Gruppenreflexion Team 2.A

Top 3 Erfolge

Aufrechterhalten einer harmonischen Arbeitsatmosphäre

Vermutlich das Wichtigste für eine erfolgreiche Entwicklung eines Produkts ist, dass sich alle beteiligten Entwickler gut verstehen und miteinander arbeiten können. In unserer Gruppe hat ein außerordentlich gutes Arbeitsklima geherrscht.

Begünstigt wurde unter anderem, dass bereits von Beginn an die wöchentlichen Meetings in Präsenz in der HTW stattgefunden haben. Im Gegensatz zu Plattformen wie Discord hat diese Form der Zusammenarbeit dazu geführt, dass wir mehr miteinander ins Gespräch kamen, uns besser kennenlernten und alle aktiv mitarbeiteten, weil das Feedback auf Aufgaben unmittelbarer und persönlicher erfolgte.

Ebenfalls (und vielleicht auch durch die analogen Meetings) war jeder motiviert, an der Entwicklung unserer Quiz-App beteiligt sein, und hat sich mit eigenen Ideen eingebracht. Dadurch konnten wir uns schnell auf einen groben Rahmen einigen, wie die App funktionieren soll und auf welchen architektonischen Strukturen sie gebaut wird. Dies hat besonders im Bezug auf die Realisierung von den ersten User Stories und -Tasks anfangs zu einem schnellen Fortschritt in der Entwicklung geführt.

Die breite Motivation zeigt sich auch dadurch, dass zugewiesene Tasks meist sehr schnell innerhalb der Iteration abgearbeitet wurden. So konnte der PO schnell die entwickelten Features ansehen und besser für die Vorstellung vor den Stakeholder vorbereiten, sowie sich über Weiterentwicklungs-Ideen Gedanken machen. Die Motivation führte sogar dazu, dass sich Teammitglieder freiwillig gemeldet haben, um mehr Tasks bearbeiten zu können. Dies hat sowohl den Fortschritt der Produkt-Entwicklung stark vorangebracht, als auch, die Arbeit für den Product Owner leichter gemacht. So hatte dieser weniger Tasks zu verteilen (da weniger zur Verfügung standen), und zusätzlich konnte dieser ein gut ausgearbeitetes Produkt den Stakeholdern vorweisen.

Dem Scrum Master das gute Arbeitsumfeld auch sehr viel Arbeit abgenommen. So hat die stetige allgemeine Motivation dafür gesorgt, dass sich alle in die Projektarbeit einbringen und eigenverantwortlich die Produktentwicklung mitgestalten. So musste der Scrum Master auch keine Konflikte zwischen Teammitgliedern lösen und konnte sich darauf konzentrieren, den allgemeinen Arbeitsprozess kontinuierlich zu optimieren.

Um den freundschaftlichen Umgang weiter zu fördern, hat unser Scrum Master auch einen gemeinsamen Billiard-Abend in den Sommerferien organisiert, wo wir uns noch besser kennlernen konnten und für noch mehr Motivation für die Arbeit im und für das Team gesorgt haben.

Drastische Änderung des Arbeitsprozesses hin zu besserer Umsetzung des Scrum-Prinzips

Wir haben uns wöchentlich getroffen, um über unseren Fortschritt zu reden, ähnlich zum Ablauf eines Weekly Stand-ups. Am Ende jeder Iteration gab es ein Sprint Review und anschließend eine Retrospektive immer mit dem gesamten Team.

Jedoch haben unsere "Stand-Up's" jede Woche 60-90 Minuten in Anspruch genommen, was überhaupt nicht mit der in der Vorlesung empfohlenen Menge von 15 Minuten übereinstimmt Dies hatte mehrere Gründe, daher ist dies auch als Misserfolg (siehe Schlechte Organisation der Weekly Meetings) gelistet und ausführlich beschrieben, da uns dies in der Entwicklung sehr eingeschränkt hat.

Dieses große Problem haben wir, wie im Misserfolg beschrieben, durch den Projektmanagement-Fachaustausch als solches erkannt. Und so haben wir in der darauf folgenden Iteration unseren Ablauf verändert: Nachdem unser Team neuen Input von den Stakeholdern in Form von User Stories erhalten hat, welche von einer Person während des Treffens protokolliert wurden, wurde unser Backlog, inklusive der neuen User Stories, im Team refinet. Dabei wurde die Komplexität der Items im Product Backlog geschätzt, sowie deren Priorität (stark beeinflusst durch den PO) festgelegt und im Status festgelegt, ob dieses Item 'Ready' ist, in der kommenden Iteration bearbeitet zu werden (also unsere 'Definition of Ready' erfüllt), oder noch im 'Backlog' verbleiben muss. Mithilfe dieser festgelegten 'Definition of Ready' kann dann der PO im Github-Board deutlich besser sehen, welche Aufgaben für die kommende Iteration zu erledigen sind. Nach diesem Review- und Refinement-Meeting erstellt der PO zahlreiche Tasks, abgeleitet aus den Items mit einer erfüllten 'Definition of Ready', sowie aus anderweitigen Anforderungen(z.B. Anwender-Dokumentation erstellen). So enstehen abhängig vom Umfang der Tasks ca. 3-6 Tasks pro Person pro Iteration.

In einem zusätzlich festgelegten Meeting über Discord, ein Tag nach dem Stakeholder-Treffen, werden die Tasks dann an das Team zugewiesen, damit es möglichst schnell wieder mit arbeiten beginnen kann. Dieses Treffen sollte nicht länger als 15 Minuten dauern.

Das Weekly Stand-up verläuft nun deutlich kürzer: Jeder im Team reihum spricht kurz an, was er bisher gemacht hat, und woran er demnächst arbeitet. Falls ein Problem besteht wird dies sehr kurz besprochen, oder es wird im Team eine Ansprechperson für die Problemlösung gefunden, an welche sich die Person wenden kann. Das einzige, was nun noch neu vergeben wird, ist die Bearbeitung von Bugs, falls sie eines dringenden Fixes bedürfen.

So konnten wir die Dauer der Meetings auf 15-30 Minuten einkürzen, womit wir sehr zufrieden sind. Auch sonst hat die Veränderung keine Probleme verursacht, sondern ist auf sehr positives Feedback im Team gestoßen, auch, da klar definiert ist, was jeder bis zum Ende der Iteration zu tun hat.

Wir finden, wir können Stolz mit uns sein, dass wir nach so vielen Wochen gemeinsamem arbeiten eingesehen haben, dass unsere bisherige Organisation nicht richtig funktioniert, und den Mut gefunden haben, diese komplett umzukrempeln. Ebenfalls, dass jeder im Team diesen Schritt mitgegangen ist und diese neue Meeting-Struktur mit einem zusätzlichen Meeting pro Iteration unterstützt hat, ist sehr lobenswert. Am Ende bleibt nur das Problem, das wir mit dieser neuen

Ordnung erst in unserer letzten Iteration anfangen konnten. So hätten wir vermutlich noch deutlich mehr schaffen können, wenn wir eher damit angefangen hätten.

Schnelle und frühe Organisation eines funktionierenden Code-Test-Systems

Womit wir zum Glück sehr früh angefangen haben, war das Beschäftigen mit sorgfältigem Code-Testing. So hat ein Teammitglied sich bereits nach dem ersten Praktikum, in welchem Tests behandelt wurde, dafür gemeldet, das Software-Testing übernehmen zu wollen. Da wir wussten, dass Software-Testing ein elementarer Bestandteil des Software Engineering ist, waren alle dafür, damit möglichst früh anzufangen. So hatten wir eine neue Rolle im Team: die Test-Verantwortliche.

Während andere Developer in unserem Team damit beschäftigt sind, Features zu implementieren, sorgt die Test-Verantwortliche dafür, dass der entwickelte Code getestet werden kann und somit wie von den Anforderungen gewünscht performt. Dafür entwickelt sie Integrations-und Unittests für den entwickelten Quellcode der Developer. Wenn ein Test Fehler ausgibt, hält die Test-Verantwortliche Rücksprache mit den Entwicklern des Codes, ob der Fehler auf der Seite der Tests oder doch im Quellcode liegt. Falls ein Fehler im Quellcode entdeckt wurde, wird dafür ein Bug-Issue auf Github erstellt. Die Testverantwortliche sorgt hauptsächlich für die Implementation von Unit-Tests und Integrationstests im Backend. Systemtests werden von allen Teammitgliedern beiläufig beim Durchklicken der Software, zum Beispiel durch Ansehen und Schnelltesten neuer Features, durchgeführt.

Da unsere Test-Verantwortliche über den gesamten Backend-Code verteilt arbeitet und testet, ist es nötig, dass die Code-Struktur des entwickelten Quellcodes in sich schlüssig und leicht verständlich ist, damit sie verstehen kann, was getestet werden muss. Somit werden ebenfalls die Entwickler dazu gedrängt, ihren Code gut zu dokumentieren und sinnvoll zu strukturieren bzw. zu modularisieren. Dies könnte auch hilfreich für die spätere Einbindung von neuen Backend-Entwicklern werden, falls dafür Bedarf besteht.

Das Wichtigste ist jedoch, dass nun sichergestellt wird, dass ein immer größerer Teil des Quellcodes korrekt funktioniert, wodurch sich die Entwickler mehr auf neue Produkt-Features konzentrieren können. Ebenfalls lässt sich beim Entwickeln von neuem Code nun schnell testen, dass der alte Code noch funktioniert, und nicht durch die neuen Eingaben Fehler auswirft.

Derzeit liegt die Testabdeckung bereits bei über 40% im Backend, womit wie sehr zufrieden sein können.

Gegen Ende unserer letzten Iteration ist es uns auch gelungen, die ersten Tests per Github Actions zu automatisieren, sodass beim Erstellen eines Pull Requests automatisch der Backend-Code getestet wird. So wird den Entwicklern in Zukunft noch mehr Arbeit abgenommen, die sie stattdessen in die Entwicklung von neuem Code stecken können.

Top 3 Misserfolge

Fehlende systematisierte Architektur-Dokumentation

Für unser Softwaresystem wurde ein deutlich größerer Teil des Codes im Backend als im Frontend entwickelt. Eine systematische Architektur- und Code-Dokumentation wäre daher frühzeitig notwendig gewesen, wurde jedoch erst sehr spät eingeführt.

So hat dort initial eine Person das Django-Framework errichtet und dann bereits einige neue Funktionen programmiert. Diese Person hat zwar die wichtisten Funktionen kommentiert, jedoch gab es immer noch einen hohen Einarbeitungsaufwand bei den dann neu hinzukommenden Backend-Entwicklern. Dies lag daran, dass zum einen das Django-Framework an sich ein großer 'Quellcode-Batzen' ist, der erstmal verstanden werden muss. Zum anderen hatten wir keine einheitichen Dokumentationsstandards, wie zum Beispiel Docstrings an jeder Funktion, da wir uns der Existenz dieser damals noch nicht bewusst waren, und, weil wir zu Beginn zu viel Wert darauf gelegt haben, einen vorzeigbares Softwareprodukt mit den entsprechenden Features zu entwickeln, wofür mehr Zeit in neuen Code investiert werden musste.

Dies hat dazu geführt, dass Entwickler, die erst später Code im Backend entwickeln sollten, sich nicht so recht getraut haben, diesen zu entwickeln. So wussten sie nicht genau, wo der benötigte Quellcode zu stehen hat bzw. wie viel davon bereits als wiederverwertbare Funktion einfach aufrufbar ist. So mussten die Entwickler sehr viel Zeit damit verbrauchen, zuerst den bestehenden undokumentierten Code zu verstehen, und kontaktierten dafür auch oft die Person, die den bestehenden Code entwickelt hat, um Ihnen beim Verstehen zu helfen. Oft haben die beiden Entwickler sich dann gemeinsam an den Code gesetzt, teilweise aber hatte dieser auch den Backend-Teil der Task vom unsicheren Entwickler komplett abgenommen, um den Zeitplan einhalten zu können. So ehrenwert es war, dass er die Aufgabe übernommen hatte, war dies jedoch langfristig nicht förderlich, da der unsichere neue Entwickler so nie gelernt hat, wie der Backend-Code funktioniert, und somit auch in Zukunft nicht in der Lage sein wird, diesen zu bearbeiten.

Diesem Problem hätte man vorbeugen können, wenn wir uns bereits früh auf Kommentier-Standards geeinigt hätten, was ebenfalls später in der Erstellung der Entwicklerdokumentation sehr hilfreich gewesen wäre. So wurde für diese Doxygen benutzt, wodurch die Einhaltung dieser Formatierungsstandards zwingend notwendig war, um eine übersichtliche Dokumentation zu erhalten.

Diese Kommentier-Standards zum Einen damals bereits ein bisschen mehr beim Verstehen des Code helfen können. Andererseits hätten wir auch bereits viel eher mit einer schnellen systematischen Erstellung einer Dokumentation des Codes beginnen können und sollen. Dafür spricht, dass die nun erstellte Entwicklerdoku laut unseren Entwicklern die Backend-Struktur sehr gut verdeutlicht und zu Beginn der Software-Entwicklung sehr hilfreich gewesen wäre, um den Code schnell zu verstehen. So hätten auch im späteren Prozess neue Backend-Entwickler keine große Hilfe von Entwicklern des bestehenden Codes benötigt, wenn wir eine solche Doku gehabt hätten. Dadurch kann dann neben der investierten Zeit der neuen Entwickler in das Verstehen des Code auch die investierte Zeit der bestehenden Entwickler in das Erklären des Codes verringert werden, auch wenn diese etwas mehr Zeit in die Dokumentation investieren müssen. Diese Zeitersparnis wird insbesondere langfristig mit dem neu hinzukommen jedes weiteren Backend-Entwicklers immer größer.

Dieses Problem mit der unzurechenden Dokumentation ist uns dann im Fachaustausch aufgefallen. So konnten wir erst spät (in der letzten Iteration) dieses Problem ausbessern, indem wir die

Entwicklung von Features deutlich zurückschraubten und dafür die fehlende Dokumentation und Kommentierung des bestehenden Codes nachgeholt haben, nach klar festgelegten Richtlinien. Ebenfalls wurde eine Person festgelegt, welche sich mit einem Dokumentationstool wie Doxygen beschäftigen sollte, damit so der entwicklten Code verständlich, auch mit Diagrammen, erklärt werden kann.

So haben wir gelernt, dass eine systematisierte Dokumentation genauso wichtig ist wie funktionierender Code.

Schlechte Organisation der Weekly Meetings

Ein Problem, was bereits früh aufgetreten ist und bis zur Veränderung unserer Meeting-Struktur durchgängig existent gewesen ist, war, dass die Stand-ups fast jedes Mal 60 - 90 Minuten in Anspruch genommen haben (wenn kein Sprint Review/Retrospektive oder ähnliches stattgefunden hat). Da wir uns gut verstanden haben, wurde dies in der Gruppe nicht als großes Problem angesehen, da die Zeit im regen Austausch schnell vergangen ist. Jedoch konnte man schon erkennen, dass etwas nicht ganz richtig lief, wenn das Meeting 4-6 Mal mehr Zeit in Anspruch genommen hat, als die ursprünglich in der Vorlesung empfohlene Menge von 15 Minuten.

So könnte doch bei Durchführen eines optimierten Stand-up allen beteiligten Teammitgliedern mindestens 60 Minuten pro Woche mehr Zeit für die Produktentwicklung zur Verfügung zustehen, wenn man die Stand-up-Dauer von 75-90 auf 15-30 Minuten verkürzt.

Der Grund für die Länge des Meetings lag daran, dass wir den Weekly Stand-up etwas falsch verstanden haben und nicht so durchgeführt haben, wie es eigentlich gemacht werden soll: So wurden für *jedes* Stand-up neue Aufgaben vom PO herausgesucht, die dieser aus gut passenden User Stories bzw. gefundenen Bugs hergeleitet hat. Falls Teammitglieder ihre Task zum Stand-up fertig gestellt haben, wurden ihnen von diesen erstellten Tasks neue zugewiesen, wobei noch einmal abgesprochen wurden, zu wem diese Aufgaben am Besten passen. Fall die Bearbeitung einer neuen User Story angefangen werden soll, wurde mit dem Team diskutiert, ob die vom PO herausgesuchte User Story vom Umfang her umsetzbar ist in der Task, bzw. welcher Developer dafür mitarbeiten müsste. Ein geordnetes Backlog Refinement der User Stories gab es bei uns nicht wirklich, die Priorität und Komplexität dieser wurde bei dieser Diskussion festgelegt. So wurde jede User Story einzeln in verschiedenen Stand-Ups refinet und die, welche es nicht in das Stand-Up geschafft haben, wurden gar nicht refinet, sodass auch eine vielleicht schnell umsetzbare User Story nicht bearbeitet wurde, wenn der PO diese nicht ausgewählt hat. Dies hat auch für eine unklare Struktur gesorgt, welche User Stories wie in der Iteration schaffen wollen, und wie viele wir zwischen den Sprint Reviews für die Stakeholder verwirklichen können

Da wir dennoch stetig Fortschritte in der Entwicklung von Features gemacht haben, schien uns die ineffiziente Bewertung von User Stories, sowie die lange Meeting-Zeit nicht als wichtiges Problem aufzufallen. Im Fachaustausch mit anderen Teams haben wir dann (endlich) herausgefunden, dass wir unsere Meeting-Organisation deutlich umstrukturieren müssen, wodurch wir dann um einiges effektiver Arbeit verteilen und erledigen können.

So hat unser Scrum Master den Ablauf unserer Meetings drastisch verändert, sodass unter anderem bereits am Anfang jeder Iteration im neu eingeführten Sprint Planning ein Backlog Refinement aller PBIs stattfand und im mächsten Meeting darauf basierend die Tasks verteilt

werden, wobei der PO diese vorbereiten muss. Dies hat unser Stand-Up - Zeit deutlich verbessert auf nur noch ca. 20-30 Minuten pro Meeting. Diese Erkenntnis, das etwas falsch läuft, und die Einarbeitung in die neu organisierte Struktur empfanden wir aufgrund der guten Funktonsweise als großen Erfolg unseres Arbeitsprozesses. Daher ist diese in [Drastische Änderung des Arbeitsprozesses hin zu mehr scrum-artigen SE-Methoden] noch einmal detaillierter aufgeführt.

Hier besteht allerdings immernoch Verbesserungsbedarf, um auf die empfohlenen 15 Minuten zu kommen. Daran müsste unser Scrum Master für die zukünftige Iteration noch arbeiten, wenn noch mehr Zeit für die Produktentwicklung wäre.

Stakeholder-Wünsche im Review zu wenig angefochten

Wir haben regelmäßig, am Ende jeder Iteration, ein Review mit den Stakeholdern durchgeführt. Dieses Treffen gab uns jedes Mal sehr konstruktiven und hilfreichen Input für die Entwicklung unseres Software-Produkts. Jedoch gab es oft, vor allem am Anfang, sehr viel Input. Dadurch haben wir nach dem Meeting meist darüber diskutiert, dass doch viel zu viel Entwicklungarbeit für manche Wunsch-Features zu investieren nötig ist, sie also praktisch in unserem kleinen Team nicht umsetzen ist. So haben wir viele User Stories gehabt, welche wohl nur sehr teilweise umgesetzt werden im gesamten überiterativen Entwicklungsprozess.

Zum einen hatten wir zu Beginn noch einen unausgereiften Product-Scope, durch welches wir noch nicht genau wussten, welche Features auch in unserem Plan stehen, noch zu implementieren, und welche Features überhaupt nicht zu unserer Software gehören sollten. Dies haben wir nach dem ersten Stakeholder-Treffen bereits erkannt, sodass wir dies ausbessern konnten und dann besser auf das nächste Stakeholder-Treffen vorbereitet waren.

Zum anderen jedoch wäre es Aufgabe des Product Owners gewesen, die Stakeholder-Wünsche einzugrenzen und auch mal 'Nein' zu den Feature-Wünschen zu sagen. Das wäre wichtig gewesen, um die zukünftigen Aufgaben des Teams schonmal vorab einzugrenzen, und es vor Überlastung zu schützen durch die übermäßige Menge an gewünschten Features, welche alle nicht wenig an Arbeitszeit benötigen. Jedoch war der PO dafür zu zögerlich, da die Stakeholder meist sehr überzeugt von ihren Wünschen waren und uns nicht klar war, wie weit wir ihre Wunsch-'Fantasie' beschränken können.

Nach Besprechungen mit anderen SE-Teams ist uns dann aufgefallen, dass wir auch die Stakeholder stoppen können und auch sollten, wenn sie zu viele Feature-Wünsche haben. Dies konnten wir dann in den letzten beiden Iterationen noch umsetzen, sodass der PO deutlich sagt, falls es genug Input ist, bzw. die Developer zu Wort kommen, falls sie erkennen, dass ein Feature zu viel Zeit in Anspruch nehmen wird. So konnten wir unser User-Story-Backlog in den letzten Iteration noch verkleinern, was auch für das Refinement wichtig geworden ist. So war es dort zum einen leichter zu entscheiden war, welche Priorität die User Storys haben, aber auch, wie sie zu schätzen sind, da dies mit weniger User Storys auch weniger Zeit in Anspruch nimmt.

Dennoch wäre es gut gewesen, die Stakeholder schon viel früher einzugrenzen, um weniger Zeit für die Diskussion über die Auswahl der User Stories zu investieren, damit das Team dann mehr Zeit hat, die wichtigen Features sorgfältig zu entwickeln