From Monolith to Microservices: A not yet defined Approach

Bachelor's Thesis of

Niko Benkler

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. B
Advisor:	Dr. Robert Heinrich

xx. Month 20XX – xx. Month 20XX

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Niko Benkler)

Bla Sbtrakt

Abstract

English abstract.

Zusammenfassung

Deutsche Zusammenfassung

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

1.2. Problem Statement

1.3. Challenges

2. CoCoME

2.1. Introduction to CoCoME

[7]

3. State of the Art

3.1. Literature Review

Link	Titel	Author (Year)	Origin	Search String
[6]	Extraction of Microservices from Monolithic Software Architectures	G. Matzlami et. al. (2017)	Google Scholar	*microservice identification*
[1]	Object-Aware Identification of Microservice	M. J. Amiri (2018)	IEEE	*identification microservices*
[2]	Microservices Identification Through Interface Analysis	L. Baresi et. al. (2017)	google scholar	*microservice identification*
[9]	Identifying Microservices Using Functional Decomposition	S. Tyszberowicz et. al. (2018)	*provided*	*n/a*
[8]	Partitioning Microservices: A Domain Engineering Approach	I. J. Munezero et. al. (2018)	IEEE	*identify microservices*
[3]	From Monolith to Microservices: A Dataflow-Driven Approach	R.Chen et. al	IEEE	monolith to microservice
[4]	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems	A. De Alwis et. al. (2018)	Google Scholar	identify microservices
[5]	Service Cutter: A Systematic Approach to Service Decomposition	M. Gysel et. al. (2016)	[2]	*n/a*

3.2. Comparison and applicability of the approaches

* bedeuted INFO
+ beudeted PRO
- bedeuted CONTRA

3.2.1. Extraction of Microservices from Monolithic Software Architectures

* informal migration patterns exists. Lack of Formal Models
* small and recent body of work on how to migrate monolith to MS
* clas based extraction model, construct graph, process by clustering algorithm
* references Service Cutter (Pros and Cons)
* 2 phases: Construction (monolith to graph), clustering (decompose graph to cluster)
* starts with code base/repo from VCS
* each class is a node, edges have weights according to coupling strategy (classes that are not coupled are discarded)
* Logical Coupling Strategy(LC): Single Responsibility principle (Software has only one reason to change), enforce strong module boundaries (concept of MS) –> developers only make changes to the module (found in Change History, Class Files changed together belong togehter) ==> Weight is : for each pair of class look how often they changed together
* Semantic Coupling Strategy(SC): each MS correspond to one bounded context (DDD) from domain, examine contents/semantics of source code, term-frequency invese-documents-frequency method (tf-idf), compute relation of two classes regarding domain concepts
* tf-idf: Compute scalar vector for each class and compute cosine similarity between pairwise distinct classes
* tf-idf: Tokenize class, set of words, filter stop words, compare two classes regarding their common words with tf-idf formula
* Main Concern: Well organized teams, cross-functional but also reduce communication overhead to external teams while maximize internal
* Contributor Coupling (CC): team/orga info used to recover relationship among sw artifacts ==> Ownership architecture read from VSC history by identifying how many developers worked on the same pair of classes (weight!)

* CLustering Algo: Invert weights to favor edges with heigh weight, Kruskal for MST (calculate remaining edges), sort edges, reverse list, delete first element (originally lowest weight; now first in the list), in each iteration step: DFS(Depth-First) on edgesMST returns number of partitions, iterate and delete while n<nGiven *Reduce Cluster: after n cluster were formed, method reduceClusters splits up unusually large clusters (due to language and framework, special classes may have extraordinary high coupling) –> Delete those classes until size of node (numbe rof classes seems to be appropriate) ——————————

+ algorithmic recommendation of ms candidates implemented in web-based prototype
+ unites traditional decomposition techniques and microservice extraction approaches/design principles
+ algorithm uses 3 different coupling strategies: Can be combined for better results
+ performance of algortihm was satisfying according to author + shows significant team size reduction (less than half), given that alls contributirs for given cluster work on a MS
+ no work in advance!!!!

——————————————

- rely on (meta-)data extracted from codebase
- needs VCS (proper change history)
- needs ORM model that models data entities as ordinary classes - 2 (independent) changes in one commit destroy SRP
- SW must have gone through evolution process for LC
- Naming of class, methods, attributes needs to reflect domain language to make tf-idf possible
- Contributor Coupling: requires more developers
- Reduce Clusters: Useful to delete those classes? wat if this is core of application that belongs together - nPart is given by user: What is the right granularity?

————————————

*Conclusion: Beta implementation existing, paper is precise in usage, master-thesis exists (LINK!!!)with detailed information, easy algorithms, good performance,
* No Information about right n, not domain oriented, ORM necessary, BUT: Not applicable to CoCoME as VCS history, software evolution, several contributor necessary

3.2.2. Partitioning Microservices:A Domain ENgineering Approach

* One BusinessCapability determines one MS, based on domain engineering * Business Capability: Something hat a business does;combined as functionality that have something in common (instead of usin collections of data entities and CRUD operations in them) * Paper states: Developers struggle to define granulaity of business capability (too small => Communication overhead, to large: heavyweight SOA) * Appropriate MS size determined by component boundaries * DDD: Choosing appropriate boundary. MAIN GOAL: systematically group requirements in domain model and implement code of that * Design Domain model with sub-domains (each sub-domain has one to more bounded contexts) (* Sub-Domain is problem space, bounded.context is solution space and maps software artifacts to sub-domains ==> Info aus Internet) * DDD Patterns: Context Map/Counded Contexts (makes explicit boundaries between domains,), Aggregates (logical boundaries for cluster of domain objects that change during one transaction, ensure consistency among parallel operations that are considered to be one unit in reagrd to change), Ubiquitous Language (ensures that impementation uses the same terms as business), Separation of Entities and Value types (Entities are Objects of DDD, Not all Objects need to be entitites, for value objects identity is not necessary)) * MS should be autonomous (changes not affecting others): One bounded context is one MS (ore more if fine grained needed, BUT never on C in different MS) * Approach: Defined domain and ubiquitous language ar prerequisites, split into sb domains if domain is too wide, find boundary of each responsibility, make it as business capability (focus on relationship among different services), each capability is a MS, find out relationships between services with domain model and reconfigure if necessary (too many dependencies)

————————————————

+uses DDD. Many papers and experience reports state that this is the way to gto

————————————————

- draw the boundaries is the ky task and requires domain experts - requires well defined domain, pre-existing ubiquitous language - "Find the boundary of each responsibility and make it as a business capability" ==> But how? Requires DOmain experts - Procedure is not concise at all and only explains approach on a top level (half a page) - Does not define how procedure analyses "loose coupling possible" - how break down in subdomains? - uses brainstorming and interaction wit domain experts to identify and define parts of the microservice

————————————————

* Summary:
*Hot topic according to other papers (interviews, reports) but this paper lacks on accuracy.
* Simply said: Get some domain-experts and let them cut your domain in different bounded contexts
* This is what was already done when migrating CoCoME (probably not following the patterns but having domain experts)

3.2.3. From Monolith to Microservices: A Dataflow-Driven Approah

*top-down dataflow driven decomposition algorithm * 3 step approach
* purified DFD (focus on data's semanteme and operations only, excludes side information), more data focused than traditional
* purified DFD (PFD) is a directed Graph with nodes (processing operations nodes and data nodes) and Approach:
* construct traditional DFD (according to business logic extracted from users' natural-language descriptions)
* Manual construction of purified dataflow by two rules (more like guidelines)
*algorithmic construction (condensing of PDS) of the decomposable Dataflow by applying a set of rules * combines same operations with same output data * each operation at the end is a MS

————————————————————

+systematic Algorithm + purified DFD represents real information flow of corresponding business logic + reduce human mistakes when drawing a composable DFD + the 5 rules are easy to understand

————————————————————

- Semi Automated
- traditional DFD constructed on users descriptions and code (detailed data flow) ==> exact? Expertise necessary
- or: DFD needs to be existant
- transforming traditional to purified is a non trivial task
- identifying same data operation requires expertise: Combination of sam operations based on operation names (what if names are different through the project) - no implementation

- not applied to larger projects - based on user's natural language: semantic verbs correspond activities (later a MS)
-really really fine-grained MS, based on single operation (most fine graines as possible)
- no domain information included
- what about get/create.... hier weiß ich nicht, wie ich beschreiben soll dass es nicht geht!
- zweites beispiel zigt das deutlicher mit der operation QUERY

————————————————————

Conclusion

4. Solution Overview

5. Evaluation Planning

5.1. Applicability to CoCoME

5.2. Comparison to Functional Decomposition Approach

6. Timetable

6.1. Milestones

Bibliography

[1] M. J. Amiri. “Object-Aware Identification of Microservices”. In: (July 2018), pp. 253–256. ISSN: 2474-2473. DOI: 10.1109/SCC.2018.00042.

[2] Luciano Baresi, Martin Garriga, and Alan De Renzis. “Microservices Identification Through Interface Analysis”. In: (2017). Ed. by Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen, pp. 19–33.

[3] R. Chen, S. Li, and Z. Li. “From Monolith to Microservices: A Dataflow-Driven Approach”. In: (Dec. 2017), pp. 466–475. DOI: 10.1109/APSEC.2017.53.

[4] Adambarage Anuruddha Chathuranga De Alwis et al. “Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems”. In: (2018). Ed. by Claus Pahl et al., pp. 37–53.

[5] Michael Gysel et al. “Service Cutter: A Systematic Approach to Service Decomposition”. In: (2016). Ed. by Marco Aiello et al., pp. 185–200.

[6] G. Mazlami, J. Cito, and P. Leitner. “Extraction of Microservices from Monolithic Software Architectures”. In: (June 2017), pp. 524–531.

[7] Frank Mittelbach. “How to influence the position of float environments like figure and table in LaTeX?” In: *TUGboat* 35 (2014), pp. 248–254. URL: https://www.latex-project.org/publications/tb111mitt-float.pdf.

[8] I. J. Munezero et al. “Partitioning Microservices: A Domain Engineering Approach”. In: (May 2018), pp. 43–49.

[9] Shmuel Tyszberowicz et al. “Identifying Microservices Using Functional Decomposition”. In: (2018). Ed. by Xinyu Feng, Markus Müller-Olm, and Zijiang Yang, pp. 50–65.

A. Appendix

A.1. First Appendix Section

Figure A.1.: A figure

...