A Kantian approach to Free Software

Benedikt Peetz

benedikt.peetz@b-peetz.de

Wednesday 20th March 2024

Contents

1 Introduction

1.1 Motivation

The argumentation for Free Software has adhered to deontological ethics since its inception, as demonstrated by the Four Freedoms outlined in subsection 1.2—rights that must be categorically granted to all users.

This deontological influence continues in every one of the four freedoms: For example, Freedom 2, outlining the right to study the program to see what it does and to transparently decide whether to use the software, emphasizes the importance of the *autonomy* of a person.

But hitherto, Immanuel Kant, who has modernized the concept of deontological ethics and introduced the concept of autonomy, has not been mentioned in the arguments made by Richard Stallman. Consequently, I will try to address that, by connecting Kantian ideas (like Kant's Categorical Imperative (CI)) directly to the ideals of Free Software.

> *That* certainly needs a citation.

1.2 Definitions

As this essay will deal with the ethics of Free Software, the terms around Free Software should be defined:

Source Code computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator[1].

Software computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system[2].

Software Library a software library containing computer readable and human readable information relevant to a software development effort[3].

Free Software Software, which adheres to the four essential freedoms[4]. These are:

0. The freedom to run the program as you wish, for any purpose [. . .].

1. The freedom to study how the program works, and change it so it does your computing as you wish [. . .]. Access to the source code is a precondition for this.

2. The freedom to redistribute copies so you can help others [. . .].

3. The freedom to distribute copies of your modified versions to others [. . .]. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

Proprietary/nonfree Software Software not following the definition of free software (see subsection 1.2)[5].

---

[1] ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary, in: ISO/IEC/IEEE 24765:2017(E), Aug. 2017, 3.2741, DOI: 10.1109/IEEESTD.2017.8016712, p. 338.

[2] Ibid., p. 329.

[3] Ibid., p. 332.

[4] GNU Project (ed.): What is Free Software?, Version: 1.169, Feb. 2021, URL: https://www.gnu.org/philosophy/free-sw.html.en (visited on 14/02/2024).

[5] Idem (ed.): Categories of Free and Nonfree Software, URL: https://www.gnu.org/philosophy/categories.html#non-freeSoftware (visited on 16/03/2024).

1.3 Philosophical Premise

1.3.1 Kant

As I have mentioned in subsection 1.1 on the preceding page, I intend to apply the Kantian ethics to examine the morality of Proprietary Software development. This requires the knowledge about *how* Kantian ethics can be used to evaluate the morality of an action.

Firstly[6], one needs to identify the maxims behind the action being evaluated. This could be something like: "It is acceptable to write Proprietary Software".

Secondly[7], one needs to, following the CI, decide, whether an action can be universalized without causing contradictions. Kant writes that "dies ist der Kanon der moralischen Beurtheilung derselben überhaupt"[8]

This last step is effectively about applying the CI to the moral question, leaving immoral or self-interest driven actions to being exposed as not universalize-able—as these actions would contradict themselves when doing so. Kant additionally expresses the further requirement, if no contradictions are found, that the law must not be impossible to will, as this will would contradict itself[9].

1.3.2 Kant's Categorical Imperative

To achieve the aforementioned testing for morality the CI used for the universalization step needs to be defined because Kant has formulated multiple ones, each one of them focusing on slightly different areas. I will

be working mostly with the "Kingdom of Ends" CI[10]:

> Demnach muß ein jedes vernünftige Wesen so handeln, als ob es durch seine Maximen jederzeit ein gesetzgebendes Glied im allgemeinen Reiche der Zwecke wäre.

I have chosen to focus on this particular CI, as it highlights the importance of treating every rational being as an end and not as a mere means to an end. This interpretation comes from the fact, that the "Kingdom of Ends" ("Reich der Zwecke") is a possible Kingdom, where every being is an end in itself never a means. A developer must therefore act, as if their maxims are treated as universal law in a possible kingdom of ends.

2 Examination of morality

This essay is about whether the development of Proprietary Software is morally acceptable. Applying the steps outlined in subsubsection 1.3.1 means that we must start with trying to encompass the maxims behind the action. Deriving from the definition of Proprietary Software we must conclude that there are four reasons to not develop Free Software, as the developer does apparently not care about providing the four freedoms to their users. Thus, a possible maxim could be: "I do not want my users to follow one of the four freedoms".

To be able to be more precise—and because access to the source code is a precondition to two freedoms—, I am going to use the following maxim for the further evaluation: "I wish, that the source code is only visible to me".

---

[6]Cf. Immanuel Kant: Grundlegung zur Metaphysik der Sitten, AA IV, 1785, URL: http://kant.korpora.org/Band4/421.html, pp. 421 – 424.

[7]Cf. ibid., p. 424.

[8]Ibid.

[9]Cf. ibid.

[10]Ibid., p. 438.

Consequently, we need to check for the possibility of universalization for this maxim as a universal law. Which is impossible to achieve without producing a contradiction:

The developer will not have learned to develop in a vacuum, she will have red source code of her colleagues or of examples to understand common patters.

> **Wrong time from?**

Additionally, a software developer is commonly not interested in "re-inventing the wheel", as one might say. That means that software products commonly depend on other, so called, *software libraries* to function. These would be rather impractical to work with, if the developer does not have the at hand to check for invariants in the code or to work around lacking documentation.

> **Document that adding the source code is really industry-practice**

But let us assume that the maxim means source code of big applications and not examples or small snippets, that could be shared between colleagues. And let us also assume, that the external dependencies somehow work without having access to their source code.

Even in this—quite favourable—situation would the application of the CI result in an immoral action because of the second condition, that needs to be fulfilled: The law must actually be something wanted. And this is where we must reach a clear position: Nobody would want their access to others people source code be cut.

> **Is that argumentation correctly applied? Because it sounds a bit like the golden rule.**

Importantly, Proprietary Software is not only defined by access to the source code. Therefore, someone might actually allow full insight into the source code and still write Proprietary Software as the other two freedoms, not depending on the source code, could be violated. These are the rights to run the program as you wish (freedom 0) and the right to distribute copies of the software (freedom 2). A maxim that could express the first case would be: "I want to control how a user runs my software". Universalizing this *would* be possible: Programs today even enforce certain limitations on the way they are run (trying to run a Linux-only program on Windows will obviously fail, as the program was not designed for this foreign environment).

But, as before, this universalization attempt would stop at the point, where the motivation of said universalization must be considered: As most developers need to run quite a lot of different programs to develop software, not one of these would accept such a law, and thus not one would want that it becomes a universal law.

The same question must be asked about the second freedom, which could also be rejected. In this case the maxim at play would probably be: "I want to fully control the distribution of my software, and do not want copies of it to be shared". This is difficult to universalize without contradictions, as, like previously mentioned, software is often depended on other software projects. Thus, to distribute ones own software the dependencies must agree to being distributed. Which in turn means that the developer does not actually control the distribution of their software and that the maxim could only be universalized if software dependencies did not exist.

3 Conclusion

After having made a point that Kantian ethics in fact suggest, that the development of Proprietary Software is immoral, the fundamental problem must be mentioned, that the whole line of argumentation relies on the theoretical concepts proposed by Kant. These have time and time again been critiqued for being too theoretical and for be-

ing not really fit to application on real world problems. The best example, that comes to mind here is the known murderer asking a friend of his victim for the location of the victim. The friend must not lie, as universalization through Kant's CI results in a contradiction: An emergency lie would be defined through the subject, it can consequently not be universalized as an "emergency" is an inherently subject concept. Thus, Kantian ethics suggest telling the murderer the position of the friend and would therefore result in being responsible in the death of a close one.

Possibly alternatives to Proprietary Software include ideas like the "closed core" model, where only the core part of the software is Proprietary Software and the surrounding part is Free Software. These have not been evaluated, as these seem to rather uncommon today. One other alternative to completely Free Software is the counterpart to the "closed core" model, where the core software is developed openly, but then modified by the company behind it to add telemetry and other non-free software. One big example of this is the `Chromium` project, where `Chromium` is Free Software, but `Google Chrome` (Googles software product derived from `Chromium`) is not. These cases are impossible to evaluate directly, as the software being developed is without a doubt Free Software. Ergo, the action that must really be evaluated is the act of forking (i.e. copying the source code) of the Free Software project, modifying it and turning it into a Proprietary Software project before releasing it.

> I could also mention the position of Kant regarding "Neigungen" to other people and their unimportance in decisions.

> Is this the correct name?

References

ISO/IEC/IEEE International Standard - Systems and software engineering– Vocabulary, in: ISO/IEC/IEEE 24765:2017(E), Aug. 2017, 3.2741, DOI: [10.1109/IEEESTD.2017.8016712](https://doi.org/10.1109/IEEESTD.2017.8016712).

Kant, Immanuel: Grundlegung zur Metaphysik der Sitten, AA IV, 1785, URL: http://kant.korpora.org/Band4/421.html.

Project, GNU (ed.): Categories of Free and Nonfree Software, URL: https://www.gnu.org/philosophy/categories.html#non-freeSoftware (visited on 16/03/2024).

Idem (ed.): What is Free Software?, Version: 1.169, Feb. 2021, URL: https://www.gnu.org/philosophy/free-sw.html.en (visited on 14/02/2024).

The HPG Logo in the header has been taken from this website: https://wpn.hpg-speyer.de/