PS Angewandte Systemwissenschaften 1

Einheit 7: Projektsetup, Git & Szenarien

Gemeinsam Code entwickeln, Traffic-Modell testen, Gruppenprojekte starten

David Maier
Institut für Umweltsystemwissenschaften
Universität Graz

Heute

Git

Ein gemeinsamer Workflow für Code, Änderungen und Teamarbeit.

Szenarien

Vom animierten Modell zu reproduzierbaren Simulationsexperimenten.

Projektsetup

Repositories prüfen, Rollen klären, erste Arbeitspakete starten.

Ziel der Einheit: Am Ende kann jede Gruppe Code gemeinsam verwalten und erste Szenarien systematisch ausführen.

Warum Git jetzt wichtig ist

Gruppenprojekt

- Mehrere Personen arbeiten am selben Code
- Änderungen müssen nachvollziehbar bleiben
- Jede Gruppe braucht eine gemeinsame Version

Reproduzierbare Arbeit

- Code, Parameter und Ergebnisse gehören zusammen
- Fehler lassen sich rückgängig machen
- GitHub ist Backup und Übergabepunkt

Merksatz: Git speichert nicht nur Dateien. Git speichert Entscheidungen im Verlauf eines Projekts.

Repository und Commit

Repository

Ein Projektordner mit Gedächtnis. Der versteckte Ordner `.git` enthält die Historie des Projekts.

Die Datei `.gitignore` sagt Git, welche lokalen Dateien nicht gespeichert werden sollen.

Commit

Ein gespeicherter Snapshot mit kurzer Beschreibung. Jeder Commit sollte eine kleine logische Änderung enthalten.

```
Add scenario sweep command
```

Gute Commits machen sichtbar, wie aus einer Idee ein lauffähiges Modell entsteht.

Working Tree, Staging, Commit

Working Tree

Dateien, an denen ihr gerade arbeitet.

```
main.py
README.md
```

Staging

Bewusste Auswahl für den nächsten Commit.

```
git add main.py
```

Commit

Gespeicherter Zustand mit Nachricht.

```
git commit -m "Add run mode"
```

> Staging ist kein Extra-Schritt ohne Sinn. Es ist die Frage: Welche Änderung gehört wirklich zu diesem Commit?

Die Git-Befehle auf den Folien sind die Terminal-Version. Ihr könnt sie verwenden, aber ihr müsst nicht: dieselben Schritte gehen auch über die VS Code Oberfläche.

Lokal und Remote

Lokal

Die Arbeitskopie auf eurem Laptop. Hier schreibt, testet und committet ihr Code.

```
git status
git commit
```

Remote

Die gemeinsame Kopie auf GitHub. Von dort holt ihr Änderungen, dorthin teilt ihr eure Commits.

```
git pull
git push
```

Clone macht ihr einmal. **Pull** und **push** gehören zu jeder Arbeitssession.

Unser Kurs-Workflow

Vor dem Arbeiten

1. Repository in VS Code öffnen
2. Prüfen: Branch `main`
3. Pull oder Sync ausführen

Nach einer sinnvollen Änderung

1. Code testen
2. Änderungsvergleich ansehen
3. Dateien stagen
4. Commit mit klarer Nachricht
5. Push oder Sync ausführen

Wir arbeiten im Kurs direkt auf `main`. Das ist einfach, funktioniert aber nur mit kurzen Commits und guter Absprache.

VS Code: Der praktische Ablauf

Source Control

- Öffnen mit `Ctrl+Shift+G`
- Geänderte Dateien erscheinen unter *Changes*
- Klick auf Datei öffnet den Diff
- `+` verschiebt Dateien in *Staged Changes*

Commit und Sync

- Commit Message im Textfeld schreiben
- Commit ausführen
- Sync Changes oder Push klicken
- Bei Problemen zuerst lesen, dann im Team fragen

Diff bedeutet Änderungsvergleich: VS Code zeigt links die alte Version und rechts eure neue Version. So seht ihr vor dem Commit genau, was ihr geändert habt.

Wer lieber klickt statt Befehle eintippt, kann vollständig mit der VS Code Oberfläche arbeiten. Die Klickwege stehen im Git Cheatsheet.

Zusammenarbeit ohne Chaos

Konflikte vermeiden

- Vor jeder Session pullen
- Absprechen, wer welche Datei bearbeitet
- Kleine, fokussierte Commits machen
- Nicht lange lokal sammeln

Konflikte lösen

- VS Code zeigt betroffene Dateien an
- **Incoming:** Änderung von GitHub
- **Current:** eure lokale Änderung
- Gemeinsam entscheiden, was bleibt
- Merge abschließen, testen, committen

> Ein Merge Conflict ist kein Fehler im Projekt. Er ist ein Hinweis, dass zwei Personen dieselbe Stelle verändert haben.

Vom Modell zur Analyse

Szenarien & Sensitivität

Warum wir Modelle nicht nur anschauen, sondern systematisch experimentieren lassen.

Animation ist noch keine Analyse

Animation

Wir sehen, ob das Modell grundsätzlich funktioniert und welche Muster entstehen.

```
uv run main.py animate
```

Experiment

Wir speichern Daten, wiederholen Läufe und vergleichen Bedingungen kontrolliert.

```
uv run main.py run
```

Railsback & Grimm: Ein lauffähiges ABM ist ein virtuelles Labor. Wir verstehen es durch kontrollierte Simulationsexperimente.

Szenario und Replikat

Szenario

Eine konkrete Kombination aus Parameterwerten, Anfangsbedingungen und Modellannahmen.

```
slowdown_prob = 0.3
n_cars = 22
safe_distance = 2
```

Replikat

Derselbe Parameter-Setup noch einmal, aber mit anderen Zufallsereignissen, also einem anderen Seed.

```
seed = 1
seed = 2
seed = 3
```

> Bei stochastischen ABMs ist ein einzelner Lauf nur eine Beobachtung, nicht schon ein Ergebnis.

Was ist ein Random Seed?

Pseudozufall

Computer erzeugen Zufallszahlen mit einem Algorithmus. Der **Seed** ist der Startwert für diese Folge.

```
random.seed(42)
```

Reproduzierbarkeit

Gleicher Seed bedeutet gleiche Zufallsereignisse. Unterschiedliche Seeds erzeugen unterschiedliche Replikate.

```
scenario A, seed 1
scenario A, seed 2
scenario A, seed 3
```

Seeds helfen uns, Zufall kontrolliert zu verwenden: wir können Ergebnisse wiederholen und trotzdem Streuung messen.

Was ist Sensitivitätsanalyse?

Sensitivitätsanalyse fragt, wie stark Modelloutputs reagieren, wenn wir Parameterwerte verändern.

Parameter

Annahmen im Modell, z.B. Zufallsbremsen, Sicherheitsabstand oder Verkehrsdichte.

Output

Gemessene Ergebnisse, z.B. Durchschnittsgeschwindigkeit oder Anzahl langsamer Fahrzeuge.

Reaktion

Ändert sich fast nichts, etwas, oder kippt das System in ein anderes Verhalten?

Railsback & Grimm (2019), Kap. 23

Warum machen wir Parameter-Sweeps?

Verstehen

- Welche Annahmen kontrollieren das Ergebnis?
- Wo entstehen Kipppunkte?
- Welche Muster sind robust?

Vertrauen prüfen

- Hängt unser Ergebnis an einem einzigen Wert?
- Ist der Effekt größer als Zufallsschwankung?
- Welche Parameter müssen wir genauer begründen?

Hohe Sensitivität ist nicht automatisch schlecht. Sie zeigt, welcher Prozess im Modell besonders wichtig ist.

Eine einfache Strategie

1. Ein Modell vorläufig einfrieren: Code läuft, Grundverhalten ist plausibel.
2. Eine Messgröße wählen: z.B. durchschnittliche Geschwindigkeit.
3. Einen Parameter über sinnvolle Werte variieren.
4. Pro Wert mehrere Replikate mit unterschiedlichen Seeds ausführen.
5. Ergebnisse plotten und als Modellverhalten interpretieren.

> Für den Einstieg reicht oft eine Single-Parameter-Sensitivitätsanalyse: ein Parameter nach dem anderen.

Unsere Messgrößen

Im Traffic-Modell speichern wir bewusst einfache Größen, die man ohne Statistik-Vorwissen lesen kann.

Zeitreihe

- `time`: Simulationszeit
- `average_speed`: mittlere Geschwindigkeit

Stau-Indikatoren

- `stopped_cars`: stehende Fahrzeuge
- `slow_cars`: langsame Fahrzeuge

Eine Messgröße ist immer eine Entscheidung: Sie bestimmt, was wir am Modell als wichtig ansehen.

Was wir gleich implementieren

Beispiel-Sweep

```
uv run main.py sweep slowdown_prob 0.0 0.1 0.2 0.3 0.4 \
  --replicates 5 \
  --sim-duration 60 \
  --output runs/slowdown-sweep
```

Leitfragen

- Ab wann sehen wir regelmäßig Staus?
- Ist der Effekt größer als die Streuung zwischen Replikaten?
- Bleibt das Ergebnis bei anderen plausiblen Parametern ähnlich?

Theorie und Code treffen sich hier: Wir formulieren eine Erwartung, lassen das Modell laufen und lesen die Daten.

Auftrag für die Projektzeit

Setup prüfen

- GitHub-Repository existiert
- Alle Gruppenmitglieder haben Zugriff auf GitHub
- Alle haben das Projekt lokal geklont
- Alle können das Projekt mit `uv run` starten

Git einmal gemeinsam testen

- Jede Person macht eine kleine Änderung in `main.py` oder `README.md`
- Jede Person erstellt einen eigenen Commit und pusht ihn
- Schaut gemeinsam auf GitHub, was passiert
- Alle pullen die Änderungen der anderen

Ziel: Jede Person hat einmal den kompletten Zyklus gemacht: ändern, committen, pushen, auf GitHub prüfen, pullen.

Abschluss

Mitarbeitsaufgabe für Zuhause (optional, 3 Punkte)

GitHub-Recherche: Sucht ein interessantes GitHub-Repository eines größeren Softwareprojekts.

Mögliche Startpunkte: microsoft/vscode, videolan/vlc, obsproject/obs-studio, brave/brave-browser, telegramdesktop/tdesktop, mastodon/mastodon, scikit-learn/scikit-learn, facebook/react.

Kurzbericht (200-300 Wörter) mit diesen Punkten:

1. Welches Repository habt ihr gewählt? Link, Projektname und kurz: Was macht die Software?
2. In welchen Programmiersprachen ist das Projekt geschrieben?
3. Wie viele Stars hat das Repository ungefähr?
4. Wofür werden Issues in diesem Projekt allgemein genutzt?
5. Wofür werden Pull Requests allgemein genutzt?
6. Was ist ein Fork?