Vorlesungsskript zu „Vertiefung Programmieren“ Build System

Dozent: Dipl.-Inf. (FH) Andreas Schmidt

DHBW
Duale Hochschule
Baden-Württemberg

Build System

Automatisierter Build-Prozess

Build System

Aufgaben des Build Systems

Zentrale Aufgabe des Build-Systems ist der automatisierte Ablauf des Build-Vorgangs

Hierzu gehören u.a.:

- Change Handling – Erkennen von Änderungen
- Dependency Management – Erkennen von Abhängigkeiten
- Target Handling – Verwalten von „Build Targets“
- Command Execution – Ausführen von definierten Befehlen

In modernen IDEs (Integrated Development Environment) sind Build-Systeme meist integriert.

- Proprietäres Build-System der IDE
- Nutzung von externen Build-Systemen

Quelle: intern

Build System

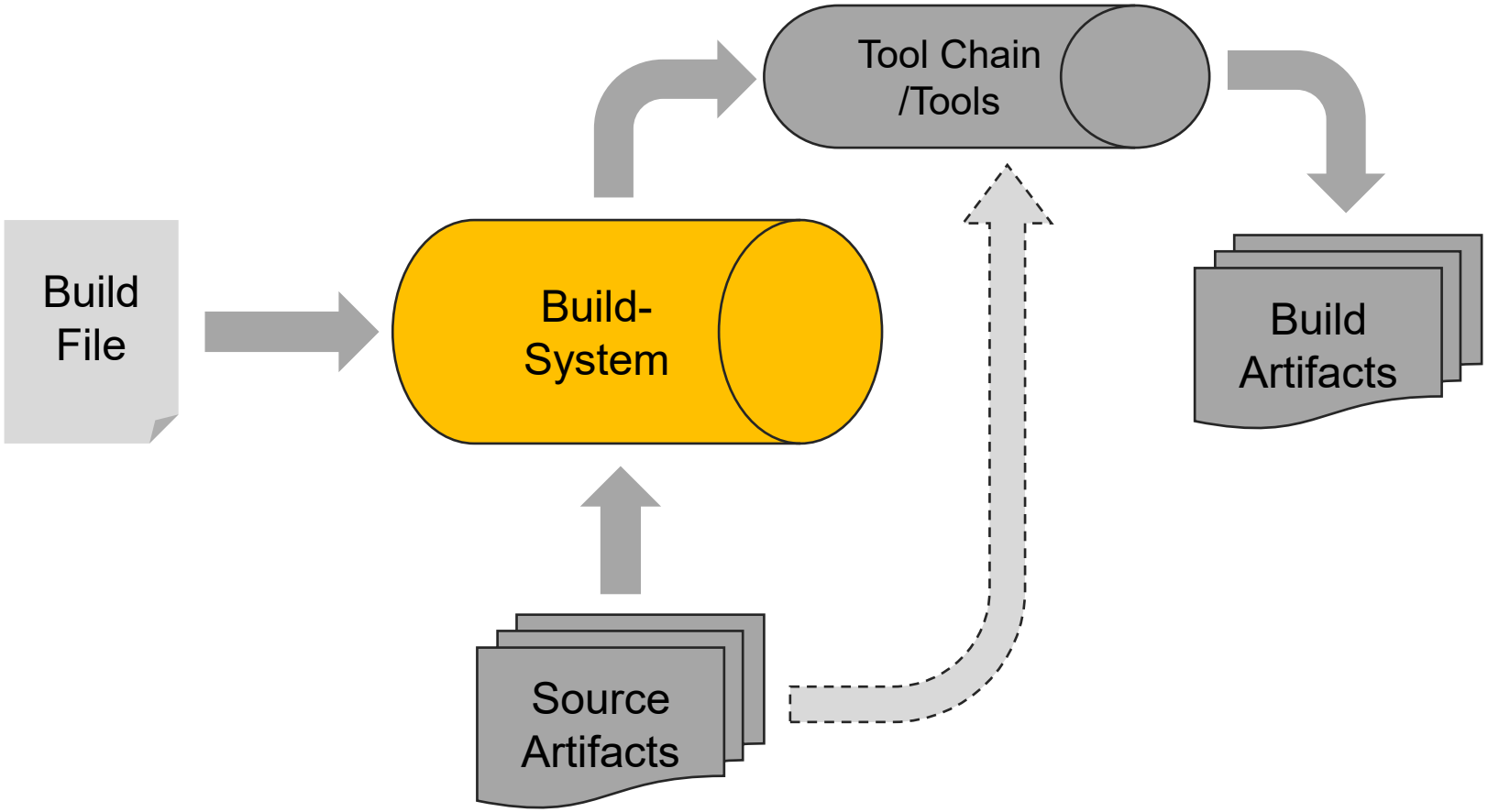
Bekannte Build-Systeme

Build System		
GNU Make	Text-Files mit Make Syntax	
SCons	Python Script basiertes Build-System Python Script Build-Files	
MS Build	Build System von Microsoft XML-basierte Build-Files	
Ant	Java basiertes Build-System speziell für Java XML-basierte Build-Files	
Maven	Java basiertes Build-System für Java, C# und Ruby XML-basierte Build-Files	
Gradle	Multi-language Build-System basierend auf Ant, Maven Domain-Specific Language Build-Files	

Quelle: intern

Build System

Grundsätzlicher Aufbau eines Build Systems



Quelle: intern

Build System

Grundlegender Aufbau eines Build-Files

Ein Build-File (meist unabhängig des Build-Systems) definiert in der Regel folgendes:

- **Target** – Wird auch Build-Target/Ziel genannt. Fasst Eingaben, Abhängigkeiten und Regeln zu einem Gesamtprozess zusammen. Häufig werden auch Abhängigkeiten für ein Target definiert

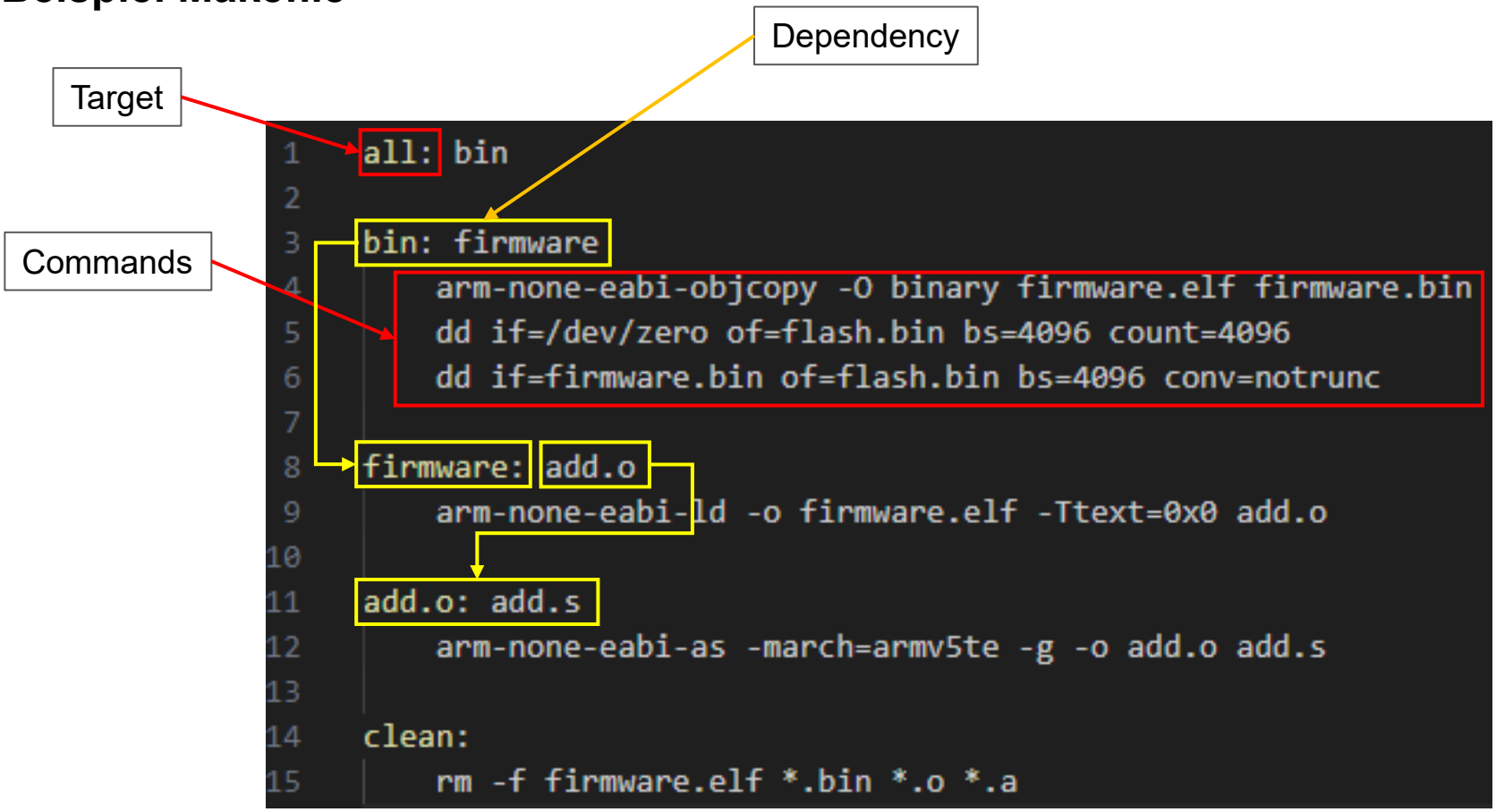
- **Rules** – Die Rules/Regeln definieren wie z.B. bestimmte Artefakte in andere Artefakte „umgewandelt“ werden (z.B. C-Source File in Object File)

- **Commands** – Befehle welche bei der Anwendung der Regeln und der Targets angewendet werden

Quelle: intern

Build System

Beispiel Makefile

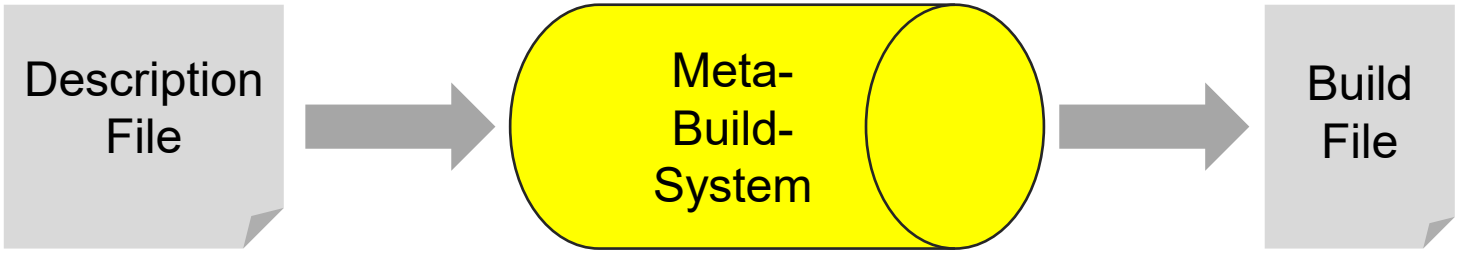


Quelle: intern

Build System

Sonderfall: Meta-Build-System

Ein Meta-Build-System (manchmal auch Build-Generator) ist kein Build-System im eigentlichen Sinne. Im Gegensatz zu einem Build-System, welches einen Übersetzungsprozess steuert, erzeugt ein Meta-Build-System aus einer Beschreibungsdatei ein Build-File für ein bestimmtes Build-System



Quelle: intern

Build System

Bekannte Meta-Build-Systeme

GNU Build Tools (Autoconf/Automake)

- Klassisches Meta-Build-System unter Linux

CMake – Sehr bekanntes Meta-Build-System (Cross-Platform)

- Kann aus einem sog. CMakeFile unterschiedliche Build-Files generieren. (z.B. Makefiles, Eclipse Projekt, Ninja, MSBuild etc.)

Meson – Meta-Build-System (Cross-Platform) mit einem einfachen Syntax

Quelle: intern

Build System

Vor- und Nachteile eines „manuellen“ Build-Systems

Vorteile

- Maximale Kontrolle über den Build-Prozess
- Jedes beliebige Kommandozeilentool kann in den Build-Prozess eingebunden werden
- Flexibler Aufbau über projekt-spezifische Verzeichnisebenen
- Wiederverwendung gut möglich (bei entsprechender Struktur)
- Bessere Integration in CI/CD System (z.B. Jenkins, Azure, etc.)

Nachteile:

- Meist komplexer Syntax der Build-Files (spezifische Wissen nötig)
- Kommandozeilen Tools müssen „manuell“ konfiguriert werden
- Fehlersuche bei Problemen im Build-Prozess nicht immer ganz einfach

Quelle: intern