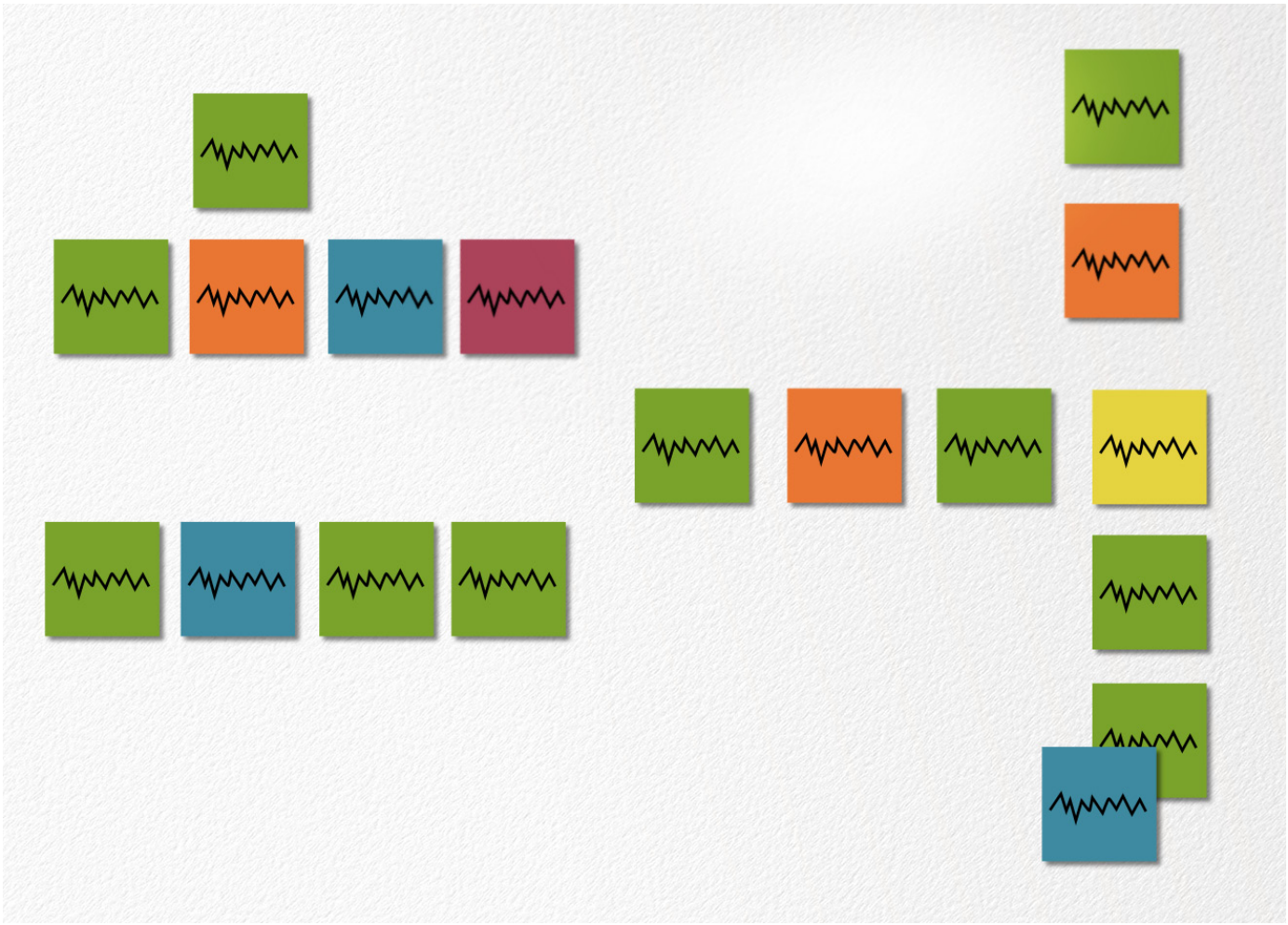


VON DER FACHDOMÄNE ZUM MODELL

Der nächste Urlaub kommt bestimmt

Vorausgesetzt, der digitale Urlaubsantrag ist bis dahin realisiert. Ein Erfahrungsbericht.

Urlaub zu beantragen ist gar nicht so einfach, wie es auf den ersten Blick scheint. Das beginnt bereits bei der Frage, was mit „Urlaub" überhaupt gemeint ist: Je nachdem, wen man fragt, erhält man eine andere Antwort. Wie lassen sich diese Fragen mit DDD und Event Storming beantworten?

Software nach den Grundsätzen von „Domain-Driven Design" (DDD) zu entwickeln, bedeutet, die fachliche Domäne absolut in den Vordergrund zu stellen und alles daranzusetzen, die zugrunde liegenden Prozesse korrekt zu erfassen, um sie dann in Software abbilden zu können.

Ein umfassendes Domänenmodell kann auf verschiedene Weisen erarbeitet werden, jedoch hat sich die Technik des „Event Storming" nach Alberto Brandolini [1] als äußerst geeignet erwiesen, um alle Mitglieder eines Entwicklungsteams – fachliche wie technische Experten – gleichermaßen in den Entstehungsprozess einzubinden und so bestmögliche Ergebnisse zu erzielen.

Schnelles Lernen in Iterationen

Um sich einer Fachdomäne mittels Event Storming zu nähern, braucht es nicht viel: Ein geeigneter Raum sollte über große, leere Wände verfügen, die idealerweise beschreibbar sind. Mindestens drei Sorten von Post-its werden benötigt, um „Events", „Commands" und „Aggregates" der Domäne visualisieren zu können.

Hier hat sich folgendes Farb- und Formschema etabliert: Hellgelbe, rechteckige Klebezettel werden zur Darstellung von Aggregates benutzt, quadratische orangefarbene ▶

Post-its stellen die Events dar und quadratische blaue Post-its werden für die Commands verwendet.

Zusätzlich können noch Klebezettel für Kontexte, externe Systeme oder auch Notizen mit jeweils eigenen Farben eingeführt werden. Jedes Team-Mitglied braucht einen Stift, denn wirklich jeder soll sich aktiv am Modellierungsprozess beteiligen.

Die Modellierung sollte in eher kurzen Sessions von insgesamt zwei bis drei Stunden stattfinden und an einigen aufeinanderfolgenden Tagen wiederholt werden. Der Anspruch auf ein vollständiges oder perfektes Ergebnis darf einem Team dabei nicht im Weg stehen: Ein Modell kann immer nur eine Annäherung an eine Domäne darstellen, niemals ein perfektes Abbild. Dieses darf also auch gar nicht das Ziel sein.

Von Iteration zu Iteration wird das gemeinsame Verständnis der zu implementierenden Domäne wachsen, und die kurzen Sessions helfen dabei, die Domäne immer wieder aus unterschiedlichen Perspektiven zu betrachten.

CRUD ist verboten

Wichtig hierbei ist es, dass sich alle auf das Experiment einlassen, sich auf die Ereignisse innerhalb der Domäne und ihre Geschäftsprozesse zu fokussieren. An Code und in Code zu denken ist in einem Modellierungsworkshop daher fehl am Platz. Gerade Entwicklern fällt dies schwer, da ihr Denken jahrelang in Richtung von Datenbankaktionen wie Create, Read, Update und Delete geprägt wurde. In diese Schubladen passen aber die wenigsten Geschäftsprozesse einer Domäne, will man nicht wichtige Benutzerintentionen aus dem Blick verlieren.

Modellieren bedeutet in diesem Zusammenhang vor allen Dingen auch das Experimentieren mit Sprache, denn diese bildet die Basis eines Domänenmodells. Im Grunde muss das Entwicklungsteam die verschiedenen Geschichten erzählen, die eine Domäne enthält. Würde man sich hierbei auf lediglich vier Verben wie Create, Read, Update und Delete beschränken, wären diese Geschichten sehr inhaltsleer und sicher nicht sehr zutreffend.

Die wichtigsten Begriffe

- **Command:** Ein Command ist ein Wunsch des Nutzers an das System, wobei das System diesem Wunsch stattgeben, ihn aber auch zurückweisen kann – beispielsweise wenn Berechtigungen fehlen oder wenn Geschäftslogik verletzt wird.
- **Event:** Ein Event ist eine Reaktion des Systems auf ein Command und stellt ein unwiderrufliches Ereignis dar. Events haben üblicherweise Statusänderungen zur Folge und dienen als Trigger für weitere Vorgänge.
- **Aggregate:** Ein Aggregate enthält die Geschäftslogik, anhand derer Commands verarbeitet und in Events umgewandelt werden. Außerdem stellt es die transaktionale Konsistenz-Grenze für die jeweilige Logik dar.
- **Bounded Context:** Ein Bounded Context bezeichnet einen Bereich innerhalb des Modells mit einer semantisch eindeutigen Sprache, die für die innerhalb des Kontexts angesiedelten Commands, Events und Aggregates verbindlich gilt.
- **Ubiquitous Language:** Die innerhalb eines Bounded Context gültige, semantisch eindeutige Sprache bezeichnet man als Ubiquitous Language.
- **Eventual Consistency:** Eventual Consistency ist ein Konzept der Datenspeicherung, bei dem die Konsistenz der Daten nicht sofort, sondern erst zu einem späteren Zeitpunkt garantiert wird.

Ab in den Urlaub

Der konkrete Einstieg in die Softwareentwicklung mit DDD und Event Storming fällt, vor allem ohne qualifizierte Projektunterstützung, recht schwer, nicht zuletzt auch aus dem Grund, dass die verfügbare Literatur oft sehr abstrakt bleibt.

Wir bei the native web [2] begleiten Unternehmen in ihrem Alltag dabei, ihre Geschäftsprozesse zu digitalisieren. In diesem Zusammenhang möchten wir einen Einblick geben, wie ein solcher Modellierungsworkshop in der Praxis ablaufen kann, um wichtige Konzepte und Methoden anschaulich darzustellen und typische Stolperfallen vermeidbar zu machen.

Unter anderem begleiten wir derzeit einen Kunden dabei, teildigitale organisatorische Prozesse vollständig zu digitalisieren. Im Juni 2017 haben wir daher das Thema „Digitaler Urlaubsantrag“ im Rahmen eines mehrstufigen Workshops modelliert.

Das Team bestand dabei aus sechs Personen und wurde zum größten Teil vom Kunden selbst gestellt: Teilgenommen haben von fachlicher Seite ein Mitglied der Geschäftsführung sowie deren Assistenz und von technischer Seite ein Frontend- und ein Backend-Entwickler. Von unserer Seite war ein Architekt als technischer Berater dabei. Außerdem haben wir die Moderatoren-Rolle mit sprachwissenschaftlich-technischem Background übernommen.

Nahaufnahme: Die Fachdomäne

Zu Beginn des Workshops haben die Fachexperten den Status quo des zu digitalisierenden Prozesses vorgestellt und gleichzeitig ihre Wünsche und Anforderungen konkretisiert. Dabei konnten sieben Schritte identifiziert werden, die der Urlaubsantrag bisher zur Genehmigung in der Realität durchläuft:

1. **Am Anfang steht der Urlaubswunsch:** Der Mitarbeiter bespricht mit den Mitgliedern seines Teams den Urlaubswunsch. Auf der Basis der Kapazitätsplanung wird entschieden, ob der Mitarbeiter einen Antrag auf Urlaub stellen darf oder nicht. In diesem Zusammenhang wird gleichzeitig die Urlaubsvertretung geregelt.
2. **Der konkrete Urlaubsantrag:** Wenn man im Team zu dem Ergebnis gekommen ist, dass nichts gegen den Urlaubsantrag des jeweiligen Mitarbeiters spricht, stellt dieser den konkreten Antrag.

3. **Die Projektleitung zeichnet den Urlaubsantrag ab:** Im Grunde ist der Urlaub zu diesem Zeitpunkt schon genehmigt – unter der Voraussetzung, dass der Mitarbeiter noch genug Urlaubstage auf seinem Urlaubskonto übrig hat.
4. **Der Urlaubsantrag wird an die Assistenz der Geschäftsführung weitergeleitet:** Momentan läuft dieser Schritt noch in Papierform ab und soll in der digitalisierten Variante direkt in der Software erfolgen, wobei die Assistenz via E-Mail benachrichtigt wird.
5. **Die Assistenz der Geschäftsführung prüft den Antrag:** Hierbei wird zunächst grundsätzlich auf Fehler geprüft und ermittelt, ob der jeweilige Mitarbeiter entsprechend ausreichend Urlaubstage auf seinem Konto zur Verfügung hat. Wenn alles korrekt ist, wird der gewünschte Urlaub im Urlaubskonto des Mitarbeiters erfasst und ist somit endgültig genehmigt.
6. **Dem Mitarbeiter geht eine Kopie seines genehmigten Urlaubsantrags zu:** Auch dieser Schritt läuft momentan noch in Papierform ab und soll zukünftig elektronisch erfolgen.
7. **Der genehmigte Urlaub wird in den Teamkalender eingetragen:** Dieser letzte Schritt wird bislang noch händisch von der Assistenz der Geschäftsführung ausgeführt und soll in der digitalisierten Variante durch einen automatischen Prozess ersetzt werden.

Es fällt auf, dass der Grad an direkter, persönlicher Kommunikation im Unternehmen des Kunden sehr hoch ist. Dies ist möglich aufgrund der Tatsache, dass es sich um ein recht kleines Unternehmen mit etwa 30 Mitarbeitern handelt, die in volatilen Teams mit hoher Flexibilität alle vor Ort zur gleichen Zeit arbeiten.

Diese direkte Kommunikation – vor allem im ersten Schritt, den Urlaubswunsch betreffend – soll erhalten bleiben. Abgeschafft werden soll hingegen der physische Urlaubsantrag, der bis dato noch in Mitarbeiter-Mappen abgeheftet wird. Hier möchte das Unternehmen papierlos werden.

Die Digitalisierung soll vor allen Dingen die Arbeitsabläufe der Geschäftsführungs-Assistenz erleichtern, aber auch den Mitarbeitern eine einfache Möglichkeit bieten, ihr Urlaubskonto online einsehen zu können, denn dies ist bislang nicht möglich. Nach dieser Einführung durch die Fachexperten hatten alle Workshop-Teilnehmer einen guten ersten Eindruck von der zu modellierenden Domäne, und wir haben die erste Iteration gestartet.

Los geht's mit den Domänen-Events

Wir haben in unserer Praxis festgestellt, dass es für Entwickler und Fachexperten lehrreich und heilsam zugleich ist, zu Beginn einer Iteration einfach „draufloszumodellieren", um dann in einer Review-Runde festzustellen, dass man sich doch nicht mehr so ganz an die korrekte Form von Commands und Events erinnert hat oder dass man doch wieder in CRUD-Denken verfallen ist.

Sinnvoll kann es in diesem Zusammenhang sein, wenn man das Schema, dem Commands und Events folgen sollten, an prominenter Stelle im Raum platziert, damit jeder aus dem Team dieses im Blick behalten kann.

- **Command:** Verb im Imperativ + Substantiv, da ein Wunsch an das System geäußert wird, der einen Event zur Folge hat.
- **Event:** Substantiv + Verb in der Vergangenheitsform, da das gewünschte Ereignis in der Domäne stattgefunden hat und sich der Status geändert hat.

Die grundlegende Frage, die im Lauf der Modellierung immer wieder gestellt werden muss, lautet: Welche Domänen-Events braucht das System, um den Anforderungen der Fachabteilung gerecht zu werden?

Ab in den Papierkorb

Neben Massen von Post-its ist ein großer, leerer Papierkorb mindestens ebenso wichtig für das Gelingen einer Domänenmodellierung. „Aufschreiben, drankleben, diskutieren, wegwerfen" – diese Abfolge wiederholt sich unzählige Male, bis wirklich nur noch die Abläufe an der Wand stehen, die tatsächliche Events und Commands in der Domäne darstellen. Der unschlagbare Vorteil dieser Vorgehensweise ist es also, dass Implementierungsfehler, die ja meist auf Missverständnissen zwischen Fachexperten und Entwicklern basieren, erst gar nicht im Code ankommen können, da sie vorher im Papierkorb enden.

Selbstverständlich heißt das nicht, dass jede Implementierung, die Ergebnis einer Modellierung auf Basis von DDD ist, völlig fehler- und missverständnisfrei ist. Die Wahrscheinlichkeit aber, dass am Ende das Ergebnis in Code herauskommt, das die Fachexperten gewünscht haben, ist um ein Vielfaches höher als bei klassischen Herangehensweisen, bei denen fachliche und technische Seite getrennt voneinander arbeiten.

Sollten doch Fehler in der Modellierung passieren, sind diese leicht zu verschmerzen: Post-its von der Wand abziehen und wegwerfen kostet so gut wie nichts; am Ende eines Projekts festzustellen, dass etwas völlig Falsches implementiert wurde, wird auf vielen Ebenen sehr teuer und schmerzlich.

Gerade zu Beginn einer Modellierungssession landen meist immer die gleichen Event-Kreationen im Papierkorb. Diese lassen sich wunderbar den verschiedenen Teilnehmer-Gruppen eines solchen Workshops zuordnen: Wie bereits erwähnt, fällt es Entwicklern oft schwer, sich von Datenbank-Aktionen zu lösen, weswegen sie sich in ihrer Sprache selbst einschränken. Während Entwickler also in Code denken, können sich Designer oft nur schwer davon lösen, in Bildern, Eingabemasken oder UIs zu denken.

Fachexperten hingegen haben üblicherweise die geringsten Probleme, Domänen-Events treffend zu bezeichnen. Auch im weiteren Verlauf einer Modellierung fällt es ihnen leicht, die zugehörigen Commands, Aggregates und übergreifende Kontexte zu identifizieren. Daher ist es immens wichtig, dass während des Modellierungsprozesses wirklich alle gemeinsam arbeiten.

In unserer Praxis haben sich einige schnelle, kurze Modellierungs-Iterationen von 15 Minuten zu Beginn eines Workshops bewährt, die auf die grundlegenden Prozesse innerhalb der abzubildenen Domäne fokussieren. In Verbindung mit kurzen Review-Runden bekommt ein Team auf diese Weise schnell eine gemeinsame Perspektive. ▶

Unverzichtbar: Der Moderator

Während Entwickler und Designer sich im Verlauf eines Modellierungsworkshops darauf einlassen können, von ihren erlernten Denkmustern abzuweichen, fällt es – gerade in den ersten Iterationen – den Fachexperten manchmal schwer, sich gegen die technischen Experten zu behaupten. Wenn Entwickler in einer Review-Diskussion den Fachexperten technische Implementierungsdetails als Argumente für oder gegen einen Diskussionspunkt entgegnen, ist die Gefahr groß, dass die fachliche Seite gedanklich aussteigt, weil sie sich unverstanden und überflüssig fühlt.

Hier kommt die Rolle des Moderators ins Spiel, der unverzichtbar dafür Sorge zu tragen hat, die jeweiligen Standpunkte voreinander zu schützen und miteinander zu integrieren. Idealerweise hat der Moderator einen Background, der ein großes sprachliches Talent mit der Fähigkeit verbindet, sich sowohl in geschäftliche als auch in technische Prozesse hineinzudenken.

Empathiefähigkeit und ein sicheres Gefühl dafür, wann sich ein Workshop-Teilnehmer außen vor fühlt und sich aus der Teamarbeit geistig verabschiedet, sind ebenso essenziell. Es ist die Aufgabe des Moderators, alle immer wieder daran zu erinnern, dass die Modellierung nur gemeinsam ein Erfolg werden kann und jeder einzelne Beitrag nötig und wichtig ist.

Nur in einer wertschätzenden Atmosphäre, die von gegenseitigem Respekt geprägt ist, wird jedes Teammitglied seinen Beitrag im Modellierungsprozess leisten können. Der Moderator sollte auch in den Review-Runden eine gewisse anleitende Funktion übernehmen und auf diese Weise dazu beitragen, dass alle ihre jeweiligen Hürden im Denken überwinden können. Dem einen gelingt dies besser und schneller, der andere braucht länger und mehrere Denkanstöße. Werden diese Hürden aber gemeinsam genommen, so wachsen Teamgeist und Zusammenhalt.

Nahaufnahme: „Urlaub" ist nicht gleich „Urlaub"

Immer unter der Prämisse, dass die Ergebnisse nur ausschnitthaft, kundenspezifisch und stark auf das Wesentliche reduziert sind, folgen nun einige konkrete Beispiele aus dem oben schon genannten Kundenworkshop, um zentrale Elemente einer Modellierung mit Leben zu füllen.

Nachdem die Teilnehmer des Workshops zunächst auf die Substantive „Antrag" und „Urlaub" fokussiert hatten, stellten wir nach kurzer Zeit fest, dass das Wort „Urlaub" die weitaus gewichtigere Rolle in den Prozessen der vorliegenden Domäne spielt. Domänen-Events wie „Urlaub beantragt", „Urlaub bewilligt" oder auch „Urlaub abgelehnt" fanden allseitige Zustimmung.

Mindestens ebenso schnell stießen wir aber auch auf den Umstand, dass das Substantiv „Urlaub" ein perfektes Beispiel dafür ist, wie verschiedene Bereiche zwar den gleichen Begriff verwenden, diesen aber mit unterschiedlicher Semantik aufladen.

„Urlaub" ist also nicht gleich „Urlaub": Für Mitarbeiter bedeutet „Urlaub" die konkrete Abwesenheit und die Verpflichtung, eine Urlaubsvertretung zu regeln. Für die Geschäftsführung ist „Urlaub" der Anspruch, der sich aus einem Arbeitsvertrag ergibt. „Urlaub" kann außerdem die Bezeichnung für den Anteil sein, der einem Mitarbeiter in einem Kalenderjahr noch zur Verfügung steht, aber ebenso gut auch die Tage bezeichnen, die der Mitarbeiter bereits frei hatte.

Nahaufnahme: Das Aggregate „Urlaub" in verschiedenen Kontexten

Was nun? Es kamen Ideen auf, die entdeckten Aggregates als „Urlaub-A" und „Urlaub-B" zu bezeichnen, oder auf Spezifizierungen wie „zustehender Urlaub", „genommener Urlaub" und „Rest-Urlaub" auszuweichen. Richtig passend waren all diese Konstrukte allerdings nicht, weshalb das Team zu keinem Konsens gelangte.

Wir haben daher die Erkenntnisse aus einem anderen Blickwinkel betrachtet und kamen zu dem Ergebnis, dass es nicht das Substantiv „Urlaub" ist, das einer Spezifizierung bedarf, sondern dass wir es mit unterschiedlichen Kontexten zu tun haben, die das Wort „Urlaub" jeweils mit der Semantik ihrer eigenen Sprache aufladen.

Was wir gefunden hatten, bezeichnet man als „Bounded Contexts", die durch sprachliche Grenzen scharf voneinander abgegrenzt sind. Die Sprache, die in einem solchen Kontext sinnstiftend ist, wird „Ubiquitous Language" genannt – allgegenwärtig, aber nur innerhalb der jeweiligen kontexturalen Grenzen. Wir konnten also zwei Aggregates mit der Bezeichnung „Urlaub" in zwei unterschiedlichen Kontexten identifizieren und diese nun mit den jeweiligen Event- und Command-Paaren in Zusammenhang setzen.

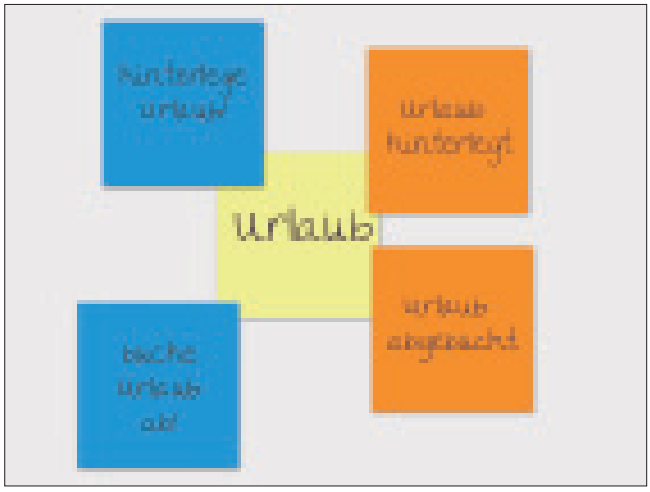
Die von uns identifizierten Kontexte haben wir als „Mitarbeiterverwaltung" und als „Urlaubskalender" bezeichnet. Der erste Kontext enthält unter anderem das Aggregate, das den allgemeinen „Anspruch auf Urlaub" verarbeitet, der zweite Kontext enthält das Aggregate, das den „Urlaubsantrag" abbildet.

Im Kontext „Mitarbeiterverwaltung" enthält das Aggregate „Urlaub" unter anderem die Commands „hinterlege Urlaub!" und „buche Urlaub ab!" zusammen mit den entsprechenden Events, um den Anspruch auf Jahresurlaub initial eintragen und bei genehmigten Urlaubsanträgen die entsprechenden Urlaubstage abziehen zu können (siehe Bild 1). Im Kontext „Urlaubskalender" umfasst das Aggregate „Urlaub" beispielsweise die Commands „beantrage Urlaub!" beziehungsweise „genehmige Urlaub!" mit den zugehörigen Events, denn hier stellt der Mitarbeiter seine Urlaubsanträge und bekommt diese genehmigt oder nicht (siehe Bild 2).

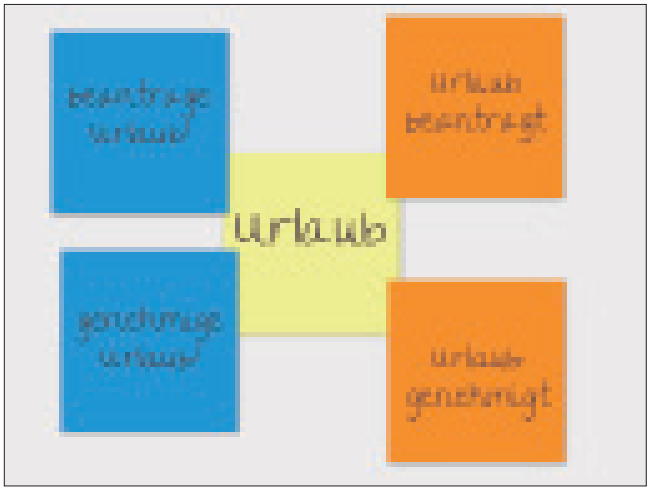
Nahaufnahme: Wie begegnet man Eventual Consistency?

Aggregate-Grenzen fungieren allerdings nicht nur als sprachliche Grenzen, sondern auch als transaktionale Konsistenz-Grenzen. Ein Prozess, der auf unbedingte Konsistenz angewiesen ist, muss zwingend so modelliert sein, dass er sich innerhalb eines einzigen Aggregate befindet. Nur innerhalb von Aggregate-Grenzen kann Konsistenz garantiert werden.

Was bedeutet dies aber nun für unsere Modellierung? Die Prüfung, ob der Mitarbeiter Anspruch auf den gewünschten Urlaub hat, findet nicht nur in einem anderen Aggregate, son-



Im Kontext „Mitarbeiterverwaltung“ bezeichnet „Urlaub“ … (Bild 1)



… etwas anderes als im Kontext „Urlaubskalender“ (Bild 2)

dern sogar in einem anderen Kontext statt als die Organisation des Urlaubswunschs innerhalb des betreffenden Teams.

Es könnte also das Problem entstehen, dass bei der Antragstellung von einer anderen Zahl an verfügbaren Urlaubstagen ausgegangen wird, als nachher tatsächlich zur Verfügung stehen. Dies ist auf die unterschiedlichen transaktionalen Grenzen der beteiligten Aggregates zurückzuführen, deren Zustände unter Umständen zu verschiedenen Zeitpunkten aktualisiert werden, was als „Eventual Consistency“ bezeichnet wird.

Hier darf nun nicht der Fehler gemacht werden, fachliche Anforderungen technisch lösen zu wollen. Die nur gelegentlich gegebene Konsistenz mag im ersten Moment problematisch anmuten, bei näherer Betrachtung ist sie aber nur eine logische Konsequenz und sehr leicht beherrschbar. Die Frage, ob der jeweilige Mitarbeiter noch so viel Urlaub zur Verfügung hat, wie er gerne beantragen möchte, wird nicht im Schritt des Urlaubsantrags selbst beantwortet.

Hier behelfen wir uns mit einer vorläufigen Prüfung im UI, die aber nicht letztgültig ist. Es wird dem Mitarbeiter also zunächst ermöglicht, seinen Antrag zu stellen, die tatsächliche Prüfung, ob noch genügend Urlaubstage vorhanden sind, erfolgt jedoch erst im Genehmigungsprozess und damit in dem Kontext, der das Aggregate verwaltet, das die Informationen darüber besitzt, wie viel Urlaub der Mitarbeiter noch zur Verfügung hat.

Nahaufnahme: Mit Flows auf Events reagieren

Ein weiteres interessantes Konstrukt, das in unserem Modellierungsworkshop zum Tragen kam, ist der „Flow“ – oft auch als „Saga“ oder „Process Manager“ bezeichnet. Flows sind Möglichkeiten, auf Events zu reagieren, und in unserem konkreten Beispiel haben wir Flows implementiert, um verschiedene E-Mail-Benachrichtigungen anzustoßen.

Auf den Event „Urlaub beantragt“ reagiert ein Flow, der die Assistenz der Geschäftsführung per E-Mail über den Antrag in Kenntnis setzt, und auf den Event „Urlaub genehmigt“ reagiert ein Flow, der den Mitarbeiter darüber informiert, dass der Urlaub genehmigt und in den Kalender eingetragen wurde. Diese Beispiele sind an IFTTT angelehnte Flows und als solche statuslos, da nur eine simple Wenn-dann-Logik nötig ist, die in immer gleicher Form greift.

Fazit

Die größte Herausforderung in der Entwicklung von Software ist nicht technischer Natur, sondern befindet sich auf der Ebene der Kommunikation: Es fehlt an exakter Kommunikation über die zu implementierende Software genauso wie an Kommunikation zwischen den verschiedenen Disziplinen, die an der Entwicklung beteiligt sind.

DDD in Verbindung mit Designtechniken wie Event Storming verlangt es Entwicklungsteams ab, sich diesen kommunikativen Herausforderungen zu stellen, und hilft zugleich auch dabei, sie zu überwinden. Angemessene Kommunikation zum richtigen Zeitpunkt ermöglicht es, genau die Software zu entwickeln, die ein Fachbereich oder ein Kunde benötigt.

Die Arbeit in interdisziplinären Teams, die die technische und die fachliche Seite miteinander integrieren, verbunden mit der Entscheidung für DDD und Event Storming ist ein Prozess des Lernens und Weiterentwickelns auf menschlicher und fachlicher Ebene, der Zeit braucht. Wird diese Zeit jedoch investiert, führt das zu besserer Software in kürzerer Zeit. ■

[1] Ziobrando's Lair: Introducing Event Storming, www.dotnetpro.de/SL1711EventStorm1
[2] the native web, https://www.thenativeweb.io

Susanna Roden
ist Gründerin und Geschäftsführerin von the native web. Mit ihren Schwerpunkten in Soziologie und Linguistik hilft sie Teams dabei, ihre interne Kommunikation effizienter zu gestalten und so zu besseren Lösungen zu gelangen.
www.thenativeweb.io

dnpCode A1711EventStorming