# Testen
# (insbesondere Fuzz Testing)

M. Anton Ertl
TU Wien

# Warum testen?

- Tut die Software, was sie soll?
- Testen kann nicht die Abwesenheit von Fehlern beweisen
- Aber immerhin die Anwesenheit
- Grundlage für Debugging
- Ist formales Beweisen nicht besser?
  benötigt eine formale Spezifikation
  hilft nicht gegen Fehler in der Spezifikation
  ist sehr aufwändig
- Testen *oder* statisches Typechecking?

# Grundlagen

- Testinputs

- erwartete Outputs
  oder Erwartungen an die Outputs

# Wie testen? Interaktiv?

- Wörter ausprobieren
  Inputs: Was einem einfällt
  Outputs werden mit den eigenen Erwartungen abgeglichen

- Teuer zu wiederholen (z.B. bei Änderung im Code)

- Nachvollziehbarkeit
  Eventuell mit Screen-Log

# Wie testen? Unit Tests?

```
t{ -10  7 /f -> -2 }t
t{  10 -7 /f -> -2 }t
t{ -10 -7 /f ->  1 }t
t{ :noname -21  4 /f ; execute -> -6 }t
t{ :noname -20  3 /f ; execute -> -7 }t
t{ :noname -20 -3 /f ; execute ->  6 }t
```

# Wie testen? Fuzz Testing?

- "Zufällige" Inputs
- Erwartungen an die Outputs
- Kosten pro Testfall viel geringer
- 36C3-Vorträge:

  Mike Sperber:
  "Getting software right with properties, generated tests, and proofs"

  Nspace, ganimo: "No source, no problem! High speed binary fuzzing"

# Fuzz Testing: Inputs

- Völlig zufällige Inputs sind nicht sinnvoll
  z.B. bei Speicheradressen

- Inputs sollten verschiedene Pfade ausführen
  Beobachtung der bedingten Sprünge etc. im getesteten Wort
  und passendes Variieren des Inputs
  Wie in Forth?

# Fuzz Testing: Erwartungen an outputs

- Oft reichen relativ grobe Überprüfungen
  Sperber prüft, ob die Mengenvereinigung assoziativ ist
  Nspace, ganimo prüfen, ob keine buffer overflows vorkommen

- Wie in Forth?

# Zusammenfassung

- Testen ist die dominierende QA-Massnahme
- Sollte systematisch gemacht werden
- Unit Tests
- Regression Tests
- Fuzz Testing