Continuous UX – „Lean“ und „Large“ unter einem Dach

Thomas Immich

UX Services, Centigrade GmbH

thomas.immich@centigrade.de

Zusammenfassung

Lean UX ist inzwischen ein populärer Begriff in der gesamten UX Community. Bei allem Charme, den ein schlankes Vorgehen versprüht, bleibt allerdings die Frage offen: wie skaliert man „lean“ zu „large“? Der Artikel zeigt auf, wie mit dem praxis-erprobten UX Management Framework „Continuous UX“ aus mehreren unverbundenen MVPs ein kohärentes, tragfähiges Design System oder eine modulare UI Plattform erwachsen kann.

1 Einleitung

Lean UX sieht vor, Annahmen und UX Konzepte möglichst früh mittels „Minimum Viable Products“ (MVP) zu testen (Gothelf & Seiden, 2012). Mit dem gelernten Wissen, kann dann der Kurs korrigiert werden und das Team anschließend noch zielgerichteter weiterarbeiten.

An seine Grenzen kommt dieser grundsätzlich sehr smarte Ansatz dann, wenn die finale Zielsetzung ein kohärentes und skalierbares Design System oder eine modulare UI Plattform umfasst, auf die von mehreren verschiedenen Abteilungen einer einzigen Organisation zugegriffen werden kann. Plötzlich möchten viele Teams mit unterschiedlichen Geschwindigkeiten, heterogenem UX Prozesswissen und letztlich auch mit konträren Zielsetzungen für ihre jeweils lokalen Probleme die besten Lösungen umsetzen, ohne sich dabei von den globalen Problemen „der anderen“ und deren spezifischen Lösungen bremsen zu lassen.

„Continuous UX“[1] ist ein praxis-erprobtes UX Management Rahmenwerk, welches es ermöglicht, Design Systeme oder UI Plattformen evolutionär sowohl vertikal („lean“) als auch horizontal („large“) aufzubauen und zu skalieren.

[1] http://www.continuous-ux.com

Veröffentlicht durch die Gesellschaft für Informatik e. V. und die German UPA e. V. 2018 in
S. Hess, H. Fischer (Hrsg.):
Mensch und Computer 2018 – Usability Professionals, 02.–05. September 2018, Dresden.
 https://doi.org/10.18420/muc2018-up-0127

2 Kontinuierliche Integration und UX

Bereits seit Jahren existiert eine gefühlte Lücke zwischen UX Designern und Software Engineers. Die Disziplin „UX Design“ wird ihrem Namen insofern nicht gerecht, dass sie es aus sich selbst heraus nicht zu leisten im Stande ist, reale User Experience bis zur Auslieferung und darüber hinaus mitzugestalten. Zwar wird es UX Designern durch moderne Prototyping Tools wie Adobe XD[2] oder Google Sketch[3] erleichtert, lebendige Spezifikationen für Engineers zu liefern, sobald jedoch eine Nutzeranforderung implementiert ist, geht die Hoheit über die Anforderung häufig schleichend an die Engineers über. Eine kontinuierliche Verbesserung der User Experience, wie es der im Kern iterative nutzer-zentrierte Ansatz ja eigentlich begrüßt, wird dann aus der UX Design Warte unmöglich. Die UX Designer stehen vor verschlossener Tür und fühlen sich eher als „Vorarbeiter“, denn als „Entwickler“.

2.1 Integration in agile Entwicklungsprozesse

Continuous UX setzt direkt auf agilen Entwicklungsprozessen auf und begrüßt deren iterative und inkrementelle Natur. Anders als Scrum möchte Continuous UX jedoch nicht formal vorschreiben, welche Rollen in jedem Fall ohne Personalunion existieren müssen oder wie lange Meetings zu dauern haben, um konform zu sein (Pichler, 2007). Continuous UX sieht sich vielmehr als ein modulares Rahmenwerk, welches zwar in sich geschlossen ist und daher den gesamten agilen Entwicklungsprozess von Anfang bis Ende begleiten kann, auf der anderen Seite aber auch als Baukasten verwendet werden kann, aus dem in bestimmten Problemsituationen bestimmte Werkzeuge und Methoden zur Lösung herangezogen werden können.

Diese Flexibilität ermöglicht, dass Continuous UX in jeder Art von agiler Prozessvariante zum Einsatz kommen kann. Aus Sicht von Continuous UX funktioniert das Zusammenspiel mit Scrum dann besonders gut, wenn jedes Team-Mitglied seine gesamte Zeit für ein einziges Projekt aufwenden kann und nicht zwischen verschiedenen Projekten und Zielen zerrissen ist. Ist eine solche Exklusivität nicht möglich, empfiehlt sich eher ein Kanban-orientierter Ansatz (Ohno, 1993).

2.2 Continuous Delivery

Der Gedanke der „Kontinuität“ ist in der modernen Software Entwicklung inzwischen allgegenwärtig. Von Continuous Testing, Continuous Deployment bis hin zu Continuous Delivery gibt es inzwischen mannigfaltige Ansätze, die nicht nur durch den gemeinsamen Zusatz „Continuous“, sondern auch eine gemeinsame Philosophie miteinander Hand in Hand gehen (Humble & Farley, 2010).

[2] https://www.adobe.com/de/products/xd.html

[3] https://sketchapp.com

Lean UX macht einen großen Schritt in Richtung „Kontinuität“, da es z. B. empfiehlt, riskante Annahme in Form von leichtgewichtigen Experimenten zu verifizieren. Das „System Under Test“[4] (SUT) wird dann sowohl kleiner und verarbeitbarer als auch günstiger und kann deshalb einfacher in volatile Projektpläne einsortiert werden. Nicht zuletzt aus der Idee heraus, jedes SUT klein zu halten, hat der Begriff des „Minimum Viable Products“ (MVP) durch Lean UX neue Popularität erlangt und ist plötzlich bei UX Designern und Software Engineers gleichermaßen in aller Munde.

2.2.1 Das nutzer-zentrierte MVP

Während der Begriff MVP unter UX Designern jedoch häufig als jegliche Art von Erzeugnis verstanden wird, welches dabei unterstützt, mehr Erkenntnisse über die Güte von Designentscheidungen zu erlangen, verstehen Software Engineers darunter häufig ein System, welches gerade so viel Funktionalität abbildet, dass man es im realen Umfeld als lauffähige Implementierung testen kann. Diese inflationäre und doppeldeutige Nutzung des MVP-Begriffes macht die Zusammenarbeit der Disziplinen aber nicht wirklich einfacher, zumal es bereits zwei Begriffe für diese beiden Anwendungsfälle gibt: „High-Fidelity Prototyp“ (Garrett, 2002) und „Walking Skeleton“[5].

Um Verwirrungen zu vermeiden und Erwartungen zu justieren, definiert Continuous UX den MVP Begriff im nutzer-zentrierten Kontext etwas schärfer: *„Ein nutzer-zentriertes Minimum Viable Product (MVP) ist ein begrenztes Set von Features, das ein einzelnes Kernbedürfnis einer einzigen Nutzergruppe in einem einzigen Nutzungskontext erfüllt.”*

Der Nutzungskontext bestimmt also darüber, was „minimal“ überhaupt bedeutet. Gilt es, eine minimale Mobile App zu entwickeln, die geschäfts-reisenden, vielbeschäftigten Hannover-Messe-Besuchern ermöglichen soll, mit ihrem eigenen Smartphone jederzeit Live-Informationen über ein ausgestelltes Roboter-Exponat zu erlangen, um so den Sinn und Zweck des Exponates in kurzer Zeit nachvollziehen zu können, dann kann „minimal“ an dieser Stelle nicht lauten: „Ein Walking Skeleton, umgesetzt als native App, welches erst einmal noch nicht ganz offline-fähig ist und in Bezug auf die Live-Datenladezeiten erstmal noch nicht Performance-optimiert ist.“

Ein solches MVP würde aus UX-Sicht grandios scheitern, denn kein Messebesucher dürfte sich in der Praxis eine native App herunterladen und bei extrem schlechten Netzbedingungen mehrere Versuche starten, bis endlich – viel zu spät – die ersten Daten unter Latenz eintrudeln. Auf der anderen Seite ist auch ein High-Fidelity Prototyp an dieser Stelle nicht sehr effektiv, da die riskantesten UX Design Entscheidungen nicht beim Navigations- oder Screendesign liegen, sondern auf technischen Einschränkungen beruhen.

[4] https://de.wikipedia.org/wiki/System_Under_Test

[5] http://wiki.c2.com/?WalkingSkeleton

Abbildung 1: Was „minimal" und „viable" an einem MVP ist, kann einzig und alleine aus dem Nutzungskontext abgeleitet werden.

Da ein nutzer-zentrierten MVP zwischen High-Fidelity Prototype und Walking Skeleton anzusiedeln ist, macht Continuous UX eine entsprechende Differenzierung beim Testing.

2.2.2 Concept Test

Ein Prototyp dient in erster Linie dem UX Designer. Getestet wird an dieser Stelle das Konzept oder die Visualisierung. Die Designvision wird beim Testen zwar durchaus effektiv geschärft, es ist aber auch herausfordernd, repräsentative Nutzer im repräsentativen Umfeld zu finden. Hantiert wird oft mit simulierten Daten, weshalb kritische Usability Probleme häufig übersehen werden. Diese Art von Test bezeichnet Continuous UX daher als „Concept Test".

2.2.3 Functional Test

Ein Walking Skeleton dient in erster Linie dem Software Verantwortlichen. Getestet wird, ob das System technisch einwandfrei funktioniert. Die User Experience dieses Produktes ist nicht im Fokus und darf erwartungsgemäß daher auch schlecht sein. Continuous UX verweist bei dieser Art von Test daher auf den „Functional Test"[6].

2.2.4 Product Test

Ein nutzer-zentriertes MVP dient einzig und alleine dem Nutzer. Wenn er oder sie mit dem MVP nicht zufriedengestellt wird, ist das Produkt auch nicht „viable". Es handelt sich bei einem nutzer-zentrierten MVP insofern also immer um ein „echtes" Produkt, dass es ein reales Nutzerbedürfnis in einem realen Nutzungskontext bedient und sich in diesem Zuge natürlich auch an einer guten UX messen muss.

Da ein nutzer-zentriertes MVP reale Bedürfnisse im tatsächlichen Nutzungskontext erfüllt, ist es entsprechend einfacher, echte (Pilot-)Nutzer für das Testing zu gewinnen. Sinnvollerweise sollte ein nutzer-zentriertes MVP daher bereits gleich von Anfang an Mechanismen zur

[6] https://en.wikipedia.org/wiki/Functional_testing

Aufnahme anonymisierter Nutzungsdaten enthalten – natürlich unter Einhaltung der geltenden Datenschutz-Regelungen.

Im Gegensatz zum Prototyp werden nutzer-zentrierte MVPs über die Zeit hin evolutionär weiterentwickelt und bleiben somit nicht dauerhaft „minimal“. Continuous UX spricht beim Testen von MVPs und späteren Ausbaustufen dieser MVPs daher allgemein von „Product Test“[7].

3 Die Evolution von MVPs

3.1 Skalierung

Mit dem Begriff der Kontinuität schwingt immer auch der Gedanke der evolutionären Weiterentwicklung und somit der Skalierung. Prototypen oder Experimente hingegen haben über den Erkenntnisgewinn hinaus in der Regel keine Zukunft in Bezug auf den kontinuierlichen Kontakt mit echten Nutzern.

3.1.1 Vertikale Skalierung

Ist eine kontrollierte Evolution eines größeren Systems gewünscht, so ist das nutzer-zentrierte MVP der ideale Startpunkt für eine „vertikale Skalierung“. Es bildet den Kern oder die „DNA“ dessen, warum ein Nutzer sich überhaupt mit einem Systemteil beschäftigen sollte. Da es sich auf die wesentlichen Features bei gleichzeitig kontext-angemessener UX beschränkt, können weitere Services und User Interface Teile über die Zeit iterativ hinzugenommen werden, ohne dass dies die UX zu früh kompromittiert. Zwar ist das Produkt dann irgendwann nicht mehr minimal, aber dennoch weiterhin lebensfähig.

3.1.2 Horizontale Skalierung

Da sich ein nutzer-zentriertes MVP am Startpunkt noch auf eine einzige Nutzergruppe in einem einzigen Nutzungskontext beschränkt, ist es in der Praxis gängig, mehrere MVPs parallel zu entwickeln, um letztlich auch der Breite der gesamten Nutzerbasis gerecht zu werden. Um bei dieser „horizontalen Skalierung“ Kosten zu sparen, führt Continuous UX die Methode der „Product Triangulation“ ein.

3.2 Product Triangulation

Ähnlich der groben Bestimmung eines Ortes per Triangulation[8], bestimmen die Teams bei der „Product Triangulation“ das ungefähre Zentrum des Gesamtsystems durch die parallele

[7] https://en.wikipedia.org/wiki/Product_testing

[8] https://de.wikipedia.org/wiki/Triangulation_(Geod%C3%A4sie)

Entwicklung dreier[9] möglichst orthogonaler nutzer-zentrierter MVPs. Orthogonal heißt in diesem Fall, dass sich die Nutzergruppen der einzelnen MVPs maximal unterscheiden (z. B. „Senioren“ vs. „Studenten“). Betrachtet man die Erkenntnisse und Entwicklungsartefakte dieser drei Einzelentwicklungen anschließend ganzheitlich, so kann der UX Designer daraus bereits erste Design System Patterns und der Software Engineer erste modulare, wiederverwendbare Komponenten ableiten, die für das angestrebte Gesamtsystem aller Voraussicht nach architekturrelevant sein werden.

Der Vorteil der parallelen MVP Entwicklung ist generell, das einzelne Abteilungen oder Sub-Teams unabhängig voneinander vorpreschen können ohne sich gegenseitig zu bremsen. Während der parallelen MVP Entwicklung ist somit für jedes Team „alles erlaubt“ und eine Abstimmung mit den anderen Teams nur lose und unverbindlich nötig. Dies erhöht das Entwicklungstempo und die Selbstwirksamkeit der einzelnen Teams.

3.3 System Consolidation

In Continuous UX folgt auf die Product Triangulation in regelmäßigen Abständen die „System Consolidation“, also die Konsolidierung der noch unzusammenhängenden Teile zu einem großen Gesamtsystem. Einfach gesprochen geht es darum, Gemeinsamkeiten, Unterschiede und Redundanzen der vorliegenden MVPs zu identifizieren, zu bewerten und zusammenzuführen. Hierfür kommen die einzelnen Product Owner, Produktmanager oder UX Manager zusammen und analysieren in einem gemeinsamen Workshop die dafür notwendigen Faktoren.

3.3.1 Unterschiedliche Lösungen für ähnliche User Needs

Zum einen werden *unterschiedliche* Lösungen mit *ähnlichem* zugrundeliegenden Nutzerbedürfnis gesammelt. Die Lösungen werden miteinander verglichen und auf Basis der jeweils zugrunde liegenden User Needs wird bewertet, ob eine der Lösungen auch das zugrunde liegende User Need der jeweils anderen Lösung bedienen oder zumindest nicht kompromittieren würde. Ist dies der Fall, so geht diese Lösung als „Sieger“ hervor und kann entsprechend als neue konsolidierte Lösung ins Gesamtsystem einfließen.

Beispielsweise ist es eine gute Idee, ein Ticketautomat für Touristen so zu konzipieren, dass diese direkt auf der Startseite des Automaten ein Ticket ziehen können, welches die Hauptverkehrsknotenpunkten wie Flughafen oder Hauptbahnhof inkludiert (und dies auch eingänglich visualisiert), denn das zugrundeliegende User Need beinhaltet in diesem Fall das Bedürfnis auch Ankunft und Rückreise gesichert zu haben. Für Geschäftsreisende wäre dieser Lösungsansatz ebenfalls von Vorteil und für Einheimische je nach Visualisierung nicht von Nachteil, weshalb die Lösung auch quasi unverändert ins Gesamtsystem einfließen kann.

[9] Natürlich können auch mehr als drei MVPs parallel entwickelt werden. Unserer Erfahrung nach bringt jedoch jedes zusätzliche MVP nicht genug Mehrwert, um die hierdurch anfallenden Kosten zu rechtfertigen, und jedes Weglassen eines MVP führt wiederum dazu, dass noch zu wenig Erkenntnisse und Artefakte vorliegen, um daraus risikoarm Design Patterns oder Komponenten ableiten zu können.

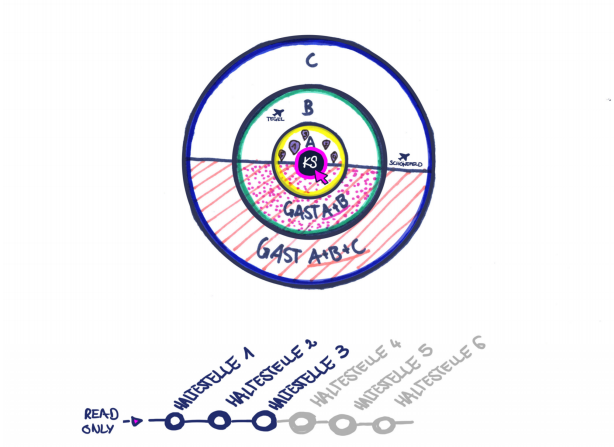

Abbildung 2: Beispiel einer Konzeptidee, welche als möglicher, ganzheitlicher Lösungsansatz für Touristen und Geschäftsreisende von Vorteil und für Einheimische nicht von Nachteil ist.

3.3.2 Unterschiedliche Lösungen für ungleiche aber vereinbare User Needs

Sind die User Needs im anderen Fall auf den ersten Blick unvereinbar, so bedeutet dies nicht zwangsläufig, dass die Lösungsansätze ebenfalls unvereinbar sind. Oft können Lösungsansätze miteinander gemischt werden („Concept Blending“).

Beispielsweise ist das User Need für einen klein gewachsenen Nutzer, dass ein Ticketautomat, möglichst tief hängt, damit die Bedienung motorisch angenehm bleibt. Ein groß gewachsener Nutzer möchte aus ähnlichem Grund, dass der Ticketautomat möglichst hoch hängt. Diese diametralen User Needs können im Lösungsraum konsolidiert werden, indem beispielsweise ein Hochkant-Display in zwei Bereiche unterteilt wird, wovon ein interaktiver Part, motorische Interaktionen erfordert und ein nicht-interaktiver Part, rein zu Informationszwecken dient und dann je nach Körpergröße des Nutzers die eine oder andere Darstellung gewählt wird.

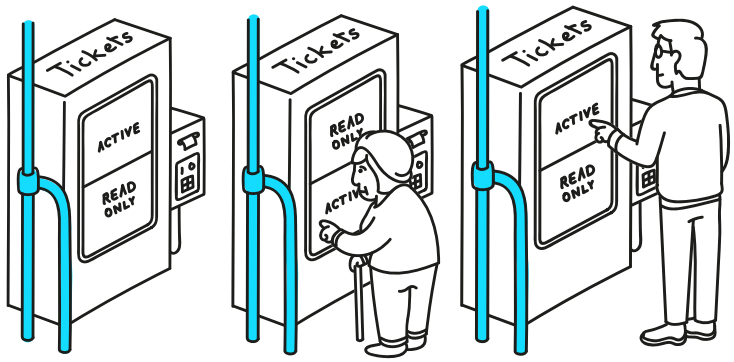

Abbildung 3: Beispiel für ein Lösungsergebnis, welches durch das Vereinen zweier diametraler Nutzeranforderung entstanden ist.

3.3.3 Unterschiedliche Lösungen für ungleiche und unvereinbare User Needs

Natürlich wird der Konsolidierungs-Workshop auch Lösungen zu Tage fördern, denen ungleiche Nutzerbedürfnisse zugrunde liegen, für die es *keine* gemeinsame Lösung gibt. Tritt dieser Fall für eines der drei MVPs besonders häufig auf, so ist es mit großer Wahrscheinlichkeit eine gute Idee, dieses MVP als eigenständiges Produkt weiterzuentwickeln. An den wenigen Stellen, wo man auch für dieses MVP Lösungen hätte harmonisieren können, wird dann ggf. ein Navigations- oder Anknüpfungspunkt zu einem der anderen MVPs gesetzt.

Beispielsweise macht es im Falle des Ticketautomaten nicht wirklich Sinn, dass Smartphone Nutzer ihre Tickets per Automat bezahlen, wenn das Ticket auch direkt in der App gekauft werden könnte. Andererseits könnte die Ticket App bei sehr niedrigem Akkustand des Smartphones den Nutzer benachrichtigen, dass er das bereits bezahlte Ticket per Automat in ein Papierticket umwandeln kann, indem er den Barcode vor die Kamera des Automaten hält. In diesem Beispiel macht es weder Sinn, die App Features im Automaten zu doppeln noch umgekehrt. Beide Produkte bleiben eigenständig aber kooperieren im Sinne des Nutzers miteinander.

4 User Stories

Wächst ein System unkontrolliert, so müssen die Teams naturgemäß mit allerhand Wachstumsschmerzen rechnen. Wachstumsschmerzen treten vor allem dann auf, wenn das Wachstum zu abrupt von statten geht. Abruptes Wachstum bedeutet nämlich zum einen zu langer Stillstand – hier schmerzt das Warten bis die Veränderung in Form des Wachstums endlich einsetzt. Gleichzeitig schmerzt aber auch die zu große Veränderung, *wenn* das Wachstum endlich eintritt, denn sie muss am Stück verarbeitet werden.

Hier lohnt es sich, die allgemeine Definition des Begriffes „Kontinuität“ herzuziehen, ist er doch allgemein definiert als *„eine Verbindung oder Entwicklungslinie ohne scharfe Unterbrechungen“*.

4.1.1 Continuous Integration

Moderne Software Engineers vermeiden scharfe Unterbrechungen in der Software Entwicklungspipeline mittels Continuous Integration, kurz „CI“ (Booch, 1991). Die Idee von CI ist, jede noch so kleine Code Änderung unmittelbar in das Gesamtsystem zu integrieren und im Verbund zu validieren. Dadurch, dass die Änderungseinheiten klein und die Validierungszyklen häufig sind, kann sich weniger technisches Risiko anhäufen. Es entsteht ein guter Durchfluss zwischen „Input“ (dem Code) und „Output“ (dem testbaren System).

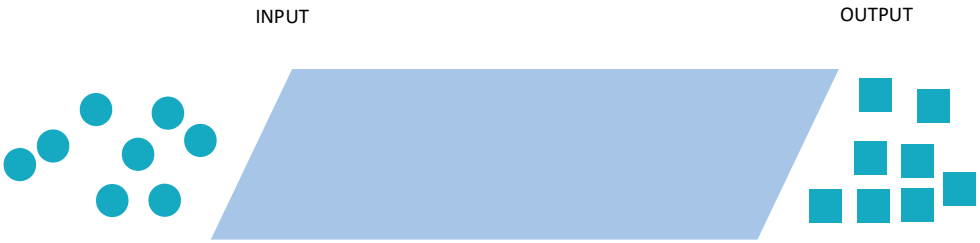

Abbildung 4: Kleine Änderungseinheiten als Input, resultieren in einem guten Durchfluss und somit kontinuierlichem Output in der Software-Entwicklungspipeline.

4.1.2 Kleinste Einheiten für UX Designer

Fragt man UX Designer nach den kleinsten Einheiten, für die sie Artefakte übergeben, so stellt man fest, dass dies zumeist einzelne Screens, Wireflows oder gar komplett durchgestaltete Klickdummies sind. Da bereits auf einem einzelnen Screen genügend implizites Spezifikationswissen versteckt sein kann, um mehrere Engineers für mehrere Sprints mit Arbeit zu versorgen, „verstopft“ an diesem Übergabepunkt oft die Software-Entwicklungspipeline: Zunächst warten Software Engineers lange auf erste implementierbare Konzepte, um dann, wenn diese Konzepte inklusive Spezifikation dann kommen, den „Informationspfropfen“ gar nicht wirklich implementierbar verarbeiten zu können.

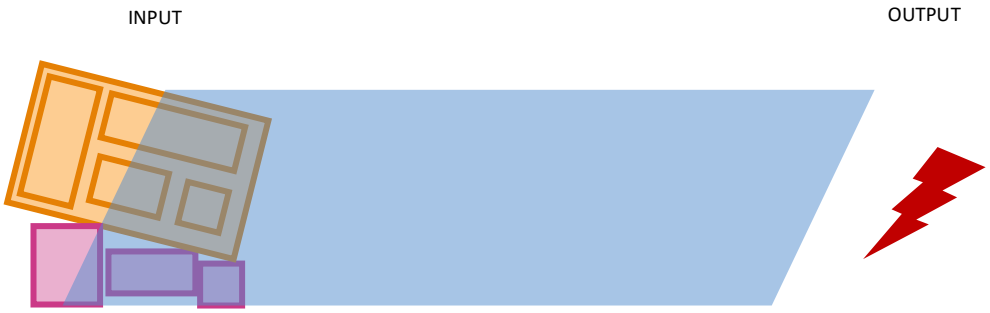

Abbildung 5: Eine „verstopfte“ Software-Entwicklungspipeline aufgrund zu schwergewichtiger Design-Artefakte.

Continuous UX greift als kleinste Organisations-Einheit daher einen „alten Bekannten“ aus der agilen Software Entwicklung auf: die User Story. Die User Story ist definiert als eine in Alltagssprache formulierte Softwareanforderung, die ein Nutzerziel und resultierenden Benefit aus der Anwendersicht beschreibt (Cohn, 2004).

Paradoxerweise ist die User Story trotz ihrer nutzer-zentrierten Kernidee bis heute nicht ganz bei der UX Community angekommen und als *„implementierbare Einheit“*[10] daher oft Hoheitsgebiet der Software Engineers. Allzu häufig werden aus bestehenden UX Konzepten (z. B. in Form von Wireflows) daher erst kurz vor der Implementierung User Stories abgeleitet, um anschließend Akzeptanzkriterien zu formulieren, Abschätzungen zu liefern und Sprintplanungen durchführen zu können.

4.1.3 Die „frühe“ User Story

Continuous UX definiert die User Story nicht länger als *„implementierbare Einheit“* (und damit auf rein technischem Hoheitsgebiet), sondern als *„entwickelbare Einheit“* (also als Disziplinen-übergreifendes Konstrukt). Konkreter sieht Continuous UX die User Story als die Formalisierung eines konkreten Nutzerbedürfnisses. Es ist ersichtlich, dass User Stories somit *weit vor* der Implementierung, sogar bereits vor der Konzeption entstehen müssen, um diesem Anspruch gerecht zu werden.

4.1.4 Bedürfnis-Trigger

Eine User Story sollte nach Continuous UX, anders als in vielen anderen Quellen beschrieben, aus mindestens vier, statt der häufig verwendeten drei, Komponenten bestehen. Neben der gängigen Rolle des *Aktors*, seines *Ziels* und des resultierenden *Nutzens*, empfiehlt Continuous UX daher, dass eine User Story auch explizit einen *Bedürfnis-Trigger* beinhaltet.

Beispielsweise möchten Nutzer in der Regel nicht zwei verschiedene Sprachen auf dem User Interface eines Ticketautomaten sehen. Im Falle, dass ein Einheimischer aber einem ausländischen Nutzer bei der Fahrkartenbuchung unterstützen möchte, wäre dies jedoch ein wichtiger Lösungsansatz. Die User Story inklusive Bedürfnis-Trigger müsste in diesem Fall somit korrekt lauten:

„Immer dann, wenn ich einen ausländischen Nutzer bei der Fahrkartenbuchung unterstütze, möchte ich als einheimischer Nutzer, dass Texte nicht nur in ausländischer Sprache angezeigt werden, sondern zudem auch in meiner eigenen Sprache, damit ich jederzeit weiß, wo wir beide uns im Buchungsprozess gerade befinden.“

Diese User Story gilt natürlich nur in dem Moment, in dem ein Nutzer auch tatsächlich Unterstützung anbieten möchte (sprich auch jemand auf Hilfe angewiesen ist), und mündet aufgrund dieser Konditionalität auch nicht zwangsläufig in einer auf dem UI omnipräsenten und in den meisten Fällen eher verwirrenden Doppel-Sprachlichkeit von Texten, sondern kann beispielsweise über einen entsprechenden „Fahrgast unterstützen“ Button nur bei Bedarf aktiviert werden.

4.1.5 Verbesserung des Durchsatzes via User Booklets

Durch die leichtgewichtige Natur von User Stories kann die Software-Entwicklungspipeline entlastet werden, da fertig konzipierte User Stories bereits an die Software Engineers übergeben werden können, während andere User Stories noch „reifen“ müssen. Übergabe-

[10] http://agiledictionary.com/277/user-story/

Lieferungen bestehen nicht mehr nur aus zusammenhängenden Screens oder Wireflows, sondern nur aus denjenigen Artefakten, die zur Erfüllung der einzelnen User Story notwendig sind.

Continuous UX spricht bei der Bündelung von UX Artefakten auf User Story Ebene von der Erstellung sogenannter „User Booklets“. Mittels User Booklets kann das UX Design Team in kleineren Schritten Designarbeiten abschätzen und liefern und somit dem kontinuierlichen Integrationsbedürfnis der Software Engineers entgegenkommen. Zudem steigt durch die kontinuierliche Abnahme und Verarbeitung dieser UX Artefakte durch die Engineers der Entwicklungsdurchsatz.

4.1.6 Backlog Priorisierung nach Erkenntnissen

Das Medium „User Booklets“ soll anders als das Medium „Story Card“ implizieren, dass ein User Booklet auch noch über die Implementierung hinaus von Wert ist. Das User Booklet wird nicht symbolträchtig „zerrissen“, sondern dient selbst nach der Implementierung noch als häufig genutzter Container, beispielsweise, um auch Erkenntnisse aus dem Testing als weiteres UX Artefakt dort abzulegen. Dies kann so weit gehen, dass – wie beim aktuellen BMBF-geförderten Forschungsprojekt Opti4Apps[11] erforscht – automatisiert erhobene, anonyme Nutzungsdaten auf User Story Ebene im entsprechenden User Booklet gesammelt werden, so dass der Product Owner oder Produktmanager die verschiedenen Stories auf Basis echter Nutzer-Erkenntnisse und Analysen im Backlog priorisieren kann.

5 Fazit und Ausblick

Continuous UX ist zum einen ein in sich geschlossenes UX Management Framework, mit dem agile Software-Entwicklungsprozesse mit mehr Fokus auf nutzer-zentrierte Resultate gesteuert werden können. Zum anderen ist Continuous UX jedoch auch ein Methoden-Baukasten mit dem, je nach Bedarf, existierende agile Prozess mit mehr Nutzer-Zentrierung angereichert werden können.

Kernbestandteile von Continuous UX sind zum aktuellen Zeitpunkt u. a. die scharf umrissene Definition des „nutzer-zentrierten MVPs“, die vertikale bzw. horizontale Skalierung auf Systemebene via „Product Triangulation“ und „System Consolidation“ sowie die frühe und kontinuierliche Sammlung von UX Artefakten auf User Story Ebene in sogenannten „User Booklets“.

Natürlich muss jeder Entwicklungsprozess entsprechend der Erfordernisse des Unternehmens individuell feinjustiert werden, weshalb es sich bei Continuous UX auch explizit um einen kohärenten Methoden-*Baukasten* und nicht um einen einzelnen, rigiden Prozess handelt. Um sowohl auf Detail- als auch auf Gesamtebene solide spielen zu können, müssen weiterhin viele zusätzliche Faktoren zusammenkommen: ein gutes Bedienkonzept, eine herausragende visuelle Gestaltung, ein moderner, agiler Software Entwicklungsprozess mit entsprechender

[11] http://www.opti4apps.de/

Kapazitätsplanung, ein flexibles User Interface Design System inklusive Living Styleguide sowie ein unternehmensweiter Qualitätssicherungsprozess.

Literaturverzeichnis

Auerbach, A. (2015): *Part of the Pipeline: Why Continuous Testing is Essential*. TechWell Insights August 2015.

Booch, G. (1991): *Object-oriented Design*. San Francisco: Benjamin/Cummings Publishing.

Cohn, M. (2004): *User Stories Applied: For Agile Software Development.* Amsterdam: Addison-Wesley.

Garrett, J.J. (2002): *The Elements of User Experience: User-Centered Design for the Web.* San Francisco: New Riders.

Gothelf, J. & Seiden J. (2012). *Lean UX: Applying lean principles to improve user experience.* Cambridge: O'Reilly and Associates.

Humble, J. & Farley, D. (2010). *Continuous Delivery.* London: Pearson Education.

Ohno, T. (1993): *Das Toyota-Produktionssystem.* Frankfurt am Main: Campus.

Pichler, R. (2007). *Scrum – Agiles Projektmanagement erfolgreich einsetzen.* Heidelberg: dpunkt.verlag.

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* New York: Currency.

Autor

Immich, Thomas

Thomas Immich ist Mitbegründer und Geschäftsführer der Centigrade GmbH und leitet dort den Bereich UX Services. Seit vielen Jahren beschäftigt er sich mit nutzer-zentrierten User Interface Design Methoden im Hinblick auf deren technische Umsetzbarkeit und Werkzeugunterstützung. Er betreute zahlreiche Kundenprojekte namhafter Unternehmen und berät in seiner Funktion als UX Manager und Trainer agile Softwareentwicklungsteams bezüglich der Optimierung von User Experience Prozessen. Er spricht außerdem regelmäßig auf einschlägigen Konferenzen und seine Arbeiten wurden bereits mit zahlreichen Auszeichnungen und Designpreisen prämiert.