CAPCO

DECOMPOSING THE MONOLITH:

OPTIMIZATION AND AUTOMATION

ABSTRACT

For the first time, we present a data affinity-driven method for decomposing monolith applications into a collection of microservices. The decomposition strategy defined by our process, comprises mapping data objects exposed by the monolith's API endpoints to an enterprise business capability framework and then clustering the business capabilities through data object cohesion. Through this, we define an optimized set of service components that embody business capabilities, but that simultaneously minimizes network latency upon implementation. Using the output from the clustering method, we've developed a strategy that determines where to initiate the decomposition process and how to move from current to target state progressively.

1. INTRODUCTION

A microservices-based architecture enables an enterprise to, through employing multiple teams that work in parallel, deliver business value independently of each other and at greater speeds while collectively avoiding the high cost of ownership that is associated with a monolith architecture[1]. With a well-defined microservices strategy and implementation, you can remove delivery bottlenecks, and develop scalable and resilient software in a manner that is highly responsive to changing business demand, ultimately allowing an organization to embrace BizDevOps practices.

Strategically planning the monolith decomposition roadmap plays a critical role in the success or failure of an enterprise's microservices strategy since:

- A monolith decomposition journey typically co-occurs with the development of new business features as well as the implementation of code patches that address production incidents on the same IT stack. Thus, a key consideration is balancing short and long-term stakeholder expectations regarding the urgency of the transformation within the organization.

- Poor choices surrounding the compositional granularity of the new service layer are hard and costly to reverse. Additionally, it may harm overall system performance and stability.

- For a microservices strategy to genuinely successful, a cultural transformation within an organization that spans well beyond the IT department may often be required[2].

- An important point to consider is that defining a microservices target state is less about the ultimate size of the codebase, but more focused around the logical separation of concerns – specifically data.

You can facilitate a successful decomposition journey by having a target state defined. By doing so, you delineate the future microservices optimally through encapsulating business capabilities,[3] while at the same time ensuring that the overall component composition is not so fine-grained that system performance will suffer when the service layer experiences high demand[4]. This optimized target state will also provide insight as to where to initiate the journey (typically smaller services that have minimal coupling to the rest of the monolith) and how to strangle the monolith systematically.

Our method consists of several techniques that collectively help define this optimal target state and, in the process, identify those features that can easily be decoupled by focusing on data. These techniques are discussed in sections three through six. Furthermore, outputs from employing the technique provide additional benefits to the enterprise that we will discuss in section seven.

2. THE BENEFITS OF BASING THE MICROSERVICES DECOMPOSITION STRATEGY ON A BUSINESS TAXONOMY

Decomposing a monolith can seldomly be done without underlying code features being enhanced or repaired at the same time. In practice, this often excludes a big bang approach to decomposition, and key then is how to systematically distill and release features from the monolith in such a way that business-as-usual project and maintenance work are not adversely affected.

You can attain this by abstracting what the enterprise does in a granular, atomic manner using business-friendly language. In practice, this can be achieved by defining a set of business capabilities and then mapping the monolith against the resultant set. By then defining an orderly manner through which you can release these business capabilities, one by association also defined the decomposition path for the monolith. However, the key to the end-to-end undertaking is also understanding the cohesion/lack of cohesion among the business functions as this knowledge will help one avoid making the service stack too granular.

Furthermore, you may describe a long-running workflow (such as selling a product/service to a new client) through a collection of business capabilities being triggered in sequence and/or parallel. Such a workflow thus represents a vehicle through which you can realize several business capabilities collectively to achieve a larger goal. In a technical implementation, a workflow orchestration engine can embody this business process workflow by executing microservices that each, in turn, aligns directly to a business capability. The approach thus helps align Product with IT more closely.

A collection of business capabilities that are structured logically in a hierarchy form a business taxonomy. Industry-standard business taxonomy frameworks do exist, such as the business process framework (eTOM) for the telecommunications industry, and the banking industry architecture network (BIAN).

When applying our method, an enterprise may choose to adopt such an industry framework, implement a derivation of one, or created a business taxonomy from first principles.

In the latter case, the taxonomy could potentially be defined through internal consultation and will most likely comprise of a functional view of the organization:

Sales | Contracting | Sign Contract,
Billing | Bill Calc | Calculate Taxes, etc.

Alternatively, the taxonomy may arise organically (bottom-up approach) by evaluating and abstracting details captured in existing technical models that describe the organization's business and IT processes (business modeling process flows, sequence diagrams, use cases, etc.). This bottom-up approach holds several advantages over predefining the functional taxonomy. These advantages include speed of analysis (significantly less time is needed for consulting activities), and avoiding the creation of artificial functional silos (with the corresponding duplication of data and associated synchronization related concerns). However, the process demands enterprise maturity in documenting their business and IT processes.

Cardinal to the custom approach, is that the taxonomy should ensure close alignment with key business concepts and language used within the enterprise to align business and IT better. Moreover, each business capability:

- Should be atomic – i.e., it provides a small, repeatable business-centric outcome.
- Is it usable across many different workflows? Workflows employ one or more capabilities to execute a long-running process, and typically provide more significant business value (such as customer onboarding).

- Optionally creates, modifies, or references data as part of the atomic task that it performs. Note that the data it creates, modifies, or references may potentially be owned by a different business capability.

Also note that while business capabilities are atomic and independent, they share an indirect connection through common data objects. A data object represents data at a macro or business level (i.e., the collective concept of the customer's billing address as opposed to the customer billing address postal code.). A data object may be conceptually viewed as a data clustering or hierarchy, which is distinct from the business capability hierarchy. Data objects may be shared among several services. As a simple example, a customer's address can be used by a CRM related service to capture the underlying information. Later, a tax calculation service could use that information to determine appropriate taxation based on location. Lastly, an invoice generation service could use the information to decide which address the customer invoice will be sent.

Data objects play a cardinal role in determining how to cluster business capabilities into an optimized services stack.

As a simple example, consider a business capability taxonomy that has distinct entities for creating a contract with the customer, having the customer sign the contract, and having the organization countersign the same contract. One could theoretically create a microservices target state where each of these three atomic capabilities aligns directly with a dedicated microservice. However, the better solution may be to cluster the three capabilities into a single microservice simply because they will all use the same data objects to achieve their respective outcomes. By using the single microservice approach, you still preserve the concept of business to IT transparency. Still, you will reduce overall network traffic will during implementation as the three functions will, for the most part, feed off data stored in their common service database, as opposed to needing to query across services. It also helps resolve the issue of data ownership, i.e., to which service does the physical implementation of the client contract data object belong to.

As a final note to the question of the granularity that business capabilities should be defined when defining a custom taxonomy. For monolith decomposition, the choices made are tolerant to under/oversizing. This is because:

- Capabilities that were defined too granular will eventually cluster together, as will be discussed in section 4.

- Capabilities that were defined at a level that is not granular enough will ultimately reveal themselves as being associated with too many data objects (using the method that we describe in section 3). In these cases, you should decompose the business capability into two or more capabilities of greater atomicity.

3. MAPPING MONOLITH BUSINESS CAPABILITIES TO DATA OBJECTS

As we previously mentioned, the method we describe requires knowledge of the association between data objects and business capabilities. Because a microservice essentially is a combination of business functions, data needed by the functions, and API endpoints through which the functions can be executed and/or create, retrieve, update and delete (CRUD) operations on the data may be performed.

As a proof-of-concept, a subset of customer journeys, collectively forming a new initiative for one of our clients, was used to define a custom taxonomy from first principles as part of a pilot project. The taxonomy was defined and refined along with seven key activities:

1. **Identifying all the long-running workflows by analyzing business process models** as well as app (Android/iOS) screen flow mock-ups and business-level requirements for the initiative. The resultant dataset became the primary dimension of the taxonomy analysis.

2. **Defining business capabilities referenced inside the business process models and app screen flows by analyzing activities and generating abstractions.** For example, a set of activities inside a logic branch in a Business Process Model (BPM) may pertain to obtaining user personal information, and others may pertain to obtaining a client's account balance. These abstractions, when defined granular and atomic as described previously, are the business capabilities, and form the secondary dimension of the taxonomy analysis. Note that business capabilities are conceptually usable across several different workflows. As an example, the business capability "lookup account balance" is shared among "make an investment" and "transfer money to a different account" workflows, even though the actual underlying implementation of the business capability may potentially be different among the workflows.

3. **Cross-mapping the workflows that were identified to the business capabilities that were defined to lock down the main taxonomy structure.** This step plots the correlation between the primary and secondary dimensions referenced above (points one and two) and becomes the platform through which you perform the rest of the analysis. A conceptual and partial example of this is shown in figure

WORKFLOWS I BUSINESS CAPABILITIES	Upload Proof of Residence	…	Send Notification	Obtain Account Balance
Account Management				
…				
…		■		
Change Mobile Device				
…				■
…				
…			■	
…				
Reset Password			■	
…				
…			■	
…			■	
…			■	
Onboard Customer to Digital Platform	■			

Figure one. Partial cross-map of workflows to business capabilities (note that some data has been masked).

one based on work done for the pilot project. Note that the collection of activated business capabilities for any given workflow fully covers all aspects of that workflow, but that no sequence of business capability invocation is implied.

4. **Aligning client app screens to the business capabilities for each workflow as part of an IT enabler deep dive.** For our pilot, we had access to approximately 200 app screen mock-ups and associated logical flows. Each app screen mock-up was cross-referenced to the appropriate workflow/business capability cluster.

5. **Identifying and mapping which API endpoints are invoked for each app screen.** In addition to business process models and app screen mock-ups and flows, we also had access to Unified Modeling Language (UML) sequence diagrams that describe the detailed execution of core aspects of the workflows. This allowed us to correlate app screens to API endpoints, and then to use this association to cross-reference the API endpoints to the taxonomy.

6. **Extracting data objects from each referenced API endpoint by analyzing its specification.** For our pilot, this exercise was relatively trivial as all the APIs are RESTful, and each endpoint references a single data object. This may not always be the case: In practice, APIs may contain hundreds of data elements that may roll up to several data objects. In these instances, we recommend that automation around extracting data objects from the API specifications or underlying code is embraced.

7. **Once the data objects were extracted from the API endpoints, they were cross-mapped to the taxonomy through inference to the API endpoints that had already been mapped (see step six).** At this stage, a new view of the taxonomy can be generated by illustrating the correlation between business capabilities and data objects. This is depicted in figure two for a partial subset of data derived as part of the pilot project. As can be seen in Fig. 2, each business capability references one or more data objects. The collection of data objects that are associated with any given business capability is then that capability's data object thumbprint.

There is a good strategy for if any business capability appears to be overloaded with data objects, and if (and only if) these data objects are not pushed to the function. The strategy features re-evaluating the capability to determine whether you can split it into two or more capabilities of greater granularity since comparing the capability as is with any other capability will probably result in low cohesion results. To illustrate this, note that the 'Send Notification' capability in figure two is associated with several data objects. However, the majority of data objects, such as account and device, are pushed to the function (the function does not pull the data as part of sending notifications), and hence there is no need to split the function. However, had this data been pulled by the 'Send Notification' function, we would likely have split the capability into more atomic functions before continuing with the remainder of the analysis.

Now that we've defined the business capabilities, and each has been associated with one or more data objects, the next analytical step is to cluster the capabilities through their mutual data affinity.

DATA OBJECT I BUSINESS CAPABILITIES	Upload Proof of Residence	…	Send Notification	Obtain Account Balance
Client	■		■	
…		■		
Account		■	■	■
Mobile Device			■	
…			■	
…		■		
…			■	

Figure two. Partial cross-map of data objects to business capabilities (Note that some of the data had been masked)

4. OPTIMIZING THE SERVICE STACK THROUGH CLUSTERING TECHNIQUES

Two clustering methods were independently used to group the business functions based on data affinity: cosine similarity analysis, and k-means. The results from the two methods were compared, and a recommendation is made around the most appropriate technique for this particular problem domain.

4.1 Cosine Similarity

When using cosine similarity measurements, the cohesion among the various business capabilities can then be expressed as numbers with values residing between 0 and 1. This allows one to define (or, if needed, redefine) a threshold value which will drive the granularity and relative size of the clusters.

In practice, the collection of data objects that had been associated with each business capability is vectorized using a one-hot encoding approach, and a cosine similarity value is then calculated to determine the cohesion, or lack thereof, among the entire set of capabilities. Figure three conceptually illustrates how the data objects are vectorized for each business capability using one-hot encoding.

Note that the cosine similarity formula (see figure four) uses the dot product between, and magnitude of, the two vectors in question, and neither will be affected by the specific ordering of the dimensions that represent a data object in the set of vectors.

Business capabilities	**Data object vector**				
Business capability 1	1	0	1	…	1
Business capability 2	0	1	1	…	0
Business capability 3	1	0	0	…	1
…	…	…	…	…	…
Business capability n	0	1	0	…	0

Figure three. Vectorization of Data Objects by Business Capability. A value of 1 in the vector representation represents the fact that a business capability references a particular data object (such as account). In contrast, a value of 0 denotes that that capability does not reference that data object.

$$\text{similarity} = \cos\theta = \frac{A.B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure four. Cosine similarity formula where A and B represent two vectors[4]. Within the context of this discussion, A and B thus represents the data object composition (thumbprint) of any two business capabilities in the taxonomy we defined.

Taxonomy	BC 1	BC 2	BC 3	…	BC n-1	BC n
BC 1	1	0.5	0.95	…	0	0.65
BC 2	-	1	0.35	…	0.01	0.90
BC 3	-	-	1	…	0.025	0.40
…	-	-	-	1	0.03	0.33
BC n-1	-	-	-	-	1	0.02
BC n	-	-	-	-	-	1

Shows high cohesion and hence capabilities that may be clustered. Clusters become candidate µSs.

A capability that is highly isolated and can as a result easily be lifted out of the monolith as a stand-alone µS.

Figure five. Business capability similarity matrix.

Calculating the cosine similarity among all business capabilities will yield a matrix, as is shown in figure five.

In the matrix, values closer to 1 show a relatively high cohesion between the two business capabilities, and these are candidates for clustering together. Examples in figure five, illustrated with the circles in the matrix, include business capabilities one and three, as well as business capabilities two and n.

Conversely, values closer to 0 show little cohesion between two business capabilities, and they should logically not be clustered. Moreover, a business capability that has low cohesion to all other capabilities, as is the case for business capability (n-1) in figure five, can be completely isolated. These capabilities are excellent candidates for starting the microservices journey.

As part of this method, a cohesion threshold value should be chosen where cosine similarity values are greater than the threshold drives when business capabilities should be clustered. Depending on this threshold choice, clusters will logically be larger or smaller, but regardless, these clusters define target state microservice candidates.

Alternatively, the clustering process could be repeated recursively until all cosine similarity measurements among clusters have relatively low values and that no new clusters can logically be formed. Each supercluster that is derived through this process will be a target state microservice. This is illustrated in figure six that shows how each microservice candidate, through tracing to the hierarchy, is still directly aligned with a business capability.

Ultimately, our method provides for an optimized set of services that encapsulate granular business functions as best practice but are balanced at the same time by defining a minimal set of physical components in the services stack, which will help ensure improved network latency.

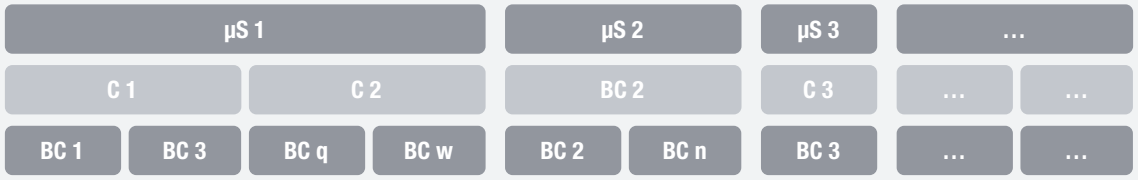

Figure six. Visualization of a target state microservice architecture that is derived through recursive clustering.

4.2 K-Means Clustering

Complementary to the Cosine analysis, the business capabilities were also clustered using the k-means method. Here, the same one-hot encoding values were used to represent the data object thumbprints for each of the business functions. A simple Python script was used to execute the algorithm, selecting the same number of output clusters as what cosine similarity analysis naturally revealed.

The clusters that resulted from executing the k-means script had approximately 75 percent direct correlation with that of the Cosine Similarity analysis. The main concern with using k-means as opposed to cosine similarity, (which requires more manual work) is that the algorithm has no context about the business capabilities themselves. So when clustering, it would, in some cases, lump business capabilities together that ideally should functionally be separated as they represent very different business domains. Accounting for most of the deltas between the two clustering methods. Note that k-means is known to have trouble clustering data where clusters are of varying sizes and density[6], which is certainly the case in this dataset.

As a result, while k-means certainly could be used to create a first approximation of the clusters, the cosine similarity approach allowed for greater control, and hence a more contextually relevant grouping of business capabilities, at the cost of more human involvement.

4.3 Data Ownership

After clustering, there may be cases where data ownership needs to be resolved as it will often be unavoidable that a subset of target microservice will need to operate on the same data object(s). This is illustrated in figure seven, where the data object composition has been rolled up to a microservices view.

For example, in figure seven, microservices one and two have a shared need for access to data object 3.

In these instances, a set of rules are defined to determine which microservice will own the data tables and API layer that provides access to the data underlying the common data object. These are, in order of importance:

1. Whether the business capability pulls the data or whether it is pushed towards it (such as in a notification service). When data is pushed towards it in the legacy system, the associated microservice will logically not be a candidate for ownership

2. Which function updates the data object more frequently

3. Which function consults the data object more frequently

	Data Object 1	Data Object 2	Data Object 3	…	Data Object n
µS 1	✓		✓	✓	✓
µS 2		✓	✓		
µS 3				✓	
…	…	…	…	…	…
µS y	…	…	…	…	…

Figure seven. Some data objects may be required by more than one microservice.

Putting this in practice, Table 1 shows a partial result from the project on which we piloted the method.

TABLE 1

The partial result from clustering and data assignment analysis for the project on which we piloted the decomposition approach.

Microservice	Contains Business Capabilities	Data Objects OWNED	Data Objects OWNED	Decomposition Priority	Consumes
CustomerService	Onboard Customer Change Profile …	Client Device …	Account	4	AccountService InformationService
AccountService	Obtain Account Balance Transfer …	Account Credit …	Client …	3	CustomerService NotificationService
NotificationService	Send Notification Cancel Notification …	Notification	Device Client	1	None
InformationService	… …	… …	… …	2	None

The type of view that is illustrated in Table 1 now helps define a decomposition roadmap.

5. CREATING A DECOMPOSITION ROADMAP

With the clustering exercise completed, a roadmap can be defined whereby clusters that can be decoupled from the monolith readily (due to low cohesion to any other clusters) are prioritized. This approach will also provide development teams the opportunity to validate and fine-tune development, testing and deployment strategies and best practices. All of which ensures the infrastructure serves the stated need (including potential data synchronization with the monolith legacy database), that non-functional requirements (performance and security) are satisfied, and that business-as-usual delivery of new business features are not impacted negatively.

Once delivery execution has matured for the first handful of microservices, a decomposition release train for the other services can be established. To construct the path of execution, we recommend decomposing, as far as possible, around those functional clusters in the business taxonomy where work intake will already occur for a given sprint. For example, if taxation rules need to change as part of the prioritized backlog for a given sprint, the corresponding business capabilities that roll up to the microservice that align to this should ideally be targeted for decomposition around the same time. In this way, stability around the rest of the IT ecosystem can be assured while optimizing the use of development, test, and project management resources.

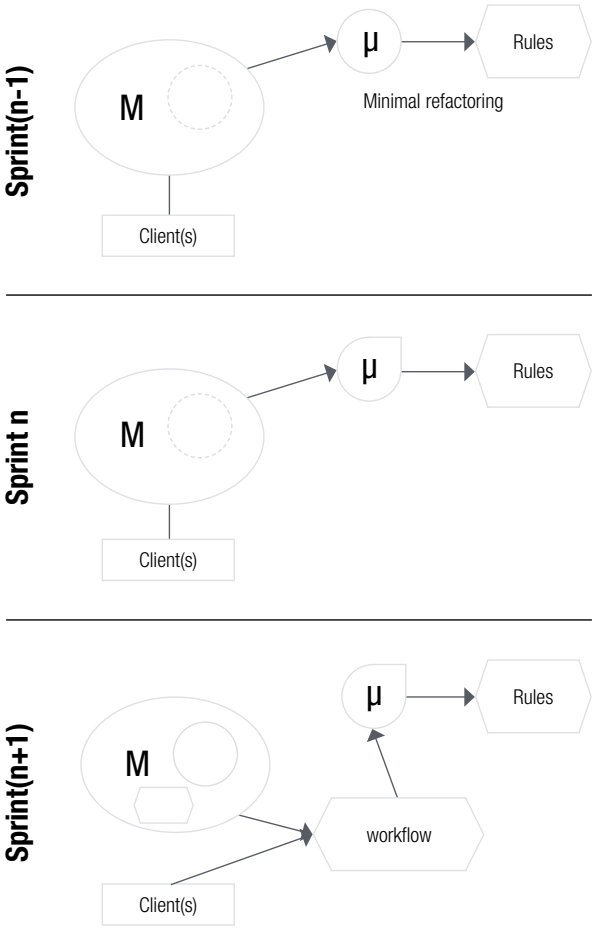

Figure Eight. The recommended decomposition strategy spread across three sprints.

An implementation strategy for this recommendation is as follows (assuming functional work will occur on a business capability cluster in sprint n - see figure eight for reference):

- Lift-and-shift the existing monolith code that corresponds to the targeted (clustered) business functions during the sprint (n-1) and regression test over this period. The code reorganization includes moving corresponding data structures, data synchronization with the legacy database, API endpoint locations, etc. Note that the API endpoint and data structure definitions should remain unchanged to ensure the stability of the greater code base, but that minor refactoring may occur (such as moving business rules to a dedicated rules-engine or to resolve technical debt). The new code base also needs to adhere to non-functional specifications that were established for microservices within the enterprise. This includes security, logging, availability, performance-related SLAs, etc., as well as to other enterprise architecture related mandates.

  At this stage, the monolith will still contain the original function call signatures (to ensure that the consumers are not impacted). Still, it will forward such calls to the new microservice and return the results to the calling consumer. Moreover, all internal monolith calls that accessed the original set of CRUD related API endpoints need to be refreshed to point to the new location/version.

- During sprint n, new business features (based on functional work intake) are added to the microservice. APIs and Data structures of the microservice may change to accommodate the new/changed features. Signatures in the monolith's forwarding calls, as well as the monolith data structures, may need to be updated as a result. Consequently, consumers calling into the monolith may need to update their codebase to account for such changes.

  Additionally, appropriate workflows will be added to the orchestration service layer (assuming an orchestration approach is followed). Still, it will not yet be wired up to the monolith and microservices collection for general consumption (but may be tested as part of a controlled introduction deployment strategy).

- During sprint (n+1), client calls will be routed to the orchestration service for general consumption, and comprehensive regression testing will be conducted.

6. MANAGING NEW BUSINESS FUNCTIONALITY

After the microservices initiative is complete, or even while it is in flight, it is expected that new business features will need IT enablement as part of standard IT project work. The question is whether such new features should be added as new services in the microservices stack or whether existing services should be extended.

The same clustering method that was used for slicing the monolith, can now be reused to help with these decisions via knowledge of the data object thumbprint for each microservice. This implies that data object thumbprint information should be maintained for the collection of microservices.

The new business feature is then analyzed in terms of its own data object needs, and its data object thumbprint is subsequently determined. Using the exact same method as was described in section 4.1, the affinity between the new business feature and the existing microservices are determined. Based on the findings, either a new service is proposed, or one or more of the existing microservices are extended to encapsulate the new feature.

7. OTHER BENEFITS OF MAPPING THE BUSINESS TAXONOMY TO THE MICROSERVICES STACK

As well as defining a monolith decomposition strategy, there are several other benefits in employing a business capability taxonomy in an enterprise. Some of these benefits can help the enterprise on its journey towards establishing BizDevOps practices, others can provide strategic insight for optimizing operational improvement initiatives. A few examples are listed below in Table 2.

TABLE 2

Short and long-term benefits of employing a business capability taxonomy

Realization	Benefit
Short Term	• It helps with scrum teams functional collision detection early in the software development lifecycle. By mapping work intake to business capabilities for each project/scrum team, an early readout can be obtained about inter-team dependencies, which, in turn, allows for more effective project management and potentially avoiding complex code merge issues by ensuring the work execution occurs in different sprints. • Helps architects systematically analyze which IT components are impacted, i.e., which monolith classes and/or microservices will need to be changed based on new work intake. Furthermore, when a specific business capability is determined to be impacted, the data objects associated with this capability are analyzed for structural impact (example, street name field length needs to be increased from 30 to 50 characters). The knowledge thereof drives secondary business capability impact analysis as data objects are often shared among business capabilities. • Helps test analysis systematically where new work extends existing features. This can significantly facilitate regression test analysis if the additional effort is taken to map the taxonomy to test cases.
Medium/Long Term	• Helps mapping of enterprise project issues, test defects and production tickets to the business capability framework and quantitatively highlight where process optimization will yield the best ROI. Assuming a defect can be traced to a microservice or API endpoint, for example, following the breadcrumb trail will lead to an impacted business capability. Through analysis of many defects and incidents, patterns will emerge that can provide an enterprise with data-driven insight around which business capabilities are the most troubled. Following the Pareto 80/20 rule, that small subset of business capabilities that collectively cause the most harm to the enterprise can consequently be isolated and analyzed for areas of improvement and re-engineering. • Predictive models via cognitive automation: Historical data around specific inputs and associated outcomes can be mapped to the business capability framework and used to train cognitive models that in turn can, via mapping of new work intake to the same taxonomy, be used to predict outcomes (e.g., whether the project will result in budget overruns, etc.).

8. EVOLVING THE METHOD

While the method yielded very satisfactory results for the pilot that was conducted, two areas were identified that would require further thought and analysis to accelerate its adoption as a standard practice. These are:

- **Mining information needed to build the taxonomy:** For the pilot, we had a very well documented business process models, App screen flows, UML Sequence Diagrams, and API specifications at our disposal – there was little to do in terms of mining the information, and the details were sufficient to construct the business taxonomy largely independently. This will not always be the case as one may be in a situation where the legacy system(s) were created 2 or 3 decades ago, and documentation is outdated or missing, and/or subject matter experts left the enterprise. In a scenario such as this, it would be helpful to have a tool that can automatically / semi-automatically mine the information needed to build out the taxonomy by scanning the underlying legacy codebase.

- **Automation in extracting data objects from API Specifications and/or code:** In our pilot, APIs were RESTful, and well documented. Moreover, we only had approximately 50 API endpoints to analyze. As a result, extracting data objects was a trivial exercise for the pilot. However, in the case where there may be thousands of poorly documented APIs and where multiple data objects may be transported through the endpoints, human labor will be an inefficient way of extracting the required information. Automation, possibly through applying machine learning techniques, may be highly valuable to reduce the amount of labor needed to perform this critical task accurately.

9. CONCLUSIONS

An optimized decomposition strategy of a monolith to a collection of microservices, can be derived through the adoption of an appropriate business taxonomy, the alignment thereof to legacy IT constructs, and clustering business capabilities in the taxonomy by means of data object affinity.

The approach outlined here is not only a once-off exercise. As the enterprise business evolves its products and services over time, and expresses this evolution via IT enhancements, the exact same technique can be used to package new functions and data in appropriate service containers – i.e., the method will recommend whether such IT functions and data belong in new microservices or whether one or more of the existing services should be enhanced. This way, a consistent, repeatable, and measurable approach to IT evolution, and that has good alignment with the product, can be established and maintained.

Moreover, while the approach described was focused on decomposing a single monolith, the approach outlined is equally valid for a distributed IT environment with several monolithic applications. Specifically, secondary and tertiary monolith applications' decomposition components can be compared to the microservices that were derived from decomposing the first monolith using the same approach. In this way, duplication of business features and data can be avoided by making enhancements to established microservices where there are areas of functionality overlap.

Key to the analysis is selecting or defining a business taxonomy that has the appropriate level of granularity. Moreover, the business functional taxonomy becomes a cornerstone against which many other IT related activities can be performed, including requirements and architectural analysis, regression testing needs, as well for providing as deep data driven insight into an enterprise's most troubled operations.

WORKS CITED

1. Fachat, André. *Challenges and benefits of the microservice architectural style*, *Part 1*, 2019, Web Page: https://developer.ibm.com/technologies/microservices/articles/challenges-and-benefits-of-the-microservice-architectural-style-part-1/

2. Kozłowski, Albert. *How Microservices Architecture Impacted the Culture of Software Development*, 2019, Web Page: https://medium.com/better-programming/how-microservices-architecture-impacted-the-culture-of-software-development-6bba363ecdf1

3. Lewis, James. *Microservices, a definition of this new architectural term*, 2014, Web Page: https://martinfowler.com/articles/microservices.html

4. Laskowski, Dave. *Moving to Microservices: How Granular Should My Services Be?*, 2019, Web Page: https://tcblog.protiviti.com/2019/09/04/moving-to-microservices-how-granular-should-my-services-be/

5. Prabhakaran, Selva. *Cosine Similarity – Understanding the math and how it works (with python codes)*, 2018, Web Page: https://www.machinelearningplus.com/nlp/cosine-similarity/

6. Yordan P. Raykov, et al. *What to Do When K-Means Clustering Fails: A Simple yet Principled Alternative Algorithm*, 2016, Web Page: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0162259

AUTHOR

Gerhardt Scriven, Principal Consultant

CONTACTS

Luciano Sobral, Executive Director
Luciano.Sobral@capco.com

Alex Corsi, Partner
Alessandro.Corsi@capco.com

CONTRIBUTORS

Renato Pedroso, Principal Consultant
Christian Roberts, Managing Principal
Ali Salamati, Senior Consultant
Paul Henry, Consultant
Alvaro Silva, Principal Consultant
Felipe Vinturini, Principal Consultant
Julio Lima, Principal Consultant
Ricardo Schuette, Principal Consultant
Carlos Dutra, Principal Consultant

ABOUT CAPCO

Capco is a global technology and management consultancy dedicated to the financial services industry. Our professionals combine innovative thinking with unrivalled industry knowledge to offer our clients consulting expertise, complex technology and package integration, transformation delivery, and managed services, to move their organizations forward.

Through our collaborative and efficient approach, we help our clients successfully innovate, increase revenue, manage risk and regulatory change, reduce costs, and enhance controls. We specialize primarily in banking, capital markets, wealth and asset management and insurance. We also have an energy consulting practice in the US. We serve our clients from offices in leading financial centers across the Americas, Europe, and Asia Pacific.

To learn more, visit our web site at **www.capco.com**, or follow us on Twitter, Facebook, YouTube, LinkedIn and Instagram.

WORLDWIDE OFFICES

APAC
Bangalore
Bangkok
Hong Kong
Kuala Lumpur
Pune
Singapore

EUROPE
Bratislava
Brussels
Dusseldorf
Edinburgh
Frankfurt
Geneva
London
Paris
Vienna
Warsaw
Zurich

NORTH AMERICA
Charlotte
Chicago
Dallas
Houston
New York
Orlando
Toronto
Tysons Corner
Washington, DC

SOUTH AMERICA
São Paulo

WWW.CAPCO.COM

CAPCO
THE FUTURE. NOW.