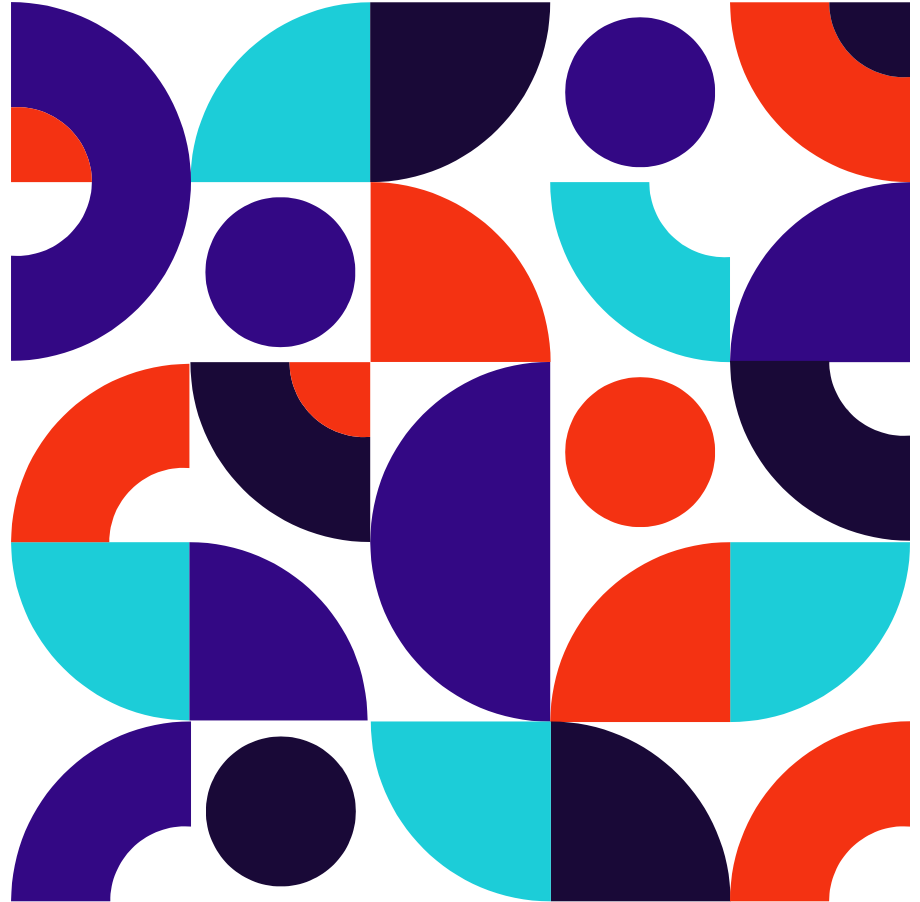


# Animations Editor

SWT SoSe 2021, 22.06.2021

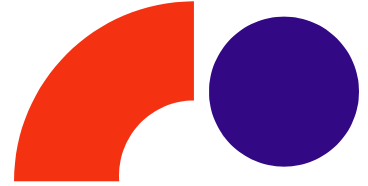
Gruppe **14**

Nick Bessin, Franziska Hradilak,  
Lukas Wenner, Cedric Lorenz,  
Jerome Stephan & Max Hoffmann



# Gliederung

1. Einleitung
2. Live Demo
3. Beschreibung der Anforderungen
4. Architektur
5. Verhaltenstechnische Zusammenhänge
6. Entwicklungsprozess
7. Testing
8. Refactoring
9. Kommunikation und Zeitmanagement
10. Metriken und Projektverlauf
11. Reflexion





# Motivation



“Für mein Squeak-Spiel muss ich jede Animation mühselig selbst coden.”



**Magische  
Nummern**

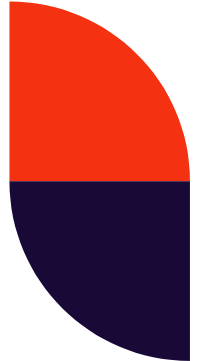
**Keine Wieder-  
verwendbarkeit**

**Erhöhter  
Aufwand**

```
AnimPropertyAnimation new
```

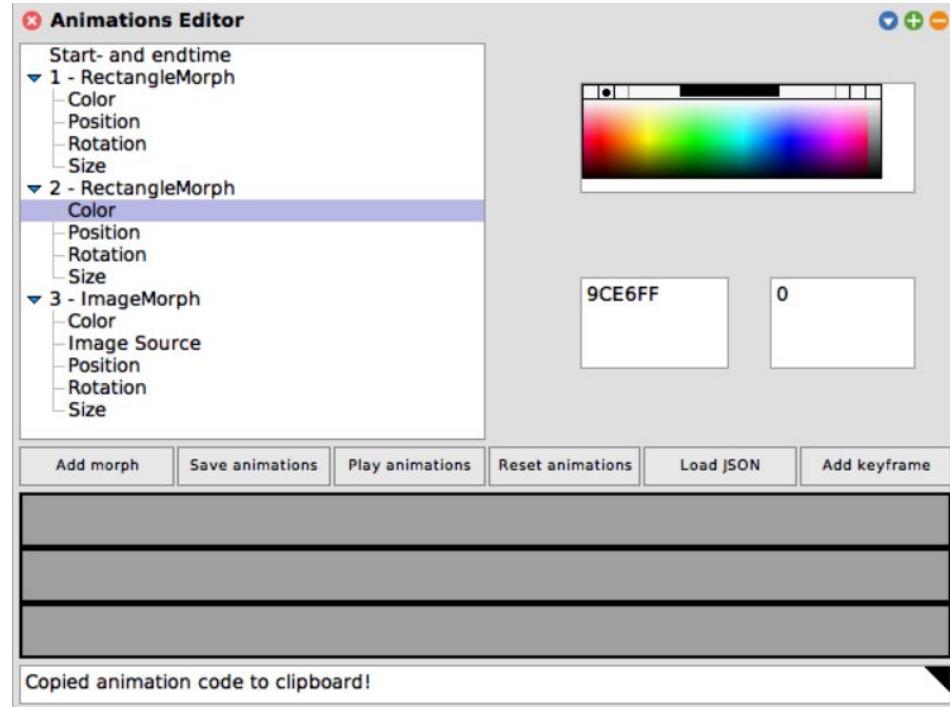
d

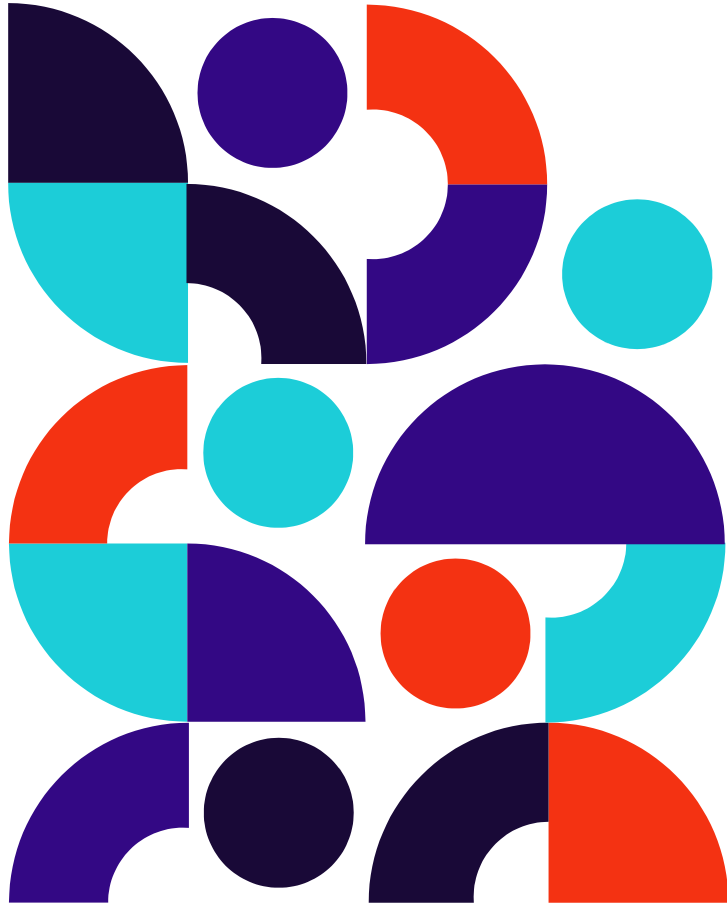
```
uration: 1000;  
property: #position;  
startValue: 25 @ 20;  
endValue: 400 @ 400;  
target: aMorph;  
state: #paused
```



# Introducing AnimationsEditor:

- Abstraktion der Animationskomplexität
- User-Interface
- Animationsvielfalt







# Live Demo des Projekts



# Anforderungen

# Funktionale Anforderungen: User Stories



Must-Haves	Should-Haves	Nice-To-Haves
Unterstützung ImageMorph	Animation speichern	Quadratische Interpolation
Mehrere Keyframes einer Eigenschaft	Animation reinladen	Nearest Interpolation
Timeline	Auswahl der Farben im Colorpicker	Unterstützung PolygonMorph
Mehrere Eigenschaften gleichzeitig animieren	Keyframes werden automatisch gesetzt	Unterstützung TextMorph
		Übergänge beim ImageMorph animieren

# Funktionale Anforderungen

## 1. Laden und Speichern von erstellten Animationen

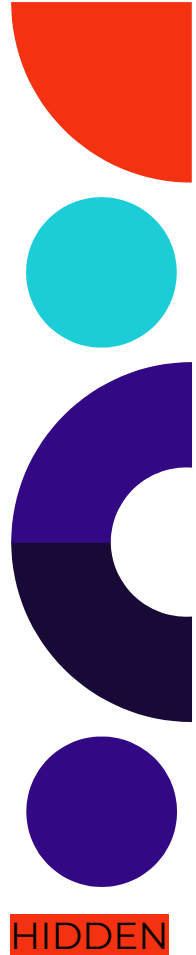
Vorteil: Animationen können über mehrere Sitzungen erstellt werden

## 2. Mehrere Keyframes animieren

Vorteil: Folgen von Animationen möglich

## 3. Unterstützung verschiedener neuer Morphs

ImageMorph: Animationen mit allen Gegenständen werden möglich. Nützlich z.B. beim Design von Spielen







# Nicht-funktionale Anforderungen

## Portability

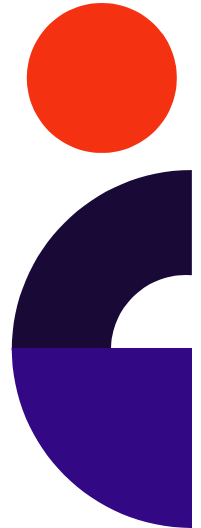
- Unterstützung von üblichen Plattformen Windows, MacOS und Ubuntu

## Robustheit

- Abfangen falscher User-Eingaben
- z.B. Verhinderung Absturz beim Abspielen nicht vorhandener Animationen

## Benutzerfreundlichkeit

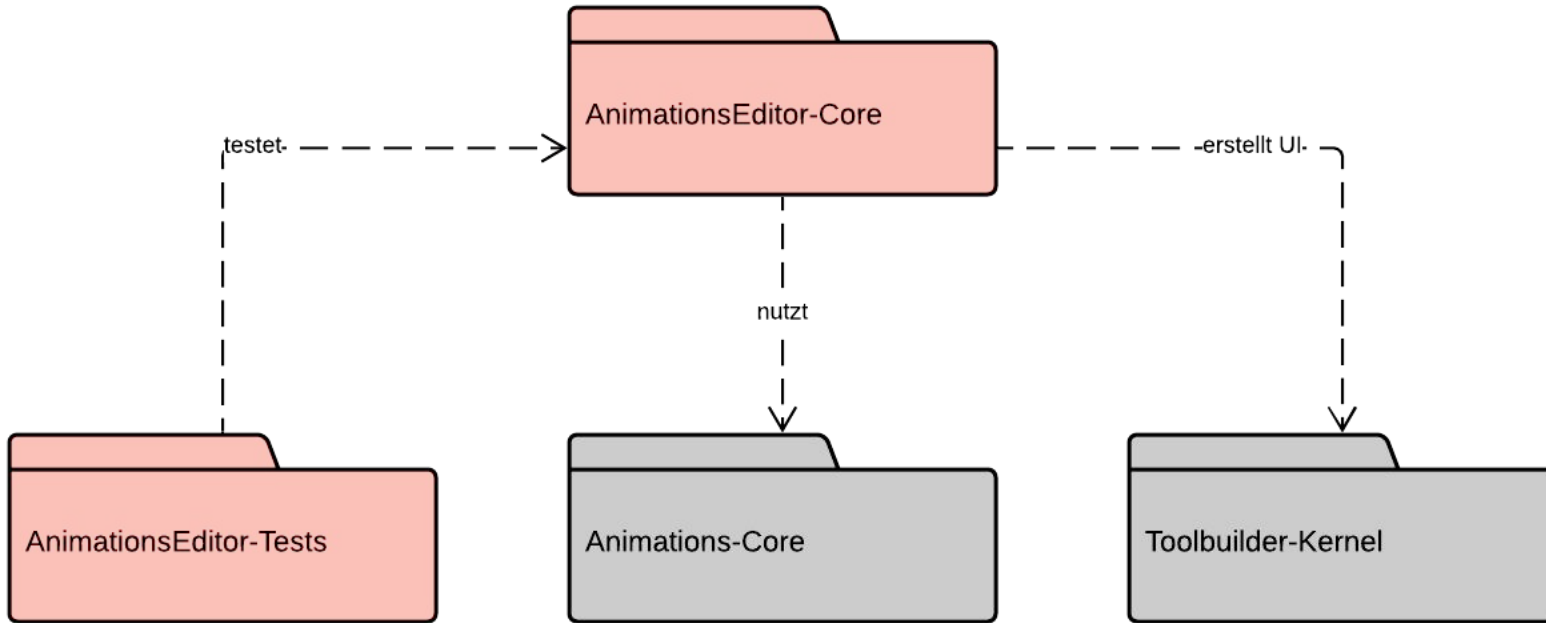
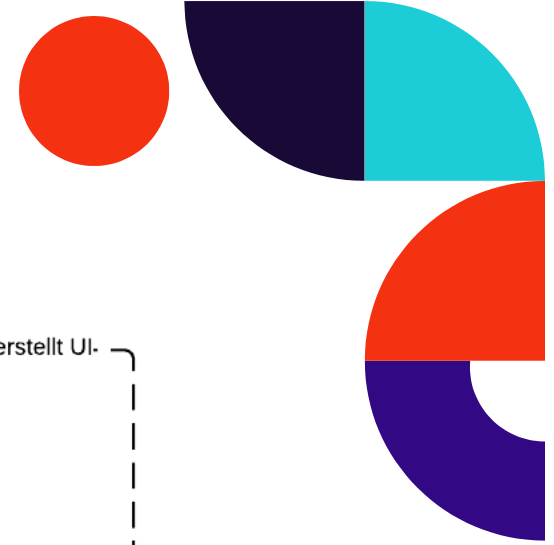
- Intuitive Farbauswahl durch Colorpicker
- Speichern und Laden von Animationen ermöglicht Arbeiten über mehrere Sessions
- Speichern & Laden in Echtzeit (< 1 sec.)



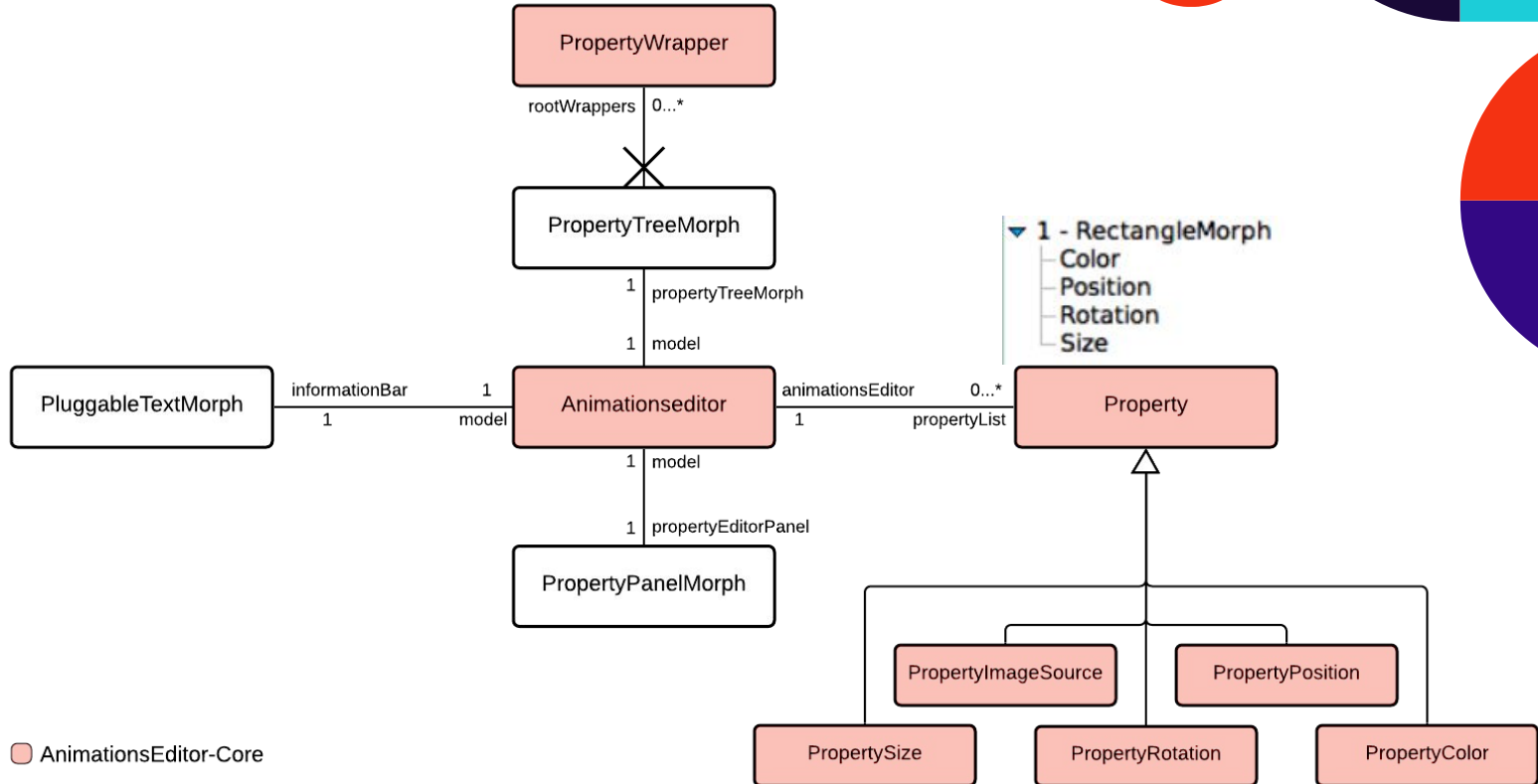


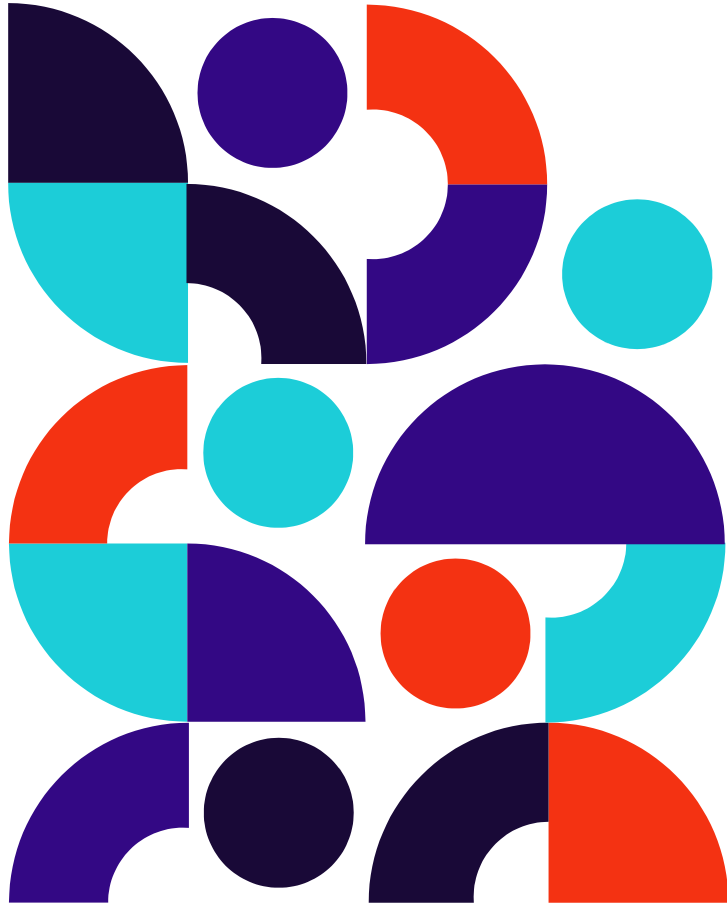
# Architektur

# Paketdiagramm



# Klassendiagramm



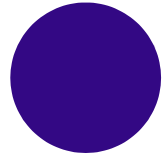
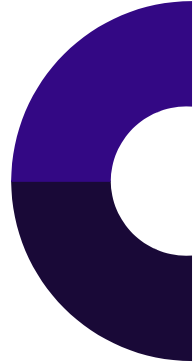
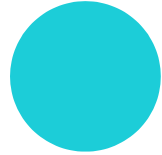
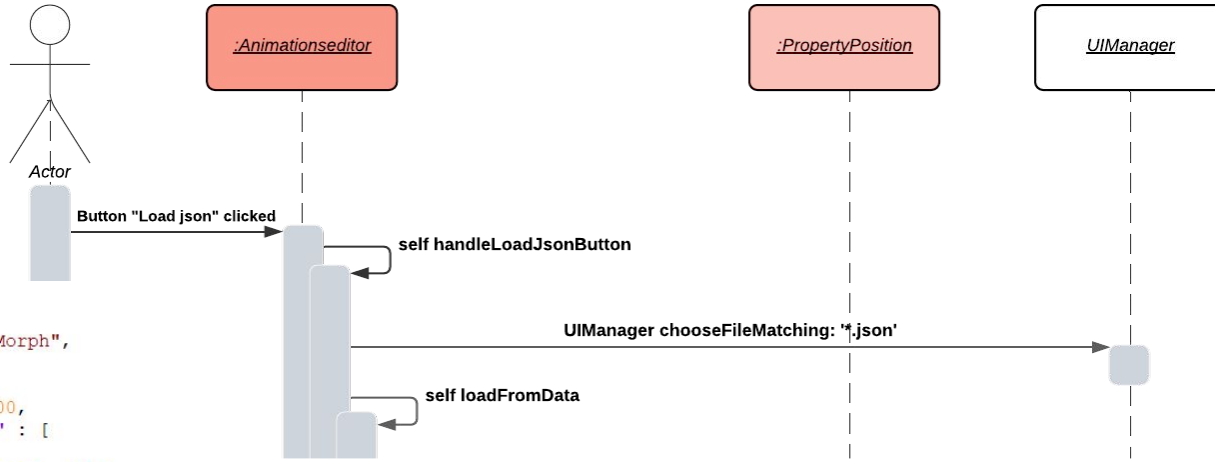


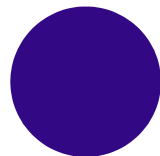
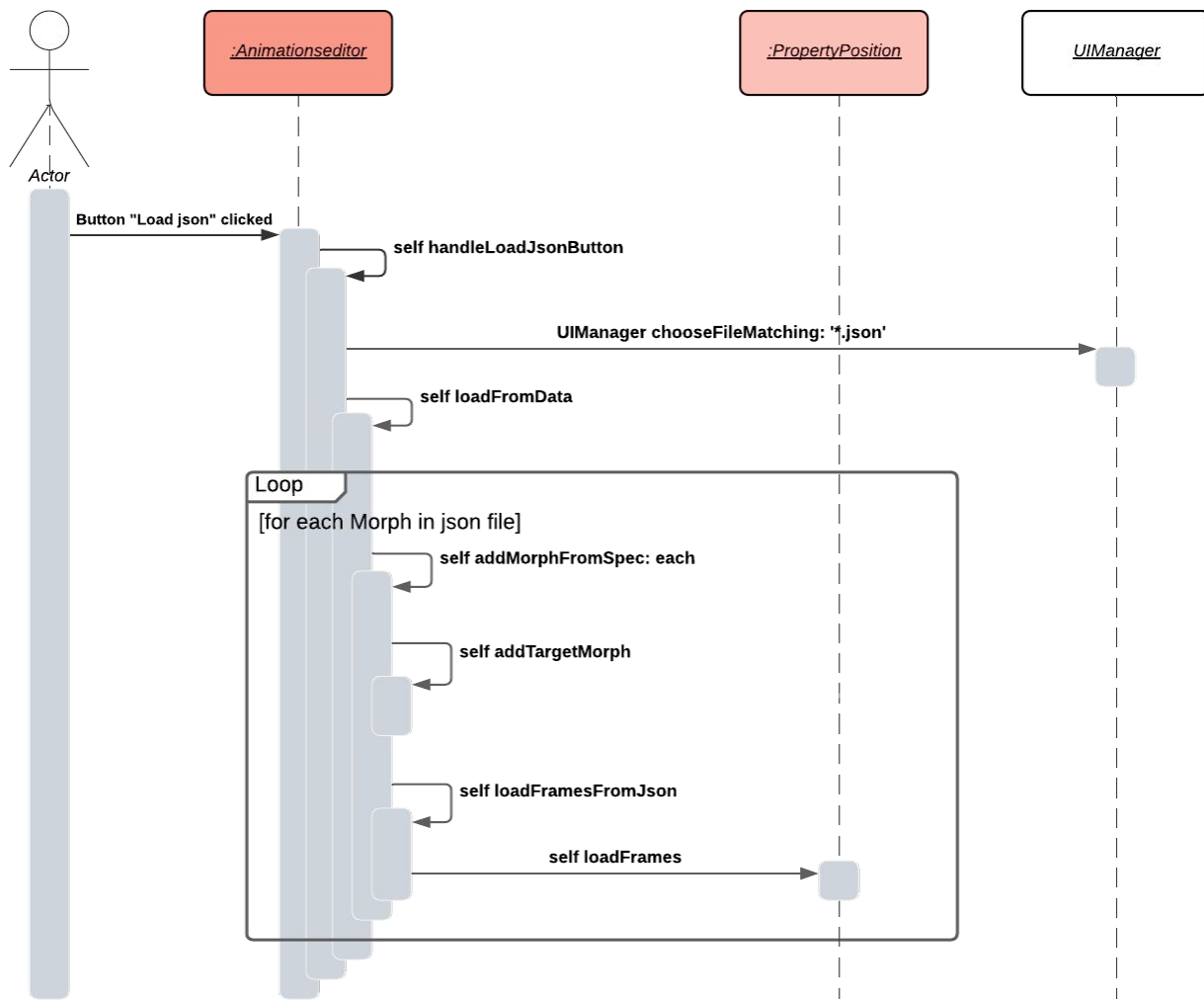
# Verhaltenstechnische Zusammenhänge - Animation laden

```

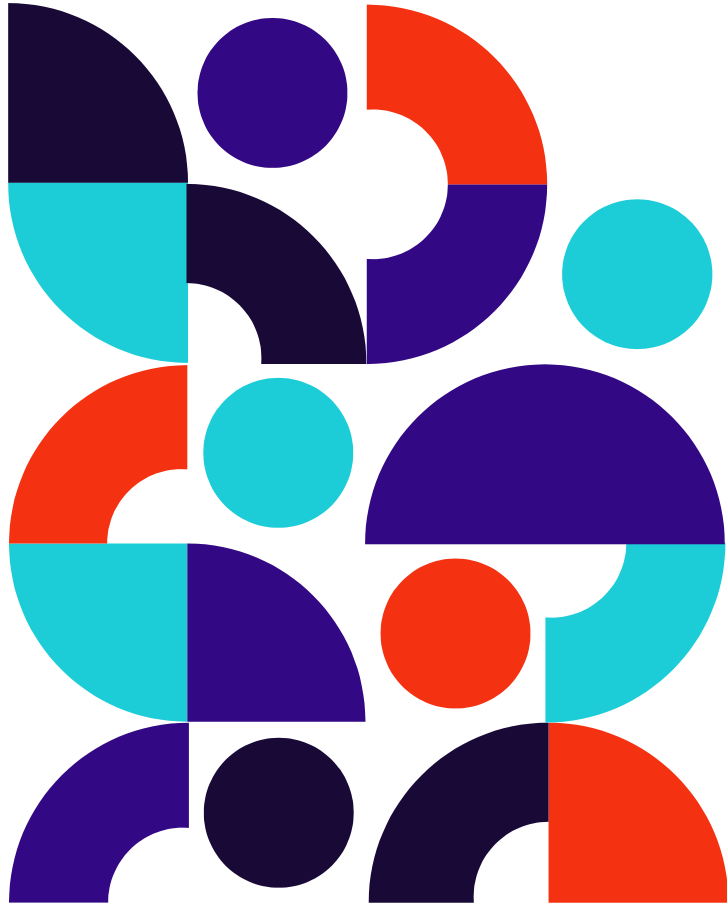
"morphs" : [
{
  "type" : "RectangleMorph",
  "frames" : [
    {
      "frame" : 500,
      "properties" : [
        {
          "value" : 120,
          "name" : "positionX"
        },
        {
          "value" : 400,
          "name" : "positionY"
        }
      ]
    },
    {
      "frame" : 1000,
      "properties" : [
        {
          "value" : 400,
          "name" : "positionX"
        },
        {
          "value" : 400,
          "name" : "positionY"
        }
      ]
    }
  ]
},
]
],
}

```





HIDDEN



# Entwicklungs- prozess



# Ausgangsbedingungen

SWA = SWT ?

- Gleiches Team, ähnliche Dynamik
- Digitales Semester
- Schreiben gemeinsam Code



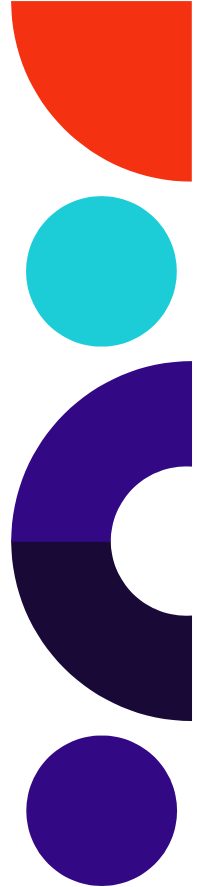
- Codebasis
- Kundin
- Vorerfahrungen



# Ausgangsbedingungen: Vorerfahrungen

## Anwendungsdomäne: Animationen

- Guter Wissensmix vorhanden:
  - Einige hatten grundlegende Vorkenntnisse
  - Eine Person betreibt hobbymäßig Motiondesign
- Wissenstransfer grundlegender Begriffe konnte schnell durch Teambesprechungen auf Alle erfolgen



# Einarbeitung in das bestehende System

## **AnimationsCore: Unsere Grundlage für jegliche Animationen**

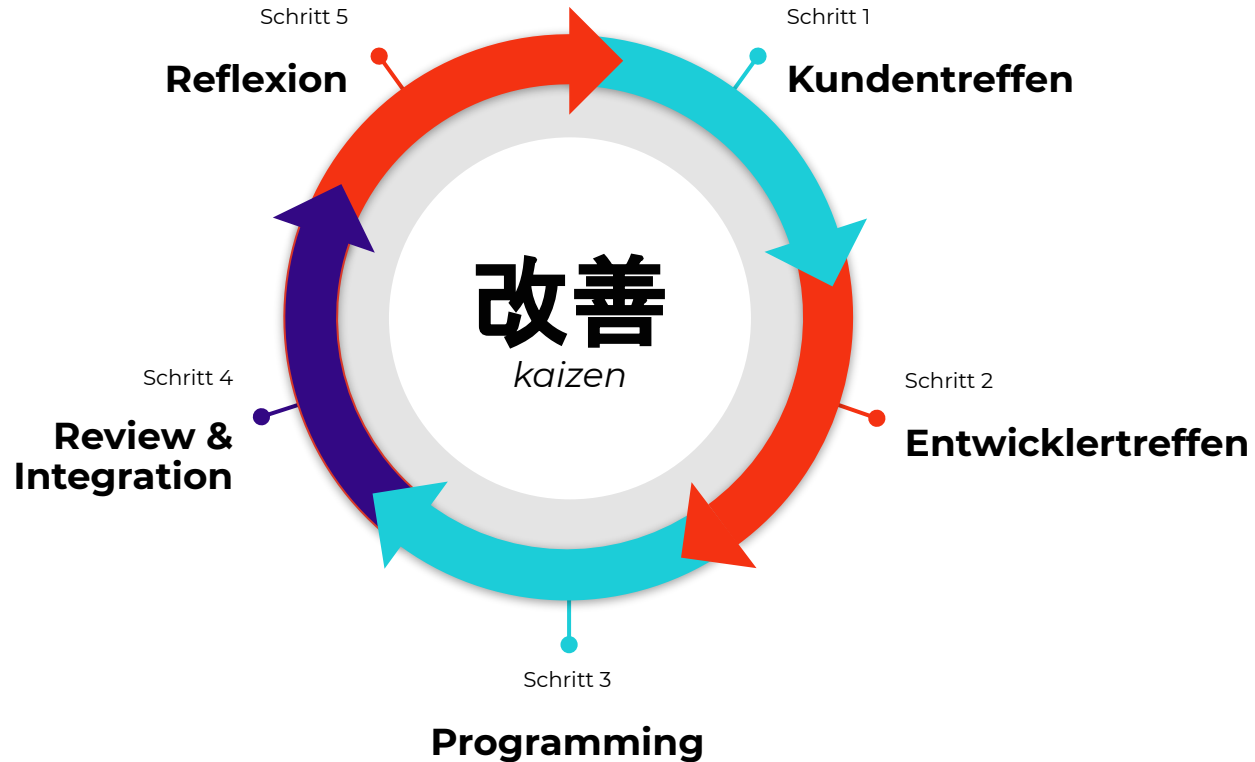
- Ausführliche Dokumentation vorhanden
  - GitHub-Repo enthält viele Beispiele & Erklärungen
  - Codebasis enthält Tests für Funktionen
- insgesamt starke Erleichterung des Verständnisses

## **ToolBuilder: Unsere Grundlage für die UI**

- Keine Vorerfahrung
  - Zeitaufwändige Einarbeitung nötig
  - Wenig Dokumentation, wenig Codebeispiele
- Erkundung der Funktionalitäten & Definitionen hat viel Zeit eingenommen



# Ablauf eines Sprints



# Wie läuft ein Sprint ab? (1)

**Kundentreffen:** Wir präsentieren der Kundin, was die letzten zwei Wochen geschafft wurde. Wir sammeln Feedback und neue Vorschläge und lassen dann neue User Stories vergeben. Dabei handelte es sich im Durchschnitt um 24 Punkte pro Sprint.

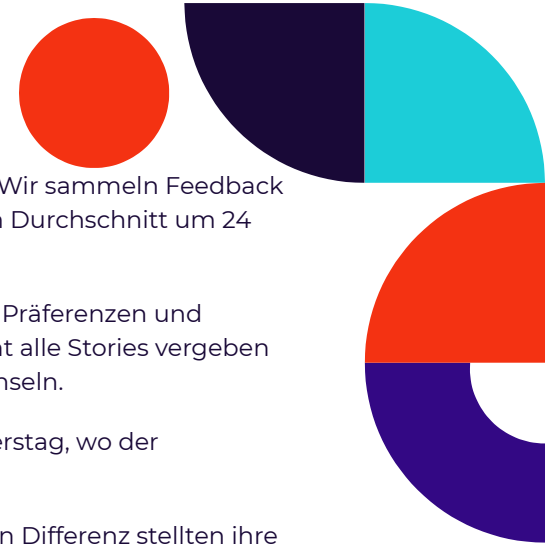
**Entwicklermeeting:** Wir legen fest, wer welche User Stories übernimmt. Dabei sammeln wir Präferenzen und versuchen, anhand dieser ideale Pair-Programming-Teams pro Story zu erstellen. Sollten nicht alle Stories vergeben werden können, wird gewürfelt. Weiterhin achten wir darauf, pro Sprint Teampartner zu wechseln.

Ein weiterer Wissensaustausch entstand bei den regulären Entwicklermeetings jeden Donnerstag, wo der Zwischenstand erfragt und Fragen beantwortet werden konnten.

Die Einschätzung von User Stories geschah durch Planning Poker. Diejenigen mit der größten Differenz stellten ihre Begründungen vor. Nach der Anhörung wird erneut abgestimmt, bis sich ein Mehrheitsergebnis finden lässt

**Programming:** Insgesamt hat sich Pair-Programming erneut als positiv herausgestellt. Vor allem, wenn zwei Leute mit unterschiedlichem Know-How aus vorherigen Sprints zusammenkommen, um einen neuen Sprint zu bewältigen und eine Synergie entsteht. (Wissenstransfer)

Weiterhin achteten wir auf Sustainable Development, was sich an manchen Stellen durch Unterschätzen der User-Stories als schwierig herausstellte. TDD stand ab dem zweiten Sprint im Zentrum unserer Entwicklung.



# Wie läuft ein Sprint ab? (2)

**Review & Integration:** Pro User Story gibt es einen eigenen Branch, für welchen wir nach Bearbeitungen eine Pull-Request auf Master stellen. Diese wird dann mit hoher Priorität reviewt, um Mergestaus zu vermeiden. Diese Erfahrung haben wir schon einmal fast gemacht, als viele User Stories fertig wurden und neue reinkamen. Selbstverständlich mussten hier erst die alten gemergt werden.

Sollte eine Pull Request nicht direkt angenommen werden, müssen Probleme gelöst werden. Unser Ideal sah so aus, dass der Reviewer und ein Assignee zusammen den Code entsprechend refaktorisieren.

**Retrospektive:** Nach der Fertigstellung unserer Funktionalität kommen wir zusammen, um uns auf das Kundentreffen vorzubereiten und eine geeignete Präsentation auszuarbeiten.

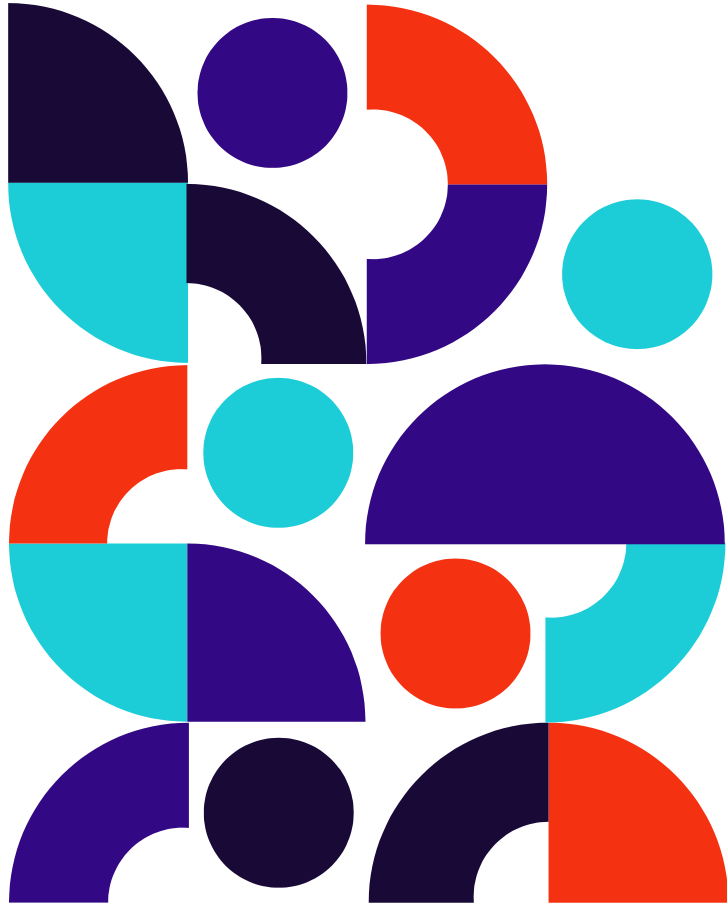
Weiterhin lassen wir die letzten zwei Wochen Revue passieren und nutzen dafür ein kanban-artiges Board, wo wir folgende Dinge festhalten: Was lief gut? Was lief in Ordnung? Was braucht Verbesserung? Was sind unsere nächsten Schritte, um uns zu verbessern?

Dabei achten wir auf Konstruktivität. Wir arbeiten konkrete Schritte aus, die wir im nächsten Sprint nutzen werden, um unsere Probleme zu verringern.





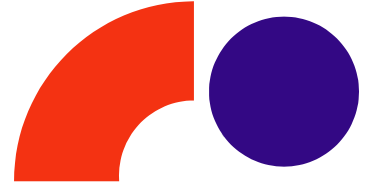
# Praktiken im Detail



# Reflexion



# Warum und Wie?



# Herausforderungen

- Schwierigkeiten werden in den nächsten Sprint weitergetragen
  - Beispiel: Bottleneck Mergen
- Absprachen/ Anforderungen nicht klar genug formuliert
- Dokumentation
  - Wie hält man Dinge so fest, dass sie wertbringend sind?



# Zufriedenheitsformular

## Zufriedenheitsformular

Woche 1

I am enthusiastic about the work that I do for my team

1 2 3 4 5 6 7 8 9 10

trifft gar nicht zu ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ trifft voll zu

I find the work that I do for my team of meaning and purpose

1 2 3 4 5 6 7 8 9 10

trifft gar nicht zu ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ trifft voll zu

I am proud of the work that I do for my team

1 2 3 4 5 6 7 8 9 10

trifft gar nicht zu ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ trifft voll zu

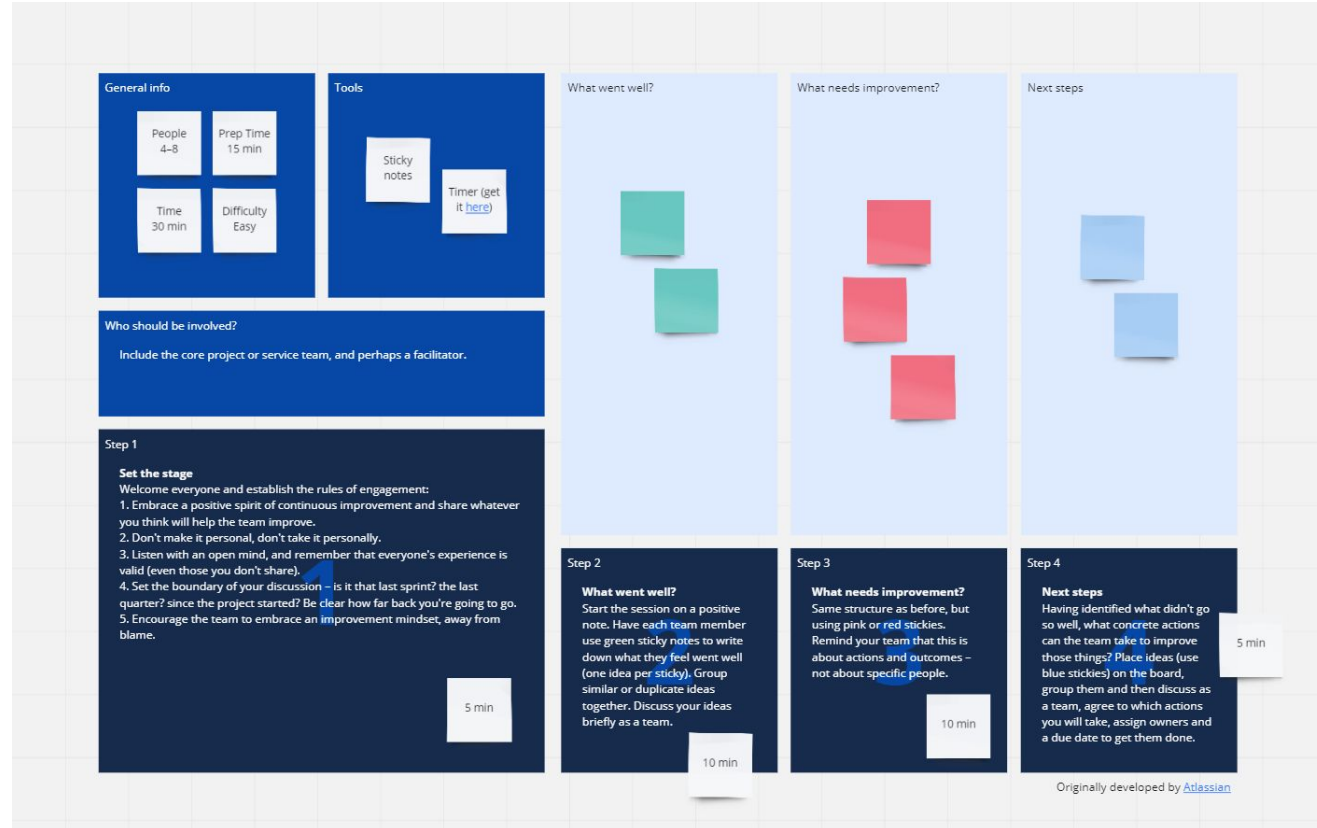
Referenz:

<https://medium.com/the-liberators/agile-teams-dont-use-happiness-metrics-measure-team-morale-3050b339d8af>





# Reflexion



# Reflexion

Sprint Nr.  
3

## General info

People 6

Time  
30 min

1 Sprint (2 Weeks)

## Tools

## Mandatory Topics

Time  
Management

Testing,  
TDD

Refactoring

PP

Docu

Vortrag

Merging

## What went well?

User Story richtig  
geschätzt  
II

Updates in  
Wiki-  
Channel  
man/festlegt

PR-Review hat zu  
verschiedensten  
Perspektiven beigetragen  
(II)

Features  
wurden fertig

Git + Review  
Feedback für  
Refactoring

## What went okay?

Funktionalität  
ist da, aber  
Zeit für Testing  
mit einplanen!

100%  
Codecoverage

Fortschritte beim  
Vortrag, aber nicht im  
Zeitlimit geblieben

Review-  
prozess  
endlich in Gang,  
aber zu viel  
Overhead Time

## What needs improvement?

Verzögerungen  
beim Mergen

Issues bauen  
sich schneller  
auf = sind  
verzögert

viele Branches die  
noch nicht zusammen  
funktionieren

Zeiteinschätzen für  
Issues mit eher  
neueren  
Themengebieten

Zeitman-  
agement (M-  
eetings)

Dokume-  
ntations  
wirrwarr

## Next steps

Mergen  
priorisieren!

vorher sich mit  
Interfaces und  
Eigenschaften  
seiner neuen  
Teildomäne  
beschäftigen

Developer aus  
der jeweiligen  
Teildomäne  
rekrutieren

realistic  
Sprint  
Points

Git, Kanban  
Struktur

Dev-  
Branch

## Step 1

### Set the stage

rules of engagement:

1. positive spirit of continuous improvement
2. don't take it personally.
3. Listen with an open mind, and remember that everyone's experience is valid (even those you don't share).
4. Set the boundary of your discussion – is it that last sprint? the last quarter? since the project started? Be clear how far back you're going to go.
5. Encourage the team to embrace an improvement mindset, avoid blame.

5 min

## Step 2

### What went well?

- positive note.
- each team member use green sticky notes (one idea per sticky).
- Group similar or duplicate ideas together. Discuss your ideas briefly as a team.

10 min

## Step 3

### What needs improvement?

-pink or red stickies.

- actions and outcomes – not about specific people.

10 min

## Step 4

### Next steps

concrete actions to improve

- group them and then discuss as a team,

agree to which actions you will take, assign owners and a due date to get them done.

5 min

# Reflexion mit Kanban (Beschreibung)

- Zufriedenheitsformular ist fehlgeschlagen, da schwer auswertbar
- Bisherige eingeräumte Zeit für Reflexion reicht nicht aus
- Trotz bereits zeitintensiver Prozesse haben wir uns zusätzlich für ein 30-minütiges Reflexion-Treffen entschieden
  - Reflexion über Praktiken, Sprint, Wohlbefinden im Team, Zusammenarbeit, Große Probleme
- Vorlage als Grundgerüst genutzt → Erweiterung der Vorlage durch:
  - 1. Welche Themen wollen wir für die Reflexion aufgreifen?
  - 2. Festlegung fester Zeiten für jeden Teilbereich
- 10 Minuten je für: Was lief gut? Was lief ok? Was braucht Verbesserung?
- Zuletzt Schwierigkeiten und Herausforderungen zusammentragen
- Pfeile für Kausalität verschiedener Punkte einfügen
- Diskussion über Probleme abschließen mit: Was muss konkret getan werden + Verantwortlichkeiten zuordnen
- 
- VORHER: PROBLEME SIND EHER UNTERGEGANGEN, evtl. noch notiert ABER häufig keine konkreten Lösungen + VERANTWORTLICHKEITEN z.B. "jeder macht sich mal Gedanken" " Wir probieren xy mal aus" → konkreter machen + notieren



# Gelernt

*„Die Definition von Wahnsinn ist, immer das Gleiche zu tun und andere Ergebnisse zu erwarten.“ - Albert Einstein*

- Reflexion von großer Bedeutung
  - ständige Anpassung
  - kann sehr zur Motivation beitragen
  - Zeitrahmen von Anfang an setzen
  - Zwischenmenschlicher Austausch nicht durch Formular ersetzbar
- Die richtigen Tools können unterstützen (auf eigene Bedürfnisse anpassen)

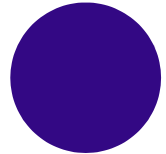
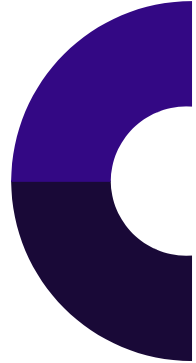
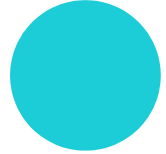


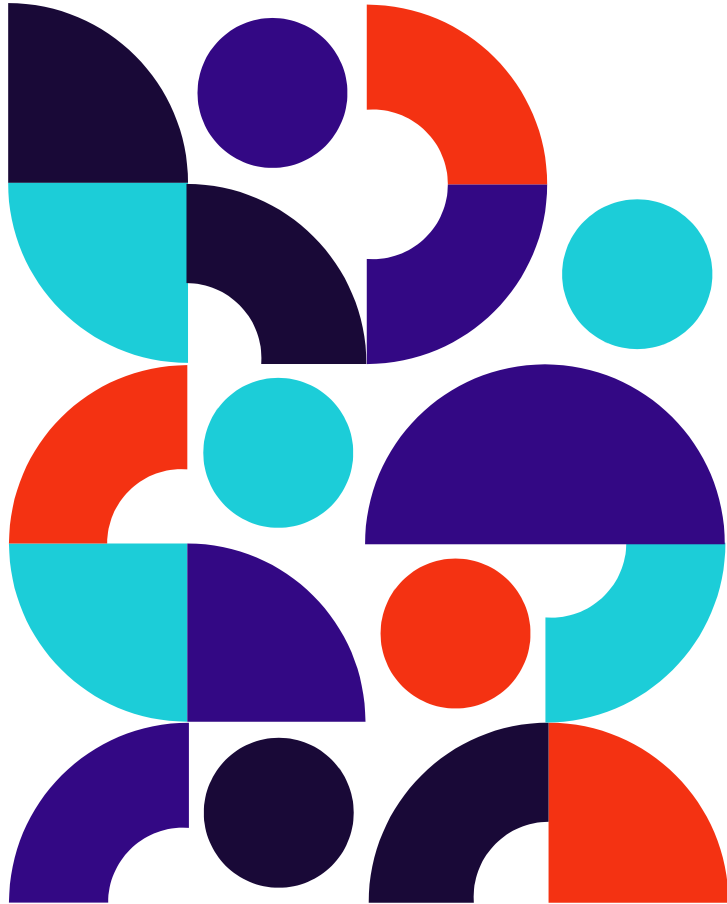


# Gelernt

*„Die Definition von Wahnsinn ist, immer das Gleiche zu tun und andere Ergebnisse zu erwarten.“ - Albert Einstein*

- Kommunikation muss gelernt werden
- Verantwortlichkeiten konkret verteilen
- Wichtige Infos sollen visuell aufgearbeitet und für jeden leicht zugängliche sein
- Reflexion von großer Bedeutung
  - ständige Anpassung
  - kann sehr zur Motivation beitragen
  - Zeitrahmen von Anfang an setzen
  - Zwischenmenschlicher Austausch nicht durch Formular ersetzbar
- Die richtigen Tools können unterstützen (auf eigene Bedürfnisse anpassen)





# Testing

# Testing im Legacy Code

- Tests für Getter, Setter und Konstanten
- Fehlen von Tests, bei denen Probleme in älteren Squeak-Versionen aufgefallen wären
- Testmethoden mit viel Code
- keine setUp und tearDown Methoden

```
testTargetPositionSetter
| p tempPosition |
p := AnimationsEditorPropertyPosition new.
tempPosition := 200@100.
p targetPosition: tempPosition.

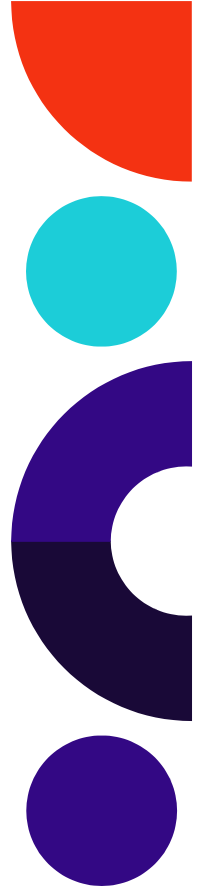
self assert: (p targetPosition = tempPosition).
```

Negativbeispiel



# Testing zu Beginn des Projekts

- Achten auf kleine Tests
  - Verwendung von setUp und tearDown
- Mock-Objekte
  - kleine Mock-Objekte
  - Trennung von Verantwortlichkeiten
- Schwierigkeiten, direkt mit TDD anzufangen



# Wie und was wollen wir testen?

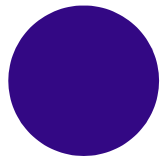
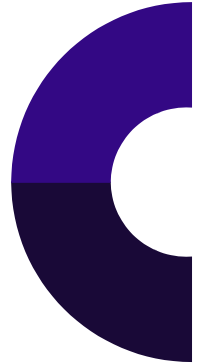
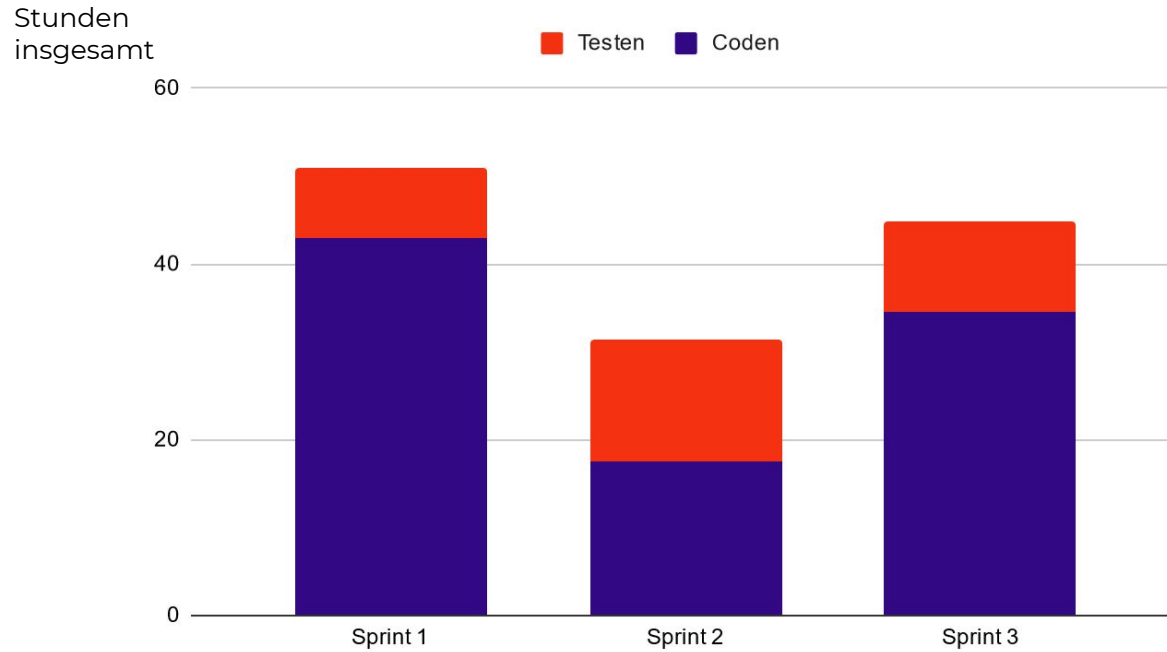
- Bedingte Aussagekraft von Code Coverage
- Probleme mit Unit-Tests
  - Basierend auf dem Verständnis der Entwickler
  - Abdeckung aller Fälle ist unmöglich

## Konsequenzen

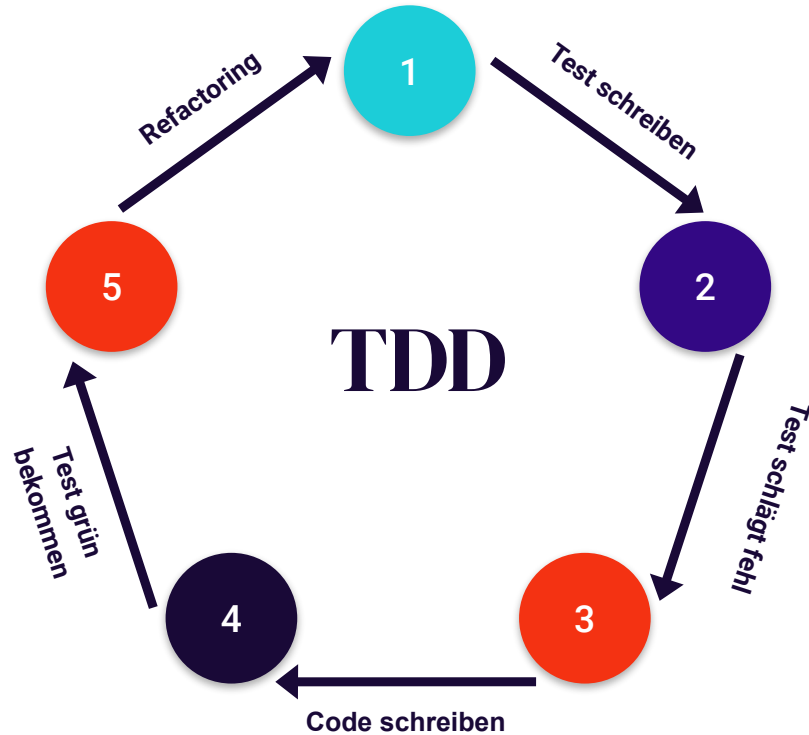
- Fokus auf funktionales Testen
- Tests priorisieren
- Acceptance Tests nicht vernachlässigen



# Testing - Zeiten



# Test-Driven-Development

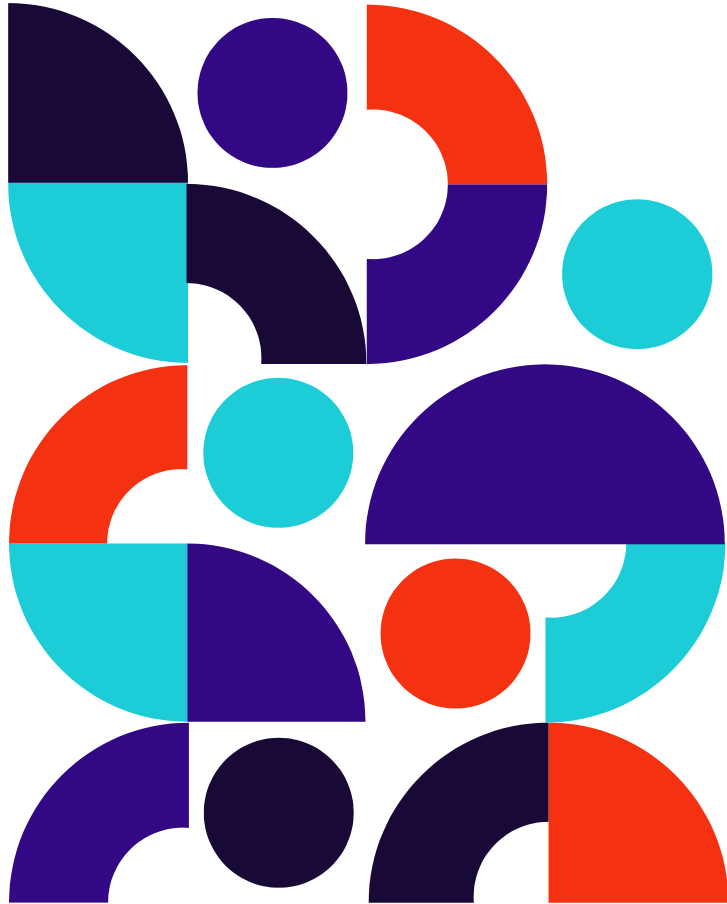


# Testing - unsere Erfahrung

Was gut gelaufen ist	Womit wir Probleme hatten
Fokus auf funktionales Testen	TDD schwer umsetzbar, wenn man keinen konkreten Lösungsansatz hat
Fehlende Funktionalität ist aufgefallen	konsequent immer "test-first" anwenden
Probleme wurden durch TDD früher erkannt	



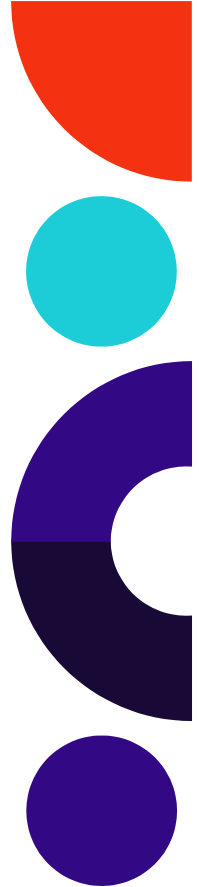




# Refactoring

# Refactoring

- Legacy Code Refactoring
  - wenn neue Features es nötig gemacht haben
- Balance: simple Methodenaufrufe vs. übersichtlicher Plan
  - Gefahr: Patterns Happy!



# Beispiel

```
key between: aStartEndProperty startTime: aStartEndProperty endTime.  
  ifTrue: [  
    animations add:  
      (AnimPropertyAnimation new  
        duration: aKey - aLastKey;  
        property: #position;  
        startValue: aLastTargetValue;  
        endValue: aValue;  
        target: aMorph;  
        state: #paused;  
        yourself).  
  ].  
  ifFalse: [  
    updateInformationText: 'Keyframe time out of bounds.'].
```

Decompose Conditional

Extract Method

Magic String

```
self insideBounds  
  ifTrue: [  
    animations add:  
      (self createAnimationFor: aMorph)].  
  ifFalse: [  
    updateInformationText: self messageOutOfBounds].
```



# Refactoring - Teameinfluss

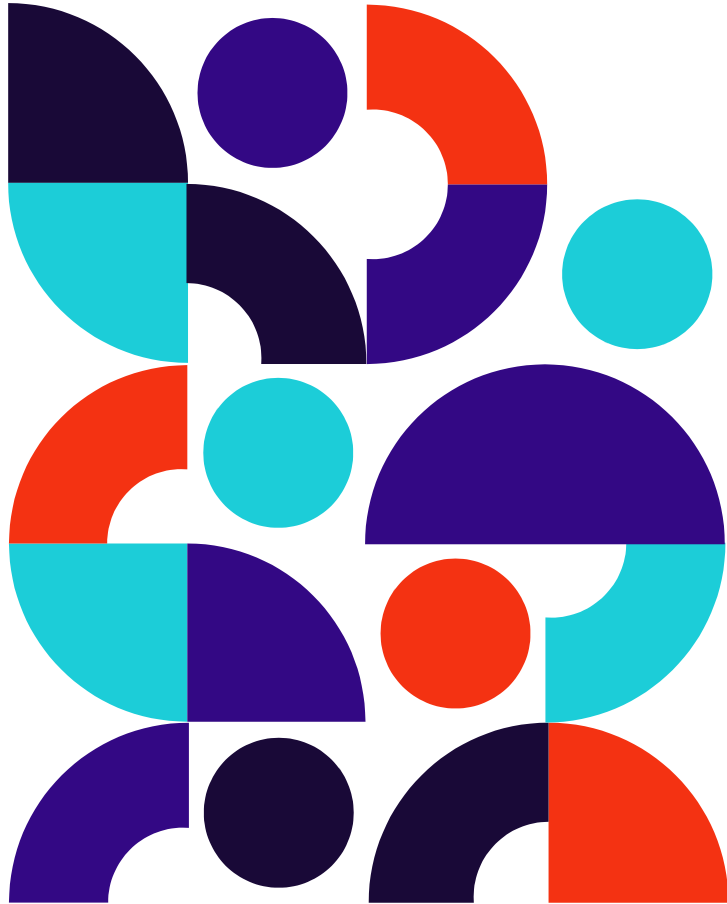
- Refactoring war eingebaut in Review-Prozess
- Durch Refactoring betroffene Methoden: Wer nimmt Veränderungen in anderen beeinflussten Bereichen vor?
- Collective Code Ownership
  - Immer neue Teams
  - Teammeetings



# Refactoring - Reflektion & Aussicht

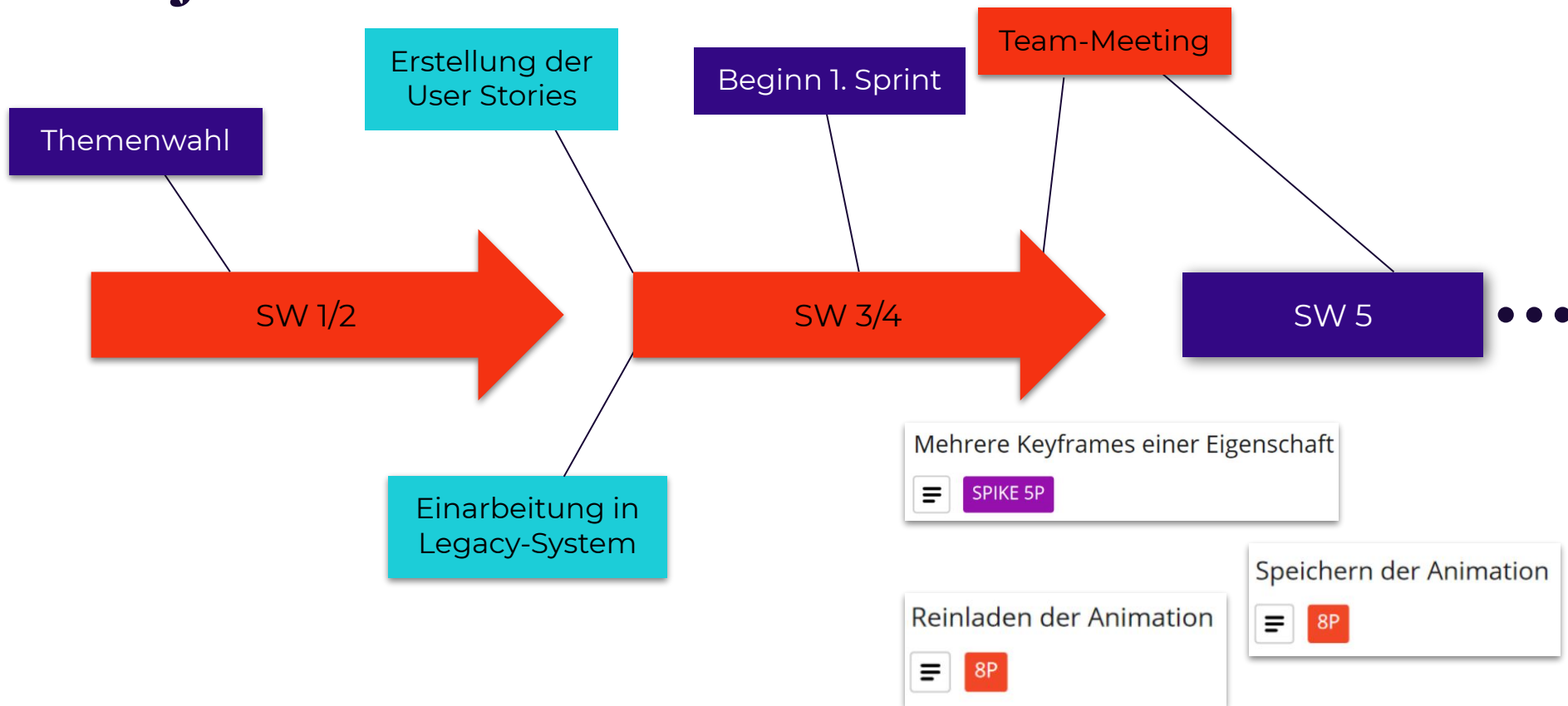
- Lokale Code-Snippets wurden gut reflektiert
- Globale Einflüsse vernachlässigt
  - Was passiert bei Zusammenführung?
  - Vernachlässigung der Architekturebene
  - Zu mächtige Klasse entstanden



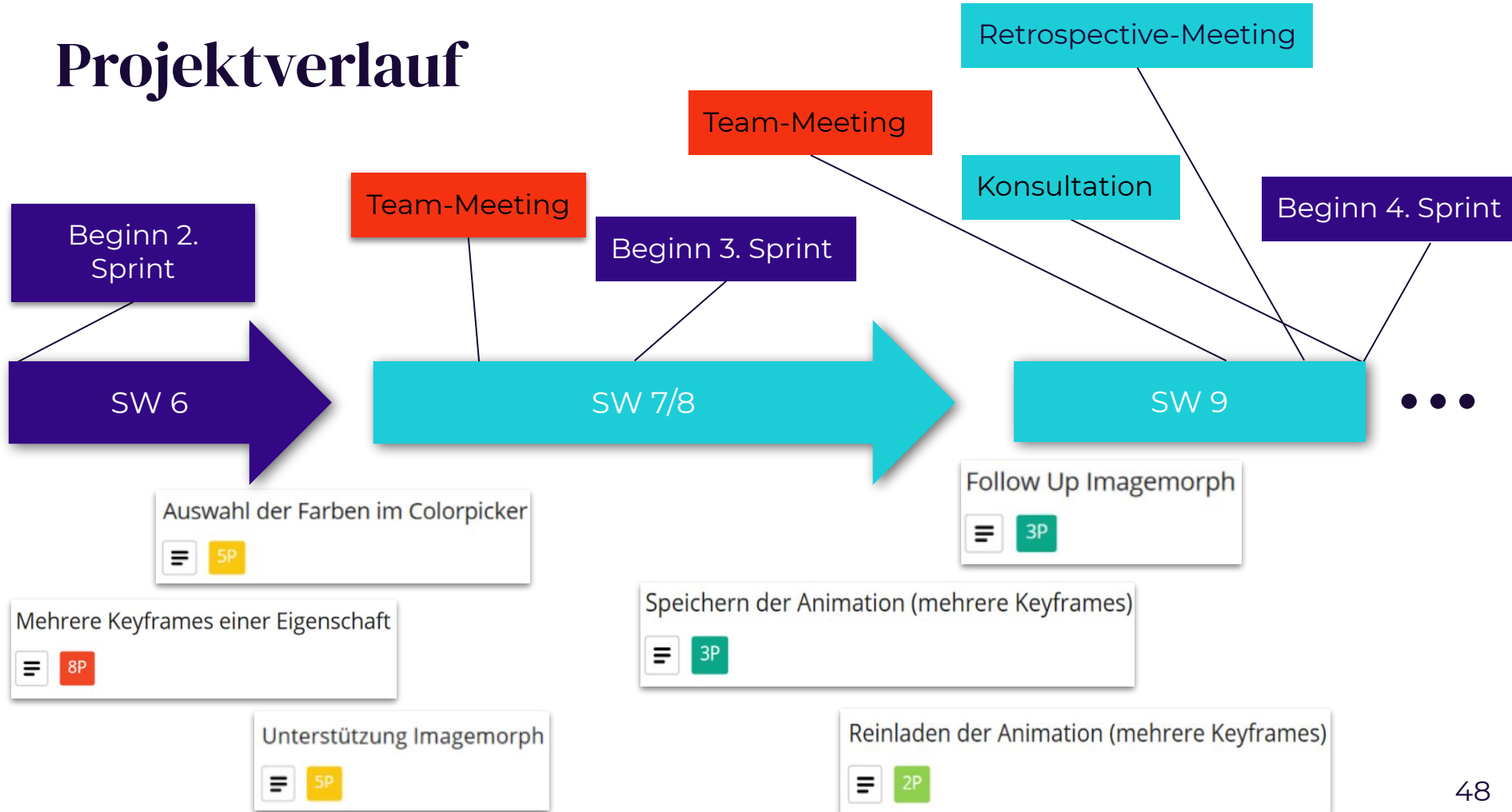


# Metriken und Projektverlauf

# Projektverlauf



# Projektverlauf





# Nächste Features

Nearest Interpolation



SPIKE 5P

Keyframes werden automatisch gesetzt  
REQUIRES: Mehrere Keyframes einer Eigenschaft, Timeline



Requires

Timeline



SPIKE 8P

Quadratische Interpolation



SPIKE 5P

Unterstützung Polygonmorph

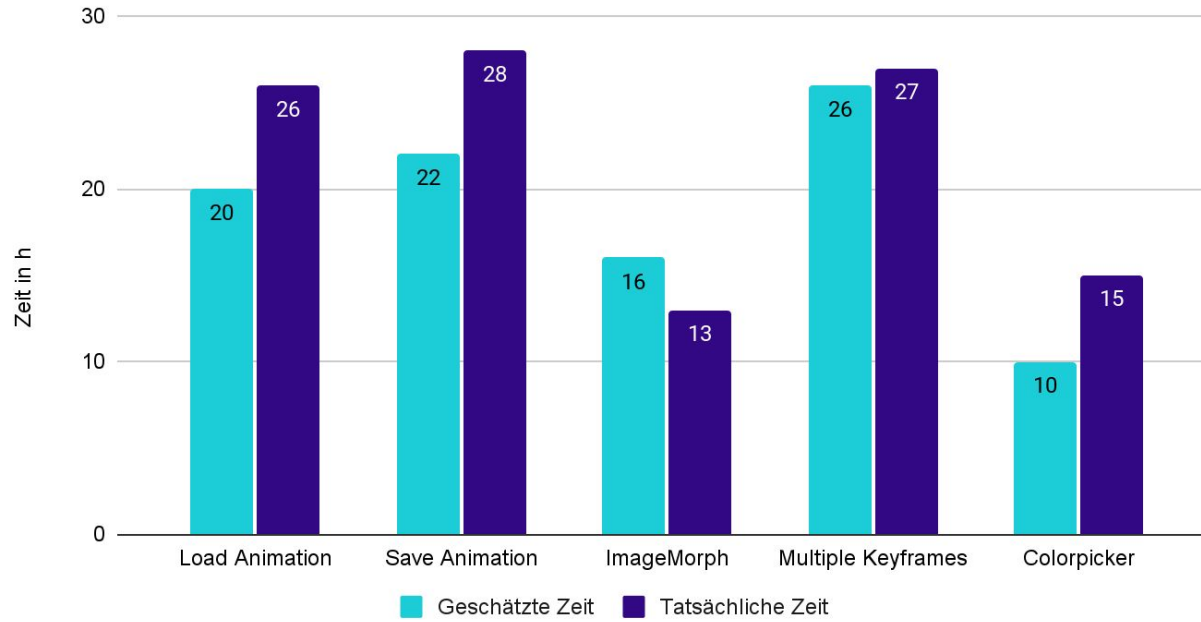


3P



HIDDEN

# PP-Zeit für User Stories (Testing & Coding)



# Overview Pyramide & Co.

		ANDC	0,6
		AHH	0,3
		9/10	NOP 1 / 1
15,2/ 24,8		NOC	9 / 10
8,3/ 5,8	NOM	137 / 248	
LOC 1134 / 1440			

Test Logic

- Code-Coverage: 76%
- Anzahl Tests: 84 (ohne SetUp/Teardown)
  - davon 39 von uns geschrieben



# Schlussfolgerungen

- **Size & Complexity**

- Übersichtliche Methoden-Komplexität
- God-Class-Eigenschaft im System vorhanden → Trennung der Verantwortlichkeiten nötig

		9/10	NOP	1 / 1
	15,2/ 24,8	NOC	9 / 10	
8,3/ 5,8	NOM	137 / 248		
LOC 1134 / 1440				

- **System Inheritance**

- Architektur breit gegliedert → Property Decorator vom Legacy-Team weiterführen

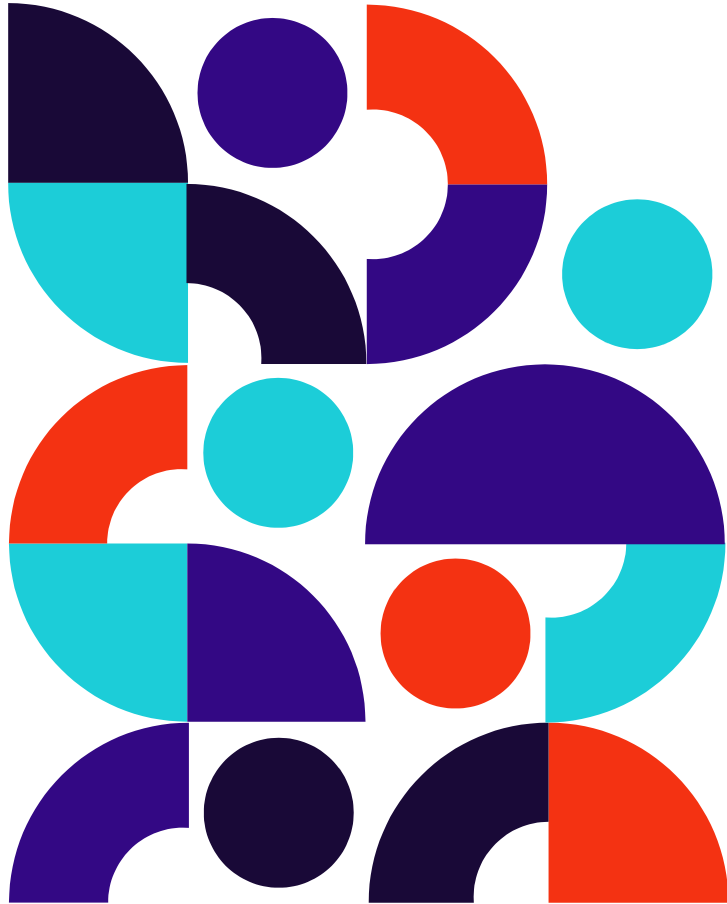
ANDC	0,6
AHH	0,3



# Prozessmetriken

- Velocity
  - Mitschleppen in den nächsten Sprint
  - Abhängigkeiten zwischen Features
- Einarbeitungszeit in Legacy Code: ca. 4-5h
  - ausreichend für System mit knapp 1000 LOC
- Doku-Zeit: ca. halbe Stunde pro Woche (Discord Work-Channel & unter jeweiligen User Stories im Doc: Probleme, Fragen notiert)
  - Tests sind Grundlage für die Doku





# Abschließende Gedanken

Was ist gut gelaufen?	Was ist schlecht gelaufen?
Reflexion und Anpassung	Sustainable Development
Pair Programming-Rotation	globale Codestruktur
Kommunikation	

