Software Product Lines

Exercise 5: Techniques for Compile-Time Features

ISF (TU Braunschweig) January 2026

1. **Preprocessor-Based Variability**

   You have inherited a codebase, in which the previous developers heavily used preprocessor macros. Below is a snippet from the thermal monitoring module.

   C Code Using Preprocessor Directives

```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) What is a preprocessor and how does it support implementing variability in software product lines?

   (b) Explain the role of the preprocessor in this code.

   (c) Assuming `PRECISION_HIGH`, `UNIT_KELVIN`, `MODE_THERMAL`, and `SENSOR_ADVANCED` are defined, while `EPSILON`, `SENSOR_COUNT`, `CONVERT`, and `p_t` are not pre-defined outside the given code snippet. Show the resulting source code as it would appear after preprocessing, with all macros expanded and conditional compilation resolved.

(d) Here we used the C preprocessor (CPP), but not all preprocessors are equally suitable in all contexts. Compare the three preprocessors `CPP`, `Munge`, and `Antenna`. In which types of projects would each be most appropriate?

2. **Variability in the Linux Kernel**

   (a) Explain the role of KConfig, KBuild, CPP, and MenuConfig. How do they differ, and what is the purpose of each?

   (b) The Linux kernel uses a combination of tools to manage variability. Below is a simplified KBuild snippet:

   KBuild Snippet from Kernel Makefile

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

      i. Explain what each line in the Makefile snippet does. How do `obj-y`, `obj-m`, and `obj-$(...)` control the inclusion of features in the build?

      ii. Given the following configuration options:

      - `CONFIG_DEBUG_MODE=y`

      Which directories or files are compiled statically, treated as modules, or excluded from the build?

      iii. What is the difference between the implementation of features with build systems (like KBuild) and clone-and-own with build systems (as discussed in an earlier exercise)?

3. **Feature Traceability**

   You are working on a Smart Grid Controller that adjusts its behavior depending on feature flags related to precision, sensor type, and logging. Below is a simplified and configurable code snippet using conditional compilation.

Java Code with Preprocessor Directives

```
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) What is feature traceability? Explain challenges related to feature traceability (i.e., code tangling, scattering, and replication). To which degree do these apply to this code snippet?

(b) What are the consequences of code tangling, scattering and replication?

(c) By which means can feature traceability be ensured? Discuss their benefits and drawbacks!

4. **Comparison of Variability Implementation Techniques**

Compare the advantages, disadvantages, and use cases of features with preprocessors and features with build systems. Also include the techniques discussed in the lecture so far in your comparison. Use examples to support your arguments.