Software Impacts 9 (2021) 100085

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

DirectDebug: A software package for the automated testing and debugging of feature models

Viet-Man Le [a,*], Alexander Felfernig [a], Thi Ngoc Trang Tran [a], Müslüm Atas [a], Mathias Uta [b], David Benavides [c], José Galindo [c]

[a] Institute of Software Technology, Graz University of Technology, Graz, Austria
[b] Siemens, Energy AG, Erlangen, Germany
[c] Computer Languages and Systems, University of Sevilla, Seville, Spain

ARTICLE INFO

Keywords:
Automated testing and debugging
Feature models
Variability models
Diagnosis
Conflict detection

ABSTRACT

Complex and large-scale feature models can become faulty, i.e., do not represent the expected variability properties of the underlying software artifact. In this paper, we propose the DirectDebug algorithm that supports the automated testing and debugging of variability models. Our approach assists software engineers in identifying an adaptation hint (diagnosis) that makes all test cases consistent with the knowledge base. We also develop the software package so-called `d2bug_eval` to evaluate the DirectDebug's performance. The software package can be re-produced thoroughly to evaluate consistency-based algorithms.

Code metadata

Current code version	v1.1
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2021-45
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/5824065/tree/v1
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Java 8, Maven, IntelliJ IDEA
Compilation requirements, operating environments & dependencies	Maven, Betty framework, SXFM library
If available Link to developer documentation/manual	https://github.com/AIG-ist-tugraz/DirectDebug/blob/main/README.md
Support email for questions	vietman.le@ist.tugraz.at

Software metadata

Current software version	v1.1
Permanent link to executables of this version	https://github.com/AIG-ist-tugraz/DirectDebug/releases/tag/v1.1
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/5824065/tree/v1
Legal Software License	MIT License
Computing platforms/Operating Systems	Linux, macOS, Microsoft Windows
Installation requirements & dependencies	Java 8
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/AIG-ist-tugraz/DirectDebug/blob/main/d2bug_eval.jar.md
Support email for questions	vietman.le@ist.tugraz.at

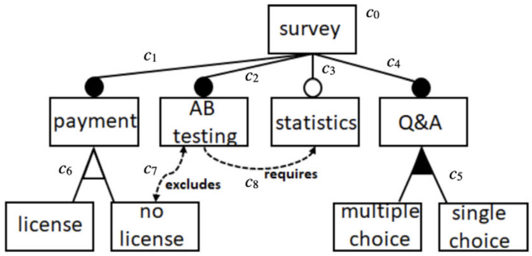
1. Introduction

The development of feature models [1,2] has to be pro-actively supported by *intelligent debugging mechanisms* that detect unexpected behaviors of a feature model knowledge base (e.g., *misinterpretations in domain knowledge communication*, *modeling errors*, or *outdated parts of a knowledge base* [3–5]). The existing literature has witnessed a few approaches supporting such mechanisms [5–7]. For instance, Junker [6]

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: vietman.le@ist.tugraz.at (V.-M. Le), alexander.felfernig@ist.tugraz.at (A. Felfernig), ttrang@ist.tugraz.at (T.N.T. Tran), muatas@ist.tugraz.at (M. Atas), mathias.uta@siemens.com (M. Uta), benavides@us.es (D. Benavides), jagalindo@us.es (J. Galindo).

https://doi.org/10.1016/j.simpa.2021.100085
Received 23 April 2021; Received in revised form 7 May 2021; Accepted 10 May 2021



(a)

c_0	$survey = t$
c_1	$survey \leftrightarrow payment$
c_2	$survey \leftrightarrow ABtesting$
c_3	$statistics \rightarrow survey$
c_4	$survey \leftrightarrow Q\&A$
c_5	$Q\&A \leftrightarrow multiplechoice \vee singlechoice$
c_6	$(license \leftrightarrow \neg nolicense \wedge payment)$ $\wedge(nolicense \leftrightarrow \neg license \wedge payment)$
c_7	$\neg(ABtesting \wedge nolicense)$
c_8	$ABtesting \rightarrow statistics$

(b)

Fig. 1. (a) An example of a *survey software* feature model and (b) generated constraints from the model.

introduces an algorithm to identify *conflicts* in a knowledge base responsible for inconsistencies. Conflicts are the basis for follow-up *diagnosis* operations that help to resolve these conflicts [4,8–10]. *A diagnosis* entails a set of elements of a knowledge base that have to be adapted/deleted to restore consistency. To improve the performance of diagnosis approaches, the idea of *direct diagnosis* was proposed [11], which supports diagnosis calculation without predetermining conflicts. In this paper, we propose the DirectDebug algorithm extending the *direct diagnosis* approach for supporting the automated testing and debugging of variability models [11]. Our approach considers a *set of test cases at the same time*, i.e., a diagnosis represents an adaptation proposal that makes all of the given test cases consistent with the knowledge base.

2. Working example

To illustrate our approach, we use an example of a *presumably faulty feature model* from the domain of *software services supporting the creation and management of surveys* (see Fig. 1a). A corresponding CSP[1]-based representation of a feature model configuration task $(F, D, C = CF \cup CR)$ can be generated [12], where F is the set of features and each feature has a specified domain $d_i = \{(t)rue, (f)alse\}$ $(d_i \in D)$, CF consists of constraints of the feature model (see Fig. 1b), and CR refers to user requirements.

3. Automated debugging

Test cases are used to induce conflicts in the feature model. There are two types of test cases [9]. (1) *Positive test cases* $(t_i \in T_\pi)$ specify intended behaviors of a knowledge base, i.e., at least one configuration is consistent with t_i. (2) *Negative test cases* $(t_i \in T_\Theta)$ specify unintended behaviors of a knowledge base, i.e., there should not exist any configuration consistent with t_i. These test cases are integrated in negated form into the background knowledge. For details of the test case generation, we refer to [13].

A diagnosis (Δ) includes constraints responsible for the faulty behavior of a feature model (see [13]). Such constraints have to be deleted or adapted to make the feature model consistent with T_π. Table 2 shows two options of deleting/adapting the constraints in CF such that the consistency between the feature model (Fig. 1a) and test cases $\{t_1..t_4\}$ (Table 1) is restored.

DirectDebug (see Algorithm 1) is activated with the diagnosis candidates $C \subseteq CF \setminus \{c_0\}$ and the background knowledge $B = CF \setminus C \cup \{c_0\} \cup T_\Theta^-$. T_Θ^- represents a conjunction of negative test cases (in negated form) which are assumed to be consistent with $C \cup B$. T_π, in the first DirectDebug call, represents test cases that are inconsistent with $C \cup B$. The algorithm returns an MSS - Γ (Maximum Satisfiable Subset — see *Definition 3* in [13]), a corresponding diagnosis is $C \setminus \Gamma$. If CF is diagnosed as a whole, then $C = CF \setminus \{c_0\}$ and $B = \{c_0\} \cup T^-$. Before starting DirectDebug, all t_i in T_π and T_Θ have to be checked for consistency with $C \cup B$. IsConsistent checks if the constraints in $C \cup B$

Table 1
Example positive test cases $T_\pi = \{t_1..t_4\}$ and corresponding conflicts identified by QuickXPlain [6]. Without losing generality, we assume negative test cases $T_\Theta = \emptyset$.

ID	Test case (Constraint)	Corresponding conflicts
t_1	$nolicense = t$	$\{c_2, c_8\}$
t_2	$license = t \wedge statistics = f$	$\{c_2, c_7\}$
t_3	$payment = f$	$\{c_1\}$
t_4	$singlechoice = f$	$\emptyset$

Table 2
Example diagnoses $\Delta_1 = \{c_1, c_2\}$ and $\Delta_2 = \{c_1, c_7, c_8\}$.

Diagnosis	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
Δ_1	×	×	–	–	–	–	–	–
Δ_2	×	–	–	–	–	–	×	×

are consistent with the individual test cases in T_π. This function returns *true* if every test case in T_π is consistent with $C \cup B$. Only test cases inducing an inconsistency with $C \cup B$ are returned in T'_π (the remaining inconsistent positive test cases). A consistency check is only activated if $\delta \neq \emptyset$ ($\delta = \emptyset$ indicates that C has already been checked for consistency with B).

Algorithm 1 DirectDebug$(\delta, C = \{c_1..c_n\}, B, T_\pi) : \Gamma$

```
T'_π ← T_π
if δ ≠ ∅ ∧ IsConsistent(C ∪ B, T_π, T'_π) then
    return(C)
end if
if |C| = 1 then
    return(∅)
end if
k = ⌊n/2⌋
C_1 ← c_1...c_k; C_2 ← c_{k+1}...c_n;
Γ_2 ← DirectDebug(C_1, C_1, B, T'_π);
Γ_1 ← DirectDebug(C_1 − Γ_2, C_2, B ∪ Γ_2, T'_π);
return(Γ_1 ∪ Γ_2)
```

An execution trace of DirectDebug is shown in Fig. 2. DirectDebug follows a *divide & conquer* approach. In each recursive call, it searches for positive test cases that are inconsistent with $C \cup B$. If there is only one constraint c_i in the consideration set C ($|C| = 1$) and at least one test case t_j where *inconsistent*$(\{t_j\} \cup C \cup B)$, then c_i is considered a part of a diagnosis Δ. If *consistent*$(\{t_i\} \cup C \cup B)$, $\forall \{t_i\} \in T_\pi$, then C is returned since no diagnosis elements can be found in C. DirectDebug returns a MSS ($\Gamma = \{c_2..c_6\}$). The corresponding minimal diagnosis Δ is $C \setminus \Gamma$ ($\Delta = \{c_1, c_7, c_8\}$).

4. Software features

We developed a Java-based software package so-called `d2bug_eval` to evaluate the performance of DirectDebug and to support the *reproducibility* of the DirectDebug evaluation process.

[1] CSP — Constraint Satisfaction Problem.

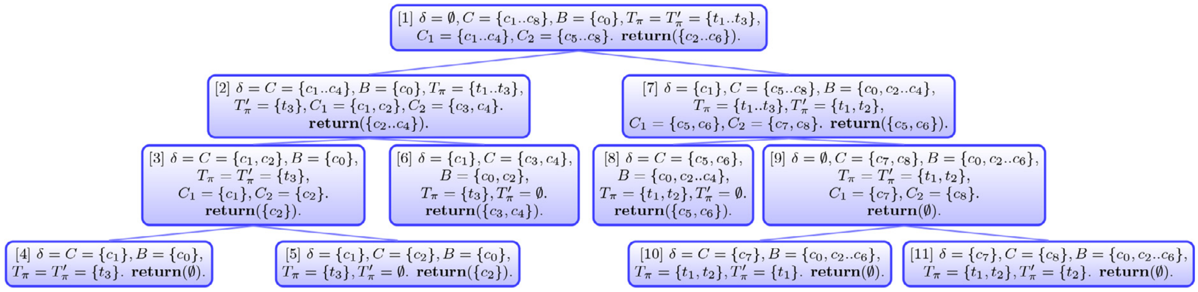


Fig. 2. DirectDebug execution trace for $C = \{c_1..c_8\}$, $B = \{c_0\}$, and $T_\pi = \{t_1..t_3\}$.

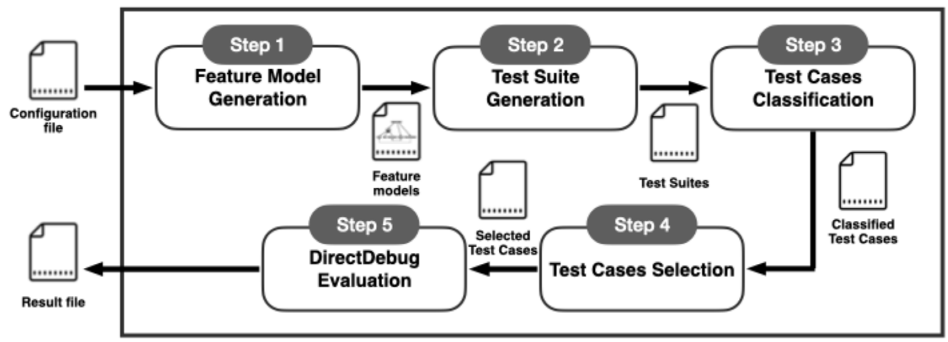


Fig. 3. The flowchart of `d2bug_eval` where five steps of the DirectDebug evaluation process are performed.

Table 3
An example of the run-time performance (in *msec*) of DirectDebug after three iterations. In each $|CF|$ variant, we generated 3 feature models. In total, there were 378 (3feature models $\times$ 6 $|CF|$ $\times 7$ $|T_\pi|$ $\times 3$) selected test-case scenarios.

$\lvert T_\pi \rvert$	$\lvert CF \rvert$					
	10	20	50	100	500	1000
5	0.1	0.4	1.1	2.4	24.8	114.0
10	0.2	0.5	1.8	4.3	35.4	170.4
25	0.6	1.3	4.3	11.7	108.1	375.5
50	1.0	2.3	7.9	22.7	241.6	695.5
100	1.9	4.5	14.7	38.8	449.0	1251.5
250	6.6	11.2	32.4	90.4	1191.3	2877.9
500	20.3	24.2	60.3	158.2	2338.1	5680.1

`d2bug_eval` can be easily integrated into external projects and compatible with different platforms (e.g., `Windows`, `Linux`, and `macOS`). Moreover, it is *self-contained*, which performs five steps of the DirectDebug evaluation process (see Fig. 3). In *Step 1*, feature models are generated randomly using BeTTy [14] and then translated into constraints in Choco Solver [15]. *Step 2* generates a test suite for each feature model consisting of five types of test cases: dead features, false optional, full mandatory, false mandatory, and partial configuration. In *Step 3*, test cases of each test suite are classified into *violated* and *non-violated* test cases. In *Step 4*, the program selects test-case scenarios where the ratio of violated test cases to non-violated test cases is a specific number predetermined by the user . The number of scenarios is selected depending on the combination of the number of constraints $|CF|$ and the number of test cases $|T_\pi|$. For each combination, the average run-time will be calculated (in *Step 5*) when a specific number of iterations ($|iter|$) is reached. Finally, in *Step 5*, the program calculates the average run-time of DirectDebug, shown in each cell of Table 3.

In addition to the *reproducibity* and the *self-contained* ability, `d2bug_eval` shows also the *parametricity* that provides a wide range of parameters (e.g., $|CF|$, $|T_\pi|$, $|iter|$, $\%violated/non\text{-}violated$) in order to facilitate the customization of the evaluation process.

Structurally, `d2bug_eval` consists of three sub-packages: `Feature Model`, `MBDiagLib`, and `Debugging`. `Feature Model` reads feature model files and supports *feature model generation* and *feature model statistics*. `MBDiagLib` provides (1) an abstract model to hold variables and constraints, (2) an abstract consistency checker for underlying solvers, (3) a Choco consistency checker using Choco Solver [15], and (4) functions to measure the performance of algorithms in terms of run-time or the number of solver calls. `Debugging` provides components w.r.t. test-cases management, the DirectDebug implementation, a debugging model with test-cases integration, and debugging-related applications (e.g., *test suite generation*, *test cases classification*, and *test case selection*).

5. Impact overview

DirectDebug [13] is a practical approach to assist software engineers in pro-actively identifying minimal sets of faulty constraints responsible for unintended behaviors of a feature model. This way, this approach helps to accelerate feature model development and evolution processes.

Our software package `d2bug_eval` implements and evaluates the performance of DirectDebug. This software can be extended to evaluate other consistency-based algorithms, such as *conflict detection algorithms* and *diagnosis identification algorithms*. Basic versions of the software have been applied in the OpenReq project[2] in the context of release planning scenarios [16]. Besides, they have also been adopted by one international configuration provider and by SelectionArts[3] in constraint-based recommender solutions.

6. Limitation

One limitation of `d2bug_eval` is related to the support of feature model formats and used solvers (only the SXFM format [17] and the Choco Solver [15] are supported in the current version of `d2bug_eval`). However, the software can be totally extended to include further formats and solvers. Another limitation lies in the ability of the software to be integrated into well-known feature model support tools, such as FeatureIDE [18]. This integration is essential to increase the utility of the DirectDebug algorithm as well as its evaluation process.

[2] openreq.eu.
[3] www.selectionarts.com.

7. Conclusion

In this paper, we proposed a feature model testing and debugging approach that pro-actively supports feature model designers and accelerate feature model development and evolution processes. We developed the software package `d2bug_eval` that can be exploited by the research community to fully reproduce our work. Future work includes developing techniques for the automated test case generation considering different coverage metrics. Besides, we will include feature model quality metrics to better predict the most relevant diagnoses. Finally, the software package can be reinforced by integrating additional functionalities, e.g., new test case generation and other off-the-shelf solvers supporting other representations of satisfaction problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially funded by the Austrian Research Promotion Agency ParXCel project (880657), and the EU FEDER program MINECO project Ophelia (RTI2018-101204-B-C22).

References

[1] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Inf. Syst. 35 (6) (2010) 615–636, http://dx.doi.org/10.1016/j.is.2010.01.001.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[3] D. Benavides, A. Felfernig, J.A. Galindo, F. Reinfrank, Automated analysis in feature modelling and product configuration, in: J. Favaro, M. Morisio (Eds.), Safe and Secure Software Reuse, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 160–175.

[4] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, M. Toro, Automated error analysis for the agilization of feature modeling, J. Syst. Softw. 81 (6) (2008) 883–896, http://dx.doi.org/10.1016/j.jss.2007.10.030, Agile Product Line Engineering.

[5] A. Zeller, Automated debugging: Are we close, IEEE Ann. Hist. Comput. 34 (11) (2001) 26–31.

[6] U. Junker, QuickXPlain: Preferred explanations and relaxations for over-constrained problems, in: Proceedings of the 19th National Conference on Artifical Intelligence, in: AAAI'04, AAAI Press, 2004, pp. 167–172.

[7] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, A. Ruiz-Cortes, Automated diagnosis of feature model configurations, J. Syst. Softw. 83 (7) (2010) 1094–1107, http://dx.doi.org/10.1016/j.jss.2010.02.017, SPLC 2008.

[8] R.R. Bakker, F. Dikker, F. Tempelman, P.M. Wogmim, Diagnosing and solving over-determined constraint satisfaction problems, in: Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1, in: IJCAI'93, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993, pp. 276–281.

[9] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence 152 (2) (2004) 213–234, http://dx.doi.org/10.1016/S0004-3702(03)00117-6.

[10] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1) (1987) 57–95, http://dx.doi.org/10.1016/0004-3702(87)90062-2.

[11] A. Felfernig, M. Schubert, C. Zehentner, An efficient diagnosis algorithm for inconsistent constraint sets, Artif. Intell. Eng. Des. Anal. Manuf. 26 (1) (2012) 53–62, http://dx.doi.org/10.1017/S0890060411000011.

[12] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, K. Wolter, Chapter 6 - Configuration knowledge representation and reasoning, in: A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen (Eds.), Knowledge-Based Configuration, Morgan Kaufmann, Boston, 2014, pp. 41–72, http://dx.doi.org/10.1016/B978-0-12-415817-7.00006-2.

[13] V.-M. Le, A. Felfernig, M. Uta, D. Benavides, J. Galindo, T.N.T. Tran, DirectDebug: Automated testing and debugging of feature models, in: Proceedings of IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), in: ICSE-NIER '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 81–85, http://dx.doi.org/10.1109/ICSE-NIER52604.2021.00025.

[14] S. Segura, J.A. Galindo, D. Benavides, J.A. Parejo, A. Ruiz-Cortés, BeTTy: Benchmarking and testing on the automated analysis of feature models, in: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, in: VaMoS '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 63–71, http://dx.doi.org/10.1145/2110147.2110155.

[15] C. Prud'homme, J.-G. Fages, X. Lorca, Choco Solver Documentation, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016, URL http://www.choco-solver.org.

[16] A. Felfernig, M. Stetinger, A. Falkner, M. Atas, J. Franch Gutiérrez, C. Palomares Bonache, Openreq: recommender systems in requirements engineering, in: Proceedings of the Workshop Papers of I-Know 2017: Co-Located with International Conference on Knowledge Technologies and Data-Driven Business 2017 (I-Know 2017): Graz, Austria, October 11-12, 2017, CEUR-WS. org, 2017, pp. 1–4.

[17] M. Mendonca, M. Branco, D. Cowan, S.P.L.O.T.: Software product lines online tools, in: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, in: OOPSLA '09, ACM, New York, NY, USA, 2009, pp. 761–762, http://dx.doi.org/10.1145/1639950.1640002.

[18] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, FeatureIDE: An extensible framework for feature-oriented software development, Sci. Comput. Program. 79 (2014) 70–85, http://dx.doi.org/10.1016/j.scico.2012.06.002.