GSC Proposal 2024

J.H. DeBlois

14 November 2024

1 A. Title: Build Confidence In Software, Appendix A

The title of the proposal is "Build Confidence In Software".

2 B. Principal Investigator: Jane Holly DeBlois, PhD

This proposal is based on teaching experience that already has some collaborations across UMB in place. This past semester, while looking for seven new software projects for the fifty-one students enrolled in CS410, the instructor discovered opportunities outside the Computer Science (CS) Department.

Prof. Edward Miller, Gerontology, contributed ideas for a project to build software for elderly people. His prior student, Debby Dowd, is Team 6 Client.

Bala Sundaram, Shubhro Sen and Maria Vasilevsky, Venture Development Center, contributed a project to build software doing climate carbon credit assessments. Alvin Tian is Team 5 Client.

2.1 Need

This proposal is focused on how to teach students to build modern software well. The assessment that building software is difficult to do has been the case for many decades.

We transitioned from "waterfall" methodology to agile methods over twenty years ago, but are still facing the challenge of finding an educational paradigm that combines traditional behaviorist methods necessary for large class size with the collaborative

methods appropriate for team-based learning and the humanist methods that can focus learning on something specific that all students see as beneficial and which can be graded by clear critieria.

It is the doing and the facts about what works that drive invention toward the client's goals. Thus, the confidence is not only confidence in the code built but more importantly confidence in the payoff of each hour worked.

2.2 Impact

The educational paradigm is unusual. The instructor uses the textbook Essential Scrum to teach the terms. The homeworks reinforce writing requirments as single sentences, writing tests so clients as well as developers understand and making a list of items to build in the days of the sprint. See the wonderful drawing of sprinting shown as Figure 1 in Section E.

Then the instructor introduces the clients who explain the team projects. The educational paradigm shifts gradually to collaboration and building code. The instructor watches. A first round of student slides explains what work each student did during the days of the first sprint and what new skills the student acquired.

The proof in Appendix A is offered to counteract the thought that software is taking over our lives. It isn't. The tech revolution continues for sure so the number of syntactically and semantically correct computer programs running keeps growing. As do the programs that have flaws.

The number of human thoughts and actions (that we call human functions) is not only infinite but uncountably infinite (like the infinity of real numbers you can find between any two real numbers). Think of the number of meetings occurring all over the world in which humans are discussing AI. Think also of the number of times in the day when your activity has nothing to do with a computer program.

Assisted by the CPIs, the PI plans to write an academic paper on the educational paradigm. It will address scouting for projects at UMB, as well as combining the behaviorist paradigm with the collaborative paradigm that has the human values dimension more in the front. It will also address ways to measuring increases in student confidence, as explained further in Section E. It will also address how the values in the classroom might translate into policy at higher levels.

2.3 Methods

The scholarship of how we build resides in the class, instructor, graders and clients. Students are not graded on their product, but instead on how they build during the sprint. The client expresses opinions about the product as frequently as they desire, so there is no way the client does not get a product that is what they want, although of course some aspects may not yet be built.

Scholarship deserves a second comment here. For learning to be sound as well as deep, it must be accomplished in a manner that meets the highest standard of academic honesty. The PI has requested the provost to add the line "Writing Code" right after the line "Writing and essays" in the description of Academic Work on the UMB website https://www.umb.edu/provost/academic-work (check this).

The instructor collects data during the semester and analyzes it further after the semester. In a manner similar to student activities in the class, the instructor will update the paradigm and the work product aspects that are assessed.

3 Appendix A. Proof that whereas the number of possible computer programs is countably infinite, it is less than the number of human functions, which is uncountably infinite.

This proof provides essential context for the proposal. A similar proof has been taught in Prof. Simovici's CS622. Focusing on the enormity of human functions enables us to teach how to build code in proper prospective.

3.1 Objective

What needs proof is: Computer programs can be modeled as words in a language and listed in order with a proof that the size of the set is countably infinite. The symbol $\aleph_0$ read as "aleph null" stands for a countable infinity.

Human functions can be modeled, but attempts to list them fail, and we can prove that the number of such functions is uncountably infinite. Thus, forever bigger than $\aleph_0$.

When a human activity is partially replaced by a computer program, some new human functions may become necessary. Our proof shows that the total of human functions must grow too, and by more. The reference for the next two sections is Theory of Formal Languages with Applications, Dan A. Simovici and Richard L. Tenney, World

Scientific, 1999.

3.2 Count the Number of Computer Programs

To count the computer programs, we use the concepts of sets, cardinality, bijection, alphabet, words and language. As Prof. Simovici mentioned (p. 57), "Programs and the data they manipulate may be regarded as words over appropriate alphabets."

On p. 57, an alphabet is a finite non-empty set. The elements are symbols. Definition 2.2.1 A word of length n on an alphabet A is a sequence of length n of symbols of this alphabet.

Also, "Example 2.2.3, p. 58, Any C program is a word over the basic alphabet of this language that includes small and capital letters, as well as special symbols, such as parentheses, brackets, braces, spaces, new line characters, quotation marks, etc. Not all these characters are visible; in other words, some characters (such as spaces) appear on paper only as white spaces. For example, the famous C program:

```
#include <stdio.h> main(){ printf("hello, world\n"); }
```

can be looked at as a word."

Now, from cs622 class notes, "Numbering Words", we have: "Let $A = \{a_0, \ldots, a_{n-1}\}$ be an alphabet containing n symbols. Words over A can be encoded as natural numbers; in other words, we can define a bijection $\phi_A : A^* \longrightarrow N$ by:

$$\phi_A(x) = \begin{cases} 0 & \text{if } x = \lambda \\ n\phi_A(y) + i + 1 & \text{if } x = ya_i \end{cases}$$

for every x in A^*."

This is a recursive function that pushes left on each step. See Example, same page, if $A = \{a_0, a_1, a_2\}$, $x = a_0a_1a_0a_2$ and $y = a_2a_2a_2$, then:

$$\phi_A(x) = 3^3 \cdot 1 + 3^2 \cdot 2 + 3^1 \cdot 1 + 3 = 51$$
$$\phi_A(y) = 3^2 \cdot 3 + 3^1 \cdot 3 + 3 = 39$$

Note that any word of a certain length will have its interval, no two distinct words in A^k can be mapped into the same number in $l(k)$, so ϕ_A defines an injection of A^k into $l(k)$; and for every number m in $l(k)$ there is a word x in A^k such that $\phi_A(x) = m$. Therefore, ϕ_A is a bijection between A^* and $\mathbb{N}$.

Therefore, the number of possible computer programs is a countable infinity.

3.3 Count the Number of Human Functions

Now, we count the number of human functions. Suppose we start with a list L of the basic things humans can think of. Then, because ideas take form in different ways, we let each human function be a subset of that list. $P(L)$ can be shown to be not countable in a proof by contradiction. Suppose $P(L)$ is countable. Then we have the mapping $f : \mathbb{N} \longrightarrow P(L)$.

Per p. 44, we could have this list:

0:	a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	$\cdots$	(a particular subset)
1:	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	$\cdots$	
2:	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	$\cdots$	
3:	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	$\cdots$	
4:	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	$\cdots$	
5:	a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	$\cdots$	
		$\vdots$					
k:	a_{k0}	a_{k1}	a_{k2}	a_{k3}	a_{k4}	$\cdots$	a_{kk}

where:

$$a_{ij} = \begin{cases} 0 & \text{if } j \notin f(i) \text{ where } f(i) \text{ is a particular subset} \\ 1 & \text{if } j \in f(i) \end{cases}$$

Then, quoting further from Simovici and Tenney, p. 44, "In other words, the a_{ij}s correspond to the characteristic function of the set $f(i)$. The set D is formed by 'going down the diagonal' and spoiling the possibility that $D = f(k)$, for each k. At row k, we look at a_{kk} in column k. If this is 1, i.e., if $k \in f(k)$ then we make sure that the corresponding position for the set D has a 0 in it by saying that $k \notin D$. On the other hand, if a_{kk} is a 0, i.e., $k \notin f(k)$, then we force the corresponding position for the set D to be a 1 by putting k into D. This guarantees that $D \neq f(k)$, because its characteristic functions differ from that of $f(k)$ in column k."

Thus no bijection. Therefore, we have a proof that the number of human functions is an uncountable infinity.

This proof technique, diagonalizaton, first appeared in the 1891 paper of Georg Cantor.

4 References:

1. DeBlois, J.H., https://www.cs.umb.edu/ hdeblois/0list51/. Click on JHD CV or explore cs410 student initial simple projects.

2. Joint Task Force on Computing Curricula, Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Progams in Software Engineering, August 23, 2004.

3. National Center for Science and Engineering Statistics, National Science Foundation, "Diversity and STEM: Women, Minorities and Persons with Disabilities: 2023", NSF report.

3. Roycroft, Sian, Elevate Your Teaching: Discover the Power of Educational Paradigms, Edge Education, https://edgeeducation.com/discover-the-power-of-educational-paradigms/, June 21, 2023.

4. Rubin, Kenneth S., Essential Scrum: A Practical Guide to the Most Popular Agil Processes, Addison-Wesley, 2013.

5. Simovici, Dan A. and Richard L. Tenney, Theory of Formal Languages with Applications, World Scientific, 1999.

6. Sotomayor, Sonia, Speaking at Radcliffe Day, June 2024, video:

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.youtube