Empir Software Eng (2018) 23:1457–1489
https://doi.org/10.1007/s10664-017-9543-z

General methods for software architecture recovery: a potential approach and its evaluation

Damian A. Tamburri[1] · **Rick Kazman**[2,3]

Published online: 22 September 2017

Abstract Software architecture is a critical artefact in the software lifecycle. It is a system blueprint for construction, it aids in planning teaming and division of work, and it aids in reasoning about system properties. But architecture documentation is seldom created and, even when it is initially created, it is seldom maintained. For these reasons organisations often feel the need to recover legacy architectures, for example, as part of planning for evolution or cloud migration. But there is no existing general architecture recovery approach nor tool that can be applied to any type of system, under any condition. We will show that one way of achieving such generality is to apply systematic code inspection following a Grounded Theory (GT) approach. Though relatively costly and human-intensive, a GT-based approach has several merits, for example: (a) it is general by design; (b) it can be partially automated; (c) it yields evidence-based results rooted of the system being examined. This article presents one theoretical formulation of a general architecture recovery method–called REM–and reports on the evaluation of REM in the context of a large architecture recovery campaign performed for the European Space Agency. Our results illustrate some intriguing properties and opportunities of GT-based architecture recovery approaches and point out lessons learned and venues for further research.

Communicated by: Martin Robillard

✉ Damian A. Tamburri
damianandrew.tamburri@polimi.it

Rick Kazman
kazman@hawaii.edu

[1] Politecnico Di Milano, Milan, Italy

[2] University of Hawaii, Honolulu, HI, USA

[3] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords Software architecture recovery · Ground-truth software architectures · Grounded theory · QVT · OCL · UML profiles · Model-driven engineering · Model-driven architecture · European space agency · Industrial experience report

1 Introduction

Software architecture is an abstraction of a software system: its composite parts, their properties, their relationships, as well as the choices that led to the system and its parts (Bass et al. 2012). This article reports on our experiences with recovering software architectures by means of an approach that makes no assumptions towards the architecture being recovered and, hence, may be **always** applicable, no matter the functions, qualities, languages, or operational conditions of the system whose architecture needs to be recovered. In fact, this article contains a first theory for defining a "general" software architecture recovery approach, evaluating that theory over a single, albeit large, industrial, mission-critical case-study. Our argument towards such general methods is quite intuitive.

On one hand, software architecture is a critical software engineering artefact given its many possible uses (e.g., documentation (Bachmann et al. 2000), verification (Tsai and Xu 2000), analysis (Clements et al. 2001), community steering and agreement, work-division, decision-making, etc.). And there may be dire consequences for not having a good software architecture, as has been reported in many research conferences, e.g., ICSE,[1] WICSA & CompArch[2] or ECSA.[3]

On the other hand, both theory and practise in architectures suggest that maintaining good quality software architectures is non-trivial. And this is particularly difficult for legacy systems because in these cases there is typically no documentation available and the original developers are often long gone. This is where software architecture recovery is crucial (Dueñas et al. 1998; Eixelsberger et al. 1998; Kazman and Carriere 1999; Schmerl et al. 2006). Software architecture recovery is the process of inspecting software systems and extracting representations of their artefacts (e.g., call graphs, file dependency graphs, etc.) for the purpose of documenting the software architecture (Ding and Medvidovic 2001). Software architecture recovery research has seen increased attention in the last few years from multiple perspectives. For example, an established body of work in this area flourishes in the form of EU projects such as REMICS[4] and ARTIST.[5] Also, considerable standardisation efforts exist such as the OMG Architecture-Driven Modernisation taskforce (Newcomb 2005; Izquierdo and Molina 2010). Finally, there has been research on discovering the Ground-Truth of architectures, such as Medvidovic et al. (Garcia et al. 2013).

Nevertheless, as recent results show (Lutellier et al. 2015), one limitation in the state of the art is that there is no single approach or technology that can be considered *general*, that is, systematically usable for recovering software architectures *for any kind of system* (e.g., a mission-critical system vs. an old web application), *for any kind of purpose (e.g., for*

[1] http://icse2017.gatech.edu/

[2] http://www.softwarearchitecture.org

[3] http://ecsa2016.iku.edu.tr

[4] http://www.remics.eu/

[5] http://artist-project.eu/

modernisation as opposed to re-documentation) or under any circumstance (e.g., without assuming artefacts other than sources).

What is more, most available approaches for software architecture recovery are inaccurate (Lutellier et al. 2015) and frequently quite expensive. Furthermore, existing approaches do not provide a way to estimate the costs of the recovery effort so that an effective business case can be made, and an evaluation of expected ROI can be done. For these reasons, project managers are often hesitant to invest in such efforts. Finally, many existing approaches target just a single *view* (Clements et al. 2002) of an architecture, for example a "component and connector" view of the running system, or a module view.

This paper, by way of contrast, offers an architecture recovery approach that, basing its root around GT: (a) may be *general by design*; (b) can be made *reliable* at will, since it allows recovery of software architectures by construction from the very basic elements they represent (code artefacts, their structure, and modularisation), hence, multiple observers may be able to retrieve high-quality software architecture artefacts by triangulation, whilst the reliability of their observations can also be assessed with classical inter-coder reliability metrics (Gwet and Gwet 2002); (c) can be, and in fact was, partially *automated*; and (d) whose *cost* can be estimated *up-front* and depends strictly on the number and size of artefacts for the system under analysis. This approach stems from an industrial research and development project called RTE,[6] recently unclassified by the European Space Agency (ESA). The key goal behind RTE was that of "delivering an effective round-trip engineering technology for heterogeneous legacy space software systems" (Sarkarati et al. 2008). As part of this project we elaborated an approach that combined: (a) Grounded Theory, the prince of Qualitative Research approaches (Corbin and Strauss 1990; Schreiber and Carley 2004); (b) Model-Driven Architecture (MDA) principles and tools, e.g., model transformation technology; and (c) reverse-engineering technology.

The key idea behind RTE is simple: manual inspection of code is unavoidable to retrieve highly-reliable, if not ground-truth software architectures... but code, at its simplest, is nothing more than structured text embodying a *theory* (Naur 1985). So, why not realise architecture recovery with a research method aimed at manual inspection and analysis of text and theory generation? This is where Grounded Theory (GT) comes into play. Grounded Theory (Schreiber and Carley 2004) is a systematic, incremental and iterative qualitative research methodology developed in the late 1960s and employed in many research domains, including software engineering. Granted, this approach is labour intensive; however, counterbalancing the cost are two benefits: a) we can accurately predict an upper bound on the expense, and b) we have techniques to help manage that expense.

The approach resulting from the RTE project came to be known as "Roundtrip Engineering Methodology (REM)" and is the key contribution of this paper. The REM has many unique features: (a) may in fact be *general* - REM, following GT, makes no assumptions as to the nature of the text that needs to be analysed (e.g., in the scope of our project REM was applied to testing specifications as well as Java code) - this paper reports a data point to argue for this hypothesis; (b) it is *reliable* - REM involves the manual inspection of source code files using a structured GT methodology which is meant for the generation of a theory grounded in factual evidence (primarily source code, in our case) and whose reliability can be increased with multiple observers by triangulation and assessed with classical inter rater reliability metrics; (c) REM can be, and was in fact, *partially automated* using MDA model transformation technology; (d) REM can be instrumented for *cost-estimation* which

[6] http://tinyurl.com/of65un2

is required to build business cases around recovery expenses - REM could do this by associating an inspection cost with every source artefact available and evaluating analysis costs through GT phases. Several approaches exist to assist this estimation or to elaborate ad-hoc estimation techniques, e.g., by Thomson (2011). Finally, REM does not substitute current analysis and reverse-engineering techniques. Rather, REM may be used in combination with approaches that provide a high-level view, or those that reverse-engineer code-level facts. Both of these provide a starting-point from which REM engineers can begin their reading of the code.

As a proof of concept to test the REM generality hypothesis, the methodology was applied to a large mission-critical software system called FARC (File ARChive, part of the ESA MYCONYS Mission Control Software[7]). MYCONYS provides a secure, highly available and dependable distributed file-system for all ESA stations (e.g., satellite observation posts or launching sites) across the world, and the data they produce and exchange.

Novelty, Limitations and Conclusions As a result of the above proof-of-concept study, we argue that REM is a novel approach with profound potential. On one hand, we observed that the instruments we used to partially automate REM are sturdy and inextricably linked to the RTE scenario. On the other hand, we found that a measurable baseline needs to be created around approaches such as REM to provide quantification instruments for appraisal of precision, convenience, as well as testing the generality of the approach further. Finally, we recognise that the great potential of recovering highly-reliable software architectures comes at a great cost. Therefore, approaches such as REM should, for the moment, be considered as *last-resort* mechanisms, applicable wherefore all other approaches are inadequate. For example, our case study was a *last-resort* scenario, presented to us by ESA, where their heterogeneous code, operations, and maintenance scripts made all existing approaches infeasible. Similarly, the precision of approaches such as REM are still an open working hypothesis and seem to heavily depend by the case at hand.

In conclusion, this manuscript outlines and tests REM, a qualitative analysis-inspired semi-automated recovery approach. Although we claim that REM is a general approach, we cannot currently *prove* this yet. Rather, the industrial case study showcased in this manuscript shows that the approach does in fact *work* with no assumptions over systems under analysis and motivates further research in this direction to actually seize and demonstrate this opportunity towards generality. This paper is a humble bootstrap of the research line above; we offer a well-articulated but limited point of evidence that is consistent with our claim of generality, resting upon the foundations of GT.

Paper Structure First, Section 2 describes REM in detail. Second, as a proof-of-concept, the paper shows how REM was applied to FARC, allowing us to evaluate our approach. Third, the paper elaborates on the ways in which REM was automated in the context of FARC, providing insights elicited from industrial practise. Fourth, the paper reports lessons learned during the study that might benefit further research into software architecture recovery research.

[7] http://www.esa.int/Our_Activities/Operations/gse/MICONYS

2 GT-Based Architecture Recovery

This section elaborates on our experience with a GT-based method for architecture recovery which makes no assumptions over the architecture to be recovered. First, we describe the Grounded Theory qualitative data analysis method (van Niekerk and Roode 2009; Corbin and Strauss 1990). Later, we map GT to architecture recovery principles for the purpose of elaborating the REM methodology.

2.1 What is Grounded Theory?

Grounded Theory is arguably the most widely-used and perhaps the most effective structured qualitative data analysis methodology ever created. GT was created by two researchers in sociology—Barney Glaser and Anselm Strauss—in late 1967. Although the method was initially designed for the analysis of structured text, recent evolutions and interpretations of the method have extended its usage to video, audio, web and other forms of content. GT is generally split into three phases:

1) *Open Coding:* during this phase the primary sources of data (usually text or qualitative data of some sort) are read or viewed and each portion of the text is constantly compared to "codes", i.e. labels for portions of text that represent the concepts expressed. For example, some source code or a code-inspection report discussing code review results may be labelled with the code "COD-REV" or several portions of the same code might be labelled with: (a) "REV-CONT" for content of the review, (b) with "REV-METH" for the method of the review, and so on. This process is known as text micro-analysis (see top-most dotted box in Fig. 1). If the elements in the text express concepts that cannot be mapped to an existing code, then an additional code is created. Moreover, the researcher applying the method is constantly noting down observations in the form of "memos", i.e. brief textual notes describing an idea or observation made on the data so far or on the portion of text just analysed. Following our code review example, for the "REV-METH" code researchers may note in a memo that these reviews seem to follow a similar method for interfaces and a slightly different method for private classes. This process of memoing is constant; for example, when new codes are created memos capture the rationale for creation. The end result of open coding is a set of *categories*, creating a taxonomy of codes.
2) *Selective Coding:* during this phase the codes are constantly compared with each other (rather than with other pieces of data) until the categories are saturated into clusters of core-categories (see middle dotted box in Fig. 1). For example, the code category "REV-METH" is merged with category "REV-LIST-TYP" that identifies the types of code review guidelines applied in review exercises. Both codes are merged into a core-category of "REVIEW-PERF" identifying the prerequisites for performing code reviews. This process is typically instrumental to devise relations between two or more categories. For example, notice the relation between "REV-METH" and "REV-LIST-TYP". This is a process of selection of the core elements that the analyst can identify, but the process is also guided by the entity of merged concepts. The biggest and "denser" clusters of codes are usually selected first, since they clearly represent valuable clusters.
3) *Theoretical Coding:* during this phase the codes and core-categories are sorted and tentatively modelled (a process sometimes known as Theoretical Modelling). A Grounded Theory is generated by letting it emerge from one (or more) theory views or available

data models, usually by observation, e.g., through theory-inference focus groups. For example, a focus-group of expert code reviewers may visually inspect, confirm, and refine all the relations inferred by researchers through open- and selective-coding.

Alternatively, the theory is "forced" by comparison with an initial set of hypotheses. For example, researchers may have previously generated a series of hypotheses regarding what they expect to find in code reviews. These hypotheses can then be tested against the generated theory.

At this point, one might consider applying further analysis techniques, such as the 6C causality modelling framework for GT, where six types of codes (Context, Cause, Consequence, Contingency, Covariance, Conditions) are tentatively related together to infer causal relations (Glaser 1978).

Figure 1 shows the series of steps to be applied sequentially and iteratively as part of a GT-based analysis. Essentially, observers of a phenomenon are required to label any piece of evidence (in our case, source code elements) with a label, indicating the meaning of that piece of evidence (e.g., a method call, an entire method, an entire Java Class, depending on the desired accuracy – a class could be labelled with class purpose or functional role).

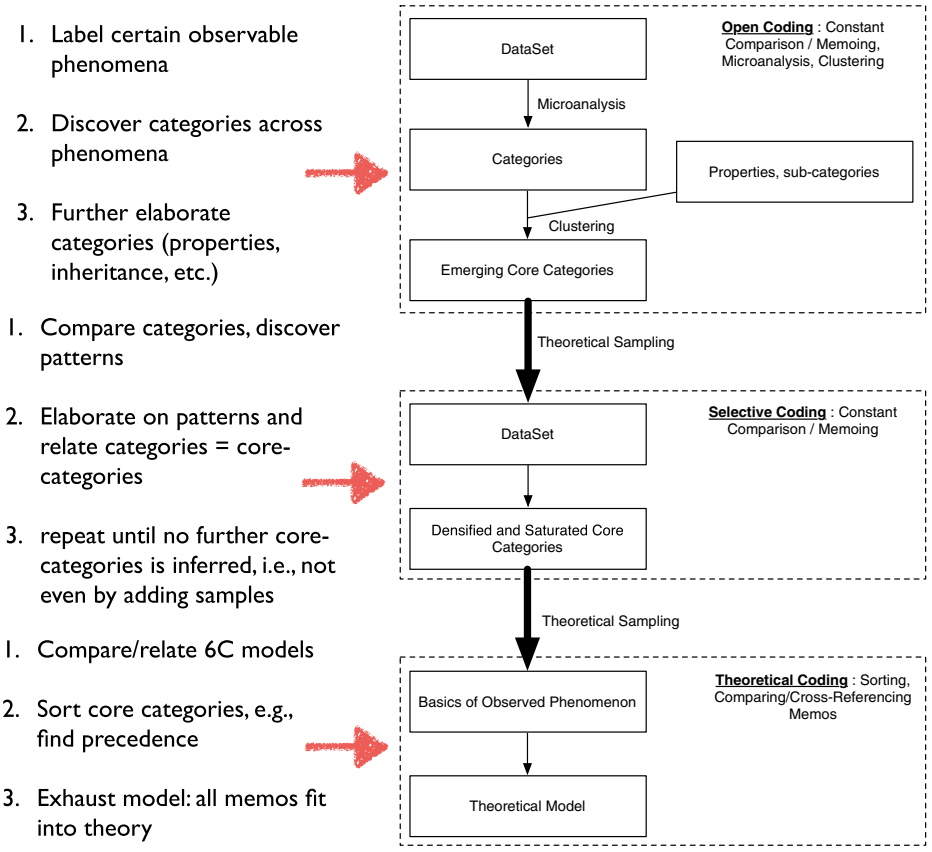


Fig. 1 Grounded theory – a general overview

These labels can then be analysed and clustered according to the emergent categorisation. Subsequently, codes can be merged, e.g., according to semantic similarity principles (Resnik 1999). As labels (or better, *codes*) are merged, categories become enriched with further details (e.g., some labels likely become properties of some others and so on). Moreover, in the shift from open to selective coding, patterns are discovered across categories (e.g., recurrent categories of sections of source code may denote a type of software component). The goal of selective coding is to relate categories together until there are no other additional possible relations to be discovered. The process to be followed is reflected in Fig. 2, tailored from a tutorial on Grounded Theory (Khandkar 2011).

In summary, GT is not just a "qualitative data analysis method" - GT encompasses all phases of the research from sampling and data collection up to theory building, rather than simply data analysis. With reference to Peter Naur's "Programming as Theory Building" (Naur 1985), we conjecture that a software system is a theory about how it is constructed, structured, and operates. GT offers us a method that allows that theory to spontaneously "emerge" from data, using close reading, analysis, and repeated abstraction. A key tenet of GT is that "All is Data" - meaning that GT procedures can be applied to any phenomena under study - typically interviews or interview transcripts, but also recordings, documentation, or in the case of this work, programme source code, operations scripts, deployment assets, configuration management, and more. The insight underlying the REM method (see Section 2.2) is that the theory of a software system, i.e., its software architecture, the guiding principles underlying its design, can be discovered or recovered by applying GT as a "recovery" research method.

2.2 REM Method Outline

As previously stated, because GT lends itself well to qualitative-quantitative analyses and theory-building exercises, it can flexibly be applied to most if not all cases of theory-recovery exercises including software architecture recovery. This section maps GT onto REM (Henriksson and Larsson 2003; Sarkarati et al. 2008) for software architecture recovery. To devise a method featuring GT for architecture recovery, a necessary first step is to examine the definition of software architecture. Paraphrasing from (Bass et al. 2012): *"Software architecture is an abstraction of a software system comprising its elements, their*

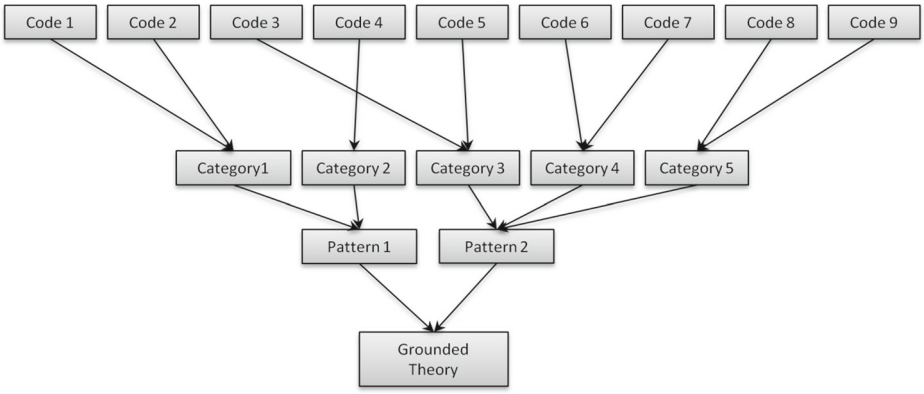


Fig. 2 Grounded theory in practise, discovering patterns (Khandkar 2011)

externally visible properties, their relations, and the choices that led to the system and its parts".

Software architecture, according to this definition, means:

1. The set of architecture elements (i.e., the set of components and connectors that are the essential parts of a software architecture specification (Malavolta et al. 2010));
2. Their visible properties (i.e., the important attributes of these elements);
3. The set of relations across architecture elements;
4. The abstractions realised by these architecture elements;
5. The set of choices that led to those elements, relations and properties;

general architecture recovery thus amounts to:

Recovery of software architecture for any software scenario, under any circumstance, with the sole assumption that source artefacts for the software to be analysed are available and analysable by means of visual inspection, possibly assisted by reverse-engineering and other tools.

Using these axioms as a basis, REM allows us to recover items 1, 3 and 4 of the above, whilst partially aiding in the elaboration of items 2 and 5. Section 2.2.3 focuses on how REM may be used to recover items 1,3 and 4. Later, Section 4 discusses how REM can be instrumented to aid in the recovery of architecture knowledge (e.g., architectural properties) and decisions, e.g., though further analysis of the elicited memos.

REM recovers architecture elements by systematically coding the entire set of available system artefacts. To bootstrap the coding, an initial set of codes (referred to as GT-codes from now on, to avoid confusion with actual software code) is inferred from whatever architecture knowledge is available. In our case, we used concerns (Lago et al. 2010) that originally drove the creation of the project under study—the FARC (see Section 3). Notwithstanding, a classical Grounded Theory methodology may also be applied without an initial set of GT-codes. For example, a pilot study on random areas of code or a reduced data sample may be used to generate an initial GT-codes list.

A key principle to drive this direct investigation is to *visually inspect and label source elements following the exact order with which the operational system is built and deployed.* This principle serves two purposes: (a) the build order is an expression of the dependencies that the software architecture has to uphold - architecture recovery efforts, therefore, need to elicit independent components and middleware (which are black-boxes in the architecture being recovered) before their dependent counterparts; (b) also, the build order allows for traceability of the recovered software architecture to make sure it is, in fact, reliably reflecting reality; this may not include all modules in the source code.

For example, assume you want to recover a modern cloud application X whose build and deploy script is specified in TOSCA, i.e., the standard "Topology and Orchestration Specification for Cloud Applications" (Wettinger et al. 2016). A valuable starting code-list may be inferred from the TOSCA specification itself, whilst the TOSCA blueprint for application X is most definitely the starting point for an architecture recovery exercise.

REM is arranged into four stages:

A) GT-Coding and Micro-Analysis This stage features the use of GT-codes as part of micro-analysis (Onions 2006) of source elements. The goal of this stage is to label source elements using GT-codes (e.g., assuming reverse-engineering to UML class or composite structure diagrams is possible, GT-codes can be expressed as stereotypes in a UML architecture recovery profile). Also, following the classical GT process, the stage includes the

incremental refinement of the GT-codes profile with insights, relations and memos learned iteratively from stage **A)**. For example, the use of UML can help ensure code coverage (i.e., allowing analysts to know what is being recovered, and where, across the system). Also, along with the use of UML to represent source code artefacts, a UML profile can contain GT-codes in the form of UML stereotypes. Profiles also make it easier to elaborate stereotype relations and characteristics as the GT process unfolds. The stage is iterated upon **until no unlabelled source element exists**;

B) GT-codes Categorisation This stage features an analysis of codes (e.g., as part of UML profiles, and potentially partially automated) from stage **A)** via the constant-comparison principle (Corbin and Strauss 1990). The goal is to generate "saturated" categories, i.e., including as many GT-codes as possible;

C) Recovery of Architecture Elements This stage features an analysis to understand structural decomposition relations *across* categories and core-categories generated as part of GT, to identify architectural concepts;

D) Theoretical Coding of Relations and High-Level Abstraction This stage features a step of constant observations and comparisons that produces the software architecture abstractions we seek. For example, UML can be used to instrument an instance of the above GT process that can be partially automated with model-to-model transformation to search recurrent clusters and interfaces to produce a corresponding UML component diagram and/or a composite-structure diagram. By enacting this process we discover any orphan elements or ambiguities.

2.2.1 GT-Coding and Micro-Analysis

REM begins by elaborating an initial list of GT-codes, using architecture knowledge elicited from existing documents, or by conducting a pilot study. Micro-analysis consists of visually inspecting every source element, analysing and establishing the source element's role as part of the system. The goal of this analysis is to label every artefact (an entire Java class, for example) that reflects a concern or a function. As part of this process, analysts are required to annotate source elements with "memos"—notes about possible structural decomposition properties or other characteristics typical of software architecture (e.g., modularisation (Baldwin and Clark 2000)). In our case-study, for example, the REM GT-coding list started from a list of seven stakeholder concerns (Lago et al. 2010) that drove the design and implementation of FARC years before (e.g., front-end decoupling, availability, etc.). Using these seven concerns as initial GT-codes we started a micro-analysis of the available software code artefacts (which we call source elements from now on).

In addition, as part of REM GT-codes definition, every GT-code includes a mnemonic aid identifying its meaning.

As previously stated, to aid the application of GT-codes throughout micro-analysis of reverse-engineered diagrams, GT-codes are realised as diagrams themselves. This aids REM in four ways: (a) GT-codes are easily applicable to reverse-engineered source elements in the form of diagram objects; (b) memos are more easily assignable to GT-coded source elements, e.g., as tagged values (Kandé and Strohmeier 2000) for defined stereotypes; (c) relations amongst GT-codes are easily captured in the diagram; (d) the profiles and annotations may be studied for further modernisation.

2.2.2 GT-codes Categorisation and Recovery of Architecture Elements

Categorisation of GT-codes (the topmost step 2 in Fig. 1) starts immediately after the first pass of micro-analysis coding is complete (i.e., after the phenomena are "labelled"—the topmost step 1 on Fig. 1). This pass typically generates hundreds of GT-codes (302 distinct labels, in our case). The REM categorisation step is conducted incrementally on these GT-codes, iteratively moving back and forth between the two artefacts involved, in our case: (a) the UML profiles containing REM GT-codes (to visually spot relations); and (b) the GT-coded source elements (to confirm the relations).

Also, during categorisation, the notion of architecture elements begins to emerge. In essence, the categorisation step uses memos—annotations on the possible structural decomposition and characteristics of the system (see Section 2.2.1) for the purpose of formulating modularity and decomposition hypotheses for analysed source elements (step 1 of "Selective Coding" in Fig. 1). These hypotheses are then confirmed or disproved by again inspecting the related source elements (step 2 of "Selective Coding" in Fig. 1). This approach is consistent with identifying what Baldwin and Clark (Baldwin and Clark 2000) call *design rules*, i.e., decisions that decouple architectural elements into related but independent modules. Note that this approach can benefit from automation beyond what REM offers in its purest form, e.g., by using architectural analysis tools such as Titan (Xiao et al. 2014).

The resulting process of categorisation (step 3 of "Selective Coding" in Fig. 1) concludes when: (a) no additional decomposition and modularity hypotheses can be formulated, and (b) there are no more memos left to aid hypotheses formulation.

The core-categories distilled in the process above match the definition of architecture elements (see Section 2), since they represent the sets of source elements clustered by semantic similarity and matched by *design rules* (Baldwin and Clark 2000). In other words, they are the *core parts of the system that cannot be further abstracted*.

2.2.3 Theoretical Coding: Recovering Relations and High-Level Abstraction

The theoretical coding part of REM (bottom part of Fig. 1) features the tool-assisted creation of a structural decomposition of the software architecture. Following the architecture-driven modernisation standard (Newcomb 2005), this view was realised, in our case study, using a UML 2.x component diagram and its creation was aided by model-transformation. For example, in our proof-of-concept application of REM, a series of 4 model transformations were sufficient for this purpose and acted jointly as follows:

1) Produce a UML package with one distinct package per core-category elicited during Selective Coding;
2) Produce a UML component diagram containing a component for every Core-Category;
3) Build relations in the component diagram across components that: (a) reflect Core-Categories with memo-ed and/or coded relations to other core-categories that emerged during GT-coding; (b) reflect relations found amongst code-constructs during reverse-engineering;
4) Produce interfaces for components that reflect Core-Categories grounded in source elements that identify system end-points by design (e.g., interfaces have public methods only).

2.2.4 *Reapplying REM*

In summary, the REM methodology is depicted in Fig. 3. The methodology can be summarised in compacted form as the incremental workflow of the following steps:

- **A)** [OPTIONAL] Elaborate initial GT-codes list, e.g., using ancestral architecture knowledge elicited from existing documents or conducting a pilot study;
  1. Use the GT-codes list to label reverse-engineered source elements, starting from programme build artefacts;
  2. Augment list containing codes with memos, i.e., code-inspection insights learned iteratively from step **A**);
  3. Repeat until no further source element exists;
- **B)** Analyse codes via the constant-comparison principle - the goal is to generate saturated categories encompassing all GT-codes;

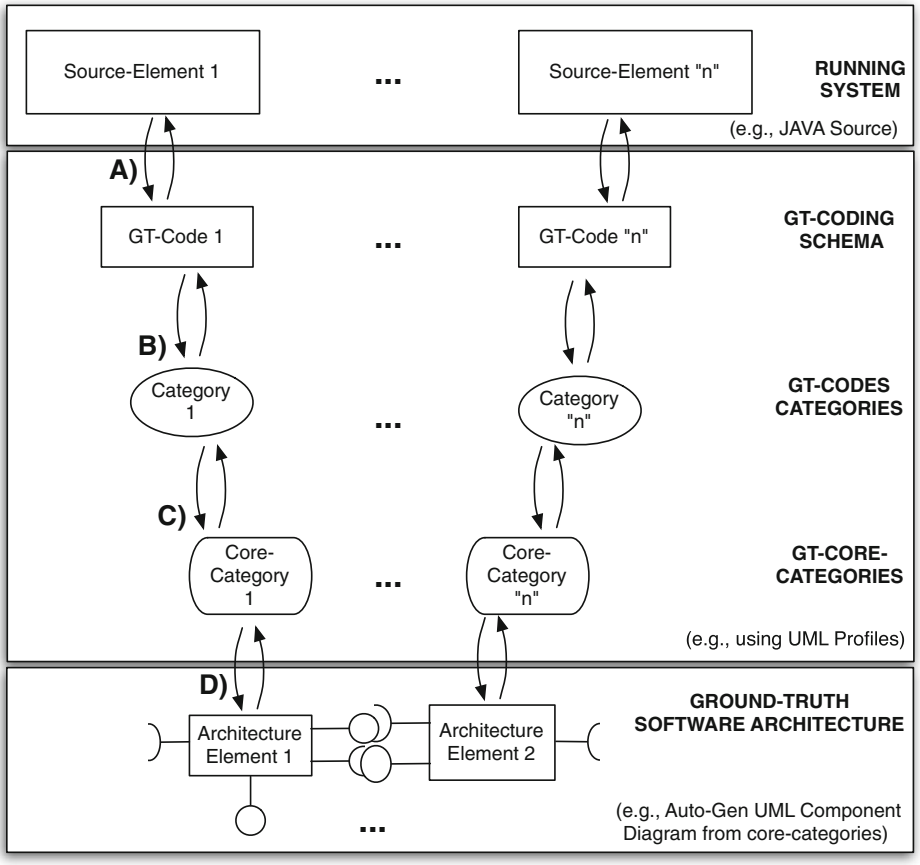


Fig. 3 REM in practise, recovering the architecture (grounded-)theory (Khandkar 2011)

– **C)** Analyse saturated categories via constant-comparison and pattern-matching principles - the goal is to generate saturated clusters of saturated categories;

3 Our Experience with REM in Action

For the purpose of proof-of-concept evaluation, REM was applied on an industrial product called FARC, provided by ESA during the RTE study. This evaluation was meant to show that REM is feasible and yields useful, actionable results. These objectives were addressed in two ways: (a) the recovered architecture was successfully used in the rest of the project for forward engineering and improvement of the FARC system; (b) our final deliverable (containing recovered architecture knowledge and structural decomposition views) was accepted after review by the original FARC designers and architects. This section introduces the FARC and outlines the application of REM for the purpose of recovering the FARC Architecture. Finally, the section concludes with a report of the qualitative evaluation that was set-up by ESA to evaluate and eventually accept REM methodology and results.

3.1 The FARC: a Brief Overview

FARC is a medium-sized (600 to 800 KSLoC) mission-critical ESA software system part of the SCOS 2000 ESA ad-hoc distributed SCOS-2000 Mission Control System (MCS). Quoting from the original design document for the FARC, "The purpose of the FARC is to store, retrieve and manage files such as configuration, database and binary files across any number of distributed sites. The FARC is able to maintain all file-based data within the SCOS-2000 runtime environment. Files are divided into two operational types - static configuration files part of the runtime environment and dynamic files updated at runtime".

3.2 Providing Crude Tool Support for REM

Several technologies were used to partially automate the application of REM. ESA requested a systematic assessment of the effectiveness of our automation technologies which was enacted by analysing the recovery contribution of the automated steps of REM against the total man-months spent applying REM. We established that the technology employed for REM effectively automated approximately 40% of the architecture recovery effort. The materials and technologies elaborated in this section are property of ESA but may be inspected online[8] for replication purposes only - further details and resources for verifiability are discussed in Section 3.8.

3.2.1 Reverse-Engineering

Reverse-engineering was carried out using closed-source parsing technology: Borland Together EA 2006 R2. This technology allowed us to obtain UML class diagrams from Java, CORBA IDL, and C/C++ sources. Moreover, our own Perl scripts were used to extract diagrams from testing specifications and deployment scripts. In addition, a number of Perl scripts and ad-hoc parsers were used to partially automate the recovery of FARC behaviour,

[8] http://tinyurl.com/q8rb8na

to be used as part of the architecture recovery. Reverse engineering and architecture recovery tools can also be employed for further automation, e.g., software modularisation and clustering tools such as Structurizr[9] or Titan (Xiao et al. 2014) may assist in the visual inspection of codes and/or the direct coding of modules.

3.2.2 UML Profiling

UML Profiles were used as a documentation vehicle for the GT-codes elicited as part of our case-study: stereotypes in the profile reflected the tags and labels used to aid the architecture recovery and its automation. Several profiles were developed in the scope of RTE. Most notably, PIMTAGS is the UML profile that contains the set of GT-codes applied as part of REM to the entire set of reverse-engineered source elements. The PIMTAGS UML profile first contained initial stakeholder concerns used to develop the FARC. It was expected that these would be reflected in the code and the architecture since they were the basic required properties of the FARC system. FARC was constructed imposing a strict separation of concerns between common system components (Frankel 2002), namely: *(i)* Data Formatting and Persistence; *(ii)* Database Handling; *(iii)* Presentation (GUI), *(iv)* Programming Language Specific Constructs; *(v)* Intercommunication Components and *(vi)* Core Business Logic; *(vii)* External Logic. The PIMTAGS profile initially contained these seven concerns and was incrementally saturated with other GT-codes as part of the GT study. In addition, the profile was augmented with notes and OCL constraints representing memos, that is, observations, relations and constraints reflecting empirical interrelations observed in source elements elicited during GT. The OCL language was used as a constraint language to aid further automation for architecture analysis.

3.2.3 Model Transformation

The Query-View-Transformation (QVT) language[10] was used as the model transformation baseline behind RTE. QVT transformations were developed to automate the recovery and representation of software architecture models as part of our REM exercise. The QVT transformations we elaborated are detailed as follows:

- *(i)* PSM_2_PIM.qvt, finds and clusters together in the same namespace all source elements bearing the recovered core-categories - a UML component diagram (see next transformation) is then used to capture the relations across namespaces clustering relations across source elements (e.g., dependencies, function calls, etc.);
- *(ii)* Model_construction.qvt, elaborates a UML component diagram with specific architectural components using clusters generated by PSM_2_PIM.qvt;
- *(iii)* Pack_2_component.qvt, realises architecture recovery by binding architecture components with appropriate relations and creates the related interfaces where needed, based on results of PSM_2_PIM.qvt;
- *(iv)* PIM_2_PIM.qvt, creates a reference between every component, interface, and relation in the output from Pack_2_component.qvt, to original source code elements (e.g., classes, scripts, etc.);

[9] http://www.codingthearchitecture.com/

[10] http://www.omg.org/spec/QVT/

– *(v)* PIM_2_EGOS_mapping.qvt, creates a reference between the structural representation of the architecture obtained using transformations *(i)* to *(iv)* with partially recovered behaviour from deployed sources - for this feature, we also devised a separate script to parse code behaviour (method calls and function invocations) into UML sequence diagrams, given a specified call nesting;
– *(vi)* CORBAinterfPub.qvt, produces CORBA IDL code for the interfaces in the recovered architecture.[11]

As an example of transformation behaviour, the PSM_2_PIM QVT script executes in a manner much similar to the pseudo-code below:

```
Begin: /* FARC side Arch-rec */
    var aux = select all classes from all nested packages
    in current project;
    divide(aux); /* divide classes according to stereotype
    and assign to one package per stereotype */
    clean_up(project);
    return(results);
End.
clean_up (in: UML project, out: UML project){
project.remove_unused_objects;
return(project);
}
divide (in: set of classes, out: set of packages){
var packages [number of stereotypes];
for (number of stereotypes amongst classes){
    build one package and assign it to packages[...];
    assign every class with current stereotype to current
    package;
    }
return(packages);
}
```

Finally, the application of all model transformations was automated using ANT scripts.[12] For the sake of space we do not discuss the model-transformations themselves but limit ourselves to discussing their operating principles.[13]

3.3 Applying REM for FARC Architecture Recovery

This section elaborates on how REM was applied to FARC in the scope of the RTE ESA study.

3.3.1 REM: Study Costs and Facts

Applying REM for the modernisation of FARC, entailed four phases: (1) initial processing; (2) architecture recovery; (3) forward engineering; (4) acceptance. The entire study cost approximately 17 person-months (PM), with a team of 6. This cost was split as follows:

[11] A number of other auxiliary model-transformations were devised but their structure and application is not discussed here for the sake of space.

[12] http://ant.apache.org/

[13] Further references on model transformations are available here: http://tinyurl.com/q8rb8na.

1. *Initial REM elaboration*: 1 PM - during this period we studied GT approaches and refined a possible contextualisation of them within the RTE study, thus eliciting and distilling the process we outlined in Section 2;
2. *REM instantiation*: 4.75 PMs - this process was carried out by 2 people working in parallel on the same source artefacts, for triangulation of results and subsequent inter-coder reliability assessment (Antoine et al. 2014); this assessment was required by ESA stakeholders to increase accuracy and was later praised during the final acceptance workshop;
3. *re-design and forward engineering*: 2 PMs - this process entailed using the recovered architecture assets to refactor the legacy system into a modernised version of at least 1 proof-of-concept feature, in our case, we chose to refine the distributed transitive clock synchronisation feature;
4. *testing and integration*: 1, 2 PM - this process entailed testing the re-engineered transactive clock synchronisation feature as part of the legacy application and redeploy the FARC in a clean-room operational environment;
5. *deployment and acceptance*: 1 PM - the previous step was reiterated as part of the final acceptance workshop and in the presence of the system's operators and space-operations controllers.

3.3.2 REM: Organisational Structure

The recovery team consisted of two senior software architects and two junior researchers/developers as well as two expert developers. Initial GT-coding was carried out in parallel by the software architects and junior researchers. Afterwards, for the purpose of assessing inter-coder reliability (Lavrakas 2008), visual inspection of conflicting coding instances selected for further discussion was carried out by expert developers and discussed as a group. *No FARC software architect was involved in the recovery team.*

3.3.3 Processing FARC with REM

Following the REM, the FARC system was reverse engineered as follows.

- **A)** At the kick-off of this stage, the FARC sources and scripts were packaged by ESA and sent to the four analysts. As part of GT-coding and microanalysis, an initial interview was held with the original FARC designers to elicit the basis from which to construct the initial set of GT-codes (Fig. 4). As a result of the interview, the initial version of a UML profile called PIMTAGS was developed, containing 7 GT-codes to be applied manually on reverse-engineered UML diagrams. For example, the excerpt on Fig. 5 is extracted from the PIMTAGS profile and identifies three clusters of GT-codes reflecting architecture links (AL_2_PIM), programming-language specific constructs (PL_2_PIM) as well as platform-independent constructs (PIM). Moreover, this excerpt highlights a memo in the form of an OCL Constraint (topmost box on Fig. 5). Finally, GT-codes inside PIMTAGS were associated with explanatory comments and UML notes (see lower-angled comment box associated with the AL_2_PIM element) as well as properties that allowed for their traceability to source elements ("trace2PIM" or "PIMComponentFlag").

  Concurrently with source element microanalysis, a market scan was initialised to find reverse-engineering technology that was able to extract UML diagrams from the source elements at our disposal, featuring, Java, C++, C, CORBA, Unix shell script,

```
 * Created on   : 11-Jul-2005
 *
 *      LCC            SPR   Programmer      Date
 *
 * ------------------------------------------------------------------------------
 */
package esa.egos.farc;

import java.awt.Button;
import java.awt.Dialog;
import java.awt.GridLayout;
import java.util.ArrayList;
import java.util.Iterator;

import org.w3c.dom.Text;

/**
 * Advanced query dialog allows user to fine tune their query.
 * @stereotype PL_2_PIM
 */

public class FARCadvancedDialog extends Dialog {

   // constants
   private final static String OPS_TYPES[] = FARCfileDefinition.OITEMS;
   private final static String ENCODING[] =  FARCfileDefinition.EITEMS;
   private final static String STATE[] = FARCversioned.STATES;
   private final static int CREATION = 0;
```

Fig. 4 Sample application of a single GT-code on a single source element

and Ant scripts. Micro-analysis was started after the preliminary version of PIMTAGS and reverse-engineered diagrams were produced. Figure 4 highlights a sample application of a GT-code from PIMTAGS, i.e., in the form of a stereotype. In this case, the stereotype was applied to the entire class.

Through micro-analysis an additional 47 GT-codes were introduced. These additional GT-codes reflected further design and implementation concerns addressed by GT-coded source elements. For example, the FARC-CLIENT-GUI or FARC-CLIENT-DAEMON were examples of GT-codes addressing visualisation and event-based reaction concerns.

- **B)** and **C)** Throughout these stages, the set of profiles produced as part of micro-analysis were analysed for the purpose of categorisation, following the GT principle of *constant comparison*. Model transformation was used to partially automate and assist the constant comparison activity by identifying GT-codes with identical structures and

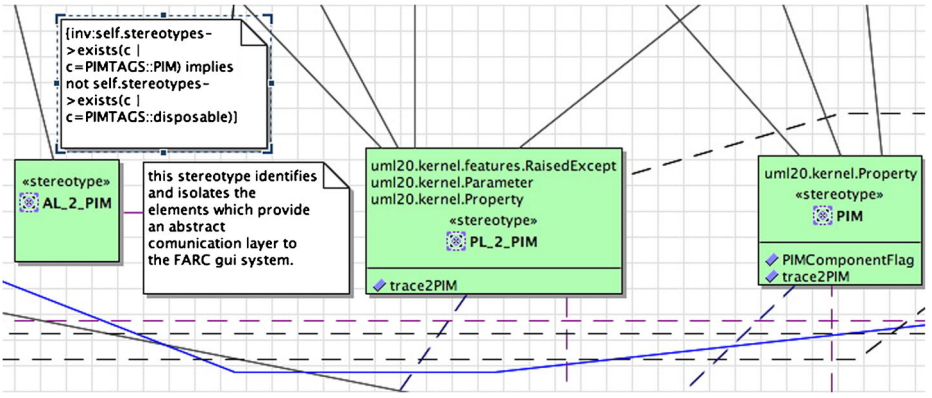


Fig. 5 Sample extract of the UML GT-codes profile for FARC called PIMTAGS UML profile

relations. As a result of this activity, GT-codes that were part of the same category were tagged with the $<< CategoryName >>$ stereotype and clustered together by means of model transformations into a single UML package called $<< CategoryName >>$. Figure 6 shows a small sample of source elements labelled with GT-codes and clustered into core categories.

As we compared codes to find categories, we observed that multiple codes were clearly related, according to memos noted throughout the micro-analysis step (step 1 of "Open Coding" on Fig. 1). At this point, we went back to inspect the source element to which every memo was associated. The reasons for this step are: (a) we needed to confirm or disprove our modularity and decomposition hypotheses; (b) we needed to further elaborate on the categories found and their mutual relations. For example, during re-analysis, we noticed that many categories had modularity and decomposition relations of their own and had "refinement" relations, i.e., some categories were properties or extensions of other categories. Core-categories emerged as part of this elaboration of categories.

- **D)** At this stage, the set of UML packages created as a result of stage B) were processed through automated model transformation as follows:
  - (a) a component was created per every package in the input set;
  - (b) the component was populated with the contents of the package so that internal component structure reflected package contents;

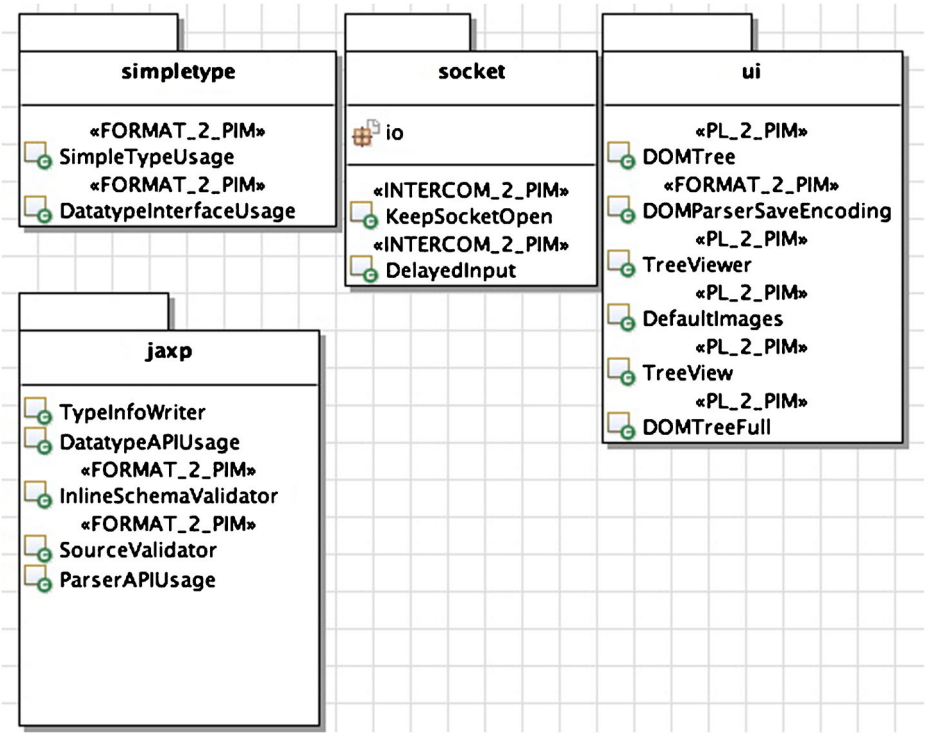


Fig. 6 Sample application of GT-codes and categories

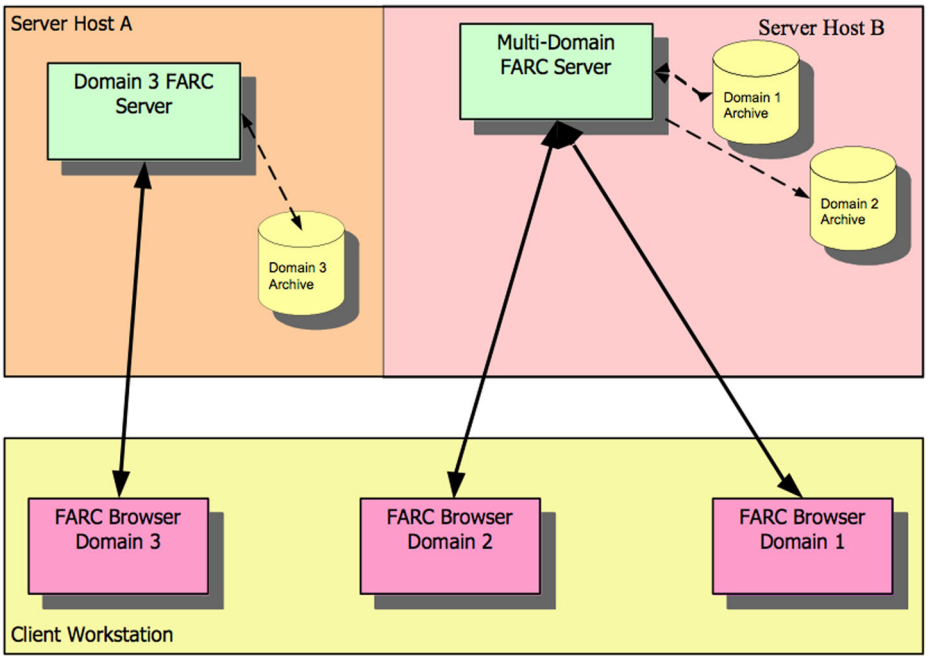


Fig. 7 Highest-level software architecture representation for FARC: a mock-up

– (c) interfaces, systems endpoints and appropriate relations were created whenever package contents were related to other packages/components;

At this point, we reported to ESA stakeholders that the REM architecture recovery elicited a highly-reliable software architecture description (following stages A to D) that differed considerably from the component structure expected by the ESA FARC stakeholders that we interviewed initially. This and similar mismatches were a critical outcome of the RTE study and are discussed further in Sections 3.5 and 4.

3.4 A Highly-Reliable Software Architecture for FARC

The result of the REM process as applied to the FARC case-study is mocked up in Fig. 7 and refined in Fig. 8. The figure shows a high-level overview of the software architecture that the RTE project recovered for FARC, represented using a simple box-line notation resembling a Deployment-Diagram.[14] Figure 8 further zooms into the diagram in Fig. 7 to show the internal structure of the Multi-Domain FARC Server. Both of these figures were realised by drawing a box for each architecture element reported in our recovered architecture, and then clustering together the relations between those elements and representing those relations with a single arrow.

This Figure highlights the structure of the FARC architecture and its deployment. In particular, the FARC followed a Multi-Tier architecture pattern since Clients (see bottom part of Fig. 7) as well as Server components are replicated and redistributed across a number

[14]Component and class names have been changed to avoid disclosure of protected information.

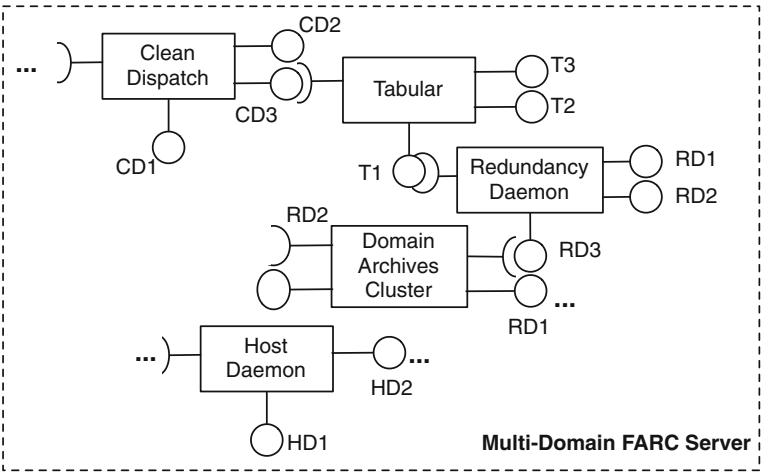


Fig. 8 A detailed view of the multi-domain FARC Server internal structure, as recovered by REM

of data domains in the European Space Operations Ground Segment (i.e., the 3 stations for which the FARC was designed). As we elaborated the set of architecture elements, relations, and their properties, over 20 architecture features and flaws were discovered, for example:

1. the GUI component is further elaborated into three sets of features mapped to three sets of data (i.e., domains) that FARC is capable of handling.
2. there is a logical and physical subdivision between an area of the architecture (e.g., domain 3) and the rest of the architecture. This is due to the sensitive nature of data addressed in domain 3.
3. we found that the FARC server component was designed to operate in single- as well as multi-domain mode beyond the current mode of operation (i.e., 3 domains).
4. we observed that 3 interfaces across the FARC Server component were securing the messages they exchanged (through marshalling and un-marshalling) rather than securing the intercommunication channels. This was due to a misinterpreted requirement. This, and similar misinterpretation insights we discovered, were deemed critical for the FARC renewal project and were praised by the stakeholder.
5. we observed that 18 APIs in the FARC architecture were implemented both in C++ and Java. There appears to have been an unclear requirement at the beginning of the FARC project concerning which language was more convenient.
6. we observed that an entire area of the FARC architecture, including several sub-components of the dedicated domain FARC server system (see top left-hand side of Fig. 7), could not be migrated to the new ESA reference architecture (EGOS-MF[15]) other than through a massive restructuring. The legacy area in question was designed to intercommunicate with an old and established product from the standard ESA operating system that was deprecated in the new ESA reference architecture.

Upon final evaluation of the recovered architecture we were required to outline our approach and key results to a focus group composed of 3 FARC operators and 2 FARC systems modernisation architects. The existence of the team of modernisation architects was

[15] http://tinyurl.com/jl39tec

obfuscated by ESA stakeholders; any interaction with our project was deliberately prohibited to avoid bias. We were also required to evaluate the REM process and key results of the RTE study against our mission objectives as part of a final acceptance workshop in a meeting with the RTE study participants, FARC engineers and ESA personnel related to FARC. This evaluation was initiated when the FARC focus-group had evaluated the REM case-study results. The RTE mission objectives were:

1. *REM could be reused in any other architecture recovery campaign across the ESA consortium;*
2. *The REM evaluation case-study could be reused as a basis for ongoing FARC systems modernisation, provided the necessary patches were applied to harmonise its architectural structure and properties with the new ESA Standard Reference Architecture;*
3. *All REM results were accepted by FARC systems modernisation architects and reused as insights for the purpose of modernisation planning.*

The next subsections elaborate on the acceptance focus-group and subsequent workshop. These served as qualitative evaluation of REM, its results, as well as the entire RTE study.

3.5 Evaluating REM: A Focus-Group Study

The objective of the ESA focus group was to evaluate wether the architecture elicited as part of REM results actually reflected the deployed FARC architecture. To facilitate this evaluation, we prepared a presentation of the architecture recovery results. This presentation was arranged according to Kruchten's 4 + 1 software architecture views model (Kruchten 1995b). After our presentation we were presented with an overview of the actual FARC architecture component model (see Fig. 9).

The evaluation of our results by the ESA focus group reported the following:

- the RTE team managed to recover and re-document knowledge concerning the architectural decomposition of FARC and its components;
- the RTE team managed to recover and re-document knowledge concerning the major distribution and interfaces across the FARC architecture;
- the RTE team managed to recover and re-document knowledge concerning the peculiar Domain-based file-management characteristics within FARC;
- the RTE team failed to recover and re-document any knowledge of the checkpointing and security management aspects of FARC;
- the RTE team managed to recover and re-document knowledge concerning the mirroring and replication facilities of FARC – this knowledge may be vital for further modernisation of FARC;
- the RTE team managed to recover and re-document the current FARC API layout – efforts were invested in modernising these APIs towards Java5 by means of JET[16] technology;
- also, as part of the FARC API recovery, the RTE team successfully isolated a critical architecture flaw and security risk connected to the existence of a partially implemented (but exposed) set of Java APIs whose behaviour mirrored their official C++ counterparts. This hazard and similar cross-cutting concerns isolated as part of the application of REM reflected several FARC software architecture flaws in modularisation;

[16] https://eclipse.org/modelling/m2t/?project=jet

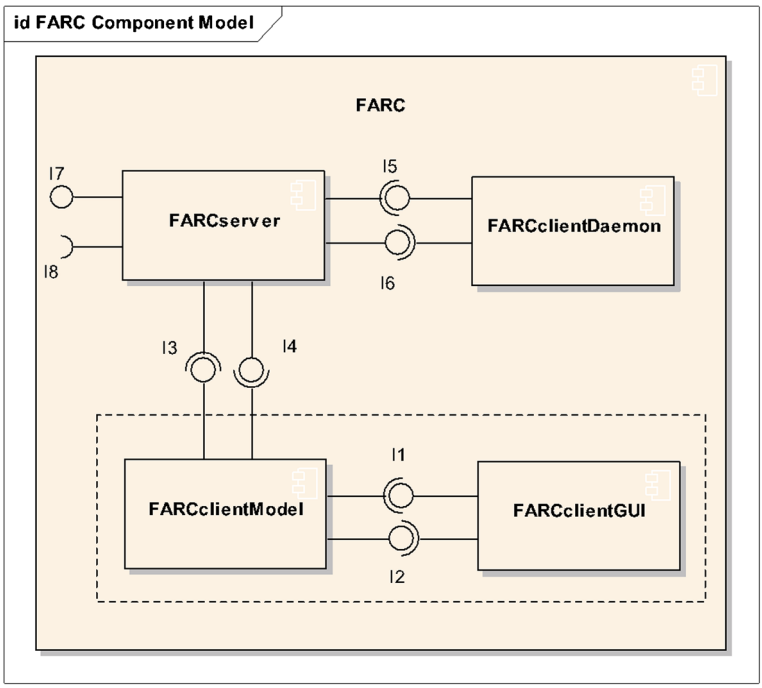


Fig. 9 The actual FARC component model

– the RTE team failed to recover and re-document networked consistency management aspects arranged across highlighted FARC components (see the dotted box in Fig. 9);

The evaluation was, on the whole, positive and the REM-FARC results were officially accepted. However, the evaluation also showed that REM needs additional refinement or further research to create mechanisms capable of eliciting functionality that is not reflected in any localised structural property, but rather emerges from interactions amongst components of the architecture, e.g., security checkpointing or consistency management.

3.6 REM Final Acceptance Workshop

Fifteen days after the ESA focus-group evaluation of the REM and its application on FARC, the entire RTE project ecosystem was required to undergo final acceptance. Four presentations served as introductions for: (a) the REM method; (b) the foundations behind REM and supporting tool-chains; (c) REM results and FARC knowledge recovery and modernisation; (d) recommendations and lessons learned. A sample presentation is available online.[17]

As part of the final acceptance workshop, RTE results and processes were evaluated against against mission objectives as follows:

1. **REM can be safely reused in any other architecture recovery campaign across the ESA consortium**: "the REM methodology is indeed generic enough to encompass any architecture recovery process to be enacted for the purpose of ESA systems

[17] http://tinyurl.com/j8x3pux

modernisation. RTE recommendations and automation facilities, however, may only serve as starting points to be configured if further experimentation with REM may be needed. Refinement of RTE results may be needed to extend REM beyond architecture recovery". *In summary, we were able to tackle our mission objective 1, however the automation tools that we provided were, according to ESA stakeholders, only partially fulfilling the goal of being highly generalisable; the automation was too focused on the specifics of the FARC case-study. As a consequence we were given a 30-day probation period to refactor the automation tools and accommodate them for further review. These followup results and refactored transformations are not discussed further in this article.*

2. **The REM evaluation case-study, i.e., the FARC-EGOS component can be reused as basis for ongoing FARC systems modernisation**: "The REM methodology and RTE team offered a great deal of knowledge from which additional insights and modernisation features may be refined in the current FARC-to-EGOS-MF modernisation project. In particular REM results concerning the API structure, dependencies and technological modernisation serve as great additions to the project's assets". *In summary, we were able to provide valuable knowledge beyond expectations from both FARC modernisation architects and current FARC operators. Essential knowledge was recovered that can drive architecture reasoning or modernisation in FARC, by means of the insights we provided.*
3. **All REM results shall be accepted by FARC systems modernisation architects**: this objective was discussed in more detail in Section 3.5.

3.7 REM: Venues for Future Automation

Improving the automation of REM was one of the key goals identified by ESA stakeholders in response to our final methodology presentation. Therefore ESA requested that we explore approaches for additional automation in the future. To this purpose we conducted bi-weekly focus groups over a period of 45 days. Focus groups involved the RTE team, software architects originally part of the development of FARC, as well as other ESA stakeholders for the RTE study (1 senior manager and 2 experts). Workshops entailed an in-depth presentation of the study phases to brainstorm proposals for further automation. Results were then analysed by the RTE team using a storytelling approach (Corbin and Strauss 2008). As a final outcome of this process, four ideas were distilled as the most reasonable and immediate routes to improving automation:

1. *Auto-scripts as memos to be run on all models* - our conjecture here was that by coding observations or memos in the form of OCL (or another object constraint language) the resulting constraints could be applied to models worked out as part of the application of REM, with the objective of partially automating the discovery of similar memos. This was explored in RTE but never tested.
2. *Profiles as input for auto-coding model-transformations* - our conjecture here was that UML profiles developed as part of GT-coding on a substantial part of source elements could become input to a model-transformation that would: (a) scan the remaining source elements; (b) apply the constant comparison principle automatically; (c) measure and report a confidence evaluation. This venue for further automation was never explored beyond the formulation above.
3. *Ad-hoc query transformations* - our conjecture here was that model transformation could be used for automating the"querying" of available models, e.g., to prove or

refute the structural dependency or decomposition hypotheses discussed earlier in Section 2.2.2. The starting point for this conjecture was that the *Query* part of the Query-View-Transformation language, the baseline transformation language for REM, was not actually exploited as part of the elaboration of our modernisation methodology. Indeed, at the time RTE was conducted, no implementation of the *Query* part existed. Further exploration of this approach is needed to confirm or disprove this conjecture.

4. *MultiREM, i.e., Multiple REMs for multiple software architecture views* - our conjecture here was that REM itself could be applied over and over again to incrementally refine the architecture knowledge elicited as part of the process. The starting point for this conjecture was that the REM approach, as it was outlined in the previous sections, focuses on a structural decomposition view (Kruchten 1995a) of software architecture whilst other architecture views remain implicit. However, the REM approach could easily be re-focused to analyse artefacts other than source elements for the purpose of recovering other architecture views. This conjecture however, was not explored further.
5. *REM-of-REMs, i.e., a scalable, multi-observer, GT-based architecture recovery* - in an effort to scale up approaches such as REM, multiple teams could work concurrently on pre-arranged divisions of the source artefacts. Because GT-based approaches are theory-building approaches they lend themselves to combining efforts, as shown in previous research (Suddaby 2006). Further experimentation in this respect would be needed to test and refine this process.

3.8 Verifiability

To encourage the generalisability of our results as well as the verifiability of our claims, we formally requested and obtained the permission to share the FARC source code for further experimentation in the scope of this study. Considering the exploratory nature of REM, we clarified to ESA stakeholders responsible for REM and FARC secrecy that sharing the code and methodology with a wider audience may encourage further refinement and automation of the methodology and further investigation into the FARC case-study. As a result, the stakeholders allowed us to share all documentation and sources online.[18] These artefacts are still under licence however, and are available for study-replication purposes only.[19]

4 Discussions, Observations, and Lessons Learned

We learned several lessons concerning architecture recovery research and practise from the RTE study. Also, a number of discussion points emerged as part of REM and the experience reported in the previous sections.

1. Architecture recovery approaches must be linked to a business case. As part of RTE we learned that industrial stakeholders are, understandably, concerned with how much of the system can be recovered and what is the trade-off between the recovery effort invested and re-development. Therefore, software architecture recovery approaches need to make their costs explicit and tie these to benefits. These concerns must be addressed at every stage,

[18] http://tinyurl.com/lf876f7

[19] The authors shall be informed of every study-replication effort, so that ESA stakeholders may be involved if needed.

starting from study planning and terminating with whatever recovered artefacts are produced. Our goal should therefore be to estimate, up-front, what kind of return on investment can be obtained by applying a reverse engineering tool or method.

Our Experience Report In the case of RTE and our application of REM, because the code-inspection time could be estimated by analysing how long each activity took for our (relatively unexperienced) analysts, we were able to devise a cost-estimation mechanism tailored from Thomson (2011).
What we learned Further research is needed to elaborate and validate such a mechanism. Regarding this point we observed that, although REM is relatively expensive and human-intensive, it provides a great benefit that many other approaches do not and that companies desire: a *cost upper bound*. In fact, the cost of the REM activity compares favourably with state of the art approaches such as Lutellier et al. (2015) where authors stated that a 2-year longitudinal study involved a team of 6 researchers and extensive interviews with software architects to analyse a 10 MSLOC system. Although the effort is comparable between their study and ours, we were able to produce our estimate a mere two weeks into the study; the same cannot be said for other approaches, such as Lutellier et al. (2015). Also, the instantiation of the approach in Lutellier et al. (2015) would likely yield cost estimation figures which are intrinsically bound to many context-specific variables (e.g., programming languages involved, level of applicability of previous automations, level of confidence, etc.). In a GT-based approach, the *only* variable is source-code quantity. Nonetheless, beyond these preliminary conjectures, it is clear that more research must be invested in validating the cost of REM.

2. Architecture recovery is even more a human activity than software architecting itself, therefore the only general approach possible is the human and machine-aided approach. As part of RTE we learned that recovering software architectures and related artefacts means also to capture the design rationale in the minds of the developers. This human involvement activity is an essential part of the recovery process and should therefore be taken into account when developing any modernisation proposals.

Our Experience Report During the evaluation of REM, for example, we learned that certain FARC developers used an ad-hoc style preferring certain coding practises, design patterns or modular decompositions rather than other ones. Therefore, the team that had to run the modernisation attempt (i.e., REM) could be prepared with human and organisational insights from the development communities that originally developed the product. In addition, whilst applying REM we observed that certain architecture decisions could be easily recovered by further analysing the memos and observations we made during GT-Coding. For example, some developers decided to adopt a modularisation strategy that was never recorded anywhere but became evident from the memo-ing exercise. The point here is that no tool is able to convey this architecture knowledge automatically, or even semi-automatically.
What we learned Further research of these knowledge recovery venues is therefore in order.

3. Fully automated approaches tend to be viewed with scepticism by industry stakeholders. Industrial stakeholders who decide whether architecture recovery is of value are

frequently non-technical and often neglect the value of automation in favour of the (more easily assessed) work of humans.

Our Experience Report When we first showcased the REM methodology to our ESA stakeholders, to our amazement we received little scepticism when we outlined the substantial manual effort required. Furthermore, in subsequent presentations on our intention to explore further automation (rather than completing the evaluation of REM as quickly as possible) we learned that too much automation might be viewed with scepticism and "mistrust in the machine".
What we learned Exploration of this human and organisational dynamic is part of our plans for further research, to establish principles for designing architecture recovery tools and methods. We further elaborate these in Section 7.2.

4. Architecture recovery approaches should also aim to reconstruct organisational structures for software products. Those structures are likely to be needed by project decision-makers to maintain the renewed code-base.

Our Experience Report Part of the recommendations requested from the RTE study were considering human and organisational structures necessary for the proper operation of the modernised software. As the RTE team discussed this request, we learned that architecture recovery attempts could be guided to elicit the organisational structure (Tamburri et al. 2013b) of the development community (Tamburri et al. 2013a, 2016) so as to recommend a structure to support the software to be modernised.
What we learned Although we did not look into this organisational recovery activity as part of RTE, this is an obvious and important research area, consistent with the emerging necessity of complex organisational structures for development and operations (e.g., Cois et al. 2014), (Herbsleb and Grinter 1999).

5. There exists a spectrum of applicability across which any architecture recovery approach can be categorised. Whilst defining this spectrum is beyond the scope of our article, we observed that, based on the number and type of assumptions that recovery approaches make over the system to be analysed, these approaches could be typed and clustered together. In this imaginary spectrum, REM would be one of the bounds, since it makes no assumptions for the targeted system. Similarly, other approaches' applicability can be measured and quantified in terms of relative distance to REM, simply counting how many assumptions are made over the system to be recovered.

Our Experience Report Within the context of RTE we were required by the industrial stakeholder to consider and use for recovery every asset in the FARC system, including: (1) Simulink maintenance models; (2) operations scripts in a closed-source shell-script variant for a fixed SLES9 OS version; (3) textual and box-and-line documents and models; (4) informal change-logs noted in user-manuals. The heterogeneous nature of this information was a key reason why we resorted to creating REM in the first place.
What we learned REM may be considered a *last-resort* methodology which offers broad applicability but at potentially substantial costs.

5 Theoretical Limitations and Threats to Validity

There are several limitations and threats to validity intrinsic to the study and results reported in this paper.

5.1 Approach Limitations

Although we have claimed that REM is a general approach, we cannot *prove* this claim. Our case study shows that the approach does in fact *work*, but no single case study can show that a method *works generally*. An appraisal of the generality behind REM is rooted in a discussion of the general application of grounded-theory, which is beyond the scope of this paper. Further research could investigate how to demonstrate generality. In this paper we simply offer one datapoint that is consistent with our claim of generality, resting upon the foundations of GT. Furthermore, the tool support that we realised in our proof-of-concept experiment suffers from a single-point observation limitation, i.e., it was designed to address a specific problem. Although much of the tool support in question does not hard-code any reference to FARC or RTE, and could potentially serve other instances of REM applications, we cannot claim anything beyond the tooling's usefulness in the proof-of-concept. Further technical evaluation and replication of REM in different contexts, both industrial and open-source, would be needed to properly assess this tooling and its generalisability. For example, our tool support is inextricably linked to the use of Model-Driven Architecture technology which is bound to function in an object-oriented fashion.

5.2 Threats to Validity

Assessing the validity of qualitative research is a subject still thriving of investigation and speculation, most prominently in social and organisational research. We focus on he major assessment school proposed by Denzin and Lincoln (2011) and related literature, which stresses the rigour of results interpretation. Here below follow a list of distresses we identified for our work along the lines defined by said frameworks.

Contextual Validity Paraphrasing from Bloor (1997), *contextual* validity refers to "the credibility of case study evidence and the conclusions drawn. The primary focus of such research is to capture authentically the lived experiences of people and to represent them in a convincing text, which demonstrates that the researcher fully understands the case". In our case, the methodology previously outlined, the results, conclusions, lessons learned, and insights reported in this paper stem from self-ethnography and an accurate analysis of final delivery reports obtained as part of the RTE study. The REM methodology was accepted as part of the BSSC[20]-based organisational and methodological baseline adopted by ESA and all descriptions are consistent with the material depicted in this pages; hence, REM represents a valid contribution with respect to the context from which it draws.

Generalisability Validity Paraphrasing from Myers (2009), *generalisability* validity refers to whether the research results are *transferable*, i.e., can be extended to a wider context, have theoretical generalisability, empirical applicability and practical usefulness. In our case, whilst the proposed REM method can in fact be extended to a wider context by

[20] http://www.esa.int/TEC/Software_engineering_and_standardisation/TECA5CUXBQE_0.html

definition and already a plethora of cases that confirm its root-method GT empirical applicability, the extensibility of our evaluation of REM in action does suffer from generalisability validity. On one hand, the proposed methodology is claimed to be general by design but, on the other hand, we dive deep into a single case-study wherefore the methodology was applied. This limits greatly the generalisability of the case-study observations, results, and lessons learned but to the best of our knowledge, *does not limit or hinder in any way the generalisability of the proposed method which is general by design*. This not withstanding, our results and lessons learned encourage further research in the direction of assessing the degree to which REM, a qualitative analysis-inspired semi-automated recovery approach, can actually be generalised across multiple, sufficiently diverse case-studies.

Procedural Reliability According to Miller et al. (Silverman 1997), procedural reliability or validity generally refers to results consistency, typically meaning that another person should be able to examine the object of study and come to similar conclusions. In our case, our conclusions are twofold (1) that REM is a theoretical hypothesis for an architecture recovery methodology which is general *by design* and (2) that we offer a single datapoint to validate such a hypothesis. Although the methods and procedures employed towards both conclusions rely on established practises in qualitative research (e.g., GT, inter-coder reliability (Lavrakas 2008) and triangulation, etc.) we cannot prove that there exist no other possible interpretations of our case-study and our proposed methodology. However, according to Teddlie and Tashakkori (2009), researchers need to establish weather their research design and procedure correctly addresses the "Did we accurately capture/represent the phenomenon or attribute under investigation?". In our case, our case-study report relies on direct and fully-embedded in-vivo experience report by direct observation. Moreover, the formulation of the REM methodology stemming from that study has been reviewed by researchers and practitioners involved in the study, confirming its procedural reliability.

6 Related Work

Architecture recovery has been a topic of research for several decades, stretching back to the 1990s. Early approaches, e.g. Guo et al. (1999), Kazman and Carriere (1999), Kazman and Carriere (1998), and Kazman et al. (1998) tended to focus on the artefacts that could be recovered, and the capabilities and limitations of existing tools, rather than on the methodology of recovery per se. However, both dimensions did emerge even in this early research.

Currently, research in modernisation approaches is based around some clustering or pattern recognition technique such as Medvidovic and Jakobac (2006) by Medvidovic et al. or Vasconcelos and Werner (2007), this last one being part of the ARES initiative.[21] A lot of such efforts were captured effectively in the survey by Pollet et al. (2007). Also, almost in parallel with our own efforts at ESA, the OMG Architecture-Driven Modernisation taskforce began the work that would later result in the Knowledge-Discovery Metamodel ideas and related works such as the MoDisco toolchain.[22] Works in software architecture recovery most closely related to REM are reported below.

[21] http://www.ares-nest.org/tiki-index.php

[22] https://eclipse.org/MoDisco/

Medvidovic et al., in Medvidovic and Jakobac (2006) and subsequent works (e.g., Lutellier et al. 2015), provide a lightweight approaches to architectural recovery based, for example, on particular clustering methods which isolate functional components according to a number of heuristics on the system's reverse-engineered structure. The approach of using fixed heuristics is intuitively similar to the idea behind REM of using a Grounded Theory based exploration of source elements, leaving the heuristics to the realm of intuitionistic reasoning and visual analysis typical of Grounded Theory as in REM. The key strongpoint of our approach is that the quasi-heuristic approach in REM is part of a systematic and general approach to qualitative text analysis rather than code inspection. Indeed, the two lines of research (code inspection and GT-based architecture recovery) could very well benefit form interaction in further research, e.g., in refining and automating said heuristics and their application.

In addition, work in Software reflexion models is related to the work we carried out at ESA. In particular, the foundational work by Murphy et al. (2001) lays the foundations and motivations for efforts that we ourselves retained during our study. Similarly, the work in software reconnaissance by Gear et al. (2005) seems related to what we tried initially in REM but failed, i.e., the sort-of manual exploration of software code by means of computer-assisted approaches. REM and reconnaissance approaches may well be combined for further automation.

Furthermore, works that semi-formalise dependencies across architecture elements and any constraints thereof, may well be similar to what we tried to do in REM by means of OCL or QVT transformations. For example, works such as Terra and de Oliveira Valente (2009) by Terra et al. offers a framework to manage object-orientation. The advantage we propose within REM is that by means of OCL and QVT, any sort of dependency may be addressed, even though with something (i.e., OCL) which is much less formalised than an actual programming language.

Finally, an interesting related initiative by Sartipi et al. (2006), proposes an approach to architectural recovery focusing on extracting views, addressing different concerns. REM focuses on a structural dependency view of the software architectures much like Sartipi et al. focus on addressing multiple views and multiple concerns. The overlaps between REM and the approach in Sartipi et al. (2006) could be a valuable starting point for further research.

7 Conclusions and Future Work

Software architectures are key artefacts in the software lifecycle. Recovering legacy software architectures is important for a variety of reasons: to assess legacy software, possibly promoting reuse. However, research and practise in software architecture recovery agree that recovery endeavours often lack ground-truth, i.e., the representations of software architecture that can be *proven* to reflect the actual architecture. Also, research has illustrated a limitation of many existing approaches in that they target a limited number of scenarios and may require a substantial learning curve.

This paper has introduced REM, an architecture recovery method that: (a) may in fact be *general* by design; (b) was partially automated in the context of a large, industrial case-study. To evaluate REM we reported on the original study that spawned REM as part of an architecture recovery and modernisation endeavour at ESA, the European Space Agency.

Evaluating REM, we learned that it has several interesting limitations that should be subject to further study and validation. As part of the contributions reported in this paper, we

also outline the ways in which REM could be further automated as well as key messages to be employed for further research in REM and similar GT-based modernisation approaches.

7.1 Conclusions

From our object of study and its evaluation, we were able to conclude that: (a) REM represents the first key and concrete step towards general, highly-reliable software architecture recovery; (b) REM is to be considered as a *last-resort* tool; (c) our proof-of-concept experimentation showed that many "soft" dimensions of software architectures (e.g., organisational structure, knowledge management, etc.) are closely related to architecture recovery exercises.

7.2 Future Work and Research Roadmap

As a result of our study, we conclude that the topic of general, highly-reliable software architecture recovery has massive potential and, in the scope of REM, we offere a formulation for a wider research agenda in this direction. In this respect, we foresee the following 3 challenges, offering possible research methods:

1. *Formally-verified Generality:* providing a formal demonstration of generality behind a GT structured approach for architecture recovery would essentially prove the general nature of the method and confirm its practical value. This research venue may start from operationalising the REM process even further, providing formal definitions of each action and evaluating whether that action is always applicable, i.e., it makes no assumption with respect to the system under study. Mathematical analysis (e.g., formal concept analysis (Wille 2005)) and theorem-proving methods (Bibel 1982) may be used in this endeavour.
2. *Computer-assisted Architecture Recovery:* so far, architecture recovery has mostly focused on providing tools that study running software artefacts to provide tentative recovery of designs or design views. Conversely, approaches such as REM require a sensibly different target, namely, tools that assist the code-inspection process of recovering software architectures by direct observation. In this work, we refined crude and rudimentary tools for the latter purpose. These tools and their design may be used as prototype instruments to define improved versions. Research methods such as design science (Hevner et al. 2004) as well as action research (Boaduo 2011) may be used in this endeavour.
3. *Scalable General Architecture Recovery:* as previously theorised, REM could be applied iteratively and in parallel by several teams which are given a selected subset of code-artefacts belonging to a bigger system. Because REM does not rely on system correctness or completeness assumptions, its execution can be parallelised in multiple ways. An appropriate research venue can be, therefore, to establish which is the most effective way to apply GT-based approaches to architecture recovery at a large scale. From a research design perspective, a mixed-methods study involving multiple open-source communities may be instrumented. For example, several research groups can be given randomly selected parts of an open-source product whilst, at the same time a single team can carry out the entire recovery exercise. Likewise, this exercise can be repeated over time until a theory emerges.
4. *Architecture Recovery Accountability:* a major limitation we observed in the state of the art is that there exists no baseline for evaluation of architecture recovery approaches. In our endeavour we confirmed the validity of our method by using a proof-of-concept

experiment but we cannot currently compare our results with any available and comparable approach. On one hand, the key approaches to architecture recovery should be studied and their validity quantified at a large scale, and, on the other hand, approaches such as REM should be instrumented to provide comparable quantifications.

Finally, defining general methods for software architecture recovery also calls for systematic, precise and scoped definitions over what software architecture *ground-truth* actually is, and how that reflects on its recovery - critical questions such as "what is ground-truth software architecture recovery?" or "Can a system have multiple ground truths? If so, what are the criteria? And how many are there?" represent a Pandora's box over what software architecture represents and the exercise of recuperating it. These questions are a paramount challenge for the software architecture community at large, which is still open to address.

Acknowledgments We acknowledge the precious comments we received from the anonymous reviewers - they helped greatly in structuring the value and contributions in this manuscript. This research has been conducted in collaboration with TERMA GmbH[23] as an ESA GSTP study founded by ESA under contract 20645/07/F/VS. The author would like to thank Drs. Gert Villemos, Antonio Bianco and Henry Muccini for support during RTE. Damian's work is partially supported by the European Commission grant no. 644869 (H2020 - Call 1), DICE.

References

Antoine JY, Villaneau J, Lefeuvre A (2014) Weighted krippendorff's alpha is a more reliable metrics for multi-coders ordinal annotations: experimental studies on emotion, opinion and coreference annotation. In: Bouma G, Parmentier Y (eds) EACL. The Association for Computer Linguistics, pp 550–559. http://dblp.uni-trier.de/db/conf/eacl/eacl2014.html#AntoineVL14

Bachmann F et al (2000) Software architecture documentation in practice: documenting architectural layers. Special Report CMU/SEI-2000-SR-004 SEI CMU

Baldwin C, Clark K (2000) Design rules: the power of modularity, vol 1. MIT Press, Cambridge, MA

Bass L, Clements P, Kazman R (2012) Software architecture in practice. SEI Series in Software Engineering. Addison-Wesley. https://books.google.com/books?id=-II73rBDXCYC

Bibel W (1982) Automated theorem proving. Vieweg, Braunschweig

Bloor M (1997) Techniques of validation in qualitative research. A critical commentary. In: Miller G, Dingwall R (eds) Context and method in qualitative research. Sage, Thousand Oaks, CA, pp 37–50

Boaduo NAP (2011) Action research in virtual communities: how can this complement successful social networking? IJVCSN 3(4):1–14. http://dblp.uni-trier.de/db/journals/ijvcsn/ijvcsn3.html#Boaduo11

Clements P, Kazman R, Klein M (2001) Evaluatinjg software architectures: methods and case studies. Addison Wesley Professional

Clements P, Bachmann F, Bass L, Garlan D, Ivers J, Little R, Nord R, Stafford J (2002) Documenting software architectures: views and beyond. Addison Wesley Professional

Cois CA, Yankel J, Connell A (2014) Modern devops: optimizing software development through effective system interactions. In: IPCC. IEEE, pp 1–7. http://dblp.uni-trier.de/db/conf/ipcc/ipcc2014.html#CoisYC14

Corbin J, Strauss A (1990) Grounded theory research: procedures, canons, and evaluative criteria. Qual Sociol 13(1):3–21

Corbin JM, Strauss AL (2008) Basics of qualitative research, 3 edn. Sage Publisher

Denzin NK, Lincoln YS (2011) The sage handbook of qualitative research. Sage, Thousand Oaks

Ding L, Medvidovic N (2001) Focus: a light-weight, incremental approach to software architecture recovery and evolution. In: WICSA. IEEE Computer Society, p 191

[23] www.terma.de

Dueñas JC, de Oliveira WL, de la Puente JA (1998) Architecture recovery for software evolution. In: CSMR. IEEE Computer Society, pp 113–120. http://dblp.uni-trier.de/db/conf/csmr/csmr1998.html#DuenasOP98

Eixelsberger W, Ogris M, Gall HC, Bellay B (1998) Software architecture recovery of a program family. In: Torii K, Futatsugi K, Kemmerer RA (eds) ICSE. IEEE Computer Society, pp 508–511. http://dblp.uni-trier.de/db/conf/icse/icse98.html#EixelsbergerOGB98

Frankel D (2002) Model driven architecture: applying MDA to enterprise computing. Wiley

Garcia J, Krka I, Mattmann C, Medvidovic N (2013) Obtaining ground-truth software architectures. In: Proceedings of the 2013 international conference on software engineering, ICSE '13. IEEE Press, Piscataway, NJ, USA, pp 901-910. http://dl.acm.org/citation.cfm?id=2486788.2486911

Gear AL, Buckley J, Collins JJ (2005) Software reconnexion: understanding software using a variation on software reconnaissance and reflexion modelling. In: ISESE. IEEE Computer Society, pp 34–43. http://dblp.uni-trier.de/db/conf/isese/isese2005.html#GearBC05

Glaser BG (1978) Theoretical sensitivity: advances in the methodology of grounded theory. Sociology Press, San Francisco, CA

Guo G, Atlee J, Kazman R (1999) A software architecture reconstruction method. In: Software architecture (proceedings of the first working IFIP conference on software architecture (WICSA1)), pp 15–33

Gwet K, Gwet K (2002) Inter-rater reliability: dependency on trait prevalence and marginal homogeneity. Stat Methods Inter-Rater Reliab Assess 2:1–9

Henriksson A, Larsson H (2003) A definition of round-trip engineering. Tech. rep., Linkpings University, Sweden. http://www.ida.liu.se/~henla/papers/roundtrip-engineering.pdf

Herbsleb J, Grinter R (1999) Architectures, coordination, and distance: conway's law and beyond. IEEE Softw 16(5):63–70. https://doi.org/10.1109/52.795103

Hevner, March, Park, Ram (2004) Design science in information system research. https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research

Izquierdo JLC, Molina JG (2010) An architecture-driven modernization tool for calculating metrics. IEEE Softw 27(4):37–43. http://dblp.uni-trier.de/db/journals/software/software27.html#IzquierdoM10

Kandé MM, Strohmeier A (2000) Towards a UML profile for software architecture. In: Kent S, Evans A (eds) UML'2000 - the unified modeling language: advancing the standard, third international conference, York, UK, October 2–6, 2000, LNCS, vol 1939, pp 513–527

Kazman R, Carriere J (1998) View extraction and view fusion in architectural understanding. In: Proceedings of the fifth international conference on software reuse, pp 290–299

Kazman R, Carriere SJ (1999) Playing detective: reconstructing software architecture from available evidence. Autom Softw Eng 6(2):107–138

Kazman R, Woods S, Carriere J (1998) Requirements for integrating software architecture and reengineering models: corum ii. In: Proceedings of the 5th IEEE working conference on reverse engineering (WCRE), pp 154–163

Khandkar SH (2011) Open coding: introduction. http://pages.cpsc.ucalgary.ca/~saul/wiki/uploads/CPSC681/open-coding.pdf

Kruchten P (1995) Architectural blueprints – the "4 + 1" view model of software architecture. IEEE Softw 12(6)

Kruchten P (1995) The 4 + 1 view model of architecture. IEEE Softw 12(6):45–50

Lago P, Avgeriou P, Hilliard R (2010) Guest editors' introduction: software architecture: framing stakeholders' concerns. IEEE Softw 27(6):20–24. http://dblp.uni-trier.de/db/journals/software/software27.html#LagoAH10

Lavrakas PJ (ed.) (2008) Encyclopedia of survey research methods. SAGE Publications Inc. https://doi.org/10.4135/9781412963947

Lutellier T, Chollak D, Joshua Garcia LT, Rayside D, Medvidovic N, Kroeger R (2015) Comparing software architecture recovery techniques using accurate dependencies. In: Proceedings of the 2015 international conference on software engineering, ICSE '15. IEEE Press, Piscataway, NJ, USA

Malavolta I, Muccini H, Pelliccione P, Tamburri DA (2010) Providing architectural languages and tools interoperability through model transformation technologies. IEEE Trans Software Eng 36(1):119–140. http://dblp.uni-trier.de/db/journals/tse/tse36.html#MalavoltaMPT10

Medvidovic N, Jakobac V (2006) Using software evolution to focus architectural recovery. Autom Softw Eng 13(32):225–256

Murphy GC, Notkin D, Sullivan KJ (2001) Software reflexion models: bridging the gap between design and implementation. IEEE TSE 27(4):364–380

Myers DM (2009) Qualitative research in business & management, 1st edn. Sage, Los Angeles. http://www.gbv.de/dms/zbw/574672206.pdf

Naur P (1985) Programming as theory building. Microprocessing and Microprogramming 15(5):253–261

Newcomb P (2005) Architecture-driven modernization (adm). In: WCRE. IEEE Computer Society, p 237. http://dblp.uni-trier.de/db/conf/wcre/wcre2005.html#Newcomb05

Onions PEW (2006) Grounded theory applications in reviewing knowledge management literature. In: Leeds Metropolitan University innovation north research conference (1962), pp 1–20. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.2036&rep=rep1&type=pdf

Pollet D, Ducasse S, Poyet L, Alloui I, Cîmpan S, Verjus H (2007) Towards a process-oriented software architecture reconstruction taxonomy. In: Krikhaar RL, Verhoef C, Lucca GAD (eds) CSMR. IEEE Computer Society, pp 137–148. http://dblp.uni-trier.de/db/conf/csmr/csmr2007.html#PolletDPACV07

Resnik P (1999) Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. J Artif Intell Res 11:95–130

Sarkarati M, Gomez E, Nanni A, Tamburri DA, Bianco A (2008) Round trip engineering for legacy space data systems based on a model driven architecture approach. In: Proceedings of SpaceOps 2008 conference, Heidelberg, Germany, May 12–16, 2008, pp 310–320

Sartipi K, Dezhkam N, Safyallah H (2006) An orchestrated multi-view software architecture reconstruction environment. IEEE Computer Society, Washington, DC, USA

Schmerl B, Garlan D, Kazman R, Yan H (2006) Discovering architectures from running systems. IEEE Trans Softw Eng 32(7):454–466

Schreiber C, Carley KM (2004) Going beyond the data: empirical validation leading to grounded theory. Computational & Mathematical Organization Theory 10(2):155–164

Silverman D (1997) Validity and credibility in qualitative research. In: Miller G, Dingwall R (eds) Context and method in qualitative research. Sage, Thousand Oaks, CA, pp 13–25

Suddaby R (2006) From the editors: what grounded theory is not. Acad Manag J 49(4):633–642

Tamburri DA, Lago P, van Vliet H (2013a) Uncovering latent social communities in software development. IEEE Softw 30(1):29–36. https://doi.org/10.1109/MS.2012.170

Tamburri DA, Lago P, van Vliet H (2013b) Organizational social structures for software engineering. ACM Comput Surv 46(1):3:1–3:35. https://doi.org/10.1145/2522968.2522971

Tamburri DA, Kazman R, Fahimi H (2016) The architect's role in community shepherding. IEEE Softw 33(6):70–79. http://dblp.uni-trier.de/db/journals/software/software33.html#TamburriKF16

Teddlie C, Tashakkori A (2009) Foundations of mixed methods research: integrating quantitative and qualitative approaches in the social and behavioral sciences. Sage, Los Angeles. http://www.amazon.com/Foundations-Mixed-Methods-Research-Quantitative/dp/0761930124

Terra R, de Oliveira Valente MT (2009) A dependency constraint language to manage object-oriented software architectures. Softw Pract Exper 39(12):1073–1094. http://dblp.uni-trier.de/db/journals/spe/spe39.html#TerraV09

Thomson SB (2011) Sample size and grounded theory. JOAAG, 184–192

Tsai J, Xu K (2000) A comparative study of formal verification techniques for software architecture specifications. Ann Softw Eng 10(1):207–223. https://doi.org/10.1023/A:1018960305057

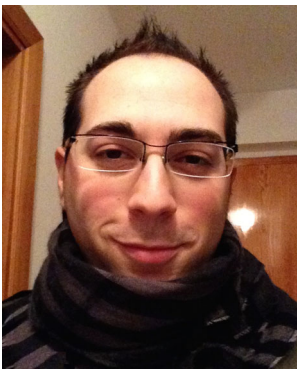
van Niekerk JC, Roode JD (2009) Glaserian and straussian grounded theory: similar or completely different?. In: Dwolatzky B, Cohen J, Hazelhurst S (eds) SAICSIT conference, ACM international conference proceeding series, ACM, pp 96–103

Vasconcelos A, Werner C (2007) Architecture recovery and evaluation aiming at program understanding and reuse. LNCS Springer

Wettinger J, Breitenbücher U, Kopp O, Leymann F (2016) Streamlining devops automation for cloud applications using tosca as standardized metamodel. Futur Gener Comput Syst 56:317–332. http://dblp.uni-trier.de/db/journals/fgcs/fgcs56.html#WettingerBKL16

Wille R (2005) Formal concept analysis as mathematical theory of concepts and concept hierarchies. In: Formal concept analysis, pp 1–33

Xiao L, Cai Y, Kazman R (2014) Titan: a toolset that connects software architecture with quality analysis. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, FSE 2014. ACM, New York, NY, USA, pp 763–766. https://doi.org/10.1145/2635868.2661677



Damian A. Tamburri is a Research Fellow at Politecnico di Milano, Italy. Though still in his very early career, he has published over 40+ papers in either Journals such as the Transactions on Software Engineering (TSE) Journal, The ACM Computing Surveys (CSUR) Journal, the IEEE Software Magazine or top software engineering conferences (such as ICSE or FSE) and top software architecture conferences (such as ECSA or WICSA). In addition, as part of his quick career, he is now an IEEE Software editorial board member and secretary of the IFIP TC2, TC6, and TC8 Working-Group on "Service-Oriented Computing". His current research interests lie mainly in social software engineering (Socio-technical congruence, Measuring Social Debt, etc.), advanced software architecture styles (e.g., SOA, Big-Data, etc.) and advanced software architecting methods (e.g., MDA, continuous architecting and DevOps). Contact him at damianandrew.tamburri@polimi.it or dtamburri@acm.org.

Rick Kazman is a Professor at the University of Hawaii and a Principal Researcher at the Software Engineering Institute of Carnegie Mellon University. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. Kazman has created several highly influential methods and tools for architecture analysis, including the SAAM (Software Architecture Analysis Method), the ATAM (Architecture Tradeoff Analysis Method), the CBAM (Cost-Benefit Analysis Method) as well as the Dali and Titan tools. He is the author of over 200 publications, and co-author of several books, including Software Architecture in Practice, Designing Software Architectures: A Practical Approach, Evaluating Software Architectures: Methods and Case Studies, and Ultra-Large-Scale Systems: The Software Challenge of the Future. Contact him at kazman@hawaii.edu.