

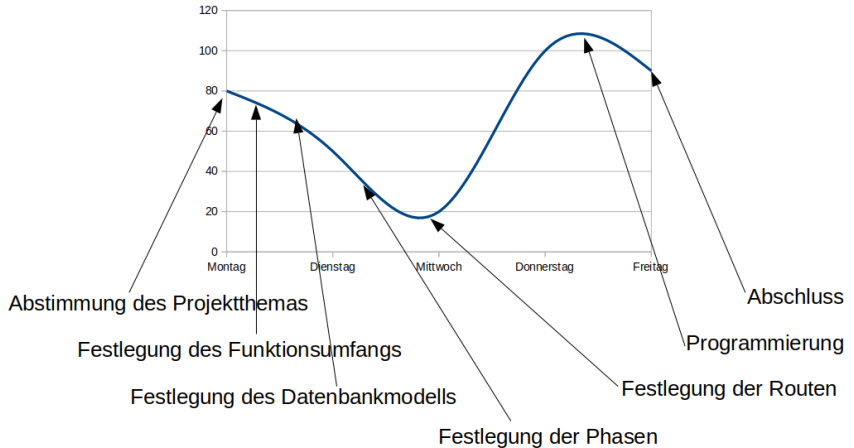
Wie haben wir gearbeitet?

Projektphasen – Versionsverwaltung – Pair-Programming

Projektphasen

- Abstimmung des Projektthemas
- Festlegung des Funktionsumfangs
- Festlegung des Datenbankmodells
- Festlegung der Phasen
- Festlegung der Routen
- Programmierung
- Abschluss

Laune der Teilnehmerinnen und Teilnehmer



Phase	Sozialform	Arbeiten
Abstimmung des Projektthemas	Plenum	Brainstorming, Abstimmungen
Festlegung des Funktionsumfangs	Plenum	Diskussionen, Abstimmungen
Festlegung des Datenbankmodells	Plenum	Diskussionen, Abstimmungen Erstellen eines Plakats
Festlegung der Phasen	Plenum, kurze PA	Diskussionen, Abstimmungen Erstellen eines Plakats
Festlegung der Routen	Plenum	Diskussionen, Abstimmungen, Erstellen eines Arbeitsauftrags zu jeder Route in GitHub
Programmierung	PA	Programmieren, Einstellen in GitHub, Dokumentation des Fortschritts
Abschluss	Plenum	Tests, Bugfixes, Dokumentation der nicht schnell behebbaren Fehler

Phase	Sozialform	Arbeiten
Abstimmung des Projektthemas	Plenum	Brainstorming, Abstimmungen
Festlegung des Funktionsumfangs	Plenum	Diskussionen, Abstimmungen
Festlegung des Datenbankmodells	Plenum	Diskussionen, Abstimmungen Erstellen eines Plakats
Festlegung der Phasen	Plenum, kurze PA	Diskussionen, Abstimmungen Erstellen eines Plakats
Festlegung der Routen	Plenum	Diskussionen, Abstimmungen , Erstellen eines Arbeitsauftrags zu jeder Route in GitHub
Programmierung	PA	Programmieren, Einstellen in GitHub , Dokumentation des Fortschritts
Abschluss	Plenum	Tests, Bugfixes, Dokumentation der nicht schnell behebbaren Fehler

Abstimmungen

Regel

Was abgestimmt wurde, wird nicht mehr geändert!
(es sei denn, es handelt sich um einen Fehler)

Rückmeldung



- Vorherige Planung hat dazu geführt, dass am Ende alles zusammengepasst hat.
- Klarheit durch genaue Planung machte effiziente Programmier-Phase möglich.
- Lange Planungsphase hat genervt, aber sachliche Durchdringung ist wichtig.
- Nach dem ersten GET/POST/DELETE-Tripel ging es schneller weiter.



- Zeitplanung für die Phasen, wenig Programmieranteil.
- lange Planungsphase im Plenum ermüdend
- Definition der Routen wäre arbeitsteilig besser gewesen.
- wenig Planung für duplizierten Code (Tabellen, Konsistenzprüfungen)

Vorschläge

- Routen arbeitsteilig erstellen
- dabei Methoden des Pair-Programming einsetzen?
- Konflikte und Inkonsistenzen im Plenum besprechen

Versionsverwaltung

Zweck

- Verwaltung unterschiedlicher Versionen von Projekten aus vielen Dateien

Begriffe

- Commit
- Repository

Umsetzung im Projekt

- git (Kommandozeilen-Programm) auf jedem Rechner
- Austausch über GitHub (Online-Plattform)

Arbeitsschritte

Vorarbeit

Gerüst der zu erstellenden Anwendung im zentralen Repository (GitHub)

Jede:r Entwickler:in

- legt eigenen Account an (GitHub),
- erstellt eigenes Repository als Fork (GitHub),
- holt die Dateien auf den eigenen Rechner (`git pull`).

Arbeitsschritte

Workflow

Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per `fetch and merge`,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow


Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per `fetch and merge`,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow


Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per  **fetch and merge**,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow


Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per  **fetch and merge**,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow


Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per  **fetch and merge**,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow



Jedes Entwickler-Team

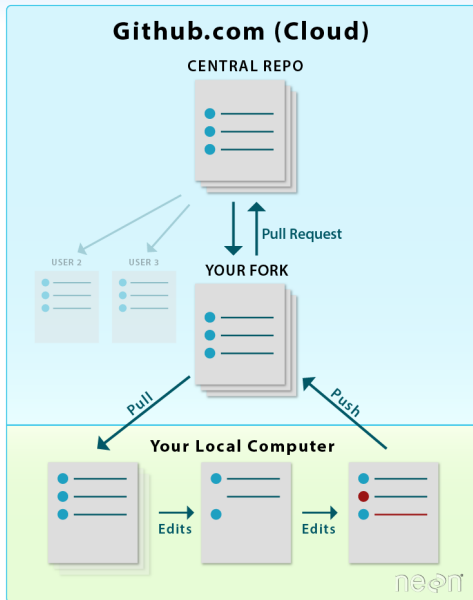
- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per  **fetch and merge**,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen (Pull-Request auf GitHub).

Arbeitsschritte

Workflow

Jedes Entwickler-Team

- arbeitet lokal,
- bringt erneut das eigene Repository auf den Stand des zentralen Repository,
 - erst auf der Weboberfläche per  **fetch and merge**,
 - dann lokal per `git pull`,
- aktualisiert die lokal bearbeiteten Dateien im lokalen Repository (`commit all`),
- überträgt die Aktualisierung in das Online-Repository (`git push`),
- stellt eine Anfrage im zentralen Repository, die eigenen Änderungen einzupflegen ( Pull-Request auf GitHub).



Quelle:
<https://www.earthdatascience.org/images/earth-analytics/git-version-control/git-push-pull-flow.png>

Programmieraufträge als Issues in GitHub

GET /lehrkraft/:id/bearbeiten #32



sebfisch opened this issue 3 days ago · 0 comments



sebfisch commented 3 days ago • edited ▾

Owner



Knopf "krank melden" POST /lehrkraft/:id/krank

Formular mit vorausgefüllten Feldern für Name, Kuerzel, istAdmin. Leeres Passwort-Feld.

Speichern-Knopf POST /lehrkraft/:id/bearbeiten

Formular mit einem Eingabefeld für eine Klasse.

Knopf "hinzufügen" POST /unterrichtet/:id (Id der Lehrkraft)

Liste mit Klassen, in denen die Lehrkraft unterrichtet mit Lösch-Knopf DELETE /unterrichtet/:id (Id des Datensatzes aus der Tabelle für die Beziehung "unterrichtet")

Verwaltung der Programmieraufgaben

- Issue-Nummern auf PostIt-Zetteln
- Gruppierung in Sinneinheiten (z.B. GET /schueler, POST /schueler und DELETE /schueler zusammen)
- Zuordnung zu einem Tisch
- nach Erledigen umkleben



Nutzung einer Versionsverwaltungssoftware

Pro und Contra

+

- ermöglicht effiziente Aufteilung und Zusammenführung

-

- (ungewohnte und lange Schrittfolge zur Aktualisierung)

Nutzung einer Versionsverwaltungssoftware

Rückmeldung

+

- Arbeitsteilig an größerem Produkt arbeiten
- Aufgaben in kleine Häppchen gegliedert

-

- Umgang nicht erst in der Projektwoche erlernen

Pair-Programming

Ablauf

Grundidee

- Arbeiten in Zweierteams mit definierten Rollen
 - Driver (Pilot)
 - Navigator
- regelmäßiger Rollentausch

Rotation

- regelmäßiger Tausch der am Tisch links sitzenden Person nach links, dann der rechts sitzenden Person nach rechts
- Die Teams werden dadurch immer neu zusammengesetzt.

Pair-Programming mit Rotation

Pro und Contra

+

- Kontrollfunktion der zweiten Person
- breiteres Wissen über den Quellcode
- beim Wechsel:
 - Lernen durch Erklären (bleibende Person)
 - frischer Blick und neue Ideen (hinzugekommene Person)
- Entwicklung in Sackgassen wird reduziert
- höhere Disziplin
- verbesserte Kommunikation

-

- Wechsel unangenehm, wenn man gerade konzentriert arbeitet
- Zeit für Einarbeitung in fremden Quelltext

Pair-Programming

Rückmeldung zur Programmierphase



- verschiedene Herangehensweisen
- Programmieren super gefallen
- Methoden des Pair-Programming auch in früheren Phasen

Ergänzende Folien

Projektphasen

Themenfindung

- Brainstorming von Ideen
- Abstimmung
- Ergebnis: Festlegung auf ein gemeinsames Projekt
(Anwendung zur Verwaltung von Elternsprechtagen)

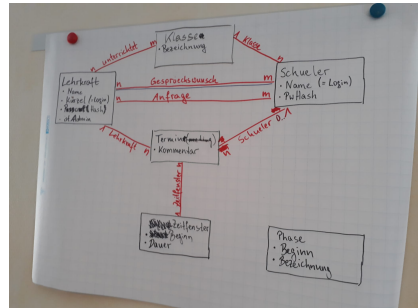
Festlegung des Funktionsumfangs

- Was soll das Programm können?

Projektphasen

Datenbankdesign

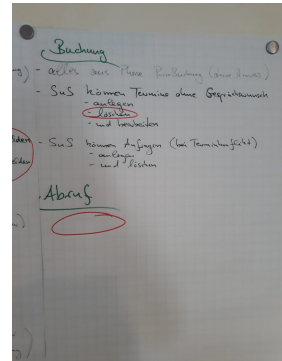
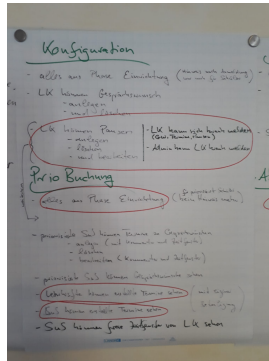
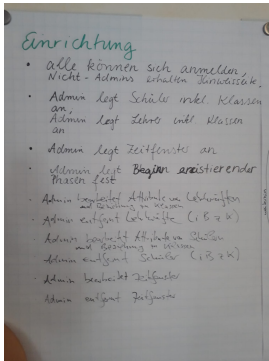
- Konventionen (Sprache, Schreibweisen)
- Entitäten (inkl. Attribute)
- Beziehungen
- Ergebnis: ER-Diagramm



Projektphasen

Festlegung der Phasen

- Wer soll was in welcher Phase können?
- Ergebnis: Phasen-Beschreibung fixiert auf Flip-Chart



Versionsverwaltung

Begriffe

Commit

- Speichern des Zustands (eines Ordners), um später darauf zurückkommen zu können
- hat eine Beschreibung (z.B. "Fehlerbehebung in `get_schueler.rb`"), aber keinen Namen
- jeder Commit hat einen Vorgänger-Commit (→ Baumstruktur)
mögliche Änderungen: Dateien hinzugefügt, geändert, gelöscht
- Checkout versetzt das aktuelle Arbeitsverzeichnis in den Zustand des Commits.

Versionsverwaltung

Begriffe

Branch

- Zeiger auf einen Commit
- hat einen Namen

Repository

- darin werden verschiedene Commits gespeichert