**OOKA - Lösung des Übungsblatts Nr. 4**-Lukas Lesch & Martin Petersohn-

Statement Nr. 1:

„*Wir haben eine strenge Komponenten- und Schichtenbildung, entsprechend haben wir*

*auch Komponenten- und Schichtenteams*. *Das kann nur Vorteile haben, das wird*

*übernommen*!“! Nach kurzer Diskussion beruht man auch für Microservices auf diese

Einteilung und verspricht sich dadurch einen klaren Vorteil. Was denken sie dazu? Was

würde sie dem Unternehmen eher empfehlen?

* Eine Beibehaltung der Komponententeams ist über der Microservice-Ansatz möglich, jedoch nur in vertikaler Form je fachlicher Kontext, die Schichtenteams würden dementsprechend obsolet werden. Durch die Aufspaltung der Komponenten in einzelne Prozessschritte bzw. eine denkbare Duplizierung der Microservices je nach Nutzergruppe, müssten entsprechend granularer Teams pro Microservice gebildet werden. Ggf. müssten die Teams nicht auf alle duplizierten Microservices aufgeteilt werden, sofern diese funktionell (fast) Identisch sind.

Statement Nr. 2:

„*Das wird schon! Die Geschäftsführung des Unternehmens muss doch für so eine*

*geringfügige Architekturänderung nicht informiert werden*.“ Was denken sie über dieses

Statement?

* Es handelt sich keinesfalls, um eine geringfügige Architekturänderung, da aufgrund des Microservice-Ansatzes eine vollständig neue Software-Architektur erzeugt werden muss. Da beim Microservice-Ansatz ggf. viele Komponenten dupliziert werden, kann dies zu einer erhöhten Server- bzw. Ressourcenkapazität führen. Dies muss mit der Geschäftsführung abgesprochen werden.

Statement Nr. 3:

„*Wir können doch schon Cloud – wir sind cloud-ready!*“! Da man die Anwendungsschicht

auf *einer* Virtuellen Maschine (VM) deployed hat, resultiert man konsequenterweise,

dass diese VM auch einfach auf einen Cloud-Anbieter (z.B. Amazon WS) ausgelagert

werden kann. Die Anwendung gilt somit als *skalierbar*! Es besteht eh keine

Notwendigkeit der Skalierung einzelner Komponenten, so die Bereichsleitung.

Wie stehen sie zu diesem Urteil?

* Die Skalierung einzelner Komponenten kann aufgrund von schwankenden Nutzerzugriffen durchaus sinnvoll sein, da sonst Flaschenhälsen innerhalb der Anwendung auftreten können, die zu einer schlechteren Performance führen.  
  Zudem ist die Architektur der Web-API relevant für die Systemperformance. Beispielsweise könnte die Web-API als Flaschenhals agieren, sofern diese keinen großen Lasten standhalten kann über Bspw. einen Load-Balancing-Ansatz. Der Ansatz über Virtuelle Maschinen ist hinfällig, da diese nur ein eigenes isoliertes System funktionieren, und nicht dazu entworfen wurden untereinander zu kommunizieren.

Statement Nr. 4:

„*Ach immer dieses API-Team...*“. Das „API-Team“ (vgl. Baustein-Sicht) hat sich wohl in

letzter Zeit oft über eine zunehmende Komplexität der Komponente „GP-API“ aus dem

Presentation Logic Layer beschwert, gibt aber Begründungen nicht explizit an (-- man

gibt zu, vielleicht auch nicht richtig hingehört zu haben...). Was für Gründe könnten das

sein aus ihrer Sicht? Wie könnte man diese Schicht ggf. im Rahmen der Microservice-

Architektur optimieren?

* Es wäre durchaus sinnvoll eine eigene dedizierte API für jeden Webservice zu erstellen, welche sich lediglich um die für Use-Case relevanten Anforderungen kümmert. Abhängig vom UI-Device erfolgt der Zugriff auf die API über alternative Wege. Aufgrund der Gleiche-Quelle-Regel könnte ein Zugriff über den Webserver nur über einen Proxy ermöglicht werden, was die Komplexität der API massiv erhöhen würde. Hierzu würde sich BFF als Ansatz für die API-Architektur anbieten, wodurch dedizierte APIs für jeden Use-Case erstellt werden.

Statement Nr. 5:

„*Die Analyse-Komponente ist DAS Problem überhaupt!*“! Leider sorgt die Analyse-

Komponente innerhalb der Komponente „Manufacturing Products“ häufig für Probleme

während der Konstruktion von Produkten (siehe Erläuterungen in Dokument). Was sind

hier konkret die Probleme? Wie sähe eine mögliche Lösungsstrategie aus?

* Ein denkbarer Ansatz wäre es, dass die Komponente „Manufacturing Products“ zu mehreren Microservices aufgespalten wird, wodurch eine parallele Verarbeitung der einzelnen Prozessschritte, wie bspw. Die Analyse-Komponente, ermöglich werden würde. Des Weiteren könnten die bereits analysierten Konfigurationen in der Datenbank gespeichert werden, damit bei gleichen Konfigurationen nicht erneut eine Analyse durchgeführt werden muss.

Statement Nr. 6:

„*Unsere langfristige Strategie: ein globales und universales Datenmodell!*“! Da man auch

immer noch diverse Bestandsysteme verwendet (SAP ERP, Oracle CRM), sieht man die

Etablierung eines gemeinsamen universalen Datenmodells als unabdingbar. Was

denken sie? Wie würden sie hier mit den heterogenen Datenmodellen umgehen? Wird

diese Meinung von allen Entwicklerteams mitgetragen?

* Ein globales und universelles Datenmodell steht im direkten Konflikt mit dem gesamten DDD-Ansatz und sorgt dafür, dass eine Umsetzung via Webservices hinfällig werden würde. Die jeweiligen Prozesse werden in separate fachliche Kontexte aufgeteilt, welche zur Steigerung der Effizienz auf ihren jeweiligen Kontext zugeschnitten sind. Entsprechend verfügt jeder fachliche Kontext über eine eigenes, individuelles Datenmodell. Zur Kommunikation außerhalb werden Schnittstellen benötigt, welche das externe Datenmodell über bspw. einen Adapter in das interne Datenmodell übersetzen.

Statement Nr. 7:

„*Das Entwicklungsteam will ja unbedingt ein Monitoring. Warum das bitte?? Zu teuer und*

*wenig aussagekräftig. Das lassen wir mal!“.* Was sind aus ihrer Sicht die Vorteile einer

Monitoring-Komponente? Gibt es Herausforderungen?

* Ein Monitoring hinsichtlich der System-Performance ist eine durchaus sinnvolle Ergänzung, da bezogen auf den Microservice-Ansatz eine dedizierte Skalierung der Microservices je nach Bedarf/Last möglich und ebenfalls sinnvoll wäre. Des Weiteren könnte das Monitoring einen Beitrag zur Ausfallsicherheit leisten, da somit analysiert werden könnte, welche Konfigurationen innerhalb der Analyse zu langen Laufzeiten führen, oder sogar einen Systemcrash herbeiführen.