Software-Produktlinien

Übung 5: Techniken für Variabilität zur Übersetzungszeit

Elias Kuiter, Gunter Saake Wintersemester 2025/26

1. Präprozessor-basierte Variabilität

Du hast eine Codebasis geerbt, in der die vorherigen Entwickler intensiv Präprozessor-Makros verwendet haben. Unten ist ein Ausschnitt aus dem Temperaturüberwachungsmodul.

C-Code mit Präprozessor-Direktiven

```c
#ifdef PRECISION_HIGH
    #define p_t double
    #define EPSILON 0.000001
    #ifdef UNIT_KELVIN
        #define CONVERT(x) ((x) * 0.1 + 273.15)
    #else
        #define CONVERT(x) ((x) * 0.1)
    #endif
#else
    #define p_t float
    #define EPSILON 0.001f
    #define CONVERT(x) (x)
#endif

// Thermal mode and sensor macros
#ifdef MODE_THERMAL
    #define SENSOR_COUNT 3
    #ifdef SENSOR_ADVANCED
        #define READ_RAW(i) (300 + (i) * 10)
    #else
        #define READ_RAW(i) (200 + (i) * 8)
    #endif
#endif

void readSensor() {
#ifdef MODE_THERMAL
    for (int i = 0; i < SENSOR_COUNT; ++i) {
        p_t value = CONVERT(READ_RAW(i));
        if (value > EPSILON) printf("Sensor[%d] active: %.2f\n", i, value);
    }
#else
    printf("Thermal mode disabled.\n");
#endif
}
```

(a) Was ist ein Präprozessor und wie unterstützt er die Implementierung von Variabilität in Software-Produktlinien?

(b) Erkläre die Rolle des Präprozessors in diesem Code.

(c) Angenommen, `PRECISION_HIGH`, `UNIT_KELVIN`, `MODE_THERMAL` und `SENSOR_ADVANCED` sind definiert, während `EPSILON`, `SENSOR_COUNT`, `CONVERT` und `p_t` außerhalb des gegebenen Code-Snippets nicht vordefiniert sind. Gib den resultierenden Quellcode an, wie er nach dem Preprocessing erscheinen würde, mit allen expandierten Makros und aufgelöster bedingter Kompilierung.

(d) Hier wurde der C-Präprozessor (CPP) verwendet, aber nicht alle Präprozessoren sind in allen Kontexten gleichermaßen geeignet. Vergleiche die drei Präprozessoren `CPP`, `Munge` und `Antenna`. Für welche Arten von Projekten wäre jeder am besten geeignet?

2. **Variabilität im Linux-Kernel**

(a) Erkläre die Rolle von KConfig, KBuild, CPP und MenuConfig. Wie unterscheiden sie sich und was ist jeweils ihr Zweck?

(b) Der Linux-Kernel verwendet eine Kombination von Tools zur Verwaltung von Variabilität. Unten ist ein vereinfachtes KBuild-Snippet:

KBuild-Snippet aus Kernel-Makefile

```
obj-y += core/
obj-m += net/
obj-$(CONFIG_USB_SUPPORT) += drivers/usb/
obj-$(CONFIG_DEBUG_MODE) += debug/
```

i. Erkläre, was jede Zeile im Makefile-Snippet macht. Wie steuern `obj-y`, `obj-m` und `obj-$(...)` die Einbindung von Features im Build?

ii. Gegeben seien die folgenden Konfigurationsoptionen:

- `CONFIG_DEBUG_MODE=y`

Welche Verzeichnisse oder Dateien werden statisch kompiliert, als Module behandelt oder vom Build ausgeschlossen?

iii. Was ist der Unterschied zwischen der Implementierung von Features mit Build-Systemen (wie KBuild) und Clone-and-Own mit Build-Systemen (wie in einer früheren Übung besprochen)?

3. **Feature Traceability**

Du arbeitest an einem Smart-Grid-Controller, der sein Verhalten abhängig von Feature-Flags für Präzision, Sensortyp und Logging anpasst. Unten ist ein vereinfachtes und konfigurierbares Code-Snippet mit bedingter Kompilierung.

Java-Code mit Präprozessor-Direktiven

```
public class GridController {

  //#if PRECISION_HIGH
  double reading;
  //#elif PRECISION_LOW
  float reading;
  //#else
  int reading;
  //#endif

  public void computePower() {

    //#if SENSOR_ADVANCED
    reading = getAdvancedSensorValue();
    //#if APPLY_SCALING
    reading = scale(reading, 1.15);
    //#endif
    //#elif SENSOR_BASIC
    reading = getBasicSensorValue();
    //#endif

    //#if DEBUG
    System.out.println("[DEBUG] Raw reading: " + reading);
    //#endif

    //#if APPLY_LOGGING
    logToFile(reading); // logs the value, may be scaled or unscaled
    //#endif
  }

  private double getAdvancedSensorValue() { return 123.45; }
  private float getBasicSensorValue() { return 78.9f; }
  private double scale(double val, double factor) { return val * factor; }
  private void logToFile(double value) { /* log to disk */ }
}
```

(a) Was ist Feature Traceability (Rückverfolgbarkeit)? Erkläre Herausforderungen im Zusammenhang mit Feature Traceability (Tangling, Scattering, und Replikation). Inwieweit treffen diese auf dieses Code-Beispiel zu?

(b) Welche Konsequenzen haben Tangling, Scattering und Replikation?

(c) Welche Möglichkeiten gibt es, um Feature Traceability sicherzustellen? Diskutiere Vor- und Nachteile.

4. **Vergleich von Variabilitäts-Implementierungstechniken**

Vergleiche die Vor- und Nachteile sowie Anwendungsfälle von Features mit Präprozessoren und Features mit Build-Systemen. Berücksichtige auch die bisher in der Vorlesung besprochenen Techniken in deinem Vergleich. Verwende Beispiele zur Unterstützung deiner Argumente.