Journal of open hardware

Semi-Automatic Generation of Assembly Instructions for Open Source Hardware

ISSUES IN OPEN HARDWARE

J.C. MARISCAL-MELGAR
PIETER HIJMA
MANUEL MORITZ
TOBIAS REDLICH

*Author affiliations can be found in the back matter of this article

]u[ubiquity press

ABSTRACT

Documentation is an essential component of Open Source Hardware (OSH) projects both for co-development and replication of designs. However, creating documentation and keeping it up-to-date is often challenging and time-intensive. There are several systems that focus on this documentation challenge but they are limited in their support for keeping documentation up-to-date and relating CAD designs to documentation. This article proposes a semi-automated solution that relates the CAD design semantically to a textual specification from which we generate assembly instructions semi-automatically. Our system contains a CAD plugin and a compiler for the textual specification with which we show that we can replicate a state-of-the-art assembly manual to a high degree, that we can automate significant parts of the documentation process, and that our system can effectively adapt to documentation changes as a result of evolving designs. Our system leads to a methodology that we name "CAD-coupled documentation" integrating CAD design with the documentation process.

CORRESPONDING AUTHORS:
J.C. Mariscal-Melgar
Helmut-Schmidt University / University of the Federal Armed Forces, Hamburg, Germany
jc@hsu-hh.de

KEYWORDS:
Open Source Hardware; OSH; Documentation; Assembly Manual; Assembly Instructions; FreeCAD; automation

TO CITE THIS ARTICLE:
Mariscal-Melgar, JC, Hijma, P, Moritz, M and Redlich, T. 2023. Semi-Automatic Generation of Assembly Instructions for Open Source Hardware. *Journal of Open Hardware*, 7(1): 6, pp. 1–18. DOI: https://doi.org/10.5334/joh.56

1 INTRODUCTION

Open Source Hardware (OSH) is becoming a more prominent extension of the Open Culture movement [1] with almost 2000 projects certified by OSHWA [2] and commercial successes such as Arduino [3] and the 3D-printer field started by the RepRap printer [4]. Academia embraces OSH as well, for example CERN with their Open Hardware Initiative [5] and licenses [6].

Documentation is widely recognized to be crucial for OSH for the ability to replicate the hardware and for co-development of designs [7, 8, 9, 10, 11]. Although providing high-quality documentation alone is already considered time-intensive and challenging [12, 11], the context of co-development makes it even more challenging because of the risk of documentation becoming out-of-date.

Several systems exist that try to alleviate documentation of physical artifacts and many focus on the "document-while-doing" methodology (as opposed to "document-after-the-fact") [12] that suits OSH best [13, 14, 15, 16]. GitBuilding [17] is arguably the state-of-art that provides a high degree of automation and is focused on collaboration via Git [18].

However, none of these systems target (1) the problem to keep the documentation up-to-date while the design evolves and (2) trying to find a high degree of integration between the design and documentation process, an important requirement formulated by Tseng et al. [11].

Our system is designed with these two problems in mind and makes use of a CAD plugin specifically for documentation to semantically link the CAD designs to a textual specification (i.e. a markup text description) of assembly instructions, supporting various assembly instruction features. This leads to a methodology that we call "CAD-coupled documentation" and allows us to stay close to the "source" of OSH [9] in order to support semi-automatically regenerating assembly instructions when the design has changed.

After discussing related work (Sec. 2), we define the problem and scope of this research (Sec. 3). Our work is inspired by the Fabulaser Mini Assembly Manual (Sec. 4) that forms the basis for our generated assembly manuals. Section 5 discusses our contributions:

- a documentation plugin for CAD,
- a compiler for our documentation language, and
- the methodology that our software exposes.

In Sec. 6 we evaluate our system in various ways and conclude (Sec. 7) that our system can reproduce the Fabulaser Mini Assembly Manual to a high degree, except for wiring and electronics that we leave to future work. We show that changes in design have a localized effect on the documentation sources, therefore supporting design evolution well.

2 RELATED WORK

Open Source Hardware is an emerging field following the footsteps of Open Source Software. Although there are challenges, the field has high potential, especially with regards to communities embracing OSH in various ways [19]. This work focuses on a technical challenge around documentation, a crucial aspect of OSH.

The importance of documentation is well recognized in the context of Open Source Hardware projects. Ackermann, in his discussion on one of the first OSH licenses, makes an important distinction between (1) the physical product that is a result of applying documentation; and (2) the documentation of an OSH project [7]. In other words, documentation is a crucial aspect of OSH projects and many consider not only CAD files as documentation but also schematic diagrams, circuit or circuit board layouts, mechanical drawings, build instructions, design rationale, bill of materials, flow charts, operation instructions, maintenance instructions, repair and recycling manuals to be documentation [7, 8, 20, 9].

This information is deemed important by many as exemplified by J. Simmons: "What we really need to do is remind ourselves of just how important documentation is. We need to remember that documenting our hardware designs is our mission as open source hardware developers." [8, p. 62]. Bonvoisin et al. state that documentation is a crucial aspect of open

source hardware and raises the question of whether open source hardware is still open source if poorly documented [9]. The most common reasons for high-quality documentation are being able to co-develop and replicate the hardware [9, 21, 10, 22, 11].

Although documenting hardware is considered valuable for sharing and community participation [11], it is deemed to be time-intensive [12] and challenging: the documentation process can conflict with the design process [11] and designers are confronted with questions such as whether to document the process or outcome and whether to include failed attempts or not [12, 11].

The benefit of graphic information is widely recognized [8, 12] and often documentation consists of graphical information combined with text. However, creating such documents is considered an elaborate process with many distinct processes, such as going through pictures, selecting them, editing them, and combining them with written text [11]. The difficulty of taking proper images and editing them is well recognized [12, 11].

In the context of the maker and DIY movement, Milara et al. make a distinction between "document-after-the-fact" and "document-while-doing" [12]. After-the-fact documentation has a risk of leaving out important steps and settings, missing information about unsuccessful steps, and missing design argumentation information [12, 23, 11]. Document-while-doing has as major problem switching contexts between designing / problem solving and documenting. Tseng et al. suggest to find ways to seamlessly integrate designing and documenting [11], something that our work aspires to.

Several related solutions have preceded our work. Dalsgaard and Halskov present a web-based tool for capturing the rationale in a design process, enabling reflection in hardware projects [13]. The tool captures notes and events in a timeline that is shared among participants.

A similar tool is "Build In Progress" [14], an online collaborative documentation tool. Individuals or groups can document their progress on building hardware. Instead of using a timeline, the process of building is visualized in a tree-like structure with color codes. Non-working ideas are color coded red and successful attempts color coded green. A node in the tree contains pictures, video, and/or text and there is possibility for others to comment on design decisions.

DoDoc is a system to document hands-on-activities [15]. It consists of a webcam, microphone, lights, a device with three buttons, and a web-based interface. It can record video, audio, pictures, and animations. The authors stress that there are two important moments: during the design, so while designing and afterwards for reflection.

The above systems differ in the following ways from our system. Firstly, they focus on documenting hardware in general, whereas we only focus on assembly instructions. Although these systems focus on document-while-doing as is a goal for us, the design process and documentation process are not "technically integrated". This means that the documentation relies on a separate process besides designing, such as pushing buttons, taking pictures, writing text. Our system is directly technically related to the CAD files. This allows designers to refer to parts in the CAD file, with the aim of creating an integrated design and documentation process.

There are several solutions specifically for assembly instructions. A well-known documentation system is the LEGO brick model creation software. These systems cannot handle generic CAD files and focus on documentation of brick assembly designs. They are similar to our approach in terms of the abstraction of assembly steps as visual representations of assemblies and sub-assemblies. The LEGO software systems typically use either a step-based approach, like our approach, or a timeline-based approach [24, 25, 26, 27, 28, 29].

A more generic documentation system for assembly instructions for Open Source Hardware is the Open Hardware Assembly Instruction Kit or OHAI-Kit by Open Source Hardware 3D manufacturer Lulzbot [30]. Creating documentation is done by means of project sets. A project set describes building a complete machine and consists of multiple projects where a project is a distinct task such as setting up a frame or doing the final assembly. A project is set up in terms of work steps where a work step describes an action accompanied by pictures. The system provides a web interface to create the instructions. The system has a strong separation between the design and the documentation: the documentation is stored in an internal database and is not part of the open source hardware project itself.

DocuBricks is another solution specifically for Open Source Hardware and defines assembly instructions in a standard format [16]. The building blocks are hierarchically defined as "bricks" and "parts". A brick can capture "assembly instructions" with "steps" with descriptions and media. It is required to use a dedicated Brick editor and the documentation is stored in an XML file referencing media such as pictures and video. Limitations of this tool are that the process is not integrated with the development of OSH, and versioning of the documentation such as with Git [18] is challenging with XML files.

The current state-of-the-art in Open Source Hardware documentation for assembly instructions GitBuilding [17] improves on Git support by using text files instead of XML. GitBuilding is used and developed in the context of the open source OpenFlexure microscope [31] - a high-quality, low-cost microscope built from mainly 3D-printed parts. GitBuilding defines a language for documentation called BuildUp based on the popular markup language MarkDown and adds various special types of links. The main goal is to automate generation of a Bill of Materials (BOM) from the documentation. Other design goals are having a simple text format that can be (1) edited in a text editor and (2) easily integrated in Git. From the specifications in BuildUp a website can be generated with links between parts, steps, and BOMs. The main form of documentation is natural language instructions with images or video. The OpenFlexure team has moved away from taking pictures by instead using renders from CAD files because images tend to become outdated, while renders can be regenerated on changes [32].

Our work shares many of the same goals such as a simple text format, good Git support, and automatically generating BOMs. However, it differs in crucial aspects: Our tool has a strong semantic relation between the items in the documentation and the CAD files, whereas the links and parts in GitBuilding have no relation to the original CAD file. This may limit the scalability of GitBuilding, particularly for large assemblies where the lack of relation between documentation and the design makes it difficult to keep the documentation consistent with the design as project complexity grows. Another difference is that GitBuilding focuses on natural language descriptions in instruction manuals, whereas we aim for documentation with a focus on high-quality visual instructions, optionally supplemented by small natural language remarks. We explain the reason for this different focus in the following section.

Finally, our work is inspired by the software world where documentation such as APIs (Application Programming Interface) is often generated from comments in source files. The problem to maintain consistency between source code and natural language descriptions has led to *literate programming* [33], where source code and natural language descriptions are close together in the source file. Similar to this goal, our system is focused on achieving consistency between CAD designs and visual documentation. Modern instances of literate programming are Org-mode [34] that is very flexible and programming language agnostic and Scribble [35] in which there is hardly separation between the markup language and programming language. Because of this flexibility, our system uses Scribble for the textual representation and our system.

3 PROBLEM DESCRIPTION

As we have seen in the previous section, documentation is a crucial aspect for Open Source Hardware for two separate goals: (1) replication of the hardware and (2) co-development of the hardware. Devising the documentation is a time-intensive and elaborate process where graphical information, often obtained by taking pictures, is deemed an important but also a time-intensive aspect.

In general, two approaches are recognized: The *document-after-the-fact* approach may work for traditional hardware products, while *document-while-doing* is a clear favorite for OSH, since an important goal is that the hardware is co-developed during its lifetime by potentially many persons distributed all over the globe.

The co-development process is also why versioning systems such as Git are valuable in Open Source Hardware and Software. These systems allow to record changes over time, explaining the design evolution. Because versioning systems excel in highlighting changes in text files, these types of files are typically preferred over binary files that rely on third-party tools to highlight changes. For example, viewing two different versions of a CAD file in a CAD program

requires native support for this in the CAD program, whereas for text files there is no need for such third-party tools.

Another related problem to versioning is the central question of what the source exactly constitutes of. In general, one would want to make the source footprint as small as possible and to derive information from the source. In terms of documentation of OSH, it is common that the source footprint is large with the design files being a source of truth and the documentation another source of truth which may lead to a disconnect between these sources of truth. Ideally, we would want to keep the source footprint small, generating the documentation as much as possible from the source, preventing a disconnect between the design and the documentation. This is related to the problem of separation between the design and documentation process that many systems show.

For example, in GitBuilding, the build instructions in natural language form should match the design in the CAD files, but if the CAD design changes, the natural language instructions do not automatically update with these changes and may become outdated. The problem is that besides the CAD files, the documentation is a second source of truth and it is the hardware designer's task to keep these sources matched up. Ideally, you would want to reduce the sources of truth and let the documentation be as much as possible dependent on the CAD design.

A special case of the different sources of truth are taking photographs of the assembly. This is not only a time-intensive process, but it is particularly prone to a disconnect between the evolving design and the picture that has been taken. Each time the design changes, a new picture has to be taken and it is the hardware designer's responsibility to keep these pictures up-to-date with the design.

3.1 SCOPE

Given the problem description above, we aim for a system that given CAD source files of an OSH project, semi-automatically generates high-quality visual documentation and BOMs while minimizing natural language instructions to reduce the potential for a disconnect between design and documentation. This documentation should have a strong semantic relation with the CAD files, thereby reducing the time and effort to create said documentation supporting co-development in which the design evolves. In addition, the documentation source files should be text files to allow efficient tracking of changes in the documentation.

Although many different types of documentation exist and are important for OSH, we focus mainly on assembly instructions in this work, motivated by Bonvoisin et al. who identify three main criteria to assess accessibility and replicability [9]:

- CAD files availability and editability,
- Assembly instructions availability and editability, and
- Bill of Materials availability and editability.

In our approach we assume that we have the CAD design of the physical artifact available in a compatible file format. To be generic to the specific CAD implementation, we expect a CAD file with the following properties:

- Each part or tool that should be named or visualized in the assembly instructions is available in the CAD file.
- Each part has a distinct identifier and can be used as source in a Bill of Materials.

From this information, our software tools assist in creating high-quality visual assembly instructions.

4 USE-CASE: FABULASER MINI ASSEMBLY MANUAL

To evaluate our work, we will compare our solution to manually created assembly instructions for the open source laser cutter Fabulaser Mini [36]. This machine is part of the Open Lab Starter Kit project that aims to design open source machines for Fab Labs. A Fab Lab is a maker space with several machines such as laser cutters and 3D printers to fabricate products [37].

The assembly instructions for the laser cutter [38] were created through a collaboration between the main designer, a CAD professional and a graphics designer. This was an elaborate and manual process summarized below:

To break down the assembly process in logical steps, the main designer, CAD professional, and graphics designer maintained a shared document, the manual workbook, in which they recorded the name, the parts, tools, and remarks associated with each step. The graphic designer designed a layout to provide clear instructions. The page is A4 landscape with in the heading the step number and a descriptive title prominently on the page, and then a subheading with the progress in terms of steps and a time indicator.

Figure 1 shows the layout of one of the steps in the Fabulaser Mini manual. The footer contains pictograms of the required tools and extra information. The body has several columns where the first column is fixed and always used for the pictograms of the required parts. The rest of the columns is free to use and the most important is the depiction of the component to assemble. Remarks are in principle in column 2 if space permits. Otherwise, they can be placed in other columns if needed. Zoomed highlights are placed freely depending on where it is logical and instructive.

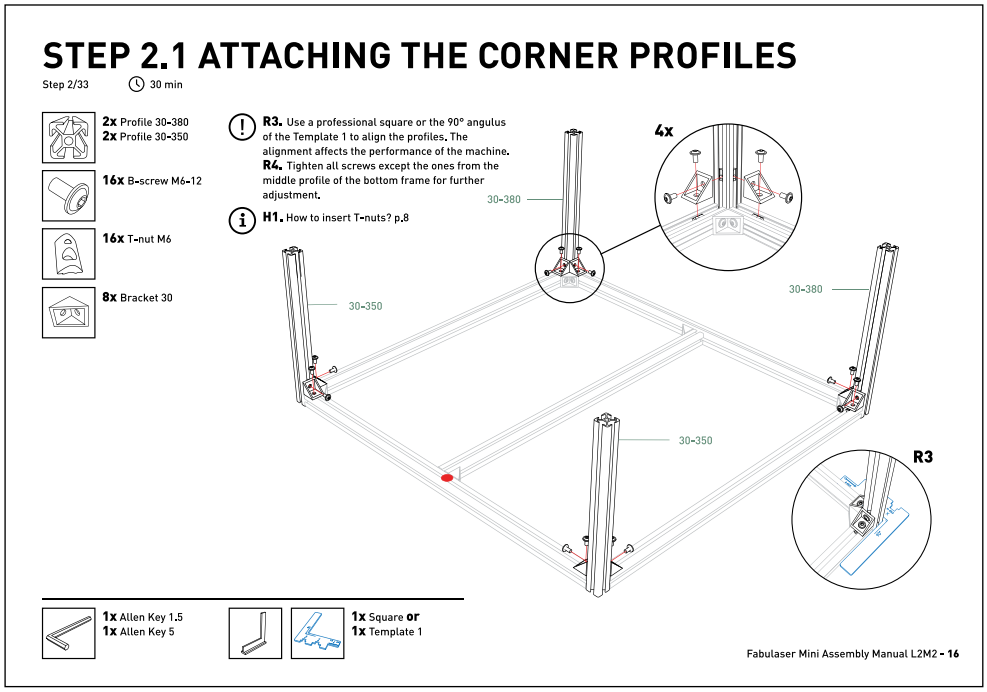


Figure 1 A page of a step in the Fabulaser Mini manual.

The CAD professional imports the CAD file from the main designer and organizes the views of the various steps for the assembly instructions, for the main images and the zoomed highlights. The visibility of the parts in an assembly step is improved by showing a subset of the machine to build and graying out parts that are important for context. For example, in Figure 1 the profiles on the floor are grayed out and serve as context for the four profiles that need to be attached. The zoomed highlight also shows that screws and other parts are selectively moved (partially exploded) with a red line showing how to insert the screws.

The viewing angle and general graphical representation are evaluated by all three collaborators and when they deem the view satisfying, the CAD professional exports the main image, the zoomed highlights, and the pictograms of the parts and tools. This information is handed to the graphic designer that lays out these elements on the page and creates the column with the used parts, counting how many time each part is used. In Figure 1, note that the left column is essentially a Bill of Materials for this step.

5 APPROACH

To automate the process of generating assembly instructions in the style of the Fabulaser Mini manual, we developed two software packages that interact closely. The OSH AutoDoc

Workbench [39] is a plugin for FreeCAD [40]. The OSH AutoDoc PDF package [41] is a compiler that takes as input the output of the OSH AutoDoc Workbench and a textual specification. The compiler combines the CAD information and the textual specification and from this generates an assembly manual in the form of a PDF file. We will first discuss the workbench, then the compiler and the textual specification, and finally the methodology that arises from using the software. All software and the examples are available under open source licenses [39, 40, 41, 42].

5.1 THE OSH AUTODOC WORKBENCH

Given a CAD file with a design, the workbench [39] reorganizes the information for assembly instruction purposes using links to the original design. The documentation information can be stored in a separate *documentation CAD file* that links to the *design CAD file*, but this is not strictly necessary.

The documentation CAD file contains a layer for each assembly step. To populate these *step layers*, the workbench has special functionality to select parts and place them in the right step layer. As soon as a part is put into a step layer, the part is hidden to uncover other parts in the design.

The workbench also makes use of *position layers*. Users can select parts and reposition them for documentation purposes, essentially creating a partial explosion. Examples are re-positioned screws that indicate how to insert them with an automatically generated red line (see Figure 1 the top-right highlight). We can automatically reposition screws by inferring the direction of the screw by subtracting the center of the bounding box from the center of gravity of the screw.

The central abstraction that the workbench provides is the *layer state* and the *layer state manager*. A *layer state* defines which layers are visible or invisible and which layers are the original color or grayed out (see Figure 2). Position layers reposition screws or other parts based on visibility of the layer. Layer states also preserve a camera position from an angle or perspective that the user chooses and as such, the layer states are well equipped to define the views for a step or a highlight in the assembly instructions.

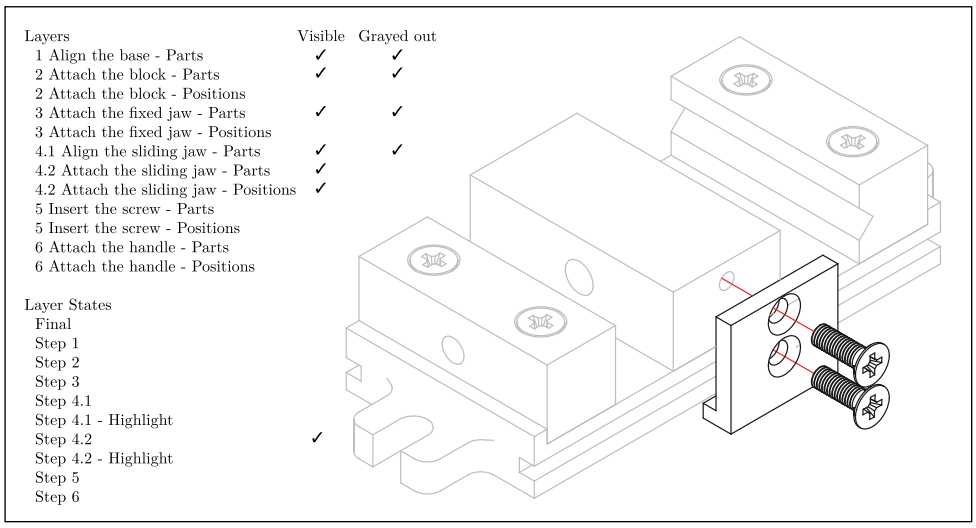


Figure 2 Activated layer state Step 4.2 makes several layers visible and grayed out.

Figure 3 shows the relation between *layer states* and the layers. A *layer state* captures *part layers*, *position layers* and a *camera position*. This information is used to export a high-quality vector image. A *layer state* also stores which layers are visible and in case of *part layers* whether the original layer is used or a grayed out version. A part layer contains the parts for an assembly step. The software can then automatically generate: (1) images for each of the parts in one go, and (2) a CSV table with the necessary information to create a mini-BOM for each of the assembly steps.

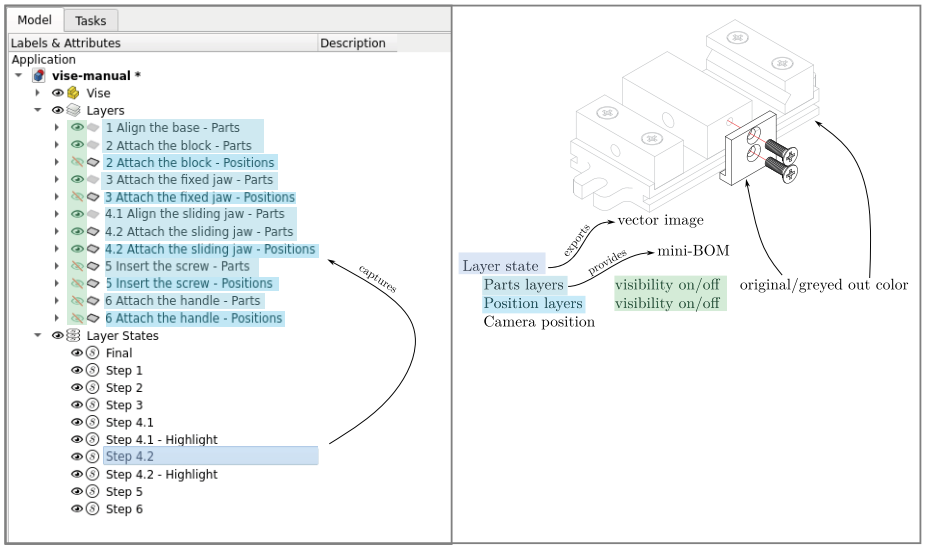


Figure 3 The relation between layer states and the layers.

5.2 THE OSH AUTODOC COMPILER

The compiler [41] parses the textual specification, using a subset of the Scribble markup language. Scribble [35] is a Domain-Specific Language (DSL) for documentation and allows you to write text with generic tags that a user can define. For our work, we defined tags that convey information for the assembly manual. Some of these tags refer directly to the abstractions in the *documentation CAD file* such as `@image` or `@highlight` that refer to a specific *layer state* or `@minibom` that refers to a *part layer*.

Figures 4 and 5 show how the Scribble statements translate to the generated page. The red underlined strings directly relate to the *documentation CAD file*, referring to a *part layer* (`@minibom`) or *layer states* (`@image, @highlight, @tool`). Note that the `@howto` statement refers to the first `@howto-item` that is defined elsewhere in the textual specification. Figures 6 and 7 show the definition and the generated page to which is referred. The compiler automatically incorporates the question and the page reference.

```
@step{
  @substep{
    @title{Attaching the Corner Profiles}
    @duration{30 mins}
    @minibom{Step 2.1 - Parts}
    @image{Step 2.1}
    @label[:from-pos (90 160) :to-pos (125 164)]{30-350}
    @label[:from-pos (325 275) :to-pos (370 279)]{30-350}
    @label[:from-pos (275 80) :to-pos (200 84)]{30-380}
    @label[:from-pos (510 150) :to-pos (435 154)]{30-380}
    @highlight[:from-pos (280 120 30)
               :to-pos (400 50 60)
               :label-top-left "4x"]{Step 2.1 - Detail 1 001}
    @highlight[:to-pos (525 325 50)
               :label-top-right "R3"]{Step 2.1 - Detail 2 001}

    @remark["column2-4"]{Use a professional square or the 90°
     angle of Template 1 to align the profiles. The
     alignment affects the performance of the machine.}
    @remark["column2-4"]{Tighten all screws except the ones
     from the middle profile of the bottom frame for further
     adjustment.}
    @howto["column2-4"]{1}
    @tool["1.5"]{allen key}
    @tool["5"]{allen key}
    @tool{template 1}
  }
}
```

Figure 4 The textual specification in Scribble for Figure 5 to replicate Figure 1.

DOI: 10.5334/joh.56

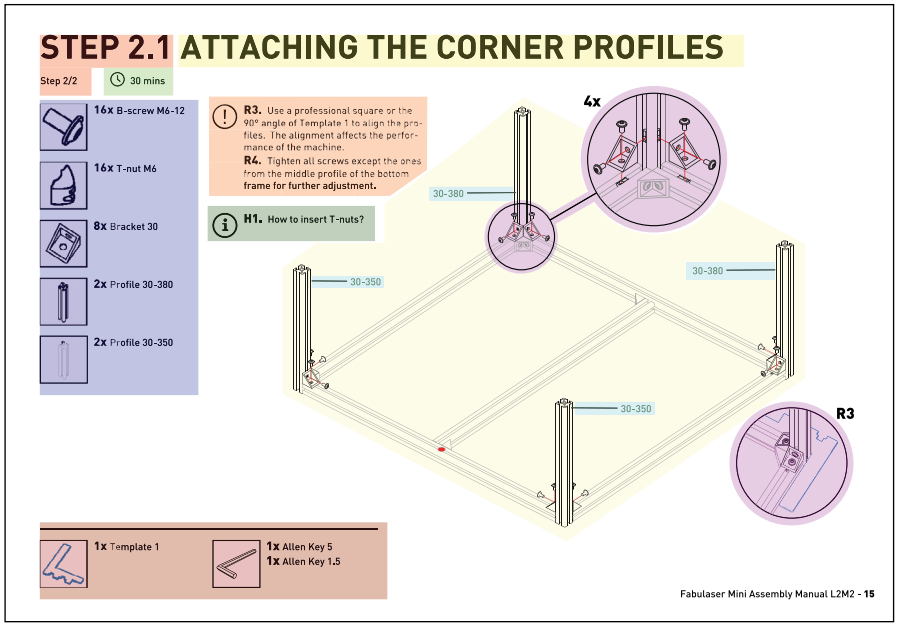


Figure 5 The generated assembly step based on the textual description in Figure 4, replicating Figure 1.

```
@howto-item{
  @title["How to insert T-nuts?"]{How to insert a T-nut in a profile}
  @text["column1-3"]{
    T-nuts need to be inserted in the profiles to enable the screws
    to be fixed. The correct position of the T-nut is with the ball
    pointing towards the center of the profile.

    There are two different ways of inserting a T-nut into a profile.
  }
  @image["column1-3"]{insert-t-nut}

  @text["column2-3"]{
    @heading{Option 1}
    If the profile is not yet blocked at the ends, the T-nut can be
    inserted from the side of the profile, by sliding it in the slit
    in the correct position.
  }
  @image["column2-3"]{insert-t-nut-slide}

  @text["column3-3"]{
    @heading{Option 2}
    Insert the T-nut sideways in the profile slit. Using a small
    screwdriver or an allen key, stick the tool in the T-nut hole
    and turn it to the correct position.
  }
  @image["column3-3"]{insert-t-nut-allen-key}
}
```

Figure 6 The definition of the first howto item shown in Figure 7.

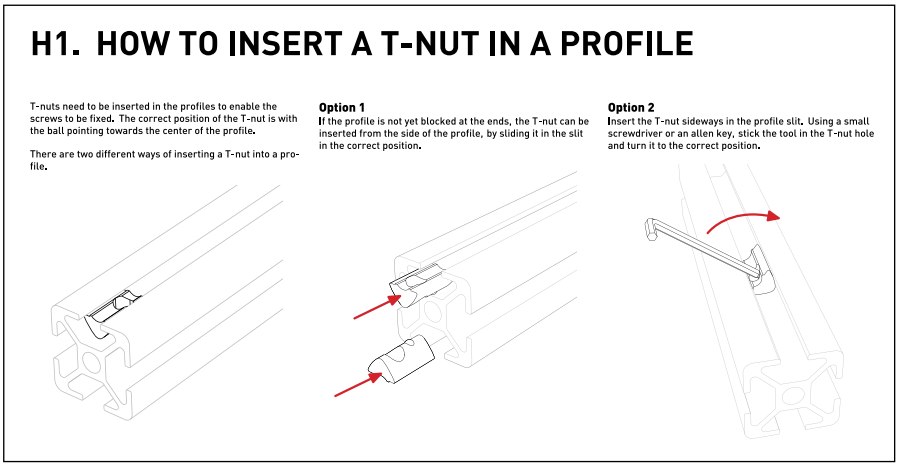


Figure 7 The generated howto page of Figure 6.

5.3 METHODOLOGY

The two software packages with their functionality and abstractions give rise to a methodology for documentation that we call "CAD-coupled documentation". Part of the process is performed in CAD software with links between the *design CAD file* and the *documentation CAD file* to support a high degree of integration between the design and documentation process. The manual itself is specified in the Scribble markup file that is concerned with the semantics and layout of the assembly instructions. Table 1 lists the various phases in the process and in which software packages they take place.

Table 1 Phases in the methodology.

PHASE	DESCRIPTION	SOFTWARE
1) Inspect Design	Determine logical steps in assembly	Workbench
2) Select Parts	Intuitive hiding and selection of parts	Workbench
3) Position Parts	Selective explosion of parts	Workbench
4) Define Layer States	Define assembly steps and highlights	Workbench
5) Set Camera positions	Save viewpoint angle and perspective	Workbench
6) Generate Images	Mini-BOM and step vector images	Workbench
7) Write Textual Specification	Fill the Scribble template	Compiler
8) Generate Assembly Instructions	Generate a PDF assembly manual	Compiler

Firstly, it is necessary to determine effective and logical assembly (sub)steps (phase 1). This cannot be automated but the workbench can help to organize the parts in part layers in the second phase. In phase 3, the workbench helps to reposition parts (partially exploded view) for clarifying the assembly process. The user then defines steps and highlights in terms of layer states that make selected *part* or *position* layers visible (phase 4). In the layer states the user can save the viewpoint of the camera from a simple context menu (phase 5) and automatically generate images and mini-BOMs (phase 6). The two last phases apply to the compiler and consist of declaring the Scribble statements for the assembly instructions and generating the manual. Although the phases are presented in a sequential fashion, at any time, it is possible to refine the assembly instructions in any phase, for example repositioning parts, after which the manual can be generated again. We published videos with an overview of the functionality [43] and the complete methodology applied to a small vise assembly [44]. We also provide a virtual machine with all the software pre-installed with instructions to replicate the videos [45] and a container image with FreeCAD and the workbench [46].

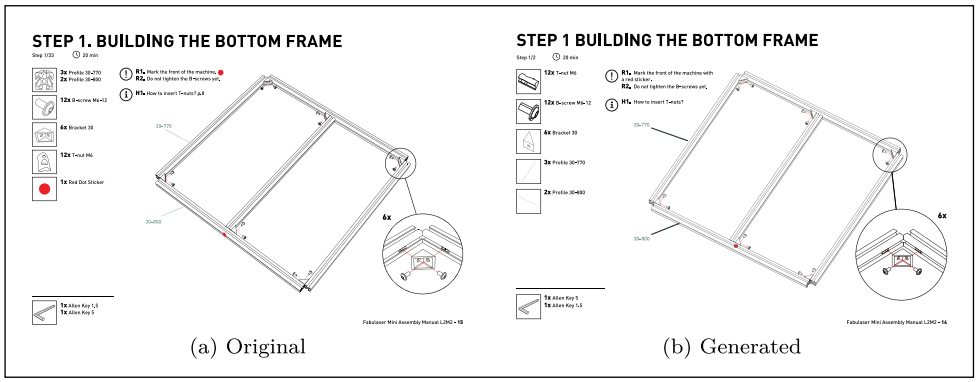
6 EVALUATION

We use several methods to evaluate our work. Firstly, we show to what extent we can replicate the original Fabulaser Mini Assembly Manual (Sec. 6.1). Secondly, we highlight each aspect of the problem description in Sec. 3 and show to what extent our approach solves them (Sec. 6.2–6.4).

6.1 REPLICATING THE FABULASER MINI ASSEMBLY MANUAL

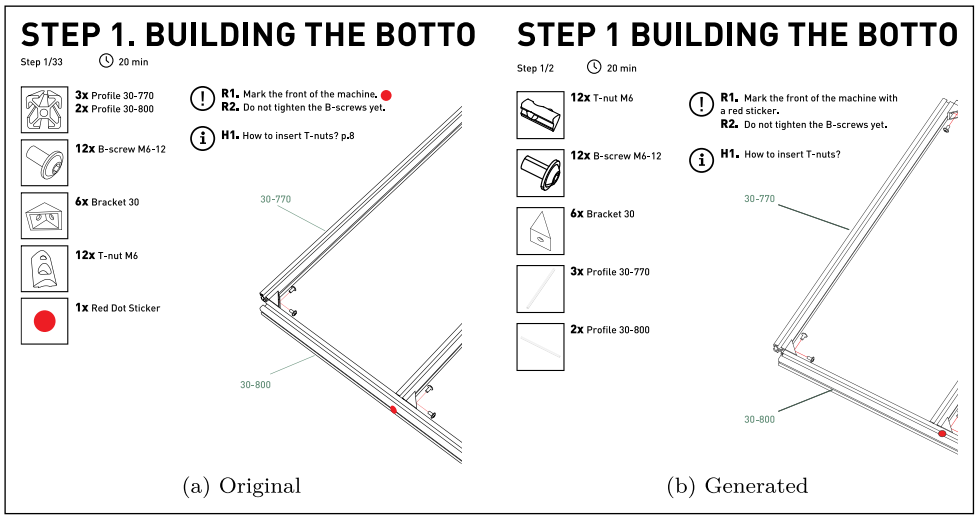
To show to what extent we can replicate the original Fabulaser Mini Assembly Manual, we recreate two of the steps and show the differences in Figures 8 to 10. In Figure 8 we can see that the two versions are virtually the same, but that there are small differences, as can be seen in subsequent figures: Figure 9 shows a step in the assembly manual, whereas Figure 10 a sub-step. In the original versions on the left side, the title text has a period character before the title description only for steps but not for sub-steps, whereas the generated versions on the right side consistently leaves out the period character for both steps and sub-steps. Since we generate two steps in this example, the total step count is different from the original manual. The mini-BOM shows the various components arguably clearer in the original version, but in the generated version, users can obtain a sense of size of the component because of thinner lines and additionally, the angle of the component's picture matches the assembly step for easier

identification of parts. Finally, in the generated version, the red sticker is not listed as a part and is mentioned in a remark instead. However, in contrast to the original assembly instructions, the red sticker is part of the CAD model in the generated version and therefore, its placement is consistent over steps, whereas in the original instructions, the red sticker is manually placed in the page, sometimes with inconsistent results.



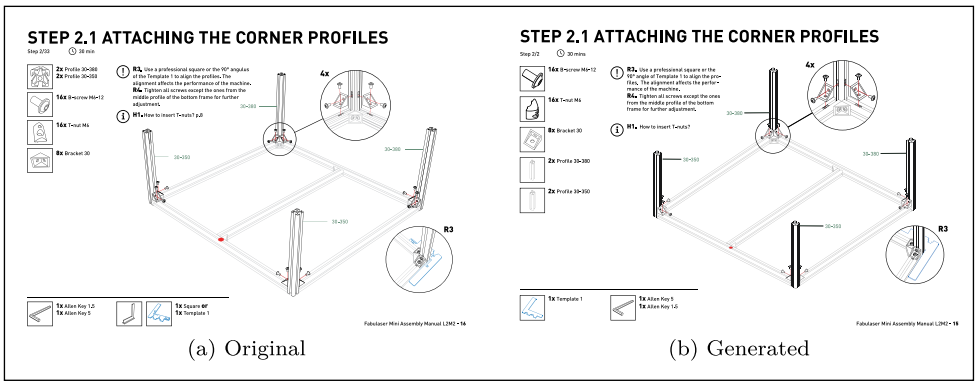
(a) Original (b) Generated

Figure 8 Difference between original and generated Step 1.



(a) Original (b) Generated

Figure 9 Difference between original and generated Step 1, zoomed in.



(a) Original (b) Generated

Figure 10 Difference between original and generated Step 2.1.

Figure 10 shows two virtually similar pages but it also highlights a drawback of automation. In the bottom left corner, the original page specifies alternatives for the use of the template or a square with two images, whereas this flexibility is challenging to achieve in our current version. We therefore only included the use of the template and leave out the square. In principle, we could implement this functionality but then questions arise whether we support two alternative equivalent tools or even more, whereas in the non-automated setting, the graphic designer has the flexibility to decide this and simply place two or more alternatives.

Table 2 presents supported features. By supported we mean that a particular layout from the original Fabulaser Mini assembly manual can be replicated. The column "Replicated" shows the parts of the original instructions that we replicated for the evaluation. The global list of

tools and the full BOM of parts are currently not supported. However, since we can already list subsets of tools and parts, a page with the full BOM of parts and tools would be a minor change in the software. In essence, we can conclude that we support all major features of the original Fabulaser Mini assembly manual.

Table 2 List of support of assembly manual features.

MANUAL FEATURE	DESCRIPTION	SUPPORTED	REPLICATED
Title-page	Page with full assembly	Yes	Yes
Index	Table of contents	Yes	Yes
How to use manual	Instructions on manual usage	Yes	No
List of tools	A list of tools to use	No	No
How-to pages	Instructions for common tasks	Yes	Yes
Safety instructions	Generic safety instructions	Yes	No
Duration time	Duration pictogram for a step	Yes	Yes
Multi-column typesetting	Typeset remarks per column	Yes	Yes
Mini-BOM	Mini-BOM with pictograms	Yes	Yes
Remarks	Step-specific remarks	Yes	Yes
Highlights	Zoomed images for details	Yes	Yes
Labels	Labels to certain parts	Yes	Yes
How-to remarks	Refer to how-to pages	Yes	Yes
Tools for each step	List of tools for a step	Yes	Yes
List of Parts	Full BOM of parts	No	No

Since we have not recreated each step of the original assembly instructions, we analyzed all the step layouts used in the original manual. This allows us to verify to what extent we support a given page layout, where supporting means that we can replicate a page with this layout. Table 3 shows that the majority of the pages are layout S1, that we fully support, albeit that we do not support arbitrary tool combinations of alternatives as is mentioned before (approximately 25% of S1). A small number of step layouts (S2) use a stand-alone, secondary image which would be a minor change in our software. About 10% of the steps use electronics and wiring. This would require a major change in our software that we leave to future work, potentially in combination with an EDA tool such as KiCad [47]. A small percentage uses S4 with a stacked highlight that would require a minor change in our software. Finally, step layout S5 mixes text with images. Supporting this would require a minor change to the software as well.

Table 3 Replicability of step layouts. [1]25% of these pages use tool alternatives that we do not support. [2]Support for this requires a minor addition to our software. [3]Support for this requires a major change to our software.

STEP LAYOUT	DESCRIPTION	EXAMPLE PAGES	% OF PAGES	SUPPORTED
S1	Layout of Figure 8 and 10	15, 16, 17	79.7	Yes[1]
S2	S1 with stand-alone image	20, 70, 73	3.3	No[2]
S3	S1 with electronics/wiring	24, 25, 26	10.1	No[3]
S4	S1 with stacked highlight	37, 40, 83	4.5	No[2]
S5	S1 with text/image columns	84, 85, 86	12.4	No[2]

6.2 ELABORATE DOCUMENTATION PROCESS

Section 3 lists several aspects of documenting hardware. The first aspect is that documentation is an elaborate and time-intensive process. As explained in Sec. 5, our solution specifically aims for integration between the design and documentation process by means of automation in what we call "CAD-coupled documentation".

By capturing the layout of the Fabulaser Mini manual in layout rules in our software, we automate the role of the graphic designer to a high degree. However, our approach still provides

flexibility in layout decisions by providing hints for placing highlights, labels, and remarks into columns or into a Cartesian coordinate system.

Table 1 lists several steps of our methodology, but the last step "Generate Assembly Instructions" leaves out many steps of the automation that we list in Table 4. Instead of focusing on these kinds of details, users can focus on the semantics, so determining what a step is, how long it takes to complete the step, or making assembly more clear by adding a highlight. Essentially, users can focus on filling a predefined Scribble template per step with the relevant information for their assembly process.

Table 4 Automatic functionality for generating assembly instructions.

AUTOMATIC FUNCTIONALITY
Cross-reference page references
Cross-reference howto references
Number steps
Layout page components
Build the title page
Build the index
Build the howto pages
Generate the mini-BOMs

Another strength of this approach is that it is a software solution that eliminates the need for taking pictures with a camera or physically disassembling the machine to understand steps or obtain clearer images. Additionally, the tight integration between the documentation and design CAD files assure that there is only one source of truth which is the collection of CAD files.

Quantifying the time savings caused by our approach is challenging because we cannot reconstruct how much time the creation of the original Fabulaser Mini manual took because of various iterations with multiple people. In addition, to fully compare the time durations, we would have to include electronics and wiring which we leave to future work. However, to give an indication of how time-intensive our approach is, we timed the creation of assembly instructions for two small projects that show that we can generate a manual of about 5 steps in the order of 1 hour. Table 5 lists the machines, the number of steps, and the amount of minutes it took to complete the manual. A video that times the steps of creating the Vise assembly is available [44].

Table 5 Example designs for evaluating time duration.

MACHINE	NR. STEPS	TIME TO COMPLETE
Vise	6	24 minutes
Vertical Lathe	5	26 minutes

6.3 SUPPORT FOR GIT AND CO-DEVELOPMENT

Another aspect listed in the problem description that is important in OSH is the support for Git as a versioning system, one of the main reasons for the current state-of-the-art documentation system GitBuilding [17]. Our approach is friendly to versioning because the specification of assembly instructions is a line-based text file.

However, since we interact with a CAD file, the support for Git for the overall system depends on the support that the CAD file has in Git. For our specific implementation in FreeCAD, support is limited for CAD files in Git, but there is the possibility to get textual summaries of differences in the binary FreeCAD files.

Our software does not incorporate co-development features like collaborative interaction, as those found in cloud-based CAD solutions. This presents a future research challenge. Instead, we rely on existing co-development tools, where assembly manual designers, create a local copies of file repositories and collaborate with existing tools compatible with Git.

6.4 DESIGN EVOLUTION CONSIDERATIONS

Although our software is usable for non-open hardware projects, the main reason for this level of automation is evolution of designs through collaboration that is typical for OSH. It is challenging to anticipate all possible design changes and the effects on the documentation process. Therefore, instead, we analyze what interactions are required for evolving designs and enumerate the interactions with our software.

For this analysis we have the following assumptions: We make use of a "design CAD file" containing a design for version 1 of some tool or machine. Then we have the "documentation CAD file", also in version 1, with the abstractions from our methodology and links into the *design CAD file*. Finally, there is the "textual specification", also in version 1. The *documentation CAD file* has layers, positions, and layer states that are used in the *textual specification* that links to the documentation CAD file and from which we can generate the manual of version 1.

Now as part of the evolution of the design, the *design CAD file* is changed into versions 2, 3, and 4 that potentially require changes in the *documentation CAD file* and the *textual specification*. The design evolution workflow that we describe below is illustrated in Figure 11. The actions for the documentation CAD file and the textual specification are listed in Table 6.

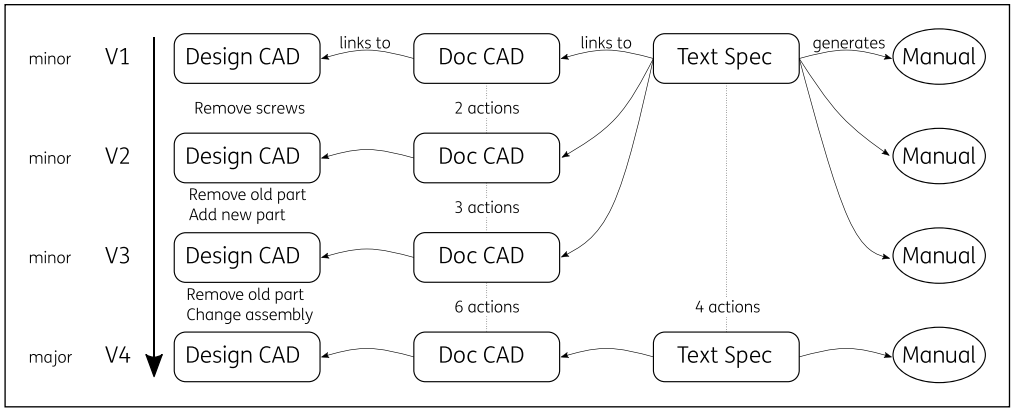


Figure 11 Design evolution workflow for different scenarios. Table 6 lists the actions required for each version.

VERSION	ACTIONS DOCUMENTATION CAD FILE	ACTIONS TEXTUAL SPECIFICATION
v2	Remove broken links (1)	
	[Remove positions (2)]	
v3	Update broken link to point to new part (1)	
	[Redo positions (2)]	
	Update camera position (5)	
v4	Update broken link to point to new part (1)	Make new step definition
	Add links to new part layer (1)	Refer to part layer for BOM
	[Redo positions (2)]	Refer to layer state for image
	Turn layers on/off on new layer state (3)	Determine coordinates image
	Gray out layers (4)	
	Establish camera positions (5)	

Table 6 Actions on the documentation CAD file and the textual specification for the versions 2, 3, and 4 for Figure 11. The actions in square brackets [] are optional depending on whether positions were created for these parts.

The effects of changing the *design CAD file* from a version to a higher one are always limited in scope for the *documentation CAD file* because of the low number abstractions in this latter file. The *documentation CAD file* consists of two types of layers: (1) layers for parts and (2) layers for positions. The layer states record which layers are (3) on or off or (4) grayed out for steps and highlights. Finally, the layer states define camera positions (5).

Consider the following changes in the design: we simplify the design to use less screws in version 2 (see Figure 11 and Table 6). Our system supports this design change in the following

way: The excess screws will be apparent in the documentation CAD file because of broken links to the design CAD file. After removing the broken links from the part layer (1) and potentially removing corresponding positions (2), the user can automatically generate the manual again to make it up-to-date.

A more complicated change would be to replace a part with a new one with different dimensions in version 3. In our system, the old part results in a broken link in the documentation CAD file. After removing the old part from the part layer (1) and adding the new part in the same layer (1), potentially redoing positions (2), the user needs to update camera positions for the layer states that feature the replaced part (5) because of changed dimensions. The user can then regenerate the manual to make it up-to-date.

The most convoluted design change would be a change that requires changes in the assembly steps, for example splitting a step or combining a step. An example is replacing a part with one that requires more assembly and explanation, needing an additional step in version 4. This would require a new part layer, moving parts from other part layers (1), and potentially redo positions (2). Additionally, we would require creating a new layer state, turning on or off layers (3), graying out layers (4), and esatblishing a camera position (5).

For version 2 and 3 it is not required to make any changes in the textual specification, but version 4 does require changes in the *textual specification*. It involves creating a new step definition in the *textual specification* (that can conveniently be adapted from a previously defined step). In this step definition, the user should refer to the new part layer and layer state in the documentation CAD file. With this information, the user can automatically generate the new mini-BOM and incorporate a new image for this step with the camera angles that the user defined in the layer state in the documentation CAD file. However, even for this most convoluted example, the actions to perform are limited in scope and after the changes, the documentation can be generated again in one go.

7 CONCLUSION

Open Source Hardware (OSH) projects have become more prominent and adoption is growing. High-quality documentation is crucial for OSH to facilitate the replication and co-development. Current state-of-the-art documentation systems offer limited support for two issues: (1) how to ensure that documentation stays up-to-date with constantly evolving designs, and (2) how to integrate the design and documentation processes.

We addressed these issues with our system that consists of a CAD plugin for documentation and a compiler for a textual specification. This allows specifying assembly manuals with a strong semantic relation between the documentation and the CAD design, and brings forward a *methodology* that we name “CAD-coupled documentation”.

We evaluate our system by replicating parts of a state-of-the-art assembly manual for OSH. We show that we can replicate this manual to a high degree and that we can automate significant parts of the documentation. Additionally, design evolution results in documentation changes that are limited in scope as a result of the abstractions our system provides.

Although our work shows high potential for integrating the design and documentation process, our work is currently limited to assembly instructions and in particular, our software cannot handle the electronics part of the manual that we aimed to replicate. For future work, we aim to apply our work to electronics and other forms of documentation that are important for OSH.

Another direction for future work is to extend the support for designs made with other CAD software than FreeCAD. Our system supports creating manuals from a STEP file acting as the design CAD file. However, design evolution with STEP files is not supported, because a change in alternative CAD software will result in a new STEP export and therefore a new design CAD file. Because of this, the documentation CAD file will not be able to maintain the links to the design CAD file. One potential direction is to record how the documentation CAD file was built from a STEP design CAD file. Then, given a new STEP file that captures a subsequent version of the design, we would apply the recording to recreate the documentation CAD file, notifying the user which recorded actions cannot be performed because of changes in the design CAD file.

In a greater context, integrating the design and documentation process and semi-automatically generating documentation allows to keep the source footprint of OSH small and allows the documentation to be of high quality and consistent over evolving designs. This can enable the highly valuable OSH communities to focus more on improving designs without distractions of elaborate documentation processes that require much discipline to keep up-to-date with the designs.

ACKNOWLEDGEMENT

We would like to thank Daniele Ingrassia, Marc Kohlen, and Liane Sayuri Honda for their collaboration and valuable insights.

FUNDING INFORMATION

This research was funded by the European Regional Development Fund (ERDF) in the context of the INTERFACER Project.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

J.C. Mariscal-Melgar orcid.org/0000-0002-2562-0316
Helmut-Schmidt University/University of the Federal Armed Forces, Hamburg, Germany
Pieter Hijma orcid.org/0000-0002-5716-1118
Hamburg Institute for Value Systematics and Knowledge Management, HIWW, Hamburg, Germany
Manuel Moritz orcid.org/0000-0001-5126-9016
Helmut-Schmidt University/University of the Federal Armed Forces, Hamburg, Germany
Tobias Redlich orcid.org/0000-0003-4129-8926
Helmut-Schmidt University/University of the Federal Armed Forces, Hamburg, Germany

REFERENCES

1. **Alison Powell.** "Democratizing production through open source knowledge: from open software to open hardware." In: *Media, Culture & Society* 34.6 (2012), pp. 691–708. DOI: https://doi.org/10.1177/0163443712449497
2. ***OSHWA Certified Projects List.*** Open Source Hardware Association. 2023. URL: https://certification.oshwa.org/list.html (visited on 01/17/2023).
3. **Massimo Banzi** and **Michael Shiloh.** *Getting started with Arduino.* Ed. by Patric Di Justo. Fourth Edition. Maker Media, Inc., 2022.
4. **Rhys Jones, Patrick Haufe, Edward Sells, Pejman Iravani, Vik Olliver, Chris Palmer,** and **Adrian Bowyer.** "RepRap the replicating rapid prototyper." In: *Robotica* 29.1 (2011), pp. 177–191. DOI: https://doi.org/10.1017/S026357471000069X
5. **CERN.** *CERN launches Open Hardware initiative.* 2023. URL: https://home.cern/news/press-release/cern/cern-launches-open-hardware-initiative (visited on 01/16/2023).
6. **Myriam Ayass** and **Javier Serrano.** "The CERN Open Hardware License." In: *International Free and Open Source Software Law Review* 4.1 (2012), pp. 71–78. DOI: https://doi.org/10.5033/ifosslr.v4i1.65
7. **John R. Ackermann.** "Toward Open Source Hardware." In: *University of Dayton Law Review* 34.2 (2009), pp. 183–222.
8. **Alicia Gibb.** *Building Open Source Hardware: DIY Manufacturing for Hackers and Makers.* Pearson Education, 2014.
9. **Jérémy Bonvoisin, Robert Mies, Jean-François Boujut,** and **Rainer Stark.** "What is the Source of Open Source Hardware?" In: *Journal of Open Hardware* 1.1 (2017), pp. 1–18. DOI: https://doi.org/10.5334/joh.7
10. **Nadica Miljkovi, Ana Trisovic,** and **Limor Peer.** "Towards FAIR Principles for Open Hardware." In: *CoRR* abs/2109.06045 (2021). DOI: https://doi.org/10.5281/zenodo.5524414
11. **Tiffany Tseng** and **Mitchel Resnick.** "Product versus Process: Representing and Appropriating DIY Projects Online." In: *Proceedings of the 2014 Conference on Designing Interactive Systems*. DIS '14. 2014, pp. 425–428. DOI: https://doi.org/10.1145/2598510.2598540

12. **Iván Sánchez Milara, Georgi V. Georgiev, Jani Ylioja, Onnur Özüduru,** and **Jukka Riekki.** ""Document-while-doing": a documentation tool for Fab Lab environments." In: *The Design Journal* 22.sup1 (2019), pp. 2019–2030. DOI: https://doi.org/10.1080/14606925.2019.1594926
13. **Peter Dalsgaard** and **Kim Halskov.** "Reflective Design Documentation." In: *Proceedings of the Designing Interactive Systems Conference*. DIS '12. 2012, pp. 428–437. DOI: https://doi.org/10.1145/2317956.2318020
14. **Tiffany Tseng.** "Build in Progress: Building Process-Oriented Documentation." In: *Makeology*. Routledge, 2016, pp. 237–254. DOI: https://doi.org/10.4324/9781315726519-16
15. **Pauline Gourlet, Sarah Garcin, Louis Eveillard,** and **Ferdinand Dervieux.** "DoDoc: A Composite Interface That Supports Reflection-in-Action." In: *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '16. 2016, pp. 316–323. DOI: https://doi.org/10.1145/2839462.2839506
16. **Tobias Wenzel.** *DocuBricks*. 2016. URL: http://www.docubricks.com/ (visited on 01/17/2023).
17. **Julian Stirling.** *GitBuilding*. 2021. URL: https://gitbuilding.io/ (visited on 01/17/2023).
18. **Scott Chacon** and **Ben Straub.** *Pro Git*. Ed. by Louise Corrigan. Second Edition. USA: Apress, 2014. DOI: https://doi.org/10.1007/978-1-4842-0076-6
19. **Manuel Moritz, Tobias Redlich,** and **Jens Wulfsberg.** "Best Practices and Pitfalls in Open Source Hardware." In: *Proceedings of the International Conference on Information Technology & Systems (ICITS 2018)*. 2018, pp. 200–210. DOI: https://doi.org/10.1007/978-3-319-73450-7_20
20. **Felix Arndt, Jérémy Bonvoisin, Tobias Burkert, Lukas Schattenhofer, Jerry de Vos, Fabian Flüchter, Martin Häuer, Dietrich Jäger, Timm Wille, Mehera Hassan, Robert Mies, Brynmor John, Manuel Moritz, Tobias Redlich, Christian Schmidt-Gütter, Emilio Velis, Joost van Well, Diderik van Wingerden, Tobias Wenzel, Lukas Winter,** and **Lars Zimmermann.** *Technical Rule DIN SPEC 3105-1:2020-07: Open Source Hardware, Part 1: Requirements for technical documentation*. Tech. rep. Berlin, Germany: Deutsches Institut für Normung e.V. (DIN), 2020, p. 13. DOI: https://doi.org/10.31030/3173063
21. **Jérémy Bonvoisin, Jenny Molloy, Martin Häuer,** and **Tobias Wenzel.** "Standardisation of Practices in Open Source Hardware." In: *Journal of Open Hardware* 4.1 (2020), pp. 1–11. DOI: https://doi.org/10.5334/joh.22
22. **Rafaella Antoniou, Romain Pinquié, Jean-François Boujut, Amer Ezoji,** and **Elies Dekoninck.** "Identifying the Factors Affecting the Replicability of Open Source Hardware Designs." In: *Proceedings of the Design Society* 1 (2021), pp. 1817–1826. DOI: https://doi.org/10.1017/pds.2021.443
23. **A. Ezoji, J.F. Boujut,** and **R. Pinquié.** "Requirements for design reuse in open-source hardware: a state of the art." In: *Procedia CIRP* 100 (2021), pp. 792–797. DOI: https://doi.org/10.1016/j.procir.2021.05.042
24. **BrickLink.** *BrickLink Studio*. 2023. URL: https://www.bricklink.com (visited on 01/17/2023).
25. **Allen Smith.** *Bricksmith*. 2023. URL: https://bricksmith.sourceforge.io/.
26. **Ronald Melkert.** *LDCad*. 2023. URL: http://www.melkert.net/LDCad (visited on 01/17/2023).
27. **James Jessiman.** *LDraw*. 2023. URL https://www.ldraw.org/ (visited on 01/17/2023).
28. **Leonardo Zide.** *LeoCAD*. 2023. URL: https://www.leocad.org/ (visited on 01/17/2023).
29. **Mecabricks.** *Mecabricks*. 2023. URL: https://www.mecabricks.com/ (visited on 01/17/2023).
30. **Lulzbot.** *OHAI: Open Hardware Assembly Instructions*. URL: https://ohai.lulzbot.com/ (visited on 01/17/2023).
31. **Joel T. Collins, Joe Knapper, Julian Stirling, Joram Mduda, Catherine Mkindi, Valeriana Mayagaya, Grace A. Mwakajinga, Paul T. Nyakyi, Valerian L. Sanga, Dave Carbery, Leah White, Sara Dale, Zhen Jieh Lim, Jeremy J. Baumberg, Pietro Cicuta, Samuel McDermott, Boyko Vodenicharski,** and **Richard Bowman.** "Robotic microscopy for everyone: the OpenFlexure microscope." In: *Biomedical Optics Express* 11.5 (2020), pp. 2447–2460. DOI: https://doi.org/10.1364/BOE.385729
32. **Joe Knapper, Julian Stirling, Joel Collins, Samuel Mcdermott, Valerian Sanga, Paul Nyakyi, Grace Anyelwisye, Greg Austic, William Wadsworth, Catherine Mkindi,** and **Richard Bowman.** "Transitioning from Academic Innovation to Viable Humanitarian Technology: The Next Steps for the OpenFlexure Project." In: *2021 IST-Africa Conference (IST-Africa)*. 2021, pp. 1–11.
33. **D.E. Knuth.** "Literate Programming." In: *The Computer Journal* 27.2 (1984), pp. 97–111. DOI: https://doi.org/10.1093/comjnl/27.2.97
34. **Eric Schulte, Dan Davison, Thomas Dye,** and **Carsten Dominik.** "A Multi- Language Computing Environment for Literate Programming and Reproducible Research." In: *Journal of Statistical Software* 46.3 (2012), pp. 1–24. DOI: https://doi.org/10.18637/jss.v046.i03
35. **Matthew Flatt** and **Eli Barzilay.** *Scribble: The Racket Documentation Tool*. 2022. URL: https://download.racket-lang.org/releases/8.7/pdfdoc/scribble.pdf (visited on 01/19/2023).
36. **Daniele Ingrassia.** *Fabulaser Mini*. 2021. URL: https://www.inmachines.net/Fabulasermini (visited on 01/17/2023).
37. **J.C. Mariscal-Melgar, Mohammed Omer, Manuel Moritz, Pieter Hijma, Tobias Redlich,** and **Jens P. Wulfsberg.** "Distributed Manufacturing: A High-Level Node-Based Concept for Open Source Hardware

Production." In: *Proceedings of the Conference on Production Systems and Logistics: CPSL 2022*. 2022, pp. 795–808. DOI: https://doi.org/10.15488/12171

38. **Marc Kohlen** and **Liane Sayuri Honda.** *Assembly Manual of the Fabulaser Mini.* 2021. URL: https://github.com/fab-machines/Fabulaser-Mini/blob/main/manual/Fabulaser%20manual%20L2M2.pdf (visited on 02/19/2023).
39. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *FreeCAD OSH Automated Documentation Workbench.* Version v0.1.0. 2023. DOI: https://doi.org/10.5281/zenodo.7633440
40. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *FreeCAD for OSH Automated Documentation.* Version v0.1.0. 2023. DOI: https://doi.org/10.5281/zenodo.7633414
41. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *OSH Automated Documentation PDF*. Version v0.1.0. 2023. DOI: https://doi.org/10.5281/zenodo.7633370
42. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *OSH Automated Documentation Data*. Version v0.1.0. Zenodo, 2023. DOI: https://doi.org/10.5281/zenodo.7633472
43. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *OSH Automated Documentation - Overview Video*. 2023. DOI: https://doi.org/10.5281/zenodo.7633581
44. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *OSH Automated Documentation - Creating an Assembly Manual for a Vise Video.* 2023. DOI: https://doi.org/10.5281/zenodo.7633593
45. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *Live Image for OSH Automated Documentation.* Version v0.1.0. 2023. DOI: https://doi.org/10.5281/zenodo.7633515
46. **J.C. Mariscal-Melgar** and **Pieter Hijma.** *OSH Automated Documentation Apptainer*. Version v0.1.0. 2023. DOI: https://doi.org/10.5281/zenodo.7652868
47. **Jean-Pierre Charras.** *KiCad EDA*. 2023. URL: https://www.kicad.org (visited on 01/17/2023).

TO CITE THIS ARTICLE:
Mariscal-Melgar, JC, Hijma, P, Moritz, M and Redlich, T. 2023. Semi-Automatic Generation of Assembly Instructions for Open Source Hardware. *Journal of Open Hardware*, 7(1): 6, pp. 1–18. DOI: https://doi.org/10.5334/joh.56

Submitted: 28 February 2023
Accepted: 19 July 2023
Published: 14 August 2023

Journal of Open Hardware is a peer-reviewed open access journal published by Ubiquity Press.