Variability Analyses, Formal Methods and Testing. Putting all together

David Benavides

Department of Computer Languages and Systems
University of Sevilla
Formal methods

Tool support
Challenge 1: Automated Analysis of FM
Challenge 2: Explanations on FM analysis
Challenge 3: Testing on FM analysis tools
Introduction

Feature models
Challenge 1: Automated Analysis of FM

Ch 1.1 with attributes
Ch 1.2 with configuration paths

Challenge 2: Explanations on FM analysis

Ch 2.1 with feature models
Ch 2.2 with configurations

Challenge 3: Testing on FM analysis tools

Ch 3.1 Functional Testing
Ch 3.2 Performance Testing

Formal methods

Tool support
Challenge 1: Automated analysis of Feature Models

Computer-aided, extraction of useful information from feature models
## Mapping to Propositional Logic

<table>
<thead>
<tr>
<th>Relationship</th>
<th>PL Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MANDATORY</strong></td>
<td>P ↔ C</td>
</tr>
<tr>
<td><strong>OPTIONAL</strong></td>
<td>C → P</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>P ↔ (∧ C₁ ∨ C₂ ∨ ... ∨ Cₙ)</td>
</tr>
<tr>
<td><strong>ALTERNATIVE</strong></td>
<td>(C₁ ↔ (∧ ¬C₂ ∧ ... ∧ ¬Cₙ ∧ P)) ∧ (C₂ ↔ (∧ ¬C₁ ∧ ... ∧ ¬Cₙ ∧ P)) ∧ (Cₙ ↔ (∧ ¬C₁ ∧ ¬C₂ ∧ ... ∧ ¬Cₙ₋₁ ∧ P))</td>
</tr>
<tr>
<td><strong>IMPLIES</strong></td>
<td>A → B</td>
</tr>
<tr>
<td><strong>EXCLUDES</strong></td>
<td>¬(A ∧ B)</td>
</tr>
</tbody>
</table>
Automated analysis of feature models:
Computer-aided extraction of information from FMs
Automated analysis of feature models: Computer-aided extraction of information from FMs

Any error?

Yes, feature "Basic" is dead
Automated analysis of SPL

Why it’s an important challenge?

Doing this by hand is an error prone task in large-scale feature models.

Detecting properties at early stage of development and along all the life cycle.

It’s the base for other more complicated tasks, i.e. product configuration.
Known real scenarios

- Cloud infrastructures configurations for green computing
- Mobile phone platform + applications evolution
- Linux and Debian-based configuration and debugging
- Firewall configurations
- Smart homes deployment

Automated analysis of variability models
Challenge 1: Automated analysis of SPL: Computer-aided, extraction of useful information from SPL models

Automated analysis of feature models 20 years later: A literature review

David Benavides *, Sergio Segura, Antonio Ruiz-Cortès

Dept. de Lenguajes y Sistemas Informáticos, University of Seville, Av. Reina Mercedes s/n, 41012 Seville, Spain

Keywords: Feature models, Automated analyses, Software product lines, Literature review

ABSTRACT

Software product line engineering is about producing a set of related products that share more commonalities than variabilities. Feature models are widely used for variability and commonality management in software product lines. Feature models are information models where a set of products are represented as a set of features in a single model. The automated analysis of feature models deals with the computer-aided extraction of information from feature models. The literature on this topic has contributed with a set of operations, techniques, tools and empirical results which have not been surveyed until now. This paper provides a comprehensive literature review on the automated analysis of feature models 20 years after their invention. This paper contributes by bringing together previously disparate streams of work to help shed light on this challenging and highly relevant research framework.
Challenge 1: Automated analysis of SPL: Computer-aided, extraction of useful information from SPL models

30 analysis operations found

CSP, SAT, BDD, DL, ad-hoc

Performance testing

Formal methods
Challenge 1: Automated Analysis of FM

Challenge 2: Explanations on FM analysis

Challenge 3: Testing on FM analysis tools

Formal methods

Tool support

Ch 1.1 with attributes
Ch 1.2 with configuration paths

Ch 2.1 with feature models
Ch 2.2 with configurations

Ch 3.1 Functional Testing
Ch 3.2 Performance Testing
Challenge 1.1: Automated Analysis of Feature Models with attributes

- David Benavides, Pablo Trinidad Martín-Arroyo, Antonio Ruiz Cortés: Automated Reasoning on Feature Models. CAiSE 2005: 491-503
- F Roos-Frantz, D Benavides, A Ruiz-Cortés, A Heuer, K Lauenroth Quality-aware analysis in product line engineering with the orthogonal variability model. Software Quality Journal
Challenge 1.2: Automated Analysis of Feature Models Configuration Paths

Challenge 1: Automated Analysis of FM

Challenge 2: Explanations on FM analysis

Challenge 3: Testing on FM analysis tools

Tool support
Challenge 2: Explanations on the Automated analysis of SPL

Ongoing PhD thesis
Challenge 2: Explanations on the Automated analysis of SPL

Deductive Reasoning

Is valid?

No
Challenge 2: Explanations on the Automated analysis of SPL

Abductive Reasoning

- Facts
- Explanation
  - Hypotheses
- Observation
- Abduction
Automated error analysis for the agilization of feature modeling


Departamento de Lenguajes y Sistemas Informáticos, University of Seville, Av. Reina Mercedes s/n, 41012 Seville, Spain
Received 27 March 2007; received in revised form 19 July 2007; accepted 10 October 2007
Available online 21 November 2007

Abstract

Software Product Lines (SPL) and agile methods share the common goal of rapidly developing high-quality software. Although they follow different approaches to achieve it, some synergies can be found between them by (i) applying agile techniques to SPL activities so SPL development becomes more agile; and (ii) tailoring agile methodologies to support the development of SPL. Both options require an intensive use of feature models, which are usually strongly affected by changes on requirements. Changing large-scale feature models as a consequence of changes on requirements is a well-known error-prone activity. Since one of the objectives of agile methods is a rapid response to changes in requirements, it is essential an automated error analysis support in order to make SPL development more agile and to produce error-free feature models.

As a contribution to find the intended synergies, this article sets the basis to provide an automated support to feature model error analysis by means of a framework which is organized in three levels: a feature model level, where the problem of error treatment is described; a diagnosis level, where an abstract solution that relies on Reiter’s theory of diagnosis is proposed; and an implementation level, where the abstract solution is implemented by using Constraint Satisfaction Problems (CSP).

To show an application of our proposal, a real case study is presented where the Feature-Driven Development (FDD) methodology is applied to a large SPL.
Ch 3.1 with feature models

Feature Model Level
- Feature Model

Diagnosis Level
- \( \text{COMPS} = \{ R_i \} \)
- \( \text{SD} = \{ ... \} \)
- \( \text{OBS} = \{ ... \} \)
- Diagnosis model

Implementation Level
- Variables:
  - \( F_j, A_{b_i} \)
- Constraints:
  - \( A_{b_i} = 1 \rightarrow ... \)
- Constraints model

Errors
- dead features, ...

Explanations
- \( \{ R_e \} \)

Diagnosis
- \( \Delta = \{ R_e \} \)

Solutions
- \( \{ F_j \rightarrow v_j, A_{b_i} \rightarrow v_i \} \)
- ...
1. Invalid Configuration:

   O_1 = 1
   - O_2 = 1
     - Brake Control Software
     - O_3 = 0
       - ABS Controller
     - O_4 = 1
       - Non-ABS Controller
   - O_5 = 1
     - Brake ECU
     - O_6 = 1
       - 1 Mbit/s CAN Bus
     - O_7 = 0
       - 250kbit/s CAN Bus

   \[ f_1 = 1 \rightarrow (f_2 = 1) \]

2. Diagnostic CSP:

   \[ f_1 = 1 \rightarrow (f_2 = 1) \]

3. Recommendations:

   - Deselect 1 Mbit/s CAN Bus, \( d_6 = 1 \)
   - Select 250kbit/s CAN Bus, \( s_7 = 1 \)

4. Valid Configuration:

   \[ f_1 = 1 \]
   - O_1 = 1
     - O_2 = 1
       - Brake Control Software
       - O_3 = 0
         - ABS Controller
       - O_4 = 1
         - Non-ABS Controller
     - O_5 = 1
       - Brake ECU
       - O_6 = 1
         - 1 Mbit/s CAN Bus
       - O_7 = 0
         - 250kbit/s CAN Bus

   \[ f_2 = 1 \]
   - O_1 = 1
     - O_2 = 1
       - Brake Control Software
       - O_3 = 0
         - ABS Controller
       - O_4 = 1
         - Non-ABS Controller
     - O_5 = 1
       - Brake ECU
       - O_6 = 1
         - 1 Mbit/s CAN Bus
       - O_7 = 0
         - 250kbit/s CAN Bus

   \[ f_5 = 1 \]
Challenge 1: Automated Analysis of FM

Challenge 2: Explanations on FM analysis

Challenge 3: Testing on FM analysis tools

Ch 1.1 with attributes
Ch 1.2 with configuration paths
Ch 2.1 with feature models
Ch 2.2 with configurations

Ch 3.1 Functional Testing
Ch 3.2 Performance Testing
Challenge 3.1: Functional Testing
How to detect faults in feature model analysis tools?

Feature Model Analysis Tool

- Translation
- Solver

• SAT, CSP, DL, OWL...
• Large programs.

Time-consuming
Error-prone
Challenge 3.1: Functional Testing

Operation

How are the products obtained? Is there any redundant feature?

Expected output

Yes, Feature C is obtained in 6 products. Version 2023.
## Challenge 3.1: Functional Testing

<table>
<thead>
<tr>
<th>FM</th>
<th>FM′</th>
<th>Metamorphic relation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Mandatory Diagram" /></td>
<td><img src="image2" alt="Mandatory Diagram" /></td>
<td># products(FM′) = # products(FM) \land \forall P′(P′ \in \text{products}(FM′) \iff \exists P \in \text{products}(FM) \cdot (pf \in \text{features}(P) \land P′ = P \cup {f}) \lor (pf \notin \text{features}(P) \land P′ = P))</td>
</tr>
<tr>
<td><img src="image3" alt="Optional Diagram" /></td>
<td><img src="image4" alt="Optional Diagram" /></td>
<td># products(FM′) = # products(FM) + # \text{filter}(FM,{pf},\phi) \land \forall P′(P′ \in \text{products}(FM′) \iff \exists P \in \text{products}(FM) \cdot P′ = P \lor (pf \in \text{features}(P) \land P′ = P \cup {f}))</td>
</tr>
<tr>
<td><img src="image5" alt="Alternative Diagram" /></td>
<td><img src="image6" alt="Alternative Diagram" /></td>
<td># products(FM′) = # products(FM) + (#C - 1)# \text{filter}(FM,{pf},\phi) \land \forall P′(P′ \in \text{products}(FM′) \iff \exists P \in \text{products}(FM) \cdot (pf \in \text{features}(P) \land \exists C \in C \cdot P′ = P \cup {C}) \lor (pf \notin \text{features}(P) \land P′ = P))</td>
</tr>
<tr>
<td><img src="image7" alt="OR Diagram" /></td>
<td><img src="image8" alt="OR Diagram" /></td>
<td># products(FM′) = # products(FM) + (2^{</td>
</tr>
<tr>
<td><img src="image9" alt="Requires Diagram" /></td>
<td><img src="image10" alt="Requires Diagram" /></td>
<td>\text{products}(FM′) = \text{products}(FM) \setminus \text{filter}(FM,{f},{g})</td>
</tr>
<tr>
<td><img src="image11" alt="Excludes Diagram" /></td>
<td><img src="image12" alt="Excludes Diagram" /></td>
<td>\text{products}(FM′) = \text{products}(FM) \setminus \text{filter}(FM,{f,g},\phi)</td>
</tr>
</tbody>
</table>
Challenge 3.2: Performance Testing

How to know the performance of FM analysis tools in pessimistic cases?
Challenge 3.2: Performance Testing

The diagram illustrates the process of evaluating an initial population through various genetic algorithm operations: Evaluation, Decoding, Encoding, Selection, Crossover, and Mutation. Fitness values are calculated (execution time, memory consumption). The cycle continues until the hardest feature model is found.
Challenge 3.2: Performance Testing

Encoding - Crossover - Mutation
Challenge 3.2: Performance Testing

Execution time in a CSP-based reasoner:
- 6.7 minutes
- 4.2 minutes (x2)

Memory consumption in a BDD-based reasoner:
- 25.3x10^6 nodes
- 27.9x10^6 nodes

0.2 seconds

<table>
<thead>
<tr>
<th>Evolutionary search</th>
<th>Random search</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 30 minutes</td>
<td></td>
</tr>
</tbody>
</table>
Challenge 3: Testing on FM analysis tools

Ch 3.1 Functional testing
Ch 3.2 Performance testing

Automated metamorphic testing on the analyses of feature models
Sergio Segura, Robert M. Hierons, David Benavides, Antonio Ruiz-Cortés

ABSTRACT

Context: A feature model (FM) represents the valid combinations of features in a domain. The automated extraction of information from FMs is a complex task that involves numerous analysis operations, techniques, and tools. Current testing methods in this context are manual and rely on the ability of the tester to decide whether the output of an analysis is correct. However, this is acknowledged to be time-consuming, error-prone, and in most cases infeasible due to the combinatorial complexity of the analyses, this is known as the oracle problem.

Objective: In this paper, we propose using metamorphic testing to automate the generation of test data for feature model analysis tools. Our approach consists of extending the oracle problem by providing a set of correct transformations that can be applied to a feature model to generate new test cases.

Methods: We present a set of metamorphic transformations (feature model transformations) that can be applied to a feature model to generate new test cases. These transformations are generated based on the valid combinations of features in the feature model. The goal is to generate test cases that are different from the original test cases, yet still valid according to the feature model. By doing so, we can automate the generation of test data for feature model analysis tools, overcoming the oracle problem.

Results: Our approach is evaluated by applying it to a real-world feature model. The results show that the proposed transformations are effective in generating new test cases that are different from the original test cases, yet still valid according to the feature model. The proposed approach is also computationally efficient, as it can be evaluated in a few seconds.

ETHOM: An Evolutionary Algorithm for Optimized Feature Models Generation

TECHNICAL REPORT IS-A-2011-TR-03 (v. 1.9)

S. Segura and J. A. Parejo Software Engineering School
University of Seville
parejo@ing.uniovi.es

R. M. Hierons
School of Information Systems, Computing and Mathematics
Brunel University
Rob.Hierons@brunel.ac.uk

D. Benavides, and A. Ruiz-Cortés
Software Engineering School
University of Seville
[benavides, aru@c]@us.es

Applied Software Engineering Research Group
http://www.isa.us.es
Challenge 1: Automated Analysis of FM

Challenge 2: Explanations on FM analysis

Challenge 3: Testing on FM analysis tools

Ch 1.1 with attributes

Ch 1.2 with configuration paths

Ch 2.1 with feature models

Ch 2.2 with configurations

Ch 3.1 Functional Testing

Ch 3.2 Performance Testing

Tool support
Tooling

Tools
- AHEAD
- pure::variants
- FaMa FW

Techniques
- SAT
- BDD
- Description Logic
- Constraint Programming
- OWL

Operations
- Valid Product
- Void
- Products
- Number of Products
- Dead Features
- Variability
- Explanations
- Corrective Explanations
- Dead Features
- Optimization

30 operations
Tooling the Automated analysis of SPL

www.isa.us.es/fama
Tooling the Testing of FM analysis tools

Benchmarking and Testing on the analysis of feature models

www.isa.us.es/betty

- Metamorphic test data generation
- Evolutionary FM generation
- Random FM generation
- Benchmarking support
Challenge 1: Automated Analysis of FM

Challenge 2: Explanations on FM analysis

Challenge 3: Testing on FM analysis tools

Ch 1.1 with attributes
Ch 1.2 with configuration paths
Ch 2.1 with feature models
Ch 2.2 with configurations
Ch 3.1 Functional Testing
Ch 3.2 Performance Testing

Tool support
How to specify analysis operations for variability models?

Using formal methods

How to gain confidence about the specification correctness?

Using metamorphic testing techniques
FLAME
(Fama formal framework)

Abstract foundation

Characteristic Model
## Basic concepts in FLAME

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Redefinition in CML</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL</td>
<td>Type for SPLs</td>
<td>optional</td>
</tr>
<tr>
<td>Feature</td>
<td>Type for features</td>
<td>optional</td>
</tr>
<tr>
<td>Model</td>
<td>Type for characteristic models</td>
<td>mandatory</td>
</tr>
<tr>
<td>_ _ _</td>
<td><em>is–instance–of</em> relation</td>
<td>mandatory</td>
</tr>
<tr>
<td>( \Phi )</td>
<td><em>features–in–a–model</em> function</td>
<td>mandatory</td>
</tr>
</tbody>
</table>
FLAME- Abstract foundation layer

[Feature]
[Model]

Product $== \mathbb{F}_1 \text{Feature}$

\[
\begin{align*}
\text{SPL} \\
\text{model} : \text{Model} \\
\text{features} : \mathbb{F}_1 \text{Feature} \\
\Phi \text{ model} = \text{features}
\end{align*}
\]

$\_ \leftrightarrow \_ : \text{Product} \leftrightarrow \text{Model}$

$\forall p : \text{Product}; \ m : \text{Model} \Rightarrow$

\[
p \leftrightarrow m \iff [p \text{ is an instance of } m]
\]

[concrete definition must be provided in the CML]
FLAME - Abstract foundation layer
operations

\[
\begin{align*}
_\prec_ : \text{Product} & \leftrightarrow \text{SPL} \\
\forall p : \text{Product}; \ spl : \text{SPL} : & \ \\
& ( p \prec spl \iff ( p \subseteq spl.\text{features} \land p \not\subseteq spl.\text{model} ))
\end{align*}
\]

\( P \) is\ValidFor\( SPL_1 \)
FLAME - Abstract foundation layer operations

\[
\begin{align*}
_\prec_ & : \text{Product} \leftrightarrow \text{SPL} \\
\forall p : \text{Product}; \ spl : \text{SPL} & . \\
\quad p < spl & \Leftrightarrow \left( p \subseteq \text{spl.features} \land p \prec \text{spl.model} \right)
\end{align*}
\]
FLAME - Abstract foundation layer operations

\[ \Pi : \text{SPL} \rightarrow \mathbb{F} \text{Product} \]

\[ \forall \text{spl} : \text{SPL} \bullet \]

\[ \Pi \text{spl} = \{ p : \mathbb{F} \text{spl.features} \mid p \prec \text{spl} \} \]

**PROLOG**

```
products( spl( F, M ), PRDS ) :-
    findall( P, valid( P, spl( F, M ) ), PRDS ).

valid( P, spl( F, M ) ) :-
    btrck_subst( P, F ), % backtrackable version of subset
    instance_of( P, M ). % abstract predicate
```
FLAME - Abstract foundation layer operations - map

- `isInstanceOf`
- `isValidFor`
- `isVoid`
- `Products`
  - `validConfig`
  - `refactoring`
  - `generalization`
  - `#products`
  - `filter`
  - `deadFeatures`
  - `.........`
FLAME
(Fama formal Framework)

Abstract foundation

Characteristic Model
FLAME

Characteristic model layer

Feature Model abstract syntax (Model type)

\[
\text{Model} ::= \text{BFM} \ll \text{TreeFeature} \times \mathbb{F} \text{ CTC} \rr
\]

\[
\text{TreeFeature} ::= \text{feature}_k \ll \text{Feature} \times \mathbb{F}_1 \text{ Relationship} \rr \\
| \quad \text{feature}_\lambda \ll \text{Feature} \rr
\]

\[
\text{Relationship} ::= \text{mandatory} \ll \text{TreeFeature} \rr \\
| \quad \text{optional} \ll \text{TreeFeature} \rr \\
| \quad \text{one_or_more} \ll \mathbb{F}_2 \text{ TreeFeature} \rr \\
| \quad \text{only_one} \ll \mathbb{F}_2 \text{ TreeFeature} \rr
\]

\[
\text{CTC} ::= \text{requires} \ll \text{Feature} \times \text{Feature} \rr \\
| \quad \text{excludes} \ll \text{Feature} \times \text{Feature} \rr
\]
Feature model semantics (instanceOf relation)

Redefinition of \( \leftrightarrow \) for BFM

\[ \forall p : \text{Product}; \; m : \text{Model} \bullet p \leftrightarrow m \iff ( p \leftrightarrow_{\tau} \text{tree} \; m \land \forall ctc_i : ctc \; m \bullet p \leftrightarrow_{\chi} ctc_i ) \]

Instance of a BFM tree

\[ \forall p : \text{Product}; \; f : \text{Feature}; \; r : \mathbb{F} \; \text{Relationship} \bullet p \leftrightarrow_{\tau} \text{feature}_{\chi} ( f ) \iff f \in p \land p \leftrightarrow_{\tau} \text{feature}_{\kappa} ( f, r ) \iff ( f \in p \land \forall r_i : r \bullet p \leftrightarrow_{R} r_i ) \]

Instance of a BFM relationship

\[ \forall p : \text{Product}; \; f : \text{Feature}; \; t_i : \text{TreeFeature}; \; t : \mathbb{F}_2 \; \text{TreeFeature} \bullet p \leftrightarrow_{R} \text{mandatory} ( t_i ) \iff p \leftrightarrow_{\tau} t_i \land p \leftrightarrow_{R} \text{optional} ( t_i ) \iff ( p \leftrightarrow_{\tau} t_i \lor p \cap \Phi_{\tau} t_i = \emptyset ) \land p \leftrightarrow_{R} \text{one_or_more} ( t ) \iff \forall t_j : t \bullet p \leftrightarrow_{R} \text{optional} ( t_j ) \land \exists t_k : t \bullet p \leftrightarrow_{\tau} t_k ) \land p \leftrightarrow_{R} \text{only_one} ( m ) \iff \forall t_j : t \bullet p \leftrightarrow_{R} \text{optional} ( t_j ) \land \exists_1 t_k : t \bullet p \leftrightarrow_{\tau} t_k ) \]
### Basic concepts in FLAME

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Redefinition in CML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPL</strong></td>
<td>Type for SPLs</td>
<td>optional</td>
</tr>
<tr>
<td><strong>Feature</strong></td>
<td>Type for features</td>
<td>optional</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>Type for characteristic models</td>
<td>mandatory</td>
</tr>
<tr>
<td>_ $\leftrightarrow$ _</td>
<td><em>is–instance–of</em> relation</td>
<td>mandatory</td>
</tr>
<tr>
<td>$\Phi$</td>
<td><em>features–in–a–model</em> function</td>
<td>mandatory</td>
</tr>
</tbody>
</table>
How to specify analysis operations for *variability models*?

Using formal methods

How to gain confidence about the specification correctness?

Using metamorphic testing techniques
Is the model consistent?

Yes, it represents at least one product
% SPL instances containing feature models generated by BeTTy
spl_db( spl_001, spl( ... ) ).
...
spl_db( spl_999, spl( ... ) ).

% Input data and expected result generated by BeTTy
test_data( spl_001, number_of_products, [ ], 12 ).
...
test_data( spl_999, number_of_products, [ ], 26 ).

% Test cases
:- begin_tests( number_of_products ).

% A test predicate for each test case that...
% 1. Retrieves an SPL instance
% 2. Retrieves input data and expected results
% 3. Performs the analysis operation
% 4. Compares the actual and expected results

test( number_of_products_001 ) :-
    spl_db( spl_001, SPL ),
    test_data( spl_001, number_of_products, [ ], EXPECTED ),
    nop( SPL, ACTUAL ),
    ACTUAL =:= EXPECTED.

test( number_of_products_002 ) :-
    ...

:- end_tests( number_of_products ).
Conclusions

Useful techniques in practice and...

- Variability Analyses
- Formal Methods
- Testing
SPL Team and collaborators

- Doctors
  - Antonio Ruiz-Cortés
  - Amador Durán
  - David Benavides
  - Sergio Segura
  - Fabricia Carneiro

- PhD Students
  - Pablo Trinidad
  - Jesús García
  - Sasha Valdés
  - José A. Galindo
  - Ana Belén Sánchez
Variability Models, Automated Analysis, Testing and Formal Methods. Putting all together

David Benavides

Department of Computer Languages and Systems
University of Sevilla