

Rückblick auf den DH Code Review Day 2025

Paul Bayer

29.06.2026

Rückblick auf den Digital Humanities Code Review Day 2025

Beim *Digital Humanities Code Review Day* am 5. Dezember 2025 haben Studierende aus verschiedenen Berliner Universitäten Gastgeber war dieses Mal das IZ D2MCM.

Teilnehmende aus verschiedenen Berliner Universitäten lernten darüber, wie die Praktiken des *Code Review* und des *Pair Programming* im Bereich der Digital Humanities eingesetzt werden können. Dabei unterstützte sie das Organisationsteam bestehend aus Lisa Poggel (FU), Viktor Illmer (FU) und Paul Bayer (Masterstudent an der HU). Den inhaltlichen Input gab der Informatiker Linus Vevers (FU), der in seinem Dissertationsprojekt an Pair-Programming-Praktiken in der Industrie forscht.

Zur Theorie des Code Review und Pair Programming

Zu Anfang seines Inputs postulierte Linus Vevers, dass eine zentrale Herausforderung der Informatik der Umgang mit Komplexität sei. Softwareprojekte werden komplexer, je mehr Stakeholder involviert sind, woraus meist viele, verschiedene und zum Teil auch widersprüchliche Anforderungen erwachsen. Hinzu kommt der Komplexitätsanstieg durch wachsende Codebases und größer werdende Entwicklungsteams.

Diese Charakteristiken der allgemeinen Informatik, wie sie zum Beispiel in der freien Wirtschaft praktiziert wird, stellte Vevers dann den gegensätzlichen Dynamiken innerhalb der Digital Humanities gegenüber. An DH-Projekten sind meist nur wenige Menschen beteiligt. Im Vergleich zu anderen Domänen bleiben die Anforderungen überschaubar und die Codebasis bleibt meist klein bis mittelgroß. Das hat Konsequenzen: Die interne Qualität, also lesbare, gut dokumentierter und gut strukturierter Code, ist weniger kritisch¹, standardisierte Prozesse sind kaum nötig und der Wissenstransfer über die Codebasis ist irrelevant, solange nur eine Person an dieser arbeitet.

Code Reviews – Klassischer und moderner Ansatz

Die Anfänge der Praxis des Code Review liegen in den 1970er und 1980er Jahren. Damals war die Norm noch die aufwändige Inspektion vollständiger Codebases mithilfe langer Checklisten. Meist im Rahmen von Merge Requests via GitHub oder GitLab sind Code Reviews in heutiger Zeit dagegen meist direkt in den laufenden Entwicklungsprozess eingebettet und sie umfassen nicht mehr das gesamte Repository, sondern im Vergleich dazu viel kleinere Änderungen, die idealerweise ein klar begrenztes Feature oder einen klar definierten Arbeitsschritt repräsentieren. Der entscheidende

¹Kritisch bedeutet hier: notwendig für den Erfolg. Gute Programmierpraktiken in den Digital Humanities sind selbstverständlich ebenfalls wertvoll, aber oft reicht es aus, dass der Code tut, was er soll.

Unterschied zwischen klassischen und modernen Code Reviews ist also der Umfang der jeweils betrachteten Code-Chunks.

Diese Dichotomie des klassischen und modernen Ansatzes ist für die Methodik in den Digital Humanities ein nützliches Begriffsgerüst: Sind bei einem Projekt mehrere Forschende bzw. Programmierende involviert, dann sind moderne Code Reviews via Merge Requests bei GitLab, GitHub, etc., der wirksamste Hebel für die Steigerung der Code-Qualität. Wird ein Projekt jedoch von einer Soloentwicklerin durchgeführt, dann sind jedoch die Checklisten aus dem klassischen Ansatz besonders gut geeignet.

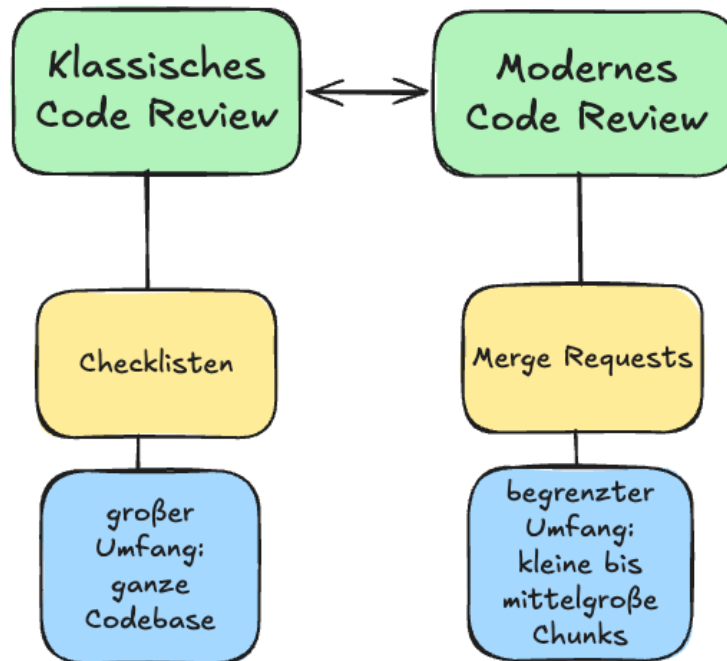


Abbildung 1: © Paul Bayer 2026 (CC BY 4.0)

Pair Programming – Code Review in Aktion

An den Input zum Code Review knüpfte daraufhin der Input zum Pair Programming an. Paarprogrammierungssessions sind extreme Code Reviews und können geplant oder spontan sein. Es handelt sich dabei nicht darum, dass eine Person programmiert und die andere zusieht, sondern dass beide Menschen gemeinsam coden und dabei gleichzeitig auch designen, analysieren und testen. Paarprogrammierung ist ein Grundpfeiler des agilen Arbeitens.

Vevers stellte als Ziele der Paarprogrammierung als ein Kontinuum zwischen den folgenden drei Polen dar:

1. **Qualitätssteigerung**
2. **Wissenstransfer**
3. **Verbesserung der Geschwindigkeit**

In diesem tripolaren Spektrum verortete er dann konkrete Formen der Paarprogrammierung:

- Das **Paar-Review** hat vor allem die Qualitätssteigerung zum Ziel

- Die **Paar-Entscheidungsfindung** kann u.A. zur Verbesserung der Geschwindigkeit beitragen
- Als weiterer Mehrwert der Paarprogrammierung lassen sich in Form der sogenannten **Paar-Courage** zudem Aufgaben angehen, die für eine einzelne Person zu komplex oder anspruchsvoll wären, alleine durchzuführen.
- Beispiele für Formen, die den Wissenstransfer fördern sind das **Gemeinsame Lernen**, sowie auch das **Paar-Debugging**, denn um einen Fehler zu lösen bedarf es oft eines tiefen Verständnis der zugrundeliegenden Funktionsweisen des Programms.

Ein zentrales Konzept, das Vevers hierbei ins Spiel brachte ist die **Togetherness**. Sie ist das fundamentale Erfolgskriterium in der Paar-Programmierung und erwächst aus einem sauber geteiltem Arbeitsbereich, einem gemeinsamen Verständnis der Codebase, einem gemeinsamen Verständnis der Softwareentwicklung und einem abgestimmten Vorgehensplan. In Hinsicht auch die Fachsprache entsteht bessere Togetherness da, wo in Hinsicht auf die Begriffe und Konzepte keine Sprachbarriere zwischen den beiden Paarprogrammierenden besteht.

Der Togetherness stehen oft die folgenden Dynamiken im Wege:

Disengaging behavior	Angst oder Scham, Fragen zu stellen das kann dazu führen, dass man den Anschluss verliert
Hierarchical behavior	Kontrolle ansich reißen, den Partner ignorieren, herabwürdigen, kann den Partner aktiv aus der Sitzung drängen
Defensive behavior	Unzufriedenheit führt zu Passive-Aggressiveness, Sabotage
Unabgestimmte und gegensätzliche Prozessziele	Z.B. <i>Qualität und Wissenstransfer</i> vs. <i>Geschwindigkeit</i>

Was hilft gegen diese Dynamiken?

Proactive Equalizing Behavior	Schaffen eines Wir-Gefühls. Als Team gegen das Problem
Reactive Equalizing Behavior	Meta-Kommunikation bei Unzufriedenheit, Partner:in zurück ins Boot holen
Prozessziele gemeinsam abstimmen	Warum bearbeiten wir die Aufgabe als Paar? Was ist uns in welchem Umfang wichtig?
Eigenes Verhalten beobachten	<i>Hierarchical Behavior</i> , <i>Disengaging Behavior</i> und <i>Defensive Behavior</i> vermeiden und ggf. gegensteuern.

Aufgrund von Ausgangsbedingungen oder dem Verlauf der Pair-Programming-Sitzung kann eine große Wissenslücke entstehen. Diese Wissenslücken und ggf. *Hierarchical Behavior* führen zum Anwachsen eines sogenannten *Power Gaps*. Als Reaktion auf fehlendes *Equalizing Behavior* können beim Partner *Defensive Behavior* und *Disengaging Behavior* entstehen. Deswegen ist

es immer am Besten, immer direkt nachzufragen, wenn man etwas nicht versteht. Wer direkt nachfragt, läuft nicht das Risiko, den Anschluss zu verlieren und die Togetherness bleibt dadurch erhalten. Auch der aktive Partner kann proaktiv Wissen transferieren. Deswegen empfehlen sich ein regelmäßiger Driver/Observer-Wechsel sowie sogenannte Mini-Retros am Ende einer Paar-Programmierungssitzung – im *Agile Programming* sind Retros eine kurze Besprechung, was gut und was schlecht gelaufen ist. Hilfreich ist es zudem, die Zielstellungen schon im Voraus abzustimmen. Dabei sollten insbesondere die jeweiligen Prioritäten in Hinsicht auf die drei großen Pair-Programming-Ziele (1.) *Speed*, (2.) *Quality* und (3.) *Knowledge Transfer* zwischen den beiden Partnern in Einklang gebracht werden.

Die Praktische Anwendung in der Gruppenarbeit

Nach dem Input bestand der zweite Teil der Veranstaltung aus dem selbständigen Ausprobieren des Gelernten durch die Teilnehmer:innen. In Zweiergruppen wurde anhand von mitgebrachten realen Programmierprojekten Wissen ausgetauscht und es wurden zusammen kreative Lösungen für die angeschauten Aspekte dieser Projekte gefunden.

Copyright Hintergrundbild: Paul Bayer 2025 (All rights reserved)

Citation Information

Author: Paul Bayer

Title: Rückblick auf den DH Code Review Day 2025

Published: 29.06.2026

Cite as: Paul Bayer: Rückblick auf den DH Code Review Day 2025. In: *IZ D2MCM Blog* [Weblog], 29.06.2026. URL: <https://izd2m.hu-berlin.de/blog/>
