Microservices Backlog - A Model of Granularity Specification and Microservice Identification

Fredy H. Vera-Rivera[1,2,4](✉), Eduard G. Puerto-Cuadros[1], Hernán Astudillo[3], and Carlos Mauricio Gaona-Cuevas[4]

[1] Universidad Francisco de Paula Santander, San José de Cúcuta, Colombia
fredyhumbertovera@ufps.edu.co

[2] Foundation of Researchers in Science and Technology of Materials, Bucaramanga, Colombia

[3] Universidad Técnica Federico Santa María, Valparaíso, Chile

[4] Universidad del Valle, Cali, Colombia

Abstract. Microservices are a software development approach where applications are composed of small independent services that communicate through well-defined APIs. A major challenge of designing these applications is determining the appropriate microservices granularity, which is currently done by architects using their judgment. This article describes Microservice Backlog (MB), a fully automatic genetic-programming technique that uses the product backlog's user stories to (1) propose a set of microservices for optimal granularity and (2) allow architects to visualize at design time their design metrics. Also, a new Granularity Metric (GM) was defined that combines existing metrics of coupling, cohesion, and associated user stories. The MB-proposed decomposition for a well-known state-of-the-art case study was compared with three existing methods (two automatics and one semi-automatic); it had consistently better GM scoring and fewer average calls among microservices, and it allowed to identify critical points. The wider availability of techniques like MB will allow architects to automate microservices identification, optimize their granularity, visually assess their design metrics, and identify at design time the system critical points.

Keywords: Microservices architecture · Granularity · Decomposition · Cohesion metrics · Coupling metrics · Complexity metrics · User stories

1 Introduction

The microservices architectural changes the way applications are created, tested, implemented, and maintained. By using microservices, a large application can be implemented as a set of small applications that can be developed, deployed, expanded, managed, and monitored independently. Agility, cost reduction and granular scalability entail some challenges such as the complexity of managing distributed systems [1]. The appropriate size (granularity) of the microservice is one of their most discussed properties and there are few patterns, methods, or models to determine how small a

© Springer Nature Switzerland AG 2020
Q. Wang et al. (Eds.): SCC 2020, LNCS 12409, pp. 85–102, 2020.
https://doi.org/10.1007/978-3-030-59592-0_6

microservice should be. Thus Soldani et al. [2] argue that there is difficulty identifying the business capacities and delimited contexts that can be assigned to each microservice. Bogner et al. [3] note that methodologies and techniques must facilitate dimensioning and versioning of microservices. Zimmerman [4] wonders how to find an adequate service cut, (i.e. “how small or fine is small enough”). Jamshidi et al. [5] notice the lack of agreement on the correct size of microservices.

We introduce the Microservice Backlog, which allows to analyze graphically microservices granularity, starting from a set of functional requirements expressed as user stories within a product backlog (prioritized and characterized list of functionalities that an application must contain [6]); we propose a model that helps to define the size and number of microservices using genetic programming; it shows the microservices that are going to be part of an application detailing its dependencies, functionalities and coupling, cohesion, and complexity metrics at design time. Therefore, we can observe and evaluate the microservices’ granularity and analyze how the application will be implemented and structured.

The major contributions from this work are: 1) a model for determining and evaluating the granularity of microservices, establishing the number of user stories assigned to a microservice and the number of microservices that are part of the application, ensuring that microservices have low coupling and high cohesion, 2) identified and adapted metrics of complexity, coupling, cohesion, and size of the microservice, 3) mathematical formalization of an application based on microservices in terms of user stories and metrics, and 4) A genetic algorithm to propose a decomposition of user stories into microservices.

The remainder of this paper is organized as follows, Sect. 2 related works; Sect. 3 Methodology and evaluation methods used; Sect. 4 our approach; Sect. 5 discussing results; and Sect. 6: Summarizes our conclusions.

2 Related Works

Methods and techniques have been proposed to define the granularity of microservices, for example:

Service Cutter a method and tool framework for service decomposition [7]. In Service Cutter approach, coupling information is extracted from software engineering artifacts. Its approach is more for SOA applications. Hassan and Bahsoon [8] propose microservice ambients, which use “aspects” to define the adaptation behavior needed to support changes in granularity at runtime. Hasselbring and Steinacker [9], to achieve adequate granularity, propose a vertical decomposition in self-contained systems throughout the business services. Gouigoux and Tamzalit [10] explain that choice of granularity should be based on the balance between the costs of quality assurance and the cost of deployment. Baresi et al. [11] propose Microservices Identification Through Interface Analysis (MITIA), they address the problem of granularity by proposing an automated process to identify candidate microservices through a light semantic analysis, independent of the domain of the concepts in the input specification concerning a reference vocabulary. They perform an analysis of the semantic similarity of the functionalities described in OpenApi. Tyszberowicz et al. [12] describe a systematic

approach to identify microservices in the initial design phase that is based on specification the functional requirements of the system and that uses functional decomposition. Abdullah et al. [13] design a method to automatically decompose a monolithic application into microservices to improve the scalability and performance. They use the application's access logs and an unsupervised machine learning method, scale weighted k-means. De Alwis et al. [14] mathematically define a business system and a microservices-based system, then define heuristics to identify microservices. They propose a microservice discovery algorithm. Mazlami et al. [15] present a Graph-based clustering algorithm and a Class-based extraction model for the extraction of microservices from monolithic software architecture based on source code. Chen et al. [16] propose a top-down decomposition approach driven by data flows of business logic. Taibi and Syst [17] propose a process-mining approach to identify business processes in an existing monolithic solution based on three steps. In the first step, a process-mining tool is used to identify business processes. In the second step, processes with common execution paths are clustered and a set of microservices. In the third step, they propose a set of metrics to evaluate the decomposition quality. Jin et al. [18] propose Functionality-oriented Service Candidate Identification (FoSCI) framework to identify service candidates from a monolithic system, through extracting and processing execution traces. Microservices API patterns (MAP) [19] define some design and implementation patterns.

The above methods are mainly used in migrations from monoliths to microservices. The use of artificial intelligence is a subject of great interest, being the clustering algorithms the most used. Few methods support development from scratch (greenfield development). Different input data have been used, such as use cases, OpenApi specification, source code, dataflow diagram, database, execution call graphs, execution logs, and execution traces; mainly these methods are used at design and development time. The proposed method is used at design time, it uses user stories as input data and focuses on agile software development, none of the identified methods focus on these aspects. We characterized the process of applications based on microservices in [20] and we used that development process in [21].

3 Methodology

Based on one approach proposed by Hevner et al. [22]. The artifact to be created is the intelligent model of specifying the granularity of microservices that are part of an application. DSR implies a continuous and iterative assessment of the proposed artifact. Figure 1 shows the research model.

1. Identify the problem. To identify the problem and its relevance, a review of the state of the art was developed, research gaps were identified and the research questions for this work were formulated.

2. Identify and adapt the metrics. A systematic literature review was done to identify metrics that can be used to define the granularity. Section 3 shows these metrics.

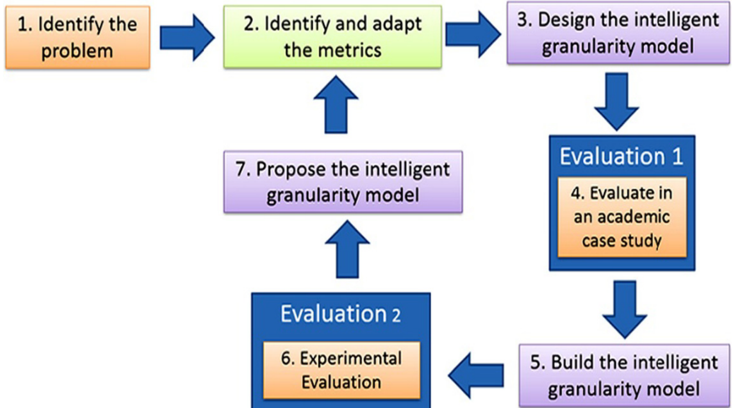

Fig. 1. Research model

3. Design the intelligent granularity model. A proposal of a formal specification of the granularity model can be found in Sect. 4.1. While a definition of a genetic algorithm for microservices decomposition is explained in Sect. 4.2

4. Evaluate in an academic case study. A state-of-the-art example called Cargo Tracking [11] was used to verify properly functioning and objectives compliance from our model. A comparison between results from the case of study and decomposition performed with DDD are shown in Fig. 4.

5. Build the intelligent granularity model. A genetic algorithm was implemented to generate the decomposition of the product backlog into microservices. An algorithm was implemented to evaluate metrics for decomposition. Sections 4 and 5 detail this implementation.

6. Experimental evaluation. Metrics of Cargo Tracking case study are analyzed by four methods: Domain-driven design (DDD), Service Cutter, Microservices Identification Through Interface Analysis (MITIA) [11] and our approach Microservices Backlog. Since DDD is the most widely used method for microservices identification, our first evaluation verified that obtained decomposition was consistent and close to that performed by DDD. A second evaluation compares decomposition made by our method versus other decompositions methods.

7. Propose the intelligent granularity model. Proposing intelligent granularity model. Based on metrics and analytical evaluation including adjustment through researching a Microservice Backlog is proposed as an intelligent specification and granularity evaluation model.

4 Our Approach

Agile practices are techniques used to control one aspect of the development process. One of the most widely used agile practices is Sprint/iteration planning [23], traditionally expressed in user stories within the product backlog. A model to define

microservices granularity from user stories and analyze some metrics is proposed. A view of the model can be seen in Fig. 2.

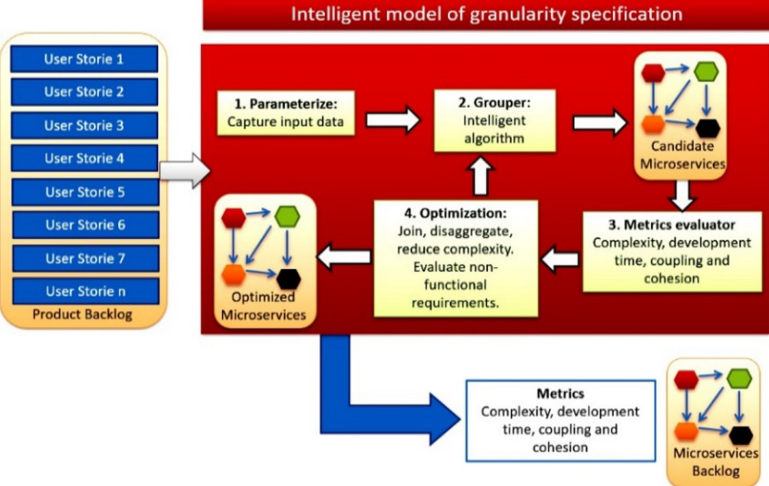

Fig. 2. Intelligent model of granularity specification

1. Parameterize. It is responsible for taking input data and converting it into a format that can be processed by the grouper. It extracts the key data, such as identifier, name, description, estimated points, estimated time, scenario, observations, and dependencies, from the user story. Later, with this data, the model can group the user stories in microservices and calculate the metrics. The format of the user stories is a JSON file or CVS where the key data are supplied.

2. Grouper. This component uses a genetic algorithm, which groups user histories into microservices, considering cohesion and coupling metrics, as well as the number of user stories associated with the microservice.

3. Metrics evaluator. This work considers the following metrics in the microservices backlog [3]: **1) Complexity – Points:** Estimated points of the effort needed to develop the user story. The story points are an indicator of the speed of development of the team. **2) Coupling – Absolute Importance of the Service (AIS):** The number of clients that invoke at least one operation of a microservice's interface [24]. **3) Coupling – Absolute Dependence of the Service (ADS):** The number of other microservices that microservice depends on. The number of microservices from which invokes at least one operation [24]. **4) Coupling – Microservices Interdependence (SIY):** Number of interdependent microservices pairs [24]. **5) Cohesion - Lack of cohesion (LC):** Measured as the number of pairs of microservices not having any dependency between them, adapted from [25]. LC of MS_i was defined by us as the number of pairs of microservices not having any interdependency between MS_i. **6) Weighted Service Interface Count (WSIC):** It is the number of exposed interface operations of the microservice [26]. For our model, a user story is related to an

operation (one-to-one); so, we adapt this metric as the number of user stories associated wiht the microservice. **7) Development Time:** Estimated time of development in hours for the microservice. Summation of the estimated time of each user story that is part of the microservice.

4. Optimization. This optimizer allows finding the most optimal solution that meets certain conditions (non-functional requirements, test costs, cost of deployment, etc.), performing operations of union and decomposition of the microservices candidates. This optimizer will be addressed in future work, due to the time and scope of the research.

5. Outputs of the model. The calculated metrics and the microservices backlog diagram. Figure 3 shows Microservices Backlog for the Cargo Tracking application.

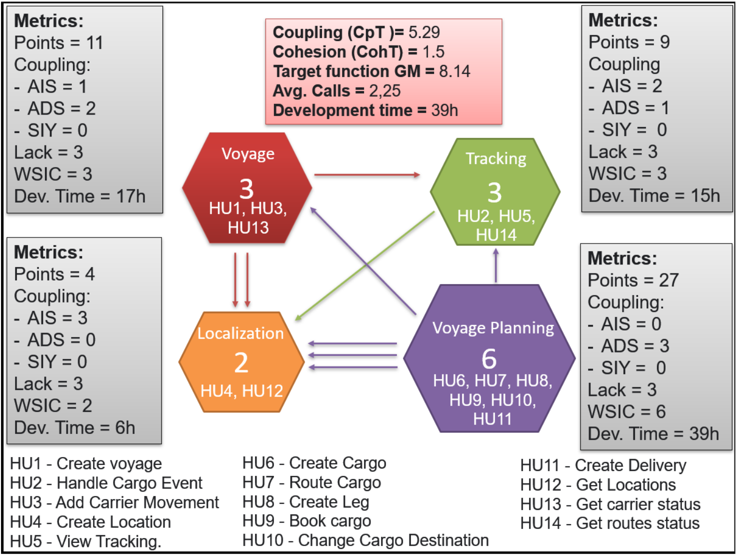

Fig. 3. Microservices backlog for Cargo Tracking using DDD decomposition

The microservices backlog in Fig. 3 was obtained by decomposition using DDD and the next steps: 1) The user stories were loaded. 2) The dependencies between the stories were defined. 3) The entities were identified. 4) The aggregates were defined, 5) The delimited contexts were established, the entities and their respective user histories were associated. 6) The metrics were calculated by the evaluator. Specific metrics for each microservice and the whole application. It can be highlighted that the grouper component of our model automatically identifies the candidate microservices, then the steps 3 to 5 are automatic.

From the model, the designer can see the size of each microservice, as well as its complexity, dependencies, coupling, cohesion, and development time. The architect can notice at first sight that the orange microservice is a critical point of the system if this microservice failure, then the whole system can fail because it is used by all the

others. The architect at design time can already think about fault tolerance mechanisms, load balancing and monitoring on that critical microservice. They can have a vision of the global system in design time.

4.1 Formal Specification of the Granularity Model

Specification formal of the granularity model will be given in terms of the metrics stated in the previous section and by the target function (GM). It is intended to MINIMIZE (GM). GM is defined below. Let microservice-based application MSBA as:

$$\text{MSBA} = (\text{MS}, \text{MT}) \tag{1}$$

Where MS is a set of microservices, MS = $\{MS_1, MS_2, \ldots MS_n\}$ and MT is a set of the metrics calculated for MSBA. Then:

$$\text{MS}_\text{i} = (\text{HU}, \text{MTS}) \tag{2}$$

Where MS_i is the ith microservice, HU is the set of user stories associated with the ith microservice, then HU = $\{HU_1, HU_2, \ldots, HU_m\}$. MTS is a set of metrics calculated for MS_i. In this case, the calculated and used metrics in the model correspond to the coupling (CpT), the cohesion (CohT) and the number of stories associated with the microservice (WsicT). These metrics are defined below.

Coupling Metrics. Coupling is defined by three metrics: 1) absolute importance of the service (AIS), 2) absolute dependence of the service (ADS), and 3) microservices interdependence (SIY). These metrics are calculated based on the dependencies of the user stories for each microservice.

AIS_i is the number of clients invoking at least one operation of MS_i. At the system level, the AIS vector is defined, which contains the calculated AIS value for each microservice. To calculate the total value of AIS at the system level (AisT), the AIS vector norm is calculated. Thus:

$$\mathbf{AIS} = [\text{AIS}_1, \text{AIS}_2, \ldots, \text{AIS}_\text{n}] \tag{3}$$

$$\text{AisT} = |\mathbf{AIS}| \tag{4}$$

ADS_i is the number of other microservices on which the MS_i depends. To calculate the total value of ADS at the system level (AdsT), the ADS vector norm is calculated. Then:

$$\mathbf{ADS} = [\text{ADS}_1, \text{ADS}_2, \ldots, \text{ADS}_\text{n}] \tag{5}$$

$$\text{AdsT} = |\mathbf{ADS}| \tag{6}$$

SIY defines the number of pairs of microservices that depend bi-directionally on each other divided by the total number of microservices. At the system level, the vector SIY was defined:

$$\mathbf{SIY} = [\mathrm{SIY}_1, \mathrm{SIY}_2, \ldots, \mathrm{SIY}_n] \tag{7}$$

$$\mathrm{SiyT} = |\mathbf{SIY}| \tag{8}$$

Let the Cp vector as the system level coupling metric, calculating the norm of the vector Cp we have the coupling value for the application (CpT):

$$\mathbf{Cp} = [\mathrm{AisT},\ \mathrm{AdsT},\ \mathrm{SiyT}] \tag{9}$$

$$\mathrm{CpT} = |\mathbf{Cp}| \tag{10}$$

Cohesion Metric. In the same way, the cohesion for the ith microservice is defined by the metric lack of cohesion (LC), The degree of cohesion of each microservice is defined as the proportion of the Lack of cohesion metric divided by the total number of microservices that are part of the application.

$$\mathrm{Coh_i} = \mathrm{LC_i}/\mathrm{n} \tag{11}$$

Where n is the number of microservices. At the system level, the vector **Coh** was defined, calculating the norm of the vector **Coh** we have the cohesion value for the application (CohT):

$$\mathbf{Coh} = [\mathrm{Coh}_1, \mathrm{Coh}_2, \ldots, \mathrm{Coh}_n] \tag{12}$$

$$\mathrm{CohT} = |\mathbf{Coh}| \tag{13}$$

Indeed, the MT vector is defined as follows:

$$\mathbf{MT} = [\mathrm{CpT},\ \mathrm{CohT},\ \mathrm{WsicT}] \tag{14}$$

Where, CpT use (10), CohT use (13) and WsicT is defined as the highest WSIC value. We adapt WSIC as the number of user stories assigned to each microservice. Finally, the value of the target function GM use (14), it is defined as the MT vector norm.

$$\mathrm{GM} = |\mathbf{MT}| \tag{15}$$

This mathematical expression allows us to determine how good or bad is the decomposition. The aim is to obtain a solution with low complexity, low coupling, and high cohesion. The genetic algorithm seeks to find the best combination, the best assignation of stories to microservices in such a way that GM is lower. The genetic algorithm is then designed as follows.

4.2 Genetic Algorithm for Microservices Decomposition

The genetic algorithms were established by Holland [27], it is iterative, in each iteration, the best individuals are selected, everyone has a chromosome, which is crossed with another individual to generate the new population (reproduction), some mutations are generates to find the optimal solution to the problem [28]. Our genetic algorithm consists in distributing or assigning user stories to microservices automatically, considering coupling and cohesion metrics. The implemented methods are explained below:

Get Initial Population Method. There is a set of user stories HU = $\{HU_1, HU_2, HU_3, \ldots, HU_m\}$, which must be assigned to the microservices. We have a set of microservices MS = $\{MS_1, MS_2, MS_3, \ldots, MS_n\}$ and some metrics calculated from the information contained in the user story. Individuals are defined from the assignment of stories to microservices, therefore, the chromosome of each individual is defined from an assignment matrix of ones and zeros, wherein the columns there are user stories and in the rows are the microservices, and the cross contains a 1 when the user story is assigned to the microservice or zero if not. In Table 1, an example is presented for 2 microservices MS = $\{MS_1, MS_2\}$ and 5 user stories HU = $\{HU_1, HU_2, HU_3, HU_4, HU_5\}$.

Table 1. Example of an assignment matrix

Microservices	HU1	HU2	HU3	HU4	HU5
MS_1	1	0	0	1	1
MS_2	0	1	1	0	0

The resulting chromosome would be the union of the assignments of each user story to each microservice, for this case, it would be: Chromosome: 10011 01100. From this chromosome, it is possible to define the function of adaptation or objective function, it uses (15).

Reproduction Method. A different assignment would be generated from selected parents. In our method, the father and mother are randomly selected from the population; to generate the child information is taken from the father and mother, from the assignment matrix the first columns of the father are taken, and the last columns of the mother are joined, generating a new assignment. It must be considered that a user story cannot be assigned twice, this means that in the assignment matrix only one can appear in each column. Example: Given the two chromosomes: 1) Father: 10011 01100. 2) Mother 01000 10111. The son would be 10000 01111.

Mutation Method. The mutation indicates changing a random bit of the chromosome, changing a bit of the chromosome of this problem from 1 to 0 or from 0 to 1, implies that a user story is assigned or unassigned to a microservice and this must be assigned or unassigned to another microservice. This implies that the mutation is done on two bits. Example: Mutate bit 7 of the obtained chromosome: 01011 10100. Mutated chromosome: 00011 11100. The mutated chromosomes must be included in the population. This process is carried out randomly, the individuals to be mutated are selected

from the population, the mutation of a bit is also carried out randomly, for the mutation the value of the target function is calculated and included in the population.

Select Better Method: In the processes of genetic selection, the strongest survive, in the case of the problem of the automatic generation of the assignment of user histories to microservices, the n individuals who best adapt to the conditions of the problem survive. The assignments that imply a lower GM. The selection is made from the objective function, this is applied to each individual and the population is ordered in ascending form, considering the first places, the best individuals, corresponding to the assignments involving lower GM using (15).

Convergence: To determine the convergence of the method, the number of iterations or generations of the population to be processed is defined. At the end of the iterations, the algorithm is stopped, and the chromosome located in the first place is selected, which would be the best assignment of user stories to microservices. For the case studies used to evaluate the proposed method, a population of 1000 individuals were generated, with 100 iterations or generations, with 500 children and 500 mutations in each generation. The algorithm was tested several times obtaining the same result, even with more individuals and more iterations.

5 Results

The genetic algorithm was implemented in Java, to evaluate its results we use a case study and a quasi-experiment.

5.1 Evaluation in an Academic Case Study – Cargo Tracking Application

Baresi et al. [11] the describe Cargo Tracking application as follow, the focus of the application is to move a Cargo (identified by a TrackingId) between two Locations through a RouteSpecification. Once a Cargo becomes available, it is associated with one of the Itineraries (lists of CarrierMovements), selected from existing Voyages. HandlingEvents then trace the progress of the Cargo on the Itinerary. The Delivery of a Cargo informs about its state, estimated arrival time, and is on track. From the domain model proposed, we extracted and raised user stories and the product backlog is detailed in Table 2. The points and times are input data to the model. I this case they were estimated according to our experience and correspond to the effort and time involved in developing each user story.

A critical point of the proposed method is the dependencies between user stories. They must be identified and provided as input to the method, this information is included within the user stories. The parameterizing component offers functionality to define dependencies between user stories. We define a dependence between HU_i and HU_j when HU_i calls or executes HU_j. For example, to create a voyage (HU_1) you must get the locations (HU_{12}), this implies that the HU_1 has a dependence on HU_{12}. Table 3 presents the dependencies identified by us among the user stories. The dependencies were calculated according to the logic of the application understood by us. To illustrate the proposed genetic algorithm the statement of these dependencies is valid.

Table 2. Product backlog for Cargo Tracking application

ID	Name	Points	Estimated dev. time (hours)
HU_1	Create voyage	3	5
HU_2	Handle cargo event	3	5
HU_3	Add carrier movement	5	7
HU_4	Create location	2	3
HU_5	View tracking	3	5
HU_6	Create cargo	7	10
HU_7	Route cargo	5	7
HU_8	Create leg	2	3
HU_9	Book cargo	5	7
HU_{10}	Change cargo destination	1	2
HU_{11}	Create delivery	7	10
HU_{12}	Get locations	2	3
HU_{13}	Get carrier status	3	5
HU_{14}	Get routes status	3	5
Total		51	77

Table 3. User stories dependences

User stories	Dependences	User stories	Dependences
HU_1	$\{HU_{12}, HU_3\}$	HU_8	$\{HU_{12}\}$
HU_2	$\{HU_{12}\}$	HU_9	$\{HU_{12}\}$
HU_3	$\{HU_{12}\}$	HU_{10}	$\{HU_{12}\}$
HU_4	$\{\}$	HU_{11}	$\{HU_6, HU_{13}, HU_{14}\}$
HU_5	$\{\}$	HU_{12}	$\{\}$
HU_6	$\{HU_7, HU_9, HU_{11}\}$	HU_{13}	$\{HU_5\}$
HU_7	$\{HU_8\}$	HU_{14}	$\{HU_5\}$

Dependencies are used to calculate the metrics, for example, to calculate the AIS metric of the decomposition obtained with DDD for the microservice called Localization (see Fig. 4). $MS_{1\ (Voyage)} = \{HU_1, HU_3, HU_{13}\}$, $MS_{2\ (Tracking)} = \{HU_2, HU_5, HU_{14}\}$, $MS_{3\ (Localization)} = \{HU_4, HU_{12}\}$, $MS_{4\ (Voyage\ Planning)} = \{HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}\}$. AIS is the number of clients that invoke at least one operation of a microservice's interface. Then we count the number of microservices that invoke or use HU_4 o HU_{12} from the dependencies. HU_4 is not used by any other user stories, it does not appear in any dependencies (See Table 3), while HU_{12} is used by HU_1, HU_2, HU_3. HU_8, HU_9, and HU_{10} corresponding to 3 microservices, therefore AIS = 3. Similarly, other metrics are calculated.

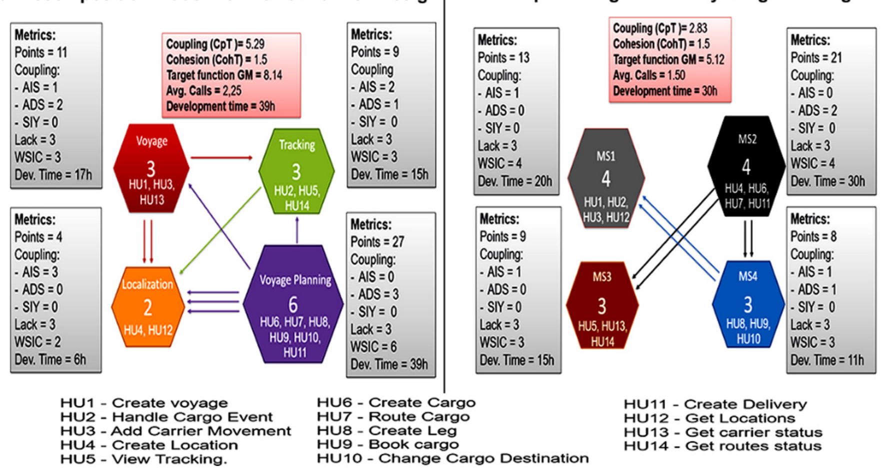

Fig. 4. Microservices backlog for the result of the DDD vs genetic algorithm.

Figure 4 presents the microservice backlog for the decompositions generated by the genetic algorithm compared with DDD for Cargo Tracking. Our method obtained the same number of microservices, this being an important approximation to DDD. Our method does not consider the semantic similarity between user stories. For example, HU_4 and HU_{12}, both are related to the Localization concept, as they are concepts related to the same, they must be associated with the same entity and therefore to the same microservice. In our method those stories were assigned in separate microservices, so they do not have any dependence between them (i.e. HU_4 has not dependence with HU_{12}). The decomposition performed by our method is different from DDD, our model does not group the entities and their stories or operation that make up the aggregate into a microservice.

With the decomposition obtained with the genetic algorithm, the critical point of failure of the proposed DDD solution is removed, Localization microservice is used for all microservices. The number of calls between microservices is reduced, thus improving performance. The maximum number of operations associated with a microservice is also reduced, as well as the estimated development time. In the decomposition generated by genetic programming, two microservices can function independently without depending on other microservices. Whereas in the solution proposed by DDD, one microservice can function independently. In the decomposition proposed by DDD, there are more dependencies. Therefore, the proposed model and the genetic algorithm considerably improves the decomposition and identification of microservices. To generalize this result, validation must be carried out in future work with more complex case studies specifically with real industry cases.

By distributing user stories differently, shorter development times of the entire system can be obtained. Considering that each microservice is developed by an independent team in parallel.

5.2 Quasi-Experimental Evaluation

To evaluate the results obtained by the model of specification of the granularity, we use the work done by Baresi et al. [11], they propose a decomposition to microservices of the Cargo Tracking case study using interface analysis and semantic similarity (MITIA), they also propose the decomposition of that same case using Service Cutter, we take those results and propose an experiment to compare our model with these methods, we include the decomposition performed by DDD. Also, we include a hypothetical case where only one user story was added per microservice, we call it 14MS, this corresponds to the case of the finest granularity, additionally, we include the monolithic solution. MITIA and Service Cutter propose the result of the decomposition in a domain model from there we determine the association of user stories and microservices. We use a quasi-experiment for evaluating our method against the other methods. The definition of the quasi-experiment is detailed below.

Scope: Compare the granularity specification model with the decomposition methods selected from the state of the art (DDD, Service Cutter, MITIA, 14MS and monolithic) for the Cargo Tracking case study. The GM granularity metric is evaluated in the decompositions obtained with each method to determine the accuracy of the proposed model. GM is calculated from coupling metrics, cohesion and number of operations assigned to each microservice.

Planning. 1) Objects of study: Microservices Backlog, DDD, Service Cutter, MITIA, 14 MS, and monolithic solution. 2) Independent variables: User stories dependences, decomposition obtained by each method. 3) Dependent variables: GM, Metrics: AisT, AdsT, SiyT, CpT, CohT, and number of microservices.

Hypothesis Formulation. $\mathbf{H_0}$: Our microservices backlog model does not present a better decomposition in microservices, therefore the value of GM is greater than GM of the other methods, then the application has not better coupling and cohesion. $\mathbf{H_1}$: Our microservices backlog model presents a better decomposition in microservices, therefore the value of GM is lower than GM of the other methods, then the application has better coupling and cohesion.

Operation. The quasi-experiment is carried out in the laboratory, the decomposition for the Cargo Tracking case study is determined for each one of the methods (see Table 4). Based on the dependencies of the user histories, the metrics are calculated and the value of GM for each decomposition (see Table 5). Another set of metrics were calculated for better analysis (see Table 6).

Analysis & Interpretation. The data collected correspond to the values calculated for the metrics and the GM function for each one of the methods compared. From the results obtained for each metric, the lowest and highest value is identified, to evaluate the hypotheses proposed. The data and hypothesis raised are simple and their validation does not require additional statistical analysis. Rejecting the H_o hypothesis indicates that the decomposition proposed by our model is better than the decomposition proposed by the other methods.

Experiment Results and Discussions. First, the decomposition obtained by each of the methods is detailed. Table 4 shows these results. Second, we tabulate the results obtained for each metric. these can be seen in Table 5. Finally, we identify the methods that obtained lower and higher values for each metric including GM.

MITIA considers the semantic similarity between the operations, for that reason a distribution closer to DDD can be appreciated. The Service Cutter has one less microservice, but the distribution is like DDD, although the number of operations exposed by MS_3 is greater. Based on the calculated metrics, it can be appreciated that our decomposition presents a smaller coupling compared to the other methods. In this case, the cohesion is given in terms of the number of microservices that are part of the application, having more microservices this value will be greater; for this reason, the highest cohesion is presented by the decomposition with 14MS. But the value of the cohesion of our method is equal to that obtained with DDD and greater than Service Cutter and MITIA.

Table 4. Comparison of the decompositions of the methods evaluated

ID	Number of microservices	Microservices decomposition
Our approach: microservices backlog	4	$MS_1 = \{HU_1, HU_2, HU_3, HU_{12}\}$ $MS_2 = \{HU_4, HU_6, HU_7, HU_{11}\}$ $MS_3 = \{HU_5, HU_{13}, HU_{14}\}$ $MS_4 = \{HU_8, HU_9, HU_{10}\}$
DDD	4	$MS_1 = \{HU_1, HU_3, HU_{13}\}$ $MS_2 = \{HU_2, HU_5, HU_{14}\}$ $MS_3 = \{HU_4, HU_{12}\}$ $MS_4 = \{HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}\}$
Service cutter	3	$MS_1 = \{HU_4, HU12\}$ $MS_2 = \{HU_2, HU5\}$ $MS_3 = \{HU_1, HU_3, HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}, HU_{13}, HU_{14}\}$
MITIA	4	$MS_1 = \{HU_3, HU_9, HU_{10}, HU_{13}\}$ $MS_2 = \{HU_1, HU_2, HU_5, HU_{11}, HU_{14}\}$ $MS_3 = \{HU_6\}$ $MS_4 = \{HU_4, HU_7, HU_8, HU_{12}\}$

As future work, other cohesion metrics will be considered and revised to be more precise in their calculation. Our method presents the lowest number of user stories or operations associated with a microservice (WsicT), with a value of 4 stories. The highest value is presented by Service Cutter with 10 stories associated with a single microservice, thus Service Cutter has a greater complexity of both implementation and operation.

Table 5. Metrics calculated for the decompositions of the methods evaluated

Metrics	Methods					
	14MS	DDD	Service cutter	MITIA	Our approach	Monolith
Number of MS	14	4	**3**	4	4	1
AisT	6.93	3.74	2.24	4.24	1,73	0
AdsT	5.48	3,74	2.24	4.69	2,24	0
SiyT	1.41	0	0	2,45	0	0
Coupling CpT	8.94	5.29	3.16	6.78	**2.83**	0
Cohesion CohT	3.44	**1.5**	1.15	1.06	**1.5**	0
WsicT	1	6	10	5	**4**	14
GM	9.63	8.14	10.55	8.49	**5.12**	14

Table 5 shows that the value of the GM obtained by our model is lower than all the other methods analyzed, additionally, the coupling (CpT) was the lowest, with the fewest number of stories associated with a microservice (WsicT), the cohesion (CohT) was the highest compared to DDD, Service Cutter, and MITIA. Therefore, we reject the H_0 hypothesis, which indicates that the decomposition proposed by our model is better than the decomposition proposed by the other methods, in terms of the metrics proposed in this work. The value obtained in the GM function for the monolithic application is the highest, in the same way, the GM value for 14MS is not the lowest, the appropriate solution is an intermediate point between the finest granularity (14MS) and the thickest granularity (Monolith), therefore, the mathematical formalization fits the expected.

Also, we calculate other metrics to evaluate the proposed methods: **1) Points:** Greater number of story points associated with a microservice. **2) Average of Calls:** that indicates the average of calls that a microservice makes to another microservice. **3) Development time:** Each user story has an associated estimated development time, therefore the estimated development time of the MS_i is the sum of the development time of each user story associated with the $MS_{i,}$ Table 6 shows these metrics.

Table 6. Other metrics for microservices backlog

Metrics	14MS	DDD	Service cutter	MITIA	Our approach	Monolith
Max. points	7	27	41	19	21	52
Avg. calls	1.14	2.25	2.67	3	1.50	0
Dev. time (hours)	10	39	61	30	30	77

The lowest number of story points without considering the metrics calculated for 14 MS corresponds to MITIA with 19 points. The shortest development time was the decomposition proposed by MITIA with 30 h. Our method obtains one close value of 21 points and 30 h of development respectively, being these values better than DDD and Service Cutter. The average number of calls of our approach is less than DDD, Service Cutter, and MITIA. This metric measure or determine the degree of dependence that have the microservices that are part of the application, a larger value implies

a greater dependence and lower performance because they require the execution of operations that belong to other microservices in other containers.

6 Conclusions

This paper proposes the Microservices Backlog a genetic-programming technique that calculates at design time each microservices' granularity. This model uses as inputs the user stories expressed in the product backlog, to decompose the functionalities or requirements of the application into microservices. To evaluate our proposal, the case study Cargo Tracking was used, the decomposition made with DDD, service Cutter and Microservices Identification Through Interface Analysis (MITIA) were compared. The decomposition performed by our model has less coupling, greater cohesion, fewer operations associated with a microservice, a better average of calls from one microservice to another and lower value in the proposed objective mathematical function (GM) used in the genetic algorithm. This algorithm allows us to model and evaluates the level of granularity of the microservices that are part of the application at design time.

To model and define the right granularity we identify and adapt metrics of complexity: estimated story points; metrics of coupling: absolute importance of the service (AIS), absolute dependence of the service (ADS), microservices interdependence (SIY); metrics of cohesion: lack of cohesion (LC) and degree of cohesion (CohT); and metrics of size of the microservice: weighted service interface count (WSIC). These metrics were used to determine the most suitable decomposition with less coupling, high cohesion, and fewer assigned user stories. Mathematical formalization of an application based on microservices in terms of user stories and metrics was proposed. Too coarse-grained microservices could lead to significant drawbacks, while too fine-grained services could increase the system's overall complexity and performance, our model found the right service granularity at design time, based on the mathematical function proposed GM for the genetic program.

References

1. Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Gil, S.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 10th Computing Colombian Conference, pp. 583–590 (2015)
2. Soldani, J., Tamburri, D.A., Van Den Heuvel, W.-J.: The pains and gains of microservices: a systematic grey literature review. J. Syst. Softw. **146**, 215–232 (2018)
3. Bogner, J., Wagner, S., Zimmermann, A.: Automatically measuring the maintainability of service- and microservice-based systems. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement on - IWSM Mensura 2017, pp. 107–115 (2017)
4. Zimmermann, O.: Microservices tenets: agile approach to service development and deployment. Comput. Sci. Res. Dev. **32**(3–4), 301–310 (2017)
5. Jamshidi, P., Pahl, C., Mendonca, N.C., Lewis, J., Tilkov, S.: Microservices: the journey so far and challenges ahead. IEEE Softw. **35**(3), 24–35 (2018)

6. Beck, K., Fowler, M.: Planning Extreme Programming. Addison Wesley, Boston (2001)
7. Gysel, M., Kölbener, L., Giersche, W., Zimmermann, O.: Service cutter: a systematic approach to service decomposition. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol. 9846, pp. 185–200. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_12
8. Hassan, S., Ali, N., Bahsoon, R.: Microservice ambients: an architectural meta-modelling approach for microservice granularity. In: Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017, pp. 1–10 (2017)
9. Hasselbring, W., Steinacker, G.: Microservice architectures for scalability, agility and reliability in e-commerce. In: Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, pp. 243–246 (2017)
10. Gouigoux, J.P., Tamzalit, D.: From monolith to microservices: lessons learned on an industrial migration to a web oriented architecture. In: Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, pp. 62–65 (2017)
11. Baresi, L., Garriga, M., De Renzis, A.: Microservices identification through interface analysis. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds.) ESOCC 2017. LNCS, vol. 10465, pp. 19–33. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67262-5_2
12. Tyszberowicz, S., Heinrich, R., Liu, B., Liu, Z.: Identifying microservices using functional decomposition. In: Feng, X., Müller-Olm, M., Yang, Z. (eds.) SETTA 2018. LNCS, vol. 10998, pp. 50–65. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99933-3_4
13. Abdullah, M., Iqbal, W., Erradi, A.: Unsupervised learning approach for web application auto-decomposition into microservices. J. Syst. Softw. **151**, 243–257 (2019)
14. De Alwis, A.A.C., Barros, A., Polyvyanyy, A., Fidge, C.: Function-splitting heuristics for discovery of microservices in enterprise systems. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 37–53. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_3
15. Mazlami, G., Cito, J., Leitner, P.: Extraction of microservices from monolithic software architectures. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 524–531 (2017)
16. Chen, R., Li, S., Li, Z.: From monolith to microservices: a dataflow-driven approach. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pp. 466–475 (2017)
17. Taibi, D., Systä, K.: From monolithic systems to microservices: a decomposition framework based on process mining. In: International Conference on Cloud Computing and Services Science - CLOSER 2019, no. March (2019)
18. Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., Zheng, Q.: Service candidate identification from monolithic systems based on execution traces. IEEE Trans. Softw. Eng. (2019)
19. Zimmermann, O., Stocker, M., Zdun, U., Lübke, D., Pautasso, C.: Microservice API Patterns (2019). https://www.microservice-api-patterns.org/introduction. Accessed 17 Dec 2019
20. Vera-Rivera, F.H.: A development process of enterprise applications with microservices. J. Phys: Conf. Ser. **1126**(17), 012017 (2018)
21. Vera-Rivera, F.H., Vera-Rivera, J.L., Gaona-Cuevas, C.M.: Sinplafut: a microservices – based application for soccer training. J. Phys: Conf. Ser. **1388**(2), 012026 (2019)
22. Bichler, M.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2006)
23. Versionone Enterprise, "13 Anual State of Agile Report" (2018). http://stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508

24. Rud, D., Schmietendorf, A., Dumke, R.R.: Product metrics for service-oriented infrastructures. In: Conference: Applied Software Measurement. Proceedings of the International Workshop on Software Metrics and DASMA Software Metrik Kongress (IWSM/MetriKon 2006) (2006)
25. Candela, I., Bavota, G., Russo, B., Oliveto, R.: Using cohesion and coupling for software remodularization: is it enough? ACM Trans. Softw. Eng. Methodol. **25**(3), 1–28 (2016)
26. Hirzalla, M., Cleland-Huang, J., Arsanjani, A.: A metrics suite for evaluating flexibility and complexity in service oriented architectures. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 41–52. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01247-1_5
27. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Michigan (1975)
28. Herrera, F., Lozano, M., Verdegay, J.L.: Algoritmos Genéticos: Fundamentos, Extensiones y Aplicaciones. ProQuest (1995)