
Software Product Lines

Exercise 9: Feature Interactions

ISF (TU Braunschweig)

January 2026

1. Feature Interactions: Fundamentals

- (a) Explain the term *feature interaction* based on an example.
- (b) Describe the meaning of a *higher-order* feature interaction. Give an example of such an interaction.
- (c) How many t -wise feature interactions are possible in a product line consisting of n optional, unconstrained features? State formulas to compute this number for any t and $t \in \{0, 1, 2, n\}$, specifically. How many feature interactions (across all t) are possible in total?

2. Consequences of Feature Interactions

- (a) Which (positive and negative) consequences can feature interactions have for software-product-line engineering?
- (b) How can the number of expected feature interactions be reduced?
- (c) After this reduction, how can the remaining feature interactions be detected?

3. Variability-Bug Analysis

Explore the *Variability Bug Database (VBDb)*¹ and select one real-world interaction fault (under “Database”).

- (a) Explain the issue in your own words. Which features or options were involved?
- (b) Classify the type of feature interaction (wanted/unwanted, static/dynamic). Is it only caused by the presence or absence of features, or both? Of which order t is it?
- (c) Analyze the bugfix provided in the VBDb. Does it apply one of the six strategies for handling feature interactions discussed in the lecture? If so, which one? Would you fix the issue differently? If so which strategy would you apply?

4. Strategies for Resolving Feature Interactions

Different strategies exist to handle feature interactions in software product lines, each with its own trade-offs. Compare any two strategies from the lecture by discussing their benefits, limitations, and appropriate scenarios for use. Illustrate your comparison using a concrete example.

5. Detecting Variability Bugs in Configurable Code

Consider the following configurable Java code snippet. The system supports the following optional features (defined via preprocessor macros): `SECURE_LOGIN`, `CACHING`, and `LOGGING`.

¹<https://vbdb.bitbucket.io/>

A configurable login handler

```
public class AccessHandler {
    UserSession session;

    public void handleLogin(String user, String password) {
        #ifdef SECURE_LOGIN
        if (!authenticate(user, password)) {
            throw new SecurityException("Login failed");
        }
        #endif

        #ifdef CACHING
        session = Cache.get(user);
        if (session == null) {
            session = new UserSession(user);
            Cache.put(user, session);
        }
        #endif

        #ifdef LOGGING
        Logger.log("User " + session.getUser() + " logged in at " + System.
            currentTimeMillis());
        #endif
    }

    #ifdef SECURE_LOGIN
    private boolean authenticate(String user, String password) {
        return user.equals("admin") && password.equals("1234");
    }
    #endif
}
```

- (a) Identify configurations in which the above code does not compile, as well as configurations in which the code compiles but fails at runtime. What causes these issues?
- (b) Are these static or dynamic feature interactions? Of which order t are they? Are these “real” feature interactions (tied to the inherent complexity of the interacting features)?
- (c) Would testing with the default configuration (SECURE_LOGIN, CACHING, and LOGGING) have caught these issues? Suppose you did not identify these interactions yet – which other configurations would you also test to increase the chances of detecting bugs?