TELEGRAM CLIENT

SWT I SoSe 2021

Gruppe 2

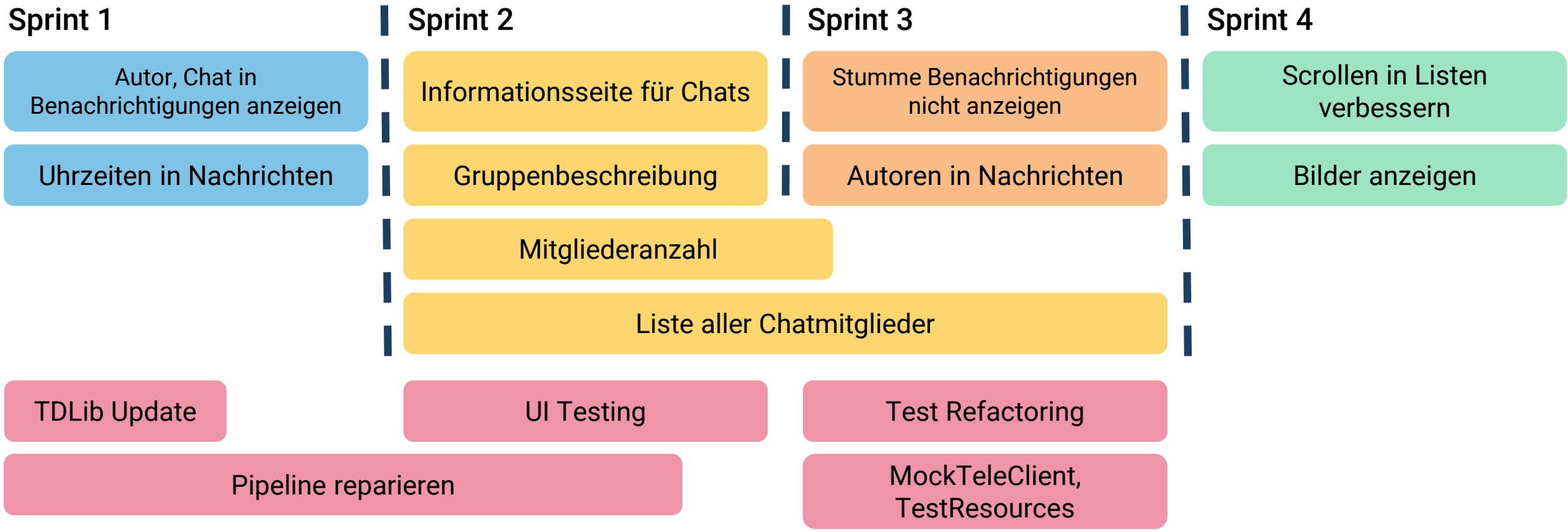
Paul Ermler
Tom Richter
Philipp Keese
Jannis Berndt
Raphael Kunert
Romeo Sommerfeld

13.07.2021

Agenda

- Live Demo
- Projektverlauf & Anforderungen
- Architektur
- Verhaltenstechnische Zusammenhänge
- Entwicklungsprozess
- Praktiken
  - Pair Programming
  - Continuous Integration
- Metriken
- Reflexion

Projektverlauf



Funktionale Anforderungen

- Nachrichten senden und empfangen (Text, Bilder, Emoji)
- Relevante Informationen zu Nachrichten anzeigen

- Videos, Audio, Sticker

- Gruppenbeschreibung
- Mitgliederliste und –anzahl
- grundlegender Support für Basic-, Super- und Private-Chats

- Channel-Support
- Einstellungen ändern

- sauberes Scrollen
- Nachladen von Nachrichten beim Scrollen

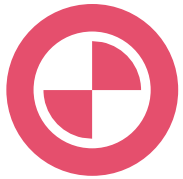
- Profilinformationen anzeigen/ändern

nicht-implementiert / implementiert

Nichtfunktionale Anforderungen

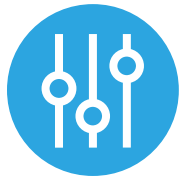
Einheitlicher und getesteter Code

Kriterium: Vor Push auf Develop alle Linter Tests grün
und Coverage maximal um −1% gesunken



Kurze Testlaufzeiten

Kriterium: Alle Tests benötigen insgesamt < 3 Minuten



UI ähnlich zum Original Telegram

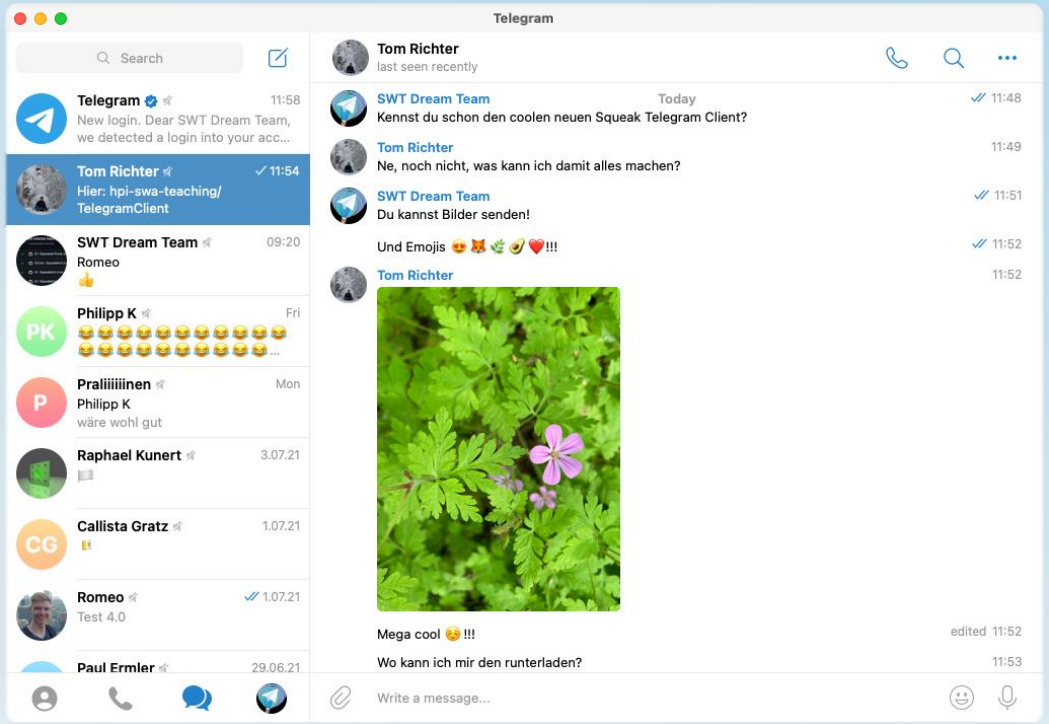
Kriterium 1: Nutzer kann alle Kernfunktionen in < 10 Sekunden durchführen

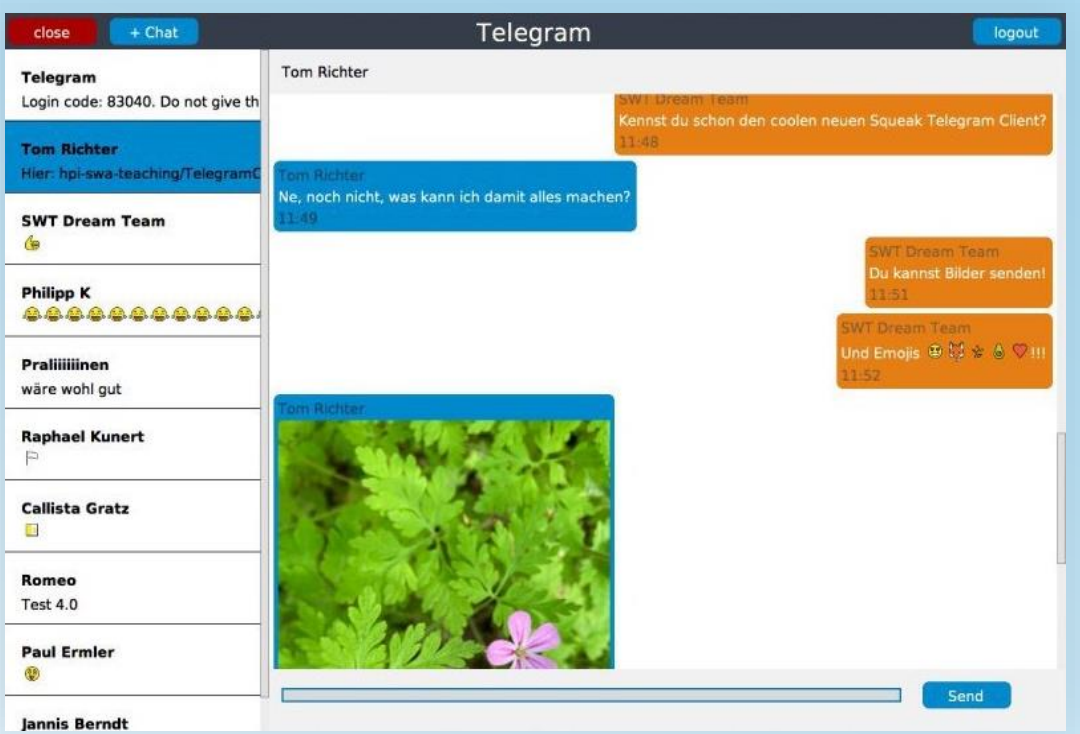
Kriterium 2: Vergleich von Screenshots

Anforderung: UI ähnlich zum Original

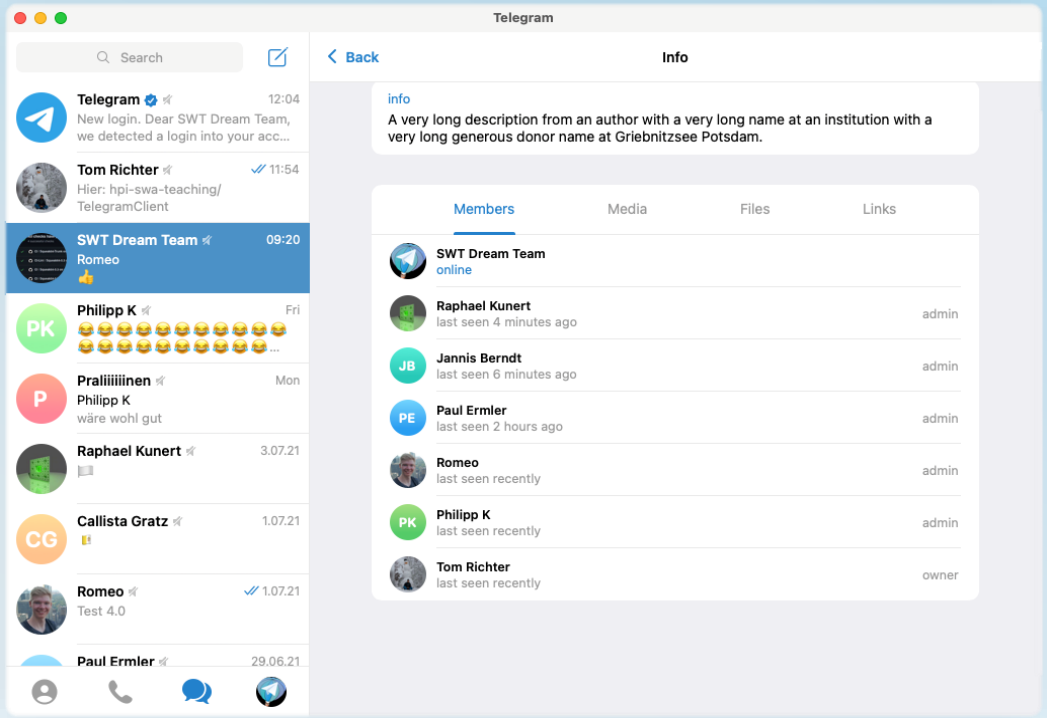
Aufgabe	Zeit in Sekunden
Öffne den Chat mit Person X.	7
Sende eine Nachricht an Person X.	6
Wie viele Mitglieder hat Gruppe X?	3
Welche Mitglieder sind in Gruppe X?	2
Wann wurde die letzte Nachricht von Person X gesendet?	3
Wer hat die letzte Nachricht in Gruppe X gesendet?	3
Was ist die Gruppenbeschreibung von Gruppe X?	4
Was ist die Beschreibung des letzten Bildes in Chat X?	4
Von wem stammt die letzte Nachricht?	3

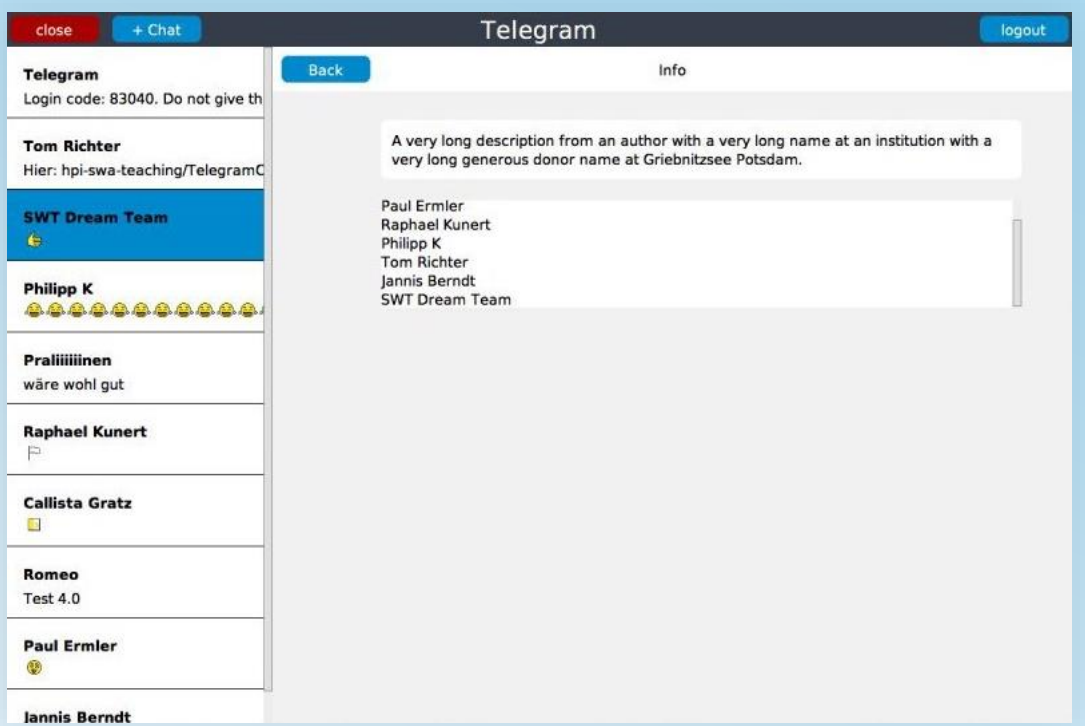
Anforderung: UI ähnlich zum Original



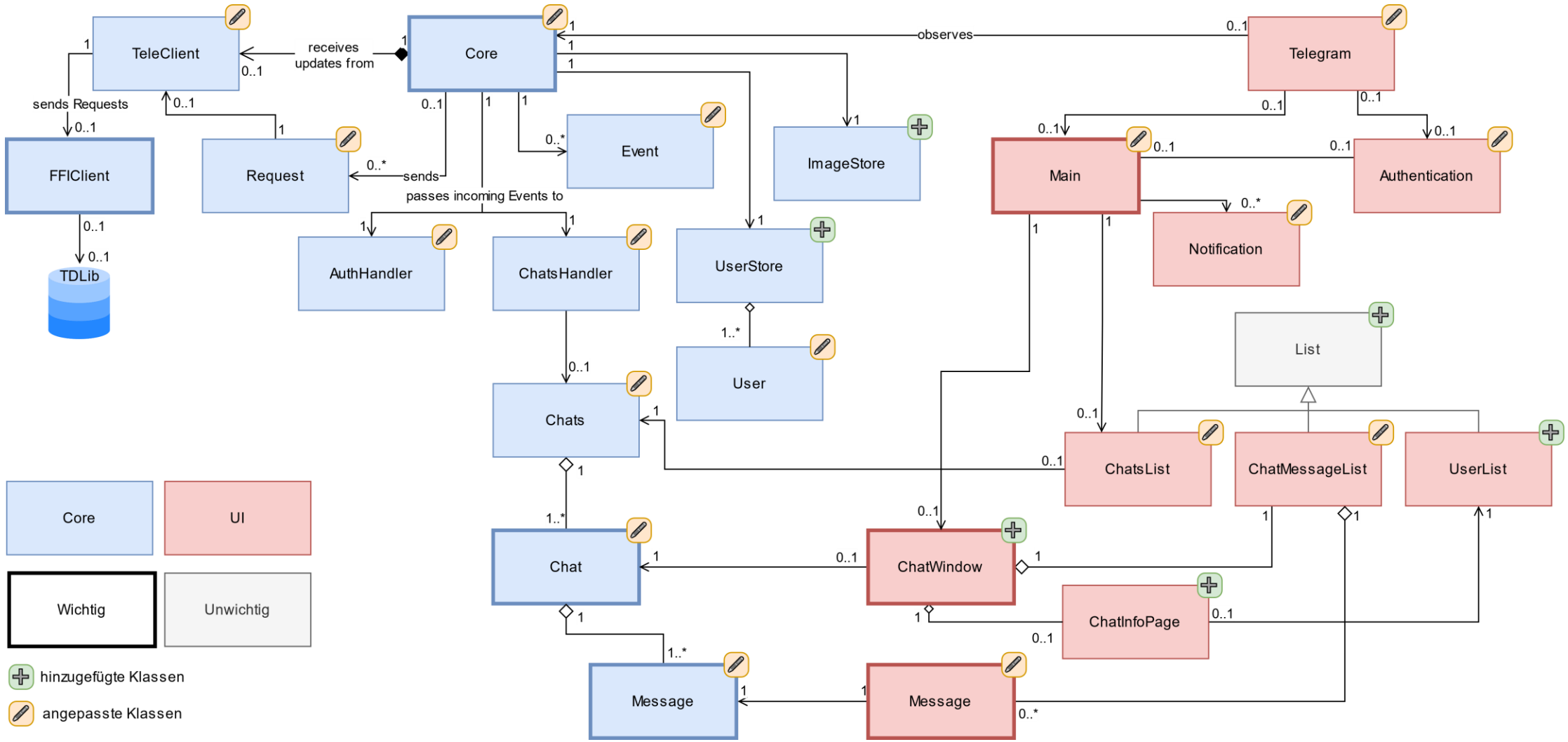


Anforderung: UI ähnlich zum Original

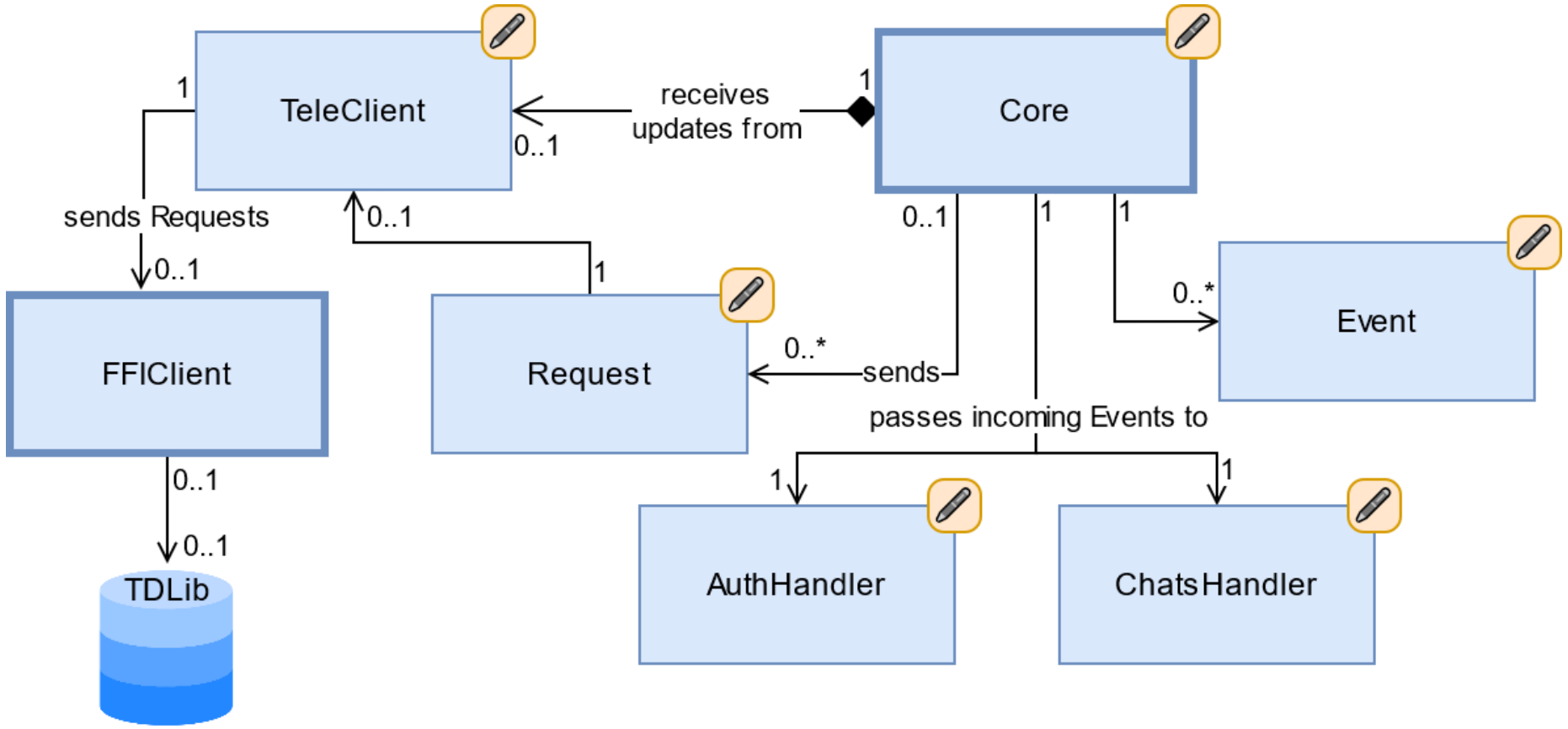




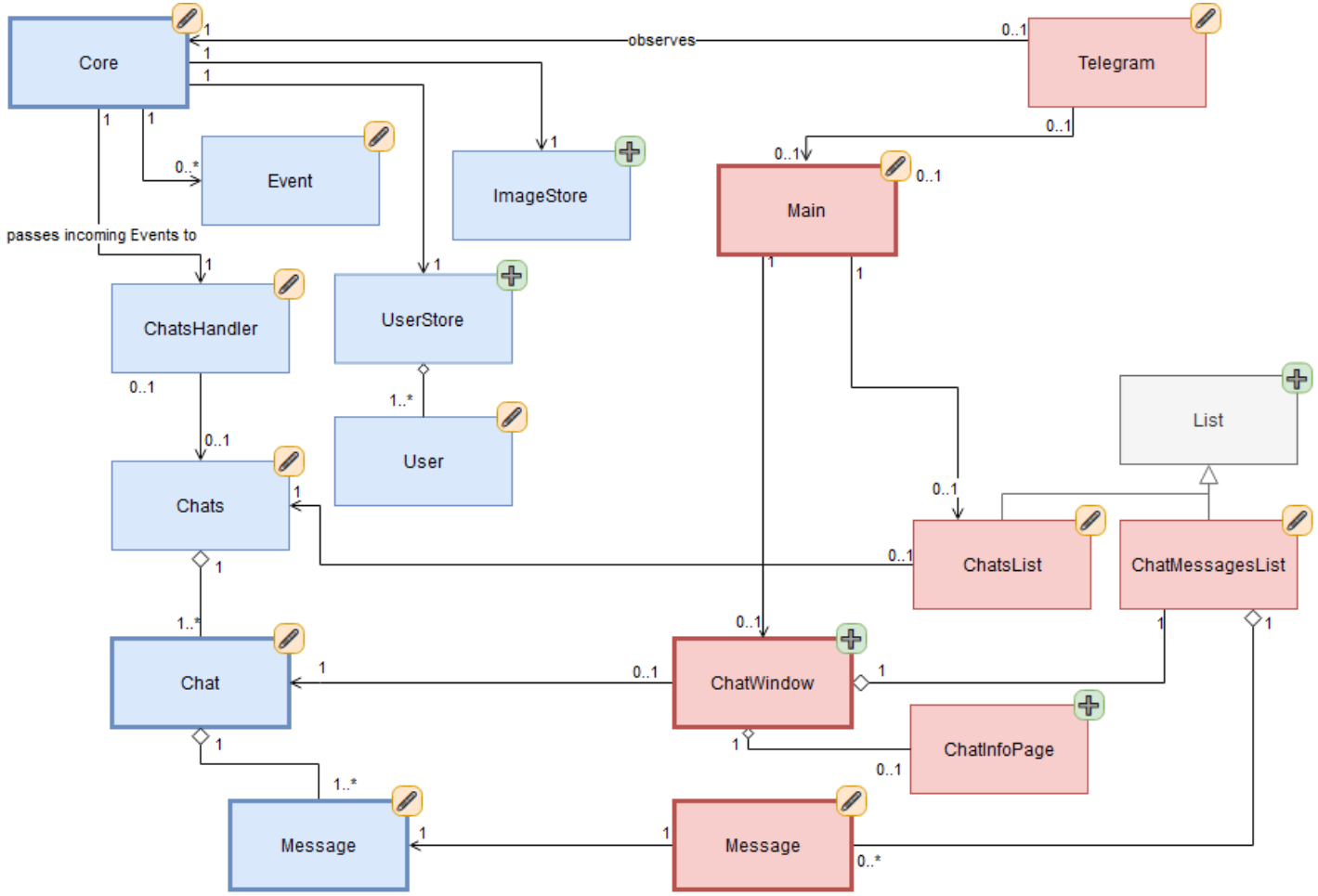
Architektur


TeleClient
receives updates from
Core
observes
Telegram
sends Requests
FFIClient
Request
sends
Event
ImageStore
passes incoming Events to
TDLib
AuthHandler
ChatsHandler
UserStore
User
Chats
Main
Notification
Authentication
List
ChatsList
ChatMessageList
UserList
Chat
ChatWindow
ChatInfoPage
Message
Message
Core
UI
Wichtig
Unwichtig
hinzugefügte Klassen
angepasste Klassen

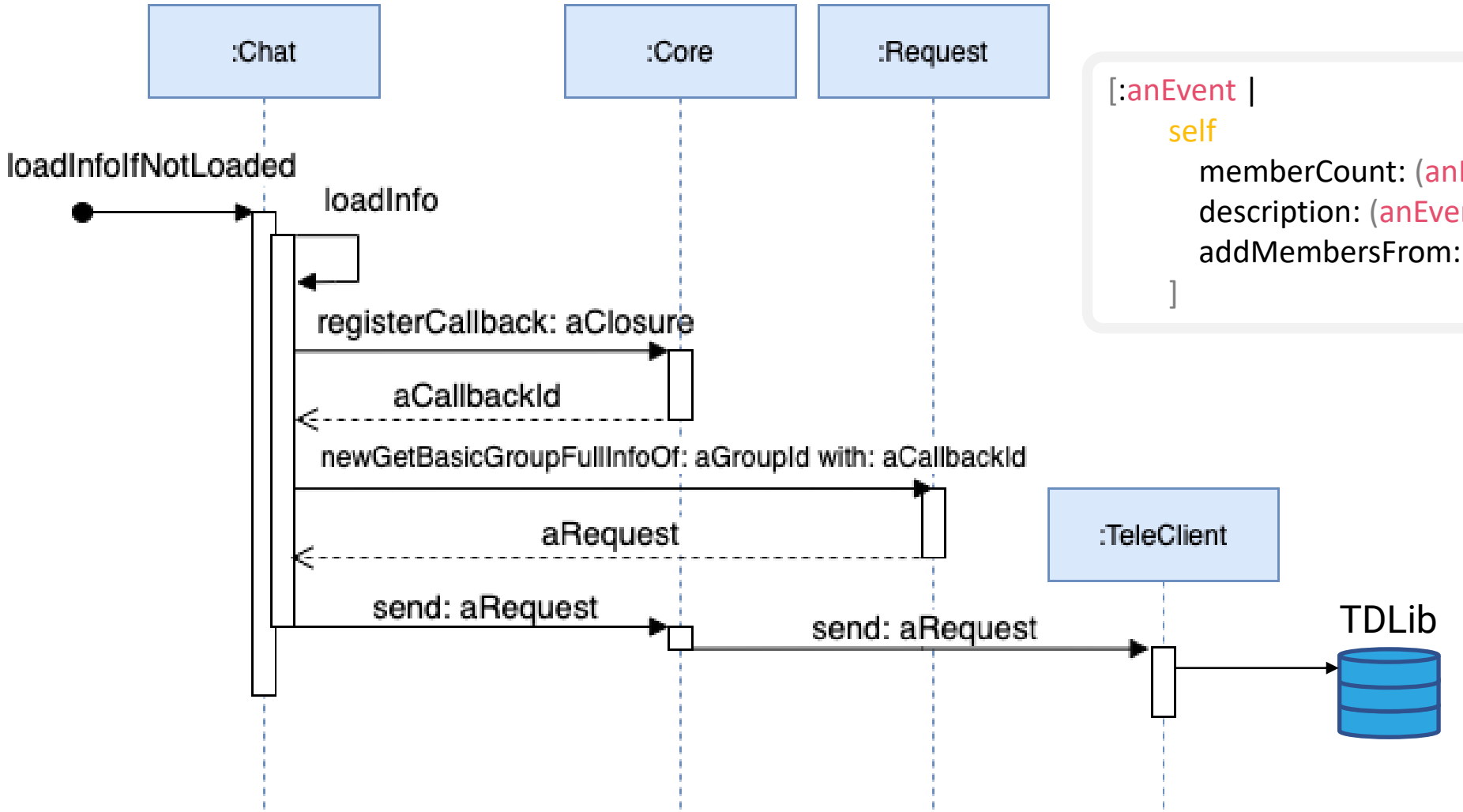
TDLib Kommunikation



Kommunikation Core - UI



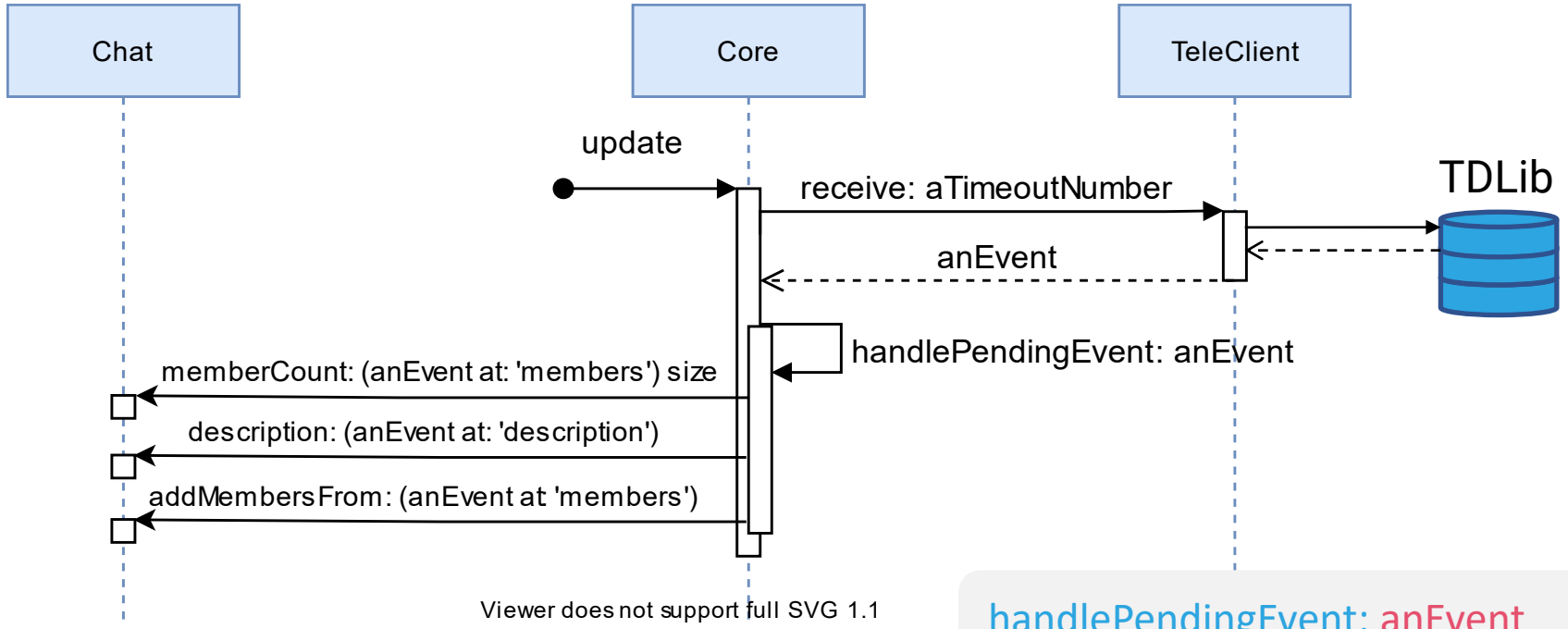
Nachladen von Informationen




```
[:anEvent |
    self
        memberCount: (anEvent at: 'members') size;
        description: (anEvent at: 'description');
        addMembersFrom: (anEvent at: 'members').
    ]
```
*

* Variable anEvent wurde zur Übersichtlichkeit umbenannt

Nachladen von Informationen



handlePendingEvent: anEvent

```
| callbackId |
callbackId = anEvent at: '@extra'.
(self pendingRequests at: callbackId) value: anEvent.
self pendingRequests removeKey: callbackId ifAbsent: []
```

Entwicklungsprozess

Ausgangsbedingungen

- Aus SWA bekanntes Team (5/6)
- Erfahrung in der Entwicklung ohne Pair Programming
- Erfahrungen als Telegram Nutzer, aber nicht mit der API

Technische Legacy

- Rote Pipeline auf dem Master
- Projekt lief nicht unter MacOS (2 / 6 Developer nutzen Macs)
- Ungetestetes UI Package (und nicht fehlerfrei)

I. Kundentreffen

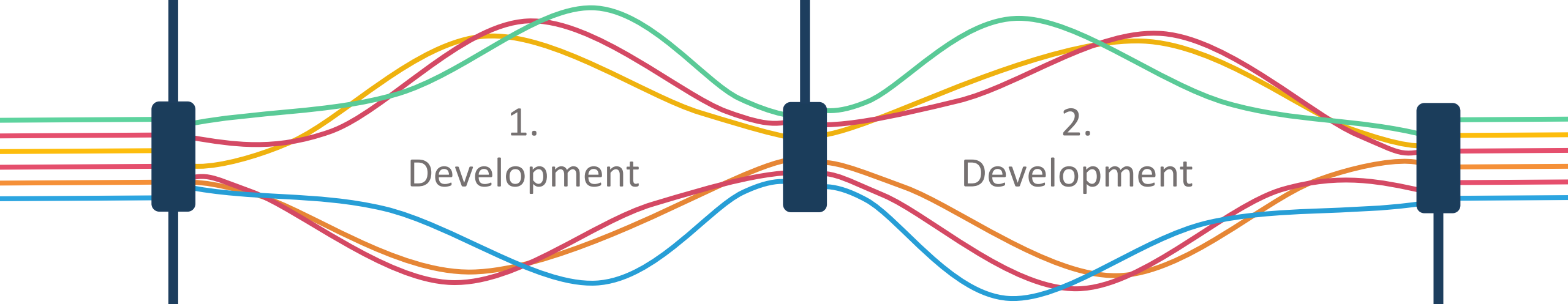
Tag 01

- Features präsentieren
- Neue User Stories wählen

II. Planungsmeeting

Tag 01

- Aufteilung der Features
- Teamaufteilung



III. Teammeeting 1

Tag 05

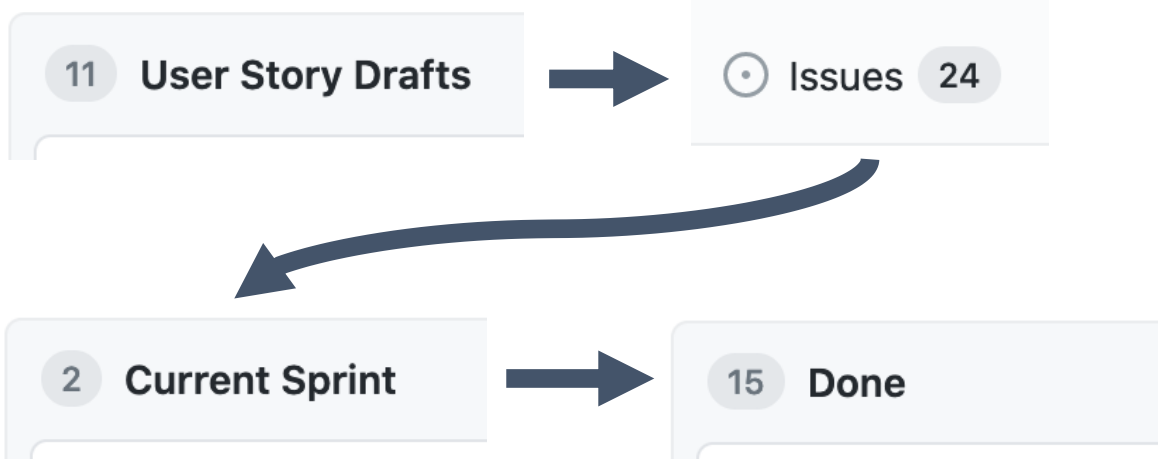
- Diskussion technischer Fragen
- Fortschritt vorstellen
- Koordination von Features

IV. Teammeeting 2

Tag 12

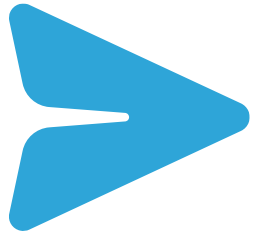
- Abstimmung der PRs
- Organisatorische Fragen
- Schätzen (Planning Poker)

Kommunikation



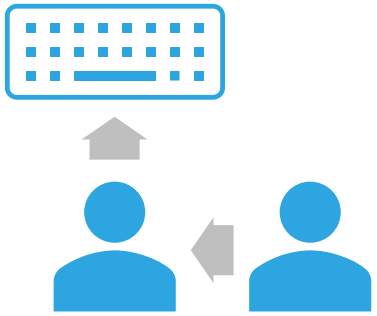
Praktiken im Detail

Pair Programming

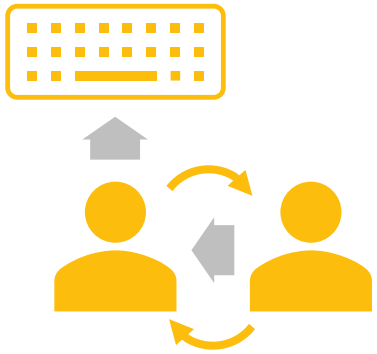


Pair Programming Styles

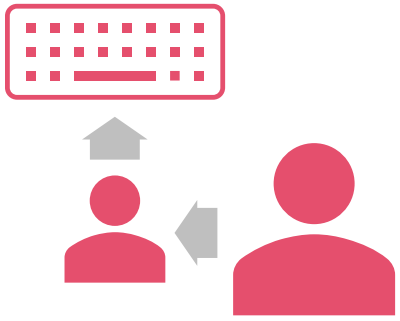
Traditional
Driver-Navigator



Ping Pong Style

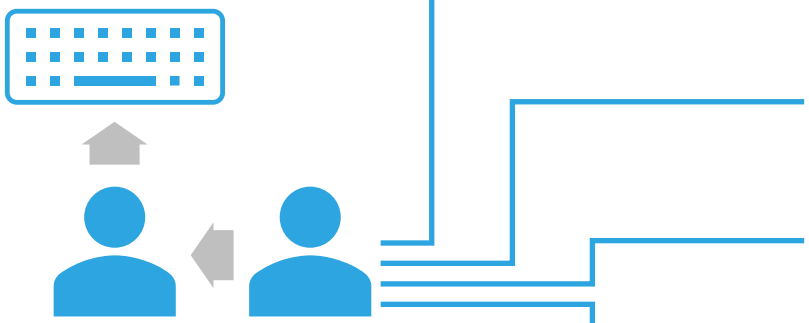


Strong Style



https://martinfowler.com/articles/on-pair-programming.html

Driver - Navigator: Aufgaben in unserem Projekt



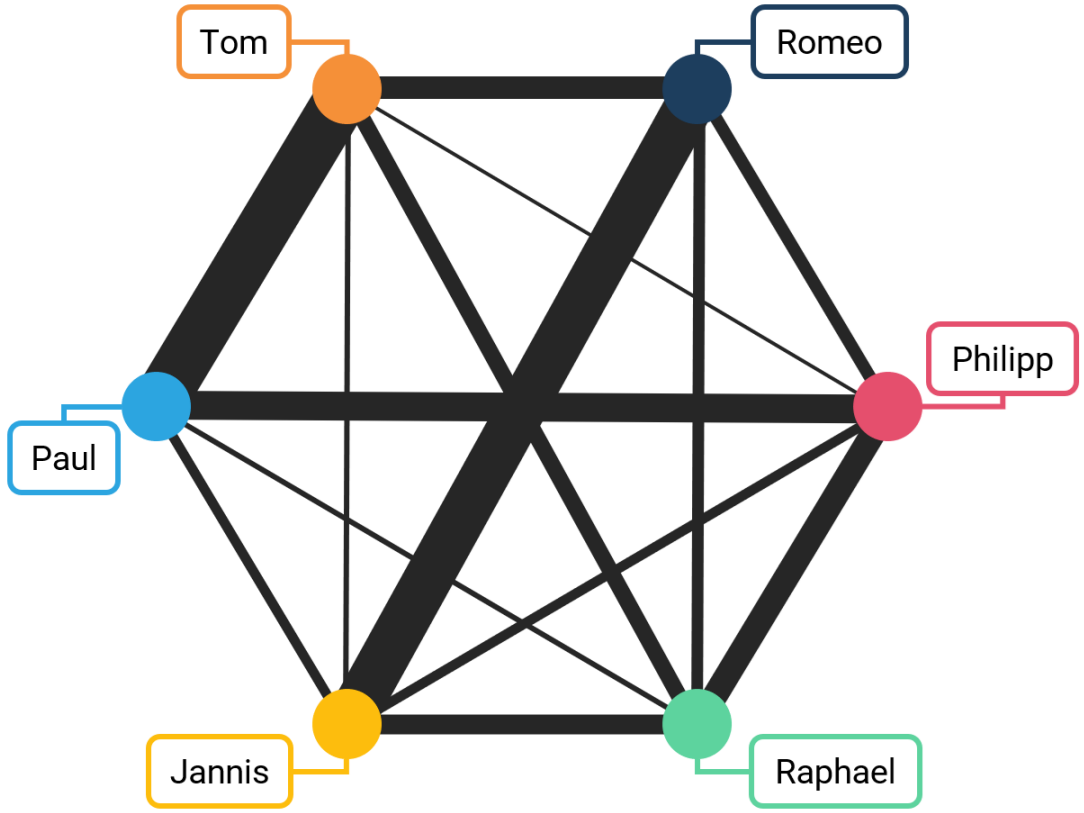
- Hat alten Code zum Vergleich offen
- Infos in Telegram API Dokumentation nachlesen
- Hilfreiche Klassen, Methoden finden
- Code im Workspace ausprobieren

Partnerwahl

Personen mit Vorwissen aus unterschiedlichen Bereichen

Fast Jeder mit jedem einmal gearbeitet

Cluster erkennbar



Pair Review: Emojis 😍

2:30h

<> code </>

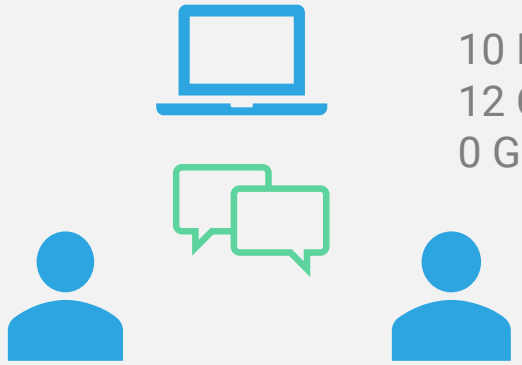
3 Linter Anmerkungen
4 Code Smells
1 Größeres Refactoring

1:20h

2:15h

3:20h

<> code </>



10 Linter Anmerkungen
12 Code Smells
0 Größere Refactorings

2 x 4:00h

Wofür eignet sich Pair Programming (nicht)?

- Neues Thema
- Kreative Aufgaben
- Diskussionspotential
- Specific Knowledge Unterschiede
- Sehr gute Code-Qualität

- Recherche
- Einfache Aufgaben
- Pull Request Reviews

Abwägung im Einzelfall, ob Pair Programming geeignet ist.

Specific/General Knowledge: https://dl.acm.org/doi/pdf/10.1145/3377811.3380925

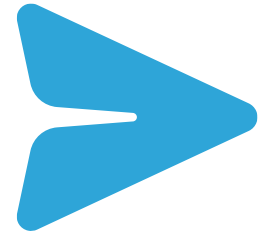
Vergleich mit SWA

SWA		SWT
Wenig Pair Programming		> ⅔ der Zeit Pair Programming
Stressiges Refactoring kurz vor Abgabe		Vertrauen in Code, Abgabe jederzeit möglich
Manche Teile nur von wenigen Personen gut verstanden		Weniger Code Ownership
Lange Meetings um Code vorzustellen		Reviews vor Mergen in Develop

Pair Programming Fazit

- Besseres Verständnis vom System / Shared Code Ownership 
- Mehr Spaß am Projekt 
- Bessere Code Qualität 
- Mehr Zeit pro Feature notwendig 
- Weniger Ablenkungen 

Continuous Integration



Continuous Integration

Motivation

Objektive Kriterien für Bewertung von Code

Build Quality In: nur funktionierender Code auf dem Develop Branch

+ Fehlerquelle lokalisieren
+ Weniger Debugging

Kein manuelles Testing

+ Konsistent Edge Cases prüfen
+ **Reproduzierbarkeit von gefundenen Fehlern**
+ **Zeitersparnis**

Continuous Integration - Umsetzung

Feature Branches
für **isolierte** Entwicklung, Reviews und Tests

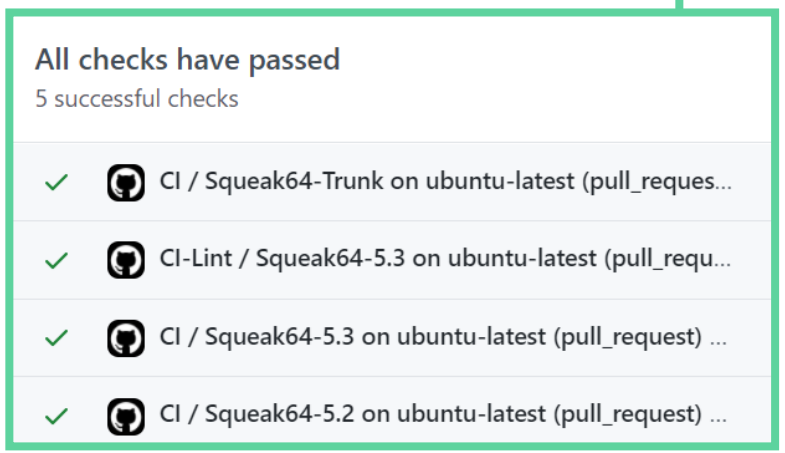
Pull Requests
als offizieller **Abschluss** eines Features

Linter Tests
für einheitliche Code **Formatierung**

Test Pipelines
zur **Sicherstellung** der Korrektheit

Approval
nach vollständigem Review

Grüne Pipelines



Continuous Integration - Fazit

Prozess

Einheitliche Code Standards durch Review und automatisierte Lintertests

Mehr Personen kennen jeden Codeabschnitt durch Reviews

Gelegentliche Merge Konflikte durch Feature Branches

Aufwendiges Reparieren der Linterregeln

Automatische Test-Suite:

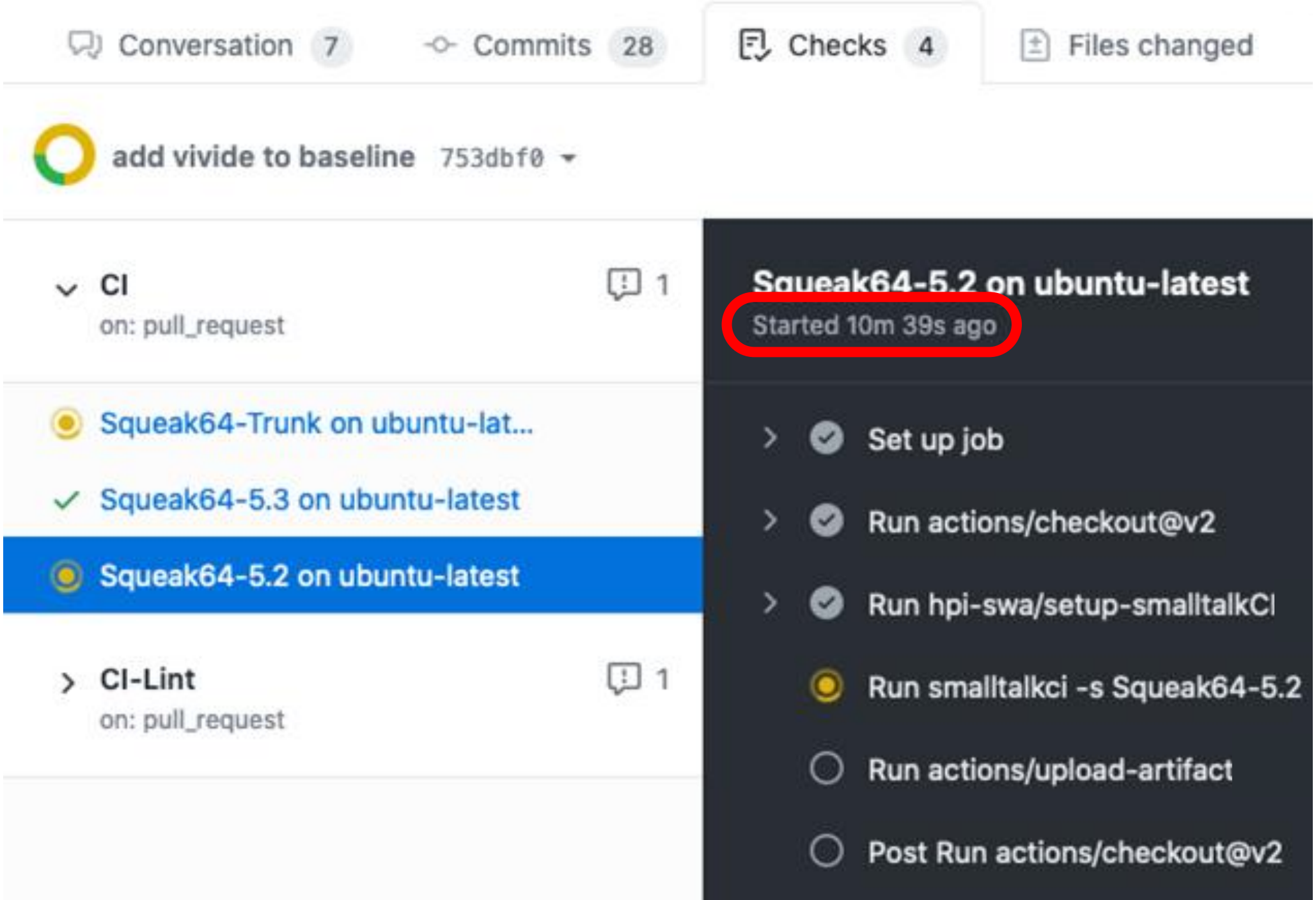
Vertrauen in den Code bei grünen Pipelines

Nur sehr selten Debugging nach dem Mergen nötig

Reviewer können sich auf Code Quality und inhaltliche Fragen konzentrieren

Reproduzierbarkeit von gefundenen Fehlern

Zeitersparnis


Conversation 7
Commits 28
Checks 4
Files changed
add vivide to baseline 753dbf0
CI
on: pull_request
1
Squeak64-Trunk on ubuntu-lat...
Squeak64-5.3 on ubuntu-latest
Squeak64-5.2 on ubuntu-latest
CI-Lint
on: pull_request
1
Squeak64-5.2 on ubuntu-latest
Started 10m 39s ago
Set up job
Run actions/checkout@v2
Run hpi-swa/setup-smalltalkCI
Run smalltalkci -s Squeak64-5.2
Run actions/upload-artifact
Post Run actions/checkout@v2

Continuous Integration - Herausforderung

Flaky Tests [1], Erfolg von äußeren Umständen abhängig

Neustarten der Pipeline führt oft zu anderen Ergebnissen

Verzögerung von Pull Requests aufgrund roter Pipelines bei lokal grünen Tests

Pipeline Laufzeit von über 10 min

> *Keep an eye on build times and **take action** as soon as you start going **slower than the ten minute rule.***
>
> Martin Fowler

[1] verschiedene Ergebnisse bei wiederholter Ausführung

Ursachen

```
259  [ 2][t 3][1622403336.782427549][NetQueryDelayer.cpp:82][#15][!NetQueryDelayer]  Delay: [Query:[id:6160384]
     [tl:0xd4982db5][state:Query]] [timeout:5.000000][total_timeout:5.000000] because of [Error : 420 : FLOOD_WAIT_5]
     from Session:1:main::Connect::Tcp::[149.154.175.10:80] to DcId{1} from [10.1.0.89:60454]
260  [ 2][t 3][1622403336.791004657][NetQueryDelayer.cpp:82][#15][!NetQueryDelayer]  Delay: [Query:[id:6094848]
     [tl:0xa0ee3b73][state:Query]] [timeout:5.000000][total_timeout:5.000000] because of [Error : 420 : FLOOD_WAIT_5]
     from Session:1:main::Connect::Tcp::[149.154.175.10:80] to DcId{1} from [10.1.0.89:60454]
```

Jeder Test verbindet sich im setUp neu mit dem Telegram Test Server

Network Overhead

DDoS Protection

Tests mit Zufallskomponente in der bestehenden Testbasis

z.B. nur grün, falls im ersten Chat auf dem Server Schreibrechte gegeben

Lösungsansätze

1 Telegram Test-Datacenter

Wiederverwendung eines Clients durch eine

TestResource

Viele Tests benötigen nur den eingeloggten Zustand

-> ensureLoggedIn muss sich nicht neu anmelden

```
setUp
    self sharedResource: TCTCCoreResource current.
    self sharedResource ensureLoggedIn.
```

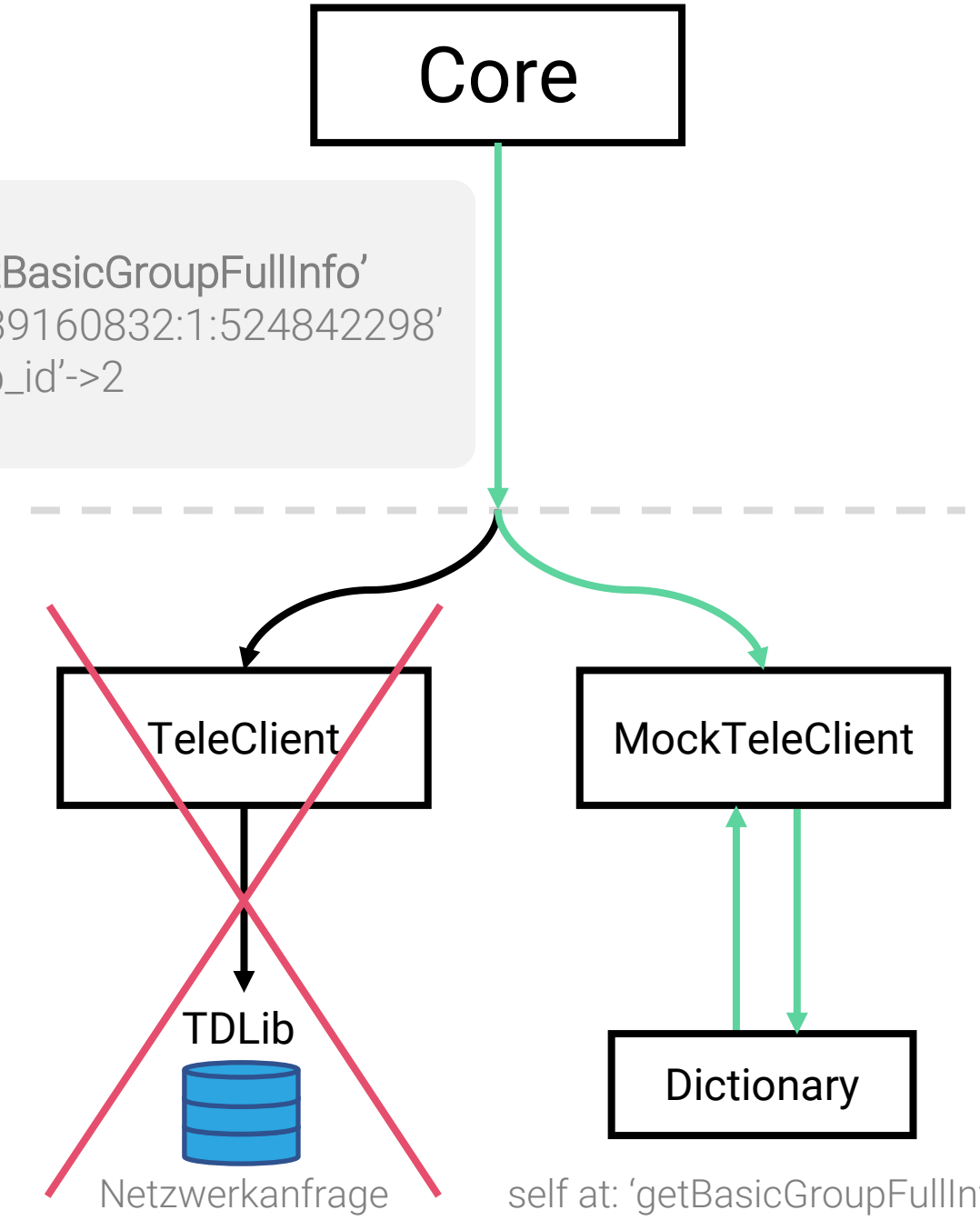
- Nur noch 2 Loginvorgänge für die gesamte Testbasis nötig
- Weniger setUp Code, da Umgang mit TestCore und Server gekapselt

2. Mocking

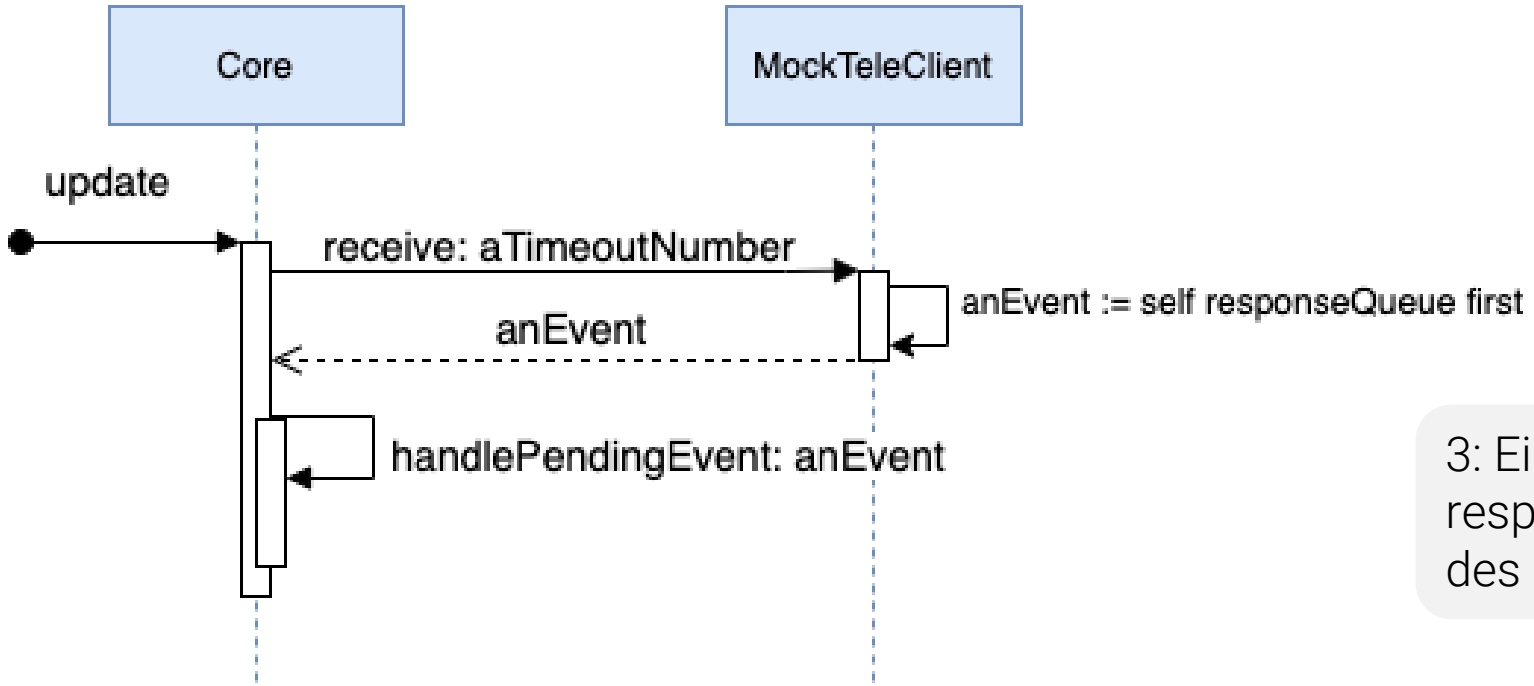
```
a TCCRequest(
        '@type'->'getBasicGroupFullInfo'
        '@extra'->'439160832:1:524842298'
        'basic_group_id'->2
)
```

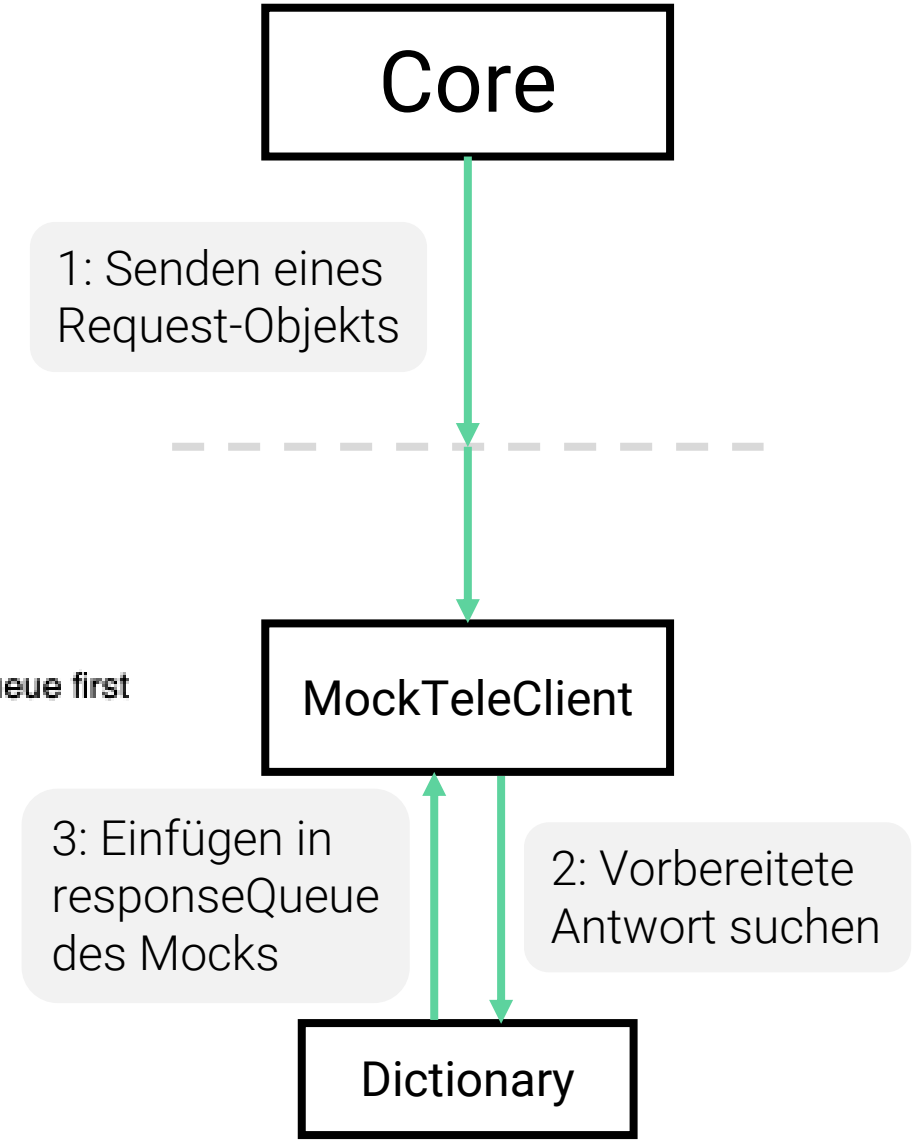
Mock across architecturally significant boundaries, *but not within those boundaries.*

Uncle Bob



2. Mocking





self at: 'getBasicGroupFullInfo'

Lösungsansätze

2 Mocking

- Um Auswirkung einer einzelnen Nachricht zu testen, oft keine echte Verbindung nötig
- Präzises Testen eines Features mit sehr geringen Kosten
- Auch Core-Features können getestet werden, da nur unterhalb von Core gemockt
- Einfaches und mächtiges Interface

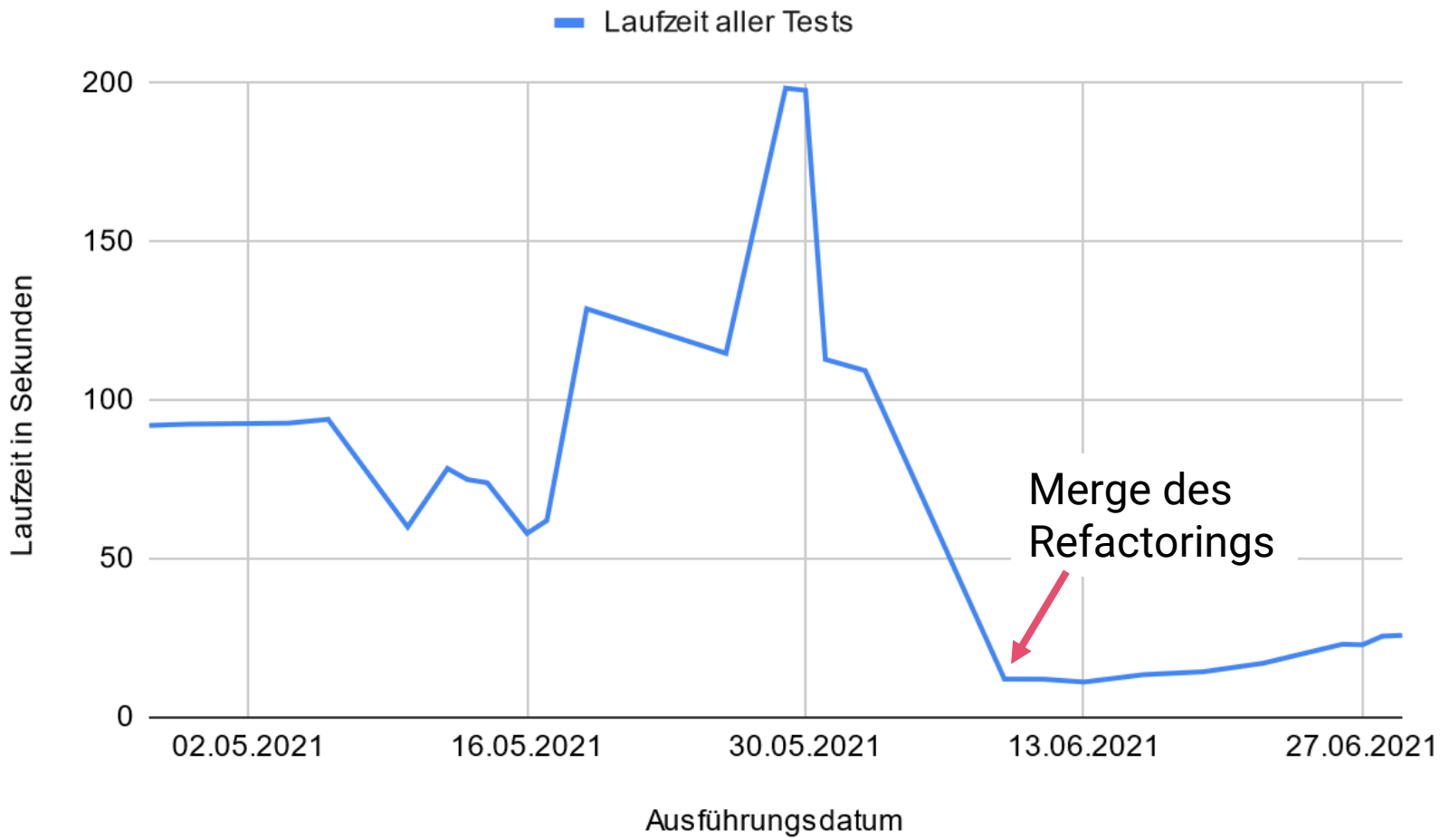
testCachesRequests

```
self mockTeleClient onRequestType: 'getUser'
    respond: TCTMMocks mockUserJson1.

"[... request a user and make assertions]"
```

Eingesparte Nachrichten, die sonst zur Anmeldung nötig sind

Auswirkung



Erfolgreiche Pipelines (ohne Lintertests) auf dem Develop

Auswirkung

konsistente, reproduzierbare Testergebnisse

schnelle Pipeline

mehr Vertrauen in den Testprozess durch weniger Zufall

```
Executed 93 Tests with 0 Failures and 0 Errors in 23.09s.
```

Metriken

Kennzahlen

> 377 Stunden Arbeitszeit

LOC*
4435 (2113 neu)

Längste Session:
6 Stunden

12 Metriken auf dieser Folie

59 Commits auf Development

15 User Stories

77 Klassen

Kumulierte Pipeline Laufzeit
> 15h in 564 runs

4,5 Sprints

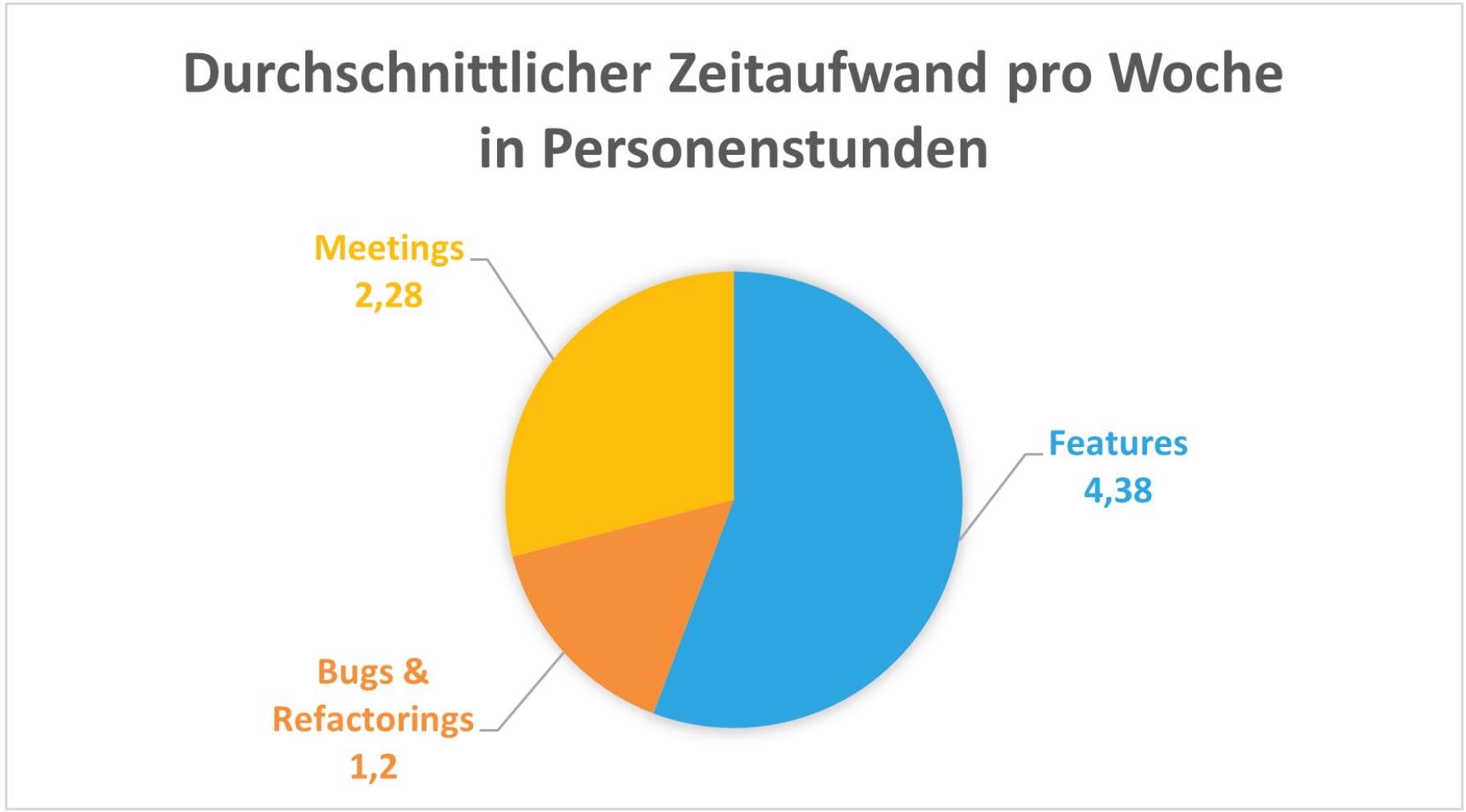
Programmier-Sessions
> 100

90,21% Coverage

128 Tests

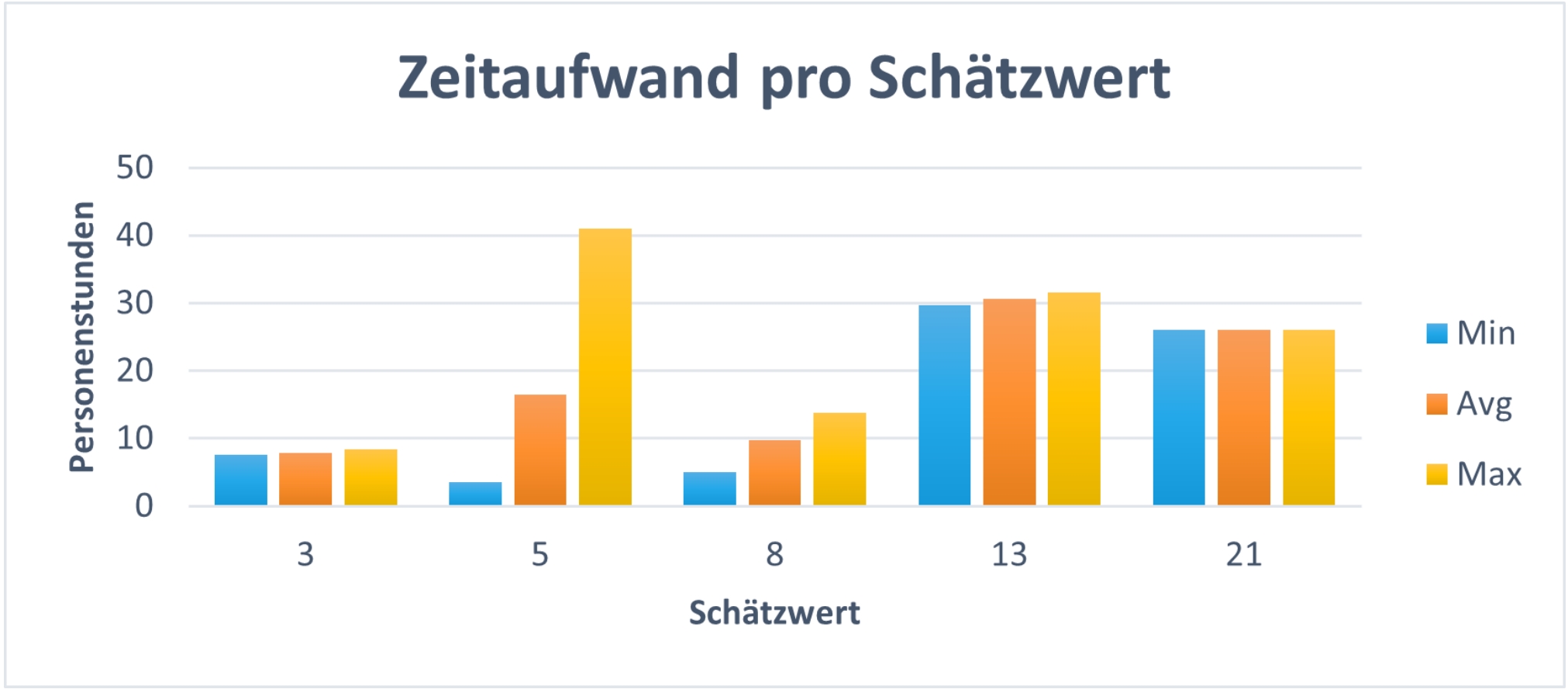
* Ohne leere Zeilen

Zeitaufteilung

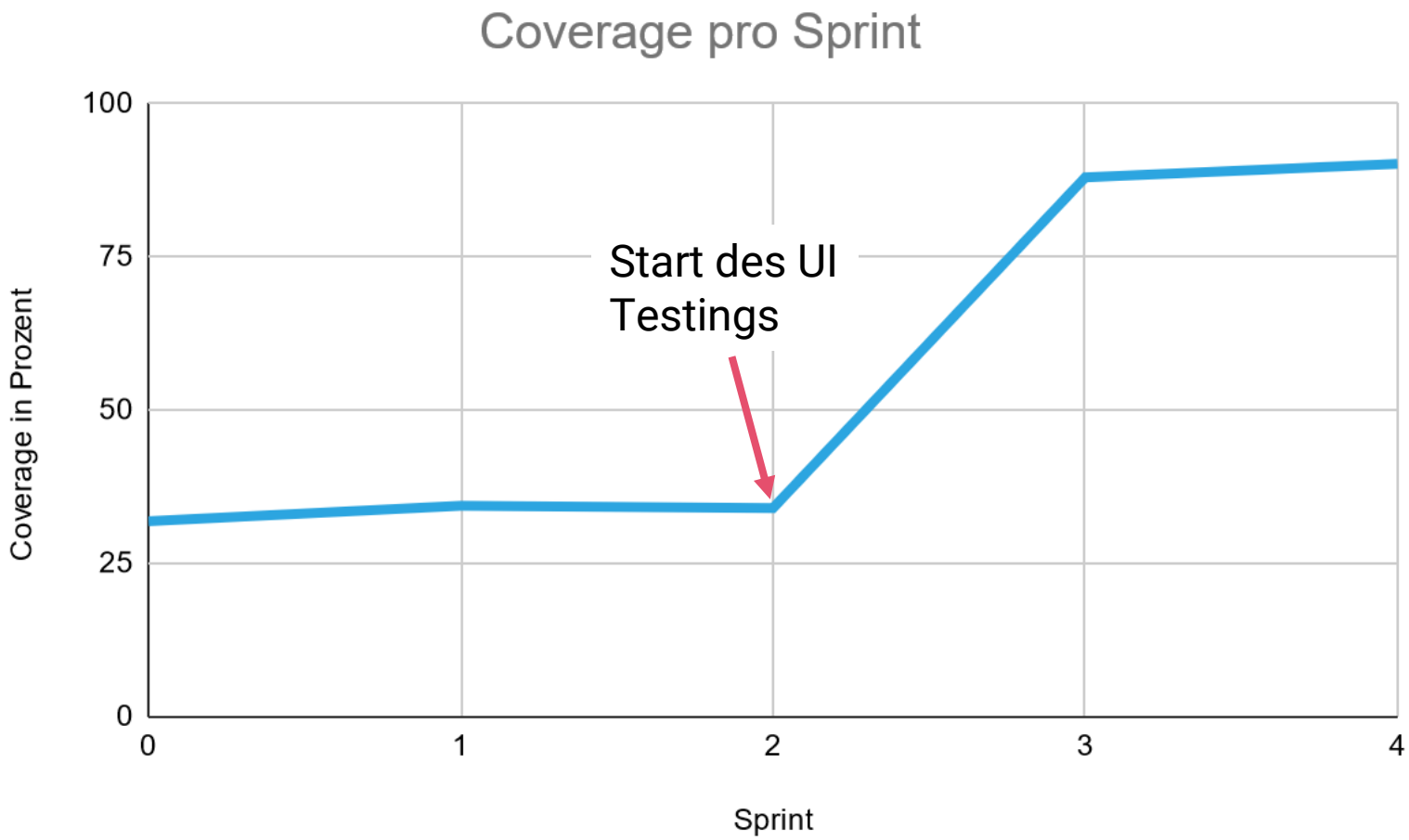


(Ver-)Schätzen

1 Sprint
=
34 Punkte



Testen


Coverage pro Sprint
100
75
50
25
0
Coverage in Prozent
Start des UI Testings
0
1
2
3
4
Sprint

Reflexion

Was hat gut geklappt?

Vor Projektbeginn **Codebase** anschauen, vorheriges Team anschreiben

Wöchentliche **Meetings** mit ≈90 min effizient, strukturiert

Teamarbeit

Einarbeitung in **Telegram API**

Spaß am Projekt

Was würden wir das nächste Mal anderes machen?

Pull Requests zeitnah reviewen, mergen, sodass **abhängige Aufagen** nicht verzögert werden, Mehraufwand beim Mergen entsteht

Experimente zu Praktiken mit **Projektbeginn** starten

Metriken zu Beginn klar definieren und **konsequent tracken**

Was haben wir gelernt?

API Testing schwer ohne gute Infrastruktur

Gute Metrik für **Produktivität** notwendig zur Erstellung von nützlichen Statistiken

Vor dem Schätzen von **User Stories** Gedanken über Umsetzung machen