Customization and 3D Printing: A Challenging Playground for Software Product Lines

Mathieu Acher, Benoit Baudry, Olivier Barais, Jean-Marc Jézéquel

Customization and 3D Printing: A Challenging Playground for Software Product Lines

Mathieu Acher, Benoit Baudry, Olivier Barais, and Jean-Marc Jézéquel
Inria / IRISA, University of Rennes 1, France
firstname.lastname@irisa.fr

ABSTRACT

3D printing is gaining more and more momentum to build customized product in a wide variety of fields. We conduct an exploratory study of Thingiverse, the most popular Website for sharing user-created 3D design files, in order to establish a possible connection with software product line (SPL) engineering. We report on the socio-technical aspects and current practices for modeling variability, implementing variability, configuring and deriving products, and reusing artefacts. We provide hints that SPL-alike techniques are practically used in 3D printing and thus relevant. Finally, we discuss why the customization in the 3D printing field represents a challenging playground for SPL engineering.

Categories and Subject Descriptors

D.2 [**Software Engineering**]; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

Keywords

3D Printing, Software Product Lines, Customization

1. INTRODUCTION

3D printing (also called additive manufacturing) is commonly used to produce physical models for a wide variety of applications (e.g., archaeology, industrial prototyping and design, medicine, rapid manufacturing). While having been available for more than a decade, the technology recently developed additional momentum [1, 2]. With 3D printers dropping in price and with 3D printing services becoming available to a large public, many additional use cases arise. The next evolution of 3D printing may well be *customization* for the masses. By printing one product at a time, 3D printers and customization's ability can fit the buyers identity, shape or preferences, so that their input becomes part of the creation process. For instance, there are commercial services that fabricate figurines from multiple-choice designs of puppets [3], robot toys [4], or foot orthotics [5].

Customization is also an inherent subject of *Software Product Line* (SPL) engineering. In essence, SPL engineering focuses on the means of efficiently producing and maintaining multiple similar software products, exploiting what they have in common and managing what varies among them. A central interest of SPL engineering is the management of *variability*, i.e., the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context [6].

Therefore the connection between SPL engineering and 3D printing is appealing; both share the goal of mastering customization. Moreover 3D printing heavily relies on software technologies (e.g., computer-aided design tools) for creating 3D objects. Yet, to the best of our knowledge, there is no existing study that establishes a relationship between SPL and 3D. On the one hand, the scope and understanding of SPL and variability continuously broadens, for example, in the operating system domain [7], in software ecosystems [8], in configurators [9, 10] or comparators [11], and in industrial contexts [12, 13]. But 3D printing has not yet drawn the attention of the SPL community. On the other hand, we are not aware of application of SPL techniques in the 3D printing domain.

Our study of the Website Thingiverse [14], the most active community in 3D printing, reveals that SPL-alike techniques are practically used in 3D printing and thus relevant. We report on the customization practices in the 3D printing domain from the perspective of SPL engineering, opening avenues for further research.

2. A FIELD STUDY OF THINGIVERSE

We conducted an exploratory study to establish a possible connection between SPL engineering and 3D printing. The field of 3D printing is rich and diverse. We chose Thingiverse [14] the most popular [15] Website for sharing user-created 3D design files.

2.1 Thingiverse

Thingiverse [14] provides a Web interface and online tools to discover, print, and share 3D models. A large 3D design community emerged, constituted of 3D printers operators (including the owner of Thingiverse MakerBot Industries, a company producing 3D printers), creators, and individuals. For instance, numerous projects use Thingiverse as a repository for shared innovation and dissemination of source materials to the public. Typically, users can browse 3D printable *Things* made by designers (or other individuals).

Things are the basic units of Thingiverse 3D community. A *Thing* centralizes information about a 3D design (general information, instructions, files, photos, etc.) as well as users' comments. Figure 1 gives an excerpt of the Web interface of a *Thing* (a case for holding batteries).

We consider Thingiverse [14] as representative of the 3D printing domain and our research interest (customization). A press release of MakerBot reported (January 24, 2014) that "Thingiverse is the largest place to discover and print 3D models and contains the world's largest 3D printing community with more than 160,000 members and 200,000+ and

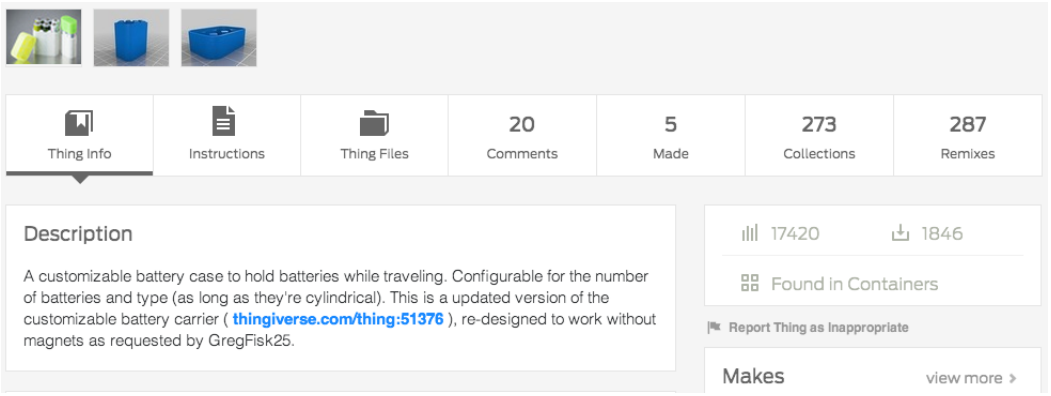

Figure 1: Excerpt of the Web Thingiverse interface exposing a Thing [16]

growing downloadable digital designs of free 3D models" [15]. Moreover Thingiverse provides a dedicated area (called *Customization*) as well as online tools and languages to customize 3D models. Besides, the *availability* of data supports our exploration. Resources (e.g., 3D models) are typically licensed under the GNU GPL or Creative Commons licenses. Exchanges and discussion inside the community can be publicly consulted (e.g., through comments in a *Thing*).

2.2 Research Method

We apply an ethnographic method [17]. We observed how a community of people – the Thingiverse community – interacts and collaborates for achieving a technical work – customizing and producing 3D models.

Research questions. In ethnography, the researcher should "explicitly consider his/her own pre-conceptions and how they influence understanding of the studied community" [17]. As stated in the introduction, we observed Thingiverse from the *perspective* of SPL engineering and customization: What are the customization practices of the Thingiverse community? How do the current 3D printing practices compare to state-of-the-art SPL practices? Specifically, two main perspectives guided our investigation: *(1) Can SPL engineering learn from 3D printing?* Are there SPL-alike tool or language targeted towards the customization of 3D models? What are their characteristics? *(2) Can 3D printing benefit from SPL engineering?* Can SPL techniques help to model and realize variability in 3D models?

Data. Our approach to data collection and analysis focused on the *"Customizable Things"* category of Thingiverse. A preliminary analysis convinced us that (1) there is a substantial material (*Things*) in this category to perform empirical observations; (2) other *Things* are not necessarily subject to customization. Moreover, the Thingiverse community promotes the use of customization tools/languages and encourages users to upload their custom 3D models in this dedicated category.

Still, performing qualitative observations (i.e., *Thing* by *Thing*) over thousands of customizable *Things* is not possible. We thus considered the popularity of *Things* to further filter the data. The rationale is that more insights can be inferred from "popular" projects. As there is no direct way to sort *Things* by popularity, we first analyzed "featured" customizable things. We considered customizable *Things* with more than four comments greater than four and took care of covering every subcategories and domains (art, fashion, toys and games, etc.). Overall, around 50 *Things* have been carefully reviewed.

Observations. The four authors of the paper performed the observations. All participants have a strong background in SPL and software engineering. All the observations have been made during April 2014. For each *Thing*, we analyzed the following artefacts (see Figure 1):

- *Thing Info:* provides a broad description in natural language of the *Thing*. In the example of Figure 1, there is an indication on the kinds of options available. There is also an indication on the provenance of the original *Thing*;
- *Instructions*: describing how to use the different files or giving advices on how to customize a *Thing*;
- *Thing Files*: .stl and .scad files are uploaded and listed. Roughly, digital 3D models are saved in STL format and can then be sent to a 3D printer. A .scad script can be compiled and rendered as a 3D model (with e.g. OpenSCAD). We only considered .scad files, which are subject to customization;
- *Comments*: users acknowledge the creator, complain, give advices, exchange information about a *Thing*;
- *Remixes*: a *Thing* can be "remixed" into another *Thing*

We also considered two additional resources (documentation) of Thingiverse that help developers to customize 3D files [18,19]. For each *Thing*, participants filled out a spreadsheet to report their observations. We got a meeting to report on key insights (including surprising observations) and to establish relationships with existing SPL techniques (see Section 3). We also discussed the challenges for SPL engineering with regards to our exploratory analysis of the Thingiverse 3D printing community (see Section 4).

3. ANALYSIS AND RESULTS

We report the results of our analysis in four sections, each corresponding to an SPL topic.

3.1 Variability Modeling and Configuration

Thingiverse developed an extension [18] of the OpenSCAD language to allow users to design parametric objects (see Figure 2 for an example). Parameters are units of variability and are placed at the top of a .scad script before the first module declaration. Special comments can be added to define the possible values of parameters:

- multiple, exclusive choices of numbers or strings: one value should be selected among the possible choices. Each possible value can be labelled;

```
/* [Customize body] */

//Set the outside length of your pencil box.
length=190;//[70:400]
//Set the outside depth of your pencil box.
depth=70;//[50:400]
//Set the total height of your pencil box.  The top of the box is set at 15mm.
//Extra height is added to the body section.
height=40;//[40:150]

//Choose divider orientation.  Long is for the X direction.
long = 1;//[0,1,2]
//Short is for the Y direction.
short = 2;//[0,1,2,3]
//When you have 2 long dividers,
// picking yes here will put short dividers in the center section.
center = 0;//[1:Yes,0:No]
 .
 .
 .

module body(fn=20) {
    difference(){
        union(){
            case(z-15,fn);
            hinge(0,z-15,180,fn);
            translate([0,-y/2+5.75+wall,(z-10)/2+t])
                cube([13,12,z-20-2*t],center=true);
            intersection(){
                translate([0,-y/2+5.75+wall,5])
                    cube([13,12,10],center=true);
                translate([0,-y/2+rout+wall,rout/1.5])
                    rotate([0,90,0])
                        cylinder(r=rout,h=13,center=true,$fn=fn);
            }
            if (long>0){
                translate([0,-(long-1)*yin/6,0]){
                    for (i=[1:long])
                        translate([0,(i-1)*yin/3,0])
                            divider(xin,z-15,2,fn);
                }
                if (short>0){
                    translate([-(short-1)*xin/6,-yin/(long+1)*(long/2),0])
                        for (i=[1:short])
                            translate([(i-1)*xin/3,0,0])
                                rotate([0,0,90])
                                    divider(yin/(long+1),z-15,1,fn);
                }
            }
```

① ②

Figure 2: .scad script: excerpt of global parameters ① and a module ② (Thing [20])

- a number interval (min/max): the value must lay between the min and max.

Values are assigned to parameters. In case a special comment is associated to a parameter, it is a *default* value. Otherwise, the possible values of the parameter are undefined. The modeling of parameters is directly *integrated* in the script and not *separated* in an external description (e.g., in a variability model).

Configurator. The parameters are exploited in the different modules of the .scad script, but may also have a visual counterpart. In fact, an online tool (called Customizer) processes the comments to present the parameters as (1) dropdown list (users can choose one and only one value from a list); (2) sliders (users can set a value by moving an indicator); (3) text boxes (in case the parameter value is fixed and no possible values have been defined). The three kinds of widgets realize the semantics of the special comments. Similar widgets are employed by Web configurators [10] to visually represent variability (configuration options).

Visibility. Some parameters can be marked as *hidden* and are not visually presented in the Customizer tool. Some other parameters are qualified as *global* and can be exploited by users for configuring a 3D model. A similar distinction is made in SPL engineering between *external* variability (visible to customers) and *internal* variability (related to implementation details) [21, 22].

Variability concepts. We observed that multiple choices can be employed to model either *optionality* (when only two values are possible, typically "true" or "false") or an *alternative* group of values, basic concepts used in feature or decision modeling [23]. Number intervals (also called sliders) are widely used and could be represented with feature attributes or specific types. The notion of *mandatory* parameters is not apparent. Our observation is that the setting of values to parameters does not mean the value is not subject to change. On the contrary, numerous informal comments suggest some alternative values ("you can set this lower if your printer can handle tiny towers"). Moreover there is no language construct (e.g., constant) to indicate that a value will not change. Hence even values of hidden parameters can change. Parameters can also be grouped into tabs (using special comments). A tab has a name and separates related parameters in different sections or themes (e.g., to focus on a certain concern of a 3D model). Tabs help to structure the parameters and can be related to the notion of *feature hierarchy* or *views* in feature modeling [22] or *decision groups* in decision modeling [23]. Tabs are also exploited by the Customizer to visually organize parameters in the interface.

Constraints. Perhaps the most surprising observation is the absence of mechanisms for:

- *specifying constraints between parameters:* We observed comments (in the instructions, in the source code of a .scad files, or in users' discussions of a *Thing*) that suggest logical relationships between parameters:

  > *// we don't suggest setting this smaller than your_nozzle_diameter X 2*
  > ...
  > *// height of the battery recess (height of top and bottom piece should be equal to or slightly larger than the battery height)*

- *restricting further possible values of a parameter*: The analysis of comments showed that some values are not possible because of, e.g., some physical properties of the product or the usage context, inabilities of the 3D printer to handle the value.

  > *// don't make too big or your gear will disappear*
  > ...
  > *// It should be a multiple of your slicer's layer height for best results.*

Some constraints are quite hard to understand (for a non expert of the product/domain) and formalize, especially those referring to physical properties. In any case, there is currently no way to express such constraints and no tool can automatically verify domain properties.

Configuration Issues. Numerous discussions are related to the problem of choosing correct values (configuring). Users complain about some combinations of parameters (reinforcing the need to support constraints):

> *I tried to create a lid with a negative part gap -.1 (3mm lip), but anything less than zero generates a lid with no recessed step to accept the lip. Did I do something wrong?*

report on their experience and suggest some modifications.

> *5width = 57.57;*
> *should be:*
> *5width = 58.57;*

Interestingly, creators of *Things* request feedbacks and recognize the difficulties of modeling parameters.

"Let us know what kinds of settings you use to print them!"
...
"Thank you for your comments. We are still playing around with settings and will keep updating it."

A major configuration difficulty arises when special comments are not specified for parameters and thus the possible values are unclear and hard to choose for users.

Complexity. The number intervals (sliders) and the absence of constraints among parameters makes possible a huge number of possible variants. As an extreme case, we observed a *Thing* with 38 parameters, 8 tabs, for an estimation of $\geq 10^{28}$ possible variants. A large proportion of customizable *Things* we observed has more than 10 parameters.

Multi Parts. A manufactured product can be constituted of multiple parts to print separately and to manually combine. Each part is subject to customization. The challenge is then to find the right combination of parameters for each part. It further complicates the task of modeling parameters and configuring.

3.2 Variability Implementation

Creators of customizable *Things* rely on the OpenSCAD language for implementing their 3D design models. It is a scripting language with variables, modules (here: procedures), basic control structures (conditional, iterator), data types (Boolean, numbers, strings, vectors, matrices, etc.) and geometry primitives. .scad scripts naturally exploit parameters (see previous sections) to customize the generation of 3D models (see Figure 2). Specifically the implementation of variability relies on the following mechanisms:

- **parameterized procedures**: developers specify modules (reusable procedure) with a list of parameters. The modules are then called: the arguments are typically the variables corresponding to global or hidden parameters declared at the top of the script. Inside a procedure, the parameter can serve to iterate a certain amount of times, to execute some statements depending on the values, or there can be calls to other parametrized modules (see Figure 2). Interestingly, named parameters (or keyword arguments) are supported, which can be useful when the list of parameters is long. It is also possible to set default parameter values, which can be seen as an additional way to model variability (see previous section). We observed numerous modules with more than 10 parameters. As an extreme case, we noticed 26 parameters for a module and around 600 lines of code;

- **external call of parameters**: some modules do not declare parameters. Instead, global or hidden parameters (outside the module) are used. It also happens that a parametrized module calls a global or hidden parameter (hybrid solution).

- **conditional statements** (if/else/else if) are widely used typically to call a specific module (see Figure 2). According to our observations, the cyclomatic complexity can reach very high values. The complexity of "cases" also: we noticed 33 imbrications of if for a *Thing*. A substantial amount of if does not come with an else, leaving undocumented what happens when the initial condition does not hold;

- **pre-compiled procedures or data structures:** some modules are the result of a compilation (e.g., import from another tool). Typically a series of if/else/else if can be employed to map a parameter to a specific module. Similarly, some vectors (e.g., representing a specific shape) are compiled and imported in the .scad file. Interestingly, the pre-compiled artefacts are also parameterized in some cases.

3.3 Derivation

If a *Thing* is categorized as a "Customizable Thing", then the *Thing* page automatically displays an "Open In Customizer" button. Users can play with the Customizer by fixing some values in the text boxes or with the sliders and choosing a value in a drop down list. After each interaction, users can visualize the resulting 3D model and thus control that the models fits the requirements.

The derivation process is simply a compilation of the .scad file with the specific parameters' values given by the Customizer tool (configurator). It is typical of what is promoted by SPL approaches.

3.4 Reuse

Clone and Own? A *Thing* can be "remixed" in two different ways: (1) full customization, leading to the derivation of a 3D model (see previous section); (2) adaptation, new version based on the original files. We did not consider the first category, since they have a limited interest: it is simply an assignment of values using the customizer tool. (There is no comment, no additional file, just a log of values automatically generated in the instruction.) The second category is potentially more interesting: the *Thing* leads to a new *Thing* with new files, new comments, made, etc. In practice, Thingiverse highlights the second category with a panel "Remixed From".

There is an effort of the community to keep traces of the derivatives and original projects. At first glance, the practice resembles to "clone and own" – a form of reuse considered as ad-hoc in SPL engineering [12]. However the majority of remixed *Things* are completely new designs and thus global/hidden parameters and modules. We identified only one *Thing* that proposes .scad files close to the original. At the moment, remixes mostly aim to connect "similar" *Things* rather than reusing artefacts.

Modularity Mechanisms. There are limited mechanisms and attempts to reuse or extend a script. A few libraries are emerging but they are rather technical (e.g., for handling 3D vectors, for converting colors). The modules exhibit a substantial amount of parameters and are highly dependent in the code base. Such complexity represents a major limitation for reuse. Moreover there are no type names in OpenSCAD and no user defined types (i.e., there is a fixed set of data types). Finally an original form of reuse is the import of compiled binaries, modules, vectors, etc. from other tools (see Section 3.2) followed by its parameterization.

4. THE FUTURE OF 3D PRINTING AND SPL

Our field study of Thingiverse showed that the 3D printing community is using SPL-alike techniques for customizing 3D models. The visualization of custom 3D model, with

respect to the setting of some values through a configurator (called a Customizer), is a derivation process as found in traditional SPL approaches. Specifically Thingiverse has developed languages and tools for:

- **modeling variability:** the solution is integrated into a standard (OpenSCAD), is quite simple to learn and use (with a configurator), and has some similarities with variability languages (e.g., feature or decision model). Surprisingly, constraints between parameters cannot be expressed;
- **implementing variability:** basic techniques are employed (essentially parameterized modules and extensive use of conditions).

The resulting complexity of the customizable *Things* can be extremely high with dozens of parameters, each with many possible values, used in numerous inter-related modules. Moreover, the Thingiverse community exchanges around configuration issues, tries to fix implementation details, and develops novel customizable *Things* each day.

Our study is exploratory and currently limited to qualitative judgement. Future work includes quantitative analyses, on the different topics we have identified in this paper. In any case, we believe that SPL engineering has a role to play, given the numerous research effort made to efficiently manage variability and customize software-intensive systems. For instance, state-of-the-art techniques and languages for modeling variability [21, 23], verifying SPLs [24], automated reasoning [25] and implementing variability [6] might play a key role in the coming boom of 3D printing. Yet, our initial analysis of variability practices in 3D printing make us wonder whether current SPL engineering can overcome the following challenges imposed in this context.

Challenge #1: capture "thing-specific" constraints. We identified many constraints that should be checked when customizing a *Thing*. Some constraints are about 3D objects in general (can the *Thing* be physically printed?) and some constraints are specific to the *Thing* to be printed (VGA monitor and AC Power plugs should not be part of the same wall plate). Will SPL engineering formalisms manage to capture all these forms of constraints?

Yet, assuming that we can come up with languages, algorithms and tools to model and reason about constrained variability in *Things*, the SPL community will then have to address socio-technical challenges.

Challenge #2: introduce systematic engineering practices while keeping a low cognitive effort. This is perhaps the most difficult challenge for SPL engineering: keep the cognitive effort low to reach an increasing amount of people – not necessarily software developers. SPL techniques are certainly more sophisticated, but the accidental and even conceptual complexity induced by the tools and languages can annihilate the intended benefits.

Challenge #3: establish a cost-benefit tradeoff. When introducing novel techniques into an existing domain, it is necessary to find a tradeoff between the cost of applying a sophisticated technology and the resulting benefit out of this upfront investment. Will a solution pay off if the customizable 3D model is of interest for hundreds of customers? thousands? billions?

With 3D printers becoming more and more accessible to a wider public, the problem of mastering customization can become arbitrarily complex. Will SPL engineering take on the challenge of occupying the 3D printing playground?

Acknowledgements We thank the anonymous reviewers for their insightful comments. This work is partially supported by the EU FP7-ICT-2011-9 No. 600654 DIVERSIFY project.

5. REFERENCES

[1] H. Lipson and M. Kurman, *Fabricated: The new world of 3D printing*. John Wiley & Sons, 2013.
[2] B. Berman, "3-d printing: The new industrial revolution," *Business Horizons*, vol. 55, no. 2, pp. 155 – 162, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0007681311001790
[3] http://makie.me/.
[4] http://cubify.com/Store/MRN.
[5] http://www.sols.co/.
[6] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques: Research articles," *Softw. Pract. Exper.*, vol. 35, no. 8, pp. 705–754, 2005.
[7] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "A study of variability models and languages in the systems software domain," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2013.
[8] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of Systems and Software*, vol. 83, no. 1, pp. 67–76, 2010.
[9] A. Hubaux, D. Jannach, C. Drescher, L. Murta, T. Mannisto, K. Czarnecki, P. Heymans, T. Nguyen, and M. Zanker, "Unifying software and product configuration: A research roadmap," in *Proceedings of the Workshop on Configuration (ConfWS'12)*, Montpellier, France, 2012, pp. 31–35.
[10] E. Khalil Abbasi, A. Hubaux, M. Acher, Q. Boucher, and P. Heymans, "The anatomy of a sales configurator: An empirical study of 111 cases," in *CAiSE'13*, 2013.
[11] N. Sannier, M. Acher, and B. Baudry, "From Comparison Matrix to Variability Model: The Wikipedia Case Study," in *ASE'13*. IEEE, 2013.
[12] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," *CSMR'13*, vol. 0, pp. 25–34, 2013.
[13] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *VaMoS'13*. ACM, 2013.
[14] http://www.thingiverse.com.
[15] Makerbot, "Press release: Announcing MakerBot Thingiverse Android App and iOS 1.1 Update," jan 2014, http://www.makerbot.com/press-releases/.
[16] http://www.thingiverse.com/thing:57281, access: April 8, 2014.
[17] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 285–311.
[18] http://customizer.makerbot.com/docs, "Developer documentation (customizer)."
[19] http://customizer.makerbot.com/best_practices, "Customizer: Parametric design best practices."
[20] http://www.thingiverse.com/thing:147178.
[21] A. Hubaux, T. T. Tun, and P. Heymans, "Separation of concerns in feature diagram languages: A systematic survey," *ACM Comput. Surv.*, vol. 45, no. 4, pp. 51:1–51:23, Aug. 2013.
[22] A. Hubaux, M. Acher, T. T. Tun, P. Heymans, P. Collet, and P. Lahire, *Domain Engineering: Product Lines, Conceptual Models, and Languages*. Springer, 2013, ch. Separating Concerns in Feature Models: Retrospective and Multi-View Support.
[23] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, "Cool features and tough decisions: a comparison of variability modeling approaches," in *VaMoS'12*, 2012, pp. 173–182.
[24] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys*, 2014, to appear.
[25] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, 2010.