Git – das Kartenspiel

Philipp Müller

25. Juni 2024

Dieses Spiel parodiert die agile Software-Entwicklung. Die Spieler bilden dabei ein Team von Entwicklern, die gemeinsam eine Software weiterentwickeln. Ziel des Spiels ist es jedoch, möglichst viele der eigenen Codezeilen durchzubringen und die anderen Spieler genau daran zu behindern.

1 – Aufbau

Die Karten mit der grünliche Rückseite werden gut gemischt und bildet den Zugstapel. Jeder Spieler erhält drei davon auf die Hand, die sog. Handkarten. Diese entsprechen dem lokalen Git-Arbeitsverzeichnis des Spielers.
Die Karten mit der rötlichen Rückseite werden auch gut gemischt und bilden den Issue-Stapel. Eine Issue-Karte wird gezogen und aufgedeckt. Sie bildet das Ziel der ersten Spielrunde.

2 – Ablauf

Das Spiel wird in Runden gespielt, die jeweils der Entwicklung einer Version der Software entsprechen. Sie beginnen mit dem Aufdecken einer Issue-Karte, die das Ziel der Runde angibt, und Enden mit einer Lieferung. Dazwischen spielen alle Spieler reihum immer genau eine Handkarte aus und sagen dabei das, was auf der Karte in Anführungszeichen („. . .“) steht.
Es gibt drei Arten von Handkarten:

- **Commit**-Karten werden ausgelegt und müssen dann mit einem Spielermarker markiert werden, damit man später weiß, wer welche Punkte bekommt. Ihr Zahlwert entspricht der Anzahl Codezeilen. Die Reihe der ausgelegten Commit-Karten entsprechen dem Remote Repository von Git.
- **Release**-Karten beenden die aktuelle Runde, die dann **ausgewertet** wird (s. u. 2.2). Sie können nur ausgespielt werden, wenn das aktuelle Ziel erreicht ist, d. h. wenn die Summe der ausgelegten Zahlwerte dem Wert der Issue-Karte entspricht oder diesen übertrifft.
- Sonstige Karten werden entsprechend ihrer Aufschrift ausgewertet und landen danach meistens auf dem Abwurfstapel.

Bestimmte Karten dürfen nur unter bestimmen Bedingungen ausgespielt werden. Falls jemand an der Reihe ist und keine seiner Handkarten ausspielen kann, legt er sie alle auf den Abwurfstapel.
Am Ende des Zuges **stockt** man seine Handkarten wieder auf (s. u. 2.1) und dann ist der nächste Spieler an der Reihe.
Die Spielrunde ist beendet, sobald jemand eine Release-Karte ausspielt. Dann wird die aktuelle Runde **ausgewertet** (s. u. 2.2) und danach beginnt die nächste Spielrunde mit dem Aufdecken der nächsten Issue-Karte oder das Spiel ist ganz beendet (s. u. 3).

2.1 – Aufstocken

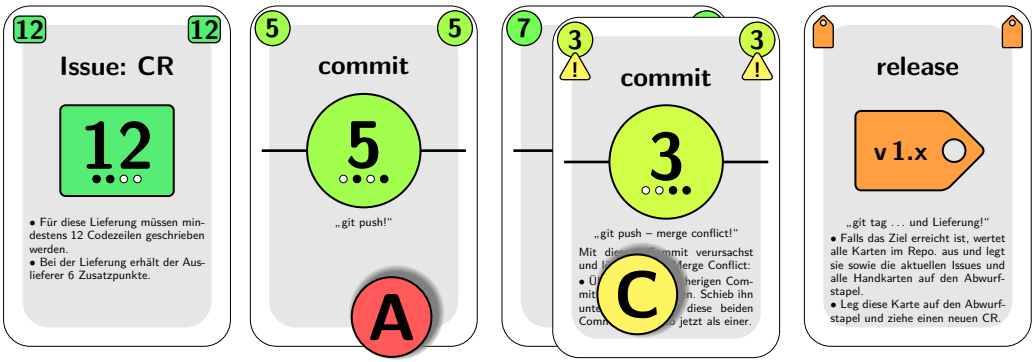
Falls man am Ende seines Zuges weniger als drei Handkarten hat, zieht man neue Karten vom Zugstapel bis man wieder drei auf der Hand hat.

2.2 – Auswerten

Sobald jemand eine Release-Karte ausspielt, wird eine neue Version der Software geliefert. D. h. die aktuelle Runde wird ausgewertet. Jeder Spieler erhält Punkte entsprechend der Codezeilen all seiner Commits im Remote Repository, also zwischen Issue und Lieferung. Der Spieler der Release-Karte erhält zusätzliche Punkte wie auf der Issue-Karte angegeben.
Nach der Auswertung werden alle ausgelegten und alle Handkarten abgeworfen. Jetzt können die Karten neu gemischt werden. Danach wird eine neue Issue-Karte ausgelegt und jeder Spieler erhält wieder drei Handkarten. Der Spieler nach dem Spieler, der die Release-Karte ausgespielt hat, beginnt die neue Runde.

Beispiel für eine Auswertung

Das Remote Repository (die ausgelegten Karten) sieht folgendermaßen aus:



Spieler A hat die Release-Karte ausgespielt.

- Spieler A erhält 11 Punkte: Zahlkarte 5 plus 6 Extrapunkte fürs Release.
- Spieler B erhält 0 Punkte: Seine Zahlkarte 7 wurde von Spieler C geklaut!
- Spieler C erhält 10 Punkte: Zahlkarte 7 und Zahlkarte 3 (Merge Conflict).

Die Punkte sollten in einer kleinen Tabelle aufgeschrieben werden:

Runde	A	B	C
v 1	11	0	10
v 2			
v 3			

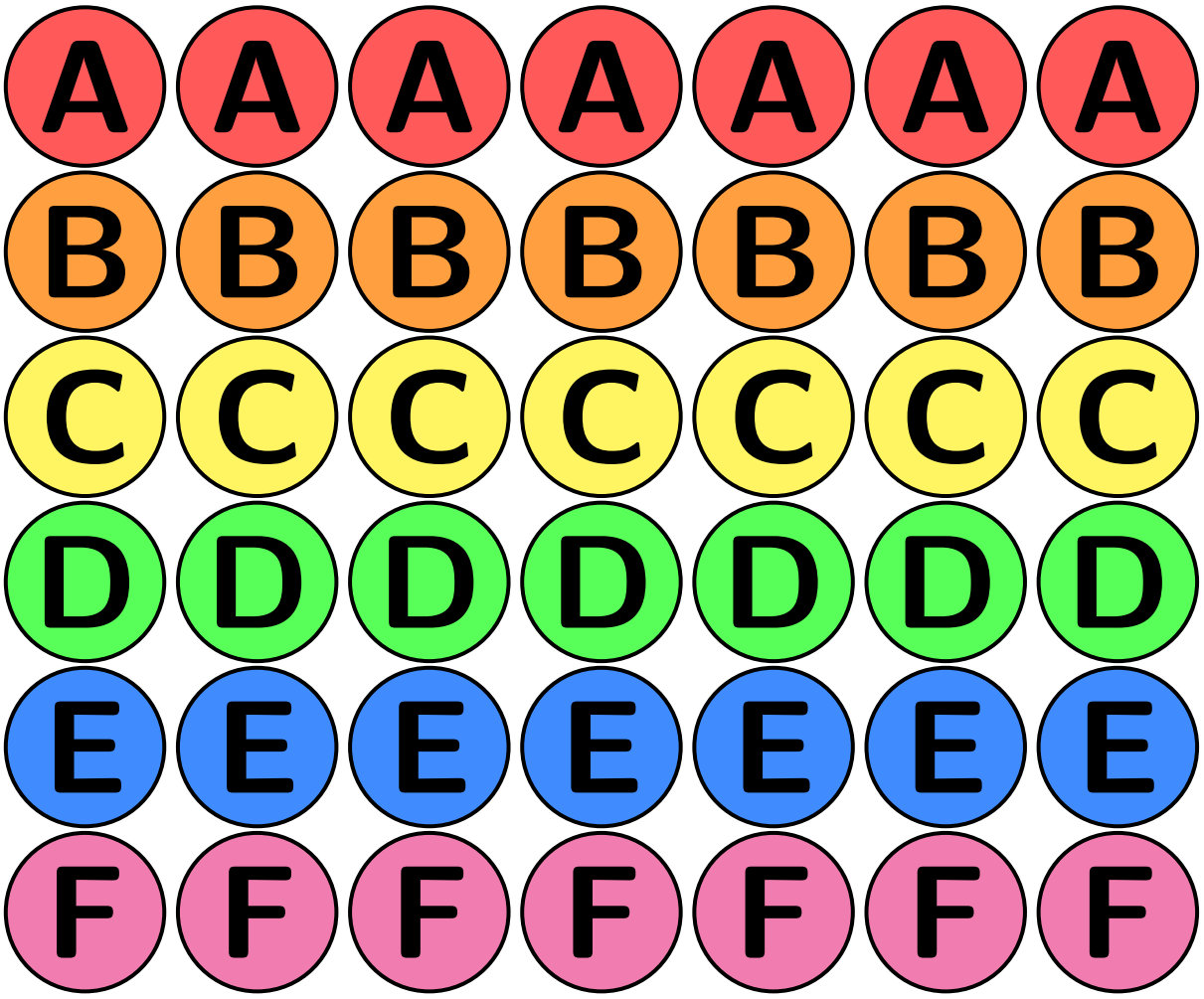
3 – Spielende

Am Anfang sollten alle Teilnehmer festlegen, wann das Spiel beendet ist (Definition of Done), z. B. nach 5 Lieferungen oder sobald ein Spiel mindestens 100 Punkte erreicht hat. Gewonnen hat dann selbstverständlich der Spieler mit der höchsten Punktezahl.

4 – Druckvorlagen

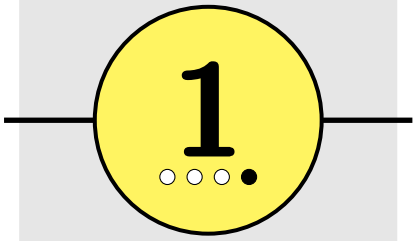
Diese und die folgenden Seiten sind zum Ausdrucken und Ausschneiden vorgesehen. Zum Drucken wird dickes Papier empfohlen, z. B. 160 g-Papier. Die Karten sollten natürlich beidseitig ausgedruckt werden. Die Ränder links und rechts sind gleich breit, so dass Vorder- und Rückseite immer aufeinander treffen.
Statt der Spielermarker auf dieser Seite können auch irgendwelche Spielfiguren oder sonstige Chips, Münzen o. ä. zum markieren der ausgelegten Karten verwendet werden.

Spielermarker



1 1

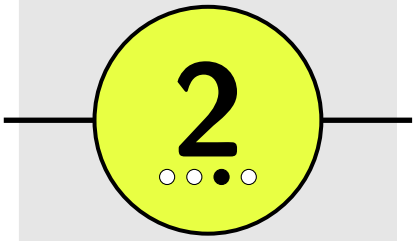
commit



„git push!“

2 2 3

commit



„git push!“

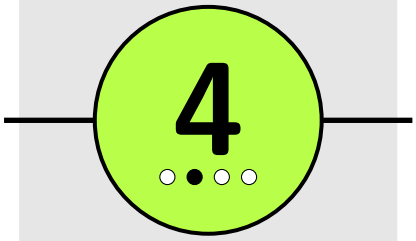
3

commit

„git push!“

4 4

commit



„git push!“

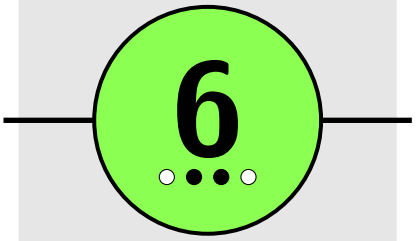
5 5 6

commit

„git push!“

6

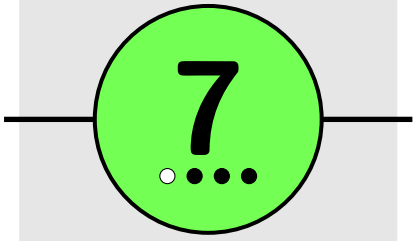
commit



„git push!“

7 7

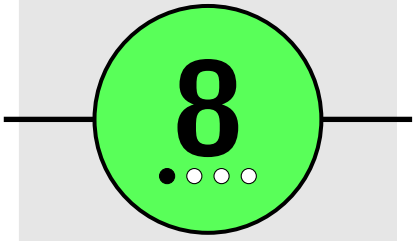
commit



„git push!“

8 8 9

commit



„git push!“

9

commit

„git push!“

git

git

git

git

git

git

git

git

git

10 10

commit

10

„git push!“

12 12

commit

12

„git push!“

0 ! 0 !

commit

0

„git push – refactoring!“

• Übernimm die vorherigen drei Commits als deine eigenen. Schieb sie unter diese Karte, alle diese Commits gelten ab jetzt als einer.

2 ! 2 !

commit

2

„git push – merge conflict!“

Mit diesem Commit verursachst und löst du einen Merge Conflict:

• Übernimm den vorherigen Commit als deinen eigenen. Schieb ihn unter diese Karte, diese beiden Commits gelten ab jetzt als einer.

3 ! 3 !

commit

3

„git push – merge conflict!“

Mit diesem Commit verursachst und löst du einen Merge Conflict:

• Übernimm den vorherigen Commit als deinen eigenen. Schieb ihn unter diese Karte, diese beiden Commits gelten ab jetzt als einer.

4 ! 4 !

commit

4

„git push – merge conflict!“

Mit diesem Commit verursachst und löst du einen Merge Conflict:

• Übernimm den vorherigen Commit als deinen eigenen. Schieb ihn unter diese Karte, diese beiden Commits gelten ab jetzt als einer.

2 ? 2 ?

commit

2

„git push – merge request!“

Mit diesem Commit *bittest* du deinen Kollegen um ein Code Review, dass er natürlich auch durchführt:

• Der nächsten Spieler muss ein Revert ausspielen oder einmal aussetzen.

3 ? 3 ?

commit

3

„git push – merge request!“

Mit diesem Commit *bittest* du deinen Kollegen um ein Code Review, dass er natürlich auch durchführt:

• Der nächsten Spieler muss ein Revert ausspielen oder einmal aussetzen.

4 ? 4 ?

commit

4

„git push – merge request!“

Mit diesem Commit *bittest* du deinen Kollegen um ein Code Review, dass er natürlich auch durchführt:

• Der nächsten Spieler muss ein Revert ausspielen oder einmal aussetzen.

git

git

git

git

git

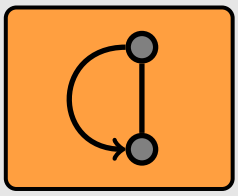
git

git

git

git

revert

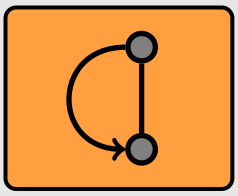


„git revert!“

Der gerade geschriebene Code ist fehlerhaft und muss wieder weg:

- Muss unmittelbar auf einen Commit folgen und negiert ihn.
- Leg diese Karte und den Commit auf den Abwurfstapel.

revert

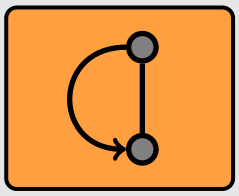


„git revert!“

Der gerade geschriebene Code ist fehlerhaft und muss wieder weg:

- Muss unmittelbar auf einen Commit folgen und negiert ihn.
- Leg diese Karte und den Commit auf den Abwurfstapel.

revert

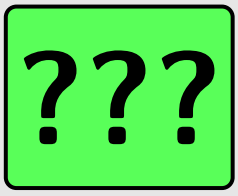


„git revert!“

Der gerade geschriebene Code ist fehlerhaft und muss wieder weg:

- Muss unmittelbar auf einen Commit folgen und negiert ihn.
- Leg diese Karte und den Commit auf den Abwurfstapel.

help

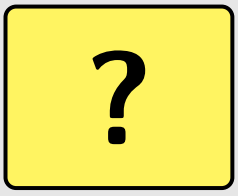


„git help!“

Du weißt gerade nicht weiter und brauchst Hilfe:

- Leg diese Karte auf den Abwurfstapel.
- Zieh drei Handkarten zusätzlich, nachdem du **aufgestockt** hast.

status

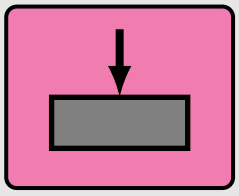


„git status!“

Du hast gerade keine Idee und brauchst erstmal einen Überblick:

- Leg diese Karte auf den Abwurfstapel.
- Zieh eine Handkarte zusätzlich, nachdem du **aufgestockt** hast.

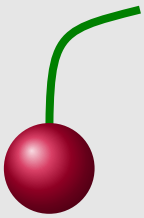
stash



„git stash!“

- Leg bis zu zwei Handkarten verdeckt vor Dir ab.
- Sie können weiterhin wie Handkarten ausgespielt werden, zählen aber nicht beim **Aufstocken** mit und gehen bei einem Release nicht verloren.

cherry-pick



„git cherry-pick!“

- **Stocke** zuerst deine Handkarten auf.
- Wähle dir danach bei einem beliebigen Spieler aus seinen Handkarten eine aus und füge sie deinen hinzu oder spiele sie sofort aus.

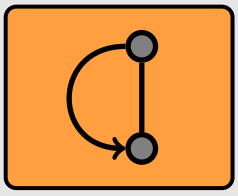
commit

„git push – merge request!“

Mit diesem Commit *bittest* du deinen Kollegen um ein Code Review, dass er natürlich auch durchführt:

- Der nächsten Spieler muss ein Revert ausspielen oder einmal aussetzen.

revert



„git revert!“

Der gerade geschriebene Code ist fehlerhaft und muss wieder weg:

- Muss unmittelbar auf einen Commit folgen und negiert ihn.
- Leg diese Karte und den Commit auf den Abwurfstapel.

git

git

git

git

git

git

git

git

git

release

v 1.x

„git tag ... und Lieferung!“

- Falls das Ziel erreicht ist, wertet alle Karten im Repo. aus und legt sie sowie die aktuellen Issues und alle Handkarten auf den Abwurfstapel.
- Leg diese Karte auf den Abwurfstapel und ziehe einen neuen CR.

release

v 1.x

„git tag ... und Lieferung!“

- Falls das Ziel erreicht ist, wertet alle Karten im Repo. aus und legt sie sowie die aktuellen Issues und alle Handkarten auf den Abwurfstapel.
- Leg diese Karte auf den Abwurfstapel und ziehe einen neuen CR.

release

v 1.x

„git tag ... und Lieferung!“

- Falls das Ziel erreicht ist, wertet alle Karten im Repo. aus und legt sie sowie die aktuellen Issues und alle Handkarten auf den Abwurfstapel.
- Leg diese Karte auf den Abwurfstapel und ziehe einen neuen CR.

release

v 1.x

„git tag ... und Lieferung!“

- Falls das Ziel erreicht ist, wertet alle Karten im Repo. aus und legt sie sowie die aktuellen Issues und alle Handkarten auf den Abwurfstapel.
- Leg diese Karte auf den Abwurfstapel und ziehe einen neuen CR.

release

v 1.x

„git tag ... und Lieferung!“

- Falls das Ziel erreicht ist, wertet alle Karten im Repo. aus und legt sie sowie die aktuellen Issues und alle Handkarten auf den Abwurfstapel.
- Leg diese Karte auf den Abwurfstapel und ziehe einen neuen CR.

5 5

Issue: Bug

+5

„Ich habe einen Fehler gefunden, der auch noch behoben werden muss.“

- Leg die Karte zum aktuellen CR.
- Für die aktuelle Lieferung müssen noch 5 Codezeilen zusätzlich geschrieben werden.

7 7

Issue: Bug

+7

„Ich habe einen Fehler gefunden, der auch noch behoben werden muss.“

- Leg die Karte zum aktuellen CR.
- Für die aktuelle Lieferung müssen noch 7 Codezeilen zusätzlich geschrieben werden.

8 8

Issue: Bug

+8

„Ich habe einen Fehler gefunden, der auch noch behoben werden muss.“

- Leg die Karte zum aktuellen CR.
- Für die aktuelle Lieferung müssen noch 8 Codezeilen zusätzlich geschrieben werden.

10 10

Issue: Bug

+10

„Ich habe einen Fehler gefunden, der auch noch behoben werden muss.“

- Leg die Karte zum aktuellen CR.
- Für die aktuelle Lieferung müssen noch 10 Codezeilen zusätzlich geschrieben werden.

git

git

git

git

git

git

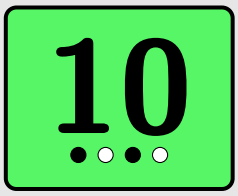
git

git

git

10 10

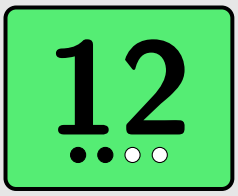
Issue: CR



- Für diese Lieferung müssen mindestens 10 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 5 Zusatzpunkte.

12 12

Issue: CR



- Für diese Lieferung müssen mindestens 12 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 6 Zusatzpunkte.

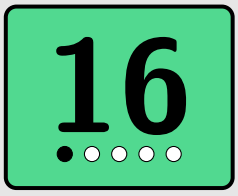
15 15

Issue: CR

- Für diese Lieferung müssen mindestens 15 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 7 Zusatzpunkte.

16 16

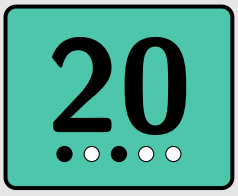
Issue: CR



- Für diese Lieferung müssen mindestens 16 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 8 Zusatzpunkte.

20 20

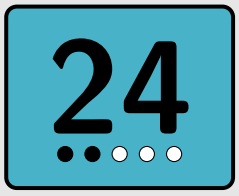
Issue: CR



- Für diese Lieferung müssen mindestens 20 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 10 Zusatzpunkte.

24 24

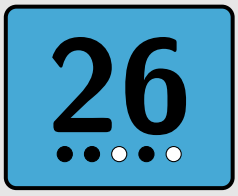
Issue: CR



- Für diese Lieferung müssen mindestens 24 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 12 Zusatzpunkte.

26 26

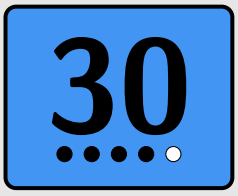
Issue: CR



- Für diese Lieferung müssen mindestens 26 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 13 Zusatzpunkte.

30 30

Issue: CR



- Für diese Lieferung müssen mindestens 30 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 15 Zusatzpunkte.

32 32

Issue: CR

- Für diese Lieferung müssen mindestens 32 Codezeilen geschrieben werden.
- Bei der Lieferung erhält der Auslieferer 16 Zusatzpunkte.

git

git

git

git

git

git

git

git

git