

Frameworks, Daten und Dienste im Web

Web Development

Vertiefungsmodul im Medieninformatik Bachelor

02.06.2022, Dirk Breuer

Tagesziel

- Gruppen stehen fest
- Arbeitsweise für Projektarbeit ist bekannt
- Anforderungen an das Konzept sind bekannt
- Grundverständnis über das "Schneiden" einer Anwendung

Kollaboration über Versionskontrolle

Empfohlenes Repository Layout

- Name: FDDWSS22_NAME_NAME_NAME
- Sichtbarkeit: privat
- Branch Protection aktivieren
- Squash Commits
- Issues für Arbeitspakete (optional auch als Projekt verwalten)
- Pull-Request zu jedem Issue
- Branchnamen: ISSUE_NUMER_short_description

Branch protection rule

Branch name pattern ⬆

main

Protect matching branches

☒ **Require a pull request before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

☒ **Require approvals**

When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.

Required number of approvals before merging. 1 ⬇

☒ **Dismiss stale pull request approvals when new commits are pushed**

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☐ **Require review from Code Owners**

Require an approved review in pull requests including files with a designated code owner.

☐ **Require status checks to pass before merging**

Choose which **status checks** must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☒ **Require conversation resolution before merging**

When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more.](#)

☐ **Require signed commits**

Commits pushed to matching branches must have verified signatures.

☒ **Require linear history**

Prevent merge commits from being pushed to matching branches.

☐ **Require deployments to succeed before merging**

Choose which environments must be successfully deployed to before branches can be merged into a branch that matches this rule.

☒ **Include administrators**

Enforce all configured restrictions above for administrators.

Rules applied to everyone including administrators

☐ **Allow force pushes**

Permit force pushes for all users with push access.

☐ **Allow deletions**

Allow users with push access to delete matching branches.

Create

Pull Requests

When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled. If you have linear history requirement enabled on any protected branch, you must enable squashing or rebasing.

☐ **Allow merge commits**

Add all commits from the head branch to the base branch with a merge commit.

☒ **Allow squash merging**

Combine all commits from the head branch into a single commit in the base branch.

☐ **Default to PR title for squash merge commits**

This will pre-populate the commit message with the PR title when performing a squash merge.

☐ **Allow rebase merging**

Add all commits from the head branch onto the base branch individually.

Control how and when users are prompted to update their branches if there are new changes available in the base branch.

☐ **Always suggest updating pull request branches**

Whenever there are new changes available in the base branch, present an “update branch” option in the pull request.

You can allow setting pull requests to merge automatically once all required reviews and status checks have passed.

☐ **Allow auto-merge**

Waits for merge requirements to be met and then merges automatically. [Learn more](#)

After pull requests are merged, you can have head branches deleted automatically.

☒ **Automatically delete head branches**

Deleted branches will still be able to be restored.

<code/late>

Monorepo vs Multi-Repo

- Verteilte Systeme brauchen nicht dringen verteilte Repositories
- Monorepo erleichtert die Kollaboration
- Monorepo erleichtert die Abnahme

```
sample_application >
├── doc/
│   └── Concept.md
├── authentication_service/
├── game_service/
├── table_service/
├── user_service/
│   └── Dockerfile
├── LICENSE
├── README.md
└── docker-compose.yml
```


Issue-Template

- Titel: Aussagekräftig und kurze Beschreibung
- Typ: Feature / Bug
- Definition of Done: Beschreibung welche Persona welche Ziel im System verfolgt und wie dieses Ziel zu erreichen ist
- Einzelne Tasks
- **Merke:** Es kann mehrere Pull-Requests zu einem Issue geben!

Beispiel


Nutzer:in kann sich mit E-Mail / Passwort anmelden

Feature

Als Nutzer:in möchte ich mich bei der Anwendung unter Angabe von meiner E-Mail-Adresse und Passwort anmelden können. Nach der Anmeldung möchte ich sehen können, dass ich angemeldet bin. Falls meine Anmeldung nicht erfolgreich war, möchte ich eine Fehlermeldung angezeigt bekommen.

Tasks

- [] Seite mit Anmeldeformular
- [] Formular mit Feld für E-Mail und Passwort
- [] Serverseitige Auswertung der Anmeldedaten
- [] Persistieren der Anmeldung in der Session
- [] Anzeige der genutzten E-Mail im Frontend
- [] Fehlermeldung im Falle von Anmeldefehler
- [] Im Fehlerfall ****keine**** Aussage über Grund des Fehlers

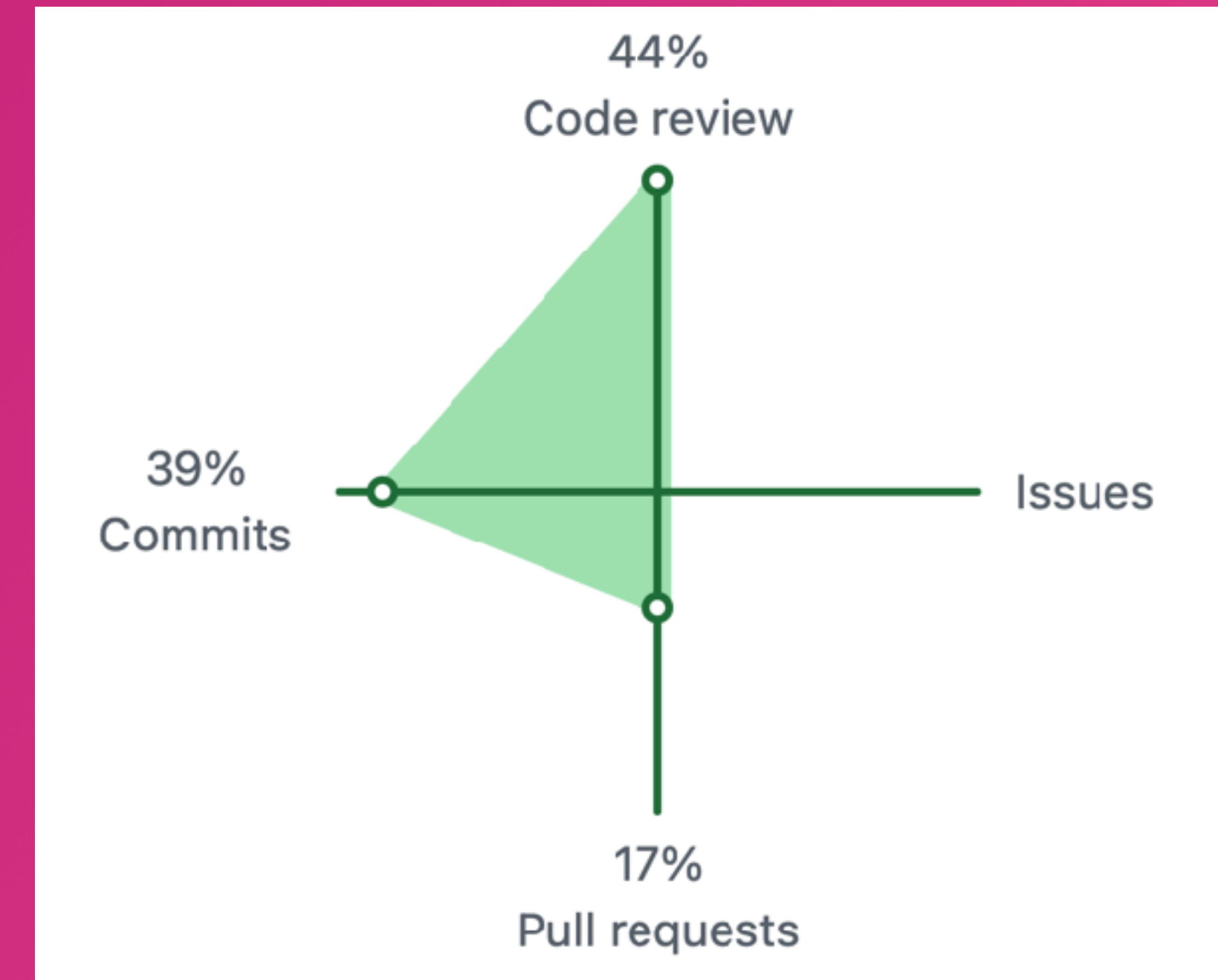


Pull-Request == Code Review

Code Review

- Mehr Leute sehen mehr Fehler
- Wissensteilung
- Code Ownership aufbrechen
- Kommunikation
- Kollaboration

Code Review



- Sind Teil des Jobs
- Genauso gewissenhaft zu erledigen wie die Entwicklung selbst
- Keine Garantie für keine Fehler ;-)

Reviewee

- Freundlich, respektvoll und sachlich
- Kleine, zusammengehörige Änderungen (< 400 LOC)
- Warum sind die Änderungen nötig?
- Welche Entscheidungen/Kompromisse wurden getroffen? Und warum?
- Auswirkungen auf die Software / den Betrieb / das Team?
- Besonders zu beachtende Punkte

Reviewer

- Freundlich, respektvoll und sachlich
- Lob aussprechen
- Fragen statt Anweisungen geben
- Wurden alle Anforderungen erfüllt?
- Entsprechen die Änderungen den Qualitätsansprüchen?
- Gibt es Verständnisprobleme?
- Dokumentation / Tests vorhanden?

Automatische Reviews

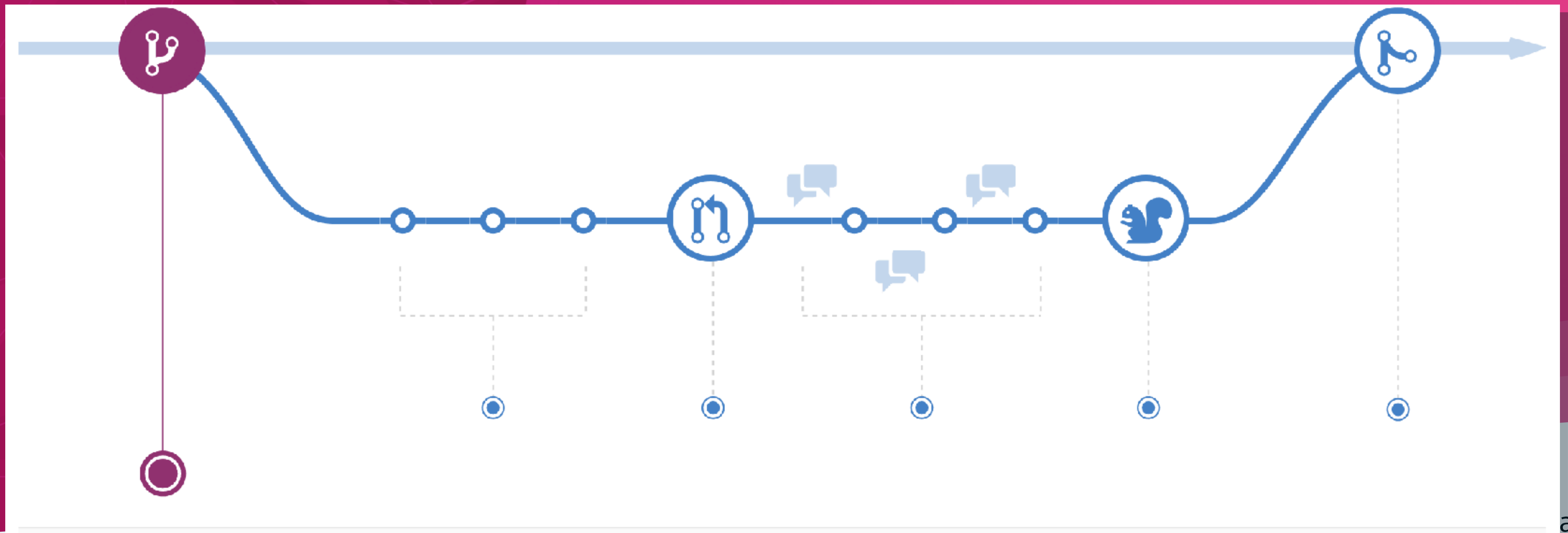
- Statische Codeanalyse
- Einhalten von Styleguides
- Analyse auf Schwachstellen
- Ausführen der Test Suite

To be continued...

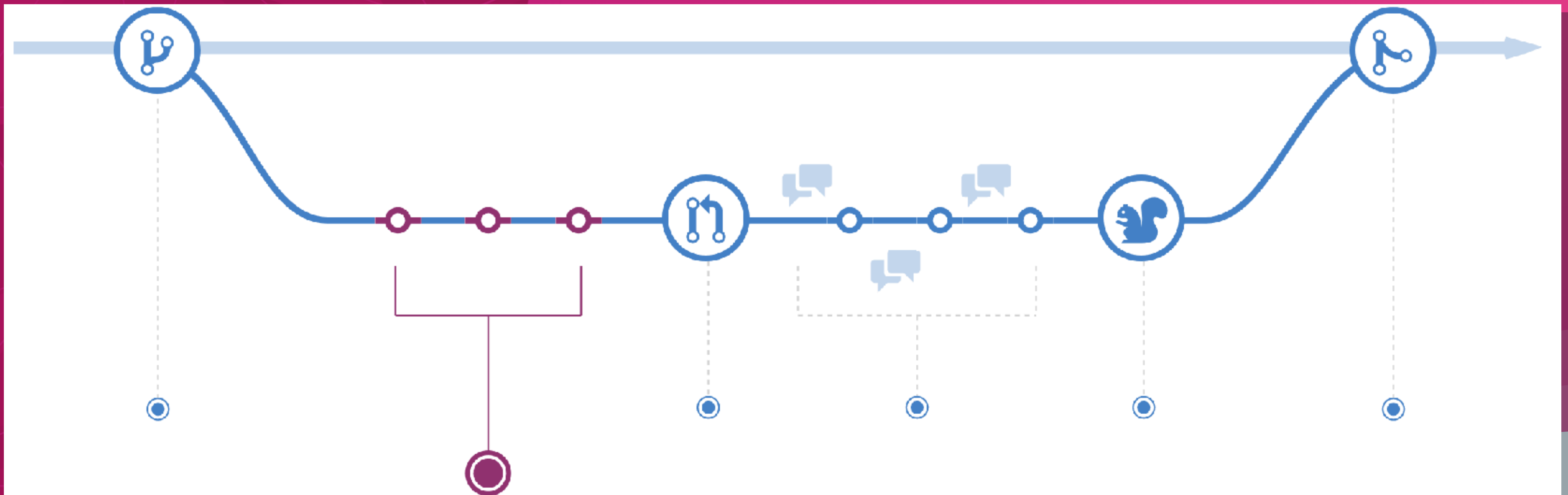
GitHub Flow

<https://guides.github.com/introduction/flow/>

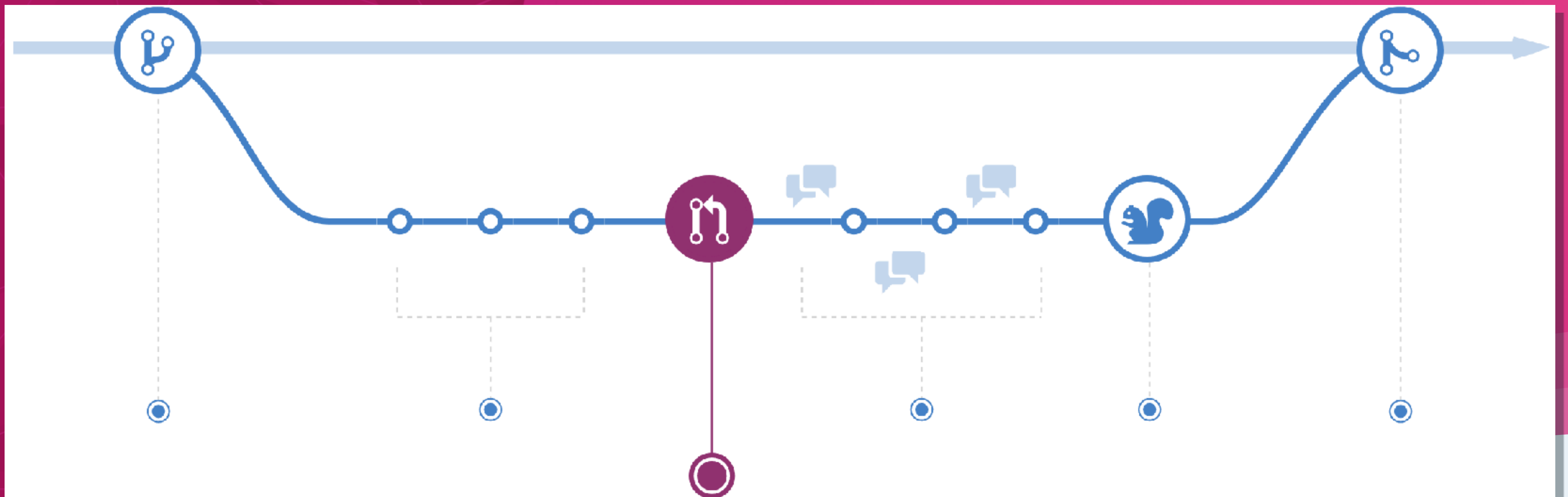
1. Branch erstellen



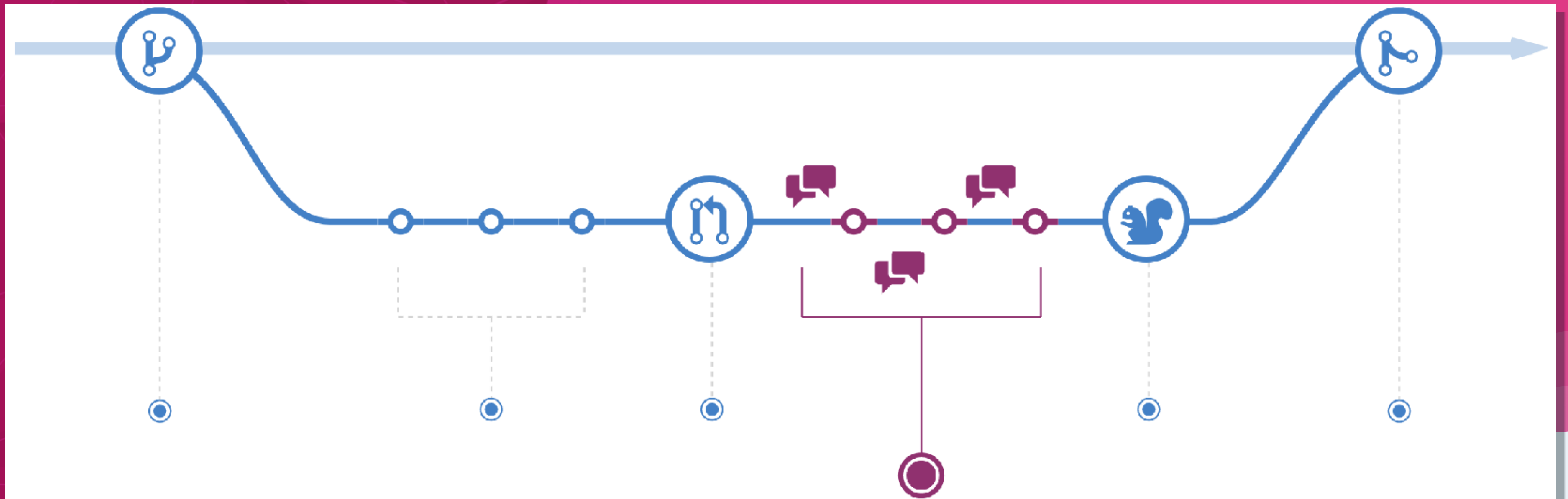
2. Commits hinzufügen



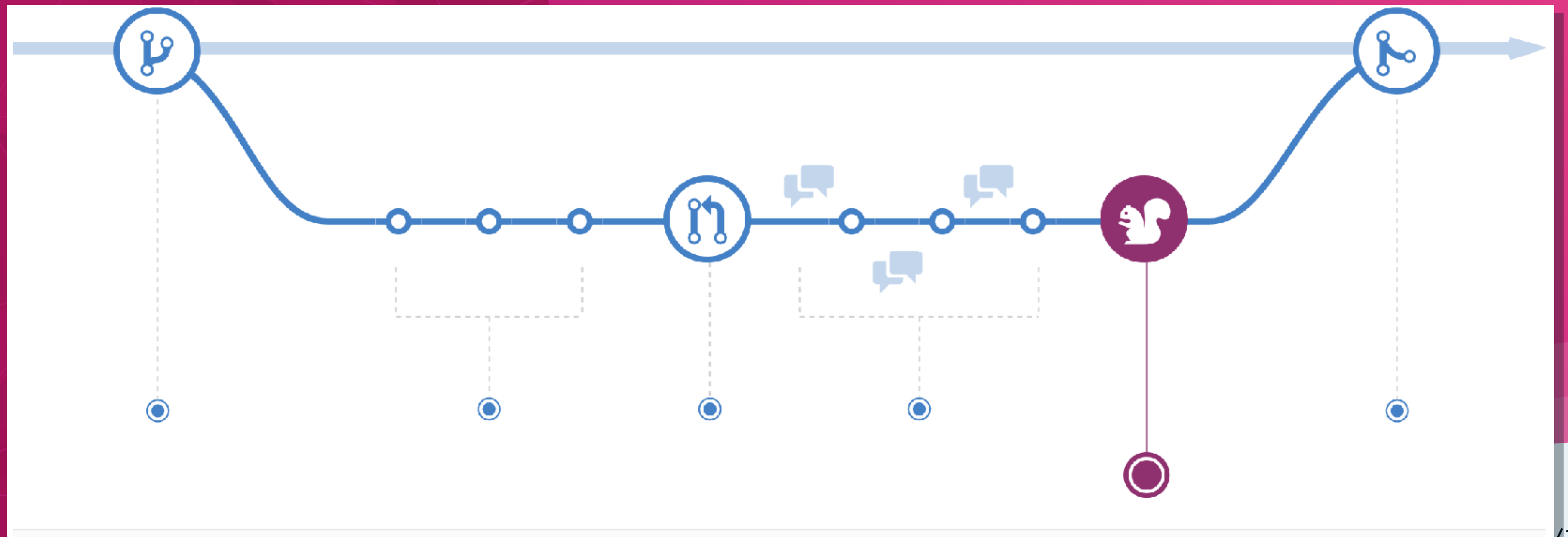
3. Pull Request erstellen



4. Review / Diskussion

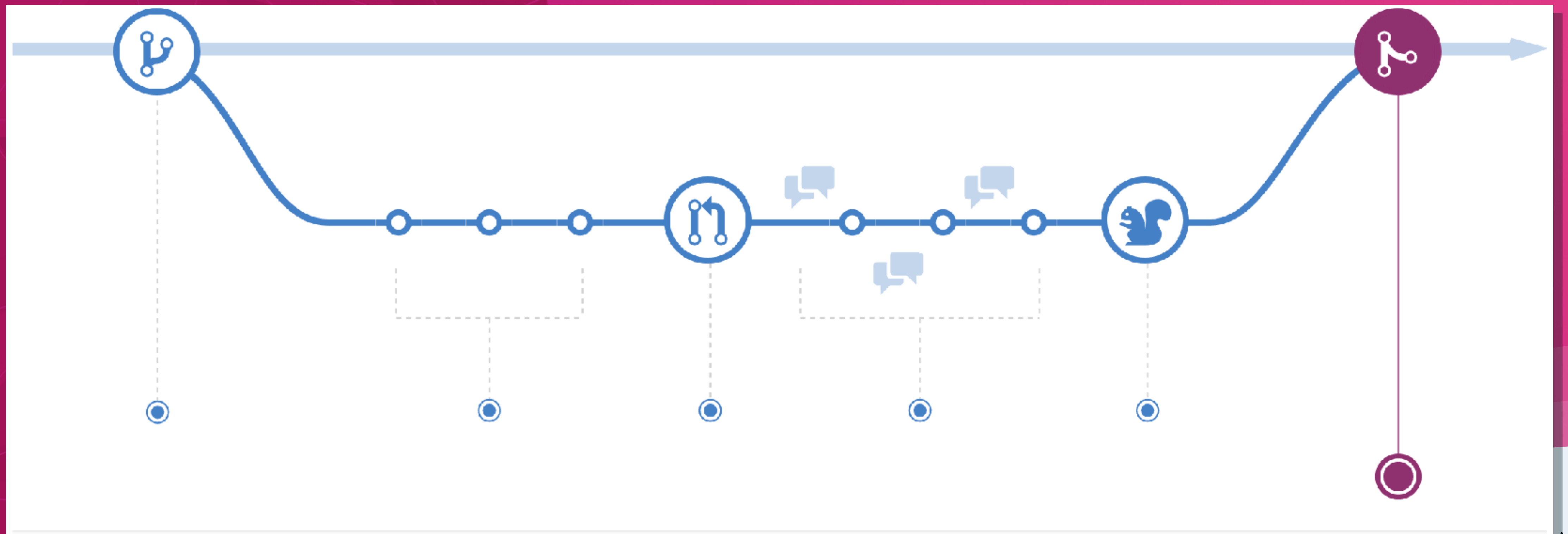


5. Ship it!



```
/later>
```


5. Merge in main





Konzept

Konzept

- Ziel ist die klare Definition der Anwendung
- Welches Problem soll gelöst werden?
- Welche fachlichen Domänen existieren in der Anwendung?
- Welche Entitäten gibt es in der Anwendung?
- Welche Kommunikationswege finden sich in der Anwendung?
- Welche Events werden in der Anwendung erzeugt?
- Welche Daten müssen gespeichert werden?
- An welchen Stellen kann die Anwendung in Services geschnitten werden?
- Welche externen Dienste werden angebunden?

Konzept

- Textuelle Beschreibung
- Mockups
- Pseudo-Code (für Event-Struktur)
- Verschiedene Diagramme zu
 - Domain Model (fachliche Entitäten und ihre Beziehungen)
 - Datenstruktur
 - Kommunikationsfluss



Beispiel

Problembeschreibung

Implementierung des Kartenspiels *Mau-Mau* als rundenbasierte Web-Anwendung für 2-5 Spielende.

Beschreibung der Domäne

Mau-Mau ist ein Kartenspiel für zwei bis fünf Spielende, bei dem es darum geht, ihre Karten möglichst schnell abzulegen. Meist wird Mau-Mau mit einem französischen oder deutschen Kartenspiel zu 32 (deutsches Skatblatt) Karten gespielt, ist aber prinzipiell mit jedem anderen Blatt mit bis zu 52 Karten spielbar.

Mau-Mau ist ein Auslegenspiel. Gewonnen hat, wer zuerst alle ihre Karten abspielen konnte. Der Gewinn wird mit einem Ausruf „Mau Mau“ kundgetan. Zu Beginn erhalten die Spielenden die gleiche Anzahl Karten (oft fünf oder sechs), die sie verdeckt – als Kartenfächer – auf ihre Hand nehmen. Die restlichen Karten werden verdeckt als Stapel (Talon) abgelegt. Die oberste Karte des Talons wird offen daneben gelegt.

*frei nach Wikipedia: [https://de.wikipedia.org/wiki/Mau-Mau_\(Kartenspiel\)](https://de.wikipedia.org/wiki/Mau-Mau_(Kartenspiel))

Beschreibung der Domäne

Reihum legen nun die Spielenden eine ihrer Karten offen auf die nebenliegende Karte – wenn dies möglich ist. Möglich ist dies, wenn die abzulegende Karte in Kartenwert oder Kartenfarbe mit der obersten offen liegenden Karte übereinstimmt. Kann oder will ein Spielender keine Karte ablegen, so muss eine Karte vom Talon gezogen werden. Das Ausspielen der Karte ist jedoch erst möglich, wenn die Spielende erneut an der Reihe ist. Sollte der Talon irgendwann aufgebraucht sein, so werden die abgelegten Karten, außer der obersten sichtbaren, gemischt und erneut als Talon ausgelegt.

*frei nach Wikipedia: [https://de.wikipedia.org/wiki/Mau-Mau_\(Kartenspiel\)](https://de.wikipedia.org/wiki/Mau-Mau_(Kartenspiel))

Beschreibung der Domäne

Einige Karten haben besondere Fähigkeiten. Falls eine sieben ausgespielt wird, muss der nachfolgende Spielende 2 Karten vom Talon ziehen. Spielende die einen Buben spielen, dürfen sich eine Farbe wünschen. Ebenso kann ein Bube auf jede Farbe oder jeden Wert gelegt werden. Das Ausspielen einer acht lässt den nachfolgenden Spielenden einmal aussetzen.

Nachdem Ausspielen der vorletzten Karte muss dies durch ein „Mau“ öffentlich angesagt werden. Vergisst der Spielende das Melden und eine andere Spielende bemerkt dies, bevor der nächste Spielende seine Karte legt, so muss er als Strafe eine Karte ziehen.

*frei nach Wikipedia: [https://de.wikipedia.org/wiki/Mau-Mau_\(Kartenspiel\)](https://de.wikipedia.org/wiki/Mau-Mau_(Kartenspiel))

Beschreibung der Domäne

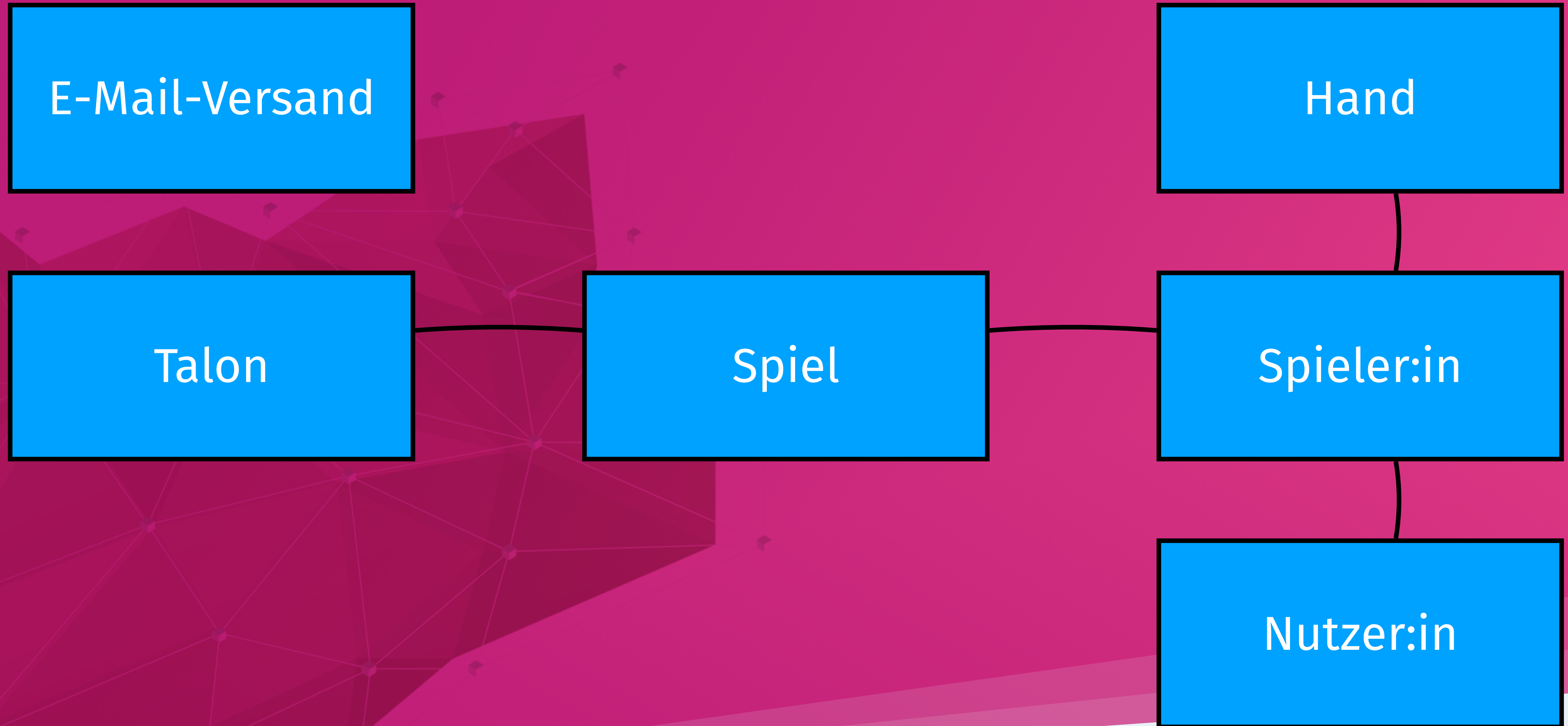
Die erste Karte offene Karte hat keine negativen Effekt. Im Falle eines Buben bestimmt der gebende Spielende welche Farbe als nächstes gespielt werden muss.

Das Spiel wird solange gespielt bis nur noch ein Spielender übrig bleibt. Am Ende erhält jede Spielende Punkte entsprechend ihrer Platzierung.

Spielende können an beliebig vielen Spielen gleichzeitig teilnehmen.

*frei nach Wikipedia: [https://de.wikipedia.org/wiki/Mau-Mau_\(Kartenspiel\)](https://de.wikipedia.org/wiki/Mau-Mau_(Kartenspiel))

Domänen



Entitäten

Player

uid
active_game
hand
user
score

Game

uid
duration
players
active_player
talon
discard_pile

Card

uid
value
suit

User

uid
email
password
avatar

Events

- DrawCard
- DealCard
- PlayCard
- NextPlayersTurn
- SkipTurn
- RequireSuit
- Announce
- CallOutPlayer

Kommunikationswege

- Fokus auf Kommunikation über Events
- Bsp.: Nicht den gesamten Game State rumschicken
- Immer alle Player ansprechen (Player entscheiden selbst wie sie mit dem Event umgehen)
- Asynchrone Integration einzelner Services in Web-UI (via HTTP Aufrufe)

Daten

- Ein Service muss alle notwendigen Daten zur Bereitstellung der Funktionalität vorhalten
- Dadurch können die gleichen Daten mehrfach an unterschiedlichen Stellen (in unterschiedlichen Services) gespeichert sein
- Austausch der Daten über Events!
 - Bsp.: Nicht alle Spielenden melden sich regelmäßig beim Spiel, sondern das Spiel emittiert Events, wenn sich der Zustand ändert.

Schnitte

- Schnitte so durchführen, dass die entstehenden Services für sich alleine stehen können
- Es dürfen keine Abhängigkeiten entstehen, die eine direkte Kommunikation bedingen
- Integration über in Web-UI möglich!

Beispiele

- Authentifizierung der Nutzer:innen
 - E-Mail / Passwort im Service gespeichert
 - Nach erfolgreicher Authentifizierung Event erzeugen mit internem Token. Nutzer:in schickt Token jedem anderen Service mit. Falls Token bekannt, ist ein Zugriff möglich.

Beispiele

- Spieler:in spielt Karte
 - Event PlayCard veröffentlichen
 - Spiel ermittelt durch das Event den neuen Game State
 - Veröffentlicht entsprechende Events (bspw. skipTurn und nextTurn)
 - E-Mail-Service verschickt entsprechende E-Mails an Spieler:innen

Beispiele

- Anzeigen einer Karte
 - PlayCard Event beinhaltet neben Farbe (`suit`) und Kartenwert (`value`) auch ein URL-Template
 - CardService stellt einen Endpunkt <https://cards.maumau.game/{suit}/{value}> zur Verfügung
 - Antwort ist Markup/CSS/JS zur Anzeige der entsprechenden Karte

Projektarbeit

- Abgabe (via GitHub): 08.09.2022
- Präsentation: 16.09.2022
- Präsentation im Plenum in Gummersbach
- 5-10 Minuten pro Projektgruppe
- Ziel der Präsentation: Ergebnisse und Herausforderungen einem Fachpublikum vorstellen
- Zu erreichende Punkte: 50

Zielsetzung

- Anwendung die in Microservices / Komponenten aufgeteilt ist
- Asynchrone Kommunikation zwischen Microservices (bspw. durch Events)
- Anbindung mindestens eines externen (offenen) Datendienstes
- Es muss State geben! (Ohne State ist alles möglich)
- Einfache! Web-UI zur Interaktion mit der Anwendung
- Deployment der Anwendung
- Betriebskonzept
- Dokumentation und Nachvollziehbarkeit in GitHub

Zu beachten

- Fokus auf inhaltliches Arbeiten setzen
- Komplizierte Infrastruktur und/oder Dienste vermeiden
- Aufwendiges UI nicht notwendig
- Umfang im Vorfeld klar definieren
- Aspekte im Vorfeld explizit ausklammern
- Umsetzung mit NodeJS auf dem Server

Thema

- Frei wählbar
- Sehr gerne Synergien mit IoT und/oder Frontend Development!
- Asynchrone Interaktionsmodelle können hilfreich sein (zB.: Chat)
- Integration auch in der UI möglich

- Verteilte Anwendungen mit asynchroner Kommunikation sind schwierig
- Selbststudium herausfordernd und zeitintensiv
- Daher: Vorlesungstermine zum gemeinsamen Arbeiten nutzen

Beispiel-Projekt

- Wir bauen gemeinsam in den Veranstaltungen eine verteilte Anwendung
- Wird die wesentlichen Aspekte der Anforderungen an die Projektarbeit beinhalten
- Projekt in Ruby
- Quellcode steht via GitHub zur Verfügung
- Wird als Beispiel für die Thematik Deployment genutzt

Vorgehen

- Bildung von Projektgruppen (02.06.2022)
- Präsentation Konzept (09.06.2022)
- Nachvollziehbare Implementierung in GitHub
- Deployment der Anwendung
- Dokumentation und Betriebskonzept
- Abgabe von Code und Zugang zur Anwendung (08.09.2022)
- Präsentation im Plenum (16.09.2022)



Nächster Termin

09. Juni 2022, 10:00 Uhr

Ersatztermin für 27.05.2022

- Am 09.06.2022 **ganztägig** Veranstaltung
- Ausgedehnte Mittagspause
- (Reichlich) Zeit für Besprechung der Projektkonzepte
- (Live) Coding der Demo-Anwendung

What's next?

- Projekt konzipiert haben und im Plenum vorstellen können
 - Welches Ziel wird verfolgt?
 - Welche Herausforderungen sind zu erwarten?
 - Welche Lösungen werden vorgeschlagen?

Todos

- git installiert
- GitHub eingerichtet und Repository bereit
- Code-Editor installiert
- NodeJS installiert
- Docker installiert: <https://docs.docker.com/get-docker/>



Danke.