Vorwort
Durch das Verfassen und Wiederholen dieses Skriptes wusste ich während der Präsentation genau, was ich zu jeder Folie sagen möchte. In der Prüfungssituation sollte frei gesprochen werden.

Slide 1
Sehr geehrte Damen und Herren,
ich begrüße Sie zu der Präsentation meiner Projektarbeit. Mein Name ist Konstantin Tieber und das Thema meines Projekts war die Analyse von Composer-Paketabhängigkeiten in PHP-Projekten.

Sollten Sie Fragen haben, beantworte ich diese gerne im Anschluss an die Präsentation.

Slide 2 Agenda
Nach einer Vorstellung des Projekts, gehe ich auf den Projektverlauf ein, der Analysephase, Entwurfsphase, Highlights aus der Implementierung, Maßnahmen zur Qualitätssicherung und Dokumentation, um schließlich ein Fazit zu ziehen.

Slide 3 Projektvorstellung
…

Slide 4 Projektumfeld
Mein Ausbildungsbetrieb war auch gleichzeitig der Auftraggeber für dieses Projekt.

Die webfactory GmbH hat aktuell 9 Mitarbeiter und entwickelt seit 20 Jahren Webanwendungen. Dabei setzen sie seit 2011 auf das PHP-Framework Symfony.
Inzwischen betreiben wir über 160 Kundenprojekte.

Slide 5 verschiedene Abhängigkeiten
Jedes Projekt setzt unterschiedliche externe Bibliotheken, Pakete oder Komponenten ein.

In dieser Grafik sehen Sie beispielsweise links verschiedene Projekte, die rechts mit ihren Paketabhängigkeiten verbunden sind.

Slide 6 Baton-Screenshot
Deshalb stellen sich unsere Entwickler oft die Frage „Welche Version des Symfony-Frameworks setzen wir für den Gemeinsamen Bundesausschuss ein?“ oder „Welche Projekte nutzen noch eine Symfony Version unter 2.2, welche vielleicht nicht mehr sicher ist?“.
Letztere Frage wird in dieser Grafik beantwortet.

Slide 7 Projektziel
Ziel meines Projekts war es, eine Anwendung zu entwickeln, die einfach zu bedienen ist und jederzeit einen aktuellen Überblick darüber geben kann,
… welche Abhängigkeiten ein Projekt hat
.. welche Projekte eine bestimmte Version eines Pakets einsetzen

Slide 8 Analyse
Welchen Weg soll ich gehen, um dieses Ziel zu erreichen?

Slide 9 Wirtschaftlichkeit
Neben einer Ressourcen- und Terminplanung, habe ich auch eine Konkurrenzanalyse eines ähnlichen Produkts gemacht, Gemnasium. Dieses ermöglicht die Erkennung von Sicherheitslücken in Paketabhängigkeiten von Projekten, aber es ist kostenpflichtig, limitiert die Anzahl an überwachten Projekten und kann nicht selbst gehostet werden.

Wie bei den meisten Projekten in einem wirtschaftlichen Unternehmen, sollte man die Amoritsationsdauer (Kosten/Rückfluss) berechnen. Zu Beginn fiel es mir schwer den jährlichen Rückfluss zu bestimmen, da ich keine realisitschen Zahlen hatte, die den Aufwand ohne mein Tool beschreiben könnten. Es schafft einen großen Mehrwert, indem es die Wartung und Absicherung von Projekten für Entwickler vereinfacht.
Und die unerwarteten Zwischenfälle in Produktion, die durch die Aktualisierung von angreifbaren Paketversionen verhindert werden können, rechtfertigen bereits die 1300€ Anschaffungskosten.
Eine weitere Anforderung war, dass der Quell-Code der Anwendung veröffentlicht werden soll,um von jedem verwendet und weiterentwickelt werden zu können. Hier schafft meine Projektarbeit zusätzlichen Wert, indem die webfactory GmbH etwas an die Open-Source zurückgibt.

Slide 10 Anwendungsfälle
Um einen Überblick über die zu implementierenden Funktionen zu bekommen, habe ich ein Use-Case-Diagramm für die Web-Anwendung erstellt. Es gibt zwei Akteure, den Entwickler und den Git-Server.
Entwickler wollen meistens alle Projekte finden, die ein spezielles Paket einbinden. Zusätzlich können sie aber auch die gesamten Paketabhängigkeiten eines Projekts in einer Liste einsehen und umgekehrt.
Der Git-Server benachrichtigt die Anwendung, wenn ein Projekt weiterentwickelt wurde und sich somit eventuell die Paketabhängigkeiten verändert haben.

Slide 11 Entwurf
Nachdem ich die Use-Cases definiert hatte, konnte ich in der Entwurfsphase damit beginnen, UI Mockups zu skizzieren.

Slide 12 UI Mockups
Hier sehen Sie die Mockups abgebildet.
Wie sollte das Formular aussehen, welche Felder gibt es? Wie strukturiere ich die Listen?

Slide 13 Datenmodell
In diesen Mockups erkennt man auch leicht die verschiedenen Entitäten, die ich hier in der Chen-Notation abgebildet habe.
Es gibt Projekte, Pakete und Paketversionen. Viele Projekte können viele Paketversionen verwenden und ein Paket hat viele Versionen.
Besonders der Unterschied zwischen dem Paket, das ein eher abstraktes Konstrukt ist und der Paketversion, welche die tatsächliche Abhängigkeit in Projekten darstellt, war für mich eine wichtige Erkenntnis beim Designen des Datenmodells.

Slide 14 Architektur
Bezüglich der Server-Architektur habe ich mich an der common Practice der webfactory orientiert. Die Anwendung wird in einer virtuellen Maschine in der Cloud deployed, wo hinter dem Apache HTTP Server eine Symfony Anwendung und eine MySQL Datenbank lebt. Symfony bringt viele Werkzeuge und Lösungen mit, die man für die Entwicklung moderner Web-Anwendungen benötigt und definiert auch eigene Best Practices, denen man folgen sollte. Eine klassiche Web-Anwendung mit Symfony ist nach dem MVC-Pattern strukturiert.

Slide 15 Projektimport
Damit die Informationen zu den Abhängigkeiten jederzeit aktuell sind, muss der Import der Projekte automatisch erfolgen.

Das läuft so ab:
Ein Entwickler pushed Änderungen ins Repository, der Git-Server benachrichtigt meine Anwendung, diese holt sich die Datei mit den Informationen zu Abhängigkeiten aus dem Repository, parsed die Inhalte und importiert das Projekt.

Das war die schwierigste Aufgabe, da viele Komponenten zusammenkommen und miteinander funktionieren müssen.

Slide 16 Implementierung
Aus diesem Grund bin ich diese auch als erstes angegangen…
Um die Logik zum Importieren auch unabhängig von der Benachrichtigung des Git-Servers testen zu können, habe ich Sie in einer Klasse gekapselt, die eine öffentliche Methode zur Importierung eines Projekts anhand der URL auf dem Git-Server zur Verfügung stellt. Diese konnte ich dann auch über die Konsole aufrufen.

Slide 17 Erster lauffähiger Import

So sah die erste lauffähige Version aus. Dieser Code ist schwer zu lesen und schwer zu testen, aber er hat mir dabei geholfen, die Aufgaben zu verstehen.

Slide 18 Eine Klasse für alles
Zu diesem Zeitpunkt lebte der Großteil der Logik des Projektimports in einer großen Klasse, wo ich erste erkannte Teilaufgaben lediglich in private Methoden extrahiert habe.

Slide 19 Refactoring
Mit einigen Refactorings habe ich die einzelnen Aufgaben in eigene Klassen verteilt. Die Logik in diesen Klassen kann ich nun bequem mit Unit Tests prüfen…

Slide 20 Code-Beispiel schön
und der Code war direkt sehr viel schlanker und lesbarer…

Slide 21 Kapselung
Ein weiteres Beispiel für Kapselung ist das Filtern von Abhängigkeiten in Projekten

Slide 22 Filtern von Projekten
Im Suchformular wählt der Entwickler erst das Paket aus, wählt dann den Operator und die Versionsummer, aus denen ich ein VersionConstraint Objekts baue und kann dann in der Paket-Entität nur jene Paketversionen holen, die zu diesem VersionConstraint passen. Dieser kapselt die Logik für den Vergleich der Versionsnummer in seiner matches() Methode. Dazu habe ich auch einen Screenshot, falls Sie diese später sehen möchten.

Slide 23 VersionConstraint Code-Beispiel
So sieht die matches Methode im VersionConstraint aus.

Nimmt PaketVersion entgegen und vergleicht dessen Version mit der Version aus dem Formular

Slide 24 Qualitätssicherung
Die zentralen Stellen im Code sind komplett durch Unit-Tests abgedeckt.

Slide 25 Testklasse
So sieht beispielsweise die Testklasse für den ComposerPackageFetcher aus. Unten sehen Sie eine Methode zum Mockem des Rückgabewertes des LockFileFetchers, der die Inhalte der Datei holt. Ich überprüfe, ob der ComposerPackageFetcher bei passenden Dateiinhalten einen Array von ComposerPackage Objekten zurückgibt und dass er eine Exception wirft, wenn keine ComposerPackages in den Inhalten gefunden werden können.

Diese grünen Balken bedeuten eine sehr hohe bis 100%ige Testabdeckung, was für eine zentrale Komponente wie den Projektimport sehr wichtig ist.

Slide 26 Continuous Integration
Jedes mal wenn ich pushe, werden die Tests zusammen mit einer statischen Code-Analyse automatisch auf einem Continuous Integration Server ausgeführt. So werde ich unweigerlich daran erinnert, wenn ich die Tests kaputt gemacht habe.

Slide 27 Dokumentation
Die Tests dienen Entwicklern zusätzlich als Dokumentation, da sie Auskunft darüber geben, wie sich der Code zu verhalten hat.

Slide 28 README
Da die Zielgruppe Entwickler sind, lebt die Dokumentation wie gewohnt in der README Datei im Repository. Hier wird beschrieben, wo man die Demo-Version aufrufen kann, wie man das Projekt selber lokal starten kann, welche Konfigurationsparamter existieren und es ist eine Übersicht über die Funktionen enthalten.

Slide 29 Docblocks
Für noch schnelleres Verständnis und Autovervollständigung durch die IDE sorgen Docblocks im Code, wie hier zu sehen im PackageProviderInterface.

Slide 30 Fazit
Das Projekt wurde erfolgreich abgeschlossen.
Bereits in den ersten zwei Wochen nach der Inbetriebname auf unserem Server erhielt ich sehr positives Feedback von meinen Entwicklerkollegen, die im Arbeitsalltag schnell hilfreiche Antworten zu Paketabhängigkeiten finden konnten.
Auf GitHub haben bereits andere Entwickler Verbesserungsvorschläge für den Code zur Verfügung gestellt, darunter auch den Code zum Starten einer lokalen Version in einem Docker Container.
Schließlich hatte ich im Mai die Gelegenheit, mein Abschlussprojekt auf der SymfonyLive Konferenz im PhantasiaLand vorzustellen, was wiederum viel Austausch mit anderen Entwicklern angeregt hat..

In der Zukunft, könnte man eine Schnittstelle anbinden, die ein Verzeichnis von Composer-Paketen führt, in denen Sicherheitslücken erkannt wurden, um die Enwtickler zu benachrichtigen, wenn ein solches Paket in einem Projekt verwendet wird.

Slide 31
Ich bedanke mich für Ihre Aufmerksamkeit und stehe Ihnen nun für Fragen zu Verfügung.