

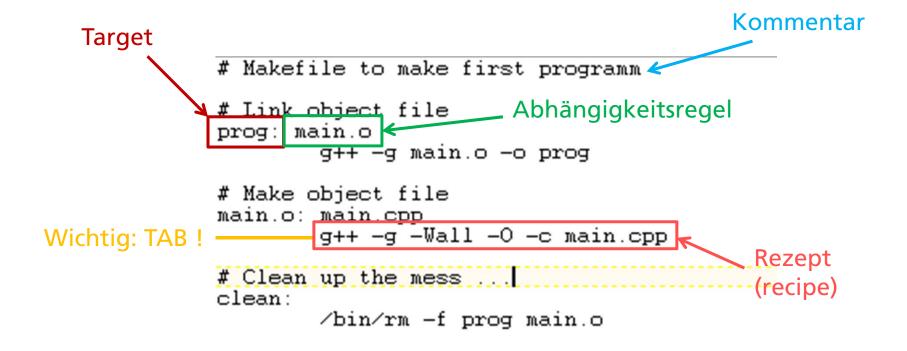
SOFTWARE ENGINEERING 2

BUILDMANAGEMENT

MAKE

- Make ist ein Werkzeug zum Automatisieren des Buildvorgangs
- Make stammt aus der UNIX-Welt Benutzung unter Windows z.B. mittels cygwin
- Abhängigkeiten werden abgebildet
- Nur benötigte Kommandos werden ausgeführt

MAKE – Erstes Beispiel



MAKE - Grundbegriffe

- Target wird erzeugt, indem
 - Abhängigkeiten geprüft werden
 - Gegebenenfalls die Anweisungen (Rezept) ausgeführt werden
- Sind die Abhängigkeiten aktuell, werden die Anweisungen nicht ausgeführt
- Existieren die Abhängigkeiten nicht, wird nach Regeln gesucht, wie diese gebaut werden können → Rekursiv
- Hat eine Regel keine Abhängigkeit, so wird sie immer ausgeführt ("Phony Targets").
- Bei Ausruf des Makefiles wird das angegebene Target gebaut. Ist keines angegeben, so wird das erste gebaut.
- Eine Regel muss nicht unbedingt eine Anweisung haben. Sie kann dazu verwendet werden, andere Regeln aufzurufen.

MAKE – Mehrere Dateien

```
# Makefile to make first programm
# Link object file
prog: main.o MyClass.o
                                       Abhängigkeit von
        g++ -g main.o -o prog
                                       Header
# Make object file
main.o: main.cpp
        g++ -g -Wall -0 -c main.cpp
# Make object file
MyClass.o: MyClass.cpp MyClass.h
        g++ -g -Wall -0 -c MyClass.cpp
# Clean up the mess ...
.PHONY: clean
clean:
        /bin/rm -f prog main.o MyClass.o
```

MAKE – Vereinfachungen (1)

- Kommandos und Flags als Variablen zusammenfassen
- Dateien in Variablen zusammenfassen

```
# Makefile to make first programm
  # Compiler call
  CC = q++
  # Linker
  LD = q++
 # Compiler flags
 CFLAGS = -q -Wall -0
  # Linker flags
  LDFLAGS =
 # Clean up command
  RM = /bin/rm - f
 # Object files
9 OBJS = main.o MyClass.o
  # Program executable
  PROG = prog
  # Link object file
  $(PROG): $(OBJS)
          $(LD) $(OBJS) -o $(PROG)
  # Make object file
  main.o: main.cpp
          $(CC) $(CFLAGS) -c main.cpp
  # Make object file
  MyClass.o: MyClass.cpp MyClass.h
          $(CC) $(CFLAGS) -c MyClass.cpp
  # Clean up the mess ...
  .PHONY: clean
  clean:
          $(RM) $(OBJS) $(PROG)
```

Generelle Regeln für Dateitypen

```
# Makefile to make first programm
   # Compiler call
   CC = g++
  # Linker
   LD = g++
   # Compiler flags
.0 CFLAGS = -g -Wall -0
  # Linker flags
  LDFLAGS =
  # Clean up command
.6 RM = /bin/rm - f
.8 # Object files
.9 OBJS = main.o MyClass.o
!1 # Program executable
2 PROG = prog
   # Link object file
   $(PROG): $(OBJS)
           $(LD) $(OBJS) -o $(PROG)
   # Make object files
   %.o: %.cpp
10
           $(CC) $(CFLAGS) -c $<
2 # Clean up the mess ...
   .PHONY: clean
4 clean:
15
           $(RM) $(OBJS) $(PROG)
```

%.cpp – Wildcard für alle CPP Dateien

%.o – Wildcard für alle Objekt-Dateien

\$< steht für die gefundene Abhängigkeit (hier: <Datei>.cpp)

Ähnlich:

\$@ Targetname \$* Ersetzung %

Automatisierung

- Einsatz von Skripten beim Einstellen ins KM
- Prüfung auf Compilierfähigkeit
- Automatisches Starten von
 - Tests
 - Dokumentation
- Buildserver
 - Automatisches Bauen eines kompletten SW-Standes
 - Zyklisch
 - Ereignisgetrieben
 - Dokumentation der "Build-Historie"
 - Bereitstellung der SW