CS-425 Software Engineering

Individual Assignment | #2

Software engineering video clips,
ACM/IEEE Code of Ethics, and
Software Licenses

Contact:

Stephen Foster
stephenfoster@nevada.unr.edu

1. Software Engineering Video Clips

The Fast Inverse Square Root – An intuitive explanation:

This clip briefly covers the famous Fast Inverse Square Root algorithm, an incredible optimization in computing the multiplicative inverse of the square root of a floating-point number. This optimization circumvents the computationally expensive method of calculation using many division operations on floating-point values. As such, this algorithm is fundamental in computer graphics as it is used to normalize vectors efficiently. The video clip is significant in regard to software engineering directly as it discusses how the algorithm leverages the IEEE Standard for Floating-Point Arithmetic (IEEE 754). Additionally, the video clip walks the audience through the intuition of the various stages in which the algorithm behaves, and showcases those stages' graphed outputs against the graph of the accurate inverse square roots. This helps rationalize the algorithm, and sheds light on the magic number in which the algorithm relies. The graphs are further aided by mathematics demonstrations in real-time, which justify the aforementioned stages. The brevity in which the video clip creator maintains without sacrificing the arc of algorithm justification makes the clip worthwhile above many others covering the same topic. For computer scientists interested in computer graphics, knowledge of the Fast Inverse Square Root is both novel and fundamental toward deeper understanding.

Optimizing the usage of std::vector in C++:

This video clip by Yan Chernikov, otherwise known as The Cherno, briefly walks through two quick optimizations one can take advantage of when using the vector class provided by the C++ standard library. Optimization is critical to certain sub-domains of software engineering, and C++ is a critical tool in achieving both time and space efficient code. Thus, understanding the usage of standard library tools and ways they have built-in to optimize code is important to engineers seeking to do so. The video clip walks us through the motivations, the relevant domain topics, as well as example code setup and optimization execution. Unlike many code-related videos, Yan does an excellent job at explaining the intuition behind every line of code he displays, and as such the video clip interjects ample amounts of discussion between each new line of code. The two optimizations discussed are the usage of reserving memory space for a vector, and the built-in utilization of move semantics thanks to C++11. These respectively allow for programmers to avoid the internal copying of objects within the vector upon resize, and the inplace construction of objects immediately inside a container. I found this video in particular incredibly worthwhile as it is informative and approachable. C++ is a language with a vastness that is essentially unmatched, and content that makes C++ information approachable is valuable in its own right.

2. ACM/IEEE Code of Ethics

COLLEAGUES:

Concisely, principle 7 of the ACM/IEEE Code of Ethics, explicitly concerning colleagues in the field of software engineering, dictates that a “software engineer shall be fair to and supportive of their colleagues.”[1] The fully extrapolated principle covers topics such as credit attribution, objectivity in criticality, fairness, competence, and professional development in regard to other software engineers. In my own words, principle 7 dictates that a software engineer aid in both the construction and maintenance of the profession of software engineering. As such, software engineers must work to support one another in both a professional and humanistic manner. An example of this principle in action would begin with considering the dynamic between a senior engineer and a junior engineer. Software engineering encompasses a vast set of dynamic domains and disciplines in which it may interact, and the ability to navigate those evolving domains and disciplines is critical to both professional performance and development. A senior engineer has a duty to guide the junior engineer towards gaining such abilities for themselves, as supporting the software engineering profession is analogous to supporting its members.

MANAGEMENT:

Concisely, principle 5 of the ACM/IEEE Code of Ethics, explicitly concerning management in the field of software engineering, dictates that “software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.”[1] The fully extrapolated principle covers topics such as the inclusion of effective procedures for promotion of quality, pragmatic expectations of work and consideration of prior education, ownership of code, in addition to the considerations of social power dynamics and conditions of employment. In my own words, principle 5 dictates that a software engineering manager has responsibilities towards their product’s owners, clients, as well as its software engineers. More explicitly, software engineering managers must maintain environments that are humanistic, efficient, and considerate as far as anyone who is involved in the production, maintenance, usage, and ownership of software. An example that demonstrates this principle in action may begin with the consideration of a general software engineering team at some company or corporation. A manager has a duty to ensure that his engineers have reasonable expectations placed upon them when considering their education and experience, as it would be a letdown to the product owner, its employees, its clients, as well as those engineers to ask of them tasks that are non-sensible humanistically or pragmatically. A result of not performing this duty would result in the degradation of the product, the performance of the engineers, and the health of the workplace environment at the cost of the integrity of the profession.

SELF:

Concisely, principle 8 of the ACM/IEEE Code of Ethics, explicitly concerning oneself in the field of software engineering, dictates that a “software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.”[2] The fully extrapolated principle covers topics such as lifelong-learning, self-discipline, and the fairness in the treatment of oneself and others . In my own words, principle 8 dictates that one takes the steps necessary to improve the quality of their own work, which is critical to the improvement of quality of work by other professionals in both the contemporary and the future. The irony that exists in the interfacing of a software engineer and their tools is that software engineers are first and foremost humans. A healthy life and a healthy mind are the most effective attributes one has toward a lifelong journey of learning and development, and as such an engineer must focus on these foundational principles as they directly contribute to the ethical practice of the profession. An example that demonstrates this may begin with considering a software engineer that consistently lacks the maintenance of their mental, and physical health. An engineer who does not maintain these two foundations, immediately sacrifices their ability to improve their ability in any software engineering domain as well as their ability to perform their skills and deliver products that are reliable, safe, and cost effective in time. Ultimately, the ethical management of self has bimodal potential in its cascade of effects across the entire profession.

3. Software Licenses

B.) MIT and Apache 2.0 licenses:

Both the MIT license and the Apache 2.0 license are permissive software licenses, however they maintain some main differences between one another. Overall, the Apache 2.0 license is thorough and explicit in regard to legal terminology and patent rights, whereas MIT makes no such restrictions and leaves patent rights ambiguous. Apache 2.0 requires derivative works to disclose major modifications to the original source code, whereas MIT makes no such explicit remark regarding any form of disclosure other than the license itself and the original works’ authors in a particular format. Another major difference is in regard to patents, where Apache 2.0 automatically grants a user a license to use the patent covered by Apache 2.0 in original works, but revokes the license upon the initiation of litigation over the patent on any entity, effectively limiting the ability to patent or lawsuit troll. Lastly, Apache 2.0 allows a user to license modified code under any new license they wish, as long as any unmodified code retains the original license.

C). Public Domain & GNU/LGPL software licenses:

The Public Domain is equivalent to an “unlicense” where no conditions apply, which effectively means works can be distributed without access to its source code and under any or no terms. The **GNU General Public License** preserves license notifications and any copyright terms of the work. Software under GPL must release alongside its source code under the same license, making it a strong copyleft license. The **LGPL**

Lesser General Public License is itself one that retains the same terms of the **AGPL Affero GPL** where a loophole of network accessed software is closed in GPL. The main takeaway of LGPL is its explicit discussion of smaller works accessed through larger works licensed under GPL, where now LGPL allows for the lack of inclusion of the larger work in the release, in addition to the fact that modifications can be distributed under license terms that are different from the larger work.

D). Permissive & Copyleft software licenses:

Permissive licenses form a group of software licenses that are designed to allow for freedom of code use through explicitly broadening the freedoms of its reuse, modification, and distribution at varying levels by others.

Copyleft licenses form a group of software licenses that form the antithesis to the concept of copyright law by inversely enforcing freedom of code use potential by others. Code derived from copyleft licensed software is enforced to inherit its license terms.

References

[1] https://ethics.acm.org/code-of-ethics/software-engineering-code/

[2] https://snyk.io/learn/open-source-licenses/