ANALOG DRIVEN DEVELOPMENT

Harnessing the Conceptual Human Mind to Ensure Software Artifact Stability

Matthew James Swann, *Bachelor of Arts*

Permission is granted to Auburn University to make copies of this document at its discretion, upon request of individuals or institutions and at their expense. The above referenced author reserves all publication rights.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Signature of Author  
  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date of Graduation

PROJECT ABSTRACT

ANALOG DRIVEN DEVELOPMENT

Harnessing the Conceptual Human Mind to Ensure Software Artifact Stability

Matthew James Swann, *Bachelor of Arts*

Directed by David Umphress, *PhD*

<insert meaningless summary drivel here, good stuff comes later>

**Table of Contents**

Table of Contents

Table of Figures

Chapter One – Introduction

Chapter Two – Background

Chapter Three – Implementation

Chapter Four – Validation

Chapter Five – Conclusions

References

Appendix Alpha

Appendix Beta

Appendix Gamma

**Table of Figures**

Table of Contents

Table of Figures

Chapter One – Introduction

Chapter Two – Background

Chapter Three – Implementation

Chapter Four – Validation

Chapter Five – Conclusions

References

Appendix Alpha

Appendix Beta

Appendix Gamma

**Chapter One – Introduction**

<insert significantly more meaningless summary drivel here, good stuff comes later>

**Chapter Two – Background**

<start with either Beck or Blachowicz, then the other current state of the Art likely then what it’s missing>

*2.1 – Test Driven Development : The First Baseline*

<BECK HERE>

*2.2 – Nature of Inquiry : The Second Baseline*

James Blachowicz, PhD, author and former professor at Loyola University Chicago provides the necessary foundation for my definition of inquiry. In *The Nature of Inquiry*, Blachowicz suggests that inquiry itself is a dualistic process that mandates “the partial generation from experience of ideas which come to explain experience, and the partial generation from ideas of consequences which come to match experience.” Blachowicz goes on to simplify this definition into a two-sided process involving both experience and thought.[1] One must be able to interact with the known portions of the problem while contemplating the unknown portions.

This dualistic consideration is important to the definition of inquiry as it contains the necessary pieces to solving any problem, be it in the venue of engineering, mathematics, logic, etc. The need to solve a problem requires one to know various pieces of information about the problem while simultaneously not knowing the problem in some way.[1] (For Blachowicz, this is the first law of inquiry.) Strictly speaking the solution to the problem is unknown, but other pieces of the puzzle may also be obfuscated. This could be a variable’s behavior over time or the effect of multi-variable interaction. However, the problem itself must have a definition. Without a bounding definition, no problem is resolved in an intelligent manner.

Accidental solutions may be discovered for various problems, but for the purposes herein the premise is that we have a specific software problem that must be solved. As such, there is a desired result and intelligent observation of the distance between the known position and the goal can be made.[1] (For Blachowicz, this is the second law of inquiry.) This provides a means for intelligent inquiry. Spontaneous inquiry and randomized creativity is outside the scope of this discourse.

Each piece of software that must be written is a unique problem requiring a unique solution. If a solution to a software problem already exists, generally the problem is not resolved again. Academic settings may require resolutions that have already surfaced for learning purposes. However, in practice, reusability is a primary tenant of development. This focus has a twofold purpose. One, reusing existing code promotes confidence if the code is known to “work”. Two, reusage detracts from overall development time. Therefore, almost every software solution is a solution unique unto itself even if the uniqueness takes the form of refinement. Facebook must only be made once. The database aspect of Facebook remains the same from access medium to access medium. The rendering of that information may change, but therein lays a novel problem requiring a novel solution. The code executed by my Playstation when I load Assassin’s Creed is the exact code run by every Playstation when Assassin’s Creed is loaded on each gaming console. It would not be Assassin’s Creed unless this held true. It might be a second installment of the game. It might be a similar game. But it would not be the same.

As each piece of software inherently contains the resolution to a novel problem, each piece of unfinished software necessitates inquiry. We must discover the solution to what it is we wish to build. Later I will discuss the location of problem resolution, but for now knowing we have a unique problem is sufficient. Above, I discussed the ability to both know and not know the solution to a given problem. Meno’s paradox suggests that this type of knowledge is impossible :

“And how will you inquire into a thing when you are wholly ignorant of what it is? Even if you happen to bump right into it, how will you know it is the thing you didn’t know?”[10]

Firstly, it may be impossible to inquire into a thing that one is whole ignorant of, for how would one know to inquire of it in the first place. Secondly, this is not the situation being explored. When solving a software problem, one knows what the desired result is. This follows suit with the second law of inquiry. One also knows both the functional and the aesthetic portions of that desired result. This knowledge can be converted into a first order map, a mechanism that intelligently determines a specific direction to head when traversing a problem.[1] The developer also is well aware of several use cases or testing scenarios that ought be passed before the software is completed. This amalgamation of knowledge paints a picture as to what the desired result of the effort is. We know exactly what we want the end behaviors to be and we know how we want it to look and feel. We do not know how we are going to model those behaviors or their actors yet. Thusly, we have knowing while not knowing.

**Chapter Three – Implementation**

<similarities to TDD, differences from TDD, VERSUS examples, SCOPE examples>

<Inherited logic from NoI; First order map references>

<How to>

**Chapter Four – Validation**

<All the wicked cool shit ADD does. List it, prove it. Get on with it>

**Chapter Five – Conclusions**

<Future work would be arrogant here. The process itself needs to simply be refined but to be refined it has to be used. I need to screw it up. I need to make it messy. I need to break it, so I can make it stronger>

<basic discipline yields marvelous results>

**References**

[1] J. Blachowicz, *Of Two Minds: The Nature of Inquiry*, State University of New York Press, Albany, NY 1998

[2] K. Beck, *Test-Driven Development by Example*, Pearson Education, Inc., Boston, MA, 2003

[3] W. Strunk, E.B. White, *The Elements of Style*, 15th Edition, Pearson Education, Boston, MA, 2009

[4] M.J. Swann, *Tailoring a Pre-Existing Software Process: A Logical Interface Between the Intellectual and the Empirical*, Auburn University, 2011

[5] F. Keenan, *Agile Process Tailoring and probLem analYsis (APTLY)*, Proceedings of the 26th International Conference on Software Engineering, 2004

[6] I. Yoon, S. Min, D. Bae, *Tailoring and Verifying Software Process*, Software Engineering Conference, 2001, ASPEC 2001 Eighth Asia-Pacific

[7] V. Basili, H. Rombach, *Tailoring the Software Process to Project Goals and Environments*, Proceedings of the 9th International Conference on Software Engineering, ICSE 1987, Review by B. Boehm

[8] T. Ashley-Farrand, *Shakti Mantras*, Random House, New York, NY, 2003

[9] J. Swift, *A Modest Proposal*. 1729. Quotidiana. Ed. Patrick Madden. 19 Dec 2007. 10 Jan 2014 <http://essays.quotidiana.org/swift/modest\_proposal/>.

[10] R.S. Bluck, *Plato’s Meno*, Cambridge Press, Cambridge, MA, 1964

**Appendix Alpha**

**Appendix Beta**

**Appendix Gamma**