**Project Submission Explanation**

I felt the need to include a brief explanation of how I am submitting my project, so please read this before continuing on to the paper below. I collected my submission items a system of files and folders on my computer using an organization system called ‘git.’ Programmers use git to organize their code and track changes to it. I wanted to present this project in that way because it ties into programmer culture. To make submission easier, I bundled all these files into one by “zipping” them. Almost any computer system can easily unzip the files to their original format simply by downloading the file and then double-clicking on it. However, I have also hosted this project on a website called “GitHub,” a place where programmers store the projects and code they use git to manage. This project is best viewed there, as the website is designed for this project structure.

Here is the link: <https://github.com/MelissaR2D2/programmer-folklore>

**Buggy Biscuits and Rubber Ducks: An Incomplete Collection of Programmer Folklore**

Melissa Robertson

Department of English, Brigham Young University

ENGL391: Introduction to Folklore

Professor Eric Eliason

December 8, 2020

**Buggy Biscuits and Rubber Ducks: An Incomplete Collection of Programmer Folklore**

Behind every program, app, or website you have ever used or visited, is a programmer. Probably more than one programmer, in fact, and most likely dozens. Programmers, simply put, are the people who write the instructions for a computer to execute in a way translatable to the 1s and 0s all computers run on. When a set of instructions put and run together to accomplish a specific task, that is called a program. Programming is an extremely broad field because both the definition of “program” and “computer” encompass more than people usually think about. Computers are not just your phone or your laptop - they are also in vending machines, ATMs, and even traffic lights. Programs aren’t just the apps you open on your phone, but also the operating system that starts up when you push the power button, and even the rules your USB port uses to talk to the mouse you just plugged in.

I became interested in programmer folklore because I am a programmer. I am studying computer science and plan to go into software development after college. Programmers are primarily an occupational folk group, because even hobbyist programmers tend to wind up working in tech. Common stereotypes of programmers include that they are white and male, geniuses but lazy, and introverted and antisocial, preferring to stare at a computer screen all day rather than interact with people. While most programmers would admit that these stereotypes have a grain of truth, programmers still share a strong culture born of common struggles and insider knowledge despite their introversion. Like all shared cultures, programmer culture generates folklore. As I have worked with other computer science students and eased into the industry through internships and similar opportunities, I have been immersed in programmer culture for three years now, and so it was only natural for me to notice the folklore I was already engaging in.

I drew my folklore primarily from sources with whom I already regularly engage in programmer folklore. These included friends, classmates, and co-workers, as well as class Slack servers and Brigham Young University’s unofficial Computer Science Discord server. Slack and Discord are both social media instant messaging apps that provide a platform for communities from dozens to thousands in size to chat. Each community has a “server,” and each server will be organized into “channels” by conversation topic to keep things easy to find and moderate. Slack is geared toward workplaces and other more formal communities, while Discord is geared towards gamers and hobbyist communities. I also took some folklore items from the Internet, specifically the ProgrammerHumor subreddit. Reddit is a social media site focused on link and content sharing. The site is divided into "subreddits," each focused on specific topics. Content is posted as "threads" that users can then comment on, creating a forum for discussion of the content posted. Reddit has a broad user base, but is generally considered to be most popular with young white males. It often reposts or links to content from Twitter and other social media sites.

Overall, I was able to find a lot of programmer folklore from a variety of sources, but my sample is far from representative. In particular, all of my non-anonymous sources were BYU students in their early twenties. Older generations’ programmer folklore may be different, and similarly folklore borne from years of experience in industry may be different from the folklore circulating among programmers who are still students.

The folklore I found spans several genres, including jokes, proverbs, memes, legends, and traditions. It exhibits many of the features of folklore: it is traditional, stereotypical, variable, and anonymous, and sometimes artistic. However, other features are expressed atypically or are almost entirely absent from programmer folklore. Finally, this folklore performs several vital functions in the programmer community: it creates and maintains a shared programmer identity, it passes on community knowledge not suited to academic settings, and it provides a release valve for common frustrations that arise when programming.

Of course, this collection is by no means exhaustive. I can think of half a dozen excellent meme formats and jokes off the top of my head that I unfortunately was not able to collect due to time constraints and the lack of a natural setting. Still, I enjoyed creating this collection, and I think it provides an accurate window into the programming community’s folklore. It was fun to reflect on something I spend so much time engaging in. This area also appears to be a ripe ground for research; I have found only one item related to programming in the William A. Wilson Folklore Archive. “Computer Savvy” is an interview collected in 2016 from a BYU Computer Science major. The interviewee explains some programmer jargon, such as calling computer errors “bugs” or saying a computer is running “slow” when really it’s running at the same speed, it just has too much to process (“Computer Savvy”). However, it’s not very clear what aspects of programmer folklore was being discussed because the interviewer did not record the questions she asked. Still, the item is a good demonstration of the significant vocabulary and technical know-how that necessarily separates programmers as a folk group from non-programmers. So it should not be surprising that, as with any distinct group that works closely together and shares common, exclusive knowledge, many cultural items have developed in programmer communities that bear features of folklore.

**Features**

The programmer folklore I gathered clearly exhibits several features that distinguish it from other types of culture. First, it is clearly variable. Just in the scope of this project, I was able to gather two distinct variations of one legend and one proverb, and I even found three variations of the traditional “Hello World” program. I was particularly shocked by the variation in our etiological legend for the jargon term “bug,” which refers to an error in a computer program. I have always heard “The Literal Bug in the Computer” variant, and had no idea any others existed until Professor Eric Eliason told me about “Bugs in Biscuits.” In general, however, I focused more on breadth than depth when gathering folklore for this project, preferring to gather a variety of folklore rather than lots of variations of one specific folklore item.

Second, it is stereotypical. Although again, I focused more on breadth than depth when gathering folklore for this project, patterns and genre conventions are evident in each of the genres of folklore I found. The jokes I gathered both follow specific structures: “Java Programmers Can’t C” is a “what do you call” joke, and the “Quality Engineer in a Bar” joke is a “walk into a bar” joke. The memes are text put on templates widely used for other memes. Finally, the traditional “Hello World” program conforms to a certain specification: it must print the words “Hello World.” The exact capitalization and punctuation can vary, and the joke version of the program, the “Most Dangerous ‘Hello World’ Program,” does something else as well, but even it still prints “Hello World.”

Thirdly, it is traditional, passed between programmers and occurring repeatedly. Almost every folklore item I gathered was one I had heard or seen at least one variant of before, showing that these items circulate widely and are repeated often. However, programmer folklore is often passed on and participated in in different ways and different settings than traditional folklore. Programmer folklore is rarely performed orally or face-to-face; I only collected one oral item, and even that was using a digital medium due to COVID-19. The rest were all digital and text or image-based. Some were still a direct performance meant for a specific person or a small group (such as those from the Discord server), but the ones I drew from Reddit are examples of folklore thrown figuratively into the void, for anyone and everyone to see and engage in.

Another example of nontraditional engagement is that some programmer folklore, specifically certain traditions, is often performed completely alone. The “Hello World” program, which simply prints the words “Hello World” to the screen, is a traditional program written when either first learning how to program, learning a new programming language, or setting up a new system for programming work. In the second and third cases, the program is usually written alone, and there is no reason to show it to anyone. Similarly, “rubber duck debugging” is a tradition where programmers explain their code to an actual toy rubber duck (or other inanimate object) in order to figure out why the code is not working. Again, this is usually done while alone (though sometimes another person does act as the rubber duck). However, this folklore is still passed on through the Internet and in traditional social settings, with more experienced programmers sharing it with newer programmers. There also exists meta-folklore about the traditions that keep them alive, such as the meme displayed in the #rubber-duck channel featured in the item “Rubber Duck Debugging.”

Between a programmer’s comfort with digital media due to their job and the sometimes uncomfortably true stereotype about programmers being at best introverted and at worst antisocial, it is not at all surprising that programmers share folklore using nontraditional, digital means that play more to their strengths, including performing some folklore alone. However, I would argue that programmer folklore is still highly social, because the functions it performs are essentially social functions that create and reinforce a sense of community among programmers. The traditions performed alone persist because, despite their isolated setting, they still fulfill these social functions. I now turn to a discussion of the functions programmer folklore performs, and the social roles of these functions.

**Functions**

The programmer folklore I collected performs several of William Bascom’s four functions of folklore. First, programmer proverbs clearly play the role of reinforcing social norms and educating new programmers on tricks and traps of programming that can often only be learned through painful experience. I collected two distinct proverbs, both of which fall squarely into this category. The first one concerns good planning practices when coding. Because code is often complex and abstract, with many separate parts that must fit together or behave in highly specific ways, a good programmer will plan out each of these parts before writing any actual code. Failure to do so can result in a lot of wasted time when you start writing code, realize it won’t fit certain requirements, and have to significantly restructure it or start over.

The two variations of this proverb I collected were given as follows: “20 hours of coding can save you 30 minutes of planning,” and “3 hours on the keyboard can save you 15 minutes with a pen and paper.” The educational use is obvious; new programmers’ instinct is often to jump right in and start coding, and only experience teaches them otherwise. This proverb is condensed wisdom of thousands of programmers trying to teach new programmers and even remind each other not to make the same mistake. It also reinforces social norms by emphasizing what good coding practice looks like, and because it is often passed on in a teacher-to-student context. Notably, both variations I collected occurred in an academic setting. The first involved a TA recommending the student follow the proverb’s advice by completing a planning assignment before the main coding project, and the second, while said by a student, was stated in response to a professor urging students to plan an upcoming project using similar language, further supporting my hypothesis about these proverbs’ functions as folklore.

The second proverb I collected warns of a different programming woe: the trap of automation. “Never spend 6 minutes doing something by hand when you can spend 6 hours failing to automate it,” says Twitter user Zhuowei Zhang. This Tweet was then posted to the ProgrammerHumor subreddit, where it quickly became one of the top posts this year. It is only one variation of this sentiment I have seen in programmer circles. Because a programmer’s entire job can be boiled down to writing computer instructions to automate things, we are trained to see inefficiency and automate it away. Unfortunately, not everything can or should be automated, and this proverb attempts to teach and reinforce that principle. All too often, automating a task is more difficult and time-consuming (but also more interesting and fun) than simply doing it by hand, and often these tasks only need to be done once or twice, making the time spent on automation clearly not worth it. To add insult to injury, you may not even be able to get the automated version working, as in the proverb. I have personally fallen victim to this more than once as a programmer. This proverb clearly functions as an educational warning that also reinforces social norms, and the woes of wasted time be upon those who do not listen.

Interestingly, both these programmer proverbs were sarcastic, creating humor by “recommending” the inefficient course of action, although every programmer knows it is actually warning against that course. This humor allows these proverbs to, in addition to educating and reinforcing social norms, perform the most common function of programmer folklore I found: allowing a release valve for the common frustrations and challenges of programming.

This function was clearly present in seven out of the fourteen folklore items I collected. This included all the jokes, memes, and proverbs, but mostly not the legends or traditions. Jokes inherently tend to function as a release valve because of the way they juxtapose incongruent elements and then release the tension through the humor of a punchline. The “Java Programmers Can’t C” joke releases tensions of rivalry between programmers who use different programming languages, while the “Quality Engineer in a Bar” joke allows programmers to laugh at the difficulty of ensuring code is bug-free.

The two memes I collected both also act as a release valve, though for different tensions. “My Advice is Probably Questionable” expresses the tension of trying to give programming advice while feeling that your own code is secretly a disaster, which generalizes to a release for the impostor syndrome that is common in technical communities. “We're Not Tech Support” releases the tension that comes from dealing with people not “in the know” about programming. While programmers have to know a lot about computers to do their jobs, no one person can know everything about all electronic devices. The most common computer problems non-technical people have, such as managing printers, installation software, and protecting the computer from viruses, are not actually part of a programmer’s core skill set at all. Those are jobs for IT people, and the two fields are different enough that they are usually different majors at universities. Unfortunately, most non-technical people do not know this difference, and so they ask their programmer family and friends to just quickly help them fix their printer, not knowing that at best this is irritating because the programmer doesn’t have the time for free labor, or at worst has no more idea how to fix it than they do. This meme releases that tension by turing it into a relatable joke and allowing programmers to express their frustration away from the pressures to be polite in the moment.

Although most of the legends and traditions did not function as release valves, the central figure of one of the legends does have certain trickster features. This is the “First Computer Virus” story, where the programmer breaks social norms by writing a program that copies itself to other computers without the knowledge of their owners. Although the programmer’s motivation was mostly innocent curiosity about how far the program would spread, one poorly designed aspect of the program turned it into a program with much more detrimental effects — a virus that copied itself over and over again, locking up computers by filling up their memory. This trickster figure could serve as a release valve because he got to do what most programmers can’t: write a cool, far-reaching program and release it just to see what happens. However, we also see here a common trickster motif: his transgressing actions result in unintended consequences as the virus grows out of control. In the end, the legend reinforces the social norm of not writing harmful code by claiming that “[the programmer] has a hard time getting a job even now.”

Finally, traditions such as “Hello World” and “Rubber Duck Debugging,” though often performed alone, reinforce a programmer’s sense of identity and community with other programmers. Knowing that thousands of other programmers have written the same program as you, or have rubber ducks sitting on their desks just like you, helps create a strong community bond without ever having to actually talk to another programmer. Similarly, all the folklore items that function as a release valve for common programming difficulties reassure us that we are not alone in our struggles.

In conclusion, programmer folklore is varied, vibrant, and fascinating, combining many traditional features and functions of folklore with some adaptations for a more introverted culture full of people more at home in front of a screen than face-to-face with a small group of people. I feel I have barely scraped the surface of the programmer folklore out there to be collected. The open-source community and the hacker community are both subgroups of the programmer community that could warrant an entire project alone. However, I hope this project has presented an enlightening and enjoyable glimpse into the rich and varied world of programmers. If you know a programmer, ask them if they’ve ever gotten lost in vim. You might be in for a fun story.

**References**

Collected by Amada DeBuse from Andrew Joseph Palmer, “Computer Savvy.” 3 Feb 2016. Wilson Digital Folklore Archives, Brigham Young University, Provo, Utah. 9 Dec 2020.