Woche 7

Backlog Refinement und Definition of Done

Funktionale Anforderungen, Qualitätsanforderungen und Rahmenbedingungen

1. Refinement des Backlogs

In dem ersten Sprint Review Meeting haben sie eventuell einiges über ihre Anforderungen gelernt. In weiteren Gesprächen mit ihrem Kunden haben Sie Details erfahren. Führen sie im Team eine Backlog-Refinement Sitzung durch und

- Ergänzen sie neue Anforderungen, wenn sich welche ergeben haben. Und schätzen sie diese.
- Streichen sie Anforderungen, die sich als überflüssig herausgestellt haben.
- **Ergänzen sie Akzeptanzkriterien**, wenn sie weitere Details zu den User Storys erfahren haben.
- Führen sie Links / Verweise ein, wenn sie zu einzelnen User Storys bzw. Anforderungen Papier Prototypen, Wireframes oder Testfälle erstellt haben.
- Möglicherweise können sie einige Anforderungen (des Konsolidierungssprints) in kleinere Anforderungen zerlegen. Führen sie wenn notwendig ein Slicing der User Storys/Anforderungen durch.

Verteilen Sie die Arbeit im Team. **Ihr Gitlab sollte am Ende des Projektes eine reichhaltige Informationsquelle sein**. Ergebnis dieser Aufgabe sollten Tickets in Gitlab mit vielen Details zu jeder ihrer User Story sein. Diese Tickets sind quasi lebende Dokumente und enthalten alle für die Anforderung relevanten Informationen. Wenn sie Werkzeuge wie z.B. Adobe XD verwenden, versuchen sie notfalls durch einfache Links hier eine Verbindung herzustellen.

Wenn sie Wireframes erstellt haben, passen sie auch diese an die aktuellen Erkenntnisse an. Nochmal: Verwenden sie sinnvolle und fachlich schlüssige Daten in ihren Wireframes und in den Beispielen, die sie verwenden.

2. Akzeptanzkriterien für Storys in Umsetzung

Gehen sie die Akzeptanzkriterien zu jeder Anforderung im laufenden Sprint noch mal durch. Ist das Kriterium schon so formuliert, dass sie daraus einen wichtigen Testfall machen könnten? Können sie das Akzeptanzkriterium im nächsten Sprint-Review-Meeting vorführen?

- Jeder / jede im Team ergänzt bei seinen / ihren übernommenen Anforderungen aus dem MVP fachlich sinnvolle Beispieldaten für die Demo bzw. Beispiele als Akzeptanzkriterien.
- Jeder / jede im Team ergänzt für seine /ihre Anforderungen die Akzeptanzkriterien mit dem GIVEN-WHEN-THEN Schema, wenn dies passt.

3. Definition of Done formulieren

Unser gemeinsames Ziel ist es, eine möglichst „gute“ Software an unseren Kunden zu liefern. Hierfür müssen wir verschiedene Maßnahmen ergreifen um das abzusichern. Einige der Anforderungen können wir in der **Definition-of-Done** dokumentieren. Was wünscht sich ihr Kunde im Bereich der Wartbarkeit? Haben sie Vorgaben erhalten, gibt es Coding-Standards, müssen sie eine bestimmte Testabdeckung erreichen? Gibt es Programmierstandards in ihrer Programmiersprache. All das gehört in ihre DoD, ebenso wie eine Policy, wie sie mit gemeldeten Problemen aus SonarQube umgehen.
Dokumentieren Sie ihre neue Defintion of Done in ihrem Wiki-System. Ändern sie ihren Buildprozess bzw. ihre Build-Pipeline und konfigurieren sie ihr Merge-Request-Verfahren (siehe Aufgabenteile 4 und 5)

4. Definition of Done umsetzen: Merge-Requests und Code Review

Ein Merge-Request ist eine Möglichkeit für Sie im Team, ein Review durch ein weiteres Teammitglied zu erzwingen. Diskutieren sie im Team, welche **Approval-Rules** für ihr Projekt passen würden. Ein Teammitglied

könnte den Code beispielsweise einfach „genehmigen“. Sie könnten einfordern, dass alle Anmerkungen des Teammitglieds eingearbeitet sind (Threads resolved).

Achten Sie bitte darauf, dass sie wirklich Code-Reviews durchführen. Wir prüfen dies am Ende der Vorlesung über die Review-Anmerkunen in den Merge Requests.

5. Definition of Done umsetzen: Testautomatisierung, Testcoverage und statische Code-Analyse

Überarbeiten sie ihre Build-Pipeline: Bauen sie SonarQube in den Buildprozess ein, wenn das nicht geht prüfen sie den Einsatz von SonarLint oder eines anderen Werkzeugs zur statischen Code-Analyse und zur Abschierung ihrer Codequalität.

Bauen sie spätestens jetzt automatisierte Unit-Tests ein, führen sie diese automatisiert im Build-Skript aus und messen sie die Testcoverage in der Pipeline.