Exploring the Feasibility and Practicality of an Abstract Software Development Framework for Maximizing Project Success

An Honors Thesis (HONR 499)

by

Seth Campbell

Thesis Advisor

David Largent

Ball State University
Muncie, Indiana

April 2020

Expected Date of Graduation

May 2020
Abstract

Success is integral to our society. We strive for it, place bets on it, and wager our future for it. Some would say success depends on a bit of luck, but what if success can be planned? That was the aim of this study particularly in the context of the ever-expanding world of technology, more specifically the computer science community. Through an in-depth look at the fundamentals of success and their applications in the computer science world, we explore the possibility of creating a master framework for achieving success as part of a software development team. With an understanding of such a framework’s components, a templated framework is created and put to the test, showing hopeful yet ambitious results for an all-purpose, one-size-fits-all solution for developers working towards a successful project.

Acknowledgments

I would like to thank David Largent for his help in guiding, advising, and thoroughly reviewing this thesis. He has not only been an advisor to me, but a mentor, professor, friend, and overall wonderful person.

I would also like to thank my wife Kyra, who has been right alongside of me through all the late nights, encouraging me, even though she has no interest in computer science.
# Table of Contents

**Process Analysis Statement** .................................................................................. 1

**Introduction** ........................................................................................................... 3

**Discovery and Analysis** ....................................................................................... 5
  - Defining Success .................................................................................................. 5
  - What exists already? ......................................................................................... 8
    - *Snippet 1.1* ............................................................................................... 11
    - *Snippet 1.2* ............................................................................................... 11
    - *Snippet 1.3* ............................................................................................... 11
    - *Snippet 1.4* ............................................................................................... 13
    - *Snippet 1.5* ............................................................................................... 13
    - *Snippet 1.6* ............................................................................................... 13
  - Framework Architecture ..................................................................................... 14

**Constructing the Framework** ............................................................................. 17
  - The Name and Forward .................................................................................. 17
    - *Image 2.1* ............................................................................................... 17
  - Stage One: Define Success ......................................................................... 18
  - Stage Two: Create a Plan ............................................................................ 18
  - Stage Three: Follow the Plan ...................................................................... 20
  - Stage Four: Change the Plan ....................................................................... 21
  - Stage Five: Finish ......................................................................................... 23
  - Stage Six: Retrospective ............................................................................. 24

**Self-Reflection** ..................................................................................................... 26

**The Future of LOAD** .......................................................................................... 30

**Works Cited** ....................................................................................................... 32

**Appendix** ........................................................................................................... 33
  - LOAD: A Framework For Software Development Success ....................... 34
Process Analysis Statement

The processes followed, as partially outlined in the thesis itself, diverge from that of a traditional research paper. As the idea evolved, I found myself creating something between a research-based paper and a multi-tiered thought experiment. Of course, the final project is tangible in the form of a document, but the actual result of using the framework produced would be an abstract experience.

I started the year by researching various fields on the topics of success, decision making, and computer science. I then furthered the research by finding accounts of connecting these areas together. Specific research was also completed throughout the year for use of specific examples within the paper itself, such as my look into Object-Oriented Programming, and clean coding practices.

With a foundation of research in place, I began to determine the necessities of a viable framework by starting with what I knew. This felt tedious at the time, because it seems self-explanatory that any origination framework should have instructions, or an audience, but determining the building blocks of such a framework provided a much-needed sense of feasibility – the idea that a framework like this could exist.

With the knowledge that such a framework was indeed possible, I embarked on a journey to create the framework. I had prepared for this by creating another software development framework for my place of work. The trial run was much shorter, and targeted for a specific audience, however it gave me the knowledge of the do’s and don’ts of framework building. I was able to create a framework with the user in mind, predicating in what ways a user would need guided and when the user would know what the next step was. Out of this process I am confident in my ability to
create future frameworks in situations where the computer science community has not already created one.

With the framework created, I took a detailed look into its benefits and detriments. This was a particularly challenging part of the process. Typically, you are trying to defend your work constantly throughout a written work, however I thought it was necessary to play devil’s advocate for my creation, as even popular frameworks in the computer science community are highly contested and their usefulness debated. I learned that self-criticism, when administered moderately, can be healthy. Even though I was aware of the framework’s flaws, I knew from the beginning that creating a perfect product on the first try was highly unlikely.

Finally, after knowing the flaws with the framework, I was able to consider the next steps. Usually, I would not write a paper that required additional work. I am typically not fond of writing, and loose ends are frustrating to me. Picking a topic with the knowledge that it would very likely not be perfect at the end allowed me to be okay with many parts of the process that I would otherwise not be. My first paper in high school was around eight hundred words, and I nearly fainted when getting the prompt outlining the word requirement. My longest formal writing before this thesis was roughly fourteen pages. Simply put, writing is usually a chore to me, but that was not the case for this thesis.

Although it may be my longest formal paper, it has been one I have dreaded the least. Of course there have been many late nights, revisions, changes in timeline, and complete refocusing, but at the end of the process, this year-long, several thousand word writing felt no more taxing than that eight-hundred word paper in high school, or that fourteen page paper that now comes in at a distant second for length. It is something I have worked hard for and am proud of.
Introduction

The idea of creating order from chaos spans hundreds of human generations. The field of science is often thought of in this manner -- a process of defining the world around us that seems random to the naked eye. The concept of organization dates back thousands of years, even making appearances in one of the most influential writings of all time, the Bible. In 1 Corinthians 1:14 Paul the apostle writes, “Let all things be done decently and in order” (King James Version). The natural drive for order combines with the natural human desire to achieve and be successful, running society, and advancing our capabilities over time.

One of the most recent and important technological advances in the span of human history is the invention of the computer. Therefore, it is no surprise that the computer science community has also adopted the desire for order and continual success. Every development project embarks with a set of goals to guide it, whether it be to please a client, learn a new skill, or even just experiment with a new way of doing things. As projects grow in size, the need for dedicated people to organize a group of developers increases, leading to the rise of positions in the industry such as project managers, or senior developers.

In cases where a management role is needed, there are obvious examples of appropriate and inappropriate methods to organizing a development team. For example, it would be appropriate to set deadlines based on a forty-hour work week, but it would be inappropriate to expect developers to know what these deadlines are without being told. Not everything in computer science is as easy to identify, however. What service should we host the app on? Should we use JavaScript, PHP, Java, or C# for our backend? Should we use a Waterfall or Agile development method? All of these questions may have a correct answer, but this answer can be difficult to discover, especially in a timely manner.
Companies often rely on management systems such as SCRUM or Extreme Programming to help alleviate the burden of answering these tough questions. A system like SCRUM heavily structures an agile ideology so that the next step is always known, and all work is accounted for. While these systems may not answer all of the tough questions, they do cut down the number of decisions significantly, allowing the users of these systems more time to handle project management tasks such as client communications, timeline creation and management, and checking in with the development team.

Although many similar systems exist, there is a distinct void in the marketplace of development management systems. Because tough questions are not easily answerable, many systems restrict the user by making choices for them to create free time, but what if there were a framework that was designed not to make choices for the user, but to prepare them to make the tough decisions, saving time by shortening the decision process? Such a framework could not guarantee successful decision making, of course, but could be oriented in such a way as to maximize the amount of success the overall project has.

The purpose of this thesis is to explore the feasibility and structure of a framework geared towards maximizing software development success. By studying the components of success, itself, and what the computer science community has already created, we will understand the necessities of such a framework. With this knowledge, the framework will be constructed and presented separately. An analysis will follow, pointing out observations made through partial testing conducted in pieces through outside experiences.
Discovery and Analysis

Defining Success

The first step in defining a framework for success is to understand and define success. Naturally, a good starting place is the dictionary. While such ideological and trivial concepts such as success are often too generalized in a dictionary to use solely as our definition, it provides a good anchor point to understand the components of success. Merriam Webster dictionary defines success as, “favorable or desired outcome” (“success”). Dictionary.com defines it as, “the favorable or prosperous termination of attempts or endeavors; the accomplishment of one's goals” (“Success”). Finally, the Lexico Dictionaries defines success as, “an accomplishment of an aim or purpose” (“Success: Definition of Success by Lexico”)

These definitions, although broad, provide us three pillars of success: favorability, accomplishment, and outcome. These pillars also come with challenging questions. First, what does it mean to be favorable? Favorability implies an audience and therefore success, too, must have an audience. An audience in this context may be one or many people, groups of people, or even a situation. For a task to be favorable it must be beneficial or pleasing to said person or situation. Tasks often have many audiences such as the client and the development team, therefore many types of success can exist for a given task and may be in conflict with each other.

Secondly, we must determine what it means to accomplish a goal. Simply completing or not completing a given set of criteria is an obvious way to accomplish a goal, but along with accomplishment comes definition. For a goal to be accomplished, the goal must first be outlined prior to the work of completing said goal. This is important because in order to determine the success of a project, goals must be outlined before the work begins. In order to have a truly
successful project, you must also outline your goals for the project. A further question then is can
goals be changed throughout the process? We change our outlook on situations frequently but
allowing for goals to be changed at any given point presents a juxtaposition. If we can change
goals at any point, then to be successful couldn't we just change our goals at the last minute to fit
the outcome of our situation?

Although goals in our daily lives are often changed, for a framework to exist to ensure
success, goals that are a part of the framework must be static, otherwise the framework would be
quite simple: do whatever you want, and right before you're done set a goal that fits what you've
done. For our purposes, we must encourage that goals be crafted with a sense variability. For
example, rather than setting a goal of “writing 100 lines of code a day” our goal should be “write
a significant amount of code a day; a significant amount of code should take between fifteen and
thirty minutes for a peer to thoroughly review it”. Using a dynamic approach to goal setting
allows for our goals to adapt overtime without having to rewrite the goals. Therefore, you may
craft goals before the work, have those goals change, and still be able to evaluate how successful
you were at accomplishing the goals at the end of the process.

Third, we must determine whether success is binary or if it is variable. Tasks can be
partially accomplished but does that nullify the success of the task? For example, if you go
shopping and only buy nine out of ten miscellaneous items on your shopping list, was the trip
unsuccessful, or 90% successful? Are these two definitions the same or different? For these
statements to be equivalent would be to say the full completion of a goal is the only way to be
successful. In this case a politician who only secured 90% of the vote would be considered
unsuccessful despite winning the candidacy. To say success is binary denies the ability for two
events to be different levels of success. Similarly, under a binary presumption an event with the
goal of raising money for charity that only raised one dollar would be considered as successful as
if the event raised one million dollars, even though a greater sum of money is certainly more
beneficial to a charity than a lower one. Based on this, and our knowledge of favorability,
success must be variable.

Lastly, we must understand why success is defined differently by different people in the
same situation. If we look at a traditional client-developer relationship, the goals set out by the
client will look much different than the goals set out by the developer. The client of course wants
a reliable product that is easy to use and fulfills their needs. A developer may have a goal to
appease the client's needs, but will also have more granular goals, such as goals oriented towards
personal improvement. The client likely does not care whether or not the developer improves
their skill set through building this website however, it is nonetheless a goal of the developer that
must be completed in order for the project to be successful from the point of view of the
developer. Similarly, the developer does not necessarily need the goals of the client to be met for
the project to be successful. Failed projects are often a huge source for learning and self-
improvement. From a client perspective, if the project is not done, that project would be
unsuccessful, but from the developer’s point of view not finishing a project due to something
beyond their control such as financing may not affect their view of the project success. Because
of this, we know that success for a project is different for every person that is a part of the
process.

With these questions answered, we will formally define success for the purposes of
building this framework as:

A measure of how much a predefined set of goals are accomplished from the
viewpoint of those who created said goals.
This definition, of course, requires that any given project may be comprised of many different successes, and therefore requires a framework with the intent to ensure the greatest amount of success to account for the possibility of many successes.

What exists already?

Now that we know what it means to be successful, it is important to understand that the computer science community has already created systems and frameworks to organize and unify developers with the goal of being successful. Of course, because there are multitudes of ways of solving any given problem through code, there are also multitudes of ways of organizing developers, both big and small, but these systems generally fall into two categories: ideological and practical. Ideological systems, such as object-oriented programming, present the developer with a set of guidelines to structure their project without specifying what lines of code or other creations should specifically be used. Practical systems, such as clean coding guidelines, specify instances of good or bad code or other work. Understanding how and why these systems are built will give insight into what makes one of these frameworks successful. Analyzing these two frameworks will provide a sufficient foundation for the composition of a like framework due to their generality and ability to cover a wide set of use cases, just as the proposed framework of this thesis will also intend to do. For each of these systems we need to answer a compositional set of questions in order to understand why these systems are used and why they work.

First, we must ask what problems these systems intend to solve. While object-oriented programming does have an organizational advantage to a more functional programming style, it fills the void of a universal system to map real-world situations into manageable chunks of code, while at the same time accounting for code redundancy and efficiency. The four principles of
object-oriented programming (encapsulation, abstraction, polymorphism, and inheritance) are defined as such to allow developers a comprehensive guideline for how to break down the situation without mandating a particular implementation style (Pushkar, “4 Pillars for Object Oriented Programming.”). This allows two different developers to understand each other's work without necessarily knowing the bigger picture. This is especially helpful when seeking advice from others. Similarly, clean coding practices offer a set of guidelines for specific instances of code with the intent to make another person's code more readable and understandable even without context.

The idea of clean coding principles stem from the all too familiar situation in which we go back to read code written long ago and have no idea how it works. Although there tends to be some debate on these principles there are many which developers agree upon. Things as simple as correct variable naming and proper indentation mix with more complicated principles such as decreasing the use of getters and setters, reducing the use of comments through more readable code, and using the “need-to-know” rule for control access modifiers. Although these principles may look slightly different depending on which language you are using, a concrete example can always be provided for each given principle making the system easy to implement.

Secondly, we must identify any disadvantages or tradeoffs for each of these systems. For object-oriented programming, the main downfalls have to do with the amount of structure required to adhere to object-oriented principles. Although a large system that is already object oriented allows for easy additions, starting a new object-oriented system can come with considerable overhead, making it undesirable for projects with a shorter timeline or smaller scope. Clean coding principles, on the other hand, have relatively few downfalls, the largest being the subjective nature of the principles themselves. This subjectivity allows for developers
to create different definitions of what is considered “clean” and therefore detracts from the unity that the system sets out to create.

Third, we must identify any prerequisites or assumptions that are made in these frameworks and analyze how those prerequisites and assumptions shape the framework. The sole assumption of an object-oriented paradigm is that the language being used supports all four principles of object-oriented programming. This means a purely functional language could not truly be used under the object-oriented standards. This assumption allows for object-oriented programming to be rather strict and therefore work well in equally strict languages such as Java or C#. Essentially, object-oriented programming assumes you have all the tools it needs in order to work and forces you to use those tools and only those tools. Clean coding principles are unique in that they do not have any assumptions, because each principle can be explicitly defined and therefore does not have to assume anything. This allows for freedom when creating a principle, but also means each principle must be extensively designed, accounting for every situation.

Lastly, we must understand what a successful project using these standards looks like. For ease of analysis, we will consider what the ideal project using both object-oriented principles and clean coding practices would look like. This is especially handy since there are many clean coding principles that can be applied to object-oriented design. It is also important to note that for this imaginary project we will define success only in relation to the frameworks so as not to confuse success of the project with successes of the frameworks. The example is shown below.
abstract class FormalThesis {

    enum ThesisStatus {
        IN_PROGRESS,
        SUBMITTED,
        REVISIONS,
        APPROVED
    }

    public String title;
    public String content;
    protected ThesisStatus currentStatus;

    public String getStatusUpdate() {
        return "The thesis is called "+this.title+".\n"
               "The current stage is "+this.currentStatus+".\n"
               "The current content is: \n\n"+this.content;
    }

    private abstract void publish();

}

Snippet 1.1

class BallStateAttendee {...}
class Undergraduate extends BallStateAttendee {...}
class FacultyMember extends BallStateAttendee {...}

Snippet 1.2

class SeniorHonorsThesis extends FormalThesis {

    private Undergraduate author;
    private FacultyMember facultyAdvisor;

    ...

    @Override
    public String getStatusUpdate() {
        return super.getStatusUpdate()+"\n"
               "The author is "+this.author.name+"\n"
               "The advisor is"+this.facultyAdvisor.name+"";
    }

    ...

}

Snippet 1.3
The above excerpt of example code, drafted for the sole purpose of analysis in this thesis, demonstrates a strong object-oriented design, and follows clean coding standards for class-based code. The snippets demonstrate use of each of the four object-oriented principles. The presence of the "abstract" keyword within Snippet 1.1, ironically, concretely represents the idea of Abstraction through code. The object FormalThesis is self-aware that all formal theses must have the ability to be published, yet the publishing process will look very different for each instance of a formal thesis. Inheritance is demonstrated with the use of the keyword extends as seen in Snippet 1.2 and Snippet 1.3. These extensions fall in-line with the “is a” principle from clean coding practices -- that is to say that an Undergraduate is a BallStateAttendee, a FacultyMember is a BallStateAttendee, and a SeniorHonorsThesis is a FormalThesis. Encapsulation is demonstrated through the use of the control access modifiers public, private, and protected. These modifiers restrict access to an object’s data in various stages, allowing for implementation to be accessible only by those parts of code that need access. Lastly, Polymorphism is present in Snippet 1.2 when both Undergraduate and FacultyMember extend from the same class of BallStateAttendee. This would allow for instances of both classes to be combined and worked with together, even though they are different subclasses. The simplest example of this is making an ArrayList<BallStateAttendee> object that contains instances of both Undergraduate and FacultyMember.

From a clean coding perspective, the snippets demonstrate many common practices that contribute to the overall code’s quality. The first, and arguably most trivial, is the snippets use of indentation and spacing. Below is an alternative version of Snippet 1.1 that performs functionally identical, yet is understandably inferior, since the benefits of the shortened format are far outweighed by the detriments to readability.
abstract class FormalThesis {
    enum ThesisStatus {IN_PROGRESS, SUBMITTED, REVISIONS, APPROVED}
    public String title, content; protected ThesisStatus currentStatus;
    public String getStatusUpdate() { return "The thesis is called "
    +this.title+". The current stage is " +this.currentStatus+". The current content is: 
    " +this.content; }
    private abstract void publish();
    ...
}

Snippet 1.4

Boiled down, the difference between Snippet 1.1 and Snippet 1.4 is that, while both may make equal sense to the computer, only Snippet 1.1 inherently makes sense to a human. This is a prime example of clean code’s mandate that code not only be efficient and make logical sense but make sense to others who may read the code. The adage that “code should be clean enough that someone who has never programmed a day in their life can look at it and know what it’s doing” comes from this mandate. We can see a difference when this notion is pushed to the extreme, as shown below with two JavaScript functions with identical output.

```javascript
function factorial(number) {
    if(number > 1) {
        return number * factorial(number - 1);
    } else {
        return 1;
    }
}

Snippet 1.5

f=n=>n*(n>1?f(n-1):n)

Snippet 1.6

Better error handling disregarded, Snippet 1.5 and Snippet 1.6 are functionally equivalent, yet the latter will create feuds and uncomfortable performance reviews. Clearly it is inherently
worse, meaning, unlike Snippet 1.1 and Snippet 1.5, it is an unsuccessful snippet of clean code. We can specifically list the errors in Snippet 1.6, such as: non-descriptive variable names, use of a “trick” rather than proper logic, non-explicit base case, and a nested ternary. Pinpointing these violations is what allows us to determine how successful each snippet is. Similarly, since we could point to the proper usage of object-oriented tactics, we could surmise that Snippet 1.1, Snippet 1.2, and Snippet 1.3 are quite successful at using object-oriented design. Therefore, the framework for development success can also be evaluated by participating in a retrospective review at the end of the project, as well as periodic reviews throughout the process. Such a review will not only provide insight on whether or not the framework is working as intended, but whether or not the team adhered to the steps of the framework in its entirety. A retrospective provides a natural feedback loop that the framework should take advantage of.

Framework Architecture

With an understanding of the challenge at hand, and the computer science community’s existing attempts at solving similar problems, we can begin to plot out the structure of the framework for success. The framework, entitled LOAD, will consist of six large portions, called stages, each broken down into smaller, strictly designed and templated components. LOAD stands for “List Organize Advance Discover”, which describes and groups the stages, which are named: Define Success, Create a Plan, Follow the Plan, Change the Plan, Finish, and Retrospective. Although not all six stages will have the same weight or time commitment to the overall project, each stage is necessary, and no more stages are needed since the framework covers the broad scope of any given project.
Stage one, **Define Success**, derives from the integral exploration of success and its meaning. The user of the framework, which could be one person, or many people, will be presented with a description of success, examples of good and bad goals, and any other tools needed in order to create and solidify any number of goals that can be used to determine the success of the project. The users’ goals will be written, checked against the criteria, and documented for later use. Just as it was important for the purposes of creating the framework to understand success and its many facets, so too is understanding how success should properly be measured from the users’ perspective. The definition created will feed into, and influence all of the other five stages.

Stage two, **Create a Plan**, will walk the user through converting their goals into a workable strategy for accomplishing said goals. Portions of this plan may include, but are not limited to: a timeline of milestones and deadlines, who will work on each task or group of tasks for a particular milestone or time segment, what compromises will be made for the success of the project as a whole, prerequisites for any portion of the project, points of no return, and reporting roles of all group members. Although stage two will give suggestions, it will also denote how to create an unusual or atypical plan for projects facing extraordinary challenges.

Stage three, **Follow the Plan**, is just that, an entire stage dedicated to enacting the plan created in the previous stage. Although this stage allows for a system such as Agile, a sprint-based, iterative process, or Waterfall, a linear, start-to-finish system, to step in and take the reins, it will guide the user through steps to keeping on track. After all, systems like SCRUM are only as good as your weakest link, and often roadblocks can be avoided with preventative measures. Such measures may be outlined in stage two, however stage three is dedicated to ensuring the user enacts these measures so as not to fall behind.
Stage four, Change the Plan, may sound contradictory, with solidity in the previous stages having vast importance, but this stage recognizes the possibility that a user may be wrong at the start of a project. For example, mandatory peer review via pull requests may solve the issue of having clean and readable code, but may also cause a bottleneck due to developers’ schedules not lining up properly. In this instance, for the sake of finishing the project on time, pull requests may need to be removed from the plan. Although it would be ideal to skip stage four altogether, it is inevitable that every project will go back and forth between stage three and stage four multiple times throughout the project process.

Stage five, Finish, is dedicated to wrapping up the project process. This includes delivery of the project, evaluation of success, and overall project completion. This should be the point in the process where the final handoff to the client is made, and any future work done on the project is done on a case by case basis. This is not when the project is first used, however. In the event that the product undergoes a trial period, or a beta release, stage five should not be started until that trial or beta is finished, and no more changes are required before the full release. It is also important to note that this stage will include the client version of stage six, Retrospective, as stage six is solely to occur after client communication has ended and the project is completely hands-off, since working with a client is in itself a portion of the project.

Finally, stage six, Retrospective, is the last stage of the framework, intended to finish the current project, and lead effectively into the next one. In this stage, the user will be directed on how to create an evaluation procedure that each member involved, including the client in stage five, will take part in. This procedure may take the form of an in-person meeting, online form, both, or something in between. The procedure may look different to those who had different roles on the project. The evaluation should take into account the evaluation of success created in
the previous stage and should be geared towards highlighting parts of the process that need revised. With this feedback, the next project to use the framework can take into account what needs changed in order to avoid obstacles in the future.

**Constructing the Framework**

**The Name and Forward**

The forward for the LOAD framework is intentionally light, due to the lengthy descriptions located at the beginning of each stage. A brief overview of LOAD’s meaning and the title of each stage is given, along with a general statement of LOAD’s purpose for being created. A note about first time user’s reading the document in full is given, as well as general guidelines for how to start using LOAD for a new project. Lastly, a graphic, shown below, illustrates the LOAD acronym and how it breaks down into each of the six stages.

![Image 2.1](image)
Stage One: Define Success

The first stage is arguably the most important for the framework document itself. Creating a list of goals is the foundation of the framework, and since the user of the framework will not necessarily be privy to the decomposition of the requirements of this stage as stated prior, it is vital that the user take this stage seriously, and performs the first step accurately. As can be seen on the first page of stage one in the framework document, a brief description provides the user with a sufficient amount of background knowledge on the fundamentals of success and its vital role in the composition of the framework itself. The main takeaways, bolded in each paragraph, ensure the user creates an exhaustive list of goals that are SMART, variable, and audience-oriented (Doran, “There’s a S.M.A.R.T. way to write managements’ goals and objectives”). The description for stage one also includes examples for those concepts created for the sole purpose of the framework, thereby implicitly explaining the meaning of these terms and reasoning for their importance.

The following page consists of the “Define Success Worksheet”, where users will list out the audience for each goal, and of what the goal consists. This page may be duplicated in the event that the space provided contains too few lines. It is also important to note that, while this page utilizes the built-in table feature, it is designed in a default manner so as to allow for the page to be printed and written on.

Stage Two: Create a Plan

Planning is no doubt an important part of any project, but there is a sweet spot for the amount of planning any given project requires. Too little planning can lead to confusion and
inorganization, whereas too much planning will not allow for enough time to finish a project by the hard deadline. Stage two is geared towards finding a happy medium between the two by using the assumption that the more goals there are to complete, the more time should be spent planning.

The device used to ensure an approximately linear relationship between the number of goals and planning time is, “We will accomplish [the goal of] ___goal___ by [doing] ___task___,” where the first blank is the goal or goals accomplished by the second blank, which is a task or tasks. This process is approximately linear, since for \( n \) goals, approximately \( n \) statements will have to be created.

With this device in the readers’ minds, a few pieces of information are requested to be written on the following worksheet. The user should choose a development pipeline, a standard for how the team should operate, a decision-making hierarchy, guidelines for team members, and a general timeline for the project. Figuring these pieces out are a necessary step for any development project, and the outcome of these decisions will affect the list of project tasks created in the next worksheet. Placing the fill-in-the-blank format before these decisions reminds the user that these decisions should be made in response to the goals, so as to prepare the team with the best tools for getting the job done.

Having completed the first stage two worksheet, a description of the second stage two worksheet is given, providing more specific context as to how to come up with these fill-in-the-blank style tasks. Notably, this section talks about making these tasks all take roughly a week, and how to conform larger and smaller tasks into this format. Finally, the benefit of week-long tasks is explained, and the worksheet is presented.
Stage Three: Follow the Plan

Due to abstraction’s non-specific nature, this stage is among the least concrete. Without knowing the context of the project at hand, or the product of the first two stages, stage three is crafted in such a way as to ensure the user is accountable, prompting the user with questions to answer and in turn routing them to the next step. The introductory paragraph explains this concept. The rest of the description of stage three focuses on three avenues that the user could end up in: the first where no changes are needed, the second where moderate changes are needed, and the last where structural changes are needed.

In explaining that it is possible to go through the development process without any bumps in the road, the framework clearly outlines that this scenario is rare -- not to be expected -- but also not to be concerned if it should happen, especially on projects of smaller scope. The framework continues with the much more likely situation, that changes are needed in the process. Although the framework aims for this situation not to happen, it is inconceivable that the framework will work perfectly for every person in every situation, one hundred percent of the time. The description also explains what to do in the state of emergency, such as during a prolonged social crisis.

It may be necessary to move back to stage one during stage three. As any seasoned developer knows, clients rarely have a true idea of what they want until they see something they do not want. In this case, or in the event of a major unforeseen change in timeline, the definition of success may need revisited. Although the ideal definition of success includes variability, there are moments in life that may drastically change the course of a project, such as office-wide sickness, public social disruption, acts of God, or other extenuating circumstances. In these
times, a redefinition of success may be necessary, in which case stages one and two must be completed again.

The document then instructs the user to pick a timeframe for how often to fill out the accompanying worksheet. Although this is left to the discretion of the user, this timing should be reasonable, occurring several times throughout the process, assuming the length of the project is long enough to accommodate more than one check in.

The worksheet for stage three formats the aforementioned scenarios into groups of questions such that, if the user answers “yes” to any question within the group, they can proceed to the given stage. Essentially, the group of questions are designed in such a manner so as to help the user understand which scenario they are currently in. Over time, a user may not need to explicitly answer each question in order to understand which scenario he or she is in, however, completing the worksheet each time will serve as reference for how the process evolved over time.

Stage Four: Change the Plan

The fourth stage guides the user through making changes to the process they have created, while noting that this stage should be avoided if at all possible. Changing the plan inherently means that something, even if beyond their control, has gone wrong. The other key aspect of stage four is that the user should strive to exit the stage as quickly as possible, as time spent fixing mistakes is time taken away from producing a final product. With these stipulations laid out, the idea of strategic solutions is introduced as explained.

Stage four’s description and accompanying worksheet focus on making changes based on how and why the user ended up in stage four in the first place, with the idea that understanding
the route to stage four will allow the user to identify mistakes before they lead to a change needing made. As the project progresses, stage four should disappear, and visits, if any, should be short, since solutions to previous problems will be purposeful. The description offers three general form factors for changes: a new solution, testing a solution, and adapting a solution.

With a new solution, the user will implement something completely different to what is already in place. This instance is reserved for a situation in which the solution is obvious and can be implemented with little friction, otherwise testing a solution would be more appropriate. These two options come with caveats however, whereas a new solution is quick, it is not well tested, and likewise, testing a solution is less of a risk, but at the expense of time. This is why the third option, adapting a solution is provided. Taking the existing solution and tweaking it may not be perfect, however it may be timely. The worksheet provided asks three questions to help the user determine what avenue works best for each particular problem.

The first question is, “why did this problem occur?” Understanding why problems happen helps to prevent them from happening in the first place. Similarly, the second question is, “how did this problem occur?” The key difference between these questions is that the “why” is the trigger to initiate stage four, and the “how” are the events causing said trigger. It is not simply enough to know one of the two. For example, if unclean code caused a significant number of bugs, the “why” would be that unclean code caused a significant number of bugs, whereas the “how” would be that the review process was not strictly enacted. The problem cannot be fixed without knowing both of those pieces of information.

Lastly, the question is asked, “what change will you be implementing to avoid returning to stage four?” Notably, the question is not the more natural, “what change will you be implementing in order to solve this problem?” The intent here is to guide the user towards not
only fixing mistakes, but strengthening their decisions. What is the use of a new solution if it is destined to need replacing again? Phrasing the question in this manner is a subtle reminder to be both backward and forward thinking while creating a solution to a problem.

Stage Five: Finish

   The document compares stage five to a marathon runner, claiming that the runner is not done with the race when crossing the finish line, but rather when they stop running after having crossed the finish line. Suffice it to say that stage five is to allow for all loose ends to be tied up. The main purpose of this stage is to ensure the client evaluates the project as the last part of their involvement of the project, so as to receive an all-encompassing response. The document outlines that this stage should only be started when there is no work left to be done on the project for the foreseeable future, and that this stage does not end until after the client evaluation is returned.

   The worksheet for the client evaluation consists of five simple questions, including the common question, “On a scale of 0%-100%, with 0% being ‘not successful at all’ and 100% being ‘successful in every aspect’, what percentage of success would you say the project had and why,” which appears on every evaluation worksheet. The questions are few so as to receive a response more quickly from the client. The worksheet does not have the quantitative section that will appear on the other evaluations not only for a timely feedback, but because that section is directed towards the LOAD process specifically, which the client has no direct interactions with.
Stage Six: Retrospective

Finally, stage six focuses on reflection, analysis, and preparation for future projects. The first portion of the description is quite simple, briefly explaining that there are two more evaluation worksheets: one for team members, and one for project managers. These worksheets are very similar in concept to the client evaluation, however they have ten short answer questions, and ten qualitative questions in which the answer for each is a number one through five, where one is “strongly disagree”, and five is “strongly agree”.

The short answer questions are written to gauge how the LOAD process was carried out and how it affected the final product. They also include the common question, asking each person what percentage of success they felt the project had. This will be compared to the success index, which is a calculated value from all of the respondents’ data. Not only will the answers to these questions be useful on their own, but the user will be asked to compare and contrast the answers to identify patterns.

This brings us to the final description and accompanying worksheet, the retrospective analysis. The retrospective analysis is a formalized summarization of the client, team member, and project manager evaluations, including calculations to compute different success indexes, a value calculated by finding the percentage of points earned by the quantitative questions. The user must also answer a set of culminating questions and draft a summarization of the results for use on future projects.

The description also stresses the importance of this final stage. Although this stage could be skipped without consequence to the current project, the data obtained by this stage is invaluable for moving forward with the LOAD process in the future. Learning from past
mistakes will better prepare the user for the next project, avoiding stage four, and therefore saving time and resources.

The first section of the Retrospective Analysis Worksheet is to compute three success indexes. These are obtained by first computing the average score for all team member evaluations and for all project manager evaluations. Since the possible average for each is a value between ten and fifty, that average is extrapolated into a percentage by shifting the range down by ten, to between zero and forty, and multiplying by a constant of five over four to give a percentage of total points earned for team members, and for project managers. The overall success index for the project is simply the average of the success index for team members, and for project managers. The overall success index can be thought of as the mathematical percentage of how successful the project utilized the LOAD process, however, it is important to note, that the success index does not necessarily imply the success of the project itself. Whereas these two values will often be similar, it is entirely possible that a very successful project was very unsuccessful at utilizing LOAD or vise versa.

The second section in the retrospective analysis worksheet is the quantitative analysis, where the user answers ten questions based on the one through five scores. These questions focus on trends in the data such as consistently high and low scores, conflicting scores, questions scored the highest and lowest, evaluations with consistently low scores, and what things need to be changed or kept the same in response. The tenth question asks if the results were surprising, since reality can often be jarring. Similarly, the third section asks questions aimed at identifying patterns in the short answer questions of all three types of evaluations. The methodology of this section is entirely the same as the last.
Finally, the last item of the framework is the retrospective summary. This is an open-ended page for the user to summarize the results of the first three sections of the retrospective analysis. The purpose of the summary is to act as a self-contained snippet of information detailing the brief history of how the LOAD process was utilized for the project. The summary should be roughly one page, so that a passer-by could use the summary as reference for what to do, and what not to do for a similar project in the future. This summary, along with all other worksheets will be saved together for future reference, but the summary should serve as an abstract for the collection of worksheets.

Self-reflection

Having constructed and analyzed the LOAD framework in its entirety, combined with the personal experience of piloting many concepts of the LOAD framework within my job and capstone course of which I am partaking in partial fulfillment of the requirements to finish my bachelor’s degree in Computer Science, I have compiled a list of inferences about the final product and its potential future within the Computer Science community. From my experience, the LOAD framework excels in many aspects, however its abstract construction inherently suffers from a lack of specificity, requiring additional processes and frameworks to be used in conjunction with the LOAD framework for certain use cases.

The two sources of practical experience the LOAD framework has received were both received in part, as the entirety of the framework was derived from these trial runs, rather than the other way around. The first, via my capstone group, focused on the aspects of the framework that affect the team members. Although the course had its own requirements for keeping track of
the project process, there was enough leeway that the middle stages of the framework were able to be tested, specifically stages three and four.

At first, the concept of stage three was quite simple, requiring the user to self-determine if changes needed made, without the use of provided questions. Whereas this worked well enough for a group of four people who have worked with each other for several years, we found ourselves in a typical rhythm, essentially asking ourselves the same set of questions over and over again. This realization translated into the formally defined questions present in the framework as it is now.

Stage four was adapted in a similar manner, only existing as a caveat in stage three initially. Of course, we had our share of problems working through our eight month long project, but recording those problems, and our changes to the process was introduced and put into its own stage when realizing that there was no record of the changes at the end of the process. In our short-term project, there was no need to keep record of these, however, if we were to start a new project, or work on a longer project, having a history of solutions to look back on would have been invaluable. Therefore, stage four was created, and a proper record of the processes problems and solutions to said problems was implemented into the framework.

These experiences justified the inclusion of the worksheets for stages three and four, and brought up the issue of specificity. Because LOAD is supposed to service any development project, there is an artificial limit on what it can and cannot define and describe. This was a challenge, but a worthwhile one. I have found that, while LOAD may not be able to plan specifics about a project, it could be used in many different contexts aside from computer science.
There is no reason that LOAD could not be easily adapted for use in other fields of study, such as video production, graphic design, or even just a large home improvement project. The abstract nature of LOAD, aside from the occasional development-oriented example, makes it useful for any project with a similar life cycle, however, my experiences at my place of work indicate that LOAD may not be useful for projects of vast scale, with multi-faceted overarching structures. For example, whereas LOAD may not be sufficient to organize the entire development process for a AAA video game title, it could be useful for a particular team or portion of the game.

Similar to my experience testing LOAD within my capstone group, I was limited in my ability to test the framework in its entirety through a Ball State project. In place of this, however, I developed a similar, much smaller framework called Spread that was an Agile-Waterfall hybrid. Although Spread has a much different purpose than LOAD, I was able to make key inferences about a user’s response to following a framework, how a framework’s authority is respected by its users, and what a user will inherently do without being instructed by the framework. All of these inferences contributed greatly to the format and success of LOAD.

Firstly, I noted that the team members for the experimental project grew frustrated when Spread was vague or open ended. Spread’s instructions were intentionally laid back, so as not to suffocate the decision-making authority of the project managers, however, this “up to you” attitude left the users confused and unsure of their actions in respect to the framework. Therefore, in the LOAD framework, decisions are left to the user, but what decisions to make are strictly given by the framework. I found that users felt much more confident when making decisions when they were given the choices to make, rather than allowing them to seek out choices.
Secondly, I found that a user, although having no inherent reason to, will trust the framework even if they have never used it before. The Spread framework was an honest attempt to try my hand at crafting a framework from scratch, and although I had the research to back up my decisions, I had no sense of credibility from the point of view of my peers adhering to the guidelines of Spread. They did not have the luxury of reading an accompanying document explaining why the framework was designed in the way it was, yet they assumed its correctness, even when following the first draft of the framework led to more frustration. This realization meant the LOAD framework needed to provide its justification for each piece of the puzzle as each piece was unveiled. This was the influence that added the note about first time users reading the document in its entirety before using the framework. Without this transparency, the framework could have still been successful, but would have had no foundation to convince people of its usefulness.

Third, creating the Spread framework showed me what actions users will take without being instructed to do so. The short answer is that the user will panic. Maybe something would get done, or maybe not, but panic was certain. This is why the LOAD framework is devoid of unexplored paths. It was a point to ensure that there could be no “what if” questions as a result of using the framework. This makes LOAD much more flexible than a system like SCRUM, which requires a SCRUM Master to be familiar with all of these little details and to call the shots when necessary.

Overall, LOAD accomplishes its initial goal, but as we have learned, success is not binary. LOAD is not one hundred percent successful, if not because of any flaws, at least because of the lack of any framework’s ability to truly guarantee a successful project. From a statistical perspective it could be argued that LOAD’s percentage of success is the percentage of
projects using LOAD that turned out as successful as possible. Since this is not a measure that can be computed, we must resort to a more subjective system for accessing its level of success.

By far, LOAD’s largest issue deals with its scalability. Although LOAD is abstract enough to theoretically be used for a project of any size, six stages will simply not be enough for projects at a certain scale. As said earlier, a AAA video game would struggle to use LOAD as their main driver for a structured process. A project of that scale would stay in stage three for years, effectively nullifying the benefits of an enforced process.

LOAD is most successful when worksheets are being filled out semi-regularly. The advantage of this is that the user of LOAD is forced to re-evaluate the state of the project at regular intervals. If there are several months between these worksheets, the process is likely to die and be disregarded all together. This does not mean a company producing a AAA video game could not make use of LOAD, however. In this situation, individual pieces of the project could take advantage of LOAD. This could be the GUI programming team, or the game engine team, or a particular mission in the game. Smaller, yet still substantial, chunks of the final product could be mapped onto the LOAD framework and provide similar benefits and results.

The Future of LOAD

Although for the purposes of this thesis, the framework construction was merely a practical justification for the feasibility of such a framework, the final product is capable enough to be fully implemented, tried, and improved for future use. Moving forward, I will be in contact with my former place of work in hopes of piloting the LOAD framework similar to how the Spread framework I created for them was tested. With testing results produced by a full round of LOAD,
I will be able to make more inferences and justifications for the framework that a limited timeline has prevented.

Beyond the scope of computer science, I am interested in the benefits of LOAD in different industries. Collaboration with other representatives from each industry of interest would be required in order to tweak the LOAD framework to meet the needs of other workflows. I suspect that LOAD could become generalized enough to account for any project regardless of field, though I believe this generalization may require supplemental sub-frameworks or adaptations to maximize the amount of success a user will have using LOAD.

LOAD truly fills a void within the computer science community, aiming to structure the user’s thinking without making decisions for the user. Nevertheless, LOAD is subject to critique, just as every other similar system is. Many may be comforted by other frameworks that make the tough decisions for the user, but to those with an open mind, or a creative management style may find solace in the idea of taking complete control at every branching pathway. The ability of LOAD to integrate into a group project or production company largely varies on the ideals and flexibility of those who have the option to try it. For those who do give it a shot, there is, at the very least, the safety net of the self-improving system. Even though LOAD is young, the intentional cyclic process will feed continued growth each time it is used.

There is no getting around learning new skills as a computer scientist, as technology rapidly improves second by second. Learning from your mistakes is just as much an imperative skill to developers as learning how to write clean code, or how to properly use an object. Breaking down and understanding the individual components of a problem is the job of a good developer. So long as humanity has problems and computers, there will be developers there to translate those problems into lines of code, creating order out of the chaos that is the natural world which we call home.
Works Cited


Appendix

The following document is the culmination of the framework created under the specifications required.
LOAD: A Framework For Software Development Success

Welcome to LOAD: A Framework for Software Development Success. LOAD stands for List, Organize, Advance, and Discover. These words describe and group the six stages LOAD is separated into, Define Success, Create a Plan, Follow the Plan, Change the Plan, Finish, and Retrospective. LOAD is a pipeline designed to maximize success for any given project carried out by a development team. It uses templated worksheets with purpose driven decision making to clearly set out a path for success, make adjustments along the way, and learn from mistakes, creating an infinite cycle for both long-term and short-term successful development.

It is recommended that new users of LOAD read through and review the entire framework document prior to starting a project. Understanding the ecosystem of LOAD before jumping in feet first will prove advantageous and will likely answer many questions along the way.

To use the process, simply follow the instructions for each stage, filling out and collecting the worksheets along the way. At the end of the project, you should store your worksheets for future reference. The worksheets are designed to leave a historical context for future readers to learn from.
Stage 1: Define Success

The first challenge of any project is understanding what would make the project successful. Identifying a list of goals at the beginning of a project will create a foundation for all of your project management needs.

Keep in mind that success may look different to different people. Your developers likely don’t care if your client prefers video calls to phone calls, and your client likely doesn’t care if your developers opt for block comments as opposed to in-line comments. Nevertheless, isn’t the world better when everyone’s happy? Ensure that you define each goal, for all parties included.

You should also make your goals SMART and variable. SMART stands for: specific, measurable, attainable, realistic, and timely. Variable means your goal should not change for the duration of the project. For example, rather than having the goal of completing all frontend code by May 1st, have the goal to complete all frontend code in time to have two weeks of testing before final delivery. This way, if the client changes the deadline, or the development team catches a nasty virus and is out for a week, or something else short of “out of the ordinary” happens, you do not need to change your list of goals.

Use the following worksheet to create an exhaustive list of goals for the project, indicating the audience for each goal, i.e. who or what successfully completing that goal would affect. (I.e. The goal of “Finish the project under budget” would have an audience of “The client”, whereas the goal of “Make the client happy” would have an audience of “The Company”).
Define Success Worksheet:
Remember to ensure each of the goals below are SMART.

<table>
<thead>
<tr>
<th>Audience(s)</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stage 2: Create a Plan

With an understanding of what qualifies your team to be successful, you can strategically make a game plan for how the project should run. Your plan should be able to state, “We will accomplish [the goal of] ___goal___ by [doing] ___task___,” for each goal defined in stage 1. Some goals may be accomplished by performing several tasks, and one task may accomplish several goals.

In terms of a development pipeline, anything that fits the constraints of your project will work, however, humans are a creature of habit, and “doing things the way we have always done them” is NOT a valid justification for using one pipeline over another. Systems like Agile and Waterfall both exist because both have advantages that may suit one project, but not another. Choose, create, or adapt a development pipeline that will suit your project best.

Upon choosing a pipeline, create a clear standard for how the team should operate, and even more importantly, stick to it! Afraid of being too harsh? Build in a punishment safety net by giving people more than one mistake. Unsure of the best chain of command? Try it out anyways. If you make an educated decision, and stick to it, the pros and cons should be known ahead of time. A source of consistency will create a well-oiled machine, which is much preferred to a rusty machine that’s more efficient on paper. Just as fixing an engine takes time, so too does fixing a flawed system. The time you gain making changes to your process usually is outweighed by the time spent making the changes in the first place.

Use the following worksheet to describe a development pipeline, management / decision-making hierarchy, include reference to the determined timeline for the project and state any additional guidelines for the project. This template should serve as reference for your team members when they have to make a tough decision, or need to gain approval, and will serve as an official record of what is expected of the team members.
Create a Plan Worksheet:

Development Pipeline:
   Pipeline Process Description:

Decision Making Hierarchy:
   

Guidelines for Team Members:
   

General Timeline:
Stage 2: Create a Plan (cont.)

Now that you have a record of how things will be done, it is time to recognize what things will be done. Many may complete this part of the framework by using a scope document, feature list, or user stories, but to have a proper collection of all the work that needs done, you will need to compile a list of all work required to finish the project. This includes planning, wireframes, mockups, reviews, regularly scheduled meetings, client communications, delivery, testing, bug fixing, research, market studies, or anything else that contributes to the final product.

These tasks do not need to be granularized, however, they should be specific enough to indicate the general amount of time needed to complete. Think in terms of weekly tasks. “Add index.html to project repository” should not take more than a few minutes, whereas “write all code” will likely take months. A happy medium would be, “complete basic project setup along with unit tests for at least ten endpoints” Your tasks should take one person roughly a week to complete on their own.

Use the following worksheet to list out all week-long tasks required to complete the project. If a task does not easily adhere to the weekly format, either reformat the task, or group it with other tasks that will add up to a full week. This will help greatly in the timeline planning process. Note that this list can be used as a checklist throughout the process, but depending on your plan, it may be more useful to implement your own checklist system using a project management software, in person checks, or some other solution. This list will be a wonderful resource in creating a detailed timeline, determining deadlines, and assigning milestones, and will serve as a tool to ensure the project is meeting all requirements.
Create a Plan Worksheet 2

Project Tasks:
Stage 3: Follow the Plan

And now for the moment your whole team has been waiting for, stage 3, an entire stage dedicated to observing how your machine runs. **Although this stage has no direct action, the observations made during this phase may lead to action being taken in another stage.** Stage 3 works as a tight cycle: do work, review by completing the worksheet below, take action if necessary, and repeat until the work is done. Reviews lead to one of three scenarios. We will start with the most ideal scenario and work our way to the least ideal scenario.

In a perfect world your observations on this stage would tell you that you have mastered the LOAD framework, human interaction, and random chance. However unlikely this scenario is, **it is not implausible to think that a project with a small scope may run smoothly enough that no additional action is needed.** If you find yourself in this scenario by completing the worksheet below, pat yourself on the back and savor the moment: it’s likely to never come again.

**The most likely scenario, and moderately ideal one, is that your observations tell you that some changes in the process, goals, or tasks are needed.** The reason for these changes could be anything considered “usual” for a project process. Examples would be additional features requested by the client, client feedback is late, a team member’s laptop freezes up for a day, or your server hosting platform has an untimely outage. Of course, nobody expects an office argument to kill productivity, but when it does, it may be necessary to adapt.

**The worst-case scenario is all up to chance.** When a thousand-year flood prevents the entire office from working for two weeks, or when a laptop with several unpushed commits is stolen, or when a third-party company changes their API just after your developers finish implementing it, your goals and priorities likely will change. In this scenario you must realize that things are not “normal” and therefore normalcy in the process goes out the door.

**These questions should be reviewed at regular intervals.** You must decide what constitutes a review. Reviewing every milestone, week, every second week, and after a percentage of tasks get done are all examples of acceptable timing. Choose a timing that fits your needs the best.
Stage 3: Follow the Plan Questions

These questions will be reviewed after __________________________________________________________________________.

If the answer to any of the following questions is “yes”, double back to Stages 1 and 2 to reassess the goals of the project.

1. Have unusual circumstances beyond the control of the client or team made it so the purpose or goals of the project are notably different?
2. Has it become clear that reaching the project’s goals is highly improbable given the current circumstances?
3. Is the project being postponed or canceled for reasons beyond the control of the team?

If the answer to any of the following questions is “yes”, proceed to Stage 4, and determine a pathway forward.

1. Is the project falling far enough behind the planned timeline that the timeline has or needs to receive significant changes?
2. Has the scope of the project changed in a significant way?
3. Have any previously unknown bottlenecks emerged that are preventing your plan from flowing smoothly?
4. Has the project lost or gained members, significantly changing the timeline?
5. Is any part of the plan you created not working?

If the answer to the following question is “yes”, change the timing for answering these questions at the top of the page.

1. Would reviewing the stage 3 questions above more or less frequently be beneficial to accomplishing the goals of the project?

If you answered “no” to all of the questions above, continue with stage 3 or, if all work for the project has been completed, move to stage 5.
Stage 4: Change the Plan

Although LOAD has pushed for creating goals that do not change over time, it is simply unfeasible and irresponsible to work under the premise that anything we plan will be perfect the first time around. Similarly, life is too chaotic to completely plan for and thus, stage 4 comes into play. Stage 4 can almost be thought of as a stage 3.5, since it should be avoided if possible, and it intrinsically mirrors the first three stages. Nevertheless, if you find yourself in stage 4, as most teams will, you will need to strategically come up with solutions to the problems you are encountering.

The best questions to ask yourself at this stage is why and how did you arrive at stage 4? The difference between “why” and “how” in this situation is that the “why” is the cause and the “how” is the effect. For example, if unclean code caused a significant number of bugs, the “why” would be that unclean code caused a significant amount of bugs, whereas the “how” would be that the review process was not strictly enacted. Understanding the path to stage 4 will allow you to become more familiar with the warning signs and things to avoid while coming up with a plan. It should be your goal to be in stage 4 as few times as possible, and as short of a time as possible. It is a careful balance. Too much time thinking of a better solution, and you will spend too much time here. Too little time, and you will make too many trips here.

When you are comfortable with understanding how and why you arrived in stage 4, you must find a solution. For example, if you are behind schedule because the client takes twice as long to communicate as you planned for, you will need to reformat your timeline to account for this extra time, perhaps by sending the client twice as much information per email. This sounds reasonable on the surface, but you must also determine whether your solution will introduce more problems leading you back to stage 4.

In the previous example, sending the client twice as much information may work to your benefit, but the client could also take twice as long to review twice as much content, making communication even slower and more inefficient. Of course, if you can predict that this will not be the case (if you have worked with the client previously, or feel confident with their communication habits), then you can proceed. If not, you will either need a new solution, test the solution, or adapt the solution. All three choices can be beneficial in different circumstances. A new solution may be found with little additional problems. Testing a solution can be effective if time permits. Adapting a solution is always a fine compromise when no problem-less solution can be found within the time constraints.

Use the following worksheet to track your journey through stage 4. Not only will this serve as a record of your decisions, but it will be a valuable resource during the final stage, dedicated to evaluation of the process.
Stage 4: Change the Plan Worksheet

For each change, fill out the following information:

Problem number __1__ defined ( __1st__ time in stage 4):

Why did this problem occur?

How did this problem occur?

What change will you be implementing to avoid returning to stage 4?

Problem number __2__ defined ( ___ time in stage 4):

Why did this problem occur?

How did this problem occur?

What change will you be implementing to avoid returning to stage 4?
Problem number ____ defined ( ___ time in stage 4):

Why did this problem occur?

How did this problem occur?

What change will you be implementing to avoid returning to stage 4?

Problem number ____ defined ( ___ time in stage 4):

Why did this problem occur?

How did this problem occur?

What change will you be implementing to avoid returning to stage 4?
Stage 5: Finish

Congratulations on making it to the “final” stage! Admittedly, “Finish” for a stage that is second to last is a bit misleading but finishing something is usually not the last thing done. After all, before a motorsports race is finished everyone takes a lap before going into the pits, right? After a marathon the runner slows down after the finish line. Suffice it to say that while the project may be finished, the process is not. There are still some odds and ends to tie up before this one hits the books.

First and foremost, ensure that all work on the project from here on out is on a case by case basis. If you still have someone actively working on something, then you are not yet in stage 5. The project must be out the door and in the hands of the clients with no immediate plans to make changes or additions. Of course, projects tend to quietly continue until every minute bug is squashed, but if you are not expecting any work to be done in the foreseeable future, you can transition to stage 5.

The first and most important part of stage 5 is obtaining client feedback. On the following worksheet there are a few questions that should be answered by the client for your use in stage 6, but it is encouraged to add to this list as you see fit. The purpose of this evaluation is to determine how successful the project was from the client’s perspective, and as a result, how appropriate the process was for the client.

These answers can be gathered in any fashion that suits your client. Face-to-face, messaging, and a digital survey are the most common ways of obtaining these answers, and all work well, but there may be another way of gathering feedback that works for your situation. Record these answers on the following worksheet and ensure that all work on the project is finished for the foreseeable future before moving to the next stage. Things such as handoffs, client communications, testing, and bug fixing should all be finished before starting stage 6.
Stage 5: Finish Client Evaluation Worksheet

1. How satisfied are you with the outcome of this project?

2. Were there any points during the duration of the project that were frustrating, and if so, what were they?

3. What would you change about the project process?

4. Did you have any doubts about the project, and if so, what were they?

5. On a scale of 0%-100%, with 0% being “not successful at all” and 100% being “successful in every aspect”, what percentage of success would you say the project had, and why?
Stage 6: Retrospective

Lastly, we have the sixth and truly final stage, which is dedicated to learning from your mistakes, and your successes. Similarly to the client evaluation, you will evaluate the process on two levels: as a team member, and as a project manager. It is okay if the project management roles are also held by someone who actively worked on the project, but they will need to fill out each set of questions on the following worksheets.

The question set is more specific than the client evaluation, since it is intended to pick apart this project's implementation of LOAD. Where did it excel? Where did it falter? What can be done better next time, and what should be done the same? You will use these responses, as well as the client evaluation, and all other worksheets from the process to come up with a Retrospective Analysis for use on the next project that you decide to use LOAD for.

Just like the client evaluation, distribute the Team Member Evaluation Worksheet to all people who actively worked on the project, and distribute the Project Manager Evaluation Worksheet to anyone who had decision-making authority in respect to the LOAD process. Note, it is not necessary for a Project Manager to fill out this sheet if they were not involved in the LOAD process and decision making. For example, if your team includes a development leader who is responsible for reviewing code, and making technical choices, but was not involved in shaping the timeline or filling out the LOAD worksheets, they would not need to fill out the Project Manager Evaluation Worksheet, but rather the Team Member Evaluation Worksheet.
Stage 6: Retrospective Team Member Evaluation Worksheet

1. How were you affected by using the LOAD process?

2. Did the LOAD process affect your ability to do work, and if so, how?

3. How did your experience with this project compare to other projects?

4. What would you have changed about the process?

5. What would you have kept the same about the process?

6. Please identify any events or parts of the process that caused significant issues:

7. Please identify any events or parts of the process that benefited you greatly:

8. Did you feel a sense of consistency with the process? Why or why not?

9. How satisfied are you with your work on this process and what could have been done both by yourself, and by others to make you more satisfied?

10. On a scale of 0%-100%, with 0% being “not successful at all” and 100% being “successful in every aspect”, what percentage of success would you say the project had and why?
Rate the following on a scale of 1-5, with 1 being strongly disagree, and 5 being strongly agree.

The project was successful ___
I did not stand in the way of the LOAD process ___
If the project went off course, the process was corrected appropriately ___
If the project went off course, the process was corrected in a timely manner ___
I knew what was expected of me ___
I knew what was expected of others ___
The LOAD process was strictly followed ___
The LOAD process was worthwhile for this project ___
I am proud of my work on this project ___
The project’s goals were accomplished ___

Total: ___
Stage 6: Retrospective Project Manager Evaluation Worksheet

1. How did you utilize the LOAD process?

2. Did the LOAD process affect your ability to do work, and if so, how?

3. Did you manage the LOAD process appropriately? Why or why not?

4. What would you have changed about the process?

5. What would you have kept the same about the process?

6. Please identify any events or parts of the process that caused significant issues:

7. Please identify any events or parts of the process that excelled:

8. Did you maintain a sense of consistency with the process? Why or why not?

9. How satisfied are you with the outcome of the LOAD process?

10. On a scale of 0%-100%, with 0% being “not successful at all” and 100% being “successful in every aspect”, what percentage of success would you say the project had and why?
Retrospective Project Manager Evaluation Worksheet (cont.)

Rate the following on a scale of 1-5, with 1 being strongly disagree, and 5 being strongly agree.

The project was successful

I did not stand in the way of the LOAD process

If the project went off course, the process was corrected appropriately

If the project went off course, the process was corrected in a timely manner

I caught errors in the process before they happened

I spent as little and as few times in Stage 4 as possible

The LOAD process was strictly followed

The LOAD process was worthwhile for this project

The LOAD process was not preventing project progress

The project’s goals were accomplished

Total: ___
Stage 6: Retrospective (cont.)

Once you have received feedback from all team members and project managers, you are ready for the last step in the LOAD process, the creation of a Retrospective Analysis Worksheet. The worksheet is intended to highlight the good and the bad of the LOAD process by analyzing the data acquired from all three types of evaluation. **You will produce a summary of both the qualitative and quantitative data and use that summary to suggest adjustments to the implementation of the LOAD process for your next project.**

The first section of the worksheet will have you compute values based on the “Answer 1 - 5” sections on the Team Member and Project Manager Evaluation Worksheets. For the purposes of these calculations, you will notice that the Client Evaluation Worksheet does not have these sorts of questions. This is intentional, as it provides a means for evaluating the process itself, separate from the final product.

Of course, since a successful project is the overall purpose of the LOAD process, you will include the client’s responses in your analysis of the short answer questions. **This will allow you to distinctly determine whether a problem lied within the product or the process.**

**After the computations, you will be asked a series of questions concerning the responses of the short answer questions.** Due to the broad coverage of the questions, some answers may be significantly longer than others, since any given project is unlikely to experience every type of notable pattern the questions cover.

Even though you may feel tired of LOADING at this stage in the process, **it is vital that this worksheet be completed with care, and not overlooked.** You should find that over time, your analyses will become more and more positive as you learn from past mistakes. The worksheet includes questions directly related to how the process can be improved for the future, and these changes should be diligently implemented in future implementations of LOAD.
Stage 6: Retrospective Analysis Worksheet

Computations

\[ T_t = \text{Total number of Team Member Responses} \]
\[ \Sigma_t = \text{Sum of Scores for Team Member Responses} \]
\[ A_t = \text{Average Team Member Score} = \frac{\Sigma_t}{T_t} \]
\[ T_p = \text{Total number of Project Manager Responses} \]
\[ \Sigma_p = \text{Sum of Scores for Project Manager Responses} \]
\[ A_p = \text{Average Project Manager Score} = \frac{\Sigma_p}{T_p} \]
\[ S_t = \text{Success Index for Team Members} = \frac{5}{2} \times (A_t - 10) \]
\[ S_p = \text{Success Index for Project Managers} = \frac{5}{2} \times (A_p - 10) \]
\[ S = \text{Overall Success Index} = \frac{S_t + S_p}{2} \]

Quantitative Analysis

1. What question(s) were scored most highly by team members?

2. What question(s) were scored most highly by project managers?

3. Do these outcomes conflict with each other, and if so why?
4. What will you keep the same in order to ensure these questions continue to be rated highly in the future?

5. What question(s) were scored worst by team members?

6. What question(s) were scored worst by project management?

7. Do these outcomes conflict with each other, and if so why?

8. What could be changed to raise the scores on these questions in the future?

9. Were there any team members or project managers that gave consistently lower scores than the rest of the group, and if so, why might this have happened, and what could be done to prevent this in the future?

10. Do any of the scores strike you as surprising? Why or why not?
Qualitative Analysis

1. Which group of people (team members, project managers, or client) responded the most positively to the short answer questions? Why do you think this is?

2. Which group of people (team members, project managers, or client) responded the most negatively to the short answer questions? Why do you think this is?

3. What could have been done differently so that the group identified in question 2 would have responded more positively? Is this negativity a result of that group, or another factor?

4. In general, how did team members’ descriptions of how successful they felt they were compare to the team members success index (S)? If they were significantly different, why do you think this was?
5. In general, how did project managers’ descriptions of how successful they felt they were compare to the project managers success index ($S_p$)? If they were significantly different, why do you think this was?

6. How did team members respond to the LOAD process?

7. How did project managers respond to the LOAD process?

8. Did any patterns, positive or negative, emerge from the short answer questions. What is the significance of these patterns?
Retrospective Summary

Provide an approximately one to two page description of everything learned in this retrospective process. This summary should be clear enough to use as a reference for any future projects.