

25.06.2020



# Gruppe 13: Telegram-Client in Squeak

Softwaretechnik I, Sommersemester 2020

Rohan Sawahn, Niklas Schilli, Jonas Schmidt, Frederik Wollny, Stefan Spangenberg, Lukas Laskowski

# Inhalt

1. Projekt- und Zieldefinition
2. Live-Demo und Architektur
3. Arbeitsweise: Agile Softwareentwicklung
4. Projektdurchführung
  - Coding Standards
  - Continuous Integration
5. Retrospektive
6. Ausblick

# Projekt- und Zieldefinition

# Projektdefinition

## Entwicklung eines Telegram-Clients in Squeak

- keine Nutzung eines **Legacy-Projekts**
- **TDLib** für Kommunikation mit Telegram
- **UI** und **Logikebene** auf TDLib aufbauend



# Funktionale Ziele

## Must have:

Versenden von  
Nachrichten



Anzeigen von  
Chats &  
Nachrichten



Authentifizieren



User-Interface  
minimalistisches Design



## Should have:

Persistenz

Zen-Mode

Versenden von  
Medien

Push-Notifications

Code-Snippets  
pinnen

Kontakt-  
management

Textsuche in  
Chats

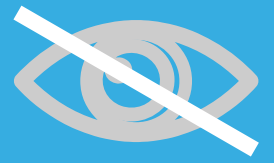
# Nicht-funktionale Ziele

- Installation der **Anwendung**
  - Automatisierte Installation der TDLib
- **Lauffähig** unter Ubuntu, Mac OS, Windows
- Zweiwöchige **Releases**
- Einhaltung unserer **Clean-Code Guidelines**
- **Evergreen** Develop- und Master-Branch



**\*Live-Demo\***

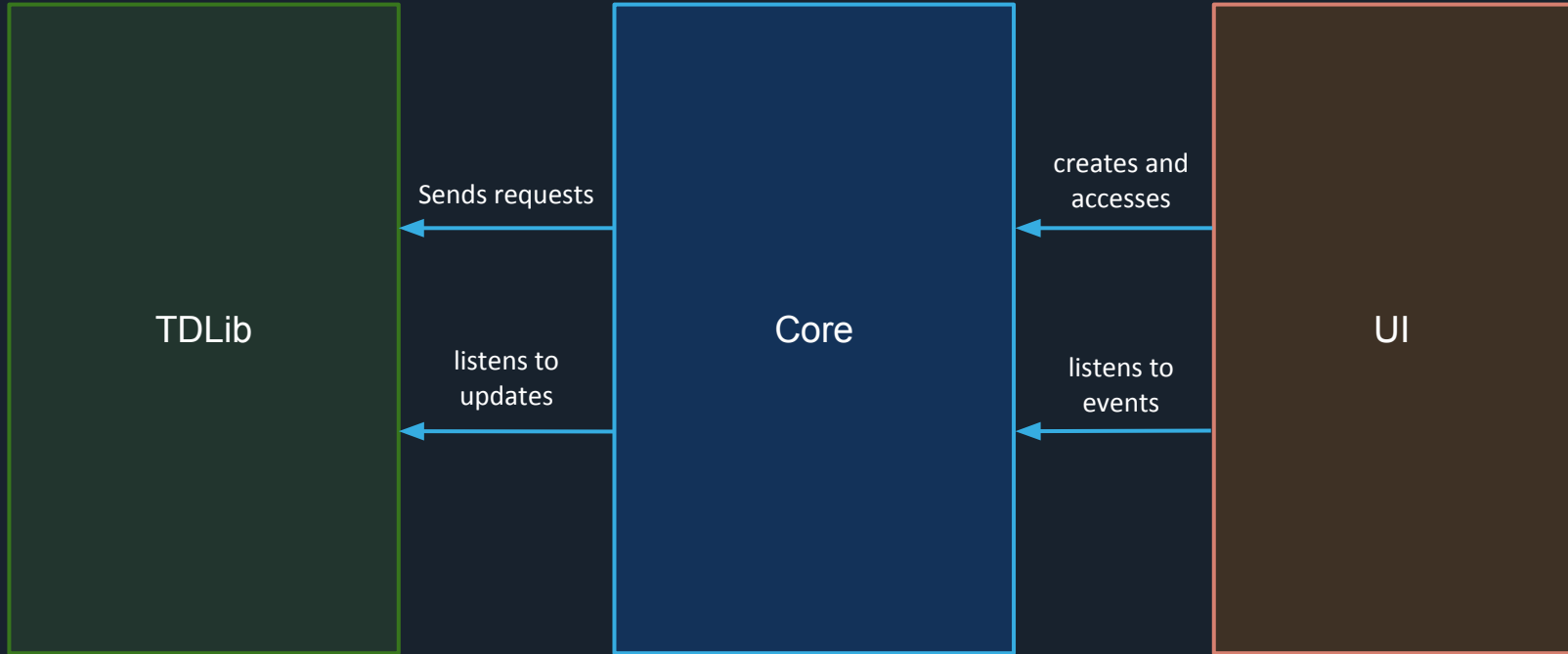




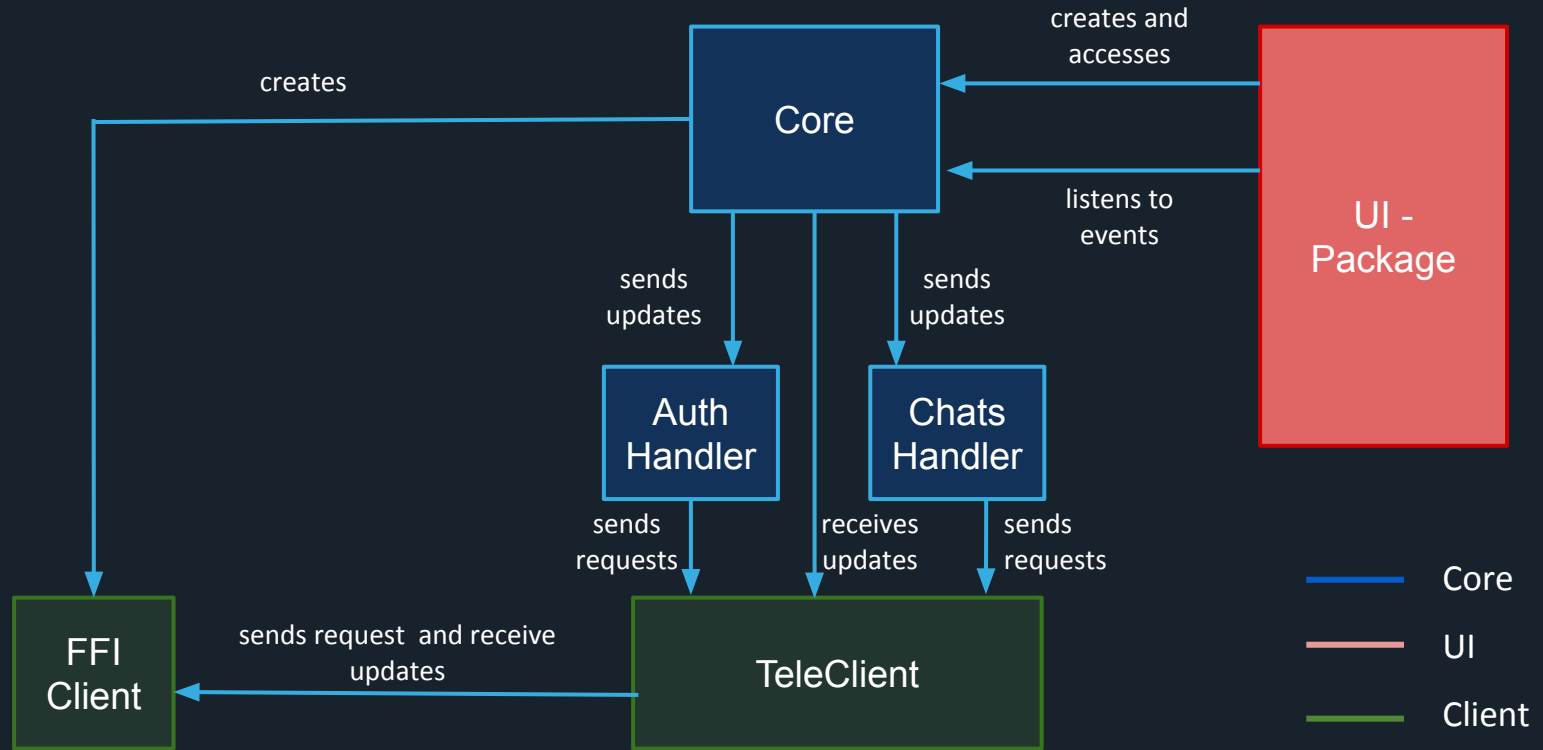


**Architektur**

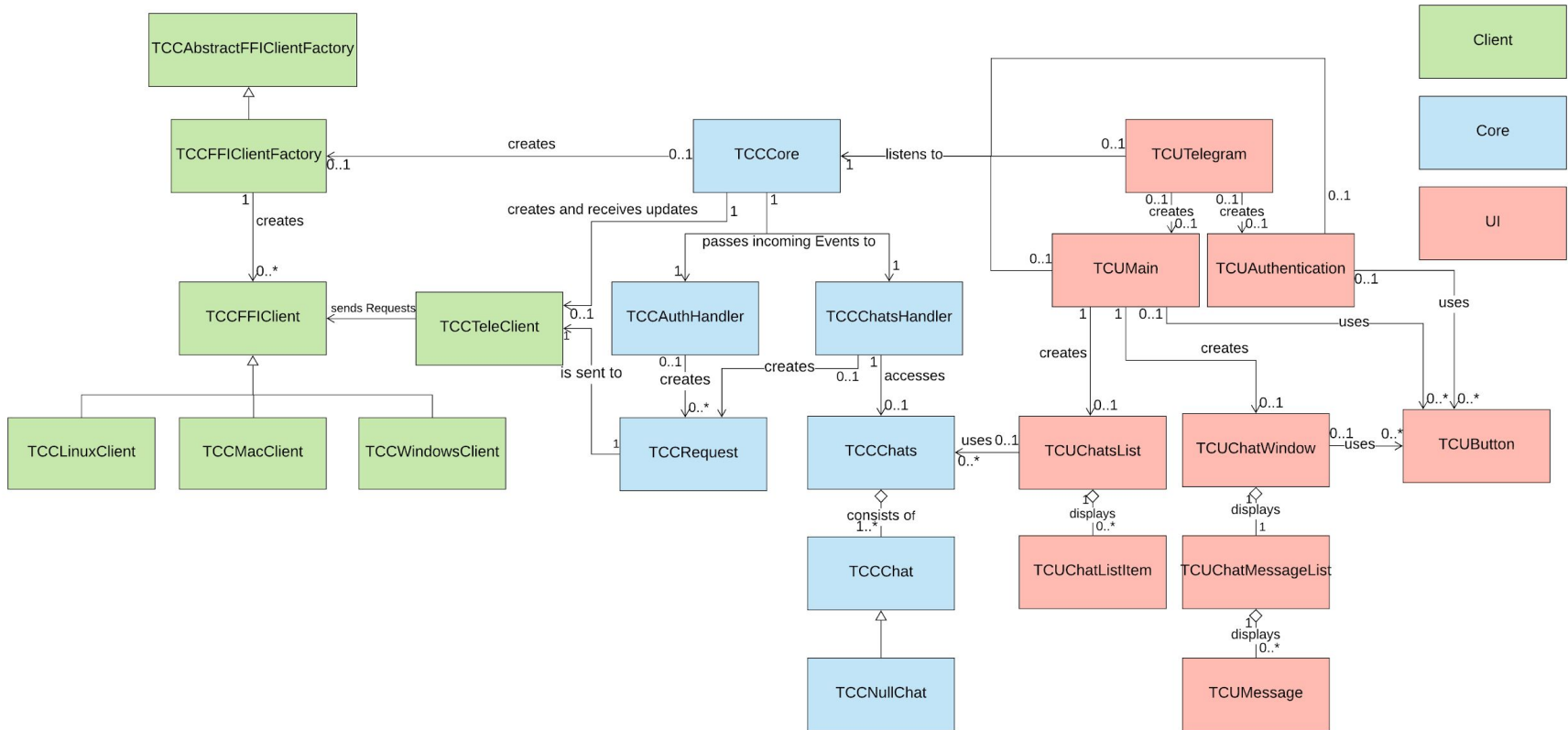
# Architektur - ein Überblick



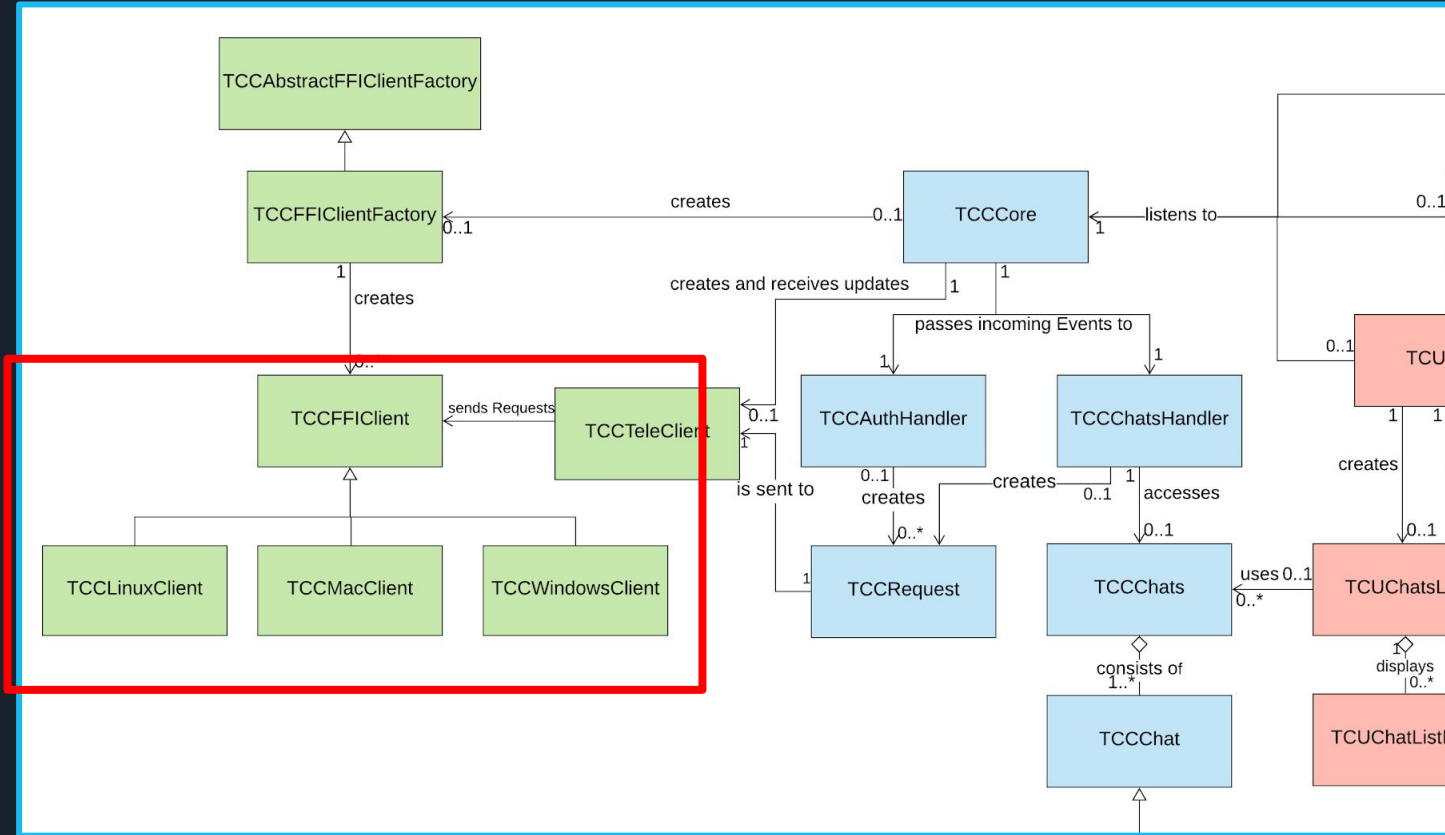
# Vereinfachte Architektur



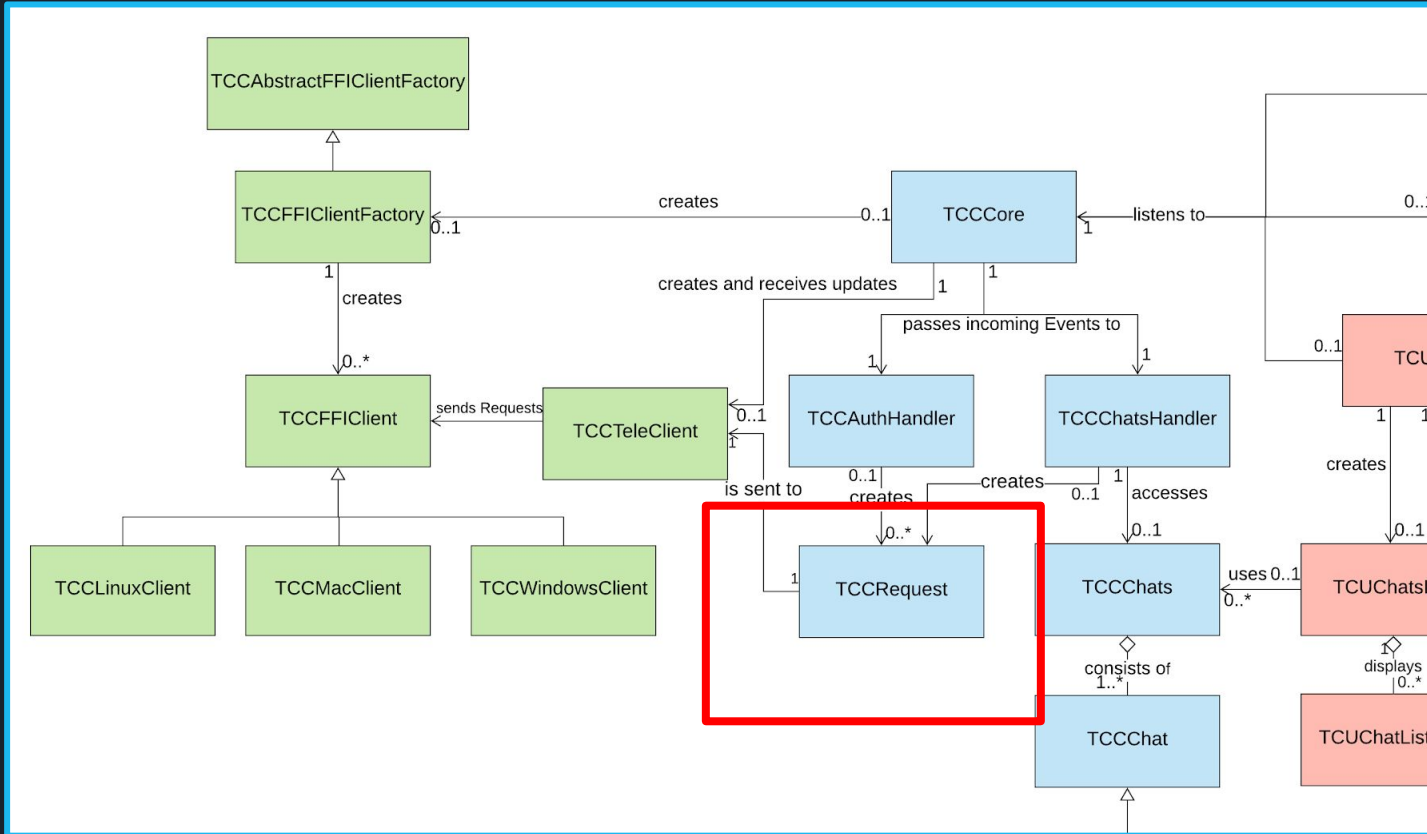
# Gesamtarchitektur



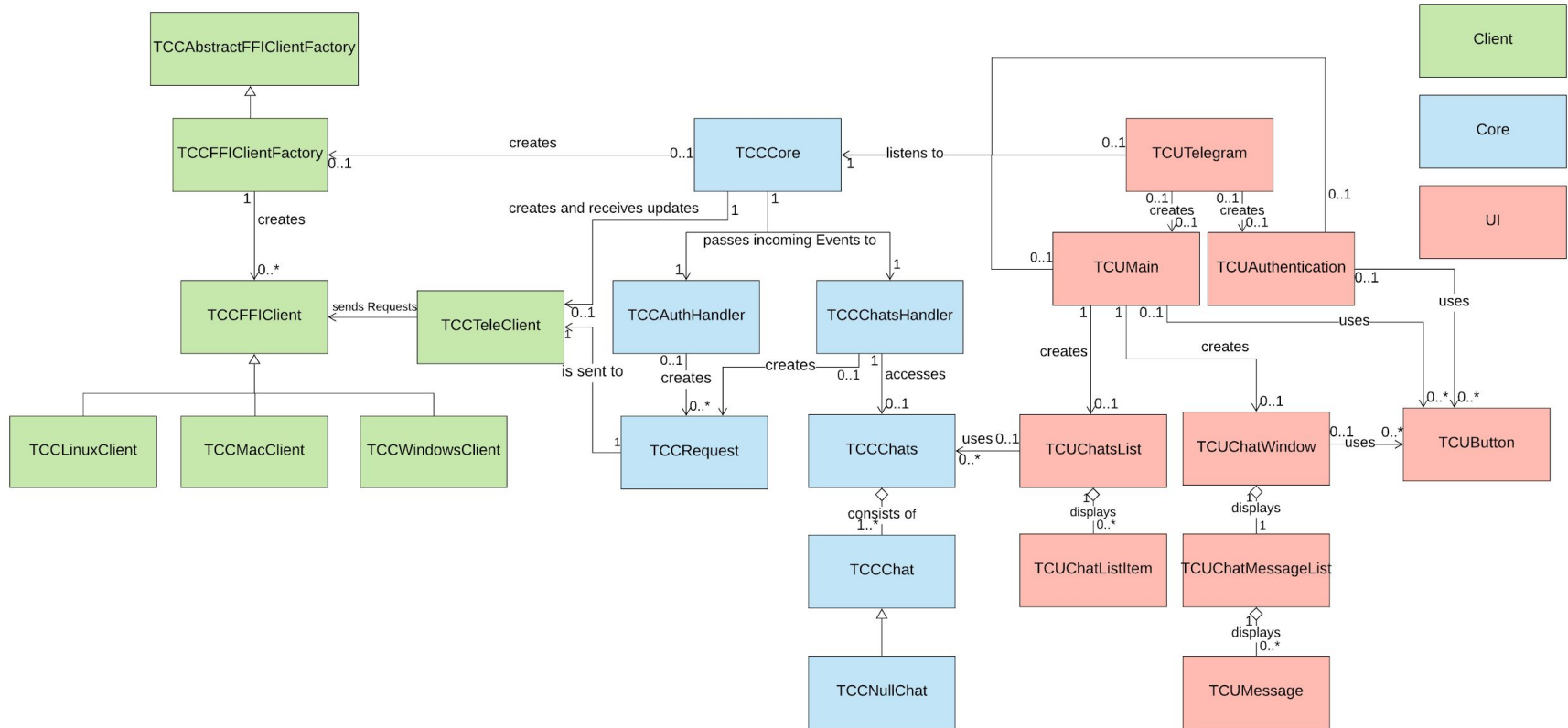
# Gesamtarchitektur



# Gesamtarchitektur



# Gesamtarchitektur



Arbeitsweise:

**Agile Softwareentwicklung**



# Entwicklungsprozess



Unser Entwicklungsprozess ist eine agile Entwicklungsart. Er besteht im Kern aus zweiwöchigen Sprints, während dieser wir vom Kunden ausgewählte UserStories bearbeitet haben.

Jeder Sprint startet mit dem "SprintPlanning". Hier wurden UserStories definiert, Aufgaben zugeteilt und Ideen ausgetauscht. Im Anschluss beginnt der eigentliche Sprint. Kernaufgabe dieses Teils ist das effiziente und nachhaltige Umsetzen von den eingangs definierten UserStories. Kommuniziert haben wir mittels Discord, Telegram und GitHub. So stellten wir sicher, dass Probleme gemeinsam angegangen werden können und keiner sich allein gelassen fühlt. Zudem haben wir wöchentliche Meetings angesetzt (Montags und Donnerstags), um unseren Fortschritt, den anderen Teammitgliedern vorstellen zu können, offene Fragen zu diskutieren und auch, um den Teamzusammenhalt zu stärken.

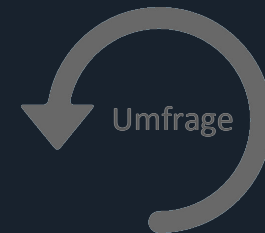
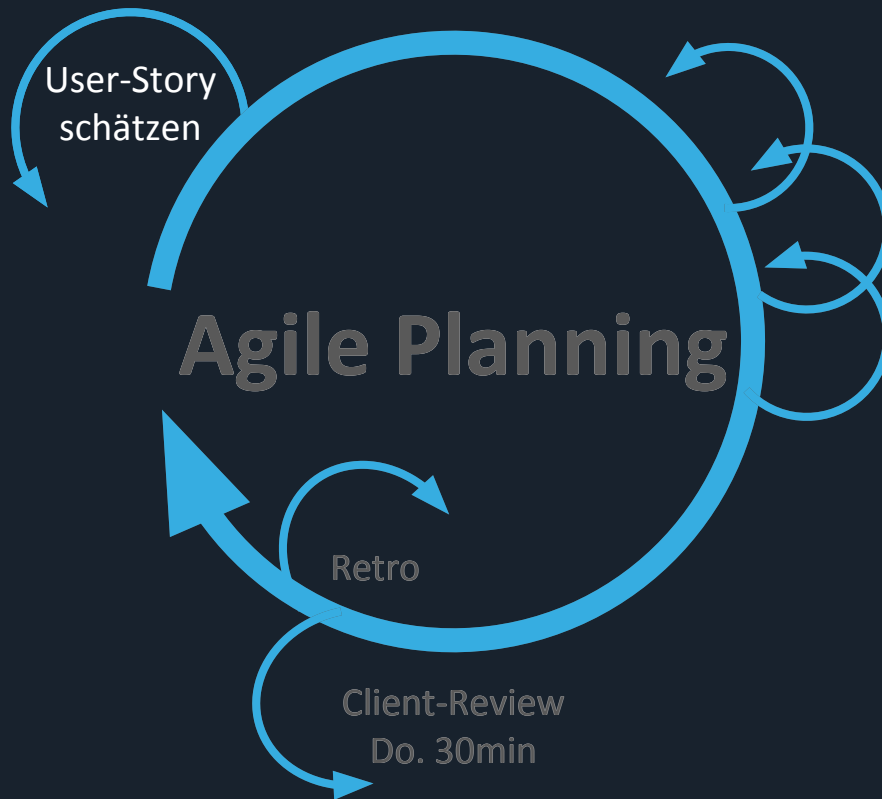
Am Donnerstag hatten wir dann immer unser Kundenmeeting. Um den Kunden stets eine stabile Version präsentieren zu können, vereinbarten wir, dass am Mittwoch immer eine funktionierende Lösung in den MasterBranch gemerged werden sollte. Zudem diskutierten wir vor dem Kundenmeeting kundenrelevante Fragen, wie beispielsweise, welche UserStories dem Kunden im nächsten Sprint angeboten werden können.

Nach dem KundenMeeting stand die Retrospektive an: Hier wurden Probleme offen angesprochen, angegangen und Maßnahmen beschlossen, wie wir das Problem in Zukunft lösen können, damit es nicht mehr vorkommt.

Um unsere Effizienz der Maßnahmen messen zu können, führten wir regelmäßig Umfragen durch. So konnten wir stets anhand von Zahlen, Stellschrauben identifizieren, diese in Retrospektiven ansprechen und anpassen.



4 Pizzen  
Zeiteinheit



alle 4 Wochen

Coding  
Meetings: Mo. & Do.

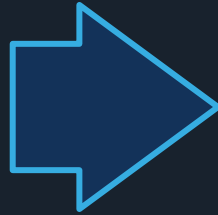




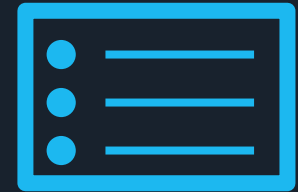
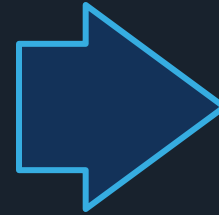
# Bestimmen von User Stories



Planning Poker



Kundentreffen



User Story

# User Stories - Kanban



**Sprint**   
Updated 4 hours ago

Filter cards + Add cards Fullscreen Menu

**11 To do** + ...

- Display push notifications** #16 opened by Mrnikbobjeff  
Blocked User Story enhancement  
📌 Sprint 1 Review
- Allow pinning of messages** #22 opened by Mrnikbobjeff  
Blocked User Story enhancement  
📌 Sprint 1 Review
- Support Push notifications with Telegram** #15 opened by Mrnikbobjeff  
Blocked User Story enhancement  
📌 Sprint 1 Review

Automated as To do Manage

**7 Workable** + ...

- Support Telegram Contact Management** #23 opened by Mrnikbobjeff  
2 User Story enhancement  
📌 Sprint 2 Review
- Display a chat** #111 opened by schmidtjonas  
3 User Story enhancement
- Add persistence for commonly requested types.** #26 opened by Mrnikbobjeff  
1 User Story enhancement  
📌 Sprint 1 Review
- Support sending messages** #20 opened by Mrnikbobjeff

**4 In progress** + ...

- Create code coverage reports** #77 opened by Mrnikbobjeff  
help wanted
- Log out of Client** #79 opened by rohansaw  
2 User Story sold
- UI for chat list** #109 opened by schmidtjonas  
enhancement sold
- Track Squeak FFI integration** #54 opened by Mrnikbobjeff

Automated as In progress Manage

**0 Review in progress**


Automated as In progress

# User Stories



Authenticate with Telegram #14 Edit New issue

Closed Mrnikbobjeff opened this issue on May 4 · 0 comments

 Mrnikbobjeff commented on May 4 • edited by rohansaw · Member 🗨 ⋮


**Is your feature request related to a problem? Please describe.**  
Authentication is the basis for all interaction with the telegram client. Understand the authentication mechanism present (are there refresh tokens, who issues API Keys, etc.)


**Describe the solution you'd like**  
As we should have presentable things at the end of every sprint, and this probably will be the first thing to be requested, we should have a workable UI to simply login and logout, or directly logout if we have a cached authentication tokens.


**Describe alternatives you've considered**  
No UI and simply authenticate via CLI. Worse, as they will expect a GUI at some point.

**Additional context**  
See TDLIB authentication here [in the docs](#).

**Estimate:**  
3 Pizzen

 Mrnikbobjeff added enhancement User Story labels on May 4

 Mrnikbobjeff added this to the **Sprint 1 Review** milestone on May 4

 Mrnikbobjeff added this to **Sprint 1 Review** milestone on May 4

**Sprint 1 Review**

Linked pull requests ⚙  
Successfully merging a pull request may

UserStory Beschreibung  
(Template)

Featurebeschreibung

Lösungsvorschlag

Alternativen bewerten

Aufwandsschätzung (Pizzen)

# User Stories



Authenticate with Telegram #14

**Closed** Mrnikbobjeff opened this issue on May 4 · 0 comments

Mrnikbobjeff commented on May 4 • edited by rohansaw - Member

Is your feature request related to a problem? Please describe.  
Authentication is the basis for all interaction with the telegram client. Understand the authentication mechanism present (are there refresh tokens, who issues API Keys, etc.)  
Describe the solution you'd like  
As we should have presentable things at the end of every sprint, and this probably will be the first thing to be requested, we should have a workable UI to simply login and logout, or directly logout if we have a cached authentication tokens.

you've considered  
authenticate via CLI. Worse, as they will expect a GUI at some point.  
on here in the docs.

Assignees  
rohansaw

Labels  
3  
Blocker  
User Story  
enhancement  
sold

Projects  
Sprint  
Done

Milestone  
Sprint 1 Review

Linked pull requests  
Successfully merging a pull request may

Mrnikbobjeff added enhancement User Story labels on May 4

Mrnikbobjeff added this to the Sprint 1 Review milestone on May 4

Aufgaben zuweisen

Statusupdates & Fragen an den Kunden notieren

Zuweisen von Labels

# Sprint: Durchführung



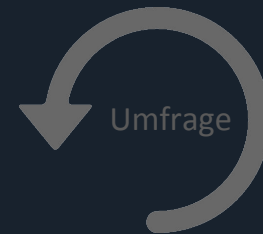
4 Pizzen  
Zeiteinheit

User-Story  
schätzen

# Agile Planning

Retro

Client-Review  
Do., 30min



alle 4 Wochen

## Coding

Meetings: Mo. & Do.







# Sprint Ablauf

- Verteilen der ausgewählten User Stories
- Bilden von Pair Programming Teams
  - “Weiches” Pair Programming - oft schwierig einen passenden Termin zu finden
- Features werden auf eigenen Branches unabhängig voneinander entwickelt

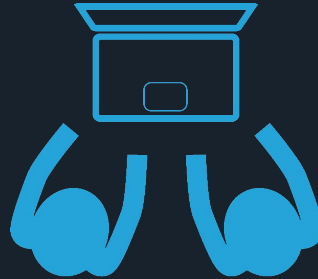
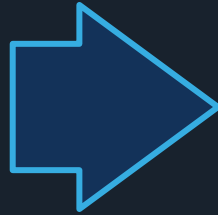
## Kommunikation

- Coding Sessions via Discord
- Discord mit Github-integration für Updates aus anderen Teams
- 2x pro Woche Stand-Up Meetings

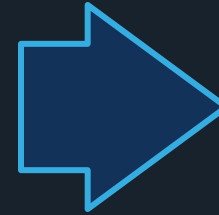
# Ablauf des Sprints



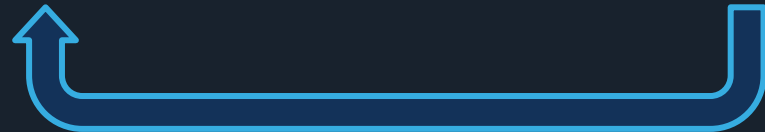
Zuweisen der  
Aufgaben



Pair-Programming



Standup &  
Review



---

Sprint: Implementierung  
**Authentifizierung**



hpi-swa-teaching / TelegramClient

Watch 4 Unstar 5 Fork 2

Code Issues 30 Pull requests 2 Actions Projects 1 Wiki Security 0 Insights

## Authenticate with Telegram #14

**Closed** Mrnikbobjeff opened this issue on 4 May · 0 comments

Mrnikbobjeff commented on 4 May · edited -

**Is your feature request related to a problem? Please describe.**  
Authentication is the basis for all interaction with the telegram client. Understand the authentication mechanism present (are there refresh tokens, who issues API Keys, etc.)

**Describe the solution you'd like**  
As we should have presentable things at the end of every sprint, and this probably will be the first thing to be requested, we should have a workable UI to simply login and logout, or directly logout if we have a cached authentication tokens.  
For first time login we should add registration functionality

**Describe alternatives you've considered**  
No UI and simply authenticate via CLI. Worse, as they will expect a GUI at some point.

**Additional context**  
See TDLIB authentication here [in the docs](#).

**Estimate:**  
3 Pizzen

Mrnikbobjeff added **enhancement** **User Story** labels on 4 May

Mrnikbobjeff added this to the **Sprint 1 Review** milestone on 4 May

Mrnikbobjeff added this to **To do in Sprint** on 4 May

Assignees: rohansaw

Labels: Blocker, User Story, enhancement

Projects: Sprint (Done)

Milestone: Sprint 1 Review

Linked pull requests: Auth-Core built

Verbinden mit der TDLib

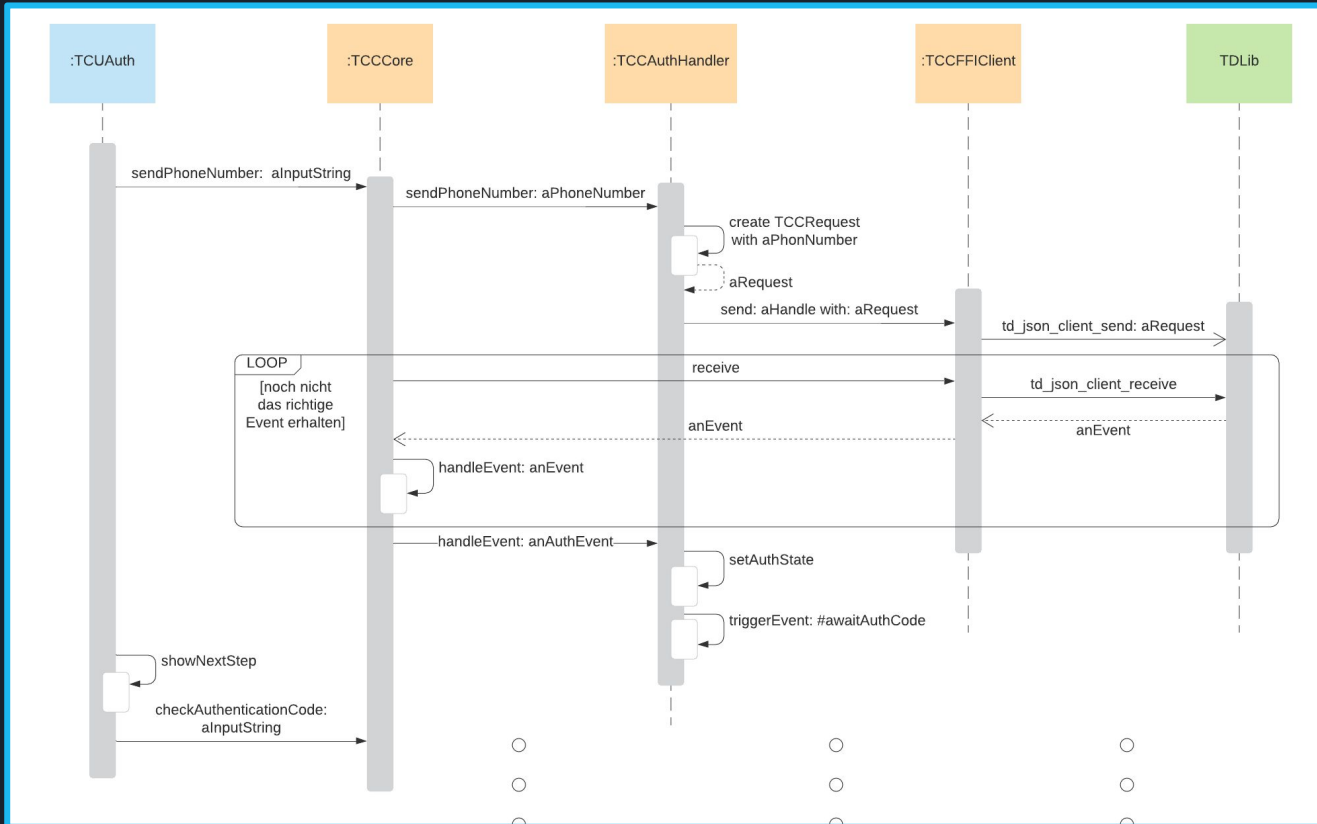
Authentifizieren des Anwenders

Entwicklung einer UI

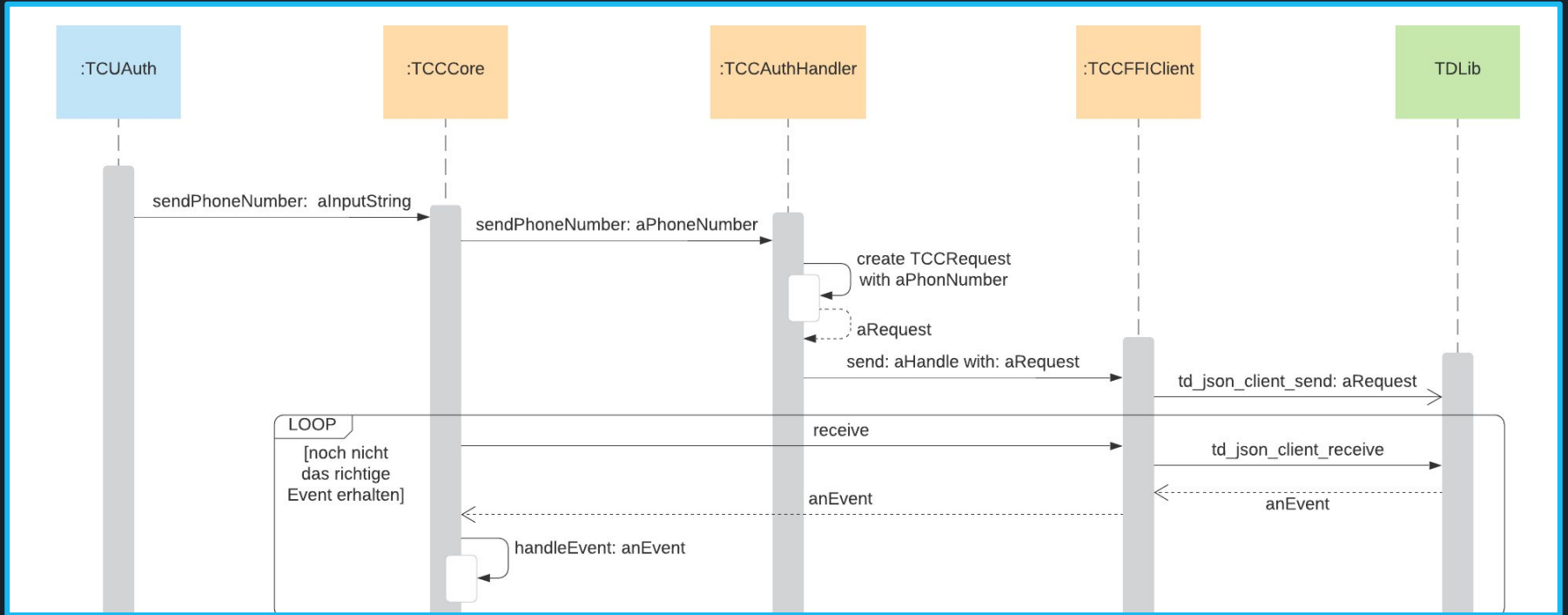
3/4



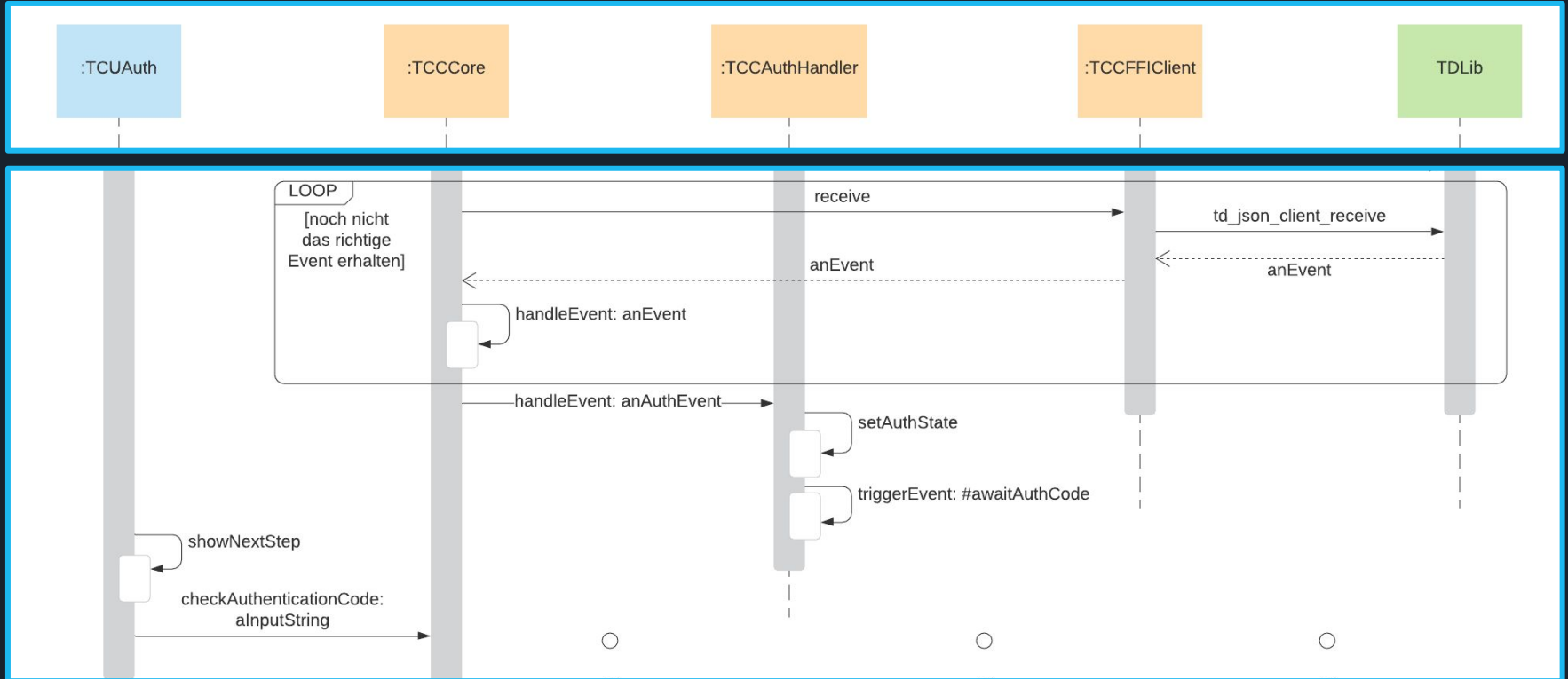
# Technische Umsetzung - Authentifizierung



# Technische Umsetzung - Authentifizierung



# Technische Umsetzung - Authentifizierung



# Technische Umsetzung - Authentifizierung

## TCCAuthHandler

```
sendPhoneNumber: aNumber
```

```
self client send: (TCCRequest  
    newWithType: 'setAuthenticationPhoneNumber'  
    from: {'phone_number' → aNumber}).
```



---

Methodik #1:

# Coding Standards



# Motivation

```
chatItemClicked: item event: event from: sender

self items do: #deselect.
(item chatID ~= selected id) ifTrue: [
    item select.
    selected := item chat.
    self
        triggerEvent: #newChatSelected
        with: item chat.
]
```

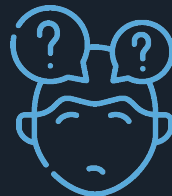
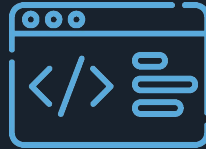
```
chatItemClicked: aChatListItem event: anEvent from: aSender
self items do: [:anItem | anItem deselect].

(aChatListItem chatID = self selectedChat id)
    ifTrue: [^nil].

aChatListItem select.
self
    selectedChat: aChatListItem chat;
    triggerEvent: #newChatSelected with: aChatListItem chat].
```

**Gleiche Semantik, aber  
Nutzung verschiedener  
Code-Styles**

# Warum einheitlicher Code?



## Collective Code Ownership

- Jeder kann jede Methode verstehen und ändern
- Nicht erkennbar, wer was geschrieben hat

## Spart wertvolle Zeit

- Beim Lesen und Verstehen
- Beim Refactoring und Bugfixing

## Verhindert Fehlinterpretation



# Coding Standards

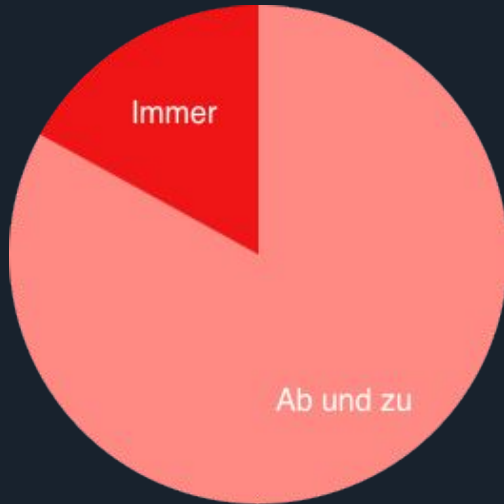
Insbesondere zu Beginn unseres Projekts haben wir am eigenen Projekt gemerkt, welchen hohen Einfluss Coding Standards auf die Teammoral und den Projekterfolg haben. Ursprünglich hatten wir zu wenig auf Coding Standards geachtet und so offenbarten sich in unseren regelmäßigen Umfragen einige Herausforderungen: Code musste häufig von anderen refactored werden und dies wirkte sich auch auf die Team/Arbeitsmoral aus. Also beschlossen wir, noch stärker auf Coding Standards zu achten. Dazu beschlossen wir Maßnahmen, wie, dass wir ein [Codequality-Dokument](#) aufsetzten, Code Reviews einführten und verstärkt im Pairprogramming Features umsetzen wollten.

Die Resultate unserer Maßnahmen waren sehr positiv: Code musste kaum noch refactored werden, um den Code in einem Stil zu halten, sondern war schnell verständlich und einheitlich. Dabei war eine Maßnahme besonders hilfreich: Wir entwickelten eigene, sehr strenge Lintertests, damit kaum noch die Möglichkeit besteht, "unsauberen" Code in den Develop oder Master-Branch zu pushen. Außerdem setzten wir verstärkt auf Pairprogramming, um schon beim schreiben des Codes zu verhindern, dass anderer Codestyle geschrieben wird.

Projekterfolg und der Teamspirit sind stark voneinander abhängig. Aus diesem Grund haben wir uns entschieden alle 4 Wochen Umfragen unter den Teammitgliedern durchzuführen, um schnell und proaktiv handeln zu können.

Hierbei haben wir unterschiedliche Themen abgefragt, wie unter anderem: Teamgeist, Selbstreflektion/Motivation, CodeQuality, etc.

# Sprint #1 - Coding Standards



50%

Solo Work

Wie oft muss ich Code  
von anderen refactoren?

# Verbesserung der Coding Standards



Eigene **Linting-Tests** &  
**Branch Protection**



Mehr Zeit für  
**Code Reviews**



**Clean-Code-**  
**Guidelines**



Mehr **Pair-**  
**Programming**

# Linting-Tests

```
testMethodParamsHaveMeaningfulNames
```

```
self methodsLinesDo: [:lines | self assert: (lines first includesSubstring: 'anObject') not].
```

```
testMethodHasEmptyLine
```

```
self methodsLinesDo: [:lines | self assert: (lines size < 2 or: [lines second isEmpty])].
```



# Linting-Tests

```
methodsLinesDo: aBlock
```

```
self methodTestObjects do: [:aSLMethodTestObject | aBlock value: aSLMethodTestObject sourceCode string lines].
```



# Clean-Code-Guidelines

```
chatItemClicked: aChatListItem event: anEvent from: aSender  
self items do: [:anItem | anItem deselect].
```

```
(aChatListItem chatID = self selectedChat id)  
  ifTrue: [^nil].
```

```
aChatListItem select.  
self  
  selectedChat: aChatListItem chat;  
  triggerEvent: #newChatSelected  
  with: aChatListItem chat].
```

```
chatItemClicked: item event: event from: sender
```

```
self items do: #deselect.  
(item chatID ~= selected id) ifTrue: [  
  item select.  
  selected := item chat.  
  self  
    triggerEvent: #newChatSelected  
    with: item chat.  
]
```

# Clean-Code-Guidelines

## verständliche Namen für Parameter

```
chatItemClicked: aChatListItem event: anEvent from: aSender  
self items do: [:anItem | anItem deselect].
```

```
(aChatListItem chatID = self selectedChat id)  
  ifTrue: [^nil].
```

```
aChatListItem select.
```

```
self  
  selectedChat: aChatListItem chat;  
  triggerEvent: #newChatSelected  
  with: aChatListItem chat].
```

## Accessormethoden für Instanzvariablen

## Leerzeile nach Methodenname

```
chatItemClicked: item event: event from: sender
```

```
self items do: #deselect.
```

```
(item chatID ~= selected id) ifTrue: [  
  item select.
```

```
  selected := item chat.
```

```
  self
```

```
    triggerEvent: #newChatSelected  
    with: item chat.
```

```
]
```

# Ausschnitt der Clean-Code Guidelines aus unserem Github-Wiki

- Choose descriptive names for objects, variables and methods
- Capitalize class names, global variables, pool dictionaries, and class variables
- Do not capitalize instance and temporary variables, parameters, and methods
- all Names should be in Camel-Case, Parameter-Names should start with `a` or `an` (e.g. `aChatListItem`)
- Do not use hard-coded (magic) numbers in an expression

### Comments

- Code should be self-documenting!
- Avoid comments that restate the code
- Write comments for unusual implementation details

### Formatting

- Insert one blank line after the method name
- Insert a dot at the end of a method if it is not a return statement
- Employ at least one blank before and after a binary selector
- Leave spaces around `@`
- Leave one blank before a left parenthesis and after a right parenthesis
- Leave one blank after but not before a comma, a semicolon, and a colon in a selector
- Put zero or one argument messages on the same lines as their receiver
- For messages with two or more keywords put each keyword / argument pair on its own line, indented one tab
- Use a Cascade to send several messages to the same receiver
- Only use parenthesis if necessary

### Instance Variables

- only set and read instance variables by accessor methods

### Misc.

- Use blocks instead of Symbols for methods like `aCollection do:` if possible

# Sprint #3 - Coding Standards



Wie oft muss ich Code von anderen refactorieren?

75%

Pair Programming

**Positive Nebeneffekte:**

- Wissen wird besser verteilt
- Soziale Komponente -> Teambuilding

---

Methodik #2:

# Continuous Integration & Repository-Management

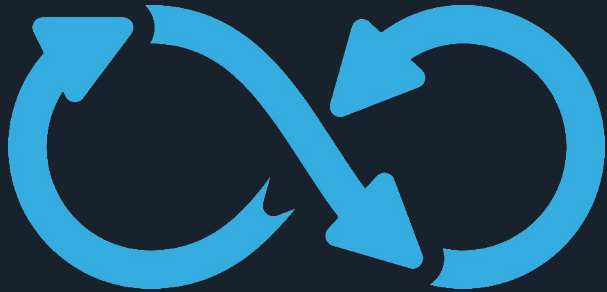




# Warum Continuous Integration?

- Regelmäßige kleine Releases
- Automatisierte Tests & kurze Testzyklen
- Kein defekter Code in Releases
  - Sicherung der Coding Standards
  - Lieber weniger Features, als schlecht funktionierende Features
- Häufige Integration verhindert, dass wir auf fehlerhaftem Code aufbauen
  - Je später wir einen Fehler bemerken, desto schwieriger wird es, den Fehler zu finden und beheben

# Continuous Integration - Umsetzung



- Nutzung von Github-Actions zusammen mit Smalltalk-Ci
  - Automatisiertes Testen
  - Tests laufen erst beim Pull-request auf Dev-Branch

**Aktuell:** Arbeiten an Code-Coverage

- Metrik zur Absicherung der Qualität

## Branch Protection

- Merge nur bei **erfolgreicher CI**
- kein Merge ohne **Review**

- Beim Merge: **Zusammenfassen** mehrerer Commits zu einem Commit
- **saubere** History

## Squash Merging



# Repository- Management

## Branches

- **Master/Dev**-Branches
- Feature/Fix-Branches
- **Releases** alle zwei Wochen

- Feature- und Fix **Templates**
- einheitliches **Reporting**

## Templates



Wird beim  
Merge-Request  
automatisch  
angezeigt

## Pull Request template

Please, go through these steps before you submit a PR.

1. Make sure that:

- a. You have done your changes in a separate branch. Branches MUST have descriptive names that start with either the `fix/` or `feature/` prefixes. Good examples are: `fix/signin-issue` or `feature/issue-templates`.
- b. You have a descriptive commit message with a short title (first line).
- c. You have only one commit (if not, squash them into one commit).
- d. Running tests doesn't throw any error. If it does, fix them first and amend your commit ( `git commit --amend` ).
- e. You have tracked ALL applicable time working in clockify!

2. **After** these steps, you're ready to open a pull request.

- a. Your pull request MUST NOT target the `master` branch on this repository. You probably want to target `develop` instead.
- b. Give a descriptive title to your PR.
- c. Provide a description of your changes.
- d. Put `closes #XXXX` in your comment to auto-close the issue that your PR fixes (if such).

IMPORTANT: Please review the [CONTRIBUTING.md](#) file for detailed contributing guidelines.

**PLEASE REMOVE THIS TEMPLATE BEFORE SUBMITTING**

# Herausforderung: Testen des Clients



## Anforderungen:

CI benötigt **TDLib Binaries**

Verifizieren der **Telefonnummer** notwendig

**Asynchrone Events** müssen getestet werden

# Lösungsansätze

## #1 Mock Tests

- keine echte Verbindung zur TDLib
  - TDLib Binaries nicht notwendig
  - Keine asynchronen Events
- Überprüfung des Verifikations-Code wird gemocked
  - Keine echte Telefonnummer mehr notwendig

## #2 Telegram Test-Datacenter

- Telegram stellt Test-Datacenter bereit
  - unbegrenzte Authentifizierungsversuche
- Der Test-Datacenter stellt (Handynummer, Verifikations-Code)-Paare bereit
- CI benötigt Binaries
  - automatisches Herunterladen der Binaries durch Squeak

## Das lief gut:



- Immer **aktuelle, getestete Version** auf dem Dev-Branch
  - Funktionierende Demo stets verfügbar
- Integrationsprobleme **frühzeitig** erkannt
  - Verhindert komplexe Merge-Konflikte

## Verbesserungswürdig:



- Unterschätzung der **Einarbeitungs- und Setup-Zeit**
- **Zähe Entwicklung** durch Tests & Evergreen
  - Anspruchsvoll, zum Release, immer alle Tests zu passen
- **Expertenwissen** über CI-Workflow
  - Anspruchsvoll, CI-Probleme, ohne Experten zu lösen

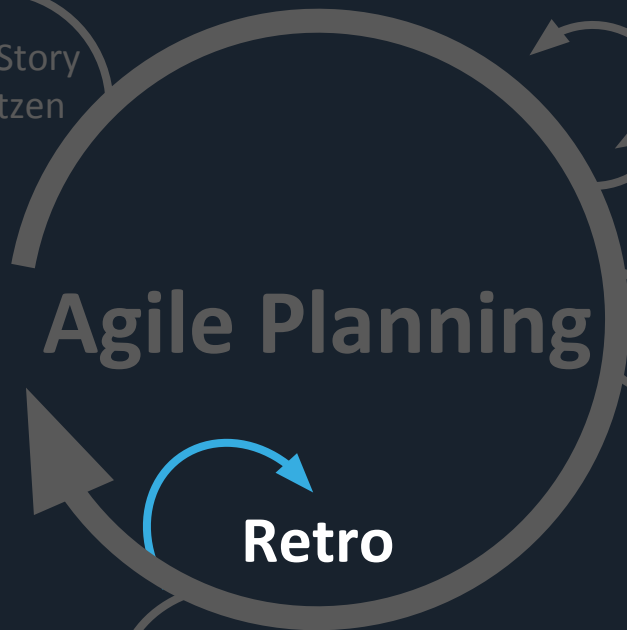
# Sprint: Retro



4 Pizzen  
Zeiteinheit



User-Story  
schätzen



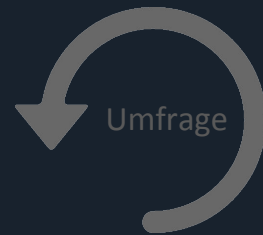
Agile Planning



Retro



ClientReview  
Do. 30min



Umfrage

alle 4 Wochen

Coding  
Meetings: Mo. & Do.





# Durchführung der Retrospektive

Zu jedem Sprint-Zyklus gehörte immer auch eine Retrospektive. Dort haben wir uns angeschaut, was gut im Sprint lief, mit welchen Problemen wir zu kämpfen hatten und wie wir diese Probleme in Zukunft verhindern werden. Dabei haben wir festgestellt, dass die Probleme mit denen wir Schwierigkeiten hatten oft nur Symptome von tiefer liegenden Problemen bzw. Kommunikationsschwierigkeiten waren.

Deutlich wurde dies zum Beispiel bei der Tatsache, dass zu Beginn öfters die gleiche Aufgabe von mehreren Personen erledigt wurde, wie sich herausstellte lag dies daran, dass wir zu wenig kommunizierten und kaum jemand wusste, woran die Andere gerade arbeiten.

# Durchführung der Retrospektive

Was hat gut funktioniert?

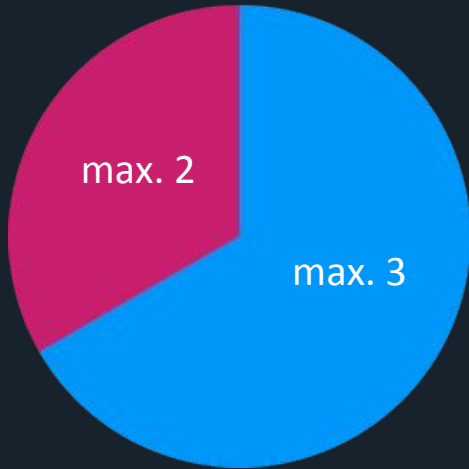
Welche Probleme sind aufgetreten?

Wie werden wir diese Probleme verhindern?

**Warum?**



# Sprint #1 - Zusammenarbeit



Von wie vielen  
Teampartnern sind die  
Aufgaben bekannt?

5/10

Motivation, am Projekt  
mitzuarbeiten

## Situation:

lange Coding-Sessions von  
mehr als 3 Stunden



## Folgen:



Vermeidbare Fehler und  
unsauberer Code



Schlechte Stimmung



Produktivität nimmt ab



Schlafrythmus leidet



# Lösung: Sustainable Development

**Eigentliches Problem:** Schlechtes Zeitmanagement

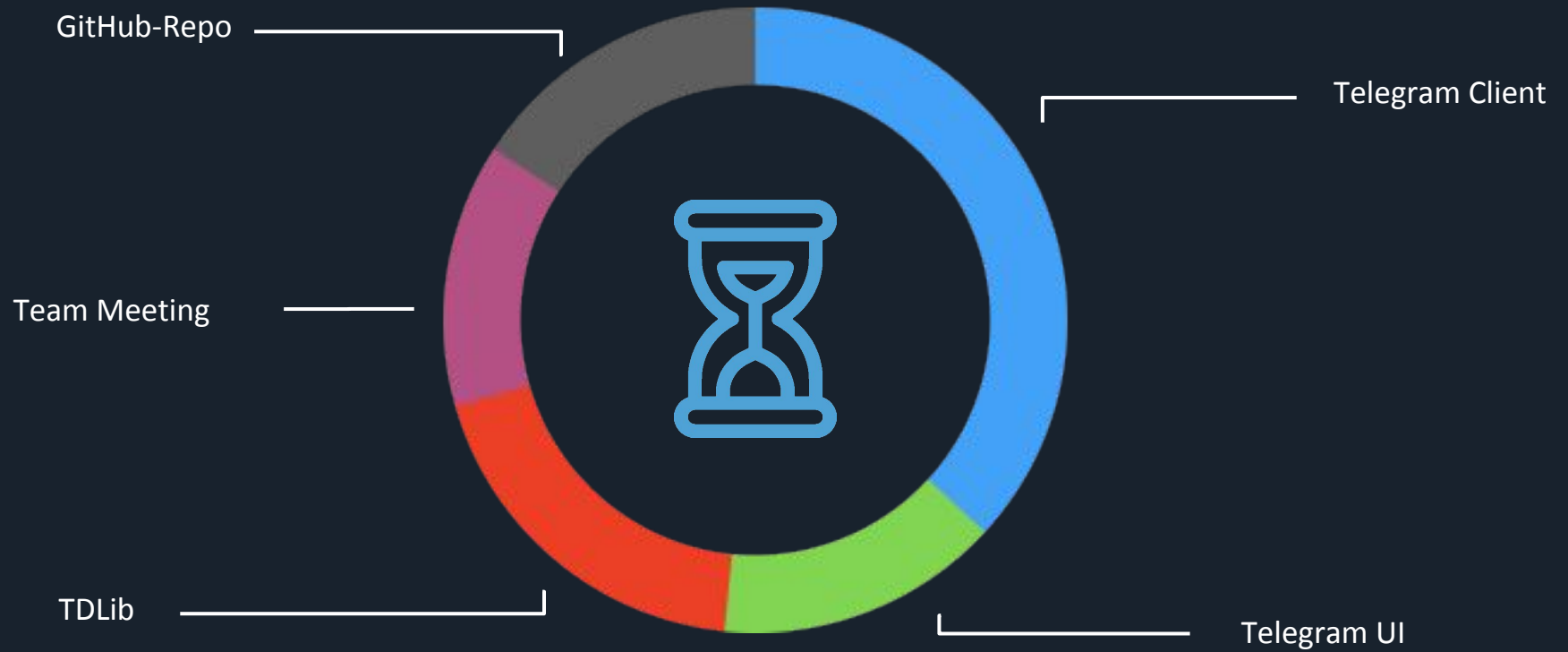
## Lösung

- Features in **kleinere Aufgaben** splitten und diese in verschiedenen **Code-Sessions** abarbeiten
  - Zu Beginn Arbeitszeiteinteilung festlegen
- Regelmäßige **Pausen**
- Rücksicht auf **Stimmung** des Code-Partners nehmen

## Folgen

Verbessertes **Teamklima**  
→ **Produktivität**  
gesteigert

Vermeiden unnötiger  
**Fehler**



## Situation:

Schwierigkeiten beim Verbinden  
mit der TDLib

```
Error: External module not found
TCCLinuxClient(Object)>>error:
TCCLinuxClient(Object)>>externalCallFailed
TCCLinuxClient(TCCFFIClient)>>create
TCCTeleClient class>>newWithClient:
Proceed Restart Into Over Through Full Stack Where Tally It
create
<cdecl: void* 'td_json_client_create' ()>
^ self externalCallFailed
self all inst vars handle name
-<- Select receiver's field
thisContext stack top all temp vars
-<- Select context's field
```

## Folgen:



**Motivation sank**, je mehr  
Lösungsansätze fehlschlugen



**Ungleiche Verteilung** der  
Arbeitslast



Schlechter Wissensaustausch



# Lösung: Organization is key

## Lösung

### Technisch:

- VMs, Entfernen von DLLs, Betriebssystemwechsel

### Methodik:

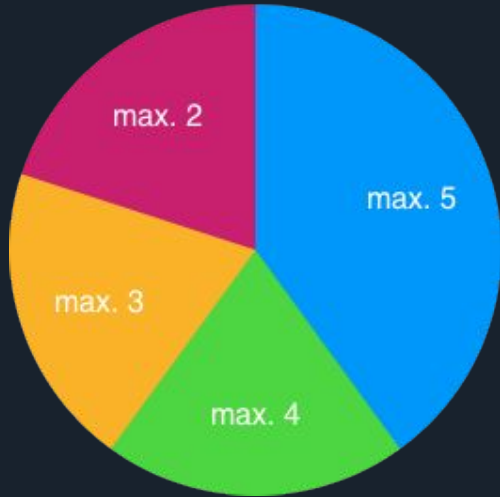
- Team dass sich darum kümmert, die Library für ein bestimmtes OS zum Laufen zu bekommen
- Pair Programming, bis es auf allen Betriebssystemen läuft
- Verstärkter Austausch und Kommunizieren von Lösungsansätzen ->Meeting-Agenda

## Folgen

**Bessere** und **zielgerichtete** Kommunikation

Probleme werden **schneller** und **effizienter** gelöst

# Sprint #3 - Zusammenarbeit



Von wie vielen  
Teampartnern sind die  
Aufgaben bekannt?

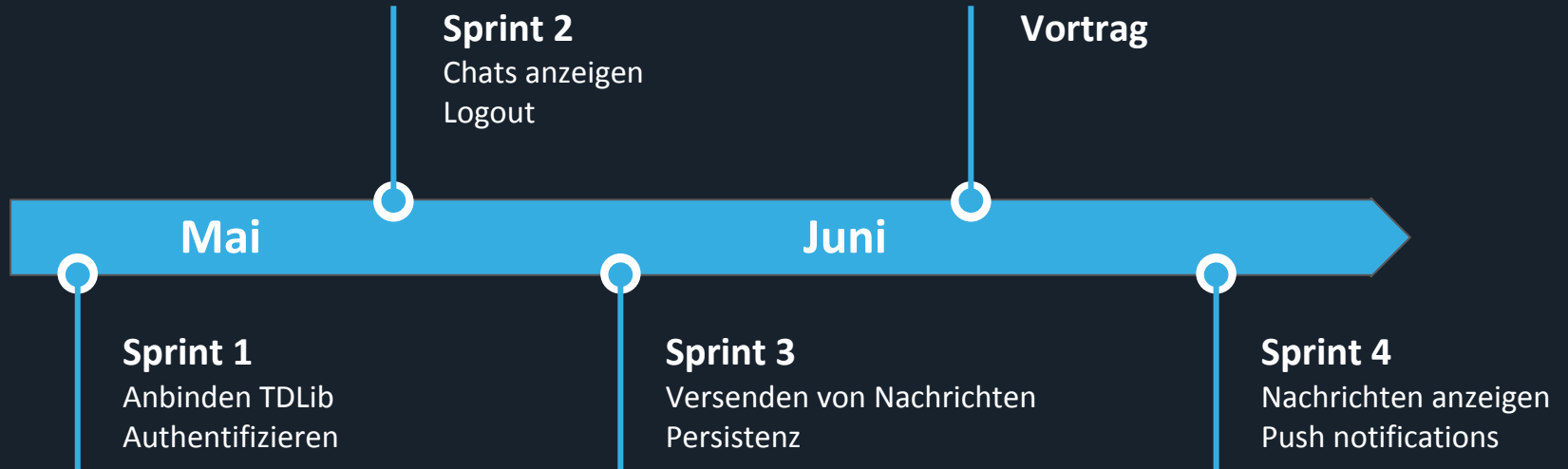
**8/10 (+3)**

Motivation, am Projekt  
mitzuarbeiten

# Projektrückblick



# Projektverlauf



# Ausblick

**Textsuche**  
in Chats



**Zen-Mode**

Blocken von  
Benachrichtigungen



**Versenden von**  
**Medientypen**  
wie beispielsweise  
Bildern oder Morphs



## Das lief gut:



- 100% **termingerechte** Umsetzung der User Stories
- **Teamklima** hat sich stark verbessert (8/10 Punkten)
- Meetings mit dem Kunden **gut vorbereitet**
- **Arbeitsprozess** deutlich verbessert

## Das lief noch nicht so gut:



- Unterschätzung der **Einarbeitungszeit** und Schwierigkeiten mit der Library
- **Aufgabenverteilung** & Austausch über aktuellen Status
  - Ungleicher Wissensstand
- Einteilung der **Arbeitszeit**



# Lessons Learned



**Kommunikation** & gute **Organisation** sind essentiell, um ein gutes Produkt entwickeln zu können



Es ist wichtig von Anfang an alle einzubeziehen und Verantwortlichkeiten zu verteilen.

**Verzweiflung** kommt vor, kann jedoch gemeinsam als Team überwunden werden.



Durch **Pair Programming** werden Probleme verhindert, bevor sie entstehen können  
**Clean-Code** und **Coding-Standards** hängen eng mit dem Projekterfolg zusammen.

**72**

Pull-Requests

**125**

Commits

**67/90**

Issues completed

**1400+**

new LoC



**54**

Tests

# Quellen

## Grafiken:

Grafiken von: [SmileTemplates](#)

Icons von: [flaticon](#), [pngguru](#)

Diagramme erstellt mit: [Lucidchart](#)

Bildnachweise der Icons:

Folie 5: [Mail](#), [Schloss](#), [UI](#), [Mail2](#)

Folie 6: [Zielscheibe](#)

Folie 17: [Pizza](#)

Folie 18: [Planning Poker](#), [Meeting](#)

Folie 25: [Zuweisen](#), [Pair Programming](#), [Auge](#)

Folie 27: [Pizza](#)

Folie 34: [Code](#), [Stoppuhr](#), [Mensch](#),

Folie 37: [Test](#), [Auge](#), [Guideline](#), [Menschen](#)

Folie 46: [CI](#)

Folie 47: [Git](#),

Folie 49: [Warnzeichen](#)

Folie 59: [Sanduhr](#)

Folie 66: [Chat](#), [Mensch](#), [Clean](#)

Folie 67: [Images](#), [Search](#)

## Github:

[Repository](#)

[CleanCodeGuidelines](#)

