.1 Practitioner Feedback on State-of-the-art

As well established in previous RADON deliverables and related content, serverless computing, and its particular instantiation into FaaS, is a response to a limitation of the microservice model that operations still need to provision a cluster of compute resources to run (micro) services, then manage and scale these compute resources [Gannon17, Lynn17]. Even purely from an architectural perspective [Lenarduzzi2020] and considering the conceptual similarity between Serverless Functions and Microservice designs, we expect maintainability and continuous sustainability of serverless solutions to become a major and costly problem in the serverless domain as much as it is established to be in the microservices technical domain. To look further into these challenges, we conducted interviews and manual coding as documented [Lenarduzzi2020]. On the one hand, we held 7 interviews with serverless computing practitioners as well as a still running series of focus-groups with the OASIS TOSCA TC consortium in the context of the Emergent Compute Ad-Hoc sub-committee, to distil a set of basic challenges arising in the serverless technical domain, with the intent of achieving a more precise mapping of such challenges to the broader RADON value proposition. Also, we interacted with three scientists who are expert and active in the cloud-native architecture landscape, with the intent of gaining a more fine-grained overview of the state of the art beyond RADON. We point to [Lenarduzzi2020] for additional details not included in this deliverable.

Stakeholders, Concerns, and Challenges

From the practitioner perspective, seven major challenges emerged in the serverless computing technical space. They appear outlined in Table 1:

ID	Name	Description	#Practitioners
1	**Unsanctioned technology use**	Quoting from our report [Lenarduzzi2020], Serverless is more event-driven than traditional environments. This requires architecting the software correctly and with sanctioned and rigorously evaluated alternatives, in a way that is possible to exchange the correct type of events over correct managed services, by using the right operational approach.	3
2	**Knowledge churn**	Practitioners agree that *"When doing it [serverless] not-right, you are just getting to an overall technical debt, but with the right experience and knowledge over serverless*	5

		concepts as well as event-driven programming, you can avoid it”. This also leads to the necessity to find efficient and effective knowledge exchange mechanisms across DevOps teams as well as individuals.	
3	**Process automation not properly set-up**	DevOps and Serverless go hand in hand. More specifically, practitioners highlight that when too many resources are combined into single stacks, shared resources (like persistence layers, communication channels, and other shared infrastructure) become intertwined and hard to decouple and co-evolve. This in turn leads to churn and lack of value-add for customers. Then a challenge emerges to properly setup process automations around the decoupling, the resource and stack management, as well as general architecture maintenance.	7
4	**Secrets management**	Managing secrets authenticators for many and diverse serverless functions and the connected Secrets management, result in the inability (or inflexibility) of easily rotating credentials in external systems. This single issue would lead to broader security implications (discussed later) but in general also force some serverless solutions to actually become server-*full*. This requires using hosted services to augment applications capacity in security—think DynamoDB for safe data storage or Mailchimp for more fine-grained email management.	3
5	**Security**	Bypassing Identity and Access Management (IAM) to speed development could create considerable lack of value, for example in the erroneous design, implementation, and/or management of critical system functions, as opposed to the management of other more regular functions. At the same time, the lack of knowledge of the developers about Security in the serverless domain and how it shall be managed is also an open issue, as highlighted by as many as four practitioners of the five.	5

6	**Testing activities postponed or not completely performed**	Postponing testing or not testing completely a serverless application, especially at integration level, hinder the verification of the entire system operability. A clear consequence is that the system does not correctly respond or does not comply with the requirements, yielding to problems that developers do not know exist.	6
7	**Inadequate or verbose logging**	Distributed serverless applications produce massive logging data. This leads to the accumulation of a lot of disparate and disconnected data, difficult to manage, and to the need for extra data fusion effort.	6

Table 1: Stakeholders perspective of major challenges in serverless computing

From an academic and research perspective, three key challenges emerge from these interviews. First, Serverless is in its infancy and hence researchers and practitioners have proposed only a few sets of best practices for its adoption and operations, (e.g., [Baldini2017]). Secondly, only very few attempts were made at defining design principles and patterns for composing and triggering serverless functions (e.g., [Leitner2019]). Finally, only a few researchers have started defining what bad practices (nor connected actionable metrics) exist in the digital-native technical domain and that should be amended [Nupponen2020, Leitner2019]. These limitations reflect a general lack of maintenance and evolution support for state-of-the-art serverless solutions across the entire DevOps lifecycle, which therefore leads to a serious lack of theoretical and technical support to service operation and continuity of serverless solutions. As discussed in Section 3, RADON addresses this need with a systematic framework that addresses the entire DevOps lifecycle.

2.2 RADON Industrial Use-Cases: Challenges

Stemming specifically from the work done previously in the scope of Deliverable D6.1, this section summarizes the usage-scenarios highlighted by RADON use case owners specifically remarked in the context of their respective cases. Individual usage-scenarios were consolidated with a simple manual card-sorting method [Lewis2010] and reported in Tables 2 and 3.

ID	Usage-Scenario	Description
8	**Optimising the current architecture to address different scales of the customer size**	RADON use case owners have diverse needs coming from often completely orthogonal customer-bases. These needs force them to have to deal with different architecture scales and structures. Current support only focuses on stove-piped decomposition from a functional and dependency perspective, disregarding essential architecture qualities such as cost-efficiency, scalability.

9	**Advancing automation in integrating services for processing content from various online sources**	RADON use case owners work with distributed and global software and services engineering and delivery and therefore need to cope with concurrent content processing as well as heterogeneous online source blending. Currently, support for this feature is not present at all.
10	**Managing secure data exchange with 3rd parties to accomplish business functions**	RADON use case owners identify a need to manage secure interaction since their architectures need interplay with external and third-party services processing privy and sensitive data; this means that security is of high-priority and current tooling does not allow them to cater for secure data exchanges if not with specific and ad-hoc point-to-point middleware.
11	**Accelerating code development and enhancing quality in developing new functionality**	RADON use case owners remark that the solutions targeted by RADON exploitation are subject of continuous and often disruptive architectural changes---also known as Continuous Architecting [Bersani2016] in the state of the art---with little to no support in the state of the art at all, especially not for Function-as-a-Service.
12	**Managing several DevOps teams in maintaining the production and new research features**	Similarly to usage-scenario #9, this scenario foresees Global Software Engineering around solutions currently catered by RADON end-users; the production line therefore poses critical challenges in concurrent and rapid code commits across a very diverse and microservice-based pipeline which is committed to live, during operations and often without a staging environment. It is currently impossible for them to support such concurrent and distributed commit operations over live non-staged service solutions.
13	**Managing concurrent software releases for two different business cases**	RADON use case owners could not find support in the state of the art to manage concurrently the business value coming from at least two concurrent software releases aimed at providing their customers with specific value-generating functions in different and concurrent versions.
14	**Advancing deployment automation level at customer site**	RADON use case owners identify a gap in the unavailability of vendor-unlocked support for customer site DevOps toolchain automation optimisation, specifically, they identify a lack of non-vendor-locked solutions to quickly and painlessly support microservice operations at customer-sites. Their customers are often non tech-savvy or with low DevOps IT knowledge and need easy-to-use off-the-shelf solutions which are flexible enough to enter the customer baseline painlessly.
15	**Monitoring and optimizing cloud resources in an existing production environment**	Although Monitoring---and recently observability [Tamburri2018b]---is subject to research in the state-of-the-art, RADON end-users report no continuous optimization monitoring

		opportunities---intended as the ability to continuously monitor the effect of adopted best-practices (e.g., cost-effectiveness) and optimizations (e.g., architecture decompositions' effectiveness)---available to them as of yet.
16	**Managing dynamically function operations based on customer requirements**	RADON use case owners remark that with current stateless computing possibilities they are still not able to manage dynamic requirements from specific customers since these would require continuous and specific updates on many functions which also incurs high costs.
17	**Customise based on geographical location**	RADON use case owners remark the inability to customise their service offer based on specific geographical locations that their clients are providing continuously. Although modern FaaS provisioning platforms allow for such support in principle, it's difficult for them to define policies which can be reusable and applicable to more clients and more FaaS at the same time and dynamically.
18	**Managing cloud security and integrity with serverless functions**	Much like challenge #17, RADON end-users remark the need to gain a more fine-grained and reusable way to handle security, possibly in a policy-based fashion, using an abstract representational format which they can show to resident customers for further security-based reasoning and policy-making sessions during architectural design.
19	**Serverless implementation of new functionalities**	Finally, RADON use case owners remark on the need to quickly realise and deploy customer-driven and value-add functionalities which are often specific to a single customer and that the customer wants to manage as easily as possible and as directly as possible. Given that Serverless Computing promises match these descriptions, RADON end-users want a painless way to implement new functionalities for customers as serverless functions to be ported to customer sites easily and in a straightforward operational manner.

Table 2: Usage-scenarios highlighted by RADON end-users

2.3 A RADON end-user feedback: Philips SSP

In period 2, the consortium has received several interesting feedbacks and organized multiple conference calls with an early adopter within Philips (Netherlands) who is trialling features of the RADON framework to develop a production application[2]. This direct user engagement has helped the consortium steer the prioritization of some technical activities and updating the requirements to better align to the end user needs.

[2] See for example the thread in: https://github.com/radon-h2020/radon-methodology/issues/11 Other threads with this unit are scattered across the Github repos.

The goal of the end-user project is to develop a machine-learning (ML) recommender system for production. The functional details of the application are confidential, but the intended use is to facilitate supplier management through automatically trained ML models that can be interactively queried by the responsible staff via a web UI. The underpinning technology stack involves a combination of microservices, databases, and machine learning training services, making use of diverse cloud technologies over multiple cloud providers (in particular, AWS and Google Cloud Platform). Their goal is to use RADON to automate the delivery of the solution and RADON models are seen in particular as a good way to internally document the application so to ensure its long-term maintainability,

Following initial contact, we have carried out an interview to identify relevant requirements with this end user in terms of supporting the definition of a microservices based application.

ID	Usage scenario	Description
20	**Ensuring high-quality documentation and adherence to design best practices**	Not all software engineers in a large multinational will be academically trained as computer scientists and this means that it is sometimes difficult for them to understand how to properly structure code, how to design an architecture, and how to do so in a way that the person that comes next on board in the project will be able to follow the same design principles. It is perceived by the end user that simplicity in the DevOps toolchain could play a role in addressing this need, while also ensuring maintainability.
21	**Using platform services from multiple clouds**	Constraints will arise in a real-world application in terms of location of the services in particular clouds, e.g., due to the inability to migrate the data to a different country. However, while the data may reside with a certain cloud operator (e.g., Google Cloud Platform) some services may be preferred for use from the platform of a different operator (e.g., AWS Lambda). This calls for the ability to deploy and orchestrate the microservices based application with an architecture that spans multiple clouds, which may benefit from the RADON orchestration features.
22	**Designing for cost efficiency**	Teams can have fixed annual budgets to operate certain applications in the cloud, however it tends to be difficult to understand at design time the long-term implications of a design in terms of its cost. There is therefore a need to estimate the expected cost of a particular design decision prior to implementation or refactoring.
23	**Monitoring functional progress**	In addition to monitoring the performance of an application, there is often the need to monitor progress in the functional application activities themselves (e.g., the level of progress of training for a ML model). It is therefore important to be able to extend a monitoring platform with custom data to track the application

		progress.

Table 3: Usage-scenarios highlighted by RADON end-users

Overall, the above feedbacks lead to drawing some lessons on the RADON framework requirements:

- We have recognized the importance for end users of the particular selection of technologies available within the framework. For this reason, we have decided to revisit in deliverable D2.4 the table of supported technologies as this was important to the end users and adopt as one of the target deployment platforms Google Cloud.
- This experience and the simplicity requirement put a strong emphasis on the importance of graphical-modelling for the end users without a strong IT background. Because the AI/ML application market is really about data, this has led us to recognize the importance of supporting mature data pipeline modelling within the Graphical Modelling Tool and to add a requirement to include support for data pipelines that acquire data from data sources we have not considered earlier such as Google Big Query.
- Deployment and operation costs can be predicted in RADON by the decomposition tool, the above requirements indicate the need for developing a methodology to look at cumulative costs over a time horizon (e.g., a year), based on expected workload profiles. This has been integrated in requirement R-T3.2-4.
- Lastly, monitoring tool support needs to be customizable and guidelines should be provided to the user since this is a task that needs to be implemented by the end-user itself, depending on the functional characteristics of the application. This is covered by requirement R-T5.1-16.

3. RADON Value Proposition

As previously mentioned, the RADON initiative aims at providing a DevOps framework to help Independent Service Vendors (ISVs) adopting serverless FaaS solutions avoiding vendor lock-in, while guaranteeing specific non-functional operational parameters.

3.1 The RADON Value Space

The RADON solution promises to be an **Open-source free-to-use** framework with loosely coupled, **event-driven**, **platform-agnostic** and **containerised** architectural structure, thus offering **maximum flexibility** to be embedded in **previously existing DevOps pipelines,** as well as enabling speedy assembly of new FaaS DevOps pipelines. The RADON framework will offer support to many possible design, development, and operations software architecture viewpoints [Kruchten1995, ISO[3]]. More specifically, Figure 3.1 below outlines the RADON value space as part of the DevOps lifecycle, highlighting RADON intended users, areas of value generation as well as intended interoperation across stakeholders.

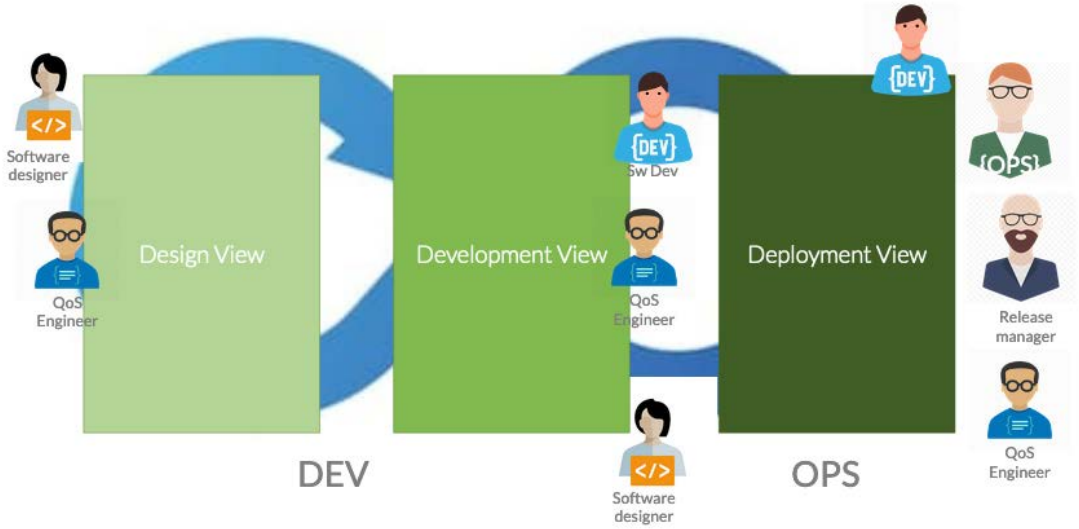

Figure 3.1. The RADON Value Space.

The figure shows software designers and Quality-of-Service (QoS) engineers interacting heavily during **service design**, with even higher interaction during **development**. Similarly, -Ops roles such as release manager and operations systems administration analysts interacting in their **deployment** and operations specific activities. In essence, the RADON value space highlights three essential software architecture views, through which RADON brings about its value, together with several actors for which that value is intended. These views are as follows:

[3] https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-1:v1:en

(1) Software design view. RADON offers advanced facilities for verification of infrastructure blueprints as well as their design-time verification, with a special focus on expensive constraints, such as privacy and security, as well as design-pattern violations which are value-detrimental and costly to address at a second stage. Similarly, RADON offers Pattern-based design assembly and reuse as well as architectural decomposition with component substitutability reasoning, directly in the RADON graphical-modelling tool, where the aforementioned policy definitions and their reuse can be planned.

Design Value Potential. Compared to existing solutions in this space, RADON is the first environment to offer a modelling language to describe, annotate with design constraints, and verify applications including serverless functions.

From a Design perspective, the value RADON offers maps well with several challenges from the state of the art, specifically, with challenges #1 and #2 which reflect knowledge concentration and trade-off analysis mechanisms, available and easy-to-use, for practitioners in their design endeavours.

(2) Software development view. From a development perspective, following the established tradition of quantitative analyses and data-driven software engineering, RADON focuses on offering design-to-runtime defects and anti-patterns analysis tools. These tools are focused on analysing and scrutinizing infrastructure-code, applying statistically established quantitative models and thresholds for quantitative defect prediction. This allows practitioners to identify weak spots in their designs immediately before actionably deploying their solution or effectively during integration testing of their application itself. At the same time --- that is, during the testing phase --- practitioners reflecting on their FaaS solutions and how to test them can use RADON to rig their continuous testing pipelines for end-to-end continuous testing, which are fully supported in RADON and intended to be used as part of a continuous DevOps loop.

Development Value Potential. Compared to existing solutions in this space, RADON is the first environment to offer a model-based testing environment that covers infrastructure-as-code defects as well as load testing of functions and their data pipelines.
From a Development perspective, the value RADON offers maps well with challenges #3-#6 by offering metrics-based tools which allow identifying weak spots in serverless applications. Such spots will need to be subject to more focused testing, thus saving downtime and future maintenance costs during service operations or even before, during blueprint preparation and orchestration. Finally, the RADON verification first, and continuous testing second, offer specific support to advanced and continuous testing demanded in the aforementioned challenges.

(3) Software deployment view. From a deployment perspective, the RADON solution makes it simple to automate FaaS deployment as well as its blending into previously existing solutions and the pipelines around them by providing a flexible, free-to-use, pluggable, containerized and

integrated development environment for cloud-native [Leitner2019] software architectures, with a focus on FaaS. Similarly, specifically designed monitoring facilities for FaaS offer the ability to link continuous online feedback with immediately actionable insights at both design and development stage, for de-facto continuous evolution.

Deployment Value Potential. Compared to existing solutions in this space, such as the Serverless framework or the Sigma framework, RADON offers a broader set of composable and extensible templates for microservices and serverless functions with a high-level of support for Ops features and also covering data pipelines.
From a Deployment perspective, the value RADON offer maps well with challenge #7, as it provides specifically-designed monitoring facilities --- which simplify logging and log management with a single-entry platform --- as well as a stand-alone integrated development environment where feedback from online systems orchestration can be made actionable at several levels of granularity.

3.2 Key features

Feature	Novelty	Benefits for adopter	Limitations	Addresses Feedback
Constraints specification and design-time verification	Annotations of entities, events and data with requirements constraints (e.g., security, privacy, performances, costs) and verification before deployment.	Ability to reasoning about correctness of design and satisfiability of critical requirements yielding to robust design, implementation, and/or management of critical system functions.	Lack of automated support for constraints verification in DevOps.	#1, #5, #10, #18
Integrated application modelling, with FaaS, data pipelines and microservices	Model-based approach to orchestration and management of cloud applications, with microservice architecture and FaaS services	Continuous update of architectures through assembling and reuse pattern-based design. Customisation and reuse of services through policy definitions.	Lack of support for serverless FaaS and data flow in basic orchestration of cloud applications	#11, #16, #17, #18
Architectural optimization	Optimality of functional decomposition,	Decoupling of teams and their updates to production	Monolith needs to be modelled and decomposition is only	#3, #7, #8

	interoperability of software and data.			
Design-to runtime defect and anti-patterns analysis	Defect prediction solutions for IaC, to detect codes prone to defects.	Improved quality assurance for IaC	Shortage of debugging tools for Infrastructure-as-Code	#6, #10, #11, #18
End-to-end pipeline testing	Automated support for testing guided by monitoring data, prior deployment.	Improved system operability and staging environment for run-time relevant data collection and detection of performance issues.	Poor quality assurance of distributed serverless applications due to overlogging and lack of clear traceability between non-functional requirements and current usage of the software.	#6, #7, #9, #11
Automation optimization	Runtime centres for deployment and orchestration of serverless FaaS and data pipelines, as a continuous cycle of development, testing, integration, deployment and provision of applications.	Management of runtime lifecycle of the applications in a self-service fashion by means of a library of reusable and actionable templates (e.g. function hub), orchestrator plugins for serverless FaaS and data pipelines, and monitoring mechanisms for security and privacy constraints.	Lack of tools for secure interactions and data exchange, for continuous optimisation monitoring, and for quickly identifying value-added functionalities for customers as serverless FaaS.	#10, #13, #15, #19, #23
Integrated Development Environment	Development environment for multi-user usage for development activities, and shared space where the models, code and data can be securely accessed within a team and/or across	Ability to use facilities such as support for different program languages, debug functionalities, source code editors with error checking capabilities during within team and cross-team development	Lack of distributed and global services engineering, which hinder the advancement of deployment automation at customer-sites.	#9, #12, #14

	teams.	activities.		

3.3 Mapping key features to workflows

This section maps the key RADON features summarised in Section 3.2. to the RADON workflows which capture, at an abstract level, how RADON tools and users can interact during the development lifecycle of an application with RADON. The detailed description of the RADON workflows is given in deliverable D2.3 (see Table 8) and an updated version deliverable D2.4 and it is therefore omitted in this section. We simply remind the reader that the RADON framework includes six workflows: (1) verification (2) decomposition, (3) defect prediction, (4) continuous testing, (5) continuous integration and continuous deployment (CI/CD), and (6) Monitoring.

Feature	Workflow	Tools involved	Actors
Constraints specification and design-time verification	• Verification	• Constraints Definition • Verification Tool	• Software Designer • QoS Engineer • Ops Engineer • Release Manager
Integrated modelling, with FaaS, data pipelines and microservices	• Verification • Decomposition • CI/CD • Monitoring	• Graphical Modelling • Constraint Definition	• Software Designer • QoS Engineer • Ops Engineer • Release Manager
Architectural optimization	• Decomposition	• Decomposition tool • Monitoring tool • Graphical modelling tool	• Software Designer • QoS Engineer • Ops Engineer
Design-to runtime defect and anti-patterns analysis	• Defect Prediction	• Decomposition tool • Monitoring tool • Graphical modelling tool	• Developer • Ops Engineer
End-to-end pipeline testing	• Continuous Testing	• Continuous testing tool • Monitoring system • RADON IDE	• Developer • QoS Engineer • Release Manager
Automation optimization	•CI/CD •Monitoring •Continuous Testing	• Continuous testing tool • Monitoring system • RADON IDE • Graphical modelling tool • Orchestrator • Decomposition Tool	• Developer • QoS Engineer • Release Manager • Ops Engineer
Integrated Development Environment	• CI/CD • Continuous Testing	• Graphical modelling tool • Orchestrator • Continuous testing tool	• Developer • QoS Engineer • Release Manager

	• Verification • Defect Prediction	• Monitoring system • Verification Tool • Defect Prediction Tool • RADON IDE	• Ops Engineer • Software Designer

4. Tool Requirements Evolution Methodology

In this section, we outline the methodology and tools adopted by the RADON project to handle requirements changes and evolution. We begin with reviewing general issues related to requirements evolutions, as highlighted in existing literature and empirical studies (e.g. [Jan13]), with particular focus on open source software development where requirements volatility and uncertainty is generally high (Section 3.1). We then discuss how these general issues have been taken into account within the RADON project, by presenting the methodology and software solutions adopted in RADON to handle requirements evolution (Section 3.2) of the tools developed as part of the general RADON framework. An overview of the RADON tools technique requirements updates will be presented in Section 4.

4.1 Overview

Requirements of software systems are generally volatile: they are likely to change during the software development process [Jan13]. Managing changing requirements is key and critical for the success of a software project, as they have impact on costs and time. Different strategies have been proposed, in the literature, on how to handle changing requirements. These can be classified, loosely speaking, into *defensive* and *reactive*, the former aiming at reducing or completely avoiding changes, the latter targeted at defining solutions for supporting changes in requirements with minimal cost and effort. Within the context of open source software development, where the volatility is high, reactive strategies are clearly most appropriate, as they allow for more flexibility. To implement a reactive strategy, it is important to identify the primary factors for requirements variability. Although problems such as incomplete knowledge and difficulty in communication, when recognised, could help improve the requirement definition process at the start of a software development and therefore reduce changes (i.e. defensive strategy), some requirements changes cannot be avoided. Examples include changes caused by increased understanding and experience of both the stakeholders and the customers during the development process. Solutions, of which some are software-based, should therefore be provided to address requirements volatility.
These solutions are normally grouped into three main categories:

1. *Anticipation capabilities*, that is capability to anticipate information in the early stage of the development. Such capability can be realised by supporting mechanisms for
    a. transferring experience from previous similar projects,
    b. allowing the involvement of all stakeholder and end users, by means of incremental delivery of prototype of the application to elicit feedback at every release.
    c. using prototyping at early stage of the development to facilitate proactive thinking

2. *Flexibility of the software*, that is adopting modular design of the software solution to help confine changes to parts of the software, hence reducing the impact of the changes on the rest of the software.
3. *Flexibility of the development process*. This can be achieved by:
    a. overlapping development activities [Ver99],
    b. adopting an incremental delivery or an iterative development process [Har92],
    c. involving flexible resources in the product development process [Ver99].

To make the development process flexible from a requirements management perspective, it is important to (i) define a *formal change request procedure* that is followed by all stakeholders involved in the development process, and (ii) introduce a *requirements traceability process* as an integral part of the development process. These two software aspects are particularly relevant in the context of industrial open source software development.

4.2 Requirements Management

The RADON framework involves a collection of tools that can be used at different stages of a serverless application development and the definition and management of requirements of such tools has been one of the core activities of the RADON project. Specifically, **defensive** and **reactive strategies** have been deployed from the start of the project.

Defensive strategy activities

As part of a defensive strategy, the RM methodology conducted, at the inception stage of the project, activities for guaranteeing an **in-depth understanding** of the software development project in hand, existing **codebases** and **industrial needs**, and **discussions meetings** with **external stakeholders**, whose inputs have been taken into account during the definition of RADON tools' main requirements.

Specifically, to gain knowledge and awareness of existing projects and serverless technology landscape, related to the RADON methodology, in-depth investigation and **survey** were conducted in the first stage of the project, identifying projects and existing solutions related to different aspects of the RADON framework, e.g. TOSCA for modelling software services, testing platforms and DevOps methodology (summary of the outcomes of this activity is given in Table 3.2 in D2.1). Similarly, existing serverless technology and tools, related to the RADON framework, were identified and evaluated (summary given in Table 3.3. of D2.1). This has helped (i) **reduce the risk** of **knowledge gaps** and **lack of information** by team members of **current technology landscape**; (ii) **identify technology gaps** and take them into account during the definition of tool requirements so to avoid the risk of external unforeseen technical obstacles for the realisation of such requirements and at the same time gain confidence in positioning the RADON tools within the landscape of existing codebases and industrial needs. External stakeholders and practitioners have also been involved from the start in the definition of the requirements for the RADON tools, through discussions and the use of a running toy example of a "thumbnail generation" serverless application that illustrates what the RADON framework is meant to offer in *practice*, making it accessible to RADON external stakeholders. The involvement of external stakeholders in the

definition of tool requirements has helped reduce the **risk of lack of information** by RADON actors and external stakeholders **of respective objectives and needs** and at the same time establish a communication channel with external stakeholders which has been maintained since the inception stage of the project.

The above-mentioned activities have supported RADON requirements elicitation process, which has led to solutions upon which a structured process for management of requirements changes and evolution have been built. Key outcomes of the requirements elicitation process have been:

- Establishment of a **shared requirement repository** supported by the definition of **specific templates for uniformly representing standardised and comparable requirements,** and automatically generating (updated) companion documents with list of tools, tool requirements and use case requirements.
- Definition of **RADON reference actors for the DevOps methodology**, which helped contextualise the definition of the requirements.

Reactive strategy activities

Because of the DevOps-inspired methodology of the RADON framework, the boundary between development and deployment/operation is not clearly defined. The delivery on the final infrastructure often follows an agile approach, supported by continuous integration and continuous development (CI/CD) and orchestration of RADON tools. This agile aspect is also reflected in the requirements engineering of RADON tools: steps of development, integration and testing of these tools are likely to identify changes and evolution of requirements as increased knowledge and expertise of the consortium team is gained throughout the project and continuous feedback from external stakeholders is provided during the review stages of the RADON framework development and its usage scenarios.

Reactive strategy activities have been put in place to manage changes and evolution of requirements. These are described below, following the categorization given in Section 3.1.

Anticipation capabilities

a. **Transfer of experience from previous similar projects**. An in-depth study of existing related projects and tools that can be re-used has been conducted at the beginning of the project. A summary of the relevant existing projects and of existing tools that can be reused and/or built upon in RADON have been listed in Deliverable D2.1 (see Section 3.2).

b. **Involvement of all stakeholder and end-user**s.

External Stakeholders. Communication with external stakeholders and 3rd parties has been maintained since the start of the project. Feedbacks have been taken into account during the qualitative analysis of the requirements constituting integral part of the justification for change, for example as part of the interaction we have having with the Philips SSP unit.

Standards Board Engagement. The RADON consortium has maintained a steady and continuous flow over a considerable bulk of RADON tools in the scope of its constant

participation to the TOSCA technical committee activities and in the scope of the Emergent Compute Ad-Hoc subgroup which the RADON consortium members of OASIS TOSCA are chairing. As part of that engagement, the RADON consortium has offered presentation pitches and technical overviews over: (a) the overall RADON Solution and Framework; (b) the RADON Serverless Compute Profile; (c) the RADON Defect Prediction framework. Direct feedback stemming from this empirical confirmation was used to both evolve the requirements and evaluate the practical applicability of the aforementioned RADON assets.

c. **Proactive thinking**

During the development of the RADON tools, individual tool owners have conducted a qualitative analysis of the current version of their prototypes against the defined requirements. This qualitative analysis has been conducted after each tool release version. It consists of (i) **applying and testing the tool over common use cases** and where needed over stress-test cases specifically developed for this analysis, as well as (ii) presenting and discussing the tool with RADON use case providers and external parties. This **qualitative analysis** activity has facilitated **proactive thinking**. In most cases, the outcomes of this **qualitative analysis** have led to **updates of the current requirements** ranging from revision of deadlines, requirements priorities (if any) and refinement (specialisation) of existing requirements. In some cases, **additional new requirements** have also been identification and formalisation, as a result of the analysis. Section 5 describes for each RADON tool prototype the outcomes of this qualitative analysis in terms of lesson learned, the impact that such analysis has had on the tool requirements (e.g. type of change and/or evolution) with related rationale and justification for change.

Flexibility of the development process.

The RADON development process has been designed to be flexible enough to accommodate changes in the tool's requirements, whilst maintaining traceability with the business workflows in which the tools are used, and the versioning of the tool prototype.

a. **Managing change of requirements**.

The **shared requirements repository** allows to **track changes and evolutions** of the **requirements**. A formal process has been put in place supported by the software features of the GitHub repository. Specifically, when a change on a tool's requirement has been identified, the tool owner issues a request for change associated with the specific requirement in the repository. The Github notification dashboard of the project coordinator, who is the project repository administrator, lists such changes and a **formal approval for change or to close a requirement** is filed by the tool owner to the coordinator who reacts to the comments on Github to approve the change or closing. A similar procedure is also used for addition of new requirements. The dashboard and the approval processes are illustrated in Figures 4.2.1 and 4.2.2.

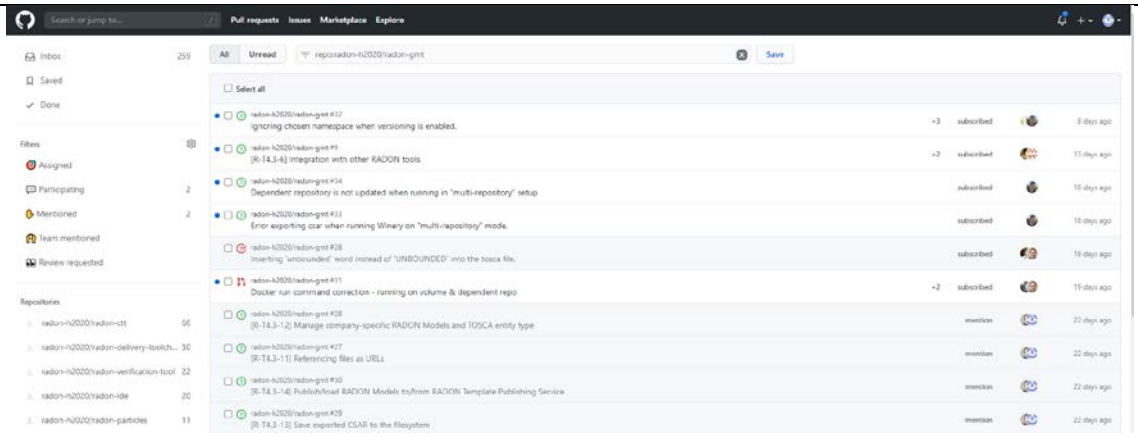

Figure 4.2.1. Github notification dashboard of the coordinator showing new and updated requirements [R-T4.3-…]

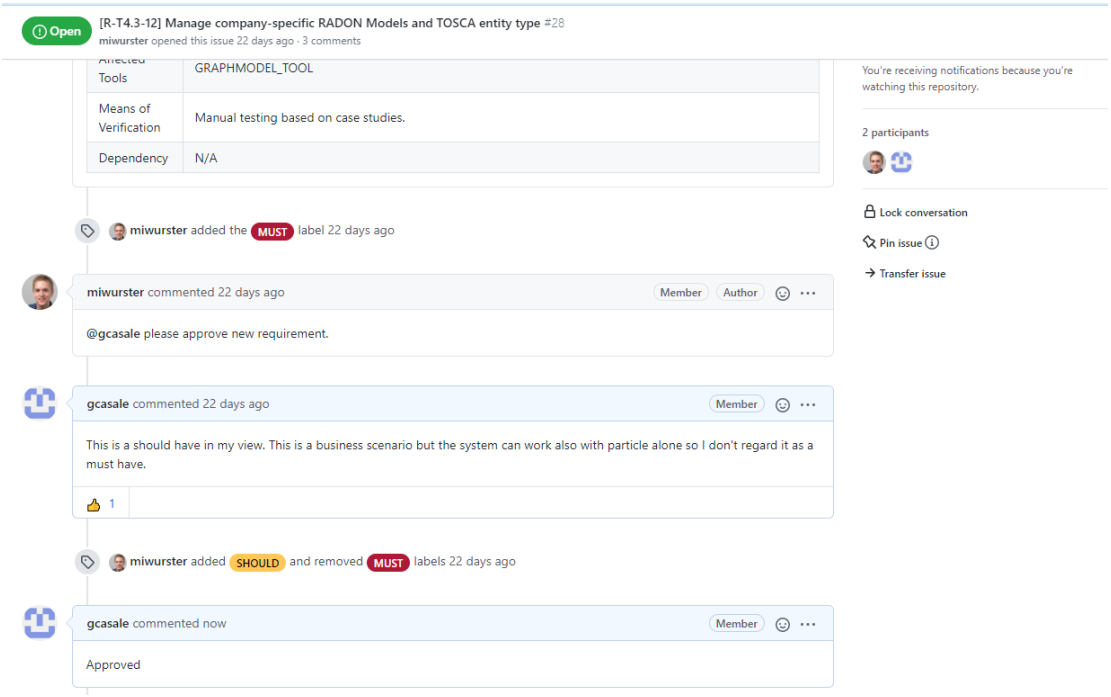

Figure 4.2.2. Requirement approval process in action.

Requests are issued by adding a comment associated with the requirement to change (or newly added), which is then reviewed by other members in the RADON team in particular those who are owners of other tools which might be affected by this change as well. **New comments and associated issues** may be raised by other team members in response to a change request that may have impact on other tools. So, a change request may trigger a sequence of additional change requests across the different tool owners. The status of the **repository** provides an **updated report** of the resolution of the raised requests and fulfilment of the changed requirements in terms of new committed versions of the tools. A request is resolved when approval has been given, related open issues have also been resolved and no new related open issues have been raised. Maintaining an updated status of

the requirements repository enables an analysis of the current project experience which is used to anticipate the rate at which the requirements are met and what is realistically achievable.

b. **Tracing requirements**.

The second important aspect of the RADON methodology for managing requirements evolution is **traceability of requirements**. This is by means of the **timestamps of the issues** and **comments raised in the repository**. Traceability is also from business workflows to the technical tool requirements. This is done by first identifying the business workflow activities which the business feature is decomposed into and then map each of these activities to the technical requirements of the tools that support these activities. Changes in the requirements are linked to the existing tool requirements they refer to so guaranteeing traceability between requirement evolution and business workflow. A description of this traceability is given in Section 2.4.

Traceability between **requirements changes** and **business scenarios** is also maintained through a mapping between tools and business scenarios and the identification of the specific requirements of the tools that support the needs of the business scenarios. The association between existing requirements of the tools and suggested changes enables to maintain traceability between the evolution in the requirements and the business scenarios.

Flexibility of the software

The **flexibility** of the RADON architecture allows also for better management of the requirements evolutions. As RADON tools are normally “composed” as function-as-a-service (FaaS) technologies, the impact of changes on a tool’s requirements over other tools tend to be limited. Where such impact is present the traceability mechanisms through the shared repository described above provide the means for minimising unforeseen side effects of requirements changes.

5. Tool technical requirement update / Lessons learned

5.1 Constraint Definition Language (CDL) and Verification Tool (VT)

In addition to the requirements set out in Deliverable 2.1, we have added three new "could have" requirements (R-T4.1-12 to R-T4.1-14) regarding the learning of CDL specifications. It was always envisaged that because the VT uses Answer Set Programming (ASP) technology to verify CDL specifications, and because there are existing tools to learn Answer Set Programs, it might be possible to learn CDL specifications in the future. We left this out of our original requirements, as it was not discussed at proposal stage, and because we were unsure whether the ASP programs used by the VT in verification mode would be a pure (single-shot) ASP approach, or a more modern multi-shot[4] ASP approach, which is not supported by the current state-of-the-art systems for learning ASP.

Now that we have completed the verification mode of the VT, it is clear that only a single-shot ASP approach was necessary; hence, the programs used by the VT are (at least in theory) learnable. In practice, we may still find that current state-of-the-art learning systems are not able to learn these programs in a reasonable timeframe. For this reason, the new requirements are given the label of "could have".

Requirements R-T4.1-12 and R-T4.1-13 are concerned with extending the input language of the CDL, allowing it to represent examples for learning (in the form of valid and invalid RADON models) and a space of constraints within which to search for a learned specification. Requirement R-T4.1-14 is about extending the VT to enable it to solve this new learning task (searching for a set of constraints which correctly classify the examples of valid and invalid RADON models).

5.2 Continuous Testing Tool (CTT)

The requirements for CTT comprise (i) the use cases/usage scenarios and (ii) the enumerated list of requirements following the agreed schema, both detailed in D2.1. Right after the first requirements deliverable and during the work on the architecture, we started developing an early prototype version of CTT focusing on the microservice/FaaS testing functionality. From the beginning, we (i) used the two RADON demo applications Thumbnail Generator and SockShop, (ii) interacted with the use case providers from ATC and PRQ, (iii) loosely evaluated interactions with the RADON tools GMT, CI/CD, xOpera, and Monitoring, as well as the Particles template library, and were in exchange with external experts in load testing.

Overall, we did not see a need for major changes in the requirements. The two most significant changes are:

- It was initially expected that CTT data pipeline module would have to ingest and parse the log data collected from the data pipeline under test to compute metrics but monitoring

[4] This involves adding an imperative program to the ASP program which manipulates the ASP (adding or deleting rules). Current systems for learning ASP only support learning a fixed ASP program.

technologies like Prometheus can generate metrics based on dynamic queries. Thus, the need to ingest and parse log data was removed from requirement R-T3.3-4.

- In the requirement R-T3.3-5, it was defined that the graphical representation of the data pipeline performance metrics should be displayed by the testing tool, however, as the visualization scope is outside the actual testing process and its priority was low we have decided to drop this requirement. For the exact same reason, we also considered dropping the requirement R-T3.3-6, which specified that it should be possible to specify upper and lower bounds for the performance metrics to be able to visually indicate when metrics of the data pipeline under test go beyond the expected range. However, the consensus was to keep this requirement, as it can be fulfilled by introducing additional configuration parameters and may influence long term development of the tool.

As one reason for seeing no major changes, we see the fact that we have intentionally kept the requirements on a technology-agnostic level. We noticed that we could gradually evolve the requirements situated in the problem space to architectural and technological decisions in the solution space. Further development was then based on deriving actionable issues that contribute to the respective upstream requirements.

The issues, along with the requirements, are being tracked in CTT's project dashboard.[5] For instance, R-T.3.3-10 demands "the generation of test cases from RADON models that are augmented by the test-related annotations". R-T.3.3-10 does not request *how* the test-related annotations are expressed in the models. Based on the decision to use TOSCA policies and integrating CTT's annotations into the Particles template library, issues such as radon-h2020/radon-ctt#60 have been created. As another example, Apache Bench is being considered as a test tool to be supported due to its use by ATC.

5.3 Integrated Development Environment

The Integrated Development Environment (IDE) is envisioned to support standard (web-based) development activities and to provide a front-end for the interaction with the RADON framework by enabling the interaction with the RADON tools and the access to the shared spaces of the RADON artifacts (i.e. RADON models).

We carried out an initial validation of the first release of the IDE at M18 based on the feedback provided by both internal validation and the involvement of an independent user (i.e. not directly involved in RADON) to trial the RADON platform & IDE.

Considering the lessons learned from the aforementioned activities in addition to the requirements set defined in the previous deliverable D2.1, we have added three new "Must have" requirements (R-T2.3-24, R-T2.3-26, R-T2.3-27) and we have updated one of the IDE's requirements, specifically the R-T2.3-23:

[5] https://github.com/orgs/radon-h2020/projects/2

Integration of Verification Tool and Decomposition Tool: Requirements R-T2.3-26 and R-T2.3-27 are related to the integration on the IDE of the Verification and Decomposition tools, respectively. During the development of the first release of the IDE the current versions of these tools have been integrated in order to achieve the preliminary set of RADON workflows defined in the deliverable D2.3 (i.e. Verification and Decomposition workflows). These two new requirements have been added to complete the set of IDE's requirements concerning the customization of the development environment to support the interaction with the RADON tools.

Customization of User Interface (UI) elements for integrated RADON tools: Requirement R-T2.3-24 is about the need to support the interaction with the RADON tools integrated into the IDE using the UI elements (e.g. menu, popup window, views, tree etc) that best fit to the tool's needs. This new requirement takes into consideration the feedback collected during the initial validation activity provided both internally (i.e. by the units involved in the project) and externally (i.e. by the user not involved in RADON). In particular, it was highlighted the need to improve the graphical way the integrated tools will be used within the IDE. Therefore, the objective of this new requirement is to increase usability.

Availability of custom RADON menus and commands: Requirement R-T2.3-23 has been updated with the requirement R-T2.3-25. This requirement is about the need of a custom RADON menu providing custom commands in order to interact with the RADON framework. The previous requirement defined in D2.1 (i.e. R-T2.3-23) defined only the need of a RADON menu in order to launch the RADON tools integrated within the IDE. Considering the lesson learned during the validation activities, the requirement R-T2.3-23 has been refined and updated with requirement R-T2.3-25 in order to include also the definition of custom commands that support the interaction with the overall framework (e.g. commands to opening the RADON Help page, to open a browser windows with deployment status and monitoring information).

5.4 Graphical Modelling Tool (GMT)

In the following months after the first GMT release at M12, we validated the GMT internally based on well-known demo applications and common serverless use cases. We also presented and discussed our solution to RADON use case providers as well as external parties. Apart from fixing minor change requests, we got four additional requirements:

Referencing files as URLs: The GMT at M12 expected all files to be physically present in a RADON Model. This means that any business logic (e.g., the function code for FaaS) needs to be uploaded in a respective packaging format (e.g., ZIP for Python- or NodeJS-based functions or JAR for Java-based functions) prior to deploying the applications. As developers may utilize multiple functions, and these functions may change very often, the manual task to push the new business logic to the GMT becomes tedious and is not very DevOps-like where you want to automate things. Therefore, there must be the possibility to reference any business logic as a URL, either staged in an accessible storage location or in a binary and build artifact management tool, such as Sonatype Nexus or JFrog Artifactory. This will make it possible to change the business logic of the application but does not require to update the RADON Model.

Manage company-specific RADON Models and TOSCA entity types: In RADON, we publish and maintain RADON Models as well as reusable TOSCA entity types to a public GitHub repository called the RADON Particles. This repository, on the one hand, serves as an example of a public RADON Template Library (cf. D5.3 Technology Library and D4.4 RADON Models II) and, on the other hand, provides a modelling baseline for RADON users that want to use the GMT to compose application blueprints. So, it is important to use the current RADON Particles master branch when starting the GMT. However, at the same time, companies that model applications want to store their models separately from the public one, e.g. to push and manage them in their internal and private Git repository, such as GitLab. Therefore, there must be the possibility to initialize and start the GMT based on the RADON Particles, but company-specific RADON Models as well as custom TOSCA entity types must be stored differently. The GMT needs a way to separate the modelling entities into multiple repositories.

Save exported CSAR to the filesystem: This requirement arose when we first integrated RADON tools with the RADON IDE based on Eclipse Che. The GMT is the central tool to generate a TOSCA CSAR, which is required by the RADON Orchestrator to deploy the application therein. Also, the integration point with other tools is also an executable CSAR, e.g., the Defect Prediction can analyze the content prior to deployment. Therefore, the GMT must be able to save the generated CSAR to Eclipse Che's workspace. This enables other tools to further process the CSAR prior to deploying it using the RADON Orchestrator.

Publish/load RADON Models to/from RADON Template Publishing Service: Recently, the RADON Template Publishing Service has been introduced (cf. D5.3 Technology Library) as an additional layer on top of VCS to manage RADON Models. This service offers enterprise-ready features such as comprehensive user management and the option to publish RADON Models in a certain version such that the content can no longer be changed, which forces users to organize each modification in a new version, inspired by software package management tools. Essentially, this service is an instantiation of what we call "Template Library" in RADON, on top of the possibility to use Git to version your models. Therefore, the GMT must be able to push RADON Models to the RADON Template Publishing Service once users decide to finalize a model and put it into production. Similarly, the GMT needs to be able to pull a model from the service to modify it and push it as a new version back.

5.5 Decomposition Tool

The decomposition tool is envisioned to support three typical usage scenarios: (1) architecture decomposition, (2) deployment optimization, (3) accuracy enhancement. Our previous work has focused particularly on the second usage scenario. We investigate the suitable performance model for serverless FaaS as well as the corresponding model parametrization methodology. Based on the findings, a practical approach to optimizing the deployment of an application that consists of Lambda functions and S3 buckets is then proposed and implemented in a prototype of the tool. We also carry out an initial validation of the tool in terms of the accuracy of the performance model and the utility of the optimization approach.

Lessons learned from the aforementioned activities lead to two changes in the requirements for the decomposition tool (see deliverable *D2.1*):

- Removed R-T3.2-3, which states that the decomposition tool must be able to optimize a platform-independent RADON model. However, it is impossible for the tool to instantiate the performance model from abstract nodes and relationships. This is because a benchmark has to be created on a specific cloud platform and the performance model normally varies across different cloud platforms.
- Updated R-T3.2-10, which states that the available solution methods for deployment optimization may include cutting planes, branch & bound and generic heuristics. Deployment optimization for a RADON model is typically a nonlinear mixed-integer programming problem. At present, genetic algorithms are the only solution method that can well handle this kind of problem.

5.6 Defect Prediction Tool

The Defect Prediction (DP) tool envisions N main scenarios: (1) detecting the defect-proneness of an IaC blueprint, (2) model training and validation …

Our previous work has focused particularly on the X scenario(s). We investigated and validated the applicability of the RADON framework for IaC Defect Prediction on 85 Ansible-based projects. Based on our findings, Random Forest and Decision Tree classifiers resulted in the best Machine-Learning algorithm for the problem of predicting the defect-proneness of IaC scripts. This led to an update of several requirements as explained below. We finally implemented an alpha version of the tool for model training and validation, which provides APIs to (1) run the detection on Ansible playbooks and task files using a predefined model and (2) get the defect prediction model of the most similar project of the problem at hand.

Lessons learned from the aforementioned activities led to two changes in the requirements for the defect prediction tool (see deliverable *D2.1*):

- Updated R-T3.4-5, which states that the defect prediction tool could provide a defect threat level to architecture elements and predict threat-level defects under certain infrastructure assumptions. This is a typical regression problem that requires to identify the number of bugs in the infrastructure to establish a threat-level. However, it requires an ontology of IaC bugs, which does not exist yet. In addition, the current defect predictor is a classification model based on Decision Tree or Random Forest. Therefore, the requirement has been changed to address its explainability by providing the user with a set of rules that identify defective-prone IaC scripts and the decision path that led to the final prediction. This requirement has also been raised by one of the industrial partners and therefore its priority changed from COULD to MUST HAVE.
- Updated priority of R-T3.4-3, which states that the defect prediction tool *could* provide a command line interface. Although initially not a required functionality, partners from industry, in particular PRQ, requested to work with a command-line interface provided by the defect predictor tool. Therefore, the new priority for the requirement is MUST HAVE.

- Removed R-T3.4-11, which states that the defect prediction tool must decrease the bug-fixing times with respect to manual inspection. First, the requirement partially overlaps R-T3.4-10, which states that the defect-prediction tool must improve performances over manual inspection. Second, the goal of the defect predictor is to locate bug-prone software components and not to automatically fix them. Indeed, bug fixing is a developers' concern and depends on their ability to find a fix for a given bug. As the defect-predictor is essentially a detector, we stick with the requirement R-T3.4-10 to improve the performance in identifying defects during manual inspection.

5.7 Orchestrator

The orchestrator is a service that takes the TOSCA service blueprint created with a set of RADON tools and deploys it on the final provider. This transiting position of this tools generated a large list of requirements from all other tools connected with the orchestrator. However, during the orchestrator development and communication with the users and other tool owners, the set was substantially updated during the project, to better reflect the role of the orchestrator.

Setting a targeted TOSCA version for RADON. During the orchestrator development before M12 three the decision of target TOSCA version was determined. Orchestrator was initially released in M4 and designed to support TOSCA v1.1, which was the current version of that time. The TOSCA group released v1.2 and v1.3 soon afterwards. Through the investigation of the changes, the decision was made to improve the version to gain new features. The TOSCA v1.3 was selected to be the official version for RADON orchestrator and all content as TOSCA module and service templates. The release of orchestrator in M13 already supported TOSCA v1.3. Future releases will not be focused on a newer version of TOSCA until the end of the RADON, but rather on stability and user and integration requirements. This planned roadmap will improve the usability and polish the interoperability of the tools to benefit DevOps operations and delivery toolchain workflow.

Orchestrator should be available as a RADON SaaS component. To follow the RADON SaaS approach, the orchestrator component should be deployable as a service inside the RADON environment or standalone SaaS component. This strives to develop a REST API around the orchestrator functionality, and in the next stage, add the components that make it SaaS ready. This includes IAM management and the possibility to run separately from other tools.

Orchestrator needs to support all TOSCA extensions required by RADON users. Most of the applications can be deployed with basic TOSCA instructions and does not need support for TOSCA extensions to be deployed on the targeted infrastructure. However, during the experiences of use-cases and RADON tool developers it became evident that some RADON use cases and examples require the support of the extensions. The current requested features are: being able to use private ssh key files as environment variables; parsing of TOSCA policy types including targets keyname support; the compressed TOSCA CSAR support which is currently under development; and a packaging command that would use TOSCA templates along with all the accompanying files needed for the deployment to prepare a zipped TOSCA CSAR. There was also an enhancement to provide the support for get_artifact TOSCA function which would allow retrieving an artifact

location between TOSCA entities defined in the same TOSCA service template and thereby user would be able to use his prepared orchestration artifacts (e.g. zipped FaaS function files) effectively. These key extensions will be added to the orchestrator coverage in the next few months.

Deployment environment. Orchestrator should be able to consider each deployment and redeployment separately. In the CLI version of the orchestrator this ability of orchestrator statefulness is achieved with a current working directory where all deployment data is stored. The user needs a command to instruct the orchestrator if the data of the state is still relevant. This means that users should be able to force *initial deployment*, that would clear history deployment data, or *resume previous deployment* based on history or interrupted state data. The requirement will be fulfilled with adding new switches to the deploy command in CLI version and orchestrator REST API.

5.8 Delivery Toolchain

Monitoring Tool

The first release of the Monitoring tool at M12 was followed by validation iterations based on feedback provided either by internal validations on demo serverless use cases or by RADON and/or external use case providers. As a result, additional requirements arose along with some that are actually refinements of the initial set of requirements defined in D2.1, specifically the R-T5.1-3.

Dynamic configuration of monitoring components: The various monitoring components (i.e. pushGateway for short-lived jobs, cAdvisor for containers, mTail for application logs) need to be added as static configuration on the Prometheus server in order to be automatically scraped. The option to allow for dynamic configuration of exporters on a central monitoring server would facilitate a centralized approach while monitoring multiple deployments at the same time. A TOSCA configuration that encapsulates the dynamic registration through a service discovery service like Consul is considered.

Provide wrapper for manually injected code when monitoring FaaS: In order to monitor the deployment of a FaaS effectively, a RADON user would have to inject manually code for pushing the collected metrics to a pushGateway component, since these kinds of jobs may not exist long enough to be scraped by the monitoring server. There is clearly the need to automate as possible the monitoring part code injection by wrapping the injected code in a library and expose the supported type of metrics as annotations directly on a FaaS level.

Export monitoring metrics from application logs: Many times the application logs hide valuable information but they require some kind of parsing in order to be properly exposed as metric types and consequently scraped by the monitoring tool. Additional TOSCA node/configuration types have to be specified to support a new monitoring component that attaches to a log file and constantly tailing it to parse the generated log records. Each log record is parsed according to a set of pattern matching rules (based on regular expressions) and the metrics of interest are then exposed to the monitoring server.

Push alerts to support deployment reconfiguration (scaling): Monitoring the runtime behaviour of a cloud deployment may result in identifying bottlenecks that should trigger a reconfiguration of the deployed topology. The RADON users should be able to define thresholds with regard to monitoring of specific metrics and in cases where those thresholds exceed predefined values the deployment should adapt to the new requirements. The idea is to push alerts towards the rest of the Delivery Toolchain components to provide feedback and let the xOpera orchestrator to redeploy and scale parts of the topology that work on full capacity.

FunctionHub

One challenge related to FunctionHub has been to formally separate the requirements between FunctionHub, the Radon tool, and Cloudstash, the use case. Most requirements towards FunctionHub are also relevant for PRQ's use case and have thus been counted as Use Case Requirements. Some requirements however, mostly the ones regarding RADON integration and support of specific vendors are kept as such.

In a development environment tightly integrated and reliant on every consortium partner, any change in a tool will have consequences for the other partners and their related requirements.

Tool integration. A success factor of RADON is how well the tools are integrated with each other. That led us to be especially cautious about getting locked down by a specific solution of integration. One main question was the interaction between GMT, FunctionHub and the Orchestrator. More precisely allowing Function-models in GMT to rely on URLs, but also whether the intermediate step of downloading the Function was necessary before the deployment of the application.

CI/CD

Tool automation. PRQ, through their Use Case, has been a driver for automation and adapting each RADON tool to fit in a CI/CD environment. Automation of processes relies on being able to script the execution of the relevant tool. Some of the updated requirements have been to ensure access to the tool functionality through a command line, either through a client[6], like Defect Prediction Tool, or an API[7] like CTT.

Pipeline execution. The discussion around CI/CD execution has been going on since the start of the project and probably will continue. This has led to the new for new implementations and further requirements. One example is the question of what role SCM tools like Git should play in executing the CD pipeline, opposed to a direct, manual trigger from the RADON editor.

6. Usage scenarios

This section outlines how the RADON solution - in terms of its value proposition and value space- - compares with challenges from the state of the art, as reflected by a recent study we performed by way of qualitative research [Lenarduzzi2020]. This section describes the new/changes tool technical requirements and illustrate diagrammatically the conceptualisation of these requirements

[6] https://github.com/radon-h2020/radon-defect-prediction-api/issues/3
[7] https://github.com/radon-h2020/radon-ctt/issues/73

into workflows and then high-level RADON feature. This contextualization is also done per use cases with the intention to push the envelope of the requirements analysis done in year 1 by moving the tool requirements from the toy application into the industrial use case context. The reader is pointed to D6.1 for the functional requirements of the industrial use case applications. This section solely maps the tool requirements into their usage scenarios in the context of the industrial demonstrators.

6.1 ATC

Based on the validation for the ATC use case, as presented in D6.1, we will use the two business scenarios to validate the RADON tools, following the respective RADON workflows. More specifically, we describe how the adoption of the RADON methodology in the development of the Viarota scenarios will demonstrate the instantiation of the RADON workflows and will validate the requirements of the tools that we analysed earlier in this document

Viarota: The implementation of the verification workflow

In the following figure, we present how ATC adopts the RADON methodology in the specification of non-functional constraints in the Viarota TOSCA enabled model and the verification that the resulting model is compliant to these constraints.

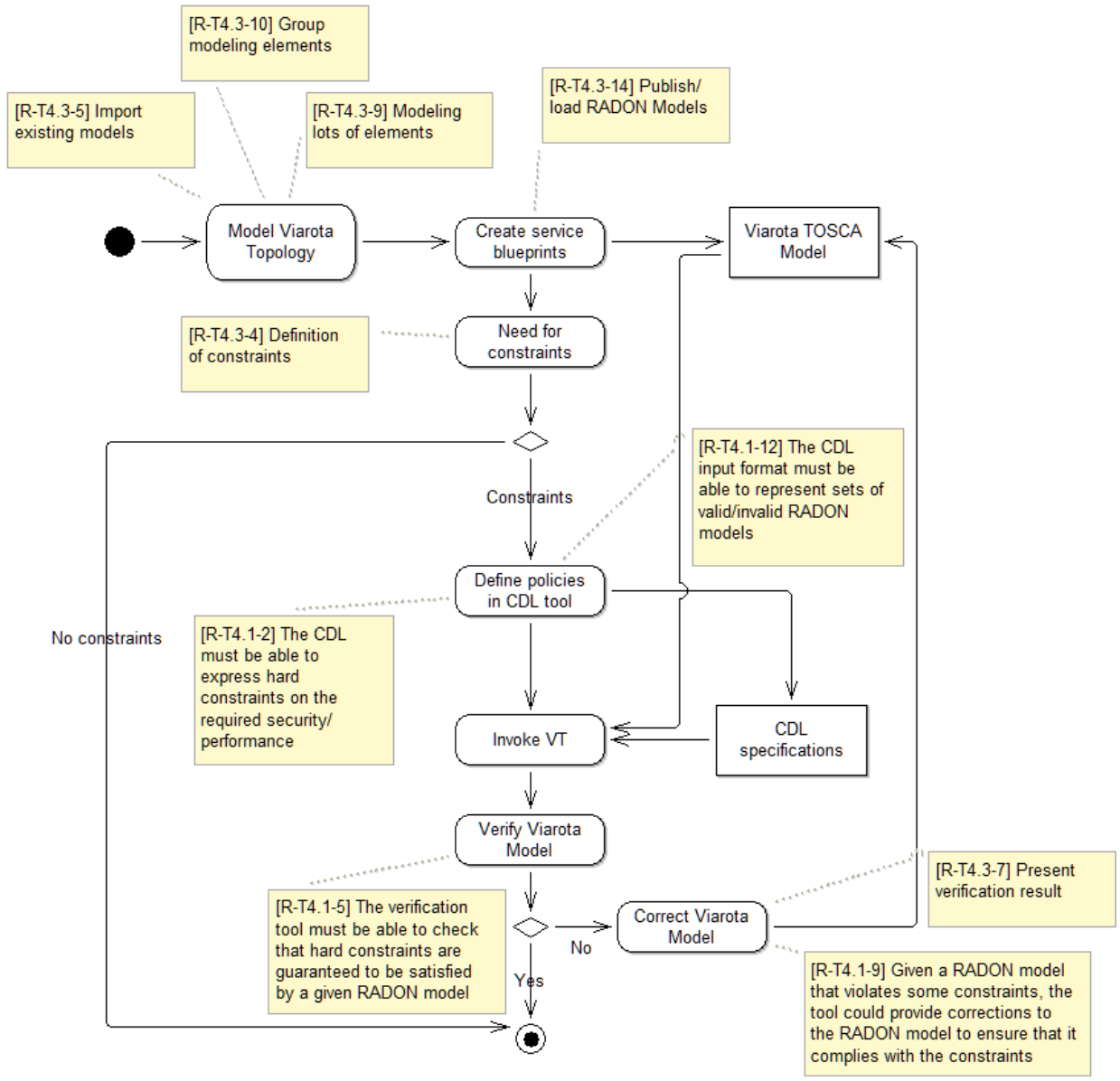

Figure 6.1.1 ATC use case: instantiating the verification workflow.

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	The DevOps team of ATC aims to use the **GMT tool** to **model the topology** of the Viarota architecture, including all the elements for supporting the definition of	R-T4.3-5[8] R-T4.3-9[9] R-T4.3-10[10]	M19

[8] https://github.com/radon-h2020/radon-gmt/issues/8
[9] https://github.com/radon-h2020/radon-gmt/issues/12
[10] https://github.com/radon-h2020/radon-gmt/issues/13

	the business logic, the deployment infrastructures and the data description, processing and storage.		
2	We will use the **GMT tool** to **create/reuse blueprints** for the Viarota service components and APIs exposed for the mobile application and **publish the** Viarota **TOSCA model**	R-T4.3-14[11]	M19
3	In case that we want to define constraints with respect to quality characteristics (i.e. performance efficiency, security, etc.), we will use the **GMT tool** to **define** these **constraints** inside the model.	R-T4.3-4[12]	M20
4	We aim to use the **CDL tool** to **specify** the **policies** that implement the defined constraints for the Viarota operational environment.	R-T4.1-12[13] R-T4.1-10[14]	M21
5	As we want to cross check that implementation code is compliant to these policies, we should be able to **invoke the Verification Tool** and verify that the developed model is compliant to the **CDL specification**	R-T4.1-5[15]	M24
6	In case that verification check shows errors, the **VT tool** will provide how the model can be made compliant to the policies.	R-T4.1-9[16]	M25
7	We will, then, use the **GMT tool** to be **check the verification report** and apply the corrections to the model	R-T4.3-7[17]	M25

[11] https://github.com/radon-h2020/radon-gmt/issues/30
[12] https://github.com/radon-h2020/radon-gmt/issues/7
[13] https://github.com/radon-h2020/radon-verification-tool/issues/21
[14] https://github.com/radon-h2020/radon-verification-tool/issues/10
[15] https://github.com/radon-h2020/radon-verification-tool/issues/13
[16] https://github.com/radon-h2020/radon-verification-tool/issues/17
[17] https://github.com/radon-h2020/radon-gmt/issues/10

Viarota: The implementation of the decomposition workflow

In the following figure, we present how the Viarota current architecture will be optimised, through the RADON decomposition workflow, considering the optimisation of the deployment practices to satisfy specific business customer needs.

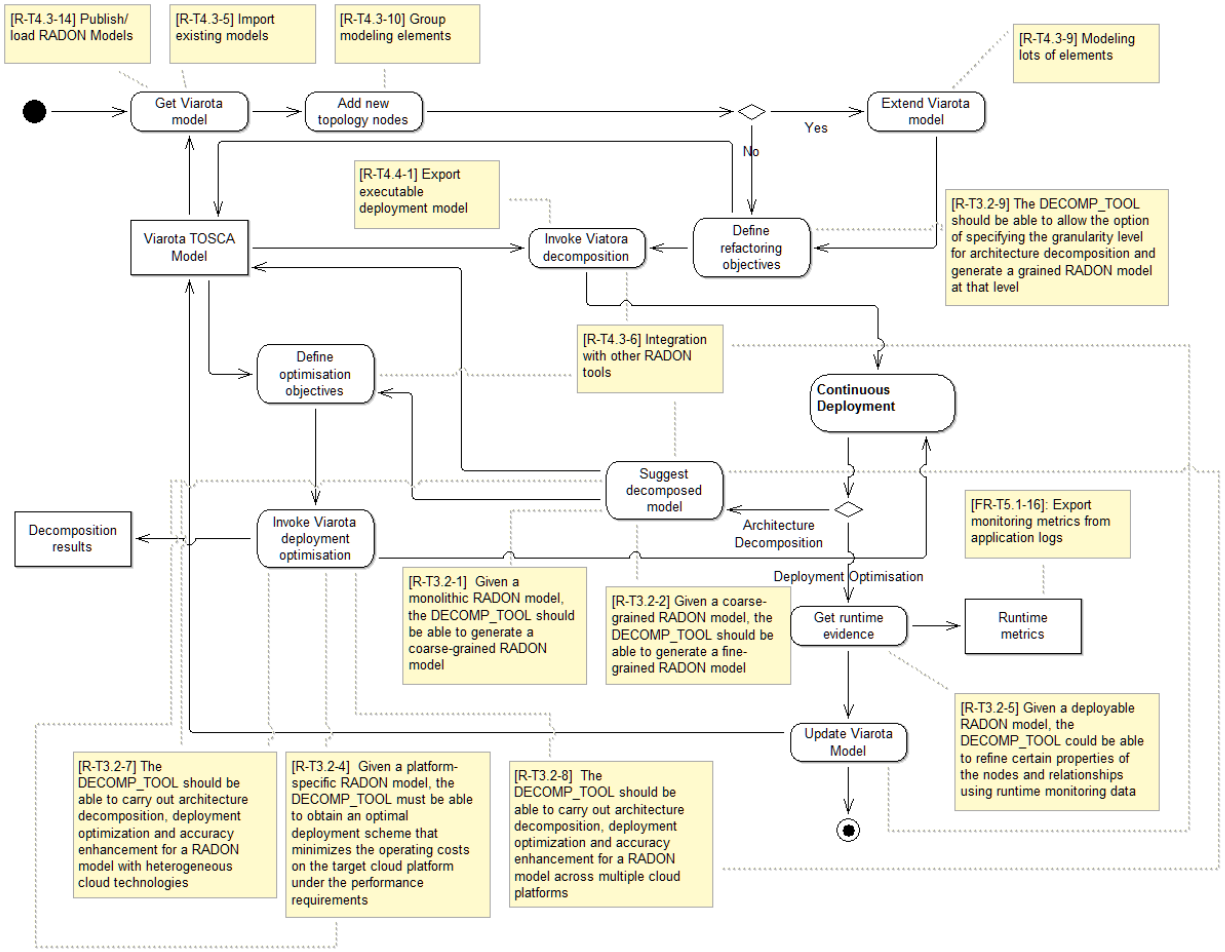

Figure 6.1.2. ATC use case: instantiating the decomposition workflow.

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	The DevOps team of ATC aims to use the **GMT tool** to retrieve the Viarota **model,** which is published in the Viarota workspace.	R-T4.3-5[18] R-T4.3-14[19]	M19

[18] https://github.com/radon-h2020/radon-gmt/issues/8

[19] https://github.com/radon-h2020/radon-gmt/issues/30

2	In case that we need to **add new nodes** to the model topology, we will use the **GMT tool** to **extend the model** and thus accommodate the new features for the Viarota solution.	R-T4.3-9[20]	M19
3	We aim to use the **Decomposition Tool** to define the granularity level of the Viarota architecture **refactoring**	R-T3.2-9[21]	M20
4	We will use the **GMT Tool** to export multiple deployment models that can be executed through the RADON **CI/CD process**.	R-T4.4.-1[22] R-T4.3-6[23]	M20
5	We expect that the **Decomposition Tool** will **suggest the optimal architecture decomposition** of the Viatora model, based on microservices, and export it to the **GMT Tool**.	R-T3.2-1[24] R-T3.2-2[25] R-T3.2-4[26] R-T3.2-7[27] R-T3.2-8[28] R-T4.3-6[29]	M20
6	We aim to use the **Decomposition Tool** to define the objectives for the **optimisation of the** Viarota **deployment package**, subject to business requests	R-T4.3-6[30]	M24
7	We will then use the **Decomposition Tool** to invoke multiple deployment configurations and get the optimised one,	R-T3.2-4[31] P-T3.2-5[32] R-T3.2-7[33]	M24

[20] https://github.com/radon-h2020/radon-gmt/issues/12
[21] https://github.com/radon-h2020/radon-decomposition-tool/issues/9
[22] https://github.com/radon-h2020/radon-gmt/issues/14
[23] https://github.com/radon-h2020/radon-gmt/issues/9
[24] https://github.com/radon-h2020/radon-decomposition-tool/issues/1
[25] https://github.com/radon-h2020/radon-decomposition-tool/issues/2
[26] https://github.com/radon-h2020/radon-decomposition-tool/issues/4
[27] https://github.com/radon-h2020/radon-decomposition-tool/issues/7
[28] https://github.com/radon-h2020/radon-decomposition-tool/issues/8
[29] https://github.com/radon-h2020/radon-gmt/issues/9
[30] https://github.com/radon-h2020/radon-gmt/issues/9
[31] https://github.com/radon-h2020/radon-decomposition-tool/issues/4
[32] https://github.com/radon-h2020/radon-decomposition-tool/issues/5
[33] https://github.com/radon-h2020/radon-decomposition-tool/issues/7

	which considers the runtime behaviour of the Viarota solution, **based on metrics**.	R-T3.2-8[34]	
8	We expect that the optimised deployment configuration can be used in the **GMT Tool** to **update the** Viarota **TOSCA model**	R-T4.3-6[35]	M24

Viarota: The implementation of the continuous testing workflow

In the following figure, we explain the adoption of RADON to support the continuous testing process of the Viarota development.

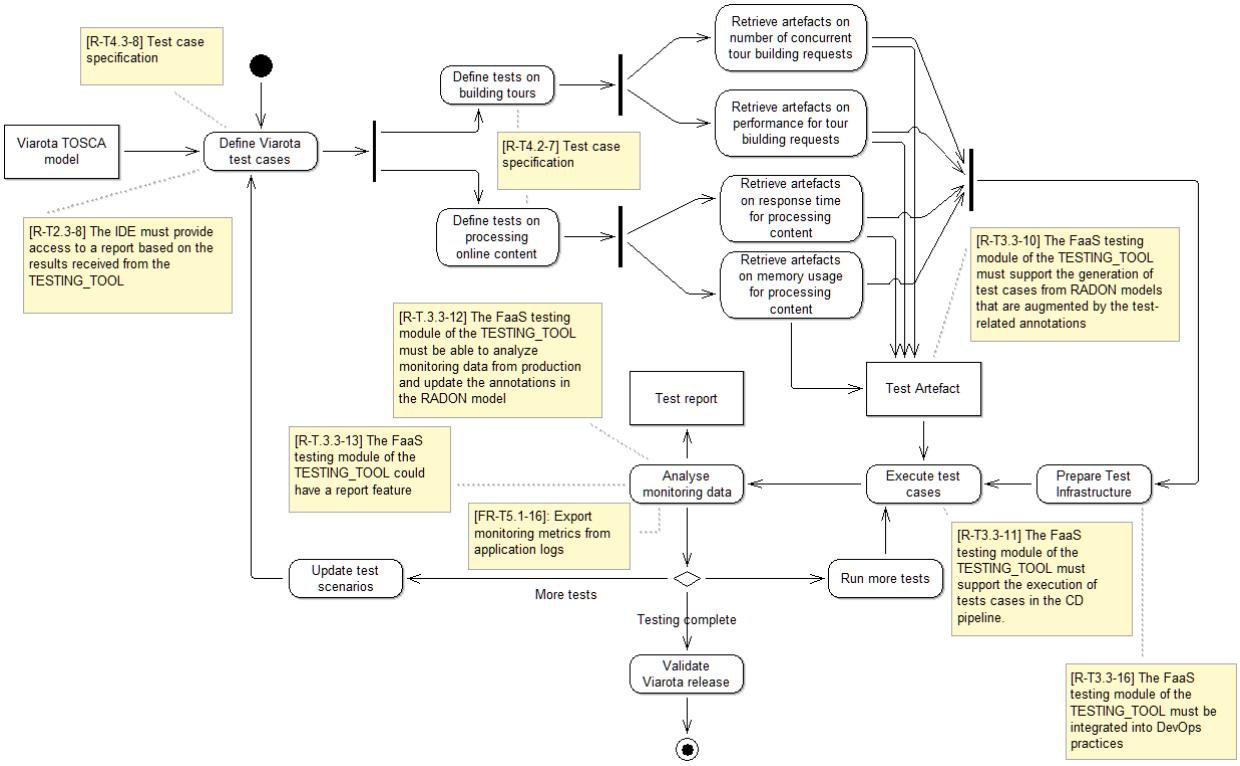

Figure 6.1.3. ATC use case: instantiating the continuous testing workflow.

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	The DevOps team of ATC aims to use the **GMT tool** to specify the test	R-T4.3-8[36]	M22

[34] https://github.com/radon-h2020/radon-decomposition-tool/issues/8
[35] https://github.com/radon-h2020/radon-gmt/issues/9
[36] https://github.com/radon-h2020/radon-gmt/issues/11

	scenarios in the **Viarota TOSCA model**		
2	We aim to use the **GMT tool** to specify test scenarios regarding the two Viarota business scenarios: a) building tours, and b) processing content from various online sources.	R-T4.2-7[37]	M22
3	In the case of the Viarota scenario for building tours, we aim to use the **Continuous Testing Tool** to **retrieve artifacts** for the specification of test cases on the number of concurrent user requests for tour development and the performance of the relevant service infrastructure.	R-T3.3-10[38]	M22
4	In the case of the Viarota scenario for processing online content, we aim to use the **Continuous Testing Tool** to **retrieve artifacts** for the specification of test cases on the response time and the memory utilization of the processing services.	R-T3.3-10[39]	M22
5	We aim to use the **Continuous Testing Tool** to link with the DevOps practices and set up the testing environment	R-T3.3-16[40]	M23
6	We will then use the **Continuous Testing Tool** to invoke the execution of the defined Viarota test cases and **analyse the runtime metrics,** collected from the **Delivery Toolchain,** to produce	R-T3.3-11[41] R-T3.3-12[42] R-T3.3-13[43] R-T5.1-16[44]	M25

[37] https://github.com/radon-h2020/radon-particles/issues/13
[38] https://github.com/radon-h2020/radon-ctt/issues/25
[39] https://github.com/radon-h2020/radon-ctt/issues/25
[40] https://github.com/radon-h2020/radon-ctt/issues/30
[41] https://github.com/radon-h2020/radon-ctt/issues/26
[42] https://github.com/radon-h2020/radon-ctt/issues/27
[43] https://github.com/radon-h2020/radon-ctt/issues/28
[44] https://github.com/radon-h2020/radon-monitoring-tool/issues/3

	a test report.		
7	If no more tests are required, the ATC DevOps team can validate the released version of Viarota. Otherwise, the **GMT Tool** will be invoked to **update the test cases** or specify additional ones.	R-T2.3-8[45]	M25

Viarota: The implementation of the continuous integration and deployment workflow

In the following figure, we describe the use of the RADON tools across the Viarota development processes to automate the integration of new features development and the deployment of new software releases in the Viarota production pipeline.

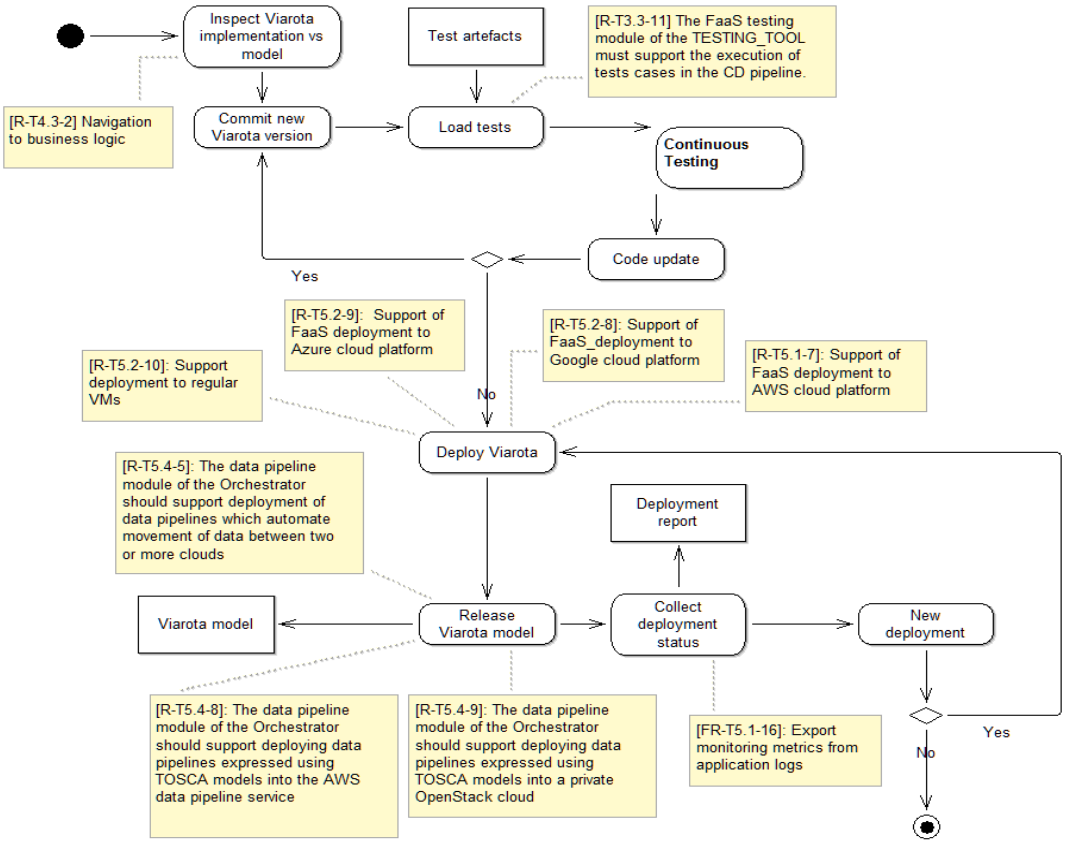

Figure 6.1.4. ATC use case: instantiating the continuous integration and deployment workflow.

[45] https://github.com/radon-h2020/radon-ctt/issues/24

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	The DevOps team of ATC aims to use the **GMT tool** to navigate to the source code of the **Viarota TOSCA model**	R-T4.3-2[46]	M22
2	In every new **commit** of the Viarota code, the Continuous Testing Tool must be automatically invoked to execute the test case specifications.	R-T3.3-11[47]	M22
3	Upon the completion of a test cycle, if no further **code update** is required, we aim to use the **Orchestrator** to **deploy** the current Viarota to the envisaged infrastructure environment, based on the Viarota TOSCA blueprints committed in the **Templates Library**.	R-T5.1-7[48] R-T5.2-8[49] R-T5.2-9[50] R-T5.2-10[51]	M25
4	The automated release process of the Viarota application includes the **deployment of data pipelines** in different cloud deployment settings (public and private clouds), according to the specifications of the TOSCA model, and to **support the data exchange** between them.	R-T5.4-5[52] R-T5.4-8[53] R-T5.4-9[54]	M26
5	Based on the **collected monitoring data**, we will use the **Delivery Toolchain** to **manage new deployments**, subject to specific requirements.	R-T5.1-16[55]	M27

[46] https://github.com/radon-h2020/radon-gmt/issues/5
[47] https://github.com/radon-h2020/radon-ctt/issues/26
[48] https://github.com/radon-h2020/radon-delivery-toolchain/issues/11
[49] https://github.com/radon-h2020/radon-delivery-toolchain/issues/12
[50] https://github.com/radon-h2020/radon-delivery-toolchain/issues/13
[51] https://github.com/radon-h2020/radon-delivery-toolchain/issues/14
[52] https://github.com/radon-h2020/radon-delivery-toolchain/issues/19
[53] https://github.com/radon-h2020/radon-delivery-toolchain/issues/22
[54] https://github.com/radon-h2020/radon-delivery-toolchain/issues/23
[55] https://github.com/radon-h2020/radon-monitoring-tool/issues/3

Viarota: The implementation of the monitoring workflow

Finally, the runtime behaviour of the Viarota solution will be used to validate the RADON monitoring workflow, as presented in the following figure.

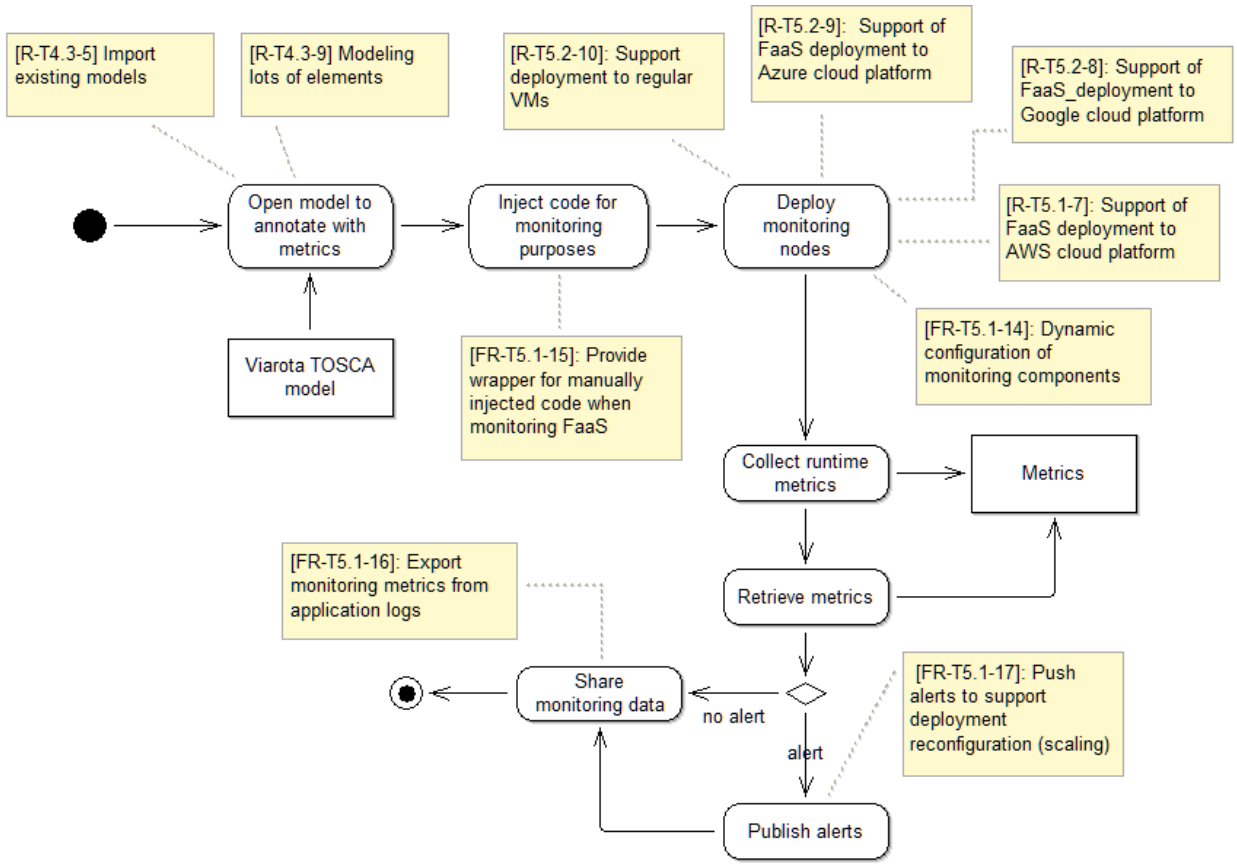

Figure 6.1.5. ATC use case: instantiating the monitoring workflow.

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	We aim to use the **GMT tool** to import the **Viarota TOSCA model** and **model elements** with respect to **monitoring different metrics**.	R-T4.3-5[56] R-T4.3-9[57]	M22
2	In order to support monitoring of the	FR-T5.1-15[58]	M22

[56] https://github.com/radon-h2020/radon-gmt/issues/8
[57] https://github.com/radon-h2020/radon-gmt/issues/12
[58] https://github.com/radon-h2020/radon-monitoring-tool/issues/2

	Viarota (micro-)services at the application level, we aim to reuse the wrappers and the monitoring service blueprints, provided in the **Templates Library**, to **manually provide** the necessary **code**.		
3	The automated release process of the Viarota application includes the **deployment** of the **TOSCA model** with the appointed **monitoring components**, which should be configured, based on this model.	R-T5.1-7[59] R-T5.2-8[60] R-T5.2-9[61] R-T5.2-10[62] FR-T5.1-14[63]	M25
4	At runtime, we will exploit the monitoring capabilities of the **Delivery Toolchain** to continuously **collect data** for the monitoring metrics defined in the model and process them to **assess the runtime behaviour** of the deployed Viarota configuration.	FR-T5.1-16[64]	M26
5	We aim to use the **Delivery Toolchain** to **share monitoring data** to other RADON tools and optimise the development and deployment of the Viarota solution for different business customers.	FR-T5.1-16[65]	M26
6	To monitor the presence of potentially exceptional runtime behaviour of the Viarota business scenarios, we aim to use the **Delivery Toolchain** to publish alerts to the Viarota admin panel.	FR-T5.1-17[66]	M27

[59] https://github.com/radon-h2020/radon-delivery-toolchain/issues/11
[60] https://github.com/radon-h2020/radon-delivery-toolchain/issues/12
[61] https://github.com/radon-h2020/radon-delivery-toolchain/issues/13
[62] https://github.com/radon-h2020/radon-delivery-toolchain/issues/14
[63] https://github.com/radon-h2020/radon-monitoring-tool/issues/1
[64] https://github.com/radon-h2020/radon-monitoring-tool/issues/3
[65] https://github.com/radon-h2020/radon-monitoring-tool/issues/3
[66] https://github.com/radon-h2020/radon-monitoring-tool/issues/4

6.2 ENG

Serverless SARA - Verification & Defect Prediction

The ENG usage of the Verification and Defect Prediction workflow aims to support software architects, designers, and any SARA stakeholder interested in utilizing an abstract representation of the FaaS solution (or an intermediate architecture representation which is under scrutiny for servicification) for the purpose of its verification and for the prediction of any defects connected to it upon deployment and later orchestration. The workflow foresees the initial modelling of the architecture to be subject of trade-off analysis within the RADON Graphical Modelling Tool (left-hand side of the diagram below) with the explicit expression of constraints (using the CDL) inside the same tool. Once this step is completed, the RADON user is expected to "Run RADON Verification Workflow" mode, which passes the current state of the model (TOSCA + Ansible files as an analyzable CSAR package) to the RADON CDL Verification and RADON Defect Prediction Tool. Such tools will return revisions necessary by the RADON User, through graphical UI highlights in the RADON GMT tool. At this point, the user is expected to enact iterative architectural refinement using the received feedback. Once this iterative refactoring terminated, the RADON user is expected to export the CSAR file and enact the RADON CI/CD Pipeline, also featuring its own monitoring backend, which closes the DevOps loop.

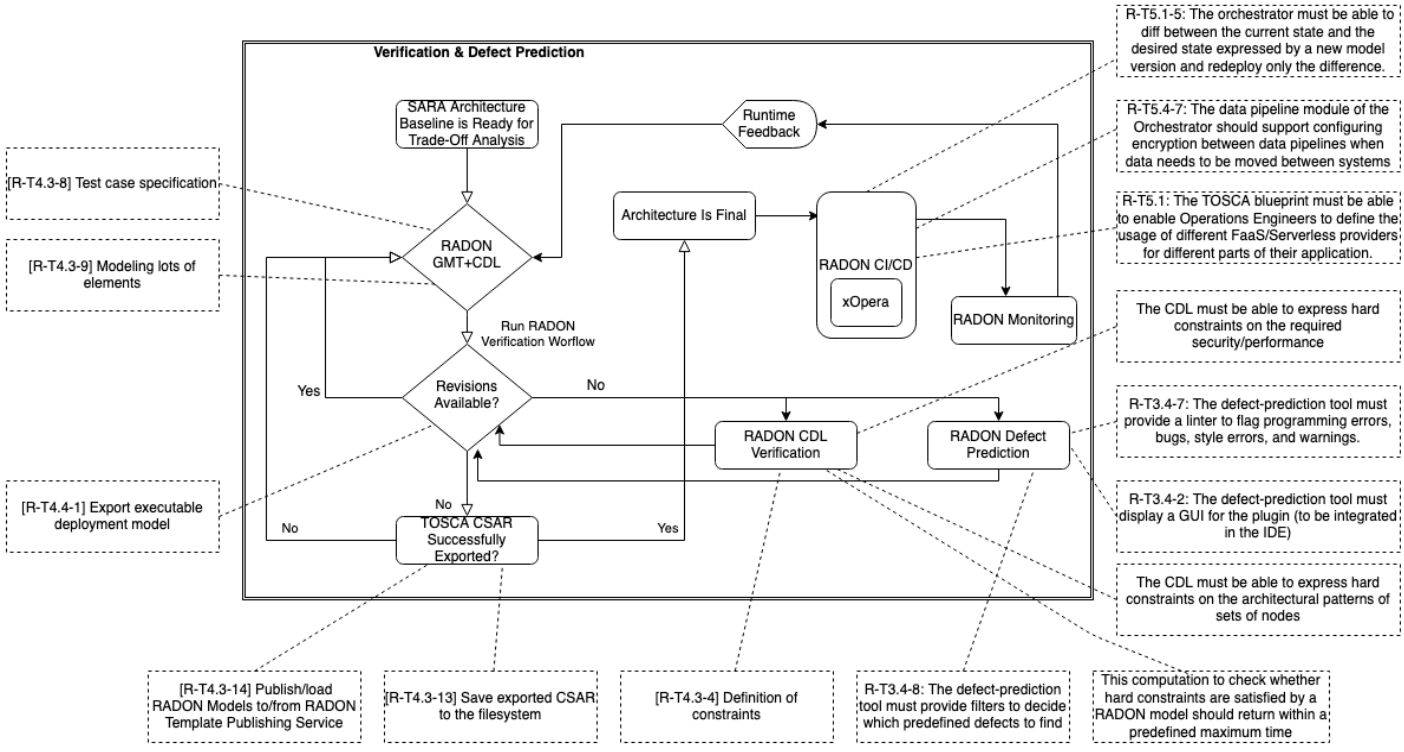

Figure 6.2.1. - SARA Verification & Defect Prediction

As part of the aforementioned scenario, several key tool requirements match almost exactly the expectations and required intended operations behind the aforementioned workflow. For example, that the GMT must provide an exportable and executable CSAR model and format or that the

CSAR itself needs to be exported to the filesystem for further analysis. Similarly, more -ops requirements are also in line with the needs of the workflow, for example that the GMT, its products, and their verification/defect-prediction must enable operators to work iteratively, and agnostically from any technology-specific infrastructure element.

#	Business Workflow Activity Description	Tool Requirement(s)	When (M19-M27)
1	The **GMT** will be used to model the architectural and topological aspects of CloE-IoT services (phase one) and the SARA assistive tasks (phase two) (TG-ENG-1). This step will produce the TOSCA models describing CloE-IoT and SARA and represent the main input for the next validation steps.	R-T4.3-1 Integration into IDE R-T4.3-10 Group modelling elements	M19 (July)
2	The Ansible scripts coding the configuration and deployment of an OpenFaaS instance within the ENG cloud infrastructure (TG-ENG-2). This OpenFaaS instance will be used during the validation process to host both CloE-IoT services and SARA functionalities.	No RADON tool used for this activity.	-
3	The **Defect Prediction Tool** will be used to generate the quality metrics concerning the Ansible scripts created in the previous step (BG-ENG-2).	R-T.3.4-5 The defect-prediction tool must provide a set of rules that identify defect-prone scripts and an interpretation of the final decision R-T.3.4-7 The defect-prediction tool must provide a linter to flag programming errors, bugs, style errors, and warnings R-T.3.4-8 The defect-prediction tool must provide filters to decide which predefined defects to find R-T.3.4-10 The defect-prediction tool must improve performances over manual inspection	M20 (August)
4	The previous three steps will then be repeated until the generated quality metrics are judged satisfactory.	No RADON tool used for this activity.	-
5	Once the Ansible scripts are ready, the **GMT** will	R-T4.3-5 Import existing models	M21 (September)

	be used to import the Ansible scripts produced in Step 2 into the TOSCA models produced in Step 1.	R-T4.3-6 Integration with other RADON tools R-T4.4-3 Import of model in different format	
6	The **Constraint Description Language (CDL)** is then used to describe the requirements of CloE-IoT and SARA. In particular the CDL will be challenged against its expressiveness in describing security and privacy requirements stemming from the EU GDPR (TG-ENG-3).	R-T4.1-1 CDL: Pre/post conditions R-T4.1-2 CDL: Hard constraints R-T4.1-4 CDL: Soft constraints	M21 (September)
7	Once the security and privacy constraints are available in the CDL language they will be used as input for the Verification Tool. The **Verification Tool** will be used to verify the compliance of the model produced in Step 5 against the constraints defined in Step 6.	R-T4.1-5 VT: Hard constraints R-T4.1-9 VT: Correction R-T4.1-11 VT: Soft constraints, search R-T3.2-13 VT: Soft constraints, improvement	M22 (October)

Table 6.2.1 - SARA Verification & Defect Prediction

Serverless SARA - Decomposition & Continuous Integration/ Deployment

The Decomposition and Continuous Integration / Deployment usage scenario (Figure 6.2.2 and Table 6.2.2) aims to support the developers during the re-engineering of the current (non-FaaS) implementation of the SARA solution and its subsequent deployment of Serverless SARA.

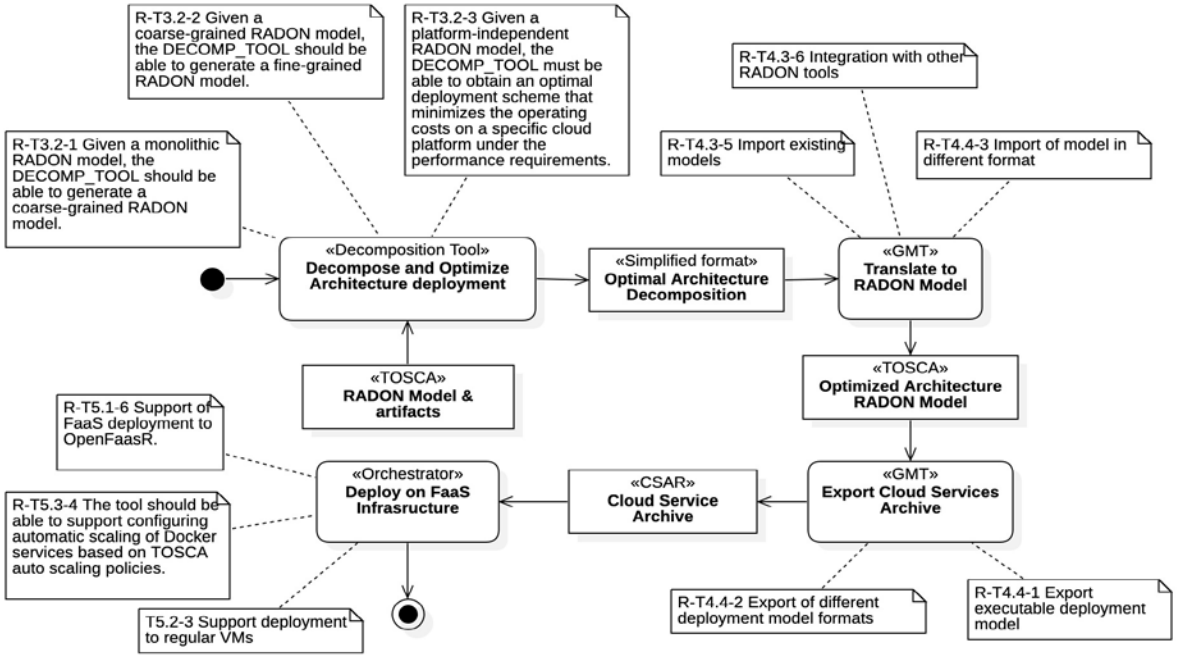

Figure 6.2.2 - SARA Decomposition and Continuous Integration / Deployment

#	Business Workflow Activity Description	Tool Requirement(s)	When
1	The **Decomposition Tool** will be used to obtain an optimized decomposition of the CloE-IoT services and SARA functionalities described in the RADON models created with the application modelling and validation scenario (BG-ENG-2). This decomposition will describe how the CloE-IoT/SARA can be decomposed in terms of functions for OpenFaaS (TG-ENG-1).	R-T3.2-1 Given a monolithic RADON model, the DECOMP_TOOL should be able to generate a coarse-grained RADON model. R-T3.2-2 Given a coarse-grained RADON model, the DECOMP_TOOL should be able to generate a fine-grained RADON model. R-T3.2-3 Given a platform-independent RADON model, the DECOMP_TOOL must be able to obtain an optimal deployment scheme that minimizes the operating costs on a specific cloud platform under the performance requirements.	M22 (October)
2	The **GMT** will then be used to translate the optimal architecture decomposition generated by the decomposition tool into a RADON Model. This model represents the description of the re-engineered version of CloE-IoT/SARA to be deployed on the OpenFaaS instance used for the validation.	R-T4.3-5 Import existing models R-T4.3-6 Integration with other RADON tools R-T4.4-3 Import of model in different format	M23 (November)
3	The **GMT** will then be used to generate the Cloud Service Archive (CSAR) of the FaaS compliant version of CloE-IoT/SARA.	R-T4.4-1 Export executable deployment model R-T4.4-2 Export of different deployment model formats	M23 (November)
4	Finally, the **Orchestrator** will be used to deploy the new version of CloE-IoT/SARA on OpenFaaS (TG-ENG-2). The outcome of this step will be an up and running FaaS version of CloE-IoT/SARA.	R-T5.1-6 Support of FaaS deployment to OpenFaas R-T5.2-3 Support deployment to regular VMs R-T5.3-4 The tool should be able to support configuring automatic scaling of Docker services based on TOSCA auto scaling policies.	M25 (January)

Table 6.2.2 - SARA Decomposition and Continuous Integration / Deployment

Serverless SARA - Continuous Testing & Monitoring

The Continuous Testing & Monitoring usage scenario (Figure 6.2.3 and Table 6.2.3) aims to support the developers for the continuous testing of the serverless version of SARA.

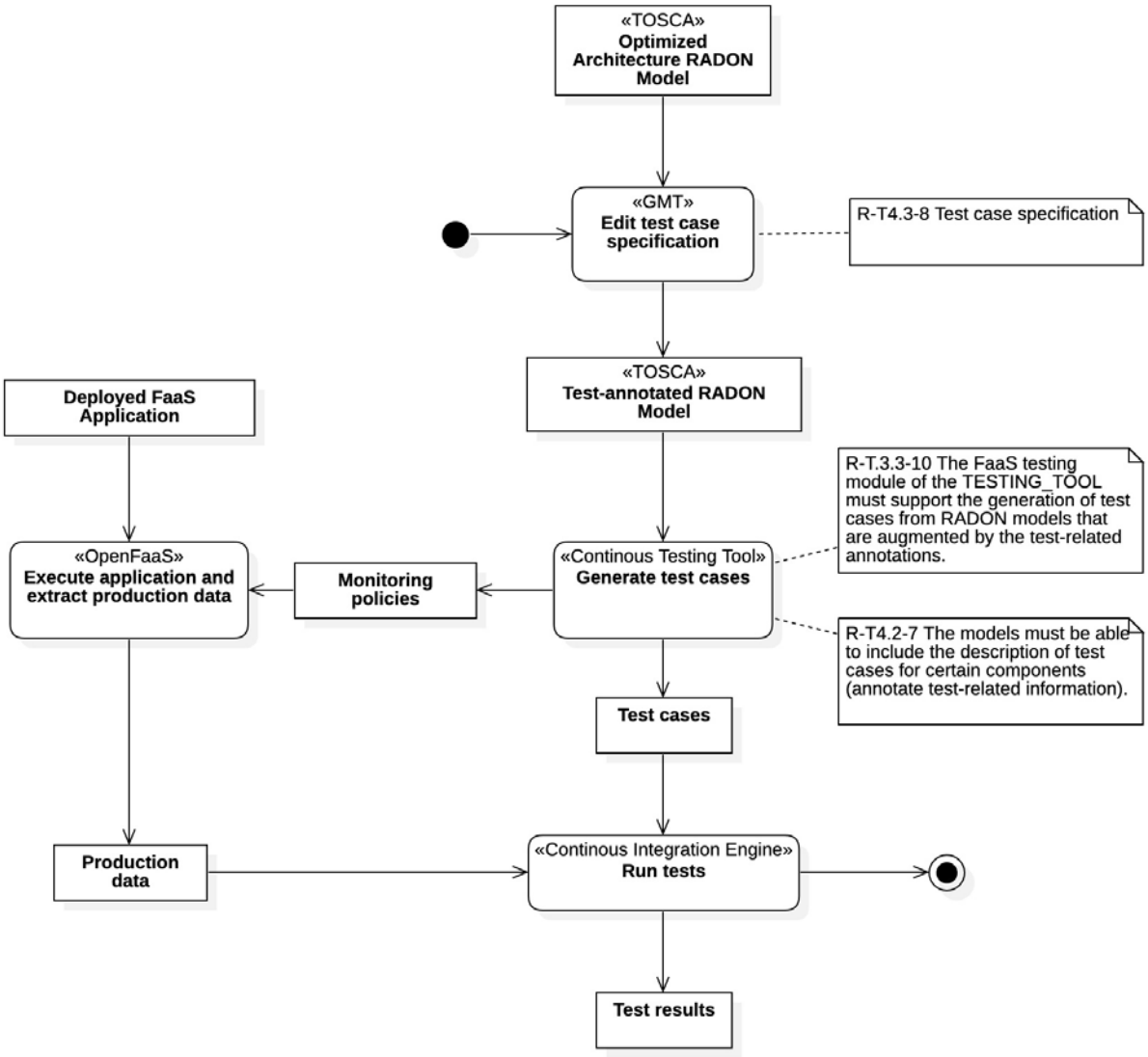

Figure 6.2.3 – SARA Continuous testing and monitoring

#	Business Workflow Activity Description	Tool Requirement(s)	When
1	The **GMT** will be used to specify the test cases for the Optimized Architecture described by the RADON Model obtained in the step 2 of the Application Deployment Scenario described above.	R-T4.3-8 Test case specification	M25 (January)
2	The **Continuous Testing Tool** will be used to generate: (a) the actual test cases to be executed by JUnit and JMeter; and (b) the monitoring policies for the OpenFaaS instance hosting CloE-IoT/SARA.	R-T.3.3-10 The FaaS testing module of the TESTING_TOOL must support the generation of test cases from RADON models that are augmented	M25 (January)

		by the test-related annotations. R-T4.2-7 The models must be able to include the description of test cases for certain components (annotate test-related information).	
3	The **OpenFaaS instance**, configured with the monitoring policies produced by the Continuous Testing Tool, will be used to generate and collect data about the actual behavior of the re-engineered version of CloE-IoT/SARA.	No RADON tool used for this activity.	-
4	The **Jenkins instance** deployed at the ENG infrastructure will be used for tests generated in Step 2 and the data produced by the OpenFaaS instance configured in Step 3.	No RADON tool used for this activity.	-

Table 6.2.3 - Continuous testing and monitoring of SARA

6.3 PRQ

As stated in ‘D6.1 - Validation plan’, the main benefit of adopting RADON methodology for us would be the integration of each stand-alone tool and the creation of a seamless workflow supporting all stages of development. In order to achieve this agile process, RADON relies on an intuitive framework integration and tool automation. This is also clearly reflected in the requirements we have highlighted in the workflow from Figure 6.3.1. This workflow envisions how we will utilise the RADON framework while showcasing the potential of customisation into the defined development approach of PRQ. Central to our work is the incremental nature of development, this is evident through the ‘iteration’ box in the model. Test frameworks locally and runtime data both in the test and production environment all provide essential feedback for the next iterative step.

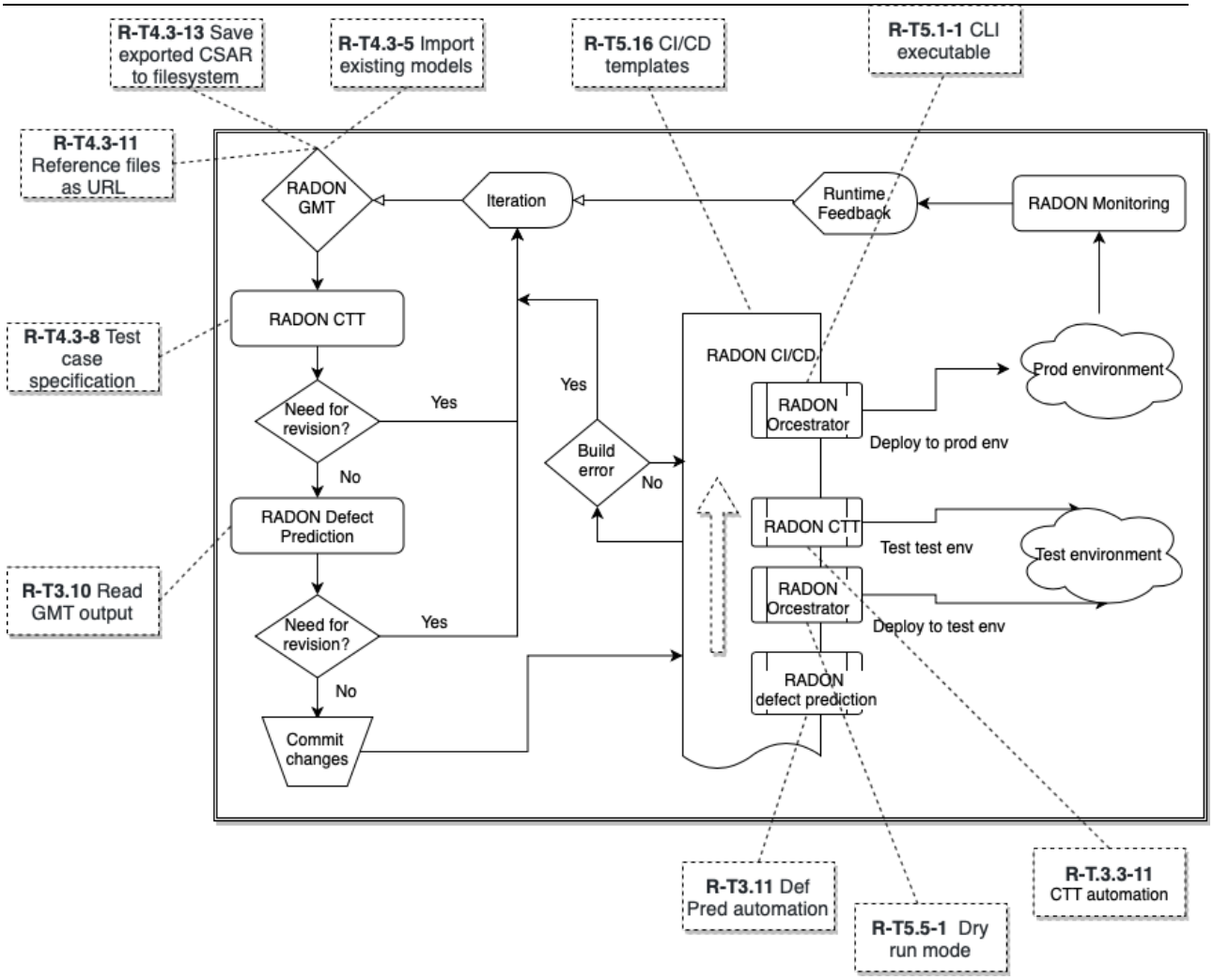

Figure 6.3.1 PRQ use case requirements

Managed DevOps - Modelling, Function-Hub and Defect prediction

#	Business Workflow Activity Description	Tool Requirement(s)	When
1	Using **GMT**, PRQ intends to create all relevant resources in order to create the Serverless Artifact Manager. Required resources: API Gateway, FaaS, table storage and buckets.	R-T4.3-9[67]	M18
2	Using **GMT**, PRQ intends to store application related data in a separate folder from the Radon Particles. This enables interaction with SCM like git, which is an essential component in our workflow.	R-T4.3-13[68]	M20

[67] https://github.com/radon-h2020/radon-gmt/issues/12
[68] https://github.com/radon-h2020/radon-gmt/issues/29

3	Using the **Defect Prediction Tool**, PRQ intends to get insight into the details of the application based on the generated code from **GMT.** By doing this we can have small incrementing changes based on feedback.	R-T3.10[69]	M22
4	Using **GMT**, PRQ intends to reopen existing tosca templates and append resources.	R-T4.3-5[70]	M20
5	Using **GMT**, PRQ intends to base the script path on a URL from **FunctionHub** instead of a physical location on the local machine.	R-T4.3-11[71]	M24

Managed DevOps - Continuous Testing & Monitoring

#	Business Workflow Activity Description	Tool Requirement(s)	When
1	Using the **GMT**, model test templates based on the **CTT**.	R-T4.3-8[72]	M22
2	Using the **CTT**, create unit-tests for verifying endpoints of the generated Functions.	CTT #38[73]	M23
3	Using the **CTT**, create load-tests for the **FunctionHub** characterization study.	CTT #39[74]	M23

Managed DevOps - Orchestration and CI/CD

#	Business Workflow Activity Description	Tool Requirement(s)	When
1	Using the **CI templates** available, PRQ intends to configure a pipeline in Jenkins.	R-T5.16[75]	M18
2	Using the **CI templates**, PRQ intends to include the **Defect Prediction Tool**, in the pipeline for compliance and improvement of code reviews.	R-T3.11[76]	M24

[69] https://github.com/radon-h2020/radon-defect-prediction-plugin/issues/7
[70] https://github.com/radon-h2020/radon-gmt/issues/8
[71] https://github.com/radon-h2020/radon-gmt/issues/27
[72] https://github.com/radon-h2020/radon-gmt/issues/11
[73] https://github.com/radon-h2020/radon-ctt/issues/38
[74] https://github.com/radon-h2020/radon-ctt/issues/39
[75] https://github.com/radon-h2020/radon-delivery-toolchain/issues/48
[76] https://github.com/radon-h2020/radon-defect-prediction-plugin/issues/8

	Using the **CI templates**, PRQ intends to include the **CTT** in the pipeline for automated deployment and testing.	R-T3.3-11[77]	M24
3	Using the **CI templates**, PRQ intends to include the **Orchestrator** in the pipeline for automated deployments and traceability	Toolchain #3[78]	M20

7. Conclusions

The deliverable presents the final set of tool technical requirements of the RADON framework. These are justified within the context of the three RADON industrial use case scenarios. A conceptualisation of these requirements is given my mapping them to the high-level RADON features and user needs through RADON workflows. The mean features and user needs have been identified through discussions and interviews with external practitioners and third parties who, which have led to the identification of gaps in the current state-of-the-art of the DevOps technologies that RADON's features are addressing. Per each tool, new and/or changes in the requirements have been identified in response to validations of the tools in the use case scenarios. Such changes have been documented and traced by applying a RADON requirement engineering management process that has also been discussed and documented in this deliverable.

References

[Baldini2017] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, P. Suter. Serverless Computing: Current Trends and Open Problems. Research Advances in Cloud Computing. 2017.

[Bersani2016] M. M. Bersani, F. Marconi, D. A. Tamburri, P. Jamshidi and A. Nodari, "Continuous Architecting of Stream-Based Systems," 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, 2016, pp. 146-151, doi: 10.1109/WICSA.2016.26.

[**Bez19**] C-P. Bezemer, S. Eismann, V. Ferme, J. Grohmann, R. Heinrich, P. Jamshidi, W. Shang, A. van Hoorn, M. Villavicencio, J. Walter, F. Willnecker, "How is Performance Addressed in DevOps?" in *International Conference on Performance Engineering (ICPE '19)*, 2019, pp. 45-50.

[**Bre11**] G. Brewka, T. Eiter, and M. Truszczy´nski. "Answer set programming at a glance.", *Communications of the ACM*, vol. 54, no. 12, pp. 92–103, 2011.

[**Cas19**] G. Casale, M. Artač, W.-J. van den Heuvel, A. van Hoorn, P. Jakovits, F. Leymann, M. Long, V. Papanikolaou, D. Presenza A. Russo, S.N. Srirama, D.A. Tamburri, M. Wurster, L. Zhu, "Rational Decomposition and Orchestration for Serverless Computing", in *The Symposium and Summer School on Service-Oriented Computing (SummerSoc)*, 2019. (accepted for publication).

[**Eyk17**] E. van Eyk, A. Iosup, S. Seif, and M. Th¨ommes, "The SPEC cloud group's research vision on FaaS and serverless architectures.: In *Proceedings of the 2nd International Workshop on Serverless Computing*. ACM, 2017. p. 1-4.

[**Fox17**] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research", arXiv preprint arXiv:1708.08028, 2017

[77] https://github.com/radon-h2020/radon-ctt/issues/26

[78] https://github.com/radon-h2020/radon-delivery-toolchain/issues/3

[Gannon17] D. Gannon, et al. “Cloud-Native Applications”, in IEEE Cloud Computing - 2325-6095/17; 2--6. 2017.

[**Gel88**] M. Gelfond, and L. Vladimir. "The stable model semantics for logic programming." In *ICLP/SLP*, vol. 88, pp. 1070-1080. 1988.

[**Gue19**] Guerriero, M., Garriga, M., Palomba, F., Tamburri, D. Adoption, “Support, and Challenges of Infrastructure-as-Code: Insights from Industry.” in *International Conference on Software Maintenance and Evolution (ICSME)*. 2019. *(accepted for publication)*.

[Har92] Harker, S.D.P, Eason, K.D., Dobson, J. E., "The change and evolution of requirements as a challenge to the practice of software engineering." *Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 266-272, 1993.

[Kosztyán2018] Kosztyán, Z. T. (2018). Serviceability of large-Scale systems.. Simul. Model. Pract. Theory, 84, 222-231.

[Kruchten1995] Kruchten, Philippe (1995, November). Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

[Jan13] Janes, A., Remencius, T. Sillitti A. Succi, G. "*Managing changes in requirements: an empirical investigation*." Journal of Software Evolution and Process, 2013; 25:1273-1283.

[Leitner2019] P. Leitner, Erik Wittern, J. Spillner and W. Hummer. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software. 2019

[Lenarduzzi2020] V. Lenarduzzi et al. “Towards a Technical Debt Conceptualization for Serverless Computing” - IEEE Software, Under Review.

[Lewis2010] Lewis, K. M. & Hepburn, P. (2010). Open card sorting and factor analysis: a usability case study.. The Electronic Library, 28, 401-416.

[Lynn17] T. Lynn, et al. “A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms”; Proceedings of ICCTCS p. 1—8, 2017. IEEE.

[Nupponen2020] J. Nupponen and D. Taibi. Serverless: What it Is,What to Do and What Not to Do. International Conference on Software Architecture (ICSA 2020). 2020.

[Rus09] Russo Barbara, et al. "Requirements Management", in Agile Technologies in Open Source Development, 2010.

[**Spi19**] J. Spillner. "Quantitative Analysis of Cloud Function Evolution in the AWS Serverless Application Repository." *arXiv preprint arXiv:1905.04800*, 2019.

[Tamburri2018] Tamburri, D. A., Bersani, M. M., Mirandola, R. & Pea, G. (2018). DevOps Service Observability By-Design: Experimenting with Model-View-Controller.. In K. Kritikos, P. Plebani & F. D. Paoli (eds.), ESOCC (p./pp. 49-64), : Springer. ISBN: 978-3-319-99819-0

[Tamburri2018b] Tamburri, D. A., Bersani, M. M., Mirandola, R. & Pea, G. (2018). DevOps Service Observability By-Design: Experimenting with Model-View-Controller.. In K. Kritikos, P. Plebani & F. D. Paoli (eds.), ESOCC (p./pp. 49-64), : Springer. ISBN: 978-3-319-99819-0.

[**Vog18**] Vögele, Christian, et al. "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems." *Software & Systems Modelling* (2018): 1-35.