Entwicklung

- __Vorlagen-Seite
- Community-Workshop „Moodle Plugin-Entwicklung“
- Moodle Development
- Plugin-Entwicklung: Kollaboratives Anforderungsmanagement
    - Plugin-Entwicklung: Arten der Anforderungen
    - Plugin-Entwicklung: Anforderungssammlung
    - Plugin-Entwicklung: Anforderungsdokumentation
    - Plugin-Entwicklung: Datenschutzanforderungen
- Plugin-Entwicklung: Initiale / Kollaborative kontinuierliche Entwicklung
    - Plugin-Entwicklung: Zielsystem
    - Plugin-Entwicklung: Entwicklungsumgebung
    - Plugin-Entwicklung: Orte des Wissens zu Coding-Standards und APIs
    - Plugin-Entwicklung: Übersichten zu Standards
    - Plugin-Entwicklung: Form des Entwicklungsprozesses
    - Plugin-Entwicklung: Zeitaufwand
    - Plugin-Entwicklung: Kompatibilität Moodle-Versionen
    - Plugin-Entwicklung: Testen
    - Plugin-Entwicklung: Maintenance
    - Plugin-Entwicklung: Best practices Dokumentation

__Vorlagen-Seite

<Kurze Beschreibung für die Vorschau>

> Bitte diesen Hinweis vor Veröffentlichung rauslöschen!!
> Link zum Redaktionsleitfaden - bitte beachten!
>
> Tags setzen!
>
> Autorenangabe: Name mit Profil im Vereinsmoodle verlinken: https://forum.moodle-an-hochschulen.de/user/profile.php?id=XX
>
> Links überprüfen:
> https://addons.mozilla.org/en-US/firefox/addon/find-broken-links/
>
> Eine andere Seite einbinden: {{@XX}}
> Anzeigen der ID: Text in zu kopierendem Artikel markieren (Lesemodus, nicht beim editieren), es öffnet sich ein modal, dort links auf das Link-icon klicken, dann springt die Anzeige auf die Text-ID um.

Überschrift 1

Inhalt

Links / Quellen

<Links zu Ressourcen und Quellen>

Autor(inn)en: <Name, Hochschule>

Community-Workshop „Moodle Plugin-Entwicklung“

Im September 2022 führte Alexander Bias einen halbtägigen Workshop (von 9-13 Uhr) zum Thema Plugin-Entwicklung durch. Im Workshop vermittelt und diskutiert wurden die Grundlagen der Moodle Plugin-Entwicklung mit dem Ziel, mehr Entwickler-/innen an der Plugin-Entwicklung beteiligen zu können und eine gemeinsame Wissensbasis zu schaffen.

Workshop-Agenda

- Die verschiedenen Plugin-Typen in Moodle und deren zielgerichteter Einsatz
- Das local-Plugin als „Allzweckwaffe“
- Die allgemeine Ordner- & Dateistruktur eines Moodle Plugins
- Typische Dateien, die man in jedem Plugin findet, und ihr Zweck
- Definition und Nutzung von Admin Settings
- Definition und Nutzung von Capabilities
- Definition und Nutzung von eigenen Datenbanktabellen
- Installations- und Upgrade-Skripte von Plugins
- Libraries vs. Classes
- Ausgabe von Text und Inhalt in der GUI
- Sprachpakete
- Veröffentlichung eines Plugins auf Github
- Github Actions (ein erster Blick auf Moodle-Plugin-CI)

Abgerundet wurde diese Themenliste mit Verweisen auf existierende Entwickler-Dokumentationen, Foren und Chat-Channels sowie die wachsende Kursliste für Entwickler bei der Moodle Academy.

Dokumentation (9 Seiten PDF)

Gerne teilen wir hier die aufbereitete Dokumentation des Workshops.

Autor: Klaus Steitz, TU Darmstadt

Moodle Development

Bei der Entwicklung von Code für Moodle sind die Ressourcen vom Moodle HQ sehr hilfreich, zuvorderst sind hier das "Developer Resource centre" und die "Moodle Academy" (Developer learning pathway) zu nennen. Zudem bietet der Community-Workshop „Moodle Plugin-Entwicklung“ einen guten Überblick.

Getting Started

Als zentraler Einstieg mit Verweis auf viele weitere Themen, wie z.B. Developer FAQ und Moodle Debugging, eignet sich Getting Started auf moodledev.io. Empfohlen werden der Moodle PHP CodeSniffer zur Prüfung auf die Coding-Standards und PHPdoc check plugin zur Einhaltung der Dokumentationsstandards. Zudem wird zur Nutzung der Community ermutigt, z.B. das Developer Forum oder der Matrix-Kanal.

Lokales Moodle

Entwickelter Code, wie z.B. Plugins, sollten auf einem lokalen Moodle-System getestet werden. Die einfachste Möglichkeit Moodle lokal aufzusetzen ist es, sich den Windows Moodle Installer in der gewünschten Version herunterzuladen und dann "Start Moodle.exe" auszuführen. Im Anschluss ist ein betriebsbereites Moodle mit MySQL, PHP und den Moodle-files unter localhost bereitgestellt. Dies hat den Vorteil, dass der Code lokal per IDE bearbeitet werden kann und nicht auf einem Server liegt. Die PHP Error-logs liegen dann unter "server/apache/logs/error.log".

Die andere Möglichkeit ist es Docker zu benutzen. Hierfür kann man sich auf moodle-docker auf github ein fertiges Set an Containern herunterladen. Darüber ist es auch möglich, mehrere verschiedene Moodle-Versionen parallel zu betreiben.
Ein weiterer Vorteil: **Behat- und PHPUnit-Tests** kann man mit diesem Docker-Setup sehr einfach laufen lassen.
Für Windows nutzt man aus Performancegründen am besten das WSL (Windows Subsystem für Linux).

Die Academy bietet zudem einen Kurs "Set up your Moodle Development Environment".

Plugin-Typen

Plugin-Entwicklung ist die gängigste Entwicklung in der Community. Hier ist zuerst zu entscheiden, was für ein Plugin entwickelt werden soll. Davon ist abhängig, wie das Plugin aufgebaut werden muss. Admin-Tools unterscheiden sich z.B. grundlegend von Blöcken oder Activity-Plugins. Eine Auflistung aller Plugin-Typen findet sich unter Plugin types (auf moodledev.io). Fast jedes Moodle-Plugin benutzt die gleichen, standardisierten Dateitypen und -namen. So wird z.B. in der "version.php" die aktuelle Version des Plugins gespeichert. Durch eine Erhöhung der Version kann zum Beispiel ein Datenbank-Update forciert werden.

Die Moodle Docs bieten auch eine Seite zum Start in die Moodle-Entwicklung:
https://docs.moodle.org/dev/Tutorial

Plugin-Entwicklungsprinzipien

- Moodle-APIs statt eigener Lösungen verwenden
- Capabilities definieren
- Sprachstrings nutzen
- Upgrade-Skripte verwenden

Plugin-Struktur

Beispielhafter Aufbau eines Plugins, siehe auch Kapitel 5 Community-Workshop "Moodle Plugin-Entwicklung":
"Typische Dateien, die man in jedem Plugin findet, und ihr Zweck".

> local_myplugin/
> ├── version.php
> ├── settings.php
> ├── db/
> │ ├── access.php
> │ ├── install.xml
> │ └── events.php
> ├── classes/
> ├── lang/
> └── lib.php

AG Plugins

Die AG Plugins ist ein guter Ansprechpartner und sollte bei Plugin-Entwicklungen konsultiert werden, vor allem hinsichtlich der Möglichkeiten der "Plugin-Werkstatt" (siehe auch Vereins-Strategie, Kaptiel "2.2 Plug-ins").

Coding-Standards

Besonders für Beiträge zum Moodle-Core ist es wichtig, sich an die Coding styles zu halten. Aber auch für Plugin-Entwicklungen ist ein einheitlicher, gut lesbarer und wartbarer Code, der die Zusammenarbeit und Code-Reviews erleichtert, hilfreich. Grundlage bilden die PHP-Standards PSR-1 und PSR-12. Die Richtlinien behandeln Konsistenz, Lesbarkeit, Namenskonventionen, Sicherheit, Automatische Prüfung (Code checker).

Coding style (auf moodledev.io)

Codesniffer

Der Codesniffer analysiert den PHP Code auf Probleme und Kompatibilität mit den Moodle Coding Guidelines.
Manche Fehler kann er auch automatisch beheben.

https://moodledev.io/general/development/tools/phpcs

phpcs mit Composer installieren

```
composer global config minimum-stability dev
```

```
composer global require moodlehq/moodle-cs
```

phpcs für ein Plugin laufen lassen

```
~/.composer/vendor/bin/phpcs /path/to/moodle/mod/foobar/
```

Auto-fix durchführen

```
~/.composer/vendor/bin/phpcbf /path/to/moodle/mod/foobar/
```

ESLint

ESLint ist das Äquivalent für Javascript zu Codesniffer für PHP.

https://moodledev.io/general/development/tools/nodejs#running-grunt

https://docs.moodle.org/dev/Linting#Javascript_(ESLint)

ESLint für in einem Plugin Verzeichnis laufen lassen

```
grunt eslint --show-lint-warnings
```

Auto-fix durchführen

```
grunt eslint --fix
```

API Guide

Moodle bietet viele Programmierschnittstellen (APIs). Diese können bei Entwicklungen z.B. für Standardaufgaben genutzt werden und sind nachhaltiger und kompatibler als eigenen Code zu entwickeln. Häufig verwendete APIs sind die **Access API** (Rechte und Rollen), **Data Manipulation API (DML)** (Datenbankzugriffe), **File API** (Dateiverwaltung), **Form API** (Formulare), **Events API** (Ereignisse) und **External Functions API** (Webservices). Die Nutzung von Events, Hooks und weiteren Core-APIs verbessert die Wartbarkeit.

API Guides (auf moodledev.io)

Moodle Database Schema

Einen Überblick über das Moodle Database Schema inklusive der Beziehungen zwischen den Tabellen bietet

https://www.examulator.com/er/output/index.html.

Security

Durch Implementierung von Schutzmaßnahmen sollten Benutzer:innen, Daten und das Moodle-System vor unbefugtem Zugriff, Datenverlust und Angriffen geschützt werden. Zentrale Elemente hierbei sind Berechtigungen, Prüfung von Nutzereingaben, SQL-Injection verhindern durch Nutzung der Moodle Data Manipulation API (DML), Cross-Site-Scripting (XSS) vermeiden, Ändernde Aktionen absichern, Sensible Informationen schützen, Risiken von Capabilities dokumentieren. Sicherheit sollte fester Bestandteil des Entwicklungsprozesses sein.

Security (auf moodledev.io)

Testing

Durch manuelle und automatische Tests sollen frühzeitig Fehler erkannt und die Stabilität über mehrere Versionen hinweg gesichert werden. Bereits während der Entwicklung sollten Tests geschrieben und regelmäßig ausgeführt werden. Moodle unterstützt u. a. **PHPUnit** für Unit-Tests und **Behat** für Akzeptanztests der Benutzeroberfläche.

Continuous Integration (CI)

Moodle Plugin CI ist das von Moodle empfohlene Werkzeug, um Plugins automatisiert zu prüfen. Es bündelt die wichtigsten Tests und Qualitätsprüfungen und lässt sich in CI-Systeme wie **GitHub Actions** integrieren. Dadurch werden Änderungen bei jedem Commit oder Pull Request automatisch validiert und Fehler frühzeitig erkannt. U.a. **PHPUnit**, **Behat**, **Moodle Code Checker**, **PHPDoc Check**, **Mustache Linting** und **PHP Linting** können automatisiert ausgeführt werden.

Moodle an Hochschulen Plugin Guidelines

Maintaining: https://github.com/moodle-an-hochschulen/moodle-plugin-maintaining/wiki

Readme-Vorlage: https://github.com/moodle-an-hochschulen/moodle-readmetemplate

---

Autor: Klaus Steitz, Technische Universität Darmstadt

Plugin-Entwicklung: Kollaboratives Anforderungsmanagement

Dies ist ein Auszug aus dem Leitfaden 'Plugin-Entwicklung' der Plugin-Werkstatt von Moodle an Hochschulen e.V., der nach und nach in die Knowledge Base übertragen wird.

Plugin-Entwicklung: Arten der Anforderungen

Was können Anforderungen im Sinne der Softwareentwicklung an mein Plugin sein?

Die präzise Definition von Anforderungen ist essentiell, um den Funktionsumfang eines Plugins einzugrenzen und zielgerichtete Entwicklung zu ermöglichen. Dabei wird bewusst auf technische Lösungsvorschläge verzichtet. Der Fokus liegt auf der Beschreibung, für wen das Plugin gedacht ist und was es letztlich leisten soll. Dazu lassen sich die Anforderungen nach ISO/IEC/IEEE 29148 wie folgt kategorisieren:

- **Funktionale Anforderungen**: Welche Aktionen können Nutzer*innen und Administrator*innen durchführen? Dazu zählen beispielsweise das Verwalten von Benutzerrollen und Rechten, das Erstellen, Bearbeiten und Löschen von Inhalten, Such- und Filterfunktionen, das Anlegen und Verwenden von Templates, Benachrichtigungen und Statusänderungen sowie Funktionen, die die Zusammenarbeit unterstützen, wie die gleichzeitige Bearbeitung und die Verfügbarkeit einer Versionshistorie.
- **Qualitätsanforderungen (Nicht-funktionale Anforderungen)**: Welche messbaren Qualitätskriterien sollen für das Plugin gelten (vgl. Kriterienliste kuratierte Liste)? Hierzu gehören Aspekte wie automatisierte Tests (PHP Unit Tests, Behat Tests), Performance, Skalierbarkeit, Unterstützung von Import/Export und LifeCycle, Usability, App-Unterstützung, responsives Design, Sicherheit und Stabilität, Dokumentation für Nutzende.
- **Rahmenbedingungen**: Woran muss sich das Plugin halten? Dazu zählen didaktische ebenso wie organisatorische, technische und rechtliche Vorgaben wie IT-Governance, Budgetgrenzen, feste Release-Termine, Systemkompatibilität, Barrierefreiheit, vorgesehene Schnittstellen, DSGVO-Konformität und Aufbewahrungsfristen für Daten.

Zur weiteren Eingrenzung des Plugin-Umfangs wird empfohlen, die MoSCoW-Priorisierung mit den Kategorien **Must have**, **Should have**, **Could have** und **Won’t have** zu verwenden. Insbesondere **Won’t have** dient als wichtiges Instrument zur klaren Scope-Eingrenzung und zum Erwartungsmanagement, indem bereits in der Konzeptionsphase bestimmte Anforderungen als *out of scope* definiert werden.

Um die Anforderungen systematisch auf Basis der ISO/IEC/IEEE 29148 zu erheben und anschließend mittels MoSCoW zu priorisieren, kommen geeignete Methoden zum Einsatz, die im nachfolgenden Kapitel näher erläutert werden.

Autor:innen: Stefan Dabrock (Moodle an Hochschulen e.V.), Philipp Dorok (Ruhr-Universität Bochum), Denise Materna-Stuttkewitz (TU Dortmund)

Plugin-Entwicklung: Anforderungssammlung

Wie sammle ich Anforderungen?

Anforderungen an das geplante Plugin können mit Hilfe der folgenden Methoden gesammelt werden:

Anforderungs-Workshop

Nutzt frühzeitig die Gelegenheit, die Akteure einzubeziehen, die mit dem Plugin arbeiten werden. Je nach Plugin-Idee können das Moodle-Administrator*innen und auch die Nutzer*innen des LMS sein. Ein entsprechender Workshop, bei dem diese Akteure ihre Bedürfnisse äußern können, bietet sich an, um die verschiedenen Perspektiven zu dokumentieren. Damit kann dann die Planung auf die Anforderungen der Akteure zugeschnitten werden.

Interviews & Clusterung

Eine gute Alternative zum Workshop ist es, mit den oben genannten Akteuren Interviews zu führen und so die Anforderungen an das Plugin zu identifizieren. Die Informationen aus den Interviews können dann zu Informationsclustern visuell aufbereitet werden.

Bestehende Lösungsmöglichkeiten analysieren und ggf. Lücken identifizieren

Ein wichtiger erster Schritt besteht stets darin die Plugins zu sichten, die es bereits gibt und sich mit der gleichen Problemlage befassen. So kann das Projektteam herausfinden, ob ein bestehendes Plugin in Kooperation mit den jeweiligen Maintainer*innen weiter entwickelt werden kann oder ob es sinnvoll ist, ein vollständig neues Plugin zu entwickeln. So können ggf. Synergien unter dem Stichwort „Das Rad braucht nicht neu erfunden werden“ genutzt werden.

Geeignete Quellen für die Recherche sind:

- Moodle Marketplace: Nachfolge des Plugin Directory
- Community Archive of Moodle Plugins (CAMP): Unabhängiges, Community-getragenes Plugin-Archiv als Alternative zum Moodle Marketplace
- Kurs Pluginwerkstatt im Vereins-Moodle

- Forum Aktiv genutzte und geplante Moodle-Plugins im Kurs „Moodle-Community“ im Vereins-Moodle
- Bereich Plugins in der Knowledge Base des Vereins
- Beispielanalyse Existing Solutions für den Quiz Archiver von Niels Gandraß

Workflow-Diagramm des zu lösenden Problems

Ein solches Diagramm bietet sich bei Plugins an, mit denen ein mehrschrittiger Arbeitsprozess durchgeführt wird. Ein gutes Beispiel für einen solchen Prozess ist die Buchung einer E-Prüfung, für die von der ersten Einreichung bis zum bestätigten Termin sowie der Prüfungsdurchführung eine Vielzahl miteinander verzahnter Schritte durch unterschiedliche Akteure (Buchende*r, Prüfende*r, Service-Team etc.) durchlaufen werden. Wenn im Vorfeld ein Diagramm mit allen Einzelschritten der einzelnen Akteure und ihrer Kommunikation entwickelt wird, dann hilft das dabei, die Anforderungen an das Plugin zu erkennen.

User Stories

User Stories sind ein gutes Mittel, um sich in die Lage der einzelnen Akteure zu versetzen, die mit dem fertigen Plugin arbeiten werden. Für jede*n Akteur*in – bei einem Buchungsprozess einer Prüfung beispielsweise die Personen des Serviceteams – kann aufgeschrieben werden, welche Aktionen die jeweiligen Personen durchführen können sollen und welche Erwartungen sie an Funktionen des Plugins haben. Hier geht es beispielsweise um die einzelnen Rechte (Schreib-, Lese- oder Administrationsrechte) der jeweiligen Akteure.

- Beispiel von User Stories für den Quiz Archiver von Niels Gandraß
- Beispiel für die Schritte der Anforderungsermittlung beim Quiz Archiver von Niels Gandraß (gilt bis Schritt „Formulation of User Stories“)

Autor:innen: Stefan Dabrock (Moodle an Hochschulen e.V.), Philipp Dorok (Ruhr-Universität Bochum), Denise Materna-Stuttkewitz (TU Dortmund)

Plugin-Entwicklung: Anforderungsdokumentation

Wie dokumentiere ich meine Anforderungen so, dass Entwickler*innen / Dienstleister*innen sie verstehen?

Die schriftlich dokumentierten User Stories bilden ein zentrales Element, um die Anforderungen an ein geplantes Plugin innerhalb der Projektgruppe festzuhalten. Gleichzeitig können sie auch externen Entwicklungsteams zur Verfügung gestellt werden, um diesen zu verdeutlichen, welche Ausgestaltung das Plugin haben soll. Da während des Entwicklungsprozesses neue Anforderungen auftauchen sowie gleichzeitig zuvor formulierte Aspekte der User Stories nicht mehr dem Bedarf entsprechen können, ist es sinnvoll die User Stories laufend zu prüfen und zu überarbeiten.

Parallel zur laufenden Überprüfung der User Stories ist es sinnvoll, die mit den unterschiedlichen Methoden eine Frage weiter oben gewonnenen Anforderungen in Issues zu übersetzen, also klar formulierten einzelnen Entwicklungsschritten, die zusammen das Gesamtbild des Plugins darstellen. Die Issues können dann in einem Planningboard im Kanban-Stil aufbereitet werden, wo sie entsprechend ihres jeweiligen Arbeitsstandes (Ausformulierte Überlegung, In Entwicklung, In der Testphase, etc.) die Position wechseln.

Dabei verweisen die im Planningboard nur grundsätzlich formulierten Issues per Link auf ihre ausführliche Darstellung in einem Github zum Plugin. Das Planningboard bietet die Möglichkeit, die einzelnen Issues konkreten Entwickler*innen zur Bearbeitung zuzuweisen.
Zentral sind noch sogenannte Mockups der einzelnen Funktionen, also bildliche Darstellungen der jeweiligen Nutzungs- oder Administrationsoberfläche wie sie im fertigen Plugin aussehen sollen. Sie sind in gewisser Weise ein Endpunkt der Anforderungen an das Plugin.

Sowohl bei einer rein internen Entwicklung und in jedem Fall bei einer Vergabe an externe Entwicklungsteams ist es sinnvoll, den Status Quo als Lastenheft zu dokumentieren. Es enthält eine grundsätzliche Beschreibung der Plugin-Funktionen und der Akteure, die damit arbeiten werden. Basierend auf den Issues wird zudem alles, was mit dem Plugin durchführbar sowie administrationsseitig einstellbar sein soll, mit Hilfe einer an Entwicklungsteams gerichteten Beschreibung formuliert. Dazu gehören auch die zuvor einzeln erstellten Mockups zur eindeutigen bildlichen Dokumentation. Auch wenn sich in einem dynamischen Prozess noch Änderungen am Plugin ergeben können, ist das Lastenheft die vorläufige Abschlussdokumentation, um den Menschen, die das Plugin entwickeln werden, einen klaren Auftrag zu geben.

Da die Entwicklung an einem Plugin grundsätzlich eine Änderungsdynamik besitzt, werden Issues, User Stories, Mockups sowie das Planningboard vom Entwicklungsteam laufend überprüft und angepasst.
Grundsätzliche schematische Darstellungen wie Organigramme oder Diagramme, die beispielsweise Prozesse und Rollen im Zusammenhang mit der Nutzung eines Plugins verdeutlichen, können ebenfalls eine sinnvolle Ergänzung sein.

Autor:innen: Stefan Dabrock (Moodle an Hochschulen e.V.), Philipp Dorok (Ruhr-Universität Bochum), Denise Materna-Stuttkewitz (TU Dortmund)

Plugin-Entwicklung: Datenschutzanforderungen

Habe ich Datenschutzanforderungen?

Grundsätzlich muss ein Plugin die Anforderungen erfüllen, die sich aus der Datenschutz-Grundverordnung (DSGVO) ergeben. Hierzu gehört, dass die Privacy API implementiert ist. Diese ermöglicht es, bei Auskunfts- oder Löschanfragen nach DSGVO die angefragten personenbezogenen Daten in menschenlesbarer Form zu exportieren oder diese zu löschen. Die Implementierung der Privacy API ist zudem eine Voraussetzung für eine Veröffentlichung des Plugins im Moodle Marketplace.

Gemäß dem Prinzip der Transparenz muss dokumentiert werden, welche Daten erzeugt, gespeichert oder übertragen werden. Dabei sind die Grundsätze der zweckgebundenen Datenverarbeitung und der Datenminimierung stets einzuhalten:

- Verbindungsdaten wie IP-Adresse, Browsertyp, Datum und Uhrzeit des Zugriffs
- Zugangsdaten wie Benutzername, Passwort in verschlüsselter Form, E-Mailadresse, Vorname, Nachname, …
- Inhaltsdaten wie hochgeladene Dateien, Forenbeiträge, Texte, …
- Prüfungsdaten im Rahmen von Prüfungen und Tests

Werden die Daten nicht allein lokal verarbeitet, sondern übertragen, ist der Standort der Datenverarbeitung zu dokumentieren, z.B. innerhalb oder außerhalb der EU.

Autor:innen: Stefan Dabrock (Moodle an Hochschulen e.V.), Philipp Dorok (Ruhr-Universität Bochum), Denise Materna-Stuttkewitz (TU Dortmund)

Plugin-Entwicklung: Initiale / Kollaborative kontinuierliche Entwicklung

Dies ist ein Auszug aus dem Leitfaden 'Plugin-Entwicklung' der Plugin-Werkstatt von Moodle an Hochschulen e.V., der nach und nach in die Knowledge Base übertragen wird.

Plugin-Entwicklung: Zielsystem

Auf welchen Zielsystemen (z.B. Moodle-Version, PHP-Version) soll das Plugin laufen?

Das Moodle HQ pflegt eine Übersicht über vergangene und zukünftige Releases. Wenn ein Plugin neu entwickelt wird, ist es lohnenswert, für alle aktuell unterstützen Moodle Versionen zu entwickeln. Dabei sollte auch ein Augenmerk auf die LTS Releases gelegt werden (deren Anforderungen möglicherweise bereits veraltete PHP Versionen beinhalten). Die Moodle Versionen wiederum stellen Anforderungen an PHP und Datenbanken. Das Plugin sollte auf jeden Fall MySQL/MariaDB sowie Postgres in der entsprechenden Version unterstützen.

Die meisten Moodle-Installationen laufen auf einer Linux-Variante. Es ist daher empfohlen ein Linux zu verwenden, wo der Code getestet wird. Gleichwohl wird auch Windows offiziell unterstützt. Entwickler, die auf Windows entwickeln, können Moodle in Docker-Containern oder in einer VM mit Linux laufen lassen.

Autor: Stephan Robotta, Berner Fachhochschule

Plugin-Entwicklung: Entwicklungsumgebung

Wo sollte ich mein Plugin entwickeln (lokal, Testserver, ..)?

Um ein Moodle-Plugin zu entwickeln, ist es am einfachsten eine Entwicklungsumgebung für Moodle (also eine lokale Moodle-Installation) auf dem eigenen Rechner einzurichten.

Hierzu gibt es 2 grundsätzliche Wege:

- Moodle MDK (nur MacOS und Linux)
- Docker (Windows mit WSL, MacOS und Linux)

Vom Moodle HQ werden beide Wege mit entsprechenden Repositories unterstützt. Welchen Weg man davon geht, hängt von den eigenen Vorlieben und Erfahrungen sowie dem eigenen Betriebssystem ab.

Zum Testen werden nach der Installation des Moodles in der Entwicklungsumgebung am besten ein Kurs sowie zwei Nutzerkonten angelegt. Ein User bekommt die Rolle „Teacher“, ein anderer die Rolle „Student“. Der Admin-Account ist durch die Installation bereits vorhanden.

Autor: Stephan Robotta, Berner Fachhochschule

Plugin-Entwicklung: Orte des Wissens zu Coding-Standards und APIs

Wo finde ich Infos zu Coding-Standards und APIs?

Erste Anlaufadresse für Coding-Standards und Dokumentation ist moodledev.io. Gleichzeitig ist es sinnvoll, in bestehenden Code in Moodle selbst oder auch in Plugins zu schauen, um eventuell eine Lösung für das eigene Problem zu finden.

Auf der Seite Support Channels sind einige Möglichkeiten (Forum, Matrix Chat) gelistet, wie man mit anderen Moodle-Entwicklern in Kontakt treten kann, um Fragen zu stellen und Erfahrungen auszutauschen.

Moodle bietet eine Plugin CI an, die für die Continuous Integration eines Plugins genutzt werden kann. Darin sind Checks und Tests enthalten, die den Code auf Konsistenz und Regeln prüfen. Für Plugin-Entwickler ist es nicht zwingend vorgeschrieben, sich an diese Regeln zu halten. Es wird aber empfohlen den Regeln zu folgen, um für Außenstehende eine nachvollziehbare Qualität des Plugins zu liefern und zu demonstrieren.

Autor: Stephan Robotta, Berner Fachhochschule

Plugin-Entwicklung: Übersichten zu Standards

Welche Standards muss ich beachten (z.B. Coding Guidelines, Datenschutz, Barrierefreiheit)?

Überblick **Moodle Policies**

Informationen zu den **Moodle Coding Guidelines**

Ein einheitlicher Programmierstil ist in jedem Entwicklungsprojekt wichtig, insbesondere wenn viele Entwickler beteiligt sind. Ein standardisierter Stil trägt dazu bei, dass der Code leichter lesbar und verständlich ist, was die Gesamtqualität verbessert.

Informationen zu den **Moodle Accessibility Guidelines**

Grundsätzlich gilt, dass alle Komponenten von Moodle für alle Benutzer verfügbar sein müssen. Barrierefreiheit muss bei der Entwicklung jeder neuen Funktion in Moodle berücksichtigt werden. Moodle strebt die Umsetzung von WCAG 2.2 Level AA an. Die Einhaltung dieses Standards wird in regelmäßigen Audits extern anhand ausgewählter Beispielseiten zertifiziert. In Deutschland ist allerdings die strengere BITV 2.0 maßgeblich.

Informationen zur **Moodle Privacy API**

Die Privacy API ermöglicht es, die von Moodle gespeicherten personenbezogenen Daten zu beschreiben und bietet die Möglichkeit, diese Daten nutzerbezogen zu ermitteln, zu exportieren und zu löschen. Dies gewährleistet die Einhaltung von Vorschriften wie der Datenschutz-Grundverordnung (DSGVO) in der EU.

Wenn ein Plugin im Moodle Marketplace veröffentlicht werden soll, müssen diese Standards eingehalten werden. Das ist in den Regeln zur Plugin Contribution ausformuliert.

Die Ziele einer „sauberen“ Programmierung im Sinne dieser Standards sind:

- Reduzierung der Fehleranfälligkeit
- Bessere Wartbarkeit
- Bessere Weiterentwickelbarkeit

Autorin: Melanie Treitinger, Ruhr-Universität Bochum

Plugin-Entwicklung: Form des Entwicklungsprozesses

Wie sollte der Entwicklungsprozess aussehen (Top-Down vs. Agil)?

Wenn ein Plugin neu entwickelt wird, gibt es in der Regel eine Anforderung, welche mit aktuellen Mitteln nicht erfüllt werden kann. Es ist empfohlen, diese Anforderung in User Stories zu formulieren, um dann einzelne Tasks bzw. Projekt-Teile aus den User Stories zu erstellen. [Siehe auch Plugin-Entwicklung: Anforderungssammlung]

Beim Start ist es unwichtig, ob eine agile oder eine Top-Down-Strategie zum Entwickeln angewendet wird. Die agile Methode ist sicher moderner und lässt sich u.U. mit gängigen Projektmanagement-Tools besser abbilden.

Es ist sinnvoll, den Entwicklungsprozess in einzelne Teilprojekte mit Tasks zu zerlegen und diese auch entsprechend, z.B. in Github-Issues, auszuformulieren. Auf diese Weise ist der Prozess auch für Außenstehende nachvollziehbar und man kann bspw. Entwicklerkapazität auch aus der Community beziehen. [Siehe auch Plugin-Entwicklung: Anforderungsdokumentation]

Autor: Stephan Robotta, Berner Fachhochschule

Plugin-Entwicklung: Zeitaufwand

Wie viel Zeit benötige ich zum Entwickeln?

Die Antwort auf diese Frage hängt von folgenden Faktoren ab:

- Umfang und Komplexität der Anforderungen
- Umfang und Detailliertheit des Lastenhefts (User Stories, Mockups)
- Erfahrungsstand der entwickelnden Person(en) mit Moodle

Grundsätzlich sollte man bei der Entwicklung von Software immer einen zeitlichen Puffer einbauen, empfohlen werden meist 20%.

Der Zeitrahmen für die Fertigstellung hängt dementsprechend von diesen Faktoren ab.

Autorin: Melanie Treitinger, Ruhr-Universität Bochum

Plugin-Entwicklung: Kompatibilität Moodle-Versionen

Wie kann ich die Kompatibilität mit unterschiedlichen Moodle-Versionen sicherstellen (Moodle Plugin-CI)?

Unterschiedliche Moodle-Versionen bedeuten in der Regel auch unterschiedliche PHP-Versionen sowie Datenbank-Versionen, für die das Plugin funktionsfähig sein sollte.

Um die Kompatibilität sicherzustellen, bieten sich automatisierte Tests an. Moodle stellt dafür eine eigene Konfigurationsdatei für Github Actions bereit. Hierzu existiert auch eine passende Anleitung zur Nutzung von Github Actions.

Autorin: Melanie Treitinger, Ruhr-Universität Bochum

Plugin-Entwicklung: Testen

Wie teste ich mein Plugin?

Bei der Entwicklung sollte man PHPUnit-Tests und Behat-Tests als automatisierte Tests mit implementieren.

Daneben sind manuelle Tests nach vorher definierten Szenarien („wenn ich Einstellung X aktiviere sollten Lehrende Funktion Y nutzen können“) sinnvoll.

Autorin: Melanie Treitinger, Ruhr-Universität Bochum

Plugin-Entwicklung: Maintenance

Wie kann ich eine Maintenance sicherstellen?

Ein Plugin sollte mindestens nach jeder Veröffentlichung einer neuen Moodle Major-Version auf Kompatibilität geprüft und angepasst werden.

Je nach Umfang und Komplexität des Plugins sollte ein entsprechendes Budget dafür vorgesehen werden. Der Dienstleister kann dafür auch ein Angebot machen.

Autorin: Melanie Treitinger, Ruhr-Universität Bochum

Plugin-Entwicklung: Best practices Dokumentation

Was sind Best Practices für Nutzer*innen- und Entwickler*innendokumentation?

Nutzer*innendokumentation (User Documentation)

Hier steht die Problembehebung und Anwendung im Vordergrund. Nutzer*innen wollen wissen, wie sie mit der Software ein bestimmtes Ziel erreichen. Der Aufbau kann entweder rollenbasiert oder aufgabenbasiert oder eine Kombination aus beidem sein.

Best Practice Beispiel: Niels Gandraß – Archiving Subsystem for Moodle (Usage)

Entwickler*innendokumentation (Developer Documentation)

Hier steht die Erweiterbarkeit, Integration und Wartung im Vordergrund.

Die Systemarchitektur, das zugrundeliegende Datenmodell sowie eventuelle Abhängigkeiten sollten erklärt werden. APIs sollten nach anerkannten Standards dokumentiert werden (z.B. OpenAPI). Code-Beispiele erleichtern die Erweiterbarkeit.

Best Practice Beispiel: Niels Gandraß – Archiving Subsystem for Moodle (Developer Documentation)

Autorin: Melanie Treitinger, Ruhr-Universität Bochum