Evaluation of the Self-Governance Developer Framework from Software Developers' Perspective

NISHAT KORADA
Evaluation Of Self-Governance Developer Framework from Software Developers’ Perspective

Nishat Korada
Acknowledgements

I would like to thank Associate Professors Anne Hakansson and Mira Miroslawa Kajko-Mattsson for guiding me through this Master’s thesis project and helping me to write this document. Especial thanks to Prof. Mira Kajko-Mattsson for taking time off her work to help me; and for introducing me to the field of software engineering in particular. Thanks also to my technical director Mr. K. Murthii for giving me the opportunity to perform research at the firm Swiesh QLurn IT Solutions, while I was working there. I am also grateful to my family members for always supporting me and being by my side while working on this thesis.
Abstract

Every software developer uses some software process to build computer software. The process may be haphazard or ad hoc; may change on a daily basis; may or may not be efficient, effective, or even successful; but a “process” does exist. Although many software development process models exist nowadays, there is a real dearth of models which focus on the job of the software developer and his/her individual effort. The Self-Governance Developer Framework, developed by Prof. Mira Kajko-Mattsson and Gudrun Jeppesen, is a new software development model aimed at the software developer in particular and how he/she can manage their individual processes at hand. It aims to be flexible yet specific in identifying all types of tasks, work activities and roles which can come up in a developer’s work and how to tackle them.

We do not know anything about the usefulness of the proposed Self Governance Developer Framework. In this thesis, research was conducted on how industry-based software engineers evaluate the newly proposed SGD Framework and how useful it is to them. The thesis researched how useful the proposed SGD Framework is in real world software development. The thesis also researched how much time was being spent on individual tasks in a software development project at my own company Swiesh QLurn IT Solutions. The SGD Framework was the tool used to gather these resources. The purpose was to find out what tasks and activities can come up during software development and to improve the own way of working to become a more efficient software developer. The goal was to provide a basis for further research into relevant models like software process models, estimation models and software process improvement models. With this basis it should be possible to build or improve these relevant models in order to better manage a developer’s activities and resources in a software project. They can also provide a boost for inventing developer-centric models just like the Self-Governance Developer Framework.

The research methods applied were both of a qualitative and quantitative nature. The research was of an inductive type where raw data was first gathered and general theories derived from the data. Along with the survey conducted by a qualitative questionnaire, quantitative data about time allocation in a software project was gathered via action research in a software project at a company. Data was also collected via a literature review of past process models. A survey questionnaire was distributed worldwide via a blog and among software professionals with a minimum of three years experience.

The results from this survey questionnaire show how working professionals evaluate the proposed SGD Framework; and the coverage of software process activities in the SGD Framework. Their feedback was important to help suggest deficiencies in the SGD Framework and to help improve it for industry-wide target audience. The thesis results in data which could prove valuable to the individual developer to enhance their software writing skills, and become more efficient and independent. Other improvements are suggested in the final analysis and discussion chapter. However, further research is needed to get more accurate results from a bigger group of developers. These studies should then include complete software projects to gather metrics and data.
# Table of Contents

1 **Introduction** ......................................................................................................................... 8

   1.1 Background .......................................................................................................................... 8
   1.2 Problem ................................................................................................................................. 9
   1.3 Research Question ............................................................................................................... 9
   1.4 Purpose ................................................................................................................................. 10
   1.5 Goals ..................................................................................................................................... 10
   1.6 Commissioned Work And Stakeholders .............................................................................. 10
   1.7 Methods ............................................................................................................................... 11
   1.8 Ethics .................................................................................................................................... 11
   1.9 Sustainability ....................................................................................................................... 11
   1.10 Increased Well Being ......................................................................................................... 11
   1.11 Delimitations ...................................................................................................................... 12
   1.12 Thesis Outline ................................................................................................................... 12

2 **Extended Background** ........................................................................................................... 13

   2.1 History of Software Development ....................................................................................... 13
   2.2 Agile Development ............................................................................................................... 14
   2.3 Personal Software Process (PSP) .......................................................................................... 17
       2.3.1 Explanation of the PSP .................................................................................................. 18
   2.4 PIPSI (Process for Improving Programming Skills in Industry) .......................................... 21
   2.5 The Self Governance Developer Framework ......................................................................... 21
   2.6 Related Work to the Thesis ................................................................................................ 22

3 The Self Governance Developer Framework .................................................................................. 23

   3.1 The Self Governance Developer Framework Overview ................................................... 23
   3.2 The Self Governance Developer Framework Process Model .......................................... 24
       3.2.1 The Pre-Work Activities ................................................................................................. 24
       3.2.1 The Work Activities ....................................................................................................... 26
       3.2.1 The Post-Work Activities ............................................................................................... 29
4 Research Methodology ........................................................................................................................................... 30

4.1 Research Strategy ............................................................................................................................................... 30

4.2 Research Type.......................................................................................................................................................... 31

4.2.1 Action Research.................................................................................................................................................. 31

4.2.2 Quantitative Research...................................................................................................................................... 31

4.2.3 Qualitative Research......................................................................................................................................... 32

4.2.4 Inductive Reasoning......................................................................................................................................... 32

4.3 Research Phases.................................................................................................................................................... 33

4.3.1 The Pre-Study Phase....................................................................................................................................... 33

4.3.2 The Study Phase............................................................................................................................................... 34

4.3.3 The Evaluation Phase..................................................................................................................................... 36

4.4 Sampling Methods................................................................................................................................................ 37

4.5 Research Instruments.......................................................................................................................................... 37

4.6 Validity Threats.................................................................................................................................................... 37

5 Evaluation Criteria and Set of Desirable Features .................................................................................................. 38

5.1 Evaluation Model.................................................................................................................................................. 38

5.2 Completeness of the SGD Framework.................................................................................................................. 39

5.3 Understandability of the SGD Framework........................................................................................................ 40

5.4 Time Spent on Activities in the SGD Framework............................................................................................... 40

5.5 Additional Survey Questions .............................................................................................................................. 41

6 Research Results for Evaluation Criteria- Completeness of SGD ........................................................................... 43

6.1 Background to the Questionnaire ....................................................................................................................... 43

6.2 Questionnaire Results to the Completeness of the SGD................................................................................... 44

6.3 Action Research Results to the Completeness of the SGD............................................................................... 48
1 Introduction

Every software developer uses some software process to build computer software. The process may be haphazard or ad hoc, may change on a daily basis, may or may not be efficient, effective, or even useful; but a “process” does exist. Watts Humphrey [1] suggests that in order to change an ineffectual personal process, an individual must move through four phases, requiring training and careful instrumentation. The systems development life cycle (SDLC), also referred to as the software development process, is a term used in software engineering to describe a process for planning, creating, testing, and deploying an information system [1]. A systems development life cycle (SDLC) is composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems. Almost any software developer is familiar with the software development process and uses it in his/her software design.

1.1 Background

Software development is the process of computer programming, documenting, testing, and bug fixing involved in creating and maintaining applications and frameworks resulting in a software product. Software can be developed for a variety of purposes, the three most common being to meet specific needs of a specific client or business, to meet a perceived need of users, or for personal use. The need for better quality control of the software development process has given rise to the field of software engineering, which aims to apply the systematic approach exemplified in the engineering paradigm to the process of software development [1]. There are many approaches to building software products, known as software development life cycle models, methodologies, processes, or models. The waterfall model [6] is a traditional version, contrasted with the more recent innovation of agile methodologies [2].

A variety of software development frameworks have evolved over the years, each with its own known strengths and weaknesses. Each of the available methodology frameworks are best suited to specific kinds of software projects, based on various technical, organizational, project and team considerations [3].

To manage this level of complexity, a number of software process models or methodologies have been created, such as "waterfall" [6]; "spiral" [17]; "Agile software development" [33]; "rapid prototyping" [41]; "incremental" [15]; and "synchronize and stabilize" which are widely used in companies. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle—from inception of the idea to delivery of the final system—to be carried out rigidly and sequentially" within the context of the frame being applied [2].

Any software process framework provides a sequence of activities for system designers and developers to follow. It consists of a set of steps or phases in which each phase of the framework uses the results of
the previous one. Software development process models usually incorporate various distinct categories, phases and roles as a result. The individual software developer may have to switch roles and keep all these phases in mind as he/she goes about their work, creating the software or information system.

There are many standards and process models that aid in the development of working software systems. The software developer’s role is not limited to only writing code and debugging it, the developer may be involved in several other phases/roles/responsibilities even and sometimes his or her role might overlap with those of other professionals. His work may include properly researching, designing, developing, and testing software. A software developer may take part in design, computer programming, or software project management as well. He may contribute to the overview of the project on the application level rather than component-level or individual programming only.

There are not many software process models that focus on the software developer and his individual activities. There is a real dearth of various processes or conventions that can aid the individual software developer in writing/compiling/testing his or her software which is why we will go on to analyze this gap and see if we can find any proposed solutions to fill it. The Self-Governance Developer (SGD) Framework devised by Prof. Mira Kajko-Mattsson and Gudrun Jeppesen [8] serves the purpose of aiding the individual software developer to improve his or her own individual software-writing process. As such the target audience of the Self-Governance Developer (SGD) Framework is software developers.

1.2 Problem

The problem is that the SGD Framework has not been evaluated by the software industry from a usefulness perspective. Hence, we do not know anything about the utility and usefulness of the SGD Framework to the individual developer or in a real world project scenario. We do not know if the SGD Framework can be used in its present form in industry projects to produce working software and how beneficial it is to the individual developer. But the problem is that the SGD Framework has not yet been evaluated by individual developers or within the software industry.

1.3 Research Question

In this thesis, the various activities and resources of an individual developer in software development are researched with the Self-Governance Developer (SGD) Framework in mind:

- How comprehensive and complete is the SGD Framework in its listing of software development activities and is it clear to understand?

This research question will help guide the research towards its goal. These activities and tasks can include gathering and specifying requirements, preliminary objectives, design goals, coding, debugging source code, testing and integration of written code etc. We have tried to be as comprehensive and diverse in collecting all such tasks as may be encountered while working on a software project. In the SGD Framework they are categorized into 3 categories: Pre-Work, Work Activities & Post-Work Activities.
1.4 Purpose

The purpose of the thesis is to find out by studying how useful is the Self-Governance Developer Framework in real world development. The thesis researched to find out the coverage of software development activities & tasks as proposed in the SGD Framework. The research carried out in this thesis investigated if the proposed list of developer activities in the SGD Framework is useful or if anything needs to be added or revised. The information gathered should be used to discipline developers and improve their way of working and self-assessment.

1.5 Goal

The goal of the research is to provide a basis for relevant models like process models, self-assessment models, and software process improvement models (SPI models) which can focus on the individual developer and his/her effort. With such a basis it should be possible to build upon or improve these relevant models in order to better manage developers’ resources and improve their way of working in software projects.

1.6 Commissioned Work and Stakeholders

The researcher was working for the company Swiesh (www.swiesh.com) [43] as a software developer when this research commenced. The assignment was to work on their server functionality and the tasks included working in a team as well as alone on both adding functionality and improving the server’s functionality already existing.

This thesis has two main stakeholders: individual software developers and academia. Regarding the first stakeholder, the primary target is any developer in need of better methods to build software and self-discipline.

Regarding academia, the fact that is very little published material on this matter makes it possible for this thesis to provide a basis for further research into this area of personal software development. This thesis can be useful for relevant researches about time management, software process models, estimation models and SPI models etc.

1.7 Methods

The research method applied was both of a quantitative and qualitative nature. An action research where the structure of the action research process [3] was followed; where the author of this thesis was also a participating member. This was followed by research of an inductive qualitative type where the instrument used was a survey questionnaire (not containing numerical data or any metrics). The research had an inductive reasoning, starting with observations and ending with general theories. Data collection was conducted using the action research and a literature study. The subjects of the study were selected with the convenience sampling method. Data evaluation was conducted using an evaluation model including 4 evaluation criteria.
1.8 Ethics

During the action research phase confidential information was obtained from the company where the research was conducted. This information was not shared with the outside world, only within the company. The right to confidentiality and copyright was also strictly followed. The identity of the respondents to the campaign survey was kept confidential and their feedback was not made available except to the supervisors of the thesis. We did our best to keep to ethical norms and standards during the research and written phases of this thesis.

1.9 Sustainability

The research will consider the sustainability problem based upon the three pillars of sustainability [4]

- Environmental Sustainability
- Social Sustainability
- Economic Sustainability

The big issue today is the focus on economically sustainable methods when it is more important that we care for our planet and environmental sustainability. A shift of focus to these 2 pillars of sustainability would also reap an economic benefit in the long term.

From an economic viewpoint, knowing how to maximize software development efficiency and time resources can save money and improve the economy of an organization. Unnecessary time spent can be avoided and spent on better activities. An organization with a good economy can focus more on environmental sustainability and also on social factors like social well-being.

1.10 Increased Well Being

A social benefit of knowing how tasks are being implemented is the increased productivity for software companies. The increased productivity comes from making the developers more efficient with their time which makes it possible to be more productive. Increased productivity can improve the standard of living from both a commercial and consumer perspective [5]. It is possible to decrease the price of the developed products, making the consumers wealthier and business more profitable.

1.11 Delimitations

The research was carried out formally over a two month period. At the beginning a pilot study was carried out with three developers with informal interviews to gauge the scope of the study and if the questionnaire was feasible. This was a small number of developers from a single firm so it would have been better if a more diverse group was consulted first. The qualitative campaign survey was restricted to developers I knew personally and the help of an online developer blog. More access to the internet
and companies would have meant a more diverse and richer amount of feedback from developers worldwide.

For the quantitative part of the action research about time allotment, another employee from Swiesh technologies filled in the data (and the author of thesis was not present when this was done). The focus of this action research was how much time was required by various activities in a project and could have been more accurate if the author of the thesis was present at all sessions.

1.12 Thesis Outline

The thesis consists of the following chapters:

Chapter 2: Extended Background: This chapter describes the SGD Framework used in the research and also covers all previous models and necessary theories to understand the thesis research.

Chapter 3: Research Methods: This chapter provides a description of the research methodology. Covering the research methods applied, research phases, sampling methods, research instruments and validity threats.

Chapter 4: Evaluation Criteria and Questionnaire: This chapter describes the various evaluation criteria used in the research. It also presents the additional questions and the motivation for adding them to the questionnaire.

Chapter 5: Action research study: This chapter presents the details of the action case study carried out in a real-world project. The time report sheet and quantitative results derived are presented.

Chapter 6: Presentation of the Research Results: This chapter presents the feedback received from respondents and the research results.

Chapter 7: Analysis and Discussion: This chapter analyzes and discusses the presented results from the research.

Chapter 8: Conclusions and Future Work: This chapter provides conclusions and discusses future work.
2 Extended Background

The Extended Background introduces us to software development starting with the basic theories about software development, and models and frameworks used in software engineering to develop software products. The Extended Background reviews all major paradigms and theories of software development and give us an examination of the major models in this. These can be roughly subdivided into “traditional methods” and “agile methods”. Section 2.1 introduces basic concepts of agile development, how they relate to the scope of this thesis and the individual software developer. Section 2.2 deals with the Personal Software Process (PSP) model developed by Watts Humphrey, and section 2.3 with the overview of the Process for Improving Programming Skills in Industry (PIPSI). Section 2.4 is an overview of the SGD Framework and its various categories. More literature review dealing with agile methods is in Appendix A.

Computer software is the product that software professionals build and then support over the long term. Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture and even our smart-phones. Virtually everyone in the industrialized world uses it either directly or indirectly. Software developers build, test and support software and from the point of view of a software developer, the work product is the set of programs, content (data and information), and other work products that are computer software. From the user’s perspective, the work product is the resultant information that somehow makes the user’s world better. Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow software developers to build high quality computer software.

2.1 History of Software Development

When a software engineer works to build a product or system, it’s important to go through a series of predictable steps—a road map that helps the engineer create a timely, high quality result. This road map that is followed is called a “software process.” Software engineers and their managers adapt the process to their needs and then follow it.

The history of software processes began in the early 1950’s when prescriptive models were originally proposed to bring order to the chaos of software development in major companies like Boeing and IBM. History has indicated that these traditional engineering models have brought a certain amount of useful structure to software development work. Traditional process models strive for structure and order; they are useful when the requirements for a problem are well understood—when work flows from communication through design and deployment in a reasonably linear fashion. These earliest software process models or frameworks comprise a systematic, sequential approach to software development that begins with customer specification of requirements and progresses sequentially through planning, modeling, construction, and deployment, culminating in ongoing support of the software product.

A classic example of this is the waterfall model [6], sometimes called the classic software life cycle. The waterfall model is the oldest framework for software development. A later variation in the
representation of the waterfall model is called the V-model [7]. In reality there is no fundamental
difference between the waterfall model and the V-model but the V-model provides a way of visualizing
how verification and validation actions are applied to earlier engineering work. Sequential process
models, such as the waterfall [6] and V-models [7], are the oldest software development paradigms.
They suggest a linear process flow that is often inconsistent with modern realities (e.g. continuous
change, evolving systems, and customer requirements) in the software world. They do, however, have
applicability in situations where requirements are well defined and stable.

But it was soon realized that software work is fast-paced and subject to a never ending stream of
changes (to features, functions and information content). The waterfall model [6] is often inappropriate
for such work. This led to the evolution of incremental process models [15] that are iterative in nature
and produce working versions of software quite rapidly. In such cases, you can choose a process model
that is designed to produce the software in increments. The incremental model [15] combines elements
of linear and parallel process flows discussed earlier. When an incremental model is used, the first
increment is often a core product. That is the basic requirements are addressed but many
supplementary features (some known, others unknown) remain undelivered.

Further need and evolution of software processes led to Evolutionary process models [16] that
recognize the iterative, incremental nature of most software development and are designed to
accommodate change. Evolutionary models, such as prototyping and the spiral model [17], produce
incremental work products (or working versions of the software) quickly. Although prototyping can be
used as a stand-alone process model, it is more commonly used as a technique that can be implemented
within the context of any one of the previous process models noted in this chapter. Using the Spiral
model [17], software is developed in a series of evolutionary releases. During early iterations, the
release might be a model or a prototype. During the later iterations, increasingly more complete
versions of the software system are produced. All these evolutionary software development changes
culminated in an attempt to draw on the best features and characteristics of traditional software
process models, but characterize them in a way that implements many of the best principles of agile
software development. Thus the Unified Process model of the 1990’s led to the agile methodologies.

2.2 Agile Development

In February 2001, 17 software developers met at the Snowbird resort in Utah to discuss lightweight
development methods. They published the Manifesto for Agile Software Development, in which they
shared that through their combined experience of developing software and helping others to do it they
had come to value. In 2001, Kent Beck and 16 other noted software developers, writers and consultants
[31] (referred to as the “Agile Alliance”) signed the “Manifesto for Agile Software Development.” It
stated:

We are uncovering better ways of developing software by doing it and helping others do it. Through this
work we have come to value:

- Individuals and interactions over processes and tools
That is, while there is value in the items on the right, we value the items on the left more.

Introducing the manifesto on behalf of the Agile Alliance, Kent Beck and others clearly said that there is a lot of value in the individual software developers and how they approach their work activities to deliver a working software product.

Just what is agility in the context of software processes? Ivar Jacobson [34] provides a useful discussion:

“Agility has become today’s buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they built or the project that creates the product. Support for changes should be built in for everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is built by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.”

In Jacobson’s view [34], the pervasiveness of change is the primary driver for agility. Software engineers must be quick on their feet if they are to accommodate the rapid changes Jacobson describes. In a thought provoking book on agile software development, Alistair Cockburn [33] argues that the traditional process models introduced before in the SDLC models have a major failing: they forget the frailties of the people who build computer software. Software engineers are not robots, he argues. They exhibit great variation in working styles; significant differences in skill level, creativity, orderliness, consistency and spontaneity. Some communicate well in written form, others do not! Cockburn [33] argues that process models can “deal with people’s common weaknesses with [either] discipline or tolerance” and that most traditional process models choose discipline. He states: “Because consistency in action is a human weakness, high discipline methodologies are fragile.” [33]

Although the underlying ideas that guide agile development have been with us for many years, it has been less than two decades since these ideas have crystallized into a “movement.” In essence, agile methods were developed in an effort to overcome perceived and actual weaknesses in conventional software engineering. Agile development can provide important benefits, but it is not applicable to all projects, all products, all people, and all situations. It is also not antithetical to solid engineering practice and can be applied as an overriding philosophy for all software work.

In the modern economy, it is often difficult or impossible to predict how a computer based system (e.g. a Web-based application) will evolve as time passes. Market conditions change rapidly, end-user needs evolve, and new competitive threats emerge without warning. In many situations, you won’t be able to define requirements fully before the project begins. You must be agile enough to respond to a fluid programming environment and change. Fluidity implies change and change is expensive. Particularly if it is uncontrolled or poorly managed. One of the most compelling characteristics of the agile approach is its ability to reduce the costs of change throughout the software process.
Proponents of agile software development take great pains to emphasize the importance of “people factors.” As Cockburn and Highsmith [35] state, “Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams.” The key point in this statement is that the process molds to the needs of the people and team, not the other way around – Successful software engineering organizations recognize this reality regardless of the process model they choose.

Agility Principles

The Agile Alliance (see [37], [38]) defines 12 agility principles for those who want to achieve agility. A few of the principles out of the original 12 are listed below since they apply to the individual developer:

1. Build projects around motivated individuals. Give them the environment and the support they need, and trust them to get the job done.

2. Continuous attention to technical excellence and good design enhances agility.

3. The architectures, requirements, and designs emerge from self-organizing members.

4. Working software is the primary measure of progress.

5. Customers and developers must work together daily throughout the project. [38]

Any agile software process is characterized in a manner that addresses a number of key assumptions [36] about the majority of software projects:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.

2. For many types of software, design and construction are interleaved. That is both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.

3. Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

Given these three assumptions, an important question arises. How do we create a process that can manage unpredictability? The answer lies in process adaptability to rapidly changing project and technical conditions. An agile process, therefore, must be adaptable above all. But continual adaptation without forward progress accomplishes little. Therefore, an agile software process must adapt incrementally. To accomplish incremental adaptation, an agile process requires customer feedback (so that the appropriate adaptations can be made on time). An effective catalyst for customer feedback is an operational prototype or a portion of an operational system.
2.3 Personal Software Process (PSP) – Watts Humphrey

The Personal Software Process (PSP) was introduced by Watts Humphrey [1] at the Carnegie Mellon University that focuses on the developer responsible for the coding (just like the SGD Framework) and empowers the developer to control the quality of all software work products that are being developed. The primary software process method, found in extended background study, used to improve the individual developer’s coding process was the PSP. PSP and SGD framework are similar in that both serve as a tool for the individual developer of self-evaluation and process development.

The *Personal Software Process* (PSP) [1] emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product in measurable terms. This is accomplished through a rigorous assessment activity performed on all work products that you produce. PSP represents a disciplined, metrics-based, individualized approach to software development that may lead to culture shock for many practitioners. In order to take a closer look at what can be done with PSP to make it a valuable framework for individual developers, this section will describe the PSP based on the books by Watts Humphrey [9], [10].

The reason that the PSP is taken up in this report is that it is a description of a process method with the purpose of improving the individual software engineer’s own process (like the SGD model). The aim of the PSP is to develop both the individual developer’s ability to plan and estimate the time for his own work. Finding defects at an early stage and to be able to solve them quickly [1]. With the help of PSP developer may also have a chance to see where and how he introduced defects in the program and cost to resolve these defects. An endpoint for productivity PSP is the number of lines of code per hour. A simple structure for how the PSP is built can be seen in the following Figure 5 below. Figure 5 below shows that the PSP is a collection of seven different processes that are divided in four different levels.

Users of PSP work up level by level by performing the various process steps. The difference between the process steps is not large; each following process builds upon the previous step. PSP training follows an evolutionary improvement approach: an engineer learning to integrate the PSP into his or her process begins at the first level – PSP0 – and progresses in process maturity to the final level – PSP2.1. Each Level has detailed scripts, checklists and templates to guide the engineer through required steps and helps the engineer improve her own personal software process. Humphrey encourages proficient engineers to customize these scripts and templates as they gain an understanding of their own strengths and weaknesses.

PSP0 as shown in Figure 1 has four phases: planning, development (design, code, compile, test) and a post mortem. A baseline is established of current process measuring: time spent on programming, faults injected/removed, size of a program. In a post mortem, the engineer ensures all data for the projects has been properly recorded and analyzed. PSP0.1 advances the process by adding a coding standard, a size measurement and the development of a personal process improvement plan (PIP). In the PIP, the engineer records ideas for improving his own process. Then, the engineer estimates how large a new program will be and prepares a test report (PSP1). Accumulated data from previous projects is used to estimate the total time.
2.3.1 Explanation of the PSP

The main aim for the level "The Baseline Personnel Process", (the boxes at the bottom of Figure 1) is to
get a foundation of data for the individual developer. This data is used as a basis for measuring the
developer's personal development. Data measured is above all: Defective data, the time to develop
systems and program size in lines of code [9].

In levels 0-2 of Figure 1, the user creates three programs per level. In level three, users create a greater
program. In the level "Personnel planning" the focus is on improving staff and project
management. User then gets the information to estimate the time and size and schedule a plan for the
program to be developed in this level [9].

The level "Personal Quality Management" (PSP2), as the name suggests is about improving the quality of
the program. The goal is to cut down on the amount of defective data. The developer may estimate the
number of defects for the programs to be developed at this level. Code and Design Reviews (PSP2) are
then used as tools to get to cut down the number of defects [10].
The last level "The Cyclic Development" is aimed at improving the developer's process of programs on a larger scale. In this step the program is divided into a series of small program steps all of which are performed within the process framework for the level "Staff Quality Management".

The developer will here also engage in high-level planning and design. A number of reports are developed continuously until the process steps where the software developer improvement among others shown by analyzing and using statistical presentation. PSP’s major disadvantages include a lot of time spent in training, documenting defects & personal timekeeping etc. and also it has been poorly implemented in academia or industry due to these disadvantages. PSP generates a lot of forms, schemes, formats, schedules and so on which are not rigorously kept by many software professionals over a longer period of time. PSP is ultimately a tool for keeping detailed records and personal improvement in software development over a period of time by keeping meticulous metrics.

2.4 Process for Improving Programming Skills in Industry (PIPSI)

PIPSI [12] was the closest to a personalized developer-oriented process model that I came across during the literature survey. The personal process and foundation of PIPSi is identical to that of the SGD Framework and both have relatively a lot in common even in higher level stages of their design. PIPSi model is like the PSP in keeping detailed metrics and records of the development activities worked on. The foundation of the entire structure is built upon the personal measurement metrics as shown in the diagram below, these metrics are strictly private to the developer at all times, and not available to management or anybody else.

The focus of PIPSi is on improving individual software programming skills thus generating bottom-up improvement. The key foundation to this bottom-up approach is the personal process to be used and adapted by the individual developer. Its main phases are PreBuild, Build, and PostBuild to be deployed in a phase-like manner. PIPSi is built around the following key concepts [12] as shown in Figure 2 below. This is a cyclical process starting out by maintaining records of personal activities during software development such as time spent on activities or how many meetings were required within a certain preparatory period. After gathering meticulous data the developer can go to the next level where he/she documents improvement areas for making their work more efficient. This is followed by the levels of personal project management and personal quality management.

Figure 2 below illustrates the three elements of personal software development - defining a personal process, personal project management and personal quality management. The entire model is buttressed and controlled through the use of measurement. By collecting data on their own performance, software engineers learn about how they develop software. The measures help them understand the fundamental relationship between size and effort and, through this understanding, enable them to improve their estimating abilities. Furthermore, by gathering data on their defect rates they witness how using practices such as personal code reviews and the use of checklists will allow them to produce higher-quality software products.
• A Personal Process to be used and adapted by the programmer. The three main phases of the process are PreBuild, Build and PostBuild. The Personal Process is the foundation needed to define Personal measurement and to use specific techniques and methods, Personal Project Management and Personal Quality Management.

• Personal Measurement which introduces personal measures (effort, size and defects) to monitor and improve the personal process.

• Personal Project Management which introduces activities and techniques to estimate the size and effort of a project, plan a project, and to schedule it.

• Personal Quality Management which deals with defect collection and analysis to track the quality of a project. Further, it introduces design and code reviews to improve the quality of a project.

PIPSI model is like the PSP in keeping detailed metrics and records of the development aspects worked on. The foundation of the entire structure is built upon the personal measurement metrics as shown in Figure 2 above, these metrics are strictly private to the developer at all times, and not available to management for productivity assessment or any similar purpose.

PIPSI focuses on two areas of concern to the software engineer – Personal Project Management and Personal Quality Management. Both of these are supported by an estimating process and the use of appropriate metrics for measuring past success in estimating and providing a basis for future planning (see Figure 2) [13]. Each program worked on is treated as a ‘project’ by the software engineer, who starts with a planning and estimating phase before starting work on the program. As the work progresses detailed metrics are kept, which are used both to monitor progress and to build up a history.
The entire personal process is cyclical, success in one personal project building on success in the previous projects. As software developers become comfortable with the concepts, new and more sophisticated techniques can be introduced to improve both Project Management and Quality Management.

Through PIPSI training [13], developers complete programming tasks on which they collect increasing quantities of data. Early exercises capture effort measures. Subsequent exercises gather size data whilst the concluding exercises capture defect and quality measures. This is hugely empowering for both the programmer and the organization as a whole. Programmers are now in a position where they can provide the project manager with achievable deadlines and the project manager can develop more accurate and predictable delivery schedules [13].

2.5 The Self Governance Developer Framework

Discipline and knowledge takes many forms and permeates almost every aspect of software development. Disciplined and knowledgeable developers know what is expected of them in specific developmental contexts. They know best what activities to choose and how to organize their work for the success of their projects. Many sources tell software developers what to do. The common ones are various software development methods [8], or guidance from managers or corporate in-house methods [8]. Today, there are no standard process models specifying complete lists of activities as required of software developers. Regarding the current software engineering literature, the lists of activities to be conducted by developers are scattered across various books or articles. The most relevant and all-inclusive sources are (1) Personal Software Process (PSP) as written by Watts Humphrey and (2) Team Software Process (TSP). Usually complete lists may be found only in the industry.

The SGD Framework provides generic activities that can be selectively chosen by software developers or teams while implementing software code and unit testing it. The goal of SGD is to support developers in their daily work by assisting them in self-managing, monitoring and controlling their own assignments. The framework’s target groups are software developers and teams whose main task is to code, compile, unit test and integrate their own code units before delivering them for integration and system testing. The Self-Governance Developer Framework is an extension of Watts Humphrey’s PSP.

The model’s activities cover a wide and all-inclusive spectrum of activities that are relevant for building, implementation and unit (developer) testing. In actual development endeavors, however, not all of the activities need be conducted. In some contexts and/or programming environments, only their subsets may be relevant. The SGD Framework does not impose any specific order of activities for the developer to follow. The listed activities can be conducted in any order and they may be included in any process phase that suits the developers and their environments. In this thesis, we have suggested Self-Governance Developer Framework outlining the activities aiding developers and/or teams in designing their own personal processes. SGD only provides a basic conceptual structure of the activities and provides guidelines for performing them. It does not provide any suggestion for any order among the activities. Neither does it define inputs and outputs of the activities. As a framework, it constitutes a
platform for creating developer process models, which in turn, are free to define their own order, inputs and outputs, and provide guidance in decision making.

Self-governance models should bring value in form of improved developer productivity and satisfaction. Developers should be able to decide upon what activities to choose based on the value the activities bring. This has been recognized in PSP as suggested by Watts Humphrey [1].

2.6 Related Work to the Thesis

The theory covered in the extended background chapter is to get a basic understanding of the field in order to understand the research. The research is connected with the theory in the following way. Software development processes and agile methods like Scrum were used during the research at the company Swishel QLurn IT Solutions. The action research phase comprised a software project using the agile methodology of Scrum to manage the project. Time allotment to the development activities was collected during 7 sprint cycles using Scrum method. Agile methodologies are also widely used in the software industry nowadays and are very popular. The majority of the respondents to the campaign survey questionnaire self-described themselves as using agile methods in their day to day work. Hence a large part of the extended background was devoted to explaining agile methodologies and this theory is needed to understand how software development proceeds.

Personal Software Process (PSP) created by Watts Humphrey was the first process model focusing on the individual developer and trying to make the developer’s work more efficient. It incorporates most of the key areas and activities which are relevant for this thesis as well and hence was included in the extended background. Process for Improvement of Programming Skills in Industry (PIPSI) was briefly presented since it is directly associated with software industry professionals and is similarly geared toward improving personal software development. Both PSP and PIPSI are useful knowledge to get a general understanding of the field concerning this thesis and the thesis itself. Both PSP and PIPSI models share many similarities with the SGD Framework and have the same aims of making the work of the individual software developer as self-governed and enhanced as possible.

The PSP has been successfully used within organizations already using software process improvement methods, where the culture of quality and process discipline is strong. However, it has not been so successful in smaller or less disciplined organizations. Some of the issues associated with the PSP are:

- There is a significant training period (excess of two weeks).
- It was developed for an academic environment, thus it is difficult to customize for industrial use.
- Many companies involved in PSP training have failed to implement it in a industrial context
- Bureaucratic overhead makes it more difficult to sustain in an industrial setting
- Lack of an adequate support tool for data gathering

Many people have studied the software quality problem and have proposed solutions, including better testing, better project planning, better practices, better programming environments and many other factors that potentially affect the development of software. We shall examine one such model ahead.
### 3 The Self-Governance Developer Framework

This chapter gives a description of the framework used in the research to collect information. It starts with section 3.1 an overview of the framework, followed by the SGD process model in section 3.2 with the included SGD activities used in the research. This description is needed to understand the framework used during the entire research, how it was used is described in Chapter 4.

The Self-Governance Developer (SGD) Framework devised by Prof. Mira Kajko-Mattsson and Gudrun Jeppesen [8] serves the purpose of aiding the individual software developer to improve his or her own individual software-writing process. As such its target audience is software developers. The SGD Framework is a newly devised software process framework which intends to contain all the activities and tasks which can come up in writing software.

#### 3.1 The SGD Framework Overview

The Self-Governance Developer Framework can be used by software developers to assist them in their daily work by self-managing, monitoring and controlling their own assignments. The SGD framework is divided into two parts, the SGD process model and my SGD process. The first includes the SGD process categories and activities.

![Figure 3. The Self Governance Developer Framework](image)
3.2 The SGD Process Model

There are three main SGD process parts included in the SGD process model, (1) pre-work, (2) work, and (3) post-work. For each of these process parts there are several activity categories with associated activities. It is not necessary to use all of the activities or perform them in any specific order [8]. The SGD activities were used in the research because it provided all types of developer activities during software projects. This was useful for the research. It also made it easier to distinguish what actually had been done.

The next sections give a short description of the SGD categories and activities. The purpose of this is to give a better understanding of what type of activities were considered during the research.

3.2.1 The Pre-Work Activities

The pre-work activities consist of the activities that usually happen before the work activities. There are preliminary and planning activities as seen in Figure 4 [8].

**Preliminary Activities**

The preliminary activities consist of activities to understand methodologies, technologies, standards, ways of working and commitments [8]. They are usually conducted before starting implementation and testing work. The reason for that is because they provide developers with the resources needed to perform high quality work. It is important to understand the project plan before getting to work (see activity PR-1 in Figure 4) to avoid problems later. Revising and ensuring that the technology is understood will save time later on (see activity PR-2 in Figure 4). Understanding organizational standards and ways of working makes it easier to start working in a correct way (see activities PR-3, PR-4 in Figure 4). It is also important to check your own way of working to match with the organizational way of working (see activity PR-5 in Figure 4).

**Planning Activities**

The planning activities includes activities for reviewing the necessary documents, determining ways of conducting the work, and planning the work [8]. Developers should review the necessary documents to understand the scope of their work. The documents to read are requirements and design specifications (see activities PL-1, PL-2 in Figure 4). While reviewing the documents or in any other activity where questions or uncertainties appear should they be resolved to avoid any misunderstanding and/or confusion (see activity PL-3 in Figure 4). Understanding the requirements and design specifications aids developers determine implementation and testing goals, strategies and practices that will guide them in their planning (see activities PL-4 - PL-6 in Figure 4). Planning the work is very important to perform as good as possible and avoid problems in the work phase. There is a lot of planning that can be done to achieve this (see activities PL-7 to PL-13) in Figure 4.
3.2.2 The Work Activities

The Work activities consist of activities to prepare for coding and testing, actual coding, testing the code, evaluate the tests and debug the code [8]. These activities are conducted after the pre-work activities.

Preparatory Activities

The preparatory activities includes activities to prepare for the implementation work by looking at low-level designs, test case designs, stubs and drivers, and testing environment [8]. The activities will help developers to become ready for writing and testing code. Preparing and reviewing low-level designs.
<table>
<thead>
<tr>
<th>Preparatory Activities</th>
<th>Coding Activities</th>
<th>Testing Activities</th>
<th>Evaluative Activities</th>
<th>Debugging Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1: Prepare (make) and/or review your low-level design(s) of the code to be written or changed.</td>
<td>C-1: Write/rewrite your code.</td>
<td>T-1: Check whether the unit (developer) test case base meets the given requirements and design.</td>
<td>E-1: Analyze your unit (developer) testing results.</td>
<td>D-1: Identify the source of an error(s).</td>
</tr>
<tr>
<td>P-2: Prepare (make) an impact analysis of your low-level design(s).</td>
<td>C-2: Compile/recompile your code as required.</td>
<td>T-2: Check whether the unit (developer) regression test base meets the given requirements and design.</td>
<td>E-2: Depending on the unit (developer) testing results, determine your next step(s).</td>
<td>D-2: Determine solution(s) for eliminating the sources of error(s).</td>
</tr>
<tr>
<td>P-3: Determine the types of unit (developer) test cases and their order.</td>
<td>C-3: Make notes on your compilation errors, if necessary.</td>
<td>T-3: Remedy requirements problems in your unit (developer) regression and/or test cases base, if any.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-4: Make notes on your defects.</td>
<td>T-4: Perform dynamic testing by executing code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T-5: Perform static (human) testing by reviewing your code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T-6: Record/write down test results.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Work Activities [8]

5) The test case design should be determined, created and revised to be prepared for testing (see activities P-3 - P-5 in Figure 5). The testing environment should also be prepared and checked if it is appropriate for the work to be conducted, to avoid problems later on (see activity P-7 in Figure 5).
Coding Activities

The coding activities include writing/rewriting code and compiling it [8]. The code should be based on the low-level designs to minimize the risk of not meeting the given requirements. There are also activities for recording compilation errors and detected defects to help developers monitor their work, evaluate the quality of the code and help them learn from their own mistakes (see activities C-1 - C-4 in Figure 5).

Testing Activities

The testing activities include the testing activities themselves and control of the test cases [8]. There is dynamic and static testing and test results recording included in the testing activities (see activities T-4 to T-6 in Figure 5). Controlling the test cases means checking if the code meet the given requirements and/or designs. It also involves checking the test cases for problems that have been noticed after coding (see activities T-1 - T-3 in Figure 5).

Evaluative Activities

The evaluative activities include analyzing the testing results and determining the next step (see activities E-1, E-2 in Figure 5). The activities should be conducted directly after testing to make sure they have a good workflow [8].

Debugging Activities

The debugging activities include identifying the source of errors and determining solutions to solve the errors (see activities D-1, D-2 in Figure 5). It is important to debug for the errors and identify their source to prevent similar errors from occurring again [8].

<table>
<thead>
<tr>
<th>Self-Assessment Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A-1: Assess your own development work.</td>
</tr>
<tr>
<td>• A-2: Identify causes of your mistakes.</td>
</tr>
<tr>
<td>• A-3: Identify improvement areas in your own way of working.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delivery Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• S-2: Deliver your code.</td>
</tr>
</tbody>
</table>

Figure 6. Post-Work Activities [8]
3.3 The Post-Work Activities

The post-work activities include activities that are conducted after the work activities have been done which includes self-assessment and delivery (see activities in Figure 6). Developers should make a self-assessment of their work in order to improve as a developer and reflect upon what has been done. They should include both mistakes and improvements to avoid similar mistakes. This will help them become more effective and efficient [8].
4. Research Methodology

This chapter presents the research methodology used in the thesis. First in section 4.1 there is an overview of the research strategy. Then in section 4.2 a description and motivation of the research type, followed by a presentation of the research phases conducted during the research in 4.3. Section 4.4 presents the sampling methods and research instruments used to collect data for the study. Validity Threats are addressed in section 4.5. The last section 4.6 presents the evaluation criteria as a part of the evaluation model.

4.1 Research Strategy

An appropriate research strategy had to be made in order to find out what activities are carried out and how time resources are spent by an individual developer in a software project. Due to the time limit of this thesis it was very important to have a proper research strategy to be as effective as possible. The survey sampling and the action research at the company were conducted simultaneously. The survey questionnaire was answered by 54 individuals and the action case had three developers.

The research strategy is illustrated in Figure 7. It includes the following components: (1) Research Methods, (2) Research Phases, (3) Research Instruments, and (4) Sampling Methods. The research conducted was of both a quantitative type as well as qualitative type with inductive reasoning. The research was conducted with three main phases: (1) a pre-study phase, (2) a study phase, and (3) an evaluation phase. The research instruments used were the SGD Framework, Scrum, a survey questionnaire, and time. The sampling method used was of a probability type the convenience sampling.

![Figure 7. Overview of the Research Strategy](image-url)
4.2 Research Type

The research type can be described with the following four keywords: (1) action research, (2) quantitative research, (3) qualitative research, and (4) inductive reasoning. More about these in the next following sections.

4.2.1 Action Research

The research method applied was an action research where the author of this thesis was also a participant developer. The steps included in the action research process are the following [21]:

- Selecting a focus
- Clarifying theories
- Identifying research questions
- Collecting data
- Analyzing data
- Reporting results
- Taking informed action

Following the action research process [21] provides a good guideline for the workflow. Due to the iterative nature of the project done at Swiesh QLurn IT Solutions, the collecting and analyzing data steps was done in several iterations, with a total of seven iterations. More iterations means more responsiveness and data. The larger the data amount the better the results.

Traditional research could have been an alternative but because the research question arose from the supervisor of this thesis, it was felt to be more relevant to conduct an action research study [8]. It was a research question based upon the needs of the supervisor for continual improvement in her own work.

4.2.2 Quantitative Research

The research type was chosen naturally due to the fact that numerical data (time) had to be collected and analyzed to fulfill the goal and purpose of the action research. This type of research is called quantitative research [22]. The alternative is a qualitative research but due to the more textual nature of such a method it was not suitable to gather numerical data [23]. The research was about numerical data and statistics, therefore the research should be of the quantitative type. Being of a quantitative type, the research could be described as a systematic empirical investigation of an observable phenomenon via statistical techniques. The aim of the research was to get a better understanding of how time resources are spent by an individual developer in a real world software project. Data collection was conducted via action research and literature study. The collected data about timesheets were analyzed to get this better understanding and come up with a theory about how much time is spent on different activities.
4.2.3 Qualitative Research

A qualitative research approach was found suitable to evaluate the usefulness of the SGD Framework and if it was useful in real world development efforts. An inductive qualitative method was used to conduct the campaign survey among industry based developers. Inductive method means that empirical data and observations were collected first and conclusions drawn from the empirical data. Empirical data was collected first using the survey questionnaire, then conclusions drawn from analyzing it, and based on these general theories.

Qualitative research as opposed to quantitative research is not research based on the results from the specific metrics, but is more of an interpretive character [23]. A description of qualitative research is that it tries to interpret why and what occurs in relation to a context or contexts. Unlike a quantitative method that is associated to numbers a qualitative approach is to words instead of numbers. Empirical data collection is semi-structured instead of structured and one works only with soft, rich data in contrast to quantitative research which mainly works with hard and reliable data, i.e. data that can be quantified in numbers or categories. The purpose of qualitative research is rather to find meaning and sense instead of detecting certain statistical data.

4.2.4 Inductive Reasoning

The research had a bottom-up approach or in another word, inductive reasoning. This means that the research started with observations and ended with more general conclusions and theories [44]. The alternative would be deductive reasoning which means that the research has a top-down approach instead. Then the research would have started with theories and ended with more specific observations to confirm the theories [44]. A deductive reasoning would not fit the need of the research because observations had to be conducted first to in the end get some general conclusions and theories about how time resources are spent by an individual developer in a software project. An image showing the differences of inductive versus deductive reasoning is shown in Figure 8.

Figure 8. Deductive versus Inductive Reasoning [45]
4.3 Research Phases

The research had the following phase, a pre-study phase followed by a study phase and concluded with an evaluation phase as illustrated in Figure 9 above. The action research steps mentioned before were included during these phases. Section 4.3.1 describes the pre-study phase followed by section 4.3.2, which is about the study phase of research. Finally section 4.3.3 presents the final evaluation phase.

4.3.1 The Pre-Study Phase

The pre study phase included the three first steps of the action research process: (1) selecting a focus, (2) clarifying theories, and (3) identifying research questions. The research started by selecting a focus for the research. After considering different approaches for the research it was decided that the research would be about the utility of the SGD Framework in helping software developers manage their workload, and how time resources are spent in software projects. To get a better understanding of the subject some literature studies were conducted, including reading about software development, time resources, personal process models, the SGD Framework and Scrum.

Time resources are always limited in both software projects and projects in general [19]. Managing time correctly is critical to be successful in the software industry. Spending the wrong amount of time on something can be devastating; projects can be delayed or even canceled. As Ian Sommerville wrote in his book, one of the most important goals in most projects is to deliver the software to the customer at the agreed upon time cost [20, p. 594]. However to do it, requires that one knows how much time is spent on different activities [20, p.595].

The potential benefits with finding out how much and where time is spent by an individual developer, would be that project managers can get a better view of how much time resources are generally spent and the distribution between activities. This can improve their resource management. It will also help
software developers improve their own way of working; knowing how much time usually will be spent on different activities in the course of their work.

The SGD Framework was studied thoroughly to get a better understanding of the framework before starting to use it. It is important that the framework is well understood for the research to be as successful as possible. A publication about the framework was read to get the necessary knowledge [8]. The agile software development framework Scrum was also studied because the company follows their own variant of Scrum. The purpose of studying Scrum was to get a good start when working with the company and their way of working on projects.

An illustration of the research phases is shown in Figure 9 above. Based upon the information gathered from the literature study the SGD Framework was enhanced by adding in a few more activities like high-level designs and reviews. Some more activities were added to the coding and testing categories in the SGD Framework. The survey instrument like the survey questionnaire was also enhanced to reflect these changes in the SGD Framework in this phase. This survey questionnaire would be used in the next phase i.e. study phase of the research.

An Evaluation Model was created in the pre-study phase. It consisted of ways to collect numerical data from the action research and note down the time spent on activities. It also comprised how the answers to the qualitative survey would be measured and what answers were to be expected from the survey among developers. All the phases are shown in Figure 9 in previous page.

4.3.2 Study Phase

This section provides an overview of the study phase consisting of three activities: (1) collecting data, (2) compiling weekly project results and (3) conduct campaign survey. This phase covered the following steps of the action research process: (1) collecting data and (2) analyzing data. The study phase also covered the qualitative survey questionnaire which was developed and disseminated among software engineers. Thus the study phase collected both time spent on software tasks and information about the usefulness of the SGD Framework, these are respectively the quantitative and qualitative aspects of the study phase.

The research method applied was action research; the structure of the action research process [21] was followed where all three of the software developers in the team were the subject of study. The research was of quantitative type, meaning that numerical data were collected and analyzed using mathematically based methods [22]. In this case it was time spent on activities in a software project that were collected and later analyzed. The research had an inductive reasoning, starting with observations and ending with theories. Data collection was conducted via action research and literature study. The software process model used was the SGD Framework.

To collect data, the Vizag Steel Plant project was selected to be conducted at the company. A software assignment was given and then it was implemented and tested following the SGD framework. After delivery a new assignment was given and so on. It was an iterative work flow and seven iterations were conducted before the end of the study phase of research. A team of seven members was assigned to
work on the project and three out of the seven were software developers. The time spent by these developers on various SGD Framework related activities was collected using an ordinary timer and scaled into timesheets.

On a daily basis notes were taken to keep track of the activities conducted in the assignment and time spent on them. The tool used for this was the software program Toggl, using this tool it was possible to keep track of what has been done and how long it took [23]. At the end of the work day, the data collected were translated into SGD activities and the time consumed were inserted into a time report sheet, see Figure 10. This time report sheet has one row for each of the SGD activities. There is also a column for each day of the week. For every week there is a sheet like this.

The Figure 10 below shows the weekly time sheet generated using Toggl software tool [23]. The time worked on various activities during the software assignment was thus collected daily and imported into this weekly time report used for collecting data. Each row corresponds to a SGD Framework activity and each column for the man hours it took daily to finish that activity.

![Figure 10: Weekly Toggl Time Sheet](image)
4.3.3 The Evaluation Phase

The evaluation phase consisted of two activities: (1) compiling the results on the whole project, and (2) analysis of the whole project results. This phase included the following steps in the action research process: (1) analyzing data, (2) reporting results, and (3) taking informed action.

At the end of the study phase, all data collected were in timesheets as mentioned before. By using mathematical methods was the data compiled into the final results. It went from several sheets for each week to one big sheet with results for the entire action research. Diagrams were also created to get a better picture of the results. The final results for the entire project were later analyzed to answer the research question and discuss what has been learned. This entire research process is shown in Figure 11.

![Research Methodology and its stages](image-url)
4.4 Sampling Methods

The sampling method used in the research was of non-probability type, a convenience sampling [46]. This is because every member of the sample population has a known and equal chance of being selected; and it is of availability reasons that this is the case. It would have be better with a sampling method of probability type. I distributed the survey questionnaire (shown in Appendix A) along with my own added questions to it among software developers that I knew of personally and who were working in the industry for more than 3 years. I also asked their help in getting other respondents to answer the survey and I was able to access many software professionals. I placed the questionnaire on my blog at https://softwaredevelopersblog.wordpress.com, which generated quite some responses. The responses collected were of software engineers with at least three years industry experience. A total of 54 software developers responded back to the survey and answered its questions in entirety or partially.

The action research part of the study used a convenience sampling too. The total number of software developers in the action research part of the study was three as these three developers were working in the team project. All three developers were chosen to be part of the study, as that is all the available developers working with the company’s project.

4.5 Research Instruments

The research instruments used during the research are the following: (1) Scrum, (2) the SGD framework and (3) Campaign survey questionnaire. Scrum was an essential instrument because it was the method used at the company to manage the project work. It was used as a guideline to work as effective as possible and to complete the project by the deadline.

The SGD framework was the main instrument during the study. It was used to keep track of conducted activities and the time spent on them, as mentioned in action research. Without the framework it would be necessary to create own activities and categories similar to the SGD activities described in Chapter 4. Luckily, the framework provided this kind of material and made the research easier to conduct.

The survey questionnaire was created with the help of an online tool FluidSurvey [14]. This was the main instrument of the qualitative research. It helped in finding the opinions of a random population of software developers about the SGD Framework and how useful it can prove to be for them. The campaign survey was conducted by disseminating this survey questionnaire to developers and also uploading it to the Web on a developer blog. It proved useful in collecting qualitative data and its analysis.

4.6 Validity Threats

For the survey we considered the four perspectives of validity and threats as presented in Wohlin et al. [22], i.e. Internal Validity, External Validity, Construct Validity and Conclusion Validity. Validity threats to internal validity of the research survey like transferability and generizability were also taken into account. These validity threats and the way we mitigated them in this research thesis is presented in full detail in Chapter 10 Analysis and Discussion. All validity threats are addressed in there.
5. Evaluation Criteria and Set of Desirable Features

This chapter presents an evaluation model in order to evaluate the Self Governance Framework and its utility to the average software programmer. Section 5.1 provides the background and justification for the evaluation model. Section 5.2 presents the first evaluation criterion of Completeness and section 5.3 for Understandability. Section 5.4 focuses on the evaluation criterion Time Resources.

5.1 Evaluation Model

Model Driven Architecture (MDA) [18] is a new approach to software development that moves standard code-centric software development to framework-centric software development. The basic idea is to specify system functionality with a platform and language independent framework and then translate this framework into platform specific model(s) and fully executable source code. It should provide for interoperability capabilities between different technologies, simplify the work of software engineers, reduce software development costs and support adaptation to rapid changes in technology [18]. In this thesis we shall investigate if the SGD Framework can indeed fulfill these criteria based upon research conducted on an application development project at Swiesh Technologies Inc, and with qualitative research conducted with the help of an online questionnaire for software developers. This chapter proposes an evaluation framework for the Self-Governance Developer Framework, setting out 3 key evaluation criteria and leads to outline, with a set of software requirements, the “portrait” of an ideal Developer Framework.

One envisioned advantage of using the Self-Governance Development (SGD) Framework is increased productivity. Since the SGD Framework can help generate a large amount of code from developers, it can save time in the development of a project, and it delivers software solutions faster from individual coders. Moreover it can incorporate the design patterns, templates, and best practices of leading software experts, which means that by using the SGD Framework it is possible to produce higher quality code than when applying traditional software development lifecycle methods. Also, the SGD Framework relies upon open standards hence there are no additional costs for SGD adoption.

As a case study for the SGD Framework in actual industry use, we used the SGD while developing a relatively simple web application called Glossary Management Tool (GMT). This was part of a project for the Vizag Steel Plant (www.vizagsteel.com) commissioned to be performed at Swiesh QLurn IT Solutions Inc, India. GMT is supported by a database and provides the regular database CRUD (Create/Repeat/Update/Delete) functions. The company used a SCRUM like approach to building software but for building the GMT only the SGD Framework was used. In the process of repeated implementation of the GMT web application using 3 different software tools, we came up with a set of criteria to evaluate the SGD Framework purely from the software developer’s point of view. To achieve this we combined evaluation criteria from other reported works [24, 25, 26, 27, 28, 29, 30], and in addition, we included several other criteria that we found useful based upon working experience. Criteria-based assessment is a quantitative assessment of the model in terms of usability,
maintainability and sustainability. This can inform high-level decisions on specific areas for process improvement. As a result we came up with the following 3 evaluation criteria for the SGD Framework:

1) **Completeness**: This criteria group evaluates coverage of developer activities in SGD Framework and if they cover all possible software development activities. This criteria measures to what extent does the SGD Framework cover all activities and processes that can come up during the course of a software project. Also checks if any development activities are left out in the SGD Framework, or if there are any redundant activities.

2) **Time Resources**: Sum of time spent on each individual activity within the scope of the project. The total time spent for each of the activities should be tabulated and then analyzed to find out how much time is spent on each activity.

3) **Understandability**: This criterion evaluates how straightforward is it to understand what the SGD Framework does, and its basic functions and scope.

### 5.2 Completeness

The criterion of completeness has been evaluated using three different approaches, the extended background study, the action research study and the survey questionnaire to get a comprehensive overview if all software activities are indeed included in the SGD Framework. We measured completeness of the SGD Framework by benchmarking it against the activities given in the PSP model [9], [10] and the PIPSI model [12] of the extended background literature review. We benchmarked them against these two models since the PSP and PIPSI have similar activities and design flow as the SGD Framework and cover the same area of research. Then the completeness criteria was tested in an actual action research carried out on the GMT project at the company to see if any software activities or tasks were to be added or were redundant in the SGD Framework. Finally, the revised SGD Framework was presented in the qualitative survey questionnaire and a whole set of questions was asked if all activities/tasks that can arise during a software project were indeed covered under the SGD Framework and if there were any problems or criticism of the same. These survey questions that were asked of respondents are shown in the table below (Table 3) and are also found in Appendix A.

The main motivation for evaluating the completeness of the SGD Framework using three very different approaches is that it provides for a more comprehensive measure of the completeness of the Framework. We find out if any activities are missing or not needed. We also find out how useful the SGD Framework proves to be in a real world application and survey of industry professionals. Based upon the literature study conducted, we expected a negative answer to completeness. We expected there would be several different activities or tasks that would be suggested to enhance the SGD Framework’s coverage.

Problems are the difficulties and consequences that you experience in a certain situation at a certain time. If you identify and analyze the problems, you can learn from them and thus prevent them from even arising again in the future. Using the criterion Problem, this study examined what problems software developers had encountered in the past during the development of their software process and
how they had solved these problems. The criterion was investigated using Question 11 in Interview Questionnaire 1 and Question 6 and Question 8 in Interview Questionnaire 2 (see Table 1).

5.3 Understandability

The main motivation to include “Understandability” as an evaluation criterion is that it is very important for a new software process model to be easy to use and be accessible to new users. Previous process models like PSP [9],[10] suffered from the pitfall that they were often difficult to understand and had too many complex metrics and features in them that failed to attract a sizeable market of developers. Software developers will not readily adopt a process model which is difficult to apply in their daily work situations or proves to be too rigid and inflexible to changing its functionality. Hence this criterion will help us judge the problem of how useful the SGD Framework will prove to be for individual developers in the long run.

We tried to measure the understandability and clarity of the Framework using a scale of 1 to 10 in the survey questionnaire distributed to developers. Measure of 1 meant that extensive explanations and documentation was required to understand the Framework while 10 meant it was very easy and intuitive what the Framework’s capabilities and application was. A score of 10 meant that no user documentation was required to understand it. We expected an answer of 8 to this criterion using the campaign survey since the SGD Framework is quite intuitive, responsive to agile and traditional methodologies and similar to most standards already in industry-use.

5.4 Time Resources

The criterion of time resources for development has been evaluated using a simple quantitative numerical analysis. The research method applied was action research in a company research project; the structure of the action research process [21] was followed where one of the software developers was the subject of study. The research was of quantitative type, meaning that numerical data were collected and analyzed using mathematically based methods [22]. In this criterion it was time spent on each activity in a software project that was collected and tabulated. On a daily basis a developer working with us took notes to keep track of the activities conducted and time consumed on them. The tool used for this was Google Sheets, using this tool it is possible to keep track of what has been done and how long time it took [23]. At the end of the work day, the data collected were translated into SGD activities and the times consumed were inserted into a time report sheet. Seven weekly time report sheets were generated in total for the duration of the project.

The main motivation for evaluating time spent on activities is that time is often the most important resource in a project [47]. The importance of time resources can be described with the following quote: "time is the scarcest resource, and unless it is managed, nothing else can be managed.", as Peter F. Drucker wrote in his book The Essential Drucker: The Best of Sixty Years of Peter Drucker’s Essential Writings on Management (Collins Business Essentials) [48]. The meaning of this is that time is limited and cannot be extended; it is not possible to create more time. You have to work with what you got and make the most of it. In a software project time is as important as in any other project. There are
deadlines that must be met by individual developers. Hence we evaluate time as a criterion to see how useful and time consuming the SGD Framework is in a real world development project, and if it can lead to savings in time spent on activities for the individual developer.

The expected answer was that the SGD Framework would prove positive upon time resources spent and would lead to better efficiency and productivity by the individual developer. This is because the SGD Framework is flexible, can be used in any order of activities and is quite open-ended in its application.

Table 1 shows the interview questionnaire which asked the questions relating to the three evaluation criteria and tried to measure them using the interview questionnaire. Interviews were conducted in rounds. Those who fulfilled the criteria from the first set of questions asked were then given the second form of interview questions and so on further.

5.5 Method

A method is a set of rules and criteria that form a repeatable description to perform one task and achieve desired results. In this study, the method corresponds to a method of application, the standard questionnaire developed for a functioning information system for SGD Framework. Although the software community strongly advocates agile methods where one allows some freedom to define and execute methods. There are some basic general steps that must be made in a clear order (Boehm, 2002) [42]. Failure to follow the obvious scheme can cause many problems and thus endanger the entire adaptation work. For example, you must define the entire business first, at least one rough level before creating a system to support it. However, some parts of the business may change in parallel with the implementation as you get a better understanding of operations. Using the Evaluation criterion Method, this study examined the proposed Framework’s choice of method steps and order between these. The purpose was to study the Framework in its entirety and find out how it went, interacted with each other method in use and with the individual developers. The survey was done through successive questionnaires as detailed in Appendix A.

In Table 1 the questions that were asked of all interview-taking software developers to help evaluate the three criteria are shown. The completeness criterion starts off by asking the developers if the SGD Framework incorporates all possible activities and tasks are included which can occur during the course of software development. It asks if the Framework is comprehensive and complete in listing out all developer activities. It then asks if the developers can think of any activities or tasks they can think of which could be included within the final Framework to make it complete. There are further detailed questions addressing each specific category of the SGD model in its Pre-Work, Work and Post-Work phases which ask for the developer’s opinion regarding the detailed listing out of activities. Finally it ends with asking if any activities are redundant or better be left out of the SGD Framework.
Table 1. Survey Interview Questions for Interviewees

<table>
<thead>
<tr>
<th>Interview Questions for Interview 1</th>
<th>Interview Form 2 for interview rounds 2-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questions asked of all interviewees:</td>
<td></td>
</tr>
<tr>
<td>What is your current position?</td>
<td>What is your current position?</td>
</tr>
<tr>
<td>How many months have you held this position at your current employer?</td>
<td>How many months have you held this position at your current employer?</td>
</tr>
<tr>
<td>How many months have you held this position in other companies?</td>
<td>How many months have you held this position in other companies?</td>
</tr>
<tr>
<td><strong>If you are a developer:</strong> Are you currently conducting traditional, agile or both types of software development?</td>
<td><strong>If you are a developer:</strong> Are you currently conducting traditional, agile or both types of software development?</td>
</tr>
<tr>
<td><strong>If you are a tester, project manager, or other role:</strong> Is your company currently conducting traditional, agile, or both types of software development?</td>
<td><strong>If you are a tester, project manager, or other role:</strong> Is your company currently conducting traditional, agile, or both types of software development?</td>
</tr>
<tr>
<td><strong>If you are a developer:</strong> How many months have you been conducting traditional, agile or both types of software development?</td>
<td><strong>If you are a developer:</strong> How many months have you been conducting traditional, agile or both types of software development?</td>
</tr>
<tr>
<td><strong>If you are a tester, project manager, or other role:</strong> How many months have you conducted traditional, agile or both types of software development (in the past)?</td>
<td>What problems did you experience in the process of using your model?</td>
</tr>
<tr>
<td>If you have to follow given development models/methods, please enter the name of the model or method:</td>
<td>Do you have any suggestions on changes in your current method(s)?</td>
</tr>
<tr>
<td>What is your software development model like?</td>
<td>If you want to explain what is your view upon current developer-centric models in software development?</td>
</tr>
<tr>
<td>Which parties are involved in the project?</td>
<td>How did the initial adaptation work and what problems were experienced?</td>
</tr>
<tr>
<td><strong>If you are a developer:</strong> Are you currently responsible for conducting implementation, unit testing or both?</td>
<td>How did you solve initial initialization problems?</td>
</tr>
<tr>
<td><strong>If you are a tester:</strong> What testing level(s) are you responsible for?</td>
<td>What was the outcome of these solutions?</td>
</tr>
<tr>
<td>Ties the over-riding design of the model with the design of your existing model of software development</td>
<td></td>
</tr>
<tr>
<td>What are the similarities?</td>
<td></td>
</tr>
<tr>
<td>What are the differences?</td>
<td></td>
</tr>
<tr>
<td>Please enter your email address.</td>
<td></td>
</tr>
</tbody>
</table>
Table 2. Understanding the SGD Framework

<table>
<thead>
<tr>
<th>Statement</th>
<th>Question</th>
<th>Answer YES/NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>The SDG Framework consists of sets of activities that may look as if they were distributed across various phases (categories). However, we strongly remind that the framework does not impose any order of the activities within the process. It only shows what activities are relevant for your work.</td>
<td>Have you understood that the framework only lists types of activities that you MAY use in your development work in the order that suits you? It does not show any order that you must follow!</td>
<td></td>
</tr>
<tr>
<td>The SDG Framework does not place any of the activities within any process phase. It only categories them into logically related categories of activities.</td>
<td>Have you understood that the framework categories do not correspond to any predefined development phases?</td>
<td></td>
</tr>
<tr>
<td>The SDG Framework’s activities may be conducted whenever they are needed, depending on the context at hand. For instance, the activity <strong>PL-13: Plan for managing any identified risks</strong>, can be conducted throughout the whole process.</td>
<td>Have you understood that the activities in the framework’s categories may be conducted whenever in an executed process, that is, in any of your implementation steps?</td>
<td></td>
</tr>
<tr>
<td>When developing the SDG Framework, we tried to be exhaustive and identify all types of activities that are relevant for development process. However, not all the framework’s activities need to be used in all development contexts and some of the activities may be used more than once. For instance, the activity <strong>PR-6: Sign your personal Service Level Agreement</strong> is not practiced within every company whereas <strong>C-1: Write/rewrite your code</strong> may be conducted many times.</td>
<td>Have you understood that the framework’s activities may be used from zero to many times in an executed development process?</td>
<td></td>
</tr>
<tr>
<td>Many of the SDG Framework’s activities may be implemented by the tool you are using. If so, then the activity should be ticked off as being used and, of course commented, on.</td>
<td>Have you understood that the activities that are managed by the tool are automatically carried out?</td>
<td></td>
</tr>
</tbody>
</table>
In Table 1 the questions that were asked of all interview-taking software developers to help evaluate the three criteria are shown. The completeness criterion starts off by asking the developers if the SGD Framework incorporates all possible activities and tasks are included which can occur during the course of software development. It asks if the Framework is comprehensive and complete in listing out all developer activities. It then asks if the developers can think of any activities or tasks they can think of which could be included within the final Framework to make it complete. There are further detailed questions addressing each specific category of the SGD model in its Pre-Work, Work and Post-Work phases which ask for the developer’s opinion regarding the detailed listing out of activities. Finally it ends with asking if any activities are redundant or better be left out of the SGD Framework.

The criterion for time resources starts by asking if the developers think the SGD model is efficient and will lead to savings in time. Two more questions ask if the SGD Framework will make it easier for the individual developer to manage his finite time resources and be more productive in his work duties. The final survey question asks if the SGD Framework has taken time resources into account during its formulation. The criterion of time for development was mostly measured by numerical data collection during the action research phase of the thesis.

Table 2, the criterion for Understandability asks how straightforward the SGD model is to understand and its purpose. It asks whether the users and market for the Framework is clear to understand and how they can benefit from using it to manage their daily work. It surveys on a scale from 1 to 10, the ease of understanding the SGD Framework, score of 1 meaning very difficult ranging to a score of 10 meaning it’s quite self-evident and intuitive to understand. The score on this scale of 1 to 10 gives us a good idea of the clarity of the SGD Framework and can be used as a benchmark against the older PSP and PIPSI Frameworks. The remaining questions ask if a design rationale has been provided along with architectural overviews with diagrams. This is to see if the developer can readily understand the paper that introduces the SGD Framework and describes it briefly. Tables 1 and 2 had been appended to the original survey questionnaire and can be used to get a measure of the three evaluation criteria.
6 Research Results for Evaluation Criteria- Completeness

We will begin presenting the results of the research conducted during 2017 for evaluating the SGD Framework among working computer professionals. The research instruments used were the qualitative survey questionnaire; interviews and literature gathering. This chapter presents the research results obtained for the criteria of ‘Completeness of the SGD Framework.’ The first section 6.1 will present the background to the major part of the research performed and how results were gathered. Section 6.2 will show the results obtained from the qualitative survey questionnaire that was distributed among software professionals. Section 6.3 will present the results as obtained from the action research study at a real world project. Section 6.4 will present the results as obtained from the course of the literature study.

6.1 Background to the Questionnaire

To get a better perspective of the results a brief overview will be presented of what had been done in the thesis project before the research started. I received the statement of the SGD Framework, introducing it and demarking its various structure/activities from my examiner. I also received a model questionnaire which had been developed by her and is shown in Appendix A. This qualitative research questionnaire deals with all the roles and activities of the proposed framework and is targeted to a software developer target audience.

I distributed this questionnaire (shown in Appendix A) along with my own added questions to it among software developers that I knew of personally and who were working in the industry for more than 3 years. I also asked their help in getting other respondents to answer the enhanced, revised questionnaire, and so I was able to access many software professionals. I also placed the questionnaire on my blog at https://softwaredevelopersblog.wordpress.com, which generated quite some responses. The entire process of sending out the questionnaire and getting feedback from respondents took approximately 2 months. During this time I carried out the action research at my company Swiesh and also interviewed my two fellow team members.

The presentation of the research results will follow the evaluation criteria model described in Chapter 5. A note to consider is that some activities, which are not presented, either were not answered at all or only a small number of respondents answered them-- so it did not feel necessary to include them in the final results. A lot of data was gathered by this qualitative survey of software developers which is available upon request; what is presented here in this chapter is a selection out of this data.

Following this background section will be the results obtained from the survey questions. They cover both what the individual questions asked in the fourth round of interviews with interviewees and the results to each individual sub-category of the SGD Framework (like Preliminary Activities sub-category for example). The full results of the research conducted will follow these in this chapter.
Table 3: Evaluation Criteria: Completeness of the SGD Framework (53 responses to this criterion)

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Yes/No, majority of responses obtained…</th>
</tr>
</thead>
<tbody>
<tr>
<td>To what extent does the SGD Framework cover all tasks or activities that can come up during a development project?</td>
<td>Not all extent, some activities are missing. 85% said no.</td>
</tr>
<tr>
<td>Is the SGD Framework comprehensive and complete in listing out all developer activities?</td>
<td>85% said No. Some activities should be added.</td>
</tr>
<tr>
<td>Are any activities left out in its categories?</td>
<td>Yes</td>
</tr>
<tr>
<td>Can you list anything that you want to see included in the final version of the Framework?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Preliminary Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Planning Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Preparatory Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Coding Activities category?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

6.2 Questionnaire Results to the Completeness of SGD Framework

Presented here are the results I got to the survey questionnaire and the additional questions that were appended to the original questionnaire. These additional questions have been shown in Appendix A. They can be seen in Appendix A along with comments they generated.

More than 85% of the respondents felt that the framework stopped abruptly with delivery of the code and sign-off agreements, and did not address the issue of maintenance of source code. Most of them had comments like “developer’s job does not end with delivery of code only”, “what about maintaining and revising existing code?” etc. This was a major finding of the qualitative survey. More than 85% of respondents commented about maintenance issues and that it should be included in the “Post-Work Activities”. The same numbers of respondents were not sure if the framework was aimed at the
individual developer or at a team of developers, there was quite some confusion if the SGD framework could be implemented as a team effort or if it’s only meant for solo developers like the PSP model.

Debo Jyoti Roy wrote that “Development work does not finish with delivery of code. It needs developer support and activities during integration, system testing and performance testing and the like. How do those activities fit in this cycle?”

Tumu Satish advised us to look at “several other frameworks like Prince, RUP available. Please refer to them. For agile, TFS is an example” to get a more complete picture of building the framework.

Vijay Vemana said that it is “Advisable to bring technical (Requirements, design, development, Reviews & Testing) along with Managerial (Project, team, risks, and changes management) in a layered approach with synchronizing to conventional and agile approaches.”

More than 90% of respondents wrote that the SGD Framework was not complete. In addition to the various suggestions shown below in their comments, they also felt that maintenance and upgrading of source code was a big issue. 85% of respondents asked about Maintenance issues and how the SGD Framework incorporates maintenance of the software product once it’s been developed. They also asked about system integration into larger pieces of software. As Tom suggested along with system integration & testing; Design Review and Code Review are activities which must be addressed, for secure code.

**Results to Preliminary Activities**

In this subsection the responses to the list of *Pre-Work activities* i.e., Preliminary and Planning activities is presented. As can be seen in the questionnaire in Appendix A, these were the first question(s) asked. The responses were quite varied and generally critical of the Preliminary Activities module. 67% of respondents felt that this was a re-wording of Requirements gathering, and asked where did we obtain the Preliminary Activities from? Tom Russ asked if:

“Did they materialize out of nowhere?”

“Is there no design activity? I also see no mention of task estimation.”

Siva Prasad thought that PR-5 (Review and revise your personal implementation and unit (developer) way of working) should be re-named to “Tester Methodology”.

Sateesh Neeladri wrote that “More inputs should be given on Requirements and documenting the requirements. Simultaneously, testing team has to write the test cases.”

15% of respondents who were using traditional methodologies wanted us to freeze the requirements itself and introduce those as Preliminary Activities in framework.
Results to Planning Activities

Tumu Satish felt that Planning Activities could be established/documentined in a formal plan which would reduce much misuse in Agile practices. He felt that many of the Planning Activities were informally followed in Agile practices; and especially PL-12 (identify Risks related to your plan) and PL-13 (Plan for managing any identified risks) were much needed activities.

Suneetha felt that for PL-4 (Determine and document your implementation and unit developer testing goals) might be difficult to document goals at such an early stage. She expressed ambiguity about several other activities like PL-5, PL-6, PL-8 at such an early stage of the process. (For instance PL-5: “Determine your implementation and unit (developer) testing strategy”, PL-6: “Determine appropriate implementation and testing practices”, PL-8: “Set your own personal deadlines to be met during your implementation and unit (developer) testing work.” She thought they depend a lot on developer’s schedule which may change drastically.

Sateesh Neeladri thought a missing activity is “Prototype which is never specified here. Moreover, once Requirements are drafted, then high level design followed by Low level design should be implemented.”

Sarvani Devara thought the activities work well with all methodologies.

Sadhu Khan felt that unit testing is “usually very small and does not need elaborate planning”, hence PL-4 (Determine and document your implementation and unit (developer) testing strategy) would suffice for unit testing.

Tom Russ wrote that in Planning Activities “you seem to be using unit-and-developer-testing as synonyms. Perhaps that is your intent but unit testing can be performed by a developer or a dedicated tester. Also I see no mention of integration (also called system) testing.”

45% of respondents wrote that the Planning Activities were sufficient for both traditional and agile methods of development and saw no need to alter anything in Planning Activities.

Results for Preparatory Activities

Fully 66% of respondents wrote that they did not follow PR-2 (Prepare an impact analysis of your low level design(s)) or PR-3 (Determine order of unit (developer) test cases and their order); or they used them informally in Agile development. That is, no record was kept of PR-2 or PR-3 in their work.

Tumu Satish felt that “High level design or architecture was missing” in Preparatory Activities list which is mandated in traditional software development, but not so much in Agile practices. Nine other respondents also asked about high-level design(s). Sateesh Neeladri wrote “Low level design comes after high level design where high level design and prototyping are never specified. Test cases written by the tester play a vital role when compared to the developer test cases which was never specified”.

Tom wrote: “it might be wise to explicitly state that your test cases include both abuse cases and use cases. It leaps out at me that the security of your code is not mentioned anywhere.”
65% of respondents felt that P-6 (Create or modify stubs and drivers) was not mandatory for all projects. 90% of respondents said they never used P-6 in their work.

**Results for Coding Activities**

Vijay Vemana felt that “code coverage aspects could be included as a separate activity” in the Coding Activities list.

All 19 of the Agile practitioners said they implemented every activity from C-1 to C-4 given in this list.

Tumu Satish wanted us to incorporate 1) coding standards; and 2) ensure traceability; which he felt was missing in Coding Activities. He also felt that both C-3 (make notes on your compilation errors, if necessary) and C-4 (make notes on your defects) were ignored in Agile practice, but very much required and thus good activities to include.

Tom Russ wrote that “Code Review is the 2nd most important process element in writing secure code and I don’t see it mentioned here.”

Sateesh Neeladri also noted that code review was missing as an activity, which is mandatory at the development phase of coding. And that “Developer Release Notes (DRN was never specified)” in the Coding Activities. Other than that he had no more suggestions. Sarvani Devara noted that she only sometimes made notes on her defects (activity C-4).

Both agile and traditional SDLC practitioners said no activity was redundant or should be reworded of the 4 Coding Activities from C-1 to C-4. They only wanted us to include more activities like code review and more time analysis for writing/rewriting code. Tom thought code security was a concern.

**Results for Unit Testing Activities**

Ten percent of the respondents felt that unit test automation could be added to this list. Tumu Satish wrote that T-3 (Remedy requirements problems in your unit (developer) regression, and/or test case base, if any) was context oriented. He felt that T-4 (Perform dynamic testing by executing code) was context-oriented too and could vary in traditional and agile approaches. Especially sprint runs would use these activities many times. Tumu felt that recording/writing down test results (T-6) was a good practice that is much needed in agile approaches.

Siva Prasad thought that performance testing was a missing activity which can be included. And a code review by a peer as is used in pair programming practices. Jyoti Roy wanted code coverage as an additional activity.

Tom Russ said that a missing activity was “integration (also called system) testing. You should make sure that your unit integrates into the larger system of which it is a part.” He also wrote that “It might be wise to explicitly state that your test cases include both abuse cases and use cases.”
Sadhu Khan commented on T-5 (Perform static testing by reviewing your code) that “Different people have different style of coding. The same logic can be written differently. If I review someone else's code, I always feel it could have been done in a better way...” None of the respondents thought any activities were redundant or required to be rephrased.

**Results for Evaluative Activities**

Three of the respondents (15% of total) said they performed Evaluative Activities E-1 (Analyze your unit (developer) testing results) and E-2 (Depending on the unit (developer) testing results, determine your next step(s)) in a different manner.

Tumu Satish noted that crucial activity which was missing was “system and acceptance testing steps are missing. These are context oriented and are needed both in Agile & Traditional development practices.” He said E-1 (Analyze your unit (developer) testing results) was done in a formal way in Agile practices.

Tom Russ said “if you perform estimation, an evaluative activity might be to record the actual time required to give you feedback so your future estimates can get better.” “The most important security practice is Design Review and it is not mentioned either.”

**Results for Debugging Activities**

Sateesh Neeladri noted a missing activity was Impact analysis.” Fixing the error is fine but need to check the impact when fixing an issue like the current fix may impact something else. As such regression is required whenever any issue is fixed.”

Jyoti Roy said a missing activity was “Regression Testing after debugging.” Otherwise we had no important comments about Debugging from the developers.

**Results for Self-Assessment Activities**

Sateesh Neeladri mentioned developers could go for RCA. “Root cause analysis (Fish bone diagram here will be helpful) is missing here which will help in reducing errors and increasing productivity. This is called as process improvement.” Suneetha said that “sometimes we cannot identify the issues in our code so peer review can help.” Others (30%) said Self-Assessment was a part of PSP.

**Results for Delivery and Sign-Off Activities**

There were no major suggestions to this section. Respondents simply agreed with the activities listed in the SGD Framework’s category for this section. A few respondents stated that they really did not carry out any sign-off deliveries and that this part of the work was usually informal in their daily activities. Hence, they did not have any suggestions for improvement.
6.3 Action Research Results to the Completeness of SGD Framework

In this section is the coverage of developer activities in the SGD framework presented. The results for this section were drawn from the action research study conducted at the company Swiesh, while working on a real world project using the SGD Framework. Nineteen of the activities were not conducted during the course of the project but there was not a single activity conducted that was not included in the SGD framework. Thus, the Framework included all activities that were performed during the project.

The following activities from SGD Framework were not conducted or felt to be redundant during the software project at the company:

1. Revise and understand any appropriate internal (organizational) and external standard(s).
2. Determine and document your implementation and unit (developer) testing goals.
3. Identify standards to be used for meeting your goals.
4. Set your own personal deadlines to be met during your implementation and unit (developer) testing work.
5. Estimate effort and resources required for carrying out your work.
6. Schedule your work.
7. Review your implementation and unit (developer) testing plan to ensure that it is realistic and achievable.
8. Identify risks related to your plan.
10. Prepare (make) an impact analysis of your low-level design(s).
11. Determine the types of unit (developer) test cases and their order.
12. Revise the existing unit (developer) regression test base, if relevant.
13. Create or modify stubs and drivers, if required.
14. Prepare your unit (developer) testing environment and check whether it is appropriate for your work.
15. Make notes on your compilation errors, if necessary.
16. Make notes on your defects.
17. Check whether the unit (developer) regression test base meets the given requirements and design.
18. Remedy requirements problems in your unit (developer) regression and/or test cases base, if any.

19. Record/write down test results.

The reason for nineteen activities being left unperformed is that everyone works in their own way and I did not conduct most of these activities during the part of the software project that I was there. Some of the activities were conducted but the time spent on them was not included in the daily reports due to the very small amount of time spent. One example is compiling the code, I did that but it took only seconds every time so it did not feel necessary to include it. It is good that all activities and tasks that needed to be performed were included within the SGD Framework. It shows that the Framework is quite complete to be used in a real world project and that it meets the first evaluation criteria of Completeness. The action research study proves that the SGD Framework is complete in its coverage of developer activities, in fact some nineteen activities were found unnecessary during the course of the action research.

I also think there was organizational inertia among us developers when having to adopt a new framework for developing software. Most projects at Swiesh are conducted on a Scrum basis and this is perhaps reflected in the developers saying that Scrum methods would have contributed to savings in time spent. This is also reflected in the fact that trying to learn about the new framework and resolving uncertainties about it took up to 16 hours of time during the project.

6.4 Research Results to Completeness of SGD Framework from the Literature Review

The SGD Framework was benchmarked against the existing software development models (PSP and PIPS) discovered during the extended background study of this thesis. Activities and features of the SGD Framework were benchmarked against Personal Software Process (PSP) of Watts Humphrey and against the Process for Improving Programming Skills in Industry (PIPS). These two process models were discovered during the course of the literature study and are described in detail in chapter 2. These two specific process models are chosen since they are oriented towards the individual software developer (similar to the SGD) and are also similar to the SGD Framework in their scope and their application.

The PSP [1] developed by Watts Humphrey is a process to be followed by an individual programmer, not a team of programmers – hence the name Personal Software Process. While many software processes are followed by a whole team, an individual programmer can practice the PSP even if his teammates are not using those practices. The PSP is a structured framework of forms, guidelines, and procedures developed by Watts Humphrey of the Software Engineering Institute. The framework guides a programmer in using a defined, measured, planned and quality controlled process. Like other processes shown in the Figure 12 it is a plan-driven development process. Figure 12 shows three development processes which are plan based methodologies, Personal Software Process (PSP), the Process for Improving Programming Skills in Industry (PIPS), and Rational Unified Process (RUP). These three processes are somewhat similar in structure although new versions of the RUP could be classified the most agile, and PSP as the most plan-driven methodology.
The Personal Software Process (PSP) of Watts Humphrey [1], the Process for Improving Programming Skills in Industry (PIPSI) [12] and Rational Unified Process [49] are all plan driven methodologies. The PIPS'I follows the three stage plan of Pre-Work Activities, Work Activities, and Post-Work Activities in order. The RUP methodology in its newer versions is more flexible and well suited for rapid agile practices like Scrum cycles. Before beginning, it is important to understand that there is not a sharp dichotomy between plan-driven and agile software development methodologies. So, these three methodologies have some elements of agility to them or can be slightly modified to incorporate agility. The PSP is probably the most plan-driven of the three process models. Some common characteristics of plan-driven frameworks, although these guidelines are more relaxed in smaller projects, are:

- Focus on repeatability and predictability
- Defined, standardized, and incrementally improving processes
- Thorough documentation
- Detailed plans, workflow, roles, responsibilities, and work product descriptions
- Process group containing resources for specialists: process monitoring, controlling, and educating
- On-going risk management
- Focus on verification and validation
SGD Framework Benchmarked against Personal Software Process (PSP)

The PSP is a structured framework of forms, guidelines, and procedures developed by Watts Humphrey of the Software Engineering Institute [1]. The PSP framework guides a programmer in using a defined, measured, planned, and quality controlled process. However, another purpose of the PSP framework is to help engineers understand their own skills so they can modify the process to meet their personal needs and preferences and to improve their own personal performance and time resources.

PSP training follows an evolutionary improvement approach. An engineer learning to integrate the PSP into his or her process begins at PSP Level 0 and progresses in process maturity to PSP Level 3 (See Figure 9). Each level incorporates new skills and techniques into the engineer’s process—skills and techniques that have been proven to improve the quality of the software process. Each level has detailed scripts, checklists, and templates to guide the engineer through required steps. The scripts, checklists, and templates are only a starting point, however. The PSP provides quantitative metric-based feedback that helps each engineer improve his or her own personal software process, efficiency, and collect time spent on each activity.
PSP has several strong tenets as follows:

1. The first is that the longer a software defect remains in a product, the more costly it is to detect and remove it. Therefore, thorough design reviews and code reviews are performed at every level for efficient defect removal.

2. The second tenet is that defect prevention is more efficient than defect removal. Careful designs are developed and data is collected to give additional input on where the programmer should adjust their own personal software process to prevent future defects.

3. Time recording log—another important PSP activity is recording how much time is spent on development activities. When a developer starts to work, he records the date, the start time, and the development phase. If the developer is interrupted during work, the number of elapsed minutes of the interruption is recorded. When activity is completed, the stop time and any comments are noted.

PSP training is based on four levels of personal process: PSP Levels 0 through 3, as shown in Figure 9. Skills at one level are mastered before the programmer moves to the next level of personal process improvement.

**Level 0 (Personal Measurement):** The input to PSP is the requirements; requirements elicitation is assumed to have been completed and a requirements document delivered to the programmer. The PSP0 has three waterfall-like phases: planning, development (including design, code, compile, and test), and a postmortem. In the postmortem, the engineer ensures all data for the projects has been properly recorded and analyzed. The software engineer begins by establishing a personal baseline of her current development process by basic measurements, such as the time spent on a program, the defects injected and removed in each development phase, and the size of the program (in lines of code), and creating some initial reports. This level 0 is then improved by adding a coding standard, a size measurement, and the development of a personal process improvement plan (PIP).

The main advantages of PSP have been demonstrated by several studies, including (Hayes and Over, 1997; Humphrey, May 1996; Ferguson, Humphrey et al., May 1997):

- Improved size estimation and time estimation
- Improved productivity
- Reduced testing time
- Improved quality

In comparison with the PSP framework, the SGD Framework could be improved by adding in thorough code reviews and design reviews in the Work Activities category. That would vastly improve the SGD Framework as against the Personal Software Process (PSP) of Watts Humphrey. This would take care of tenet 1 listed above.

The second tenet is that defect prevention is more efficient than defect removal. The SGD should incorporate detailed metric and data collection to give additional input to the programmer and help
preventing future defects. If careful designs and data are collected, it would help the individual developer in knowing their weaknesses and where coding/development errors might creep in.

The PSP collects detailed data and metrics about time spent on each activity in the development process, including breaks and interruptions in the work. Collecting quantitative data about time spent on individual activities or tasks is missing in the SGD Framework and could greatly increase efficiency and spending of limited time resources. As we know, time resources are one of the greatest concerns during software development and the least predictable. In this thesis, we have tried to collect data on time spent on individual activities as part of the action research study, and we feel that this is an important aspect of any software process framework.

A great advantage of the SGD Framework over the PSP we noticed is that the SGD Framework is much more flexible and easier to use than the PSP. The PSP framework is very structured, rigid, and follows almost a waterfall model like phases in its usage with complicated scripts, forms, templates, standards, and checklists (each of which have to be filled out in detail). The SGD does not incorporate any rigidly structured plan-driven pattern like the PSP framework; SGD Framework provides a much greater level of autonomy and open-mindedness to the individual developer. The activities in the SGD Framework can be used in any order anywhere in the development process at the choice of the software developer. This is a much more open approach to software development and bypasses the inherent rigidity of the PSP framework’s graduated levels. This would lead to more developers using the SGD Framework out of their own volition and due to its flexibility.

**SGD Framework Benchmarked against Process for Improving Programming Skills in Industry (PIPSI)**

PIPSI is built around the following key concepts:

- A Personal Process to be used and adapted by the individual developer. The three main phases of the process are Pre-Build, Build, and Post-Build. The Personal Process is the foundation needed to define Personal measurement and to use specific techniques and methods.
- Personal Measurement which introduces personal measures (effort, size, and defects) to monitor and improve the personal process.
- Personal Project Management which introduces metrics and techniques to estimate the size and effort of a project, plan a project, schedule it, and track its advancement.
- Personal Quality Management which deals with defect collection and analysis to track the quality of a project. Further, it introduces design and code reviews to improve the quality of a project.

The focus of PIPSI is on bottom-up process improvement. The entire model is buttressed and controlled through the use of measurement. By collecting data on their own activities, software engineers learn about how they develop software. The quantitative measures help them understand the fundamental relationship between size and effort and, through this understanding, enable them to improve their
estimating abilities on the time spent on activities. Furthermore, by gathering data on their defect rates they witness how using practices such as personal code reviews and the use of checklists will allow them to produce higher-quality software products.

In the PIPS! Framework the three phases of Pre-Build, Build and Post-Mortem is to be followed in a sequential order and they use different forms, metrics and checklists to collect data. This data is then analyzed and provides feedback to the developer to improve their work. In the PIPS! Framework just like in the PSP Framework these phases have to be carried out in a sequential fashion. This is unlike the flexibility of the SGD Framework in which any activity might be carried out in any order or point of time, or activities can also be used multiple times.

In benchmarking the SGD Framework against the PIPS! Framework it was observed that greater amounts of both numerical and qualitative data are gathered in the PIPS! model. PIPS! focuses on proven quality control mechanisms such as design reviews and code reviews which enable developers to remove defects earlier in the software development process. This achieves the twin objectives of removing defects at the front end of the development cycle where they are cheaper and easier to fix and, as a corollary; means testing time is more focused as fewer defects are escaping into test. Design reviews and code reviews could be incorporated into the SGD Framework to overcome this deficiency. We believe the incorporation of such tools in the SGD Framework will ensure the buy-in of participants and subsequent continued usage of the SGD Framework.
7 Research Results for Evaluation Criteria- Understandability of the SGD Framework

In Section 7.1 presents the research results obtained through the qualitative survey questionnaire about the second evaluation criterion- Understandability. The chapter presents the views of the survey respondents about how straightforward it is to understand and implement the SGD Framework in their regular work activities. Section 7.2 presents the results of the action research study, how easy it was to incorporate the SGD Framework’s activities in the programming project, how clear was it to implement and how much time did it take to understand the functions and activities of the SGD Framework. Section 7.3 presents the results obtained by benchmarking the SGD Framework against other software process frameworks mentioned in the literature review and how useful it is to the programmer.

In our survey we asked the respondents to rate how straightforward the SGD Framework was to understand on a scale of 1 to 10. A rating of 1 corresponds to impossible to understand or use and the rating of 10 means that the Framework needs no introduction or training to use at all. We asked all the respondents after they had read the paper about SGD Framework to rate it on this scale of 1 to 10, and also give their comments and feedback.

Table 3: Evaluation Criteria: Understandability of the SGD Framework (42 respondents to this Criterion)

<table>
<thead>
<tr>
<th>Understandability</th>
<th>Yes/No, majority of respondents...</th>
</tr>
</thead>
<tbody>
<tr>
<td>How straightforward is it to understand:</td>
<td>Answer received:</td>
</tr>
<tr>
<td>• What the SGD Framework does and its purpose?</td>
<td>Not straightforward, better introd</td>
</tr>
<tr>
<td>• The intended market and users of the SGD Framework?</td>
<td>uction required</td>
</tr>
<tr>
<td>High level description of what/who the SGD Framework is for is available.</td>
<td>No</td>
</tr>
<tr>
<td>High level description of what the SGD Framework can do is available.</td>
<td>Yes</td>
</tr>
<tr>
<td>High level description of how the SGD Framework performs is available.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

On a scale from 1 to 10, how do you rate the clarity of the SGD Framework? List a digit from 1 to 10 in right column.

Score 1 implies needs extensive documentation

Score 10 implies very clear and intuitive to understand.

(average of all answers received)
7.1 Survey Questionnaire Results for Evaluation Criteria-Understandability

More than 75% of the surveys gave a rating of 6 or lower to the understandability of the SGD Framework and its intended market of users. Only one respondent gave it a rating of 10 on the ease-of-use scale. Six respondents gave it a rating of 8 on the scale and five interviewees gave it a rating of 7 on the scale.

More than 75% of the respondents said that it was hard to understand what the SGD Framework does and how it can be utilized. They said they required better introduction and documentation to help them understand it, and that the supplied paper was not sufficient. More than 75% of respondents wanted us to come up with a clearer and more precise description of the SGD model and how its activities could be useful to software developers.

Debo Jyoti Roy wrote “In my opinion, it is more of a methodology/guidelines/principle rather than a framework. Also, I did not understand the significance of self-governance in the name.”

Inspite of the introductory paper we distributed to all developers (which none felt to be too detailed), we noticed the following facts about understanding the paper and the SGD model:

Fully 77% did not understand that the framework only lists types of activities that you MAY use in your development work in the order that suits you. It does not show any order that you must follow!

Fully 80% of those surveyed did not understand that the framework categories do not correspond to any predefined development phases.

Tom Russ said that “the provided paper does not give clear step-by-step instructions, and that it does not list available resources for further reading.”

Sateesh Neeladri stated that a “better high level overview of the SGD Framework could be provided.”

Sixty eight percent did not understand that the activities in the framework’s categories may be conducted whenever in an executed process, that is, in any of your implementation steps during the development phase. Also, more than 65% did not understand that the framework’s activities may be used from zero to many times in an executed development process. There was also much confusion about the purpose of the Framework and whom it was aimed toward. More than 65% of the surveyed people thought that it was to be used as a regular SDLC within a team setting. The same people recommended that we try out the SGD in small team efforts before launching it to MNC’s and bigger development companies.

Seventy three percent responded that the diagrams and figures provided in the paper were not sufficiently clear or help them understand how the framework could be implemented in use cases.

7.2 Action Research Results for Evaluation Criteria- Understandability
I observed similar trends in the action research conducted as in the survey questionnaire about the clarity and straightforwardness of the framework. A majority of time in the “Preparatory Activities” part of the project was spent on trying to understand the SGD Framework and what it can be useful for. This consumed 16.5 man hours in the initial stage of the project in week 1. Out of the three developers interviewed, two developers said the getting started to guide provided was sufficient for basic functional understanding of the framework, but not for advanced functionality. The same two developers at Swiesh.com thought that the introductory paper should have provided an example of using the framework in test use cases. Seventy five percent of these developers asked for instructions to be provided to test use cases.

7.3 Research Results for Evaluation Criteria-Understandability, based upon Literature Review

We compared the ease of use of the SGD Framework to other frameworks which also have only one role—the individual software programmer. The Personal Software Process (PSP) of Watts Humphrey provides for an intensely structured order of steps the individual programmer should go through and fill in a large number of scripts to the relevant standards, forms, templates, guidelines and measures. The PSP has six different kinds of scripts. In many cases, there is a different version of the script for every relevant level in the PSP. Due to this extensive methodology the possible drawbacks of PSP as compared with the SGD Framework are as follows:

- Some people are not receptive to the detailed data recording
- The longevity of the PSP requires discipline. Several studies, including (Webb, 1999), have noted that engineers stop using the PSP over time unless they are managed.

Based on the literature review it is understood that the PSP of Watts Humphrey is a complicated step of levels which require extensive training for the individual programmer. The programmer will also need to be dedicated to the PSP’s data-collection methods and needs a lot of reading & training before he/she can proceed to adopt the PSP. The PIPSI framework is similar to the PSP in this regard too. PIPSI collects extensive metrics and forms to be filled in by the individual programmer and the programmer will need to first review the PIPSI methodology thoroughly before he/she can understand it.

The literature review of other frameworks too suggests that the SGD Framework is easier to be understood and can be followed without much training. This is a big advantage of the SGD Framework to the software developer.
8 Research Results for Evaluation Criteria - Time Spent

We researched the implementation of the SGD Framework in an actual company setting and if it led to savings in time of development. The SGD Framework was deployed in a 7 week long project and using Toggl timesheets, data was collected on time spent by three developers in performing tasks. In section 8.1 presents the background of this real world deployment in a project setting. Section 8.2 presents the responses gathered by the survey questionnaire as to time taken. Section 8.3 presents the results of the action research study carried out regarding this evaluatory criterion of time spent on activities. This is the quantitative data collected as part of the application of SGD Framework in a real world project. Section 8.4 will present the results of time spent as compared against the other process models in literature review.

In a software project time is as important as in any other project. There are deadlines that must be met by individual developers. Hence we evaluate time as a criterion to see how useful and time consuming the SGD Framework is in a real world development project, and if it can lead to savings in time spent on activities for the individual developer.

8.1 Action Research Case Study

The author of this thesis was working at a company called Swiesh QLurn IT Solutions Inc., as a software developer for their backend operations. Since the beginning of a software development project with Vizag Steel Plant (www.vizagsteel.com) commenced, there was an opportunity to deploy the SGD Framework in actual industry operation and study how it performs in a real-world setting. There were two other back end developers working along with me on a smalltime project called the Glossary Management Tool (GMT) that had a specific deadline (May 18th) to be delivered to VizagSteel. The assignment was to work on their server functionality. This was a seven week long assignment and the tasks included both adding functionality and improving the functionality already existing. The research was conducted at a company office developing the application for the .Net platform. This software project had a team of five members, with three software developers, that all work with their part of the project. The focus of the research was on the developer role in a software project, and how much time was committed to various activities mentioned in the SGD Framework.

The research method applied was action research; the structure of the action research process [21] was followed where one of the software developer’s was the subject of study. The research was of quantitative type, meaning that numerical data were collected and analyzed using mathematically based methods [22]. In this case it was time spent on activities in a software project that were collected and later analyzed. The research had an inductive reasoning, starting with observations and ending with theories. Data collection was conducted via action research and literature study. The software process model used was the SGD Framework although certain amounts of related methodologies might also have crept in.
The research consisted of seven iterations, all a week long. During these iterations there were three main assignments that were conducted. The first one was to update deprecated code on the server. The second one was to develop support for notifications on the server. The last one was never finished but it was about adding some API features to the server. The software project with VizagSteel was still ongoing when the research was over.

8.2 Survey Questionnaire Results for Time Spent on Activities

Sixty-four percent of the questionnaire respondents felt that the SGD would save time in development. Three of the respondents asked if any time metrics would be incorporated into Evaluative or Self-Assessment Activities which would help the developer improve their time spent on a project. 24% said that the SGD may actually not lead to savings in time. These respondents favored the purely Scrum based approach as time saving.

The three developers working on the project were asked to fill in the questionnaire giving their view about time management while using the SGD Framework. Two out of the three developers were of the view that other software process methodologies were better than the SGD Framework at managing and saving time in their daily work. They thought that the Scrum methodology would save more time than the SGD framework that we used in this assignment to build the Glossary Management Tool. We thought that less time would be spent on coding if we had used other methodologies (coding activities in SGD Framework are quite limited and could be defined better). We, together, thought that the SGD Framework had all the activities that came up during the project, in fact we did not use a lot of the activities and nineteen activities were found redundant to our work.

Table 4: Questionnaire Results for the Time Spent on SGD Activities (40 responses to this criterion)

| Time for Development                                                                 | Yes/No, majority of respondents ...
|--------------------------------------------------------------------------------------|--------------------------------------
| To what extent can the SGD Framework lead to savings in development time              | Yes                                  |
| Will the implementation of the SGD Framework lead to time savings?                    | Yes                                  |
| Will using the SGD Framework mean more efficient use of time resources for the developer | Yes                                  |
| Does the SGD Framework allow the developer more time to manage his/her workflow      | Yes                                  |
| Are time issues taken into consideration by this Framework?                          | No                                   |
8.3 Action Research Results to Time spent on Activities

On a daily basis a developer working with the action research project at Swiesh took notes to keep track of the activities conducted and time consumed on them. The tool used for this was Google Sheets, using this tool it was possible to keep track of what has been done and how long time it took [23]. At the end of the work day, the data collected were translated into SGD activities and the time consumed were inserted into a time report sheet, see Table 5. The entire action research phase of the thesis involved only the team of three developers working at Swiesh.com.

Table 5 shows the result of the quantitative action research conducted during the seven week project at Swiesh. It shows the total number of hours spent on each activity in the SGD Framework’s process model as collected by using the timesheet. As can be seen time spent was quite spread out.

Table 5: Sum of time spent on activities while using SGD Framework by team of three developers

<table>
<thead>
<tr>
<th>ACTIVITY (Name and ID)</th>
<th>Total Effort (in hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preliminary Activities</strong></td>
<td></td>
</tr>
<tr>
<td>PR-1: Review and agree on the overall or part of the project plan</td>
<td>1.5</td>
</tr>
<tr>
<td>PR-2: Revise and ensure that the technology to be used is tested and understood</td>
<td>5</td>
</tr>
<tr>
<td>PR-3: Revise and understand any appropriate internal (organizational) &amp; external standards</td>
<td></td>
</tr>
<tr>
<td>PR-4: Learn/Relearn the organizational implementation and unit (developer) way of working</td>
<td>3.5</td>
</tr>
<tr>
<td>PR-5: Review and revise your personal implementation and unit (developer) way of working</td>
<td>2</td>
</tr>
<tr>
<td><strong>Planning Activities of a Developer Role</strong></td>
<td>41</td>
</tr>
<tr>
<td>PL-1: Review the requirement(s) for the unit(s) to be developed</td>
<td>8.75</td>
</tr>
<tr>
<td>PL-2: Prepare (make) and/or review the design specification(s) for the unit(s) to be developed</td>
<td>5.75</td>
</tr>
<tr>
<td>PL-3: Resolve unclear questions and uncertainties</td>
<td>16</td>
</tr>
<tr>
<td>PL-4: Determine and document your implementation and unit (developer) testing goals</td>
<td>1</td>
</tr>
<tr>
<td>PL-5: Determine your implementation and unit (developer) testing strategy</td>
<td>3.5</td>
</tr>
<tr>
<td>PL-6: Determine appropriate implementation and testing practices</td>
<td>2</td>
</tr>
<tr>
<td>PL-7: Identify standards to be used for meeting your goals</td>
<td>0</td>
</tr>
<tr>
<td>PL-8: Set your own personal deadlines to be met during your implementation and unit (developer) testing work</td>
<td>1</td>
</tr>
<tr>
<td>PL-9: Estimate effort and resources required for carrying out your work</td>
<td>0</td>
</tr>
<tr>
<td>PL-10: Schedule your work</td>
<td>0</td>
</tr>
<tr>
<td>PL-11: Review your implementation and unit (developer) testing plan to ensure it is realistic and achievable</td>
<td>2</td>
</tr>
<tr>
<td>PL-12: Identify risks related to your plan</td>
<td>0</td>
</tr>
<tr>
<td>PL-13: Plan for managing any identified risks</td>
<td>1</td>
</tr>
</tbody>
</table>
### Preparatory Activities of a Developer Role

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1: Prepare (make) and/or review your low-level design(s) of the code to be written or changed</td>
<td>9.75</td>
</tr>
<tr>
<td>P-2: Prepare (make) an impact analysis of your low-level design(s)</td>
<td>0</td>
</tr>
<tr>
<td>P-3: Determine the type of unit (developer) test cases and their order</td>
<td>0.75</td>
</tr>
<tr>
<td>P-4: Create and/or revise your unit (developer) test case base</td>
<td>1</td>
</tr>
<tr>
<td>P-5: Revise the existing unit (developer) regression test base, if relevant</td>
<td>0</td>
</tr>
<tr>
<td>P-6: Create or modify stubs and drivers, if required</td>
<td>1</td>
</tr>
<tr>
<td>P-7: Prepare your unit (developer) testing environment and check if it is appropriate to your work</td>
<td>0.5</td>
</tr>
</tbody>
</table>

### Coding Activities of a Developer Role

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-1: Write/rewrite your code</td>
<td>62</td>
</tr>
<tr>
<td>C-2: Compile/Recompile your code as required</td>
<td>0.25</td>
</tr>
<tr>
<td>C-3: Make notes on your compilation errors if required</td>
<td>0</td>
</tr>
<tr>
<td>C-4: Make notes on your defects</td>
<td>1</td>
</tr>
</tbody>
</table>

### Unit Testing Activities of a Developer Role

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-1: Check whether the unit (developer) test case base meets the given requirements and design</td>
<td>3.75</td>
</tr>
<tr>
<td>T-2: Check whether the unit (developer) regression test base meets the given requirements and design</td>
<td>0</td>
</tr>
<tr>
<td>T-3: Remedy requirements problems in your unit (developer) regression and/or test cases base, if any</td>
<td>0</td>
</tr>
<tr>
<td>T-4: Perform dynamic testing by executing code</td>
<td>6.75</td>
</tr>
<tr>
<td>T-5: Perform static (human) testing by reviewing your code</td>
<td>1</td>
</tr>
<tr>
<td>T-6: Record/write down test results</td>
<td>0</td>
</tr>
</tbody>
</table>

### Evaluative Activities of a Developer Role

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-1: Analyze your unit (developer) testing results</td>
<td>2.5</td>
</tr>
<tr>
<td>E-2: Depending upon the unit (developer) testing results, determine your next step(s)</td>
<td>1</td>
</tr>
</tbody>
</table>

### Debugging Activities of a Developer Role

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-1: Identify the source of (an) error(s)</td>
<td>9.75</td>
</tr>
<tr>
<td>D-2: Determine solution(s) for eliminating the sources of error(s)</td>
<td>8.5</td>
</tr>
</tbody>
</table>

### Self-Assessment Activities (Document aside your self-assessment results)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A-1: Assess your own development work</td>
<td>4</td>
</tr>
<tr>
<td>A-2: Identify causes of your mistakes</td>
<td>2</td>
</tr>
<tr>
<td>A-3: Identify improvement areas in your own way of working</td>
<td>2</td>
</tr>
</tbody>
</table>

**Delivery and Signoff Activities of a Developer Role**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S-1: Check that your code fulfills the commitment(s) stated in the Service Level Agreement</td>
<td>0</td>
</tr>
<tr>
<td>S-2: Deliver your code</td>
<td>0,5</td>
</tr>
<tr>
<td>S-3: Sign-off your personal Service Level Agreement</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5 shows that the majority of the time was spent on writing and rewriting code (63.5 hours out of the total 170.5 man hours). Compiling and recompiling this written code did not consume much time; it took just 0.25 hours of total time that was because it was performed instantaneously. The ‘Preliminary Activities’ part took 11.5 hours of time spent of which the activity of testing and ensuring the technology to be used is understood took 5 hours. PR-4 relearning organizational implementation and unit (developer) way of working took 3.5 hours of time. PR-5: Review and revise your personal implementation and unit (developer) way of working took two hours of time.

In this section is the sum of time spent on each individual activity within the scope of the SGD project presented. The first action research results can be seen in Table 5 showing the sum of time spent on each individual activity. As can be seen, the time spent on each individual activity is very spread out. Some activities required no time resources at all while others required much. Writing code was the activity that most time resources was spent on, with 63.25 hours spent of the total 170.5 hours for the entire project. That is about 40% of the total time spent just on coding.

In the rest of the SGD Framework activities that deserve note, the Preliminary activities were most time spent on revising and ensuring that the technology used was tested and understood. Resolving unclear questions and uncertainties was the Planning Activities most time was spent on. The Preparatory Activities that most time was spent on was preparing and reviewing the low-level designs of the code that was written or changed. All this data is presented in Table 5.

The ‘Planning Activities’ part took up 41 man hours of the total 170.5 man hours put into the project at Swiesh.com. PL-1 Review the requirement(s) for the unit(s) to be developed took the majority of the time spent in Planning Activities with 8.75 hours. PL-2 Prepare (make) and/or review the design specification(s) for the unit(s) to be developed took 5.75 hours. PL-3 resolving uncertainties and unclear questions took up 16 hours of the Planning Activities cycle. PL-4: Determine and document your implementation and unit (developer) testing goals took 1 hour. PL-5: Determine your implementation and unit (developer) testing strategy took 3.5 hours of time. PL-6: Determine appropriate implementation and testing practices took 2 hours. PL-8: Set your own personal deadlines to be met during your implementation and unit (developer) testing work took 1 hour. PL-11: Review your implementation and unit (developer) testing plan to ensure it is realistic and achievable took 2 hours.
and the Planning for managing any identified risks took one hour. The activities which were not performed at all during the project work are all given a value of 0. PL-13 Planning for identifiable risks took 1 hour of time.

The ‘Preparatory Activities’ part of the Pre-Work category took up 13 man hours of the total of 170.5 man hours put into the project. P-1: Preparing (make) and reviewing your low-level design(s) of the code to be written or changed took up the majority of time with 9.75 hours. P-3: Determine the type of unit (developer) test cases and their order took 0.75 hours. P-4 and P-6 each took 1 hour of time. The other activities like P-5 were not used.

For Work category Coding Activities of the Developer was by far the part on which maximum amount of time was expended in the project, consuming 63.25 man hours of the total of 170.5 man hours. Activity C-1: Writing and rewriting the source code consumed 62 man hours in itself. The remaining activities did not take up so much time with C-2: Compile/Recompile your code as required taking 0.25 man hours. C-4: Making notes on defects in source code took up 1 hour.

In Work category, ‘Unit testing activities’ took up 11.5 hours. In these activities T-1: Check whether the unit (developer) test case base meets the given requirements and design consumed 3.75 hours. T-4: Perform dynamic testing by executing code took majority of time with 6.75 hours. T-5: Review of code took 1 hour. The other activities under this Unit testing were not performed during the project.

In Work category ‘Evaluative Activities of the Developer’ consumed a total of 3.5 hours. E-1: Analyze your unit (developer) testing results took up 2.5 hours and E-2: Determining the next step took up 1 hour. ‘Debugging Activities of the Developer’ took a total of 18.25 hours of the entire project. D-1: Identify the source of error(s) took 9.75 hours and D-2: Determining solution(s) to eliminate error(s) took up 8.5 hours.

‘Self Assessment Activities’—document aside your self-assessment results took up a total of 8 man hours. A-1: Assess your own development work consumed 4 hours, and A-2: Identify causes of your mistakes took 2 hours. A-3: Identify improvement areas in your own way of working took two hours.

In Post-Work category, ‘Delivery and Sign-Off Activities’ took 0.5 hours as the source code was delivered immediately to the company. The other activities were not performed so a value of zero is assigned to them.

8.4 Research Results to Time Spent on Activities based upon Literature Review

The evaluatory criterion of time spent on SGD activities was benchmarked against the Personal Software Process (PSP) and Process for Improving Programming Skills in Industry (PIPSI) frameworks which were studied in the literature review of chapter 2. The SGD Framework does not collect any data or metrics upon the time spent on programming activities in a project. There are simply no tools used to collect such data in the SGD Framework. But PSP does have a time recording form which records time metrics.
In PSP, forms such as shown in Figure 14 are used for thorough, complete data collection. Time recording log—an important PSP activity is recording how much time is spent on development activities. This data is recorded in a form like the one shown in Figure 14 which was also designed for students learning PSP. When a developer starts to work, he records the date, the start time, and the development phase. If the developer is interrupted during work, the number of elapsed minutes of the interruption is recorded. When activity is completed, the stop time and any comments are noted. Time-Recording log—a summary form that is used for the summarization, analysis, and utilization of the time data that has been entered, as shown in Figure 14. Often, the time and data logging and the completion of the project plan form can be automated via available PSP tools [51].

The Process for Improving Programming Skills in Industry (PIPSI) framework also uses a similar time-recording log form to collect how much time is spent on development activities. In PIPSI also any time spent on interrupted work, unintentional time spent, or on break is recorded.

We feel that data collection about time spent on SGD activities has been left out of the SGD Framework which could help the developer to achieve optimal and efficient use of time resources while working. This is especially important since time is one of the most important factors in developing software and time schedules/deadlines have to be met while working in the real world. Maintaining a time log like the PSP one shown in Figure 14 would be useful to the individual developer optimize their work efforts. This was also the reason we carried out the action research about time spent on SGD activities in this thesis, since time is such an important facet in any developer’s work. This will prove useful to the developer.
9 Analysis and Discussion

This chapter discusses and argues about the research results of the study by analyzing interview results, suggests improvements and by arguing for its guidelines with the support of the research results so far published. Chapter 9.1 analyzes and discusses the research structure while section 9.2 analyzes and discusses SGD framework’s guidelines. The analysis and discussion follows the three evaluation criteria as defined earlier in Chapter 5.

9.1 Thesis Structure

A discussion of the most important points of the thesis research and the action case study are presented in this section. Overall the majority of work expended in this thesis lay in studying the newly proposed SGD Framework and researching any closely related frameworks in existing literature. This led to some in depth studying of the area of software processes, and the models of PSP, and TSP of Watts Humphrey, and PIPSI emerged as a result. We then compared these different software process models and proposed a new revised structure of the SGD Framework based upon the literature review. Subsequently the thesis consisted of carrying out a qualitative survey based research of software professionals. The questionnaires of increasing difficulty were devised and distributed to qualified developers of whom 16 responded with completely filled questionnaires. An action research case study was also carried out simultaneously at my company Swiesh.com to get information about how long the SGD activities took. Thus we had three sources of input information to evaluate the SGD Framework—the survey, the action research at Swiesh, and the literature review. The SGD Framework was then evaluated as against the three evaluation criteria according to the model shown in chapter 5 (“Completeness”, “Understandability”, and “Time Taken”).

9.2 Completeness of the SGD Framework

The results we obtained from the survey questionnaire mailed out to recipients unanimously said that the SGD Framework was not complete. They recommended adding in a few other activities which they said will help make the Framework complete and more secure. As shown in the table below, more than 85% of interviewees wanted us to add in activities to make the code more secure, more stable and easier to insert into already existing systems. The most important point to emerge from the surveys was the need to address maintenance. This aspect of the developer’s work which is fairly common to all developers was not presented in the SGD Framework. Development work does not finish with delivery of code. It needs developer support and activities during integration, system testing and performance testing etc. As Sateesh pointed out it’s not unusual for a company to spend as much as 60 to 70% of all resources on software maintenance. This was the major finding of the survey questionnaire.
Table 6. Results to Survey Questionnaire: Completeness of SGD Framework (53 responses collected)

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Yes/No, majority of responses obtained...</th>
</tr>
</thead>
<tbody>
<tr>
<td>To what extent does the SGD Framework cover all tasks or activities that can come up during a development project?</td>
<td>Not all extent, some activities are missing. 85% said no.</td>
</tr>
<tr>
<td>Is the SGD Framework comprehensive and complete in listing out all developer activities?</td>
<td>85% said No. Some activities should be added.</td>
</tr>
<tr>
<td>Are any activities left out in its categories?</td>
<td>Yes</td>
</tr>
<tr>
<td>Can you list anything that you want to see included in the final version of the Framework?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Preliminary Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Planning Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Preparatory Activities category?</td>
<td>Yes</td>
</tr>
<tr>
<td>Are any activities missing in the Coding Activities category?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

As Table 6 shows respondents reported there were missing activities in every sub-category of the framework. Over 85% of respondents to the survey questionnaire said we should add in more activities to make the SGD Framework complete and that no activities were redundant. The same percentage reported the SGD Framework errs by exclusion rather than by inclusion of activities. They wanted us to look at other models and process frameworks to get an idea of what steps to include in the new revised SGD Framework.

As part of the survey questionnaire the same majority of respondents expressed ambiguity about the SGD Framework and whom it can help the most—the individual developer or the team of developers. They could not understand if it is aimed at the engineer or the team. Sateesh Neeladri thought a missing activity in Planning Activities is “Prototype which is never specified here. Moreover, once Requirements are drafted, then high level design followed by low level design should be implemented.” Tom Russ said: “it might be wise to explicitly state that your test cases include both abuse cases and use cases. It leaps out at me that the security of your code is not mentioned anywhere.”
However, as per the action research part of the study conducted at the company during the VizagSteel project the result is that the SGD Framework is complete and no further activities need to be added to it. A seven week long iterative project was commissioned to be performed for building a Glossary Management Tool for VizagSteel at a company. The project was performed using the SGD as its toolbase, with three developers assigned to the project. This was the action research case study part of the thesis which collected all the quantitative data on time spent and what activities were conducted in a real world setting.

The result that emerged about the Completeness criteria of the SGD Framework is that the Framework is indeed complete. In fact 19 activities were felt to be redundant and not used while at the project site in those seven weeks. These 19 activities are listed in the Framework but were not performed at all during the course of the project. Moreover every activity that was performed during the work at the company was already found to be included within the activities of the SGD Framework. Hence the conclusion from the action research case study was that the SGD Framework is complete and robust.

The Literature Review part of the research was used to evaluate the Completeness of SGD Framework as well. Activities and features of the SGD Framework were benchmarked against Personal Software Process (PSP) of Watts Humphrey and against the Process for Improving Programming Skills in Industry (PIPSI). These two process models were discovered during the course of the literature study and are described in detail in chapter 2. These two specific process models were chosen since they are oriented towards the individual software developer (similar to the SGD). It emerged that there are three strong points in PSP and PIPSI which could well be adapted to make the SGD more fruitful.

PSP has several strong tenets as follows:

1. The first is that the longer a software defect remains in a product, the more costly it is to detect and remove it. Therefore, thorough design reviews and code reviews are performed at every level for efficient defect removal.
2. The second tenet is that defect prevention is more efficient than defect removal. Careful designs are developed and data is collected to give additional input on where the programmer should adjust their own personal software process to prevent future defects.
3. Time recording log—another important PSP activity is recording how much time is spent on development activities. When a developer starts to work, he records the date, the start time, and the development phase. If the developer is interrupted during work, the number of elapsed minutes of the interruption is recorded. When activity is completed, the stop time and any comments are noted.

We think that including thorough code reviews and design reviews at every step of the Pre-Work and Work process in the SGD Framework could make it better. The PSP has been proven to be a high quality process model for this very reason. PSP aims at defect prevention and takes a whole lot of care to make sure defects are noticed early in the development process by the individual programmers themselves. SGD Framework would also benefit by incorporating a time recording log for the same reasons which are shown in PSP’s strong tenets above.
Table 7: Results of Survey Questionnaire to Understandability of SGD Framework (42 respondents)

<table>
<thead>
<tr>
<th>Understandability</th>
<th>Yes/No, majority of respondents…</th>
</tr>
</thead>
<tbody>
<tr>
<td>How straightforward is it to understand:</td>
<td></td>
</tr>
<tr>
<td>• What the SGD Framework does and its purpose?</td>
<td>Not straightforward, better introduction required</td>
</tr>
<tr>
<td>• The intended market and users of the SGD Framework?</td>
<td></td>
</tr>
<tr>
<td>On a scale from 1 to 10, how do you rate the clarity of the SGD Framework? List a digit from 1 to 10 in right column.</td>
<td>7 (average score for Understandability)</td>
</tr>
<tr>
<td>Score 1 implies needs extensive documentation</td>
<td></td>
</tr>
<tr>
<td>Score 10 implies very clear and intuitive to understand.</td>
<td></td>
</tr>
<tr>
<td>High level description of what/who the SGD Framework is for is available.</td>
<td>No</td>
</tr>
<tr>
<td>High level description of what the SGD Framework can do is available.</td>
<td>Yes</td>
</tr>
<tr>
<td>High level description of how the SGD Framework performs is available.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

9.3 Understandability of the SGD Framework

The results we obtained from the survey questionnaire mailed out to recipients stated that most recipients did not understand what the SGD Framework was aimed towards and what it can accomplish. More than 75% of the respondents said that it was hard to understand what the SGD Framework does and how it can be utilized. They said they required better introduction and documentation to help them understand it, and that the supplied paper was not sufficient. More than 75% of respondents wanted us to come up with a clearer and more precise description of the SGD model and how its activities could be useful to software developers. Only one respondent gave it a rating of 10 on the ease-of-use scale. Six respondents gave it a rating of 8 on the scale and five interviewees gave it a rating of 7 on the scale.

Fully 77% of interviewees did not understand that the framework only lists types of activities that you may use in your development work in the order that suits you. It does not show any order that you must
follow! Fully 80% of those surveyed did not understand that the framework categories do not correspond to any predefined development phases.

In the action research part of study, I observed similar trends as in the survey results regarding Understandability of the SGD Framework. In the real-world action research we had a team of developers working with the SGD Framework on a real world project to develop and test software as part of a project assignment. There were three software developers (myself included) who first learned about the SGD Framework from the introductory paper provided and used it to build a GMT tool. In the first week during Preparatory part of the project, learning about the SGD Framework from the introductory paper took up 16.5 man hours of time. This was an enormous amount of time as compared to time spent on other activities within the first stage. A majority of time in the “Preparatory Activities” part of the project was spent on trying to understand the SGD Framework and what it can be useful for. This consumed 16.5 man hours in the initial stage of the project in week 1. Out of the three developers interviewed, two developers said the getting started to guide provided was sufficient for basic functional understanding of the framework, but not for advanced functionality.

The same two developers at Swiesh.com thought that the introductory paper should have provided an example of using the framework in test use cases. Seventy five percent of these developers asked for instructions to be provided to test use cases. Hence “training” is an essential component and should be added to the Framework as a Preliminary Activity. The Framework should evolve proper introduction and documentation to help explain it to the average software developer, otherwise it would remain unused. Once the Framework is complete, perhaps, training tutorials or online help could be offered on the framework’s website, to help train future developers. The Software Engineering Institute (SEI) employs such measures with PSP and TSP.

Research Results for Evaluation Criteria-Understandability, based upon Literature Review: We compared the ease of use and clarity of the SGD Framework against other frameworks which also focus upon only one role—that of the individual programmer. Here, the result obtained was that the SGD Framework is much easier and understandable by the average worker than existing frameworks like the PSP and the PIPSI. In part, this is because PSP and PIPSI maintain the strict sequential use of templates, forms and metrics to discipline their users. There is a huge reliance on over six types of forms in the PSP and many standards, procedures and templates etc have to be maintained in detail. Unduly this makes the process of software development laborious and many developers stop using the PSP over the course of time.

In many cases, there is a different version of the script for every relevant level in the PSP. Due to this extensive methodology the possible drawbacks of PSP as compared with the SGD Framework are as follows:

- Some people are not receptive to the detailed data recording
- The longevity of the PSP requires discipline. Several studies, including (Webb, 1999), have noted that engineers stop using the PSP over time unless they are managed.

The programmer will also need to be dedicated to the PSP’s data-collection methods and needs a lot of reading & training before he/she can proceed to adopt the PSP. The PIPSI framework is similar to the
PSP in this regard too; it relies on detailed collection of data and metrics. The literature review of other frameworks too suggests that the SGD Framework is easier to be understood and can be followed without much training. This is a big advantage of the SGD Framework to the software developer.

9.4 Time Taken by Activities within the SGD Framework

The time taken by various activities as a whole under the SGD Framework was researched primarily by the action research undertaken at the company. This was the 7 week long project where the activities of three developers were logged into timesheets and data collected as to time spent on various activities by the three developers. The evaluation criterion of time spent was also studied by the survey questionnaire and the literature review. Additional questions were added to the initial questionnaire that asked the respondents regarding time usage while following the SGD Framework and these can be seen in the questionnaire in Appendix A. Time usage was also encountered while doing the literature review under the PSP model which collects precise data about time metrics of the developer in going about his tasks. Hence we took a good look at time and how many hours were consumed doing which activities during this research study as well. We will begin by reviewing the responses to time usage gathered from the survey questionnaire followed by results we obtained to time spent from the action research carried out in the project. This will be followed by the results we obtained from literature review.

Survey Questionnaire results to time usage in SGD Framework: Sixty-four percent of the questionnaire respondents felt that the SGD would save time in development. Three of the respondents asked if any time metrics would be incorporated into Evaluative or Self-Assessment Activities of the SGD Framework which would help the developer improve their time spent on a project. 24% said that the SGD may actually not lead to savings in time. These respondents favored the purely Scrum based approach as time saving. The people involved in the action research were also surveyed regarding their views about time usage under the SGD. Most of the team-members were of the view that the SGD would not significantly lead to any savings in time as compared with the Scrum approach.

The first action research results can be seen in Table 5 showing the sum of time spent on each individual activity. As can be seen, the time spent on each individual activity is very spread out. Some activities required no time resources at all while others required much. Writing code was the activity that most time resources was spent on, with 63.25 hours spent of the total 170.5 hours for the entire project. That is about 40% of the total time spent just on coding.

In the rest of the SGD Framework activities that deserve note, the Preliminary activities were most time spent on revising and ensuring that the technology used was tested and understood. Resolving unclear questions and uncertainties was the Planning Activities most time was spent on. For Work category Coding Activities of the Developer was by far the part on which maximum amount of time was expended in the project, consuming 63.25 man hours of the total of 170.5 man hours. Activity C-1: Writing and rewriting the source code consumed 62 man hours in itself.
As the literature review about like models showed we feel that data collection about time spent on SGD activities has been left out of the SGD Framework which could help the developer to achieve optimal and efficient use of time resources while working. This is especially important since time is one of the most important factors in developing software and time schedules/deadlines have to be met while working in the real world. Maintaining a time log like the PSP one shown in Figure 14 would be useful to the individual developer optimize their work efforts.

The evaluation criterion of time spent on SGD activities was benchmarked against the Personal Software Process (PSP) and Process for Improving Programming Skills in Industry (PIPSI) frameworks which were studied in the literature review of chapter 2. The SGD Framework does not collect any data or metrics upon the time spent on programming activities in a project. There were no tools used to collect such data in the SGD Framework. But PSP does have a time recording form which records time metrics, and PSP also has a time recording log sheet. We feel that having a time logging sheet. When a developer starts to work, he records the date, the start time, and the development phase. If the developer is interrupted during work, the number of elapsed minutes of the interruption is recorded. When activity is completed, the stop time and any comments are noted in the PSP. We feel that the same type of time recording log will be beneficial to the individual developer maintain some discipline in his work, and so should be used even in SGD Framework.

The need to address maintenance of existing code or legacy code was one of the prime results to emerge as a result of the survey questionnaire. This along with issues of integration of code/systems, re-engineering, supportability and design reviews were the issues to emerge from the survey carried out about the Completeness of the SGD Framework. Mostly re-engineering at the software level is performed by software engineers as well. It encompasses inventory analysis, document restructuring, reverse engineering, and forward engineering to create versions of existing code that exhibit higher quality and better maintainability. Also, supportability is one of many quality factors which should be considered during the analysis and design actions which are part of the software process. The SGD Framework should ideally involve it as part of the Preparatory Activities (Work) and consider it as the design evolves and coding commences. Software maintenance and support are ongoing activities that occur throughout the lifecycle of an application.

There are in general three different types of software maintenance:

1. Maintenance to repair software faults
2. Maintenance to adapt the software to a different operating environment
3. Maintenance to add to or modify the system’s functionality [20, p. 517].

I was not able to find out what specific types of maintenance activities can fit into the SGD Framework. However I think that they would be required as a separate process category in the SGD Framework; perhaps after the Post-Work Activities there could be a whole new process category called “Maintenance and Adaptability”.

71
Similarly it was found that the majority of time taken was by coding and debugging activities—over 40% of the time at the company project was consumed in coding and rewriting code. A lot of respondents wrote that there should be more activities to help code in a more efficient manner and test for errors while writing code itself. Two new activities that can be added to Coding Activities are Design Review and Code Review. The most important security practice is Design Review which could be included either in Coding Activities or Evaluative Activities. Code Review should be incorporated as an important activity in the same way, for writing secure code.

As a few interviewees noted code review was missing as an activity, which is mandatory at the development phase of coding. And that “Developer Release Notes (DRN was never specified)” in the Coding Activities.

Under Evaluative Activities perhaps “system testing” could be included as an activity. This should help make sure that the software being made would integrate well into the larger system of which it is a part. System and acceptance testing activities are missing. These are context oriented and are needed both in Agile & Traditional development practices, so they could be put under the Evaluative Activity category.

Under Self-Assessment Activities, the Framework could employ metrics style of gathering data about time estimation for the various tasks. This can help the developer make estimates on time allocation and also keep track of how much time they spend on each of their activities. The PSP and PIPSI models (chapter 2.4) both keep track of time resources which are important in a project.

Apart from the activity categories, there is a real need to put up a webpage dedicated to the SGD Framework project and put general instructions/documentation there. Most respondents complained that they could not find any information about the SGD Framework and that the material I supplied them with was insufficient. In the future it would be valuable to trademark the name, generate a domain space and keep all up-to-date instructions regarding the Framework online.

9.5 Validity Threats

For the survey we considered the four perspectives of validity and threats as presented in Wohlin et al. [22], i.e. Internal Validity, External Validity, Construct Validity and Conclusion Validity.

**Internal Validity:** Designing a readable and understandable survey questionnaire mitigates the threats related to internal validity enhanced by a high completion rate of the respondents (61%). The survey respondents were selected by using various professional networks like LinkedIn, Academia and also developer blogs. This indicates that researchers have no influence on the selection of the subjects for the survey. Furthermore, the sampling error associated with the selection of subjects is formally depicted through the use of statistical hypothesis testing.

**External Validity:** The survey-makers’ control and influence on the respondents answering the questionnaire is limited as the survey was conducted over the Web. Also the samples selected for the study are from very diverse project settings, e.g. regarding the agile and traditional methods, application type, project sizes, project duration, companies and distribution of the development effort. In addition,
participants have fairly heterogeneous experience in process models, agile and waterfall development. Thus, heterogeneous experience of the participants, the diverse distribution of the respondents, and a sufficiently large sample size, show the possibility to generalize the survey findings beyond the selected sample.

**Construct Validity:** The developer activities in our research are identified and measured through the survey using close-ended questions. Inadequate pre-operational explication of construct threat was avoided by designing the survey instruments in a way to clearly represent the object of the research. This was done by defining the research aim, procedures, objectives, the evaluation criteria such as cost and benefit in the introductory section of the survey. Moreover, the definitions of SGD Framework activities are also linked to the survey questions. Hence Mono-operation bias is avoided. The online survey responses are completely anonymous so the evaluation apprehension is mitigated. Another influential risk is the background of the respondents (e.g. experience). However, due to the sufficiently large sample size and the respondents’ level of experience in agile development and writing code, we consider that the risk associated with the background of the subjects as limited.

**Conclusion Validity:** Due to the sufficiently large sample size we consider that the overall influence of the risk related to violation of assumption of statistical tests is limited. The research conclusion is purely based upon the independent analysis of the study outcome.
10 Conclusions and Future Work

The individual software developer is a very integral and key part of the entire development process of software. Knowing how he/she can effectively manage his/her activities and resources is the key to productivity and success. The agile software models of the 1990’s had already begun to realize the importance of the individual developer and developer-centric development models. However there was not much research into this area of inventing new individual-centric software process models. The pioneer efforts were those of Watts Humphrey [1] from the Software Engineering Institute at Carnegie-Mellon University and process models like PSP still relied on a traditional, sequential and industrial method of making software. When I began the literature study on process models pertaining to the developer I found almost no studies on this subject. The lack of published material about this matter to guide the developer made us formulate a research question about the proposed SGD Framework and how it can guide the developer in his/her daily work activities. To address the research question, we conducted research using the SGD Framework as the basis of the research. The research consisted in large part of a survey of software professionals working in the industry for over 3 year period and what they thought of the newly proposed SGD Framework. Would they like to use the framework in their daily professional work? What did they think was missing in the framework or would they like to think was redundant in it? We revised our framework, came up with the latest version of it using available literature studies and sent it out to over 50 developers across the world to capture their opinions. And we got a lot of opinions from working professionals in the software industry.

Simultaneously, we also deployed the SGD Framework in an actual real world project conducted at a company Swiesh.com to see how it can perform under real-world situations and deadlines. This was deployed over a 7 week long period. We gathered data about how the various activities listed in the SGD Framework were necessary to come up with a working software product, and how time resources were allocated to these various activities. This quantitative and qualitative research helped us understand the strengths and weaknesses of the SGD Framework and come up with various suggestions for improvement. Thus, our research can contribute towards the goals of this thesis—to help come up with working process models to help the individual developer and lay the ground for future research in this area of software engineering.

The social benefit of this research is the possibility of increased knowledge about tasks and time spent in individual developer work. If the developers can manage their own time and resources well their productivity and efficiency will increase. This also leads to better quality software and social benefits to the community. This can lead to that companies can decrease the price of the developed products, making the consumers wealthier and business more profitable. The standard of living can be improved from both a commercial and consumer perspective [42].

The results suggest that as an individual software developer most of the time was spent on coding, resolving unclear questions and uncertainties, preparing and reviewing the low-level designs of the code to be written and changed, and identifying the source of errors. This took up the majority of time that a developer had available.
The SGD framework was found very useful to conduct this kind of research. It included all necessary activities and provided good guidelines of how to work as a developer. Only a few more activities regarding maintenance, design review, and code review have to be added to the Framework to make it complete and marketable. The SGD Framework combines well with agile methodologies and can easily be adapted for use in Scrum sprint like cyclic development or XP.

As I had only two months to devise and conduct the survey I could not reach a lot of developers who might have been interested in this survey. Yet I still managed to get a lot of feedback from the few that sent in fully complete questionnaires, what is presented in this thesis is but a selection from this large amount of survey information. In the future I hope to be able to expand upon all this data which was collected during the survey. Individual developer-centric models are a very new area of research and there is a lot than can be done in this field. There is genuine need to develop models and research a wider arena of software professionals to get a more accurate picture of what is currently needed in this field. Due to lack of time resources I have only been able to research a little in this area, but still obtained a large amount of feedback. The evaluation criteria used in this thesis can hopefully guide future research. More studies of developers’ work have to be made to get a general overview of how resources are spent by an individual developer, and how we can best assist them with the revised Framework.
11 References


[37] The Agile Alliance Homepage, Available: [www.agilealliance.org/home](http://www.agilealliance.org/home)


[43] Swiesh QLurn IT Solutions, Visakhapatnam, India, Available: [http://www.swiesh.com](http://www.swiesh.com)


[47] Business Truths, "Time is your important resource", Available: [http://www.businesstruths.com/articles/your-most-important-resource-is-your-time/](http://www.businesstruths.com/articles/your-most-important-resource-is-your-time/)


