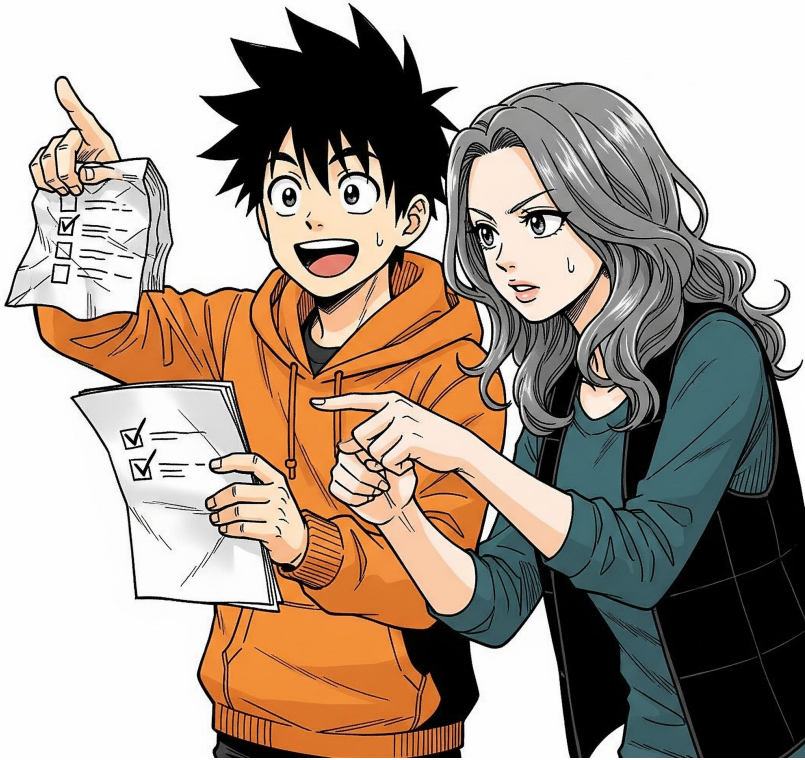
Software Engineering Best Practices

Bachelor-Praktikum and Agiles Software-Engineering-Projekt

CQSE | Sven Amann

Checklist



- [ ] Ein Team bilden
- [ ] Ticketsystem nutzen
- [ ] Versionskontrolle einsetzen
- [ ] (Code) Peer Reviews machen
- [ ] Automatisiert testen
- [ ] Continuous Integration leben
- [ ] Experimentiert auch mit Dingen, die für das BP vielleicht unnötig sind

Ein Team bilden

Ein funktionierendes Team kann mit minimalem Tooling gute Software entwickeln. Umgekehrt, wenn das Team nicht funktioniert, ist das Projekt nicht zu retten.

- Lernt euch persönlich kennen.
- Trefft euch wöchentlich, am besten vor Ort.
- Sprecht Probleme an, auch persönliche bzw. zwischenmenschliche – es wird nicht besser, wenn ihr darüber schweigt. Ansprechen von Problemen ist etwas Positives und sollte NIE verurteilt werden.

Ticketsystem nutzen

JIRA hat mit die größte Verbreitung, für das BP tut es aber auch ein simpleres System, wie GitHub Issues oder Gitlab Issues. Was ihr braucht, sind Tickets mit einer Beschreibung, einer zuständigen Person und einem Status (z.B. Idea, Planned, In Progress, In Review, Done).

- Nutzt ein gemeinsames Ticketsystem für alle Aufgaben im Projekt.
- Startet mit den Defaults, verliert euch nicht in Konfiguration.
- Haltet das Ticketsystem aktuell.

Versionskontrolle einsetzen

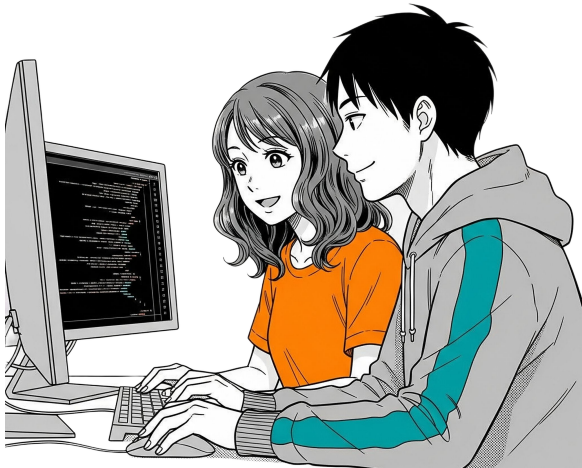
Versionkontrolle ist unverzichtbar, um gemeinsam Software zu entwickeln. `git` ist in den meisten Bereichen heute der de facto Standard.

- Nutzt ein Versionskontrollsystem, ggf. in Kombination mit einer Code-Collaboration-Plattform wie Gitlab, GitHub, Bitbucket oder Azure DevOps.
- Legt auch Dokumente in die Versionskontrolle.
- Fügt Ticket Ids aus eurem Ticketsystem in die Commit-Messages ein (`#42 fixed bug`)
- Probiert Feature-Branches aus.

(Code) Peer Reviews machen

Vier Augen sehen mehr als zwei. Peer Reviews steigern die Qualität, sorgen für Wissensverteilung und ermöglichen so, dass jeder Verantwortung übernehmen kann.

- Gebt jedes Arbeitsprodukt einer anderen Person zum Review.
- Jeder macht Reviews und gibt die eigenen Arbeitsprodukte in Review.
- Beginnt jedes Review mit einem positiven Kommentar.
- Reviewt das Arbeitsprodukt, nicht die Person dahinter.
- Pair/Mob Progamming sind eine Form von Reviews.
- Nutzt Pull/Merge Requests in einer Code-Collaboration-Plattform (GitHub, Gitlab, …) für asynchrone Reviews.



Automatisiert testen

Ein Softwaretest wird einmal geschrieben, aber 1000x ausgeführt. Automatisierung macht das möglich. Andernfalls gehen immer irgendwann Dinge kaputt, die schon mal funktioniert haben (Regression).

- Nutzt eins der gängigen Testautomatisierungsframe-works zur Technologie eurer Wahl.
- Versucht euch an Test-Driven-Development.
- Führt die Tests lokal aus UND in eurer Continuous Integration.

Continuous Integration leben

»Was nicht automatisch geprüft wird, geht kaputt«, John Penix (Google). Und es gibt nichts blöderes, als das erst zu merken, wenn keine Zeit mehr bleibt…

- Nutzt ein Build-Management-Tool (z.B. Gradle, Maven, …).
- Richtet ein CI ein, über Gitlab Pipelines, GitHub Actions, Azure DevOps Pipelines, Bitbucket Pipelines, …:
- Baut eure Software bis zum Auslieferungsartefakt.
- Nutzt statische Codeanalyse (z.B. mit Teamscale).
- Führt eure Tests aus.

Experimentiert auch mit Dingen, die für das BP vielleicht unnötig sind

Das Bachelor-Praktikum (genauso wie das SEP) soll euch helfen, erste Erfahrung mit Softwareentwicklung zu sammeln. Das Projekt ist viel kleiner als ein reales Projekt (weniger Leute, weniger Code, weniger Anforderungen, weniger Druck, …).

- Nutzt diese Freiheiten, um eure Fähigkeiten, verschiedene Technologien und Tools auszuprobieren.
- Habt Spaß 😉

Nutzt Teamscale

Wenn ihr Lust habt, nutzt Teamscale, um eure Softwarequalität zu erhöhen. Z.B. für statische Codeanalyse, Architekturanalyse oder Test-Gap-Analyse.

Ich stelle euch gerne eine kostenfreie Lizenz aus.

Schreibt mir!

Sven <amann@cqse.eu>
#GernePerDu

PS: Wir suchen auch Werkstudis für Entwicklung, Beratung, …!