A Study Of Software Engineering Practices for Micro Teams

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Shweta Deshpande, B.E.

Graduate Program in Computer Science and Engineering

The Ohio State University

2011

Thesis Committee:

Dr. Rajiv Ramnath, Advisor

Dr. Jay Ramanathan

ABSTRACT

For decades, software development has faced big problems – projects running over budget, over time, becoming unmanageable in terms of sheer size, software being inefficient, of low quality, and not fully meeting requirements. As a solution to these problems, various software engineering practices gradually developed and were practised by the developers and project managers. Over the decades, a number of standard methodologies came up and became widely used in the industry. These methodologies (waterfall model, spiral model, agile methodologies, Rational Unified Process, etc) proved to be fairly useful for the teams that used them – both large and small teams.

What is observed, though, is that a large number of software projects – both in large and small enterprises – are in fact done by micro teams. These micro teams typically consist of three or four persons, with just a single or at most two developers, a business analyst and a project manager or surrogate customer. Surprisingly, there are no formal methodologies developed for such micro teams, leaving them to either follow one of the standard methodologies for larger teams or none at all. When micro teams try to apply the standard methodologies to their projects, though, the methodologies often fall short in managing all the issues that are specific to the micro teams because they were essentially developed for large teams in the first place. There is, thus, a need for an exclusive software development framework that will provide guidelines and best practices to such micro teams for developing projects efficiently and successfully.

As a step in that direction, in this thesis we try to address the issue by coming up with a set of practices that are particular to the software development process followed by micro teams with a single developer. We do this by doing a multiple-case case study involving five real world software projects of commercial value that were developed by micro teams with single developers in an academic setting as a part of university-industry collaboration. We focus on the entire development process (from requirements collection to the final implementation) of each project followed by the development teams, and choose a set of steps from the processes that were of special relevance to micro teams.

The thesis makes the following contributions:

1. A definition of a micro team – how it is different from a small/medium/large team
2. It presents a case study of micro team software projects that studies the steps that single developers in micro teams follow while developing software.
3. It presents a set of steps specific to micro teams and single developers, as opposed to being common to a development team of any size.
4. It presents a comparison of these set of steps with the ones followed in existing methodologies.
5. Finally, it presents a set of guidelines that may be used as a basis for developing a separate and complete software development framework for micro teams.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my advisor Professor Rajiv Ramnath. His immense knowledge and critical thinking process have been of great value for me. I thank him for his patience and encouragement. I could not have imagined having a better mentor.

I also wish to express my warm and sincere thanks to Professor Jayashree Ramanathan for her continued support, insights and suggestions that helped to shape my research.

VITA

June 2008..... B.E. Computer Engineering, University of Pune

September 2008 to present ..... Graduate Student, Department of Computer Science and Engineering, The Ohio State University

Field of study:

Major Field: Computer Science and Engineering

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In 1972, Edsger Dijkstra described succinctly the problem that the software development industry was facing: “As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.” [2]

The software crisis of the early 1960’s gave rise to the field of software engineering. Software, delivering the most important aspect of our time – information [1] – had a definite place in the world, since hardware couldn’t be used without software. Yet the development of software was facing some tough problems with the rapid growth of increasingly complex hardware. The software crisis arose out of many reasons:

- Projects running over-budget
- Projects running over-time
- Software produced was inefficient
- Software produced was of low-quality
- Software did not meet requirements
- Projects were unmanageable
- Code difficult to maintain
- Software was never delivered

As a response to this crisis, the first NATO Software Engineering Conference was held in 1969. In it, Dr. Fritz Bauer gave software engineering its first definition: the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. A number of software engineering techniques were introduced: tools, discipline, formal methods, process and professionalism [3].

- Tools: Especially emphasized were tools: Structured programming, object-oriented programming, CASE tools, Ada, documentation, and standards were touted as silver bullets.
- Discipline: Some pundits argued that the software crisis was due to the lack of discipline of programmers.
- Formal methods: Some believed that if formal engineering methodologies would be applied to software development, then production of software would become as predictable an industry as other branches of engineering. They advocated proving all programs correct.
- Process: Many advocated the use of defined processes and methodologies like the Capability Maturity Model.
- Professionalism: This led to work on a code of ethics, licenses, and professionalism.

Thanks to the continuous development and use of such techniques, over the past five decades, software engineering has morphed from being an obscure line of engineering to a widely acknowledged and practised discipline worldwide. This growth has occurred hand in hand with the growth and widespread use of computers and

software itself. Though the debate whether software engineering is a legitimate field of engineering still continues, there is no doubt that the development of the field itself has tremendously improved the quality of software being produced. As a result, many industry observers, like Pressman, had a change of heart about the software crisis. According to him,

"the 'crisis' never seemed to materialize… It is far more accurate to describe the problems we have endured in the software business as a chronic affliction than a crisis. We live with this affliction to this day—in fact, the industry prospers in spite of it."

Dr. Winston Royce had a similar opinion [4]:

"The construction of new software that is both pleasing to the user/buyer and without latent errors is an unexpectedly hard problem. It is perhaps the most difficult problem in engineering today, and has been recognized as such for more than 15 years. It is often referred to as the "software crisis". It has become the longest continuing "crisis" in the engineering world, and it continues unabated."

What everyone seemed to agree on was that there was no silver bullet [5] to any of these problems. The software crisis has been slowly fizzling out, because it is unrealistic to remain in crisis mode for more than 20 years. Software engineers are accepting that the problems of software engineering are truly difficult and only hard work over many decades can solve them [3].

One of the more popular and hotly debated software engineering techniques is the software development process. A software development process is basically a structure imposed on the development of a software product. A number of different software

development processes have been developed, practised and widely accepted over time. At present, a number of methodologies are widely accepted as standard and used:

Traditional methodologies – Waterfall (with different variations), Prototyping, Incremental model, Spiral model and

Agile methodologies – Extreme Programming, Scrum, RUP, etc; this variety of methodologies can be applied to different kinds of projects.

1.1 THE PROBLEM:

Almost all software methodologies developed are for large teams – since software engineering techniques were developed primarily to combat problems in large projects (large scope as well as large number of people). Basically, large projects tend to fail spectacularly as opposed to their smaller counterparts – hence the focus on methodologies dealing with large teams for large projects. But studies of large and small enterprises show that a significant number of software projects are done by micro teams of 3 to 4 – often with just a single developer. There are no software development methodologies that target such micro teams, and using existing methodologies meant for large teams with micro teams leads to gaps in implementation. There is, hence, a need to define a framework that will provide guidelines to micro teams in developing software.

This problem can be further broken down to the following questions:

- What are the steps that single developers in micro teams follow while developing software?

- Of these, which steps are specific to micro teams and single developers, as opposed to being common to a development team of any size?
- How are these steps different from or comparable to steps followed in existing methodologies?
- What can be a set of steps that can be generalized to provide guidelines to any micro team and be a basis for building a development methodology?

Through this thesis we seek to answer these questions.

1.2 SOLUTION APPROACH:

Case Study Research [29]:

The factors that affect which research strategy is chosen to answer a particular research question are:

- The type of research question
- The control an investigator has over actual behavioral events
- The focus on contemporary as opposed to historical phenomena

When the type of question is 'how' or 'when', an investigator has little control over the events and the focus is on contemporary phenomenon within real-life context, *case study research* is the most suitable strategy.

A case study is an empirical inquiry that

- Investigates a contemporary phenomenon within its real-life context, especially when
- The boundaries between phenomenon and context are not clearly evident

Hence to answer these questions, we chose the case study approach.

Our approach involves studying the development processes of micro team projects and identifying the important steps in each that contributed to the efficiency and success of the process. We do a multiple-case case study involving real life software projects of commercial value developed by micro-teams as a part of university-industry collaboration.

CHAPTER 2

RELATED RESEARCH AND BEST PRACTICES

A large and growing body of software development organizations implements process methodologies. A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of writing software. Others apply project management techniques to writing software. Without project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, effective project management appears to be lacking. Several models exist to streamline the development process. Each one has its pros and cons, and it's up to the development team to adopt the most appropriate one for the project. Sometimes a combination of the models may be more suitable. The following two sections (2.1 and 2.2) are referenced from M. Awad's "A Comparison between Agile and Traditional Software Development Methodologies" [10].

2.1 TRADITIONAL DEVELOPMENT PROCESSES:

Traditional methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high level design development and inspection. Due to these heavy

aspects, this methodology became to be known as heavyweight. Heavyweight methodologies are considered to be the traditional way of developing software. These methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing and deployment. Heavyweight methodologies require defining and documenting a stable set of requirements at the beginning of a project. Fowler criticizes that these methodologies are bureaucratic, that there is so much to follow the methodology that the whole pace of development slows down [6]. The traditional methodologies have these similar characteristics.

- Predictive approach
- Comprehensive Documentation
- Process Oriented
- Tool Oriented

A few of the most common traditional methodologies are:

2.1.1 WATERFALL MODEL

Winston Royce in 1970 proposed the waterfall methodology. The waterfall approach emphasizes a structured progression between defined phases. Each phase consists on a definite set of activities and deliverables that must be accomplished before the following phase can begin. The phases are always named differently but the basic idea is that the first phase tries to capture What the system will do, its system and software requirements, the second phase determines How it will be designed. The third stage is where the developers start writing the code, the fourth phase is the Testing of the system and the final phase

is focused on Implementation tasks such as training and heavy documentation. However, in engineering practice, the term waterfall is used as a generic name to all sequential software engineering methodology.

2.1.2 SPIRAL MODEL

Another heavyweight software development model is the spiral model, which combines elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. The spiral model was defined by Barry Boehm, based on experience with various refinements of the waterfall model as applied to large software projects. There are four main phases of the spiral model [7]:

- Objective setting – Specific objectives for the project phase are identified
- Risk assessment and reduction – Key risks are identified, analyzed and information is obtained to reduce these risks
- Development and Validation – An appropriate model is chosen for the next phase of development.
- Planning – The project is reviewed and plans are drawn up for the next round of spiral

2.1.3 UNIFIED PROCESS

All efforts, including modeling, is organized into workflows in the Unified Process (UP) and is performed in an iterative and incremental manner. To

determine the length of the project, UP divides the project into four phases which are discussed below [8]:

- Inception – By the end of this process a business case should have been made; feasibility of the project assessed; and the scope of the design should be set.
- Elaboration – In this phase a basic architecture should have been produced and a plan of construction agreed. Furthermore, a risk analysis takes place and those risks considered to be major should have been addressed.
- Construction – This process produces a beta-release system. A working system should be available and sufficient enough for preliminary testing under realistic conditions.
- Transition – The system is introduced to the stakeholders and intended users. It is crossed when the project team and the stakeholders agree that the objectives agreed in the inception phase have been met and the user is satisfied.

There are approximately 50 work products to be completed in UP. All this documentation and this rigid approach add a lot of complexity to UP. As well, UP predefines roles to the project team making it less flexible.

2.2 AGILE DEVELOPMENT PROCESSES:

The name “agile” came about in 2001, when seventeen process methodologists held a meeting to discuss future trends in software development. They noticed that their

methods had many characteristics in common so they decided to name these processes agile, meaning it is both light and sufficient. In consequence to this meeting, the "Agile Alliance" and its manifesto for agile software development emerged. The agile methods claim to place more emphasis on people, interaction, working software, customer collaboration, and change, rather than on processes, tools, contracts and plans. The following principles of agile methodologies are seen as the main differences between agile and traditional ones:

- People Oriented
- Adaptive
- Conformance to Actual
- Balancing Flexibility and Planning
- Empirical Process
- Decentralized Approach
- Simplicity
- Collaboration
- Small Self-Organizing Teams

A few of the most common agile methodologies are:

2.2.1 EXTREME PROGRAMMING:

Extreme programming (XP) has evolved from the problems caused by the long development cycles of traditional development models [9]. The XP process can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design [10].

With all the above qualities, XP programmers respond to changing environment with much more courage. Further according to Williams [11], XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day. The term 'extreme' comes from taking these commonsense principles and practices to extreme levels.

2.2.2 SCRUM:

Scrum is an iterative, incremental process for developing any product or managing any work. Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment. At the end of every iteration it produces a potential set of functionality. The term 'scrum' originated from a strategy in the game of rugby where it denotes "getting an out-of-play ball back into the game" with teamwork [12].

Scrum does not require or provide any specific software development methods/practices to be used. Instead, it requires certain management practices and tools in different phases of Scrum to avoid the chaos by unpredictability and complexity [13].

Key Scrum practices are

- Product Backlog
- Sprints
- Sprint Planning meeting

- Sprint Backlog
- Daily Scrum

2.2.3 FEATURE DRIVEN DEVELOPMENT:

The FDD approach does not cover the entire software development process but rather focuses on the design and building phases. The first three phases are done at the beginning of the project. The last two phases are the iterative part of the process which supports the agile development with quick adaptations to late changes in requirements and business needs. The FDD approach includes frequent and tangible deliverables, along with accurate monitoring of the progress of the report.

- Develop an Overall Model - A high level walkthrough of the system scope and its context is performed by the domain expert to the team members and chief architect. Documented requirements such as use cases or functional specifications are developed.
- Build a Features List - A categorized list of features to support the requirements is produced
- Plan by Feature - The development team orders the feature sets according to their priority and dependencies and assigned to chief programmers. Furthermore, the classes identified in the first phase are assigned to class owners (individual developers). Also schedule and milestones are set for the feature sets.
- Design by Feature and Build by Feature - Features are selected from the feature set and feature teams needed to develop these features are chosen

by the class owners. The design by feature and build by feature are iterative procedures during which the team produces the sequence diagrams for the assigned features. These diagrams are passed on to the developers who implement the items necessary to support the design for a particular feature. There can be multiple feature teams concurrently designing and building their own set of features. The code developed is then unit tested and inspected. After a successful iteration, the completed features are promoted to the main build.

2.2.4 ADAPTIVE SOFTWARE DEVELOPMENT:

Adaptive Software Development (ASD), developed by James A. Highsmith, offers an agile and adaptive approach to high-speed and high-change software projects. It is not possible to plan successfully in a fast moving and unpredictable business environment. In ASD, the static plan-design life cycle is replaced by a dynamic speculate-collaborate-learn life cycle.

ASD focal point is on three non-linear and overlapping phases:

- Speculate - To define the project mission, make clear the realization about what is unclear.
- Collaborate – Highlights the importance of teamwork for developing high-change systems
- Learn – This phase stresses the need to admit and react to mistakes, and that requirements may well change during development.

2.3 SOFTWARE METHODOLOGIES AND TEAM SIZES:

An important observation that can be made from studying all these methodologies is that all of them were developed for large or small teams. A large team can be defined as one which has more than 25 developers; a medium team has 11 – 25 developers, while a small team has 10 or less number of developers. At the beginning of the software engineering era, the focus was on developing processes for large teams, since most of the problems that characterized the software crisis were observed in large teams. Large software projects failed spectacularly compared to the smaller ones, and their need for a solution was naturally bigger than that of smaller teams. Gradually, as the efficiency and consequently the importance of smaller teams started becoming clearer, a number of methodologies targeted towards them started coming up, known collectively as agile methodologies.

Even then, the problem is not solved. Separate formal and informal studies, as well as general observations and experiences of software developers have proved that software development does not always occur in large, medium or small teams. In fact, a large majority of development that takes place is done by 1 or 2 developers, or what can be termed as a micro team.

2.3.1 DEFINITION OF MICRO TEAM:

Our definition of a micro team as a team has two dimensions – quantitative and qualitative. The quantitative characteristics consist of the number and roles of persons involved, while the qualitative characteristics consist of the effects of the quantitative ones.

2.3.1.1 QUANTITATIVE CHARACTERISTICS:

- Consists of not more than 3 – 4 people actively involved
- At least and only one developer programming full time
- Rest of the team has either of these roles: a business analyst, a project manager, a surrogate customer, a technical advisor

2.3.1.2 QUALITATIVE CHARACTERISTICS:

- Constrained by one developer's knowledge and perspective
- Developer becomes single point of failure
- Multi-person practices don't work
- No peer review possible

A 2005 study published by the software consultancy firm Quantitative Software Management shows that over 50% projects (of the 564 included in the study) are done by teams sized 1.5 – 3 [25].

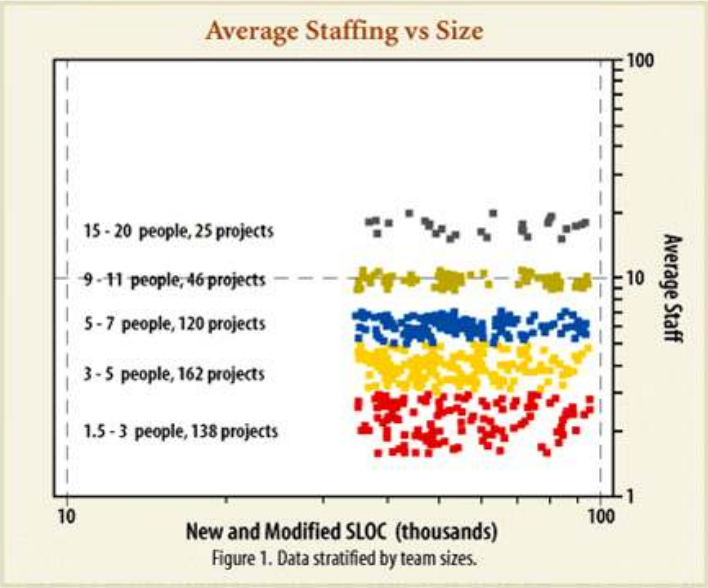

Figure 2.1 QSM Study of Project Sizes

In another 2009 study by Scott Ambler, 34% of 123 were projects with team size 1 – 5 [26].

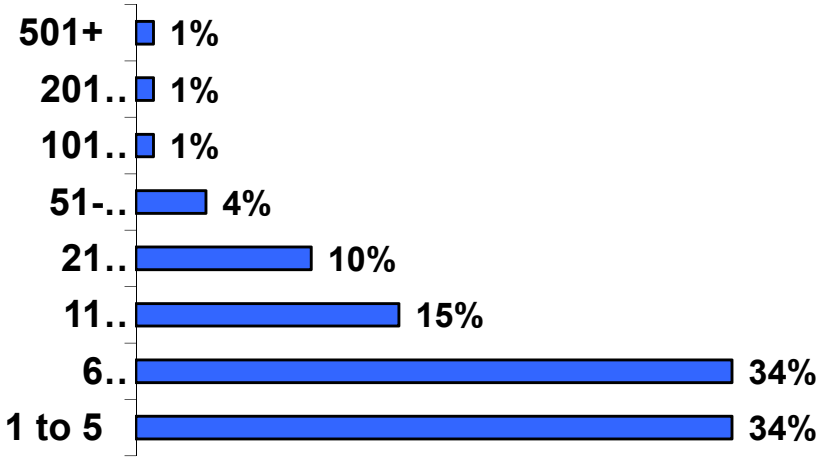

Figure 2.2 Agility Survey Project Sizes

In yet another study made by the author, a number of projects from the Free Software Foundation [30] were analyzed, and out of the 6259 projects included, a massive 5034 number of projects were single developer projects, 747 were 2-developer projects, just over 400 were done by teams sized 2-6, while the rest had team size greater than 6.

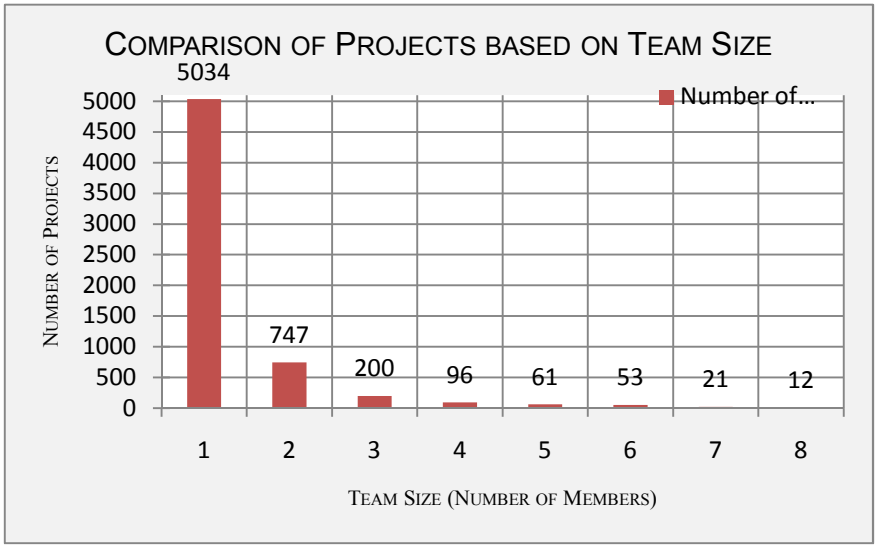

Figure 2.3 FSF Data Project Sizes

In spite of the presence of micro team projects on such a large scale, they have hardly formed an area of focus for developing software development methodologies. As a result, most teams follow adhoc processes, or try to tailor the existing methodologies for developing their software. Since all existing methodologies were developed either for large or small teams, there exist too many gaps when trying to fit them to micro teams. For instance, in teams with just a single developer, there cannot be pair programming. Similarly, there cannot be separate architects or testers or analysts to brainstorm ideas with.

This highlights the need to create an exclusive and complete software development framework for micro teams which can lead to efficient software development. This problem thus forms the basis of this thesis, which tries to address a part of the problem by doing a case study to come up with those activities that are characteristic of micro team development processes, and which could be used as a foundation for building a complete software development framework for them.

CHAPTER 3

THE CASE STUDY

3.1 INTRODUCTION:

A significant part of software development that occurs all around is actually done by micro teams of one to four persons, with just a single full time developer. Despite this fact, software development methodologies currently practised deal with either large development teams consisting of more than 25 developers, or smaller ones with 5-25 developers. There is no single methodology or framework which seeks to address the problems of software development by such micro teams.

This leaves micro teams with two ways to handle development: they follow no structured process, or use one of the existing methodologies by adapting it to their team structure. What is observed quite commonly, though, is that scaling down existing methodologies to micro teams leaves too many gaps in the implementation for the methodology to work efficiently. For instance, it is simply not possible to do pair programming in the presence of just one developer. Similarly, having daily stand up meetings to discuss the developer's progress has no meaning when there is no other technical person present.

Given that almost half of the software development that takes place is done by micro teams, there is thus an urgent need to create a software development framework that specifically addresses the challenges of micro team development processes.

3.2 PROBLEM STATEMENT:

The solution to the challenge of creating a software development framework can be approached in a few ways. One approach would be to recursively take an existing methodology and modify it randomly to suit the micro team structure, apply it to a micro team project and study the results of the process; now change the things that don't work, and repeat the same process – apply, get results, change methodology if needed – till a desired methodology is created. Another approach would be to first study a few micro team projects, find the common steps they do or steps which gave successful results, and then use them to create a framework. This thesis follows this second approach to come up with a solution.

To go about the approach, the question can be refined into four sub-questions:

i. What are the steps that micro teams follow while developing software?
ii. Of these, which steps are specific to micro teams and single developers, as opposed to being common to a development team of any size?
iii. How are these steps different from or comparable to steps followed in existing methodologies?
iv. What can be a set of steps that can be generalized into a software development framework to provide guidelines to any micro team?

This case study tries to answer all these questions by doing a multiple-case case study of micro team projects.

3.3 DESIGN COMPONENTS:

In accordance to a good case study design, our case study specifies four design components:

i. **Study Question:**

What are the steps that micro teams follow while developing software?

ii. **Study Proposition:**

Micro teams do not follow a standard software development process while developing software.

iii. **Unit of Analysis:**

Each step in the process followed by a developer in micro team projects.

iv. **Data Analysis:**

Pattern matching

- *We match the steps from each case of the study to steps from the other cases*
- *We then narrow down on those steps which are either common to all or most cases, or those steps which prove to be efficient or successful*
- *We then compare this set of steps to key practices in existing methodologies*

3.4 METHODOLOGY:

The methodology that we followed consisted of three steps: designing the case study and data collection methods, performing data collection and analyzing the data conforming to the design of case study.

To design the case study, we studied the case study research method and a few case study reports to come up with a preliminary case study design. Next, we determined the data collection methods to be used. We finalized two sources for collecting data:

- Interviews with team members of the micro teams whose projects were being studied:

  For the five teams available, we selected six people to be interviewed, each of them primarily a single developer of that team in different phases of project development, and/or a technical advisor.

- Collecting and studying the work products that were created in the process of the development:

  We got access to the work products of all teams on request to the team members and used them in our analysis.

3.5 DESCRIPTION OF CASES:

The case study consisted of five real life projects of commercial value developed by micro teams under university-industry collaboration. All of these projects were developed in the past three years, and had a development time ranging from three months to under two years. Each project was developed by a typical micro team – one single developer working full time on programming, and a few persons playing one or more of the following roles: a business analyst, a technical advisor, a project manager and a surrogate customer.

The five projects selected were as follows:

- Calendar
- Sensor Cloud
- Website
- Health Survey
- Complex Flow Analysis

The following table shows the business perspective of the projects selected:

PROJECT NAME	TYPE OF WORK DONE	DEVELOPMENT APPROACH	BUSINESS CRITICALITY	SIZE	STAKEHOLDERS
Calendar	Application partly built from slightly modified existing pieces of software; rest from scratch	Iterative and Incremental	Medium	Large	3 technical; 4 non-technical
Sensor Cloud	Prototype for application completely from scratch, with no pre-built software available for use	Prototyping / Iterative	High	Large	4 technical
Website	Improving performance of existing application by refactoring or external tools	Iterative	Low	Small	4 technical
Health Survey	Verification and minor feature extensions	Iterative	High	Small	3 technical; 1 non-technical
Complex Flow Analysis	Application built from scratch	Iterative and Incremental	Low	-	1 technical; 4 non-technical

Table 3.1 Case Study Projects

3.5.1 BUSINESS CRITICALITY AND SOFTWARE SIZE ESTIMATION:

3.5.1.1 BUSINESS CRITICALITY:

We defined business criticality as how important it was to complete the project in terms of the customer's needs. For e.g., could the customer live without it? How essential was the application to customer's business process? We asked the interviewees for their take on the business criticality of the project they were working on by asking them to rate it from 1-5. On a scale of 1 to 5, 1 being lowest, 5 highest; so 1 is not very essential, 5 absolutely essential.

3.5.1.2 SOFTWARE SIZE:

Software size was estimated using metrics by Larry Putnam [28]. We measured three things: number of features (customer's view of the software), number of components (developer's view of the software) and number of lines per component (actual work done). We then combined these three to get an approximate measure of whether the software size was small, medium or large [A.3].

Now we describe each of the projects below:

3.5.2 CALENDAR:

The Calendar project envisioned as a tool to help busy families replace chaos with order. With family-oriented time-management tools, Calendar would help its users save time, reduce stress, and better balance obligations at work, school and home:

- Shared Web Calendar

The Calendar marries the efficiency of an electronic calendar with the benefits of a traditional wall calendar by enabling registered users to create and manage a web-based calendar that

1) Synchronizes multiple electronic calendars from a variety of sources;

2) Can be viewed remotely from a lap top, mobile phone or PDA; and

3) May be conspicuously displayed in the home even when the user is not online

Not only does the Calendar effectively coordinate family members' appointments and events, but it can also manage schedules for other activities such as dinner menus, household chores, and home maintenance.

- List Management

Calendar enables users to create and share lists, including task ('to-do') list, grocery/shopping list, upcoming event reminders, and more. Lists can be linked to the calendar, such that, for example, upcoming activities auto-generate a to-do or reminder list, and dinner menus auto-generate a grocery list. Like the calendar, lists can be viewed remotely; they can also be retrieved or shared via email or text message.

Calendar is different from currently existing family management tools in the following ways:

- Simple, straight-forward look and feel
- Options to add travel time and get directions

- View and integration of other calendars
- Electronic wall calendar
- In-built list management

To develop a solution for this application, a group of seven persons was created in the beginning – a developer, a trainee developer, a project manager, a designer, and three surrogate customers. Based on a couple of brainstorming sessions, the architecture was decided upon, as shown below.

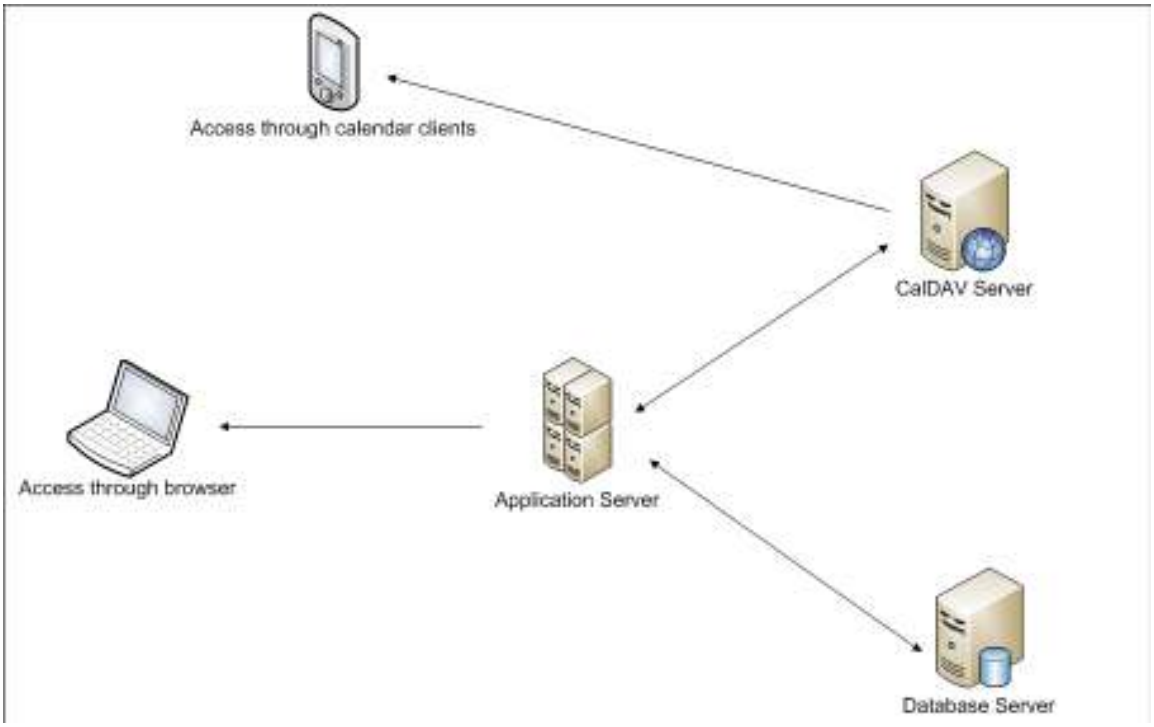

Figure 3.1 Calendar General Architecture

The technology to be used for development was decided as:

i. Presentation layer – Flex framework

ii. Business layer – Enterprise Java Beans framework

iii. Persistence – Hibernate (for ORM), MySQL Database Server

iv. Application Server – JBoss Application Server

v. CalDAV server – Bedework Server

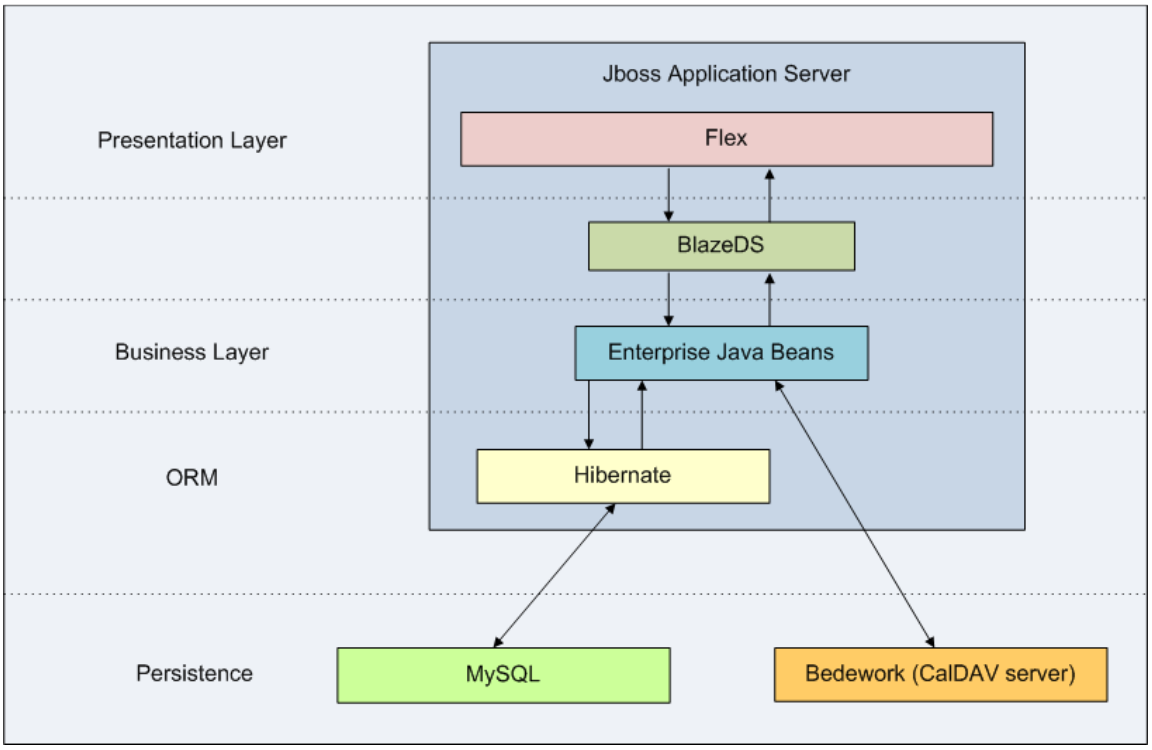

Figure 3.2 Calendar Technology Architecture

The solution that was developed primarily consisted of building an application in which a few components were built by modifying existing pieces of software slightly and using them, while the rest were constructed from scratch. The scope was medium in terms of business criticality and also in terms of the number of features to be developed. The entire development occurred over 13 months, and still continues.

3.5.2.1 DEVELOPMENT APPROACH:

The approach followed was agile and iterative, as seen by the different iterations and number of times SDLC phases were repeated. It was also incremental in that there were roughly three different cycles that delivered new features built on top of the previous ones.

3.5.2.1.1 CYCLE 1:

- Cycle one was pretty structured in its approach and roughly followed all the SDLC phases.
    - Requirements Gathering Phase 1
        - Clients, project manager, mentor developer, developer had a couple of meetings to transfer vision from client to developer
        - A tentative list of features was drawn
    - Analysis and Design Phase 1
        - Developer analyzed requirements and came up with a tentative design
    - Requirements Gathering Phase 2

        - Tentative list of features was refined further with the help of client, and turned to a list of stories
    - Analysis Phase 2
        - Client ranked stories by business value
        - Developer ranked stories by effort estimate
        - Based on both, stories were given priorities
        - Stories were then divided into groups and each group was assigned to an iteration in which they needed to be completed
    - Design Phase 2
        - In a couple of sessions, mentor developer, developer and project manager came up with a design and system architecture, with major input from mentor developer
        - Design was presented to clients, and approved
- This was followed by a round of several iterations in which actual implementation took place.
    - Iteration 1
        - Mentor developer implemented a framework for the entire system
        - The implementation of stories was based on this framework
        - After completion of implementation of stories in the iteration, they were shown to the client for approval
    - Iterations 2 – 7
        - Each iteration started with leftover stories from the earlier iteration and added them to the stories for that iteration.

        - All the stories for an iteration were then implemented in that iteration
    - Each iteration lasted one week
    - Stories were left unimplemented in iterations mostly because of the following reasons:
        - Effort estimate was calculated incorrectly
        - Lack of technical expertise
        - Business value changed
    - In some cases feedback from the client changed the stories and they had to be re-implemented in one of the next iterations
    - At the end of the 7 iterations, first version was released

3.5.2.1.2 CYCLE 2:

- Cycle two differed from cycle one in that unlike cycle one, it was not as structured around the release of a second version of the application
- The stories implemented in this phase were a few leftover from cycle one, and new features added by the client
- Also, the original developer was not a part of cycle two, allowing the trainee developer to take over as the full time developer
    - Requirements Gathering and Analysis
        - New features were added to the original unimplemented stories
        - Features were further refined into stories
        - Client ranked this new set of stories by business value
        - Developer ranked the stories by effort estimate
        - Based on both, stories were assigned priorities

- This time implementation started directly, without dividing stories into iterations
- There was no hard and fast time limit for implementing each feature, but a broad deadline was set for a beta release of the application.
- Each story took about 8-14 days, depending on the level of difficulty
- Most of the features required were finished in cycle two

3.5.2.1.3 CYCLE 3:

- One major task (deployment) and bug fixes needed to be completed in cycle three.
- At this time the developer was only available part time.
- Typically the bug fixes were done in 7-8 days after a bug popped up
- After most of the major bugs were fixed, deployment was completed

3.5.2.1.4 MISCELLANEOUS POINTS:

- For development, the developer used framework/template implemented by original developer – this allowed for easy and rapid development
- Code review sessions were held twice to check understanding of developer
- Risks were medium; so a simple iterative & incremental model used

3.5.2.1.5 SCREENSHOTS:

Figure 3.3 Calendar Screenshot

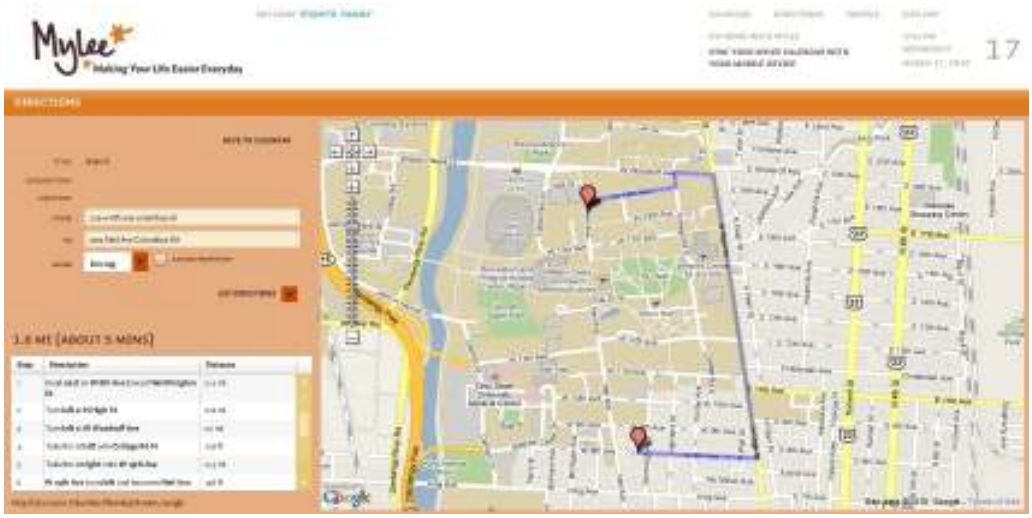

Figure 3.4 Calendar Directions View Screenshot

3.5.3 CLOUD SENSOR:

The purpose of the Cloud Sensor system is to provide the counterpart to the gateway module. The gateway module and the Cloud Sensor System are intended to form a tightly coupled whole, which connects a Wireless Sensor Network (WSN) to the global computing infrastructure (symbolized by the Internet). The gateway module will be the portion of this duo that resides in the WSN and the Cloud Sensor system will be a universally accusable Internet service.

The relationship of the Cloud Sensor System to its "ecosystem" is shown in

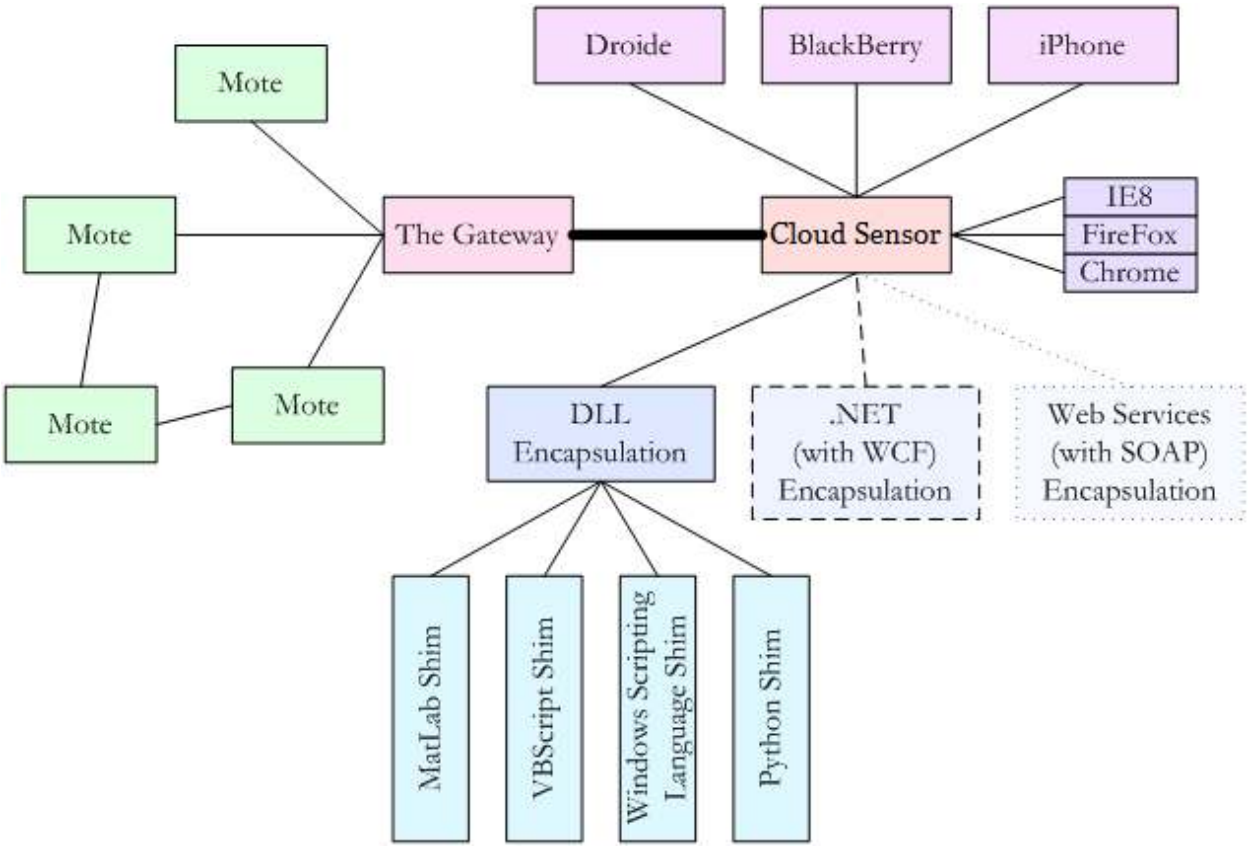

Figure 3.5 Cloud Sensor Architecture

3.5.3.1 RELATIONSHIP WITH THE ENVIRONMENT:

3.5.3.1.1 THE GATEWAY:

Each Gateway will connect to the Cloud Sensor system through the Internet. It is assumed that this connection will sometimes be a TCP connection and sometimes a stream of UDP packets. TCP provides reliable communication, but imposes a minimum overhead of approximately 1 packet every 90 seconds. As a result keeping a TCP connection open all the time may be inappropriate if the useful communication level is far lower than the overhead level. In these cases it will be necessary to either create a TCP connect for a short burst of reliable communication and then tear-down the connection, or to use UDP and implement some sort of reliability at the Cloud Sensor level. It is expected that the latter case will be more appropriate in situations where some loss of data may be acceptable at the application level.

In any event it is expected that Cloud Sensor and The Gateway will be tightly coupled. For example they should assume between 10 ms and 1 sec of latency and will be designed to exploit this low latency messaging. Furthermore they should exploit this low latency link in order to maintain tightly coupled state.

However, the design should go to some length to conserve bandwidth. Is should work for connections as slow as 50 kbps. It should reasonably make use of increased throughput up to about 10 Mbps. The design should include a data throttling mechanism that can be used for both testing and to lessen the impact of a high bandwidth run-away user.

3.5.3.2 PRIMARY FUNCTIONS:

The Cloud Sensor system will need to perform the following primary functions:

a) Allow applications to be loaded from a desktop computer, through the internet service, to the gateway so that a gateway service could use to push these applications onto individual WSN nodes.
b) Provide an API to gateway service module can expose to applications that might ultimately be run on the

3.5.3.3 KEY INTERFACES FOR THE CLOUD SENSOR SYSTEM:

- Gateway server - Cloud Sensor gateway service: An API that a gateway server will use to
  - Connect to the Cloud Sensor system
  - Upload mote application data
  - Download applications from the system
  - Download application configuration data (i.e. changes made on Cloud Sensor by human users/programs)
- Cloud Sensor system - Connecting to phones: APIs to connect to Cloud Sensor and get data for human consumption, from mobile platforms
- Cloud Sensor - DLL encapsulation: Libraries that can be imported and used by developers to programmatically
  - Access application related data from Cloud Sensor
  - Control applications through Cloud Sensor

3.5.3.4 DEVELOPMENT APPROACH:

The project basically consisted of building a prototype for an application completely from scratch, with no pre-built software available for use. Scope was large in terms of business criticality as also in terms of the number of components/features. The development model followed was prototyping and continued to be iterative. Development occurred over 7 months, and still continues. The development team initially consisted of 4 persons, and later expanded to comprise of 6 persons.

The various phases that took place were as follows:

- Requirements Gathering
  - The project managers played the role of clients too
  - Clients and developer had a few sessions to transfer clients' vision to the developer
- Analysis and Design
  - Developer started creating a design document based on requirements documents provided by clients
  - The design document was submitted back to the clients/project managers for feedback
  - Based on the feedback from clients/project managers, developer modified design document
- Architecture design
  - Consisted of two main parts:

        - Intensive research to find all possible technology choices
        - Actual architecture design using the selected technology
    - Most of the architecture design phase was consumed by research on available technology which resulted in a document of a list of technologies and their details
    - Project managers and developer evaluated the document, and the best choice was selected
    - Developer created an architecture using selected technology
- Implementation
    - Developer built a prototype model based on the architecture and it was shown to clients for approval
- Miscellaneous points
    - Design was constrained by the software platform in case of the server side, and the hardware platform in case of the client side
    - Risks associated with the project were huge, so a prototyping model for development worked better (impact of failure huge; recovery from failure might involve redesign of entire system)

3.5.4 WEBSITE:

The Website was the company's website designed to display information and contact details about the company, allowing any person to interact with the company. The project consisted of four members: a developer, two surrogate customers who also played the role of project managers, and a technical advisor who was also a past developer.

The technology that was used in this project was:

a) Google App Engine

b) Java Spring framework

3.5.4.1 SCREENSHOTS:

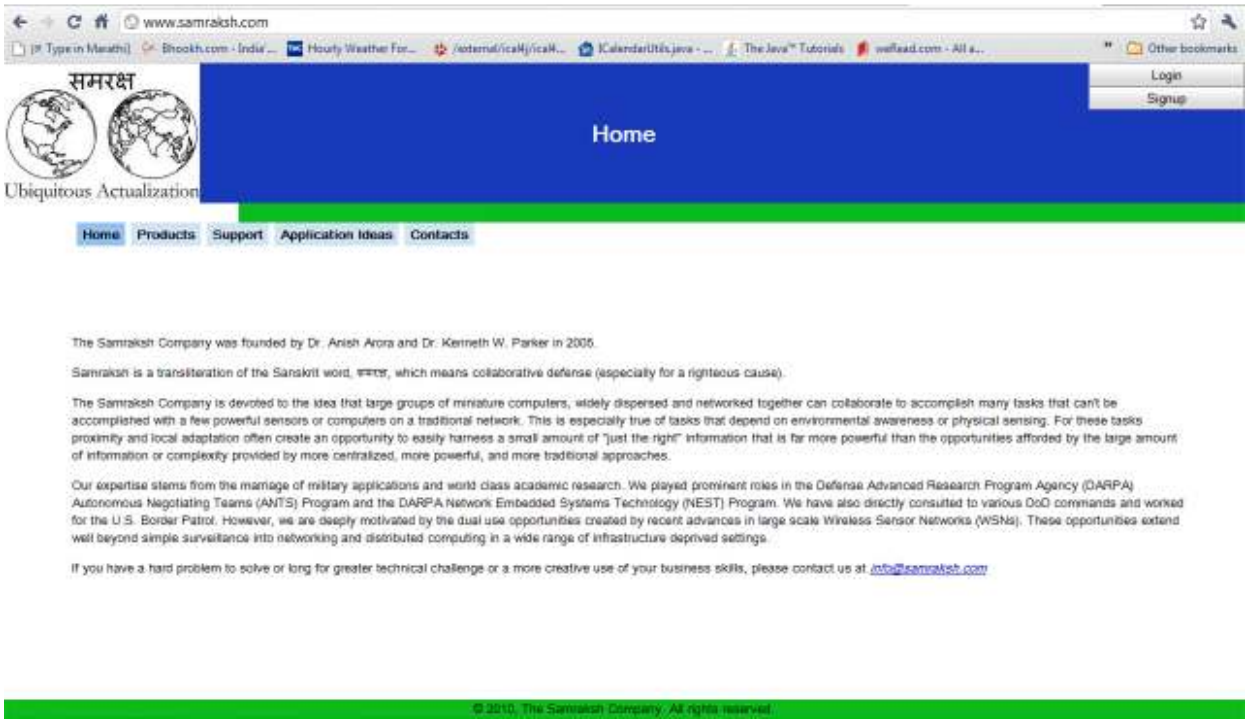

Figure 3.6 Website Screenshot 1

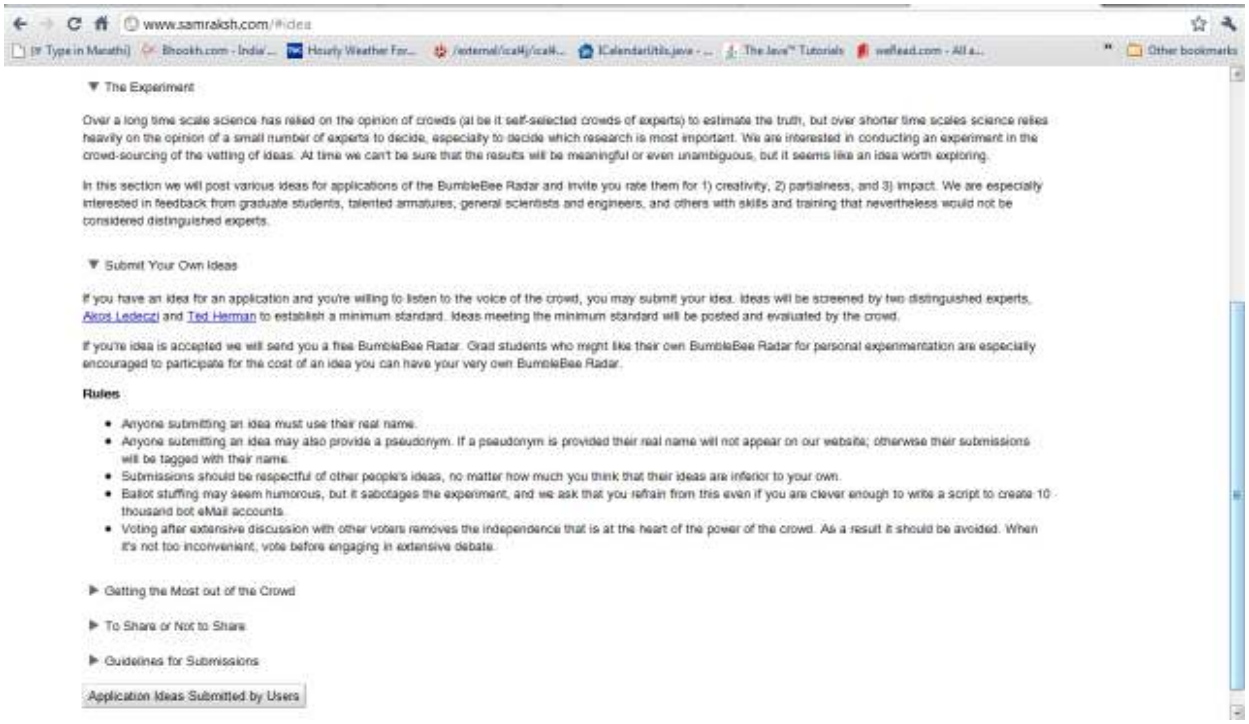

Figure 3.7 Website screenshot 2

3.5.4.2 DEVELOPMENT APPROACH:

The project primarily consisted of improving performance of an existing application either by refactoring a considerable part of the code or using external tools. Scope was medium in terms of business criticality. The development approach was iterative. Development occurred over 3 months. The development team consisted of 5 persons. The phases followed in the process were as follows:

- Requirements Gathering
    - A few meetings with clients and the developer working on the project before current developer joined helped in getting the requirements clear
    - Learning the existing system and technology also clarified requirements
- Analysis

- Consisted of researching how similar problems were solved and whether those approaches could be used in this particular case
- This led to two approaches to design

– Design and Implementation
  - Based on analysis, two designs were developed and presented to the project managers
  - Based on project managers' evaluation, the selected design option was selected for implementation
  - This turned out to be unsuccessful due to lack of technical expertise, resources and time

– Re-Analysis and Implementation
  - After re-evaluating, the second design was chosen for implementation instead of changing the first one and re-doing it.
  - The second design led to a successful implementation

– Miscellaneous points
  - Risks associated with the project were minimal, so an iterative model worked (i.e. impact of failure minimal; recovery from failure straightforward)
  - Reason for above is that project consisted of modifications to existing software
  - Two cases present – minimal modification vs. a major code refactoring. Here the tendency seen was to start with simplest solution first

3.5.5 HEALTH SURVEY:

The Health Survey project was a research project based upon a set of questions which are to be answered by a user who has had a heart attack in the past and underwent treatment for it. The primary aim of this study is to increase our understanding of care-seeking behavior surrounding acute coronary syndromes through the application and testing of an integrated self-regulatory care-seeking model and the utilization of a world wide web based ISCM survey instrument. At present, care-seeking delay among individuals stricken with ACS prevents many patients from obtaining the full therapeutic benefit of hospital based thrombolytic and/or mechanical reperfusion therapy to reduce the morbid and mortal consequences of acute cardiovascular disease. Results from prior studies of ACS care-seeking delay generally rely on measures of time duration from acute symptom onset to hospital emergency department arrival and measures of social demographic and clinical variables, extracted from emergency medical system logs and/or ED charts. By relying on such limited data sources, these studies have failed to take into consideration the complexity of the social and behavioral processes by which individuals make decisions to seek medical care for symptoms of ACS.

By applying and testing an integrated self-regulatory care-seeking model to ACS care-seeking, it will be possible to delineate decision points and situations and circumstances that are critical to producing patterns of care-seeking that are efficient and expeditious or are protracted and delayed. The care-seeking process in the ISCM is viewed in terms of five analytic care-seeking phases from warning or premonitory symptoms to ED arrival. By noting the presence or absence of three central phases, it is possible to distinguish patterns of care-seeking and to record the duration of each care-

seeking phase and total duration of each care-seeking path to the ED. The ISCM assumes that coping behaviors emerge over time and initially care-seeking behaviors are guided by demographic and structural factors. As the self-regulatory and coping processes emerge over the course of the ACS episode, the influence of these factors diminishes and emergent ACS evaluations, advice from others and health care providers, emerging symptoms and emotions, come to dominate the care-seeking process and directly determine the duration of ACS care-seeking. The study will test and explore the five hypotheses in all.

The proposed study utilizes a twice tested pilot ISCM survey instrument placed on two secure dedicated web servers to collect patient care-seeking data. The target sample size for this study is 2,314 patients who have experienced an ACS episode. Subjects will be recruited using: 1. extensive marketing of the study URL on websites of interest to cardiac patients and oriented toward cardiac health and ACS risks, such as, diabetes or hypertension; 2. placement of study information on discussion boards and chat rooms with a cardiac health focus; 3. emailing of a study flyer PDF for posting in public libraries, senior citizen centers, African-American churches, county and municipal health clinics, and HMO web sites; 4. application of a viral marketing strategy; and 5. use of selected print media, for example, AARP publications. Marketing of the study site URL will emphasize the recruitment of women, the elderly and minorities. In the data collection phase of the study, the proportions of myocardial infarction and angina pectoris subjects falling into each cell of the cross classification of age, sex and race will be derived from studies. These estimates of the population proportions in these cells will be used to determine quotas for each of the cells in an attempt to ensure all risk groups

are adequately represented. We will over sample African-Americans. Statistical models will be used to estimate the effects of variables that characterize the decision process on the time delay until ED arrival. Time required to move from one phase of the decision process to the next will also be modeled, as will the different paths chosen by the subjects. Models will also include patient characteristics and other circumstantial and situational variables in the decision process.

A report is generated in CSV format for about 147 answers given by every user and it is used by the researcher in his research. The project is basically a website developed using CakePHP framework. This was earlier developed by another developer before being handed over to the micro team. When the project was handed over to the micro team, the website was not fully functional. Basically, the output generated was not in sync with the user answers. The work done by the micro team was mostly bug fixing kind of work.

The duration of project was approximately 2 months time. But major development work started in 2nd week of December and finished on 10th January 2011. A particular challenge was that the previous developer had left abruptly and it took time to communicate with him to understand what code changes he had made. Other challenges that were a part of the project were:

- When the project was received it was behind and had been neglected. The customer also felt that the current development team was neglecting it. The team communicated with the customer what they were doing and gave him a realist commitment for completion as well as level of effort.

- Keeping the project scope to the committed tasks. When people tried to add tasks, the team would take care to remember they wanted to first complete the stuff they had committed to.
- Lack of familiarity with the frame work. A lot of time was spent reading the documentation and researching the framework.

3.5.5.1 TECHNOLOGY USED:

4 Presentation and Business layer – CakePHP framework

5 Backend – MySQL database server

3.5.5.2 DEVELOPMENT APPROACH:

The project primarily consisted of system verification and minor feature extensions. Scope was medium in terms of business criticality and small in terms of the number of components/features. The development approach was iterative and incremental. Development occurred over 3 months. The development team consisted of 3 persons.

The phases in the process were as follows:

6 Requirements Gathering

6.1 This was done mainly through exchange of emails between the client, project manager, current developer, mentor developer and ex-developer

6.2 No in-person meeting with the client took place

7 Analysis

7.1 The project consisted of mainly fixing broken features, so a lot of analysis was in the form of how the code worked and where the problems lay in it

8 Implementation

8.1 Most implementation took the form of code correction

- Miscellaneous points
  - Risks associated with the project were minimal, so an iterative model worked (i.e. impact of failure minimal; recovery from failure straightforward)
  - Reason for above is that project consisted of minor modifications and extensions to existing software

3.5.6 COMPLEX FLOW ANALYSIS:

Complex Flow Analysis was the name given to a methodology being developed by the customer. The center was building a practice in which they coordinated with various companies to build up by-product synergy networks (basically, groups of companies willing to work with one another to redirect their waste streams by giving or taking waste from each other and using it as an alternative fuel, recycled materials, etc.). The goal of these networks was to both reduce costs and their environmental footprint. At the time this micro team began working with them, they were primarily working with a group of companies that had already agreed to work together (in Kansas) and were trying to establish and similar network of companies locally in Ohio. In addition to getting the companies connected to one another, the center acted as a mediator that could determine how to redirect the waste streams and make proposals based on their calculations (if a single company did this they might want to do things that were in their best interest rather than in the best interests of all the companies). Getting all the data from each company, doing the mathematical optimizations, communicating proposals (cost/benefit of changes to each company), took a lot of work so the Complex Flow Analysis software was intended to be a tool that would make it easier.

The mathematical analysis that the center did as part of their analysis was often difficult to communicate to clients and they used custom spreadsheets and drawings to share data with their clients and communicate the result. The Complex Flow Analysis software was designed to allow clients to be more engaged in the modeling and optimization part of the process without having to know all the details. They also hoped it would allow them to standardize data collection & share results more easily without the

needs for custom spreadsheet, etc. The micro team developed a software package based on Eclipse RCP that could organize "projects" and associated data (models, solutions, etc.). A large part of it was a GUI that allowed networks to be created, saved, updated, viewed, etc. via drag/drop operations. Material flows in these networks could be optimized (by a user definable set of criteria) with a single click. This also helped avoid the possibility of errors because we found that even when expert graduate students tried to formulate these networks by hand they often made errors. Complex Flow Analysis was eventually linked with another tool, which the micro team developed at the same time with this group.

The team consisted of a single developer, two grad students and two directors at the customer's company. The grad students worked with the clients to model the by-product synergy networks, did the data analysis, collected data, etc. They were also the primary users of the Complex Flow Analysis software. The clients used it as well, but the grad students were usually there to answer questions, help them do the modeling in the tools, etc. The two directors also worked with clients and set the overall vision for the group.

For the majority of the project the requirements were handled much like a top ten list. The team had group meetings where they would talk about what needed to be implemented next. The developer would maintain this list of the top requirements and share it. The developer also kept a private backlog, which he occasionally used to start up discussions. He would typically demo the software to the group in later meters for feedback/acceptance. The developer had very little interaction with the center's clients that used the tool. Although, the grad students were really the "power" users of the tool

(because it helped them do their work, which was much more intensive than anything the clients did with the tool), so the developer got lots of feedback from them during these meetings. The team rarely had deadlines, although the developer would provide rough estimates. The grad students would occasionally check the results Complex Flow Analysis generated against solutions worked out in other software.

The process was largely a result of how the customers wanted to work. The developer avoided asking them to do very much technical stuff, like looking over use cases, etc – which was how a previous developer had approached the project.

The project consisted of building a tool for getting data from different sources, doing mathematical optimizations and communicating proposals. Scope was large in terms of business criticality as also in terms of the number of components/features. The development approach was iterative and incremental. Development occurred for about 2 years. The development team consisted of 5 persons.

3.6 QUESTIONNAIRES

The questionnaires prepared focused on finding out what process was followed by the micro teams from the perspective of the team members.

3.6.1 FIRST QUESTIONNAIRE:

The first questionnaire had questions common to all participants:

1) Can you give an overview of the project?
2) Can you describe –
    a. Who were the persons involved in the project, and in what capacity?
    b. How did the project start and end?
    c. What kind of work was done in it?
3) In the development life cycle of the project, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?
4) As a single developer, what challenges did you face in the course of the project? For instance, learning new technology, problems related to programming/deployment, interaction with the customer/project manager, etc. How did you resolve these problems?
5) What was the duration of the project? How did you keep up the motivation to keep working? Were there any other non-technical challenges you faced? If yes, how did you overcome them?
6) Any other thoughts about the project and the process?

3.6.2 SECOND QUESTIONNAIRE:

The second questionnaire was customized for each participant based on their previous answers. Each one was asked a subset of the following questions:

1) What did your development process look like? Were there clearly separable phases like a requirements collection phase, analysis, design, implementation phases, etc? If not, what was the process? Was a waterfall model followed (with all requirements known at the beginning)? If not, how did you manage changing requirements?
2) How did you come up with the development process? Was the process similar to development processes of other projects you worked on? Or was it shaped to suit the client's style of working? If the latter, can you give an example of the client's style of work and how it shaped your process?
3) While developing different features/components in the project, for the development of each component was the same process used? Or did the implicit nature of a component/feature warrant a different kind of process? E.g., one component might have been suited best for a Test Driven Development approach, so you created the test cases before implementing, while another followed the traditional approach.
4) Did you have to learn new technology for implementing the project? If yes, how much time did it take? What did you do when you were stuck on something? For e.g., did you go to someone who was an expert, or used any other resource? If you went to an expert, would you say that the learning curve and development process was shortened due to the presence of this technical expert in your team?

5) When the project was passed on to you, did you have any difficulties in the knowledge transfer process? Was the earlier developer available easily to clarify doubts?
6) Did you create any kind of a document to record your progress? For example, say, a Word file that would contain all the requirements, bugs, completed tasks, etc?
7) What kind of collaborative workspace did you and your customers and/or team members use to communicate?
8) Can you specify how many and what kind of work products were created in the entire process? For instance, the source code would be one; the requirements document another, and so on. You can list any document that was created along with what its contents were in the process. (Note - if it is possible for you to provide the document, it will help me a lot; if not, a detailed description of what the document was for and its contents will also work.)
9) How frequently do you have meetings with the customer? Is this frequency sufficient, or do you need more frequent or less frequent meetings? When certain issues are discussed in a meeting, are they addressed completely during the time leading up to the next meeting? Or is this period too short to address the issues or too long such that you have free time on your hands?

3.6.3 THIRD QUESTIONNAIRE:

A third questionnaire was then sent out to get a measure of project size and business criticality. It consisted of objective type questions and was common to all interviewees.

3.7 WORK PRODUCTS

For each project, the work products created during the process were studied. The types and number of work products in each project were as follows:

PROJECT	WORK PRODUCTS
Calendar	Use cases, Test cases, Document with stories ranked and prioritized, Source code for the application, Internal Trac page for tracking progress – workbook
Cloud Sensor	Source code, Requirements document, Design document
Website	Source code
Health Survey	Source code, Heart study code book, Developer's manual, Internal Trac page for tracking progress – workbook
Complex Flow Analysis	Test cases, User specifications, Internal Trac page for tracking progress – workbook

Table 3.2 Work Products

3.8 PARTICIPANT RESPONSES:

All the participants in the case study who were given questionnaires were very cooperative and responsive. The responses to the quantitative questions are captured and presented below.

3.8.1 Use of workbook for intra-team communication

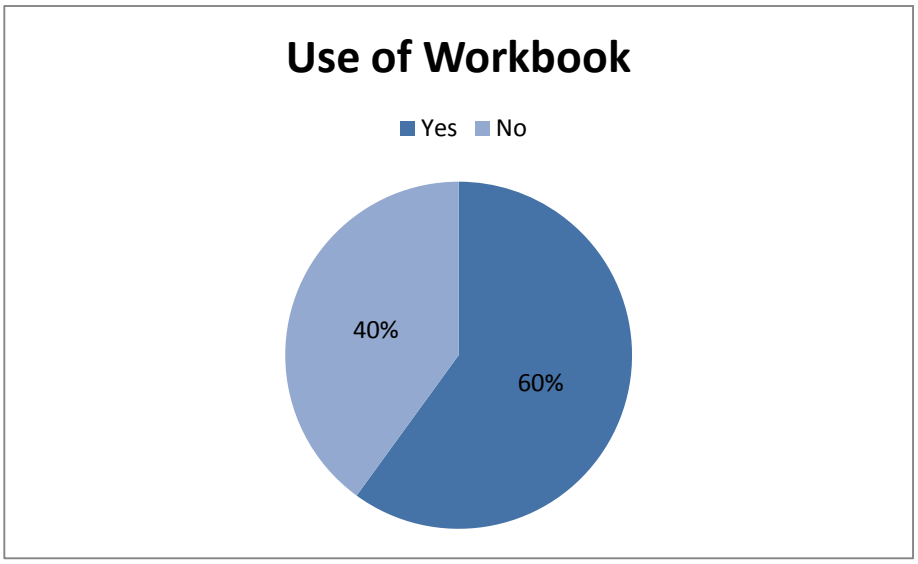

Figure 3.8 Use of workbook

3.8.2 Number and types of work products

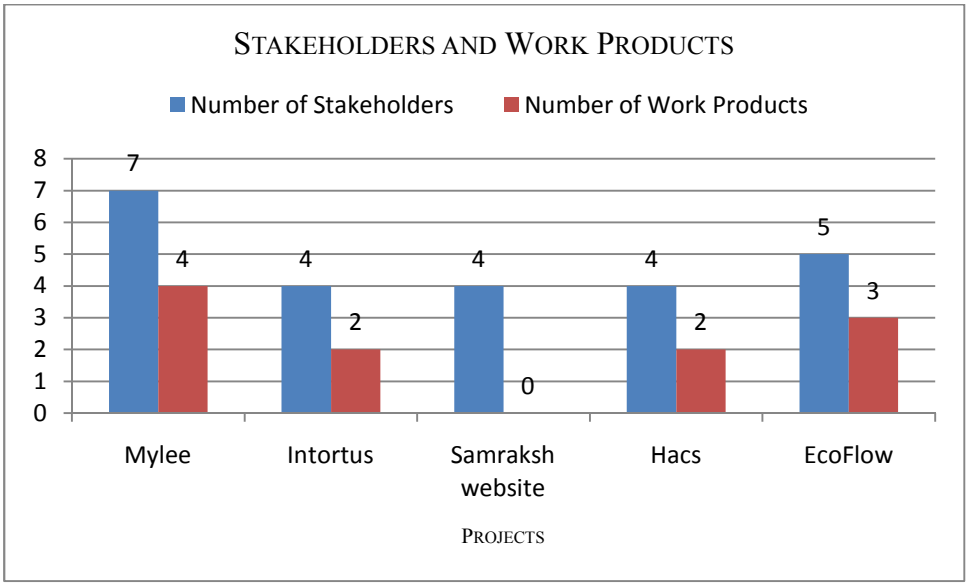

Figure 3.9 Stakeholders and Work Products

3.8.3 Interaction with customers

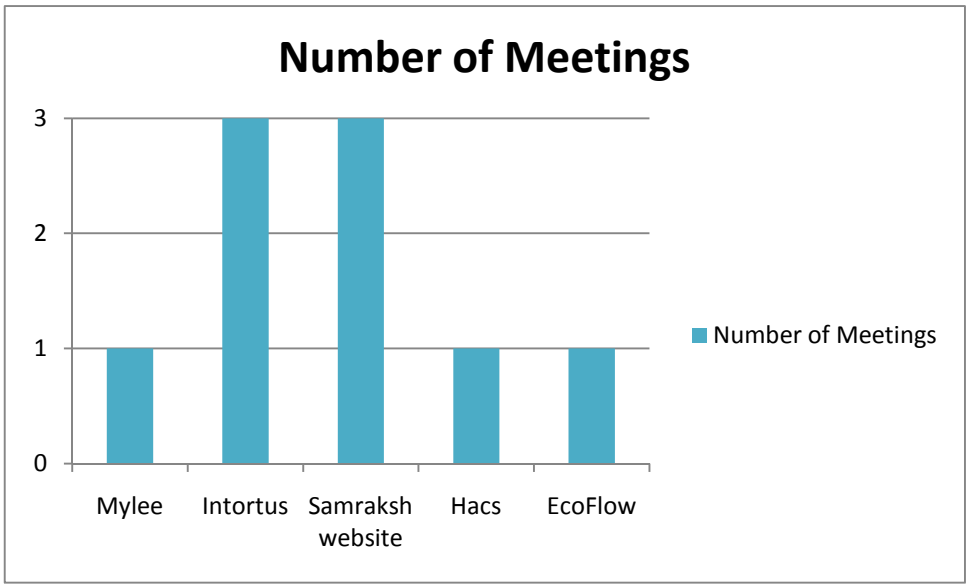

Figure 3.10 Meetings per week

3.8.4 Presence of a technical expert

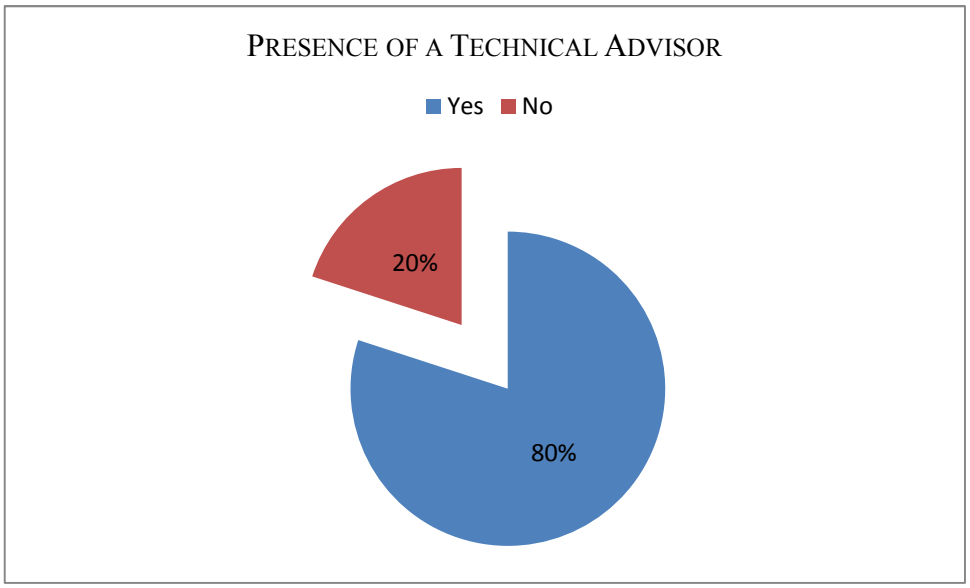

Figure 3.11 Presence of Technical Advisor

3.8.5 Knowledge transfer challenge

PROJECT	ADEQUATE DOCUMENTATION	FORMAL K.T. PROCESS	AVAILABILITY OF PREVIOUS DEVELOPER
CALENDAR	×	✓	✓
CLOUD SENSOR	×	×	✓
WEBSITE	×	×	✓
HEALTH SURVEY	×	×	×

Table 3.3 Knowledge Transfer Issues

3.9 OBSERVATIONS AND INSIGHTS:

The results of the case study yielded ten important observations and insights, listed below in a table compared to existing agile and non-agile practices.

KEY PRACTICES	MICRO TEAM APPROACH	AGILE APPROACH	NON-AGILE APPROACH
Intra-team Communication	Use of a workbook	Story chart/pair programming	Documentation
Documentation	Non-existent	Very light	Heavy documentation
Number and Types of Work Products	Depend on number and types of stakeholders	Depend on number/types of stakeholders; but a few are fixed	Fixed, changing only with application type
Cost to Add / Replace New Developers	Very high	Flexible – can be low if planned	High
Process Flexibility w/ Development Approaches	High	Medium	Low
Customer Interaction	Frequent, impromptu	Frequent, impromptu	Structured, scheduled
Subject Matter Experts	May be in team or brought from outside of team	Present in team	Present in team
Technical Design of Solution	Constrained by knowledge of single developer	More than one developer to evaluate/design	More than one developer and/or architect
Development Process Style	Reflects balance of power between customer and team	Reflects team's style of working, constrained by customer input	Reflects team's style of working entirely
Knowledge Transfer	Very challenging – no procedure	Easy – team interaction	Time consuming – documentation based

Table 3.4 Key Practices

This section answers the first three questions of the problem statement, viz.

a. What are the steps that single developers in micro teams follow while developing software?
b. Of these, which steps are specific to micro teams and single developers, as opposed to being common to a development team of any size?
c. How are these steps different from or comparable to steps followed in existing methodologies?

The last question is answered at the end of the chapter.

A detailed discussion of each observation follows:

3.9.1 INTRA-TEAM COMMUNICATION:

- In three of the five projects in the case study, an internal workspace was used by the development team to record the progress of the project.
- Items recorded included requirements, bugs, feedback from customer/other stakeholders, work products created in the process, and other issues.
- The workbook approach served as a communication space for the team and also ensured that everyone on the development team was using consistent terminology and had the same understanding about each issue.
- In contrast, the agile approach uses tools such as story charts and pair programming for communication within the team [32].
- Non-agile approach uses heavy documentation for communication; several documents are created in each development phase [10].

3.9.2 DOCUMENTATION:

- Often when a single developer is developing a project, the emphasis on creating developer-specific documentation is minimal. As opposed to this, when multiple developers are developing a project, communication and coordination between developers warrants creation of developer-specific documentation.
- This lack of adequate documentation was observed in 4 of the 5 projects we studied
- Problems were caused when these projects were handed over to the next developer and they had no information about where the project was in terms of solution implemented except for the source code
- In contrast, agile teams produce very light documentation – only as much as required. Instead, practices such as pair programming and team co-location [33]
- Non-agile approach makes use of very heavy documentation [10].

3.9.3 NUMBER AND TYPES OF WORK PRODUCTS

- The scope of the project and the number of stakeholders involved in it determined how many work products were created, and of what type
- We found that the more the number of stakeholders, more the number of work products were produced
- The types of stakeholders also affected the kinds of work products produced. For instance,

- Presence of more non-technical customers produced user stories documents (as in Calendar)
- Presence of technical customers produced more formal documents like requirements document, design document (as in Cloud Sensor)
- Presence of only a single developer stakeholder in the team meant that developer-related work products i.e. technical documentation never got created.

- The work products created in agile also depend on stakeholders; but some work products are always produced – story cards, iteration charts etc [33].
- The work products created in non-agile processes, on the other hand, are almost always fixed in number and type – only a few work products, depending on the application type, may differ in type and number [10].

PROJECT	NUMBER OF STAKE-HOLDERS	TYPE OF PROJECT	TYPE OF STAKEHOLDERS	COLLABORATIVE WORK PRODUCTS PRODUCED
Calendar	7	New application	4 non-technical customers	4 – workbook, use cases, test cases, document with user stories
Cloud Sensor	4	New application	2 technical customers	2 – requirements document, design document
Health Survey	4	Modification to existing application	1 non-technical customer	1 – workbook
Website	4	Modification to existing application	2 technical customers	None
Complex Flow Analysis	5	New application	4 non-technical customers	3 – workbook, test cases, specifications

Table 3.5 Work Products & Stakeholders

3.9.4 COST TO ADD/REPLACE NEW DEVELOPERS:

- Cost of adding a new developer is very high for teams with only single developers in comparison to teams with more than one developer, since there is only one developer available to train as well as write code (while adding a new developer), and in some cases no one to code at all (while replacing a developer).
    - E.g. – this situation arose in both Calendar and Health Survey projects. At the point of transition, there was no developer actively programming, with the new developers mostly just undergoing training. This brought work to a complete halt for some time in both the projects.
- In contrast, this cost is relatively a lot lower in agile teams. Agile teams use pair programming and team co-location as primary tools [33]. Also, the presence of more than one programmer ensures that development, though slowed down for some time, does not come to a halt.
- Non-agile teams also have the advantage of more than one developer in the team, which means development does not come to a halt. But training new developers is a lot more time consuming [31] than in agile teams since training is done mainly through documentation and there are no tools such as pair programming and team co-location to supplement it.

3.9.5 PROCESS FLEXIBILITY WITH DEVELOPMENT APPROACHES:

- Development processes of any team – whether micro or small following agile or large following non-agile approaches – show flexibility with respect to the development approaches that they follow [33].
- What differs, though, is the degree of flexibility seen in each – micro team processes are seen to be the most flexible ones, closely followed by agile and last of all non-agile teams. This flexibility depends on the team size – smaller the team size, lower the momentum, and easier it is to change approaches; whereas larger the team, greater the momentum, and it becomes more difficult to change approaches.
- The development process followed is usually not clearly defined, and consists of only a single developer, hence very flexible – allows different parts of the same project to be developed using different approaches
    - In one of the projects (Complex Flow Analysis), the traditional development process was followed for developing most components
    - For developing one specific component, though, a test driven development approach was more suitable; hence the developer changed the development process from traditional to test driven
    - Another project (Cloud Sensor), also showed this flexibility – a data storage component was developed using a test driven approach, while a security component using the traditional approach

- This would have been more difficult in a larger team, because larger teams need more coordination, and getting out of a fixed process would be more difficult

3.9.6 CUSTOMER INTERACTION:

- In all the projects studied, the development team had weekly meetings with the customer, either over the phone or in person
- Each meeting served three purposes:
    - Demo of the software built and/or discussion of other work done by development team
    - Requirements discussion – determining the next tasks on the to-do list
    - Feedback from customer/other users on solution implemented till now
- This was important because the development process had no clearly defined path, so constant validation of work done and specifying next steps was needed
- Defining long term steps and infrequent feedback on work done (e.g. feedback every month as opposed to every week) might lead to waste of time and effort in case there was misunderstanding of task/requirement by developer or misunderstanding of what deliverable would be by customer
- Agile teams also have frequent and impromptu interactions with customers, with customers being considerably engaged in the process.
- Non-agile teams, on the other hand, have interactions with customers that are a lot more structured and scheduled.

3.9.7 SUBJECT MATTER EXPERTS / TECHNICAL EXPERTS:

- In micro teams, an SME or technical expert might not always be part of the team; instead, they may have to be specially brought in from outside.
- In four of the five projects, there existed a technical advisor either in the team or outside the team but related to the project.
- The developer's learning curve for the technology and/or project shortened as well as the general development process sped up due to the presence of such a person.
- In agile and non-agile teams, on the other hand, SMEs and technical experts are part of the team itself [34] [35].

3.9.8 TECHNICAL DESIGN OF SOLUTION:

- Since only a single developer worked on each project, the technical design of the solution was constrained by their knowledge.
- Systematic evaluation and identification for the best framework is not done in most cases since there is only one developer
- For e.g., CakePHP for Health Survey, Google App Engine and Spring for Website, Calendar with J2EE framework
- As a result, any new developer who replaces the previous one is stuck with this design; they have to learn it if it is not already known, and then have to work on the design – whether or not it is the best solution.
- Agile and non-agile teams, on the other hand, have more than one developer to evaluate and design a solution; in fact, both these teams will in most cases have a special architect or group of architects to take care of the design [34] [35].

3.9.9 DEVELOPMENT PROCESS STYLE:

- The style of development that the process follows in a micro team environment reflects the balance of power between the customer and the team.
- A non-technical customer was seen to be less powerful than the micro team (Calendar, Health Survey), and the development process reflected the micro team's style of work.
- A technical customer, though, was more powerful than the micro team and had more control in how development took place. In two other projects (Cloud Sensor, Website), it was the customer who wanted stand up meetings every other day, so work had to structured taking that into account
- What also played a part was the composition of the team – when a majority of the team was made up of surrogate customers as opposed to other roles, the balance again tilted in the customer's favor. In one project (Complex Flow Analysis – 1 developer, 4 customers), the developer fit into the customer's schedule and meetings instead of creating his own schedule and requirements and then asking the customer to fit into that.
- Agile teams, though, reflect the team's style of working – but the process is shaped considerably by customer input.
- Non-agile teams' development processes completely reflect the team's style of work.

3.9.10 KNOWLEDGE TRANSFER:

- Of the five projects, four were handed over from one developer to another
- Knowledge transfer in these cases proved to be a challenge because of one or more of three reasons:
    - Lack of adequate documentation
    - No formal knowledge transfer process followed (meaning both developers sitting together to teach and understand)
    - Unavailability of previous developer to answer questions
- One project (Health Survey) found the knowledge transfer especially challenging because of the presence of all the above three reasons
- This process is relatively easier in the agile approach, with a lot of team interaction happening through tools such as pair programming and team co-location [33].
- The non-agile approach usually has formal knowledge transfer processes in place, with a lot of stress on documentation based knowledge transfer – this results in a time consuming process [31].

3.10 SET OF GUIDELINES:

The last question of the problem statement, i.e. what can be a set of steps that can be generalized to provide guidelines to any micro team and be a basis for building a development methodology, can be answered as follows:

I. Intra-team communication – workbook oriented:

The workbook approach worked well for teams that used it for communication, and progress tracking, so it can be a good option for other micro teams as well.

II. Focus on developer-specific documentation:

Projects invariably get transferred from one developer to another, either within a team or between teams. In such a case, it is important to create developer's documentation.

III. Work products specification based on stakeholders:

It might be a good idea to loosely draw up a list of work products to be created that would best suit each team's composition and type and number of customers; this would enable the team to structure their process around those work products.

IV. Offsetting new developer costs:

Since costs of adding/replacing developers are very high, ways to offset these costs need to be thought of.

V. Use of subject matter / technical experts:

If the micro team does not already have a technical advisor, means of getting access to one from outside should be available.

VI. Technical design of solution:

Ways to deal with the constraints imposed by the knowledge of a single developer need to be found in order to enable better informed decisions on technical designs.

VII. Style of development process:

Again, since it is not always known whether more power lies with the customer or the development team, ways to deal with both situations should be come up with beforehand to minimize unanticipated situations.

VIII. Ease knowledge transfer process:

Knowledge transfer, one of the hardest problems in micro team environments, should be addressed.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

This thesis gives us some important insights on what things are typical of micro team projects. It provides a definition of a micro team. The case study approach proved to be a good way to get insights in software development processes. It showed that micro team approaches are a mix of new, agile and non-agile practices. These observations can be useful for micro teams to develop their own process for developing projects in the absence of a complete framework. Finally, these observations can act as a base to build upon a separate complete software development framework for micro teams with single developers.

Future work can include studying the following issues:

- Develop a software development methodology for micro teams
- Study the impact on the micro team development process when the team transitions from being micro (with a single developer) to a bigger team (more than one developer)
- Study how the development environment (IDEs) might change to assist micro teams – for instance, include an automatic agent reviewing the code in the background, or automatic refactoring tools etc.

- How the development process is affected when there is collaboration between two or more micro teams (i.e. distributed development teams) working on a common project – observed in crowd sourcing, open source projects.

REFERENCES

1. Pressman, Roger, “Software Engineering: A Practitioner’s Approach”, McGraw Hill
2. Dijkstra, Edsger, “The Humble Programmer”, 1972
3. http://en.wikipedia.org/wiki/History_of_software_engineering
4. Royce, Winston, "Current Problems," in Aerospace Software Engineering: A Collection of Concepts, edited by Christine Anderson and Merlin Dorfman, American Institute of Aeronautics, Inc., Washington DC, 1991.
5. Brooks, Fred, “No Silver Bullet – Essence and Accidents of Software Engineering”, 1986
6. M. Fowler, “The New Methodology,” http://www.martinfowler.com/articles/newMethodology.html
7. B. Boehm, “A Spiral Model of Software Development and Enhancement,” IEEE Computer, May 1998
8. K. Beck, Embracing change with Extreme Programming. IEEE Computer, Vol. 32, Issue 10 October 1999.
9. L. Rising and N. S. Janoff, The Scrum software development process for small teams, IEEE Software, Issue 17, pp. 26-32 , 2000
10. M. Awad, “A Comparison between Agile and Traditional Software Development Methodologies”, Honors program thesis, University of Western Australia, 2005

11. Rodriguez-Martinez, L.C.; Mora, M.; Alvarez, F.J., A Descriptive/Comparative Study of the Evolution of Process Models of Software Development Life Cycles (PM-SDLCs), Computer Science (ENC), 2009 Mexican International Conference on, 10.1109/ENC.2009.45, 2009
12. Larman, C., Basili, V., "Iterative and Incremental Development: A Brief History", June 2003, IEEE Computer Society, 0018-9162/03
13. Gao, Y., "Research on the Rule of Evolution of Software Development Process Model", Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on, 10.1109/ICIME.2010.5477884
14. Jayaswal, B. K., Peter C. Patton, P. C., "Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software", Prentice Hall, August 31, 2006, 0-13-187250-8
15. Suganya, G., Sahaya, S. A., Mary, Arul, "Progression towards Agility: A Comprehensive Survey", 2010 Second International conference on Computing, Communication and Networking Technologies, 978-1-4244-6589-7/10
16. Royce, Winston (1970), "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON 26 (August): 1–9
17. Riehle, Dirk, "A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn from Each Other", Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2000)

18. Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani, "Agile software development methods: Review and analysis", VTT Publications, 1455-0849 (http://www.inf.vtt.fi/pdf/)
19. Highsmith, Jim, "Agile Software Development Ecosystems", Addison-Wesley Professional, March 26, 2002, ISBN-10: 0-201-76043-6
20. Agarwal, R., Umphress, D., "Extreme Programming for a Single Person Team", ACM-SE '08, March 28-29, 2008, ISBN 978-1-60558-105-7/08/03
21. Kruchten, P., "Rational Unified Process, The: An Introduction", Third Edition, Addison-Wesley Professional, December 10, 2003, 978-0-321-19770-2
22. Schwaber, K., Sutherland, J., "Scrum", February 2010
23. Naur, P., Randell, B., "Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE", Garmisch, Germany, 7th to 11th October 1968
24. "The Software Crisis", http://media.wiley.com/product_data/excerpt/94/08186760/0818676094.pdf
25. Putnam, D., "Team Size Can Be the Key to a Successful Software Project", http://www.qsm.com/process_01.html
26. Ambler, S., http://www.ambysoft.com/surveys/practices2009.html
27. Heires, J. T., "What I did Last Summer: A Software Development Benchmarking Case Study", IEEE Software, September/October 2001, 0740-7459/01
28. Putnam, D., "Software Size Estimation Challenges", http://www.psmsc.com/UG2004/Presentations/PutnamDoug_PSMSizeestimationbriefing.pdf

29. Yin, R. K., “Case Study Research: Design and Methods”, 1994, Second Edition, Thousand Oaks: Sage
30. FLOSSmole Project (2004-2010), http://flossmole.org
31. Brooks, F., “The Mythical Man-Month”, Addison-Wesley, ISBN 0-201-00650-2
32. http://xprogramming.com/classics/natural/
33. Opelt, K., "Overcoming Brooks' Law," agile, pp.208-211, Agile 2008, 2008
34. http://www.objectpartners.com/2010/01/21/agile-project-inception-success-and-pitfalls/
35. http://www.the-software-tester.com/Models/software_waterfall_model.html

APPENDIX

A.1 Responses to Questionnaire 1:

A.1.1 Questionnaire_1_Participant_1:

Q. Can you give an overview of the project?

Ans. The project is a research project based upon a set of questions which are to be answered by a user who has had a heart attack in the past and underwent treatment for it. A report is generated in CSV format for about 147 answers given by every user and it is used by the researcher in his research. We have a website where a user will create his own login and answer about 147 questions.

Q. Can you describe -

Who were the persons involved in the project, and in what capacity?

Ans. The project is developed by the researcher, who is a Research Scientist. He is the main customer responsible for supervising the work and interacts with us. From our micro team, we have a supervisor of the project and who is responsible for delivering the project with customer requirements. There is a head liaison for this project and deals with customer regarding administrative purposes. I worked as developer on this project and had complete interaction with client.

Q. How did the project start and end?

Ans. The project is basically a website developed using CakePHP framework. This was developed by a previous developer before this project was handed over to our micro team. When the project was handed over to us, the website was not fully functional. Basically, the output generated was not in sync with the user answers. So I had to start with correcting minor faults which led to creating a csv report which had exact user answers as specified in codebook.

Q. What kind of work was done in it?

Ans. Work done by us was bug fixing kind of work. Following were the important fixes:

- Validation of textbox. ie. it should take only numeric input from user.
- Jump problem in Q19 was solved. Even if user checks options 1 and rest, there is no jump from Q19 to Q34.
- Validation of "other" textbox in check box type questions. ie. if it is left blank.
- Validation of "others" when the question is not answered.
- Show integer output in csv report for checkbox type question.
- Input answer into database for dropdown type questions.
- Input the answers of dropdown, time, day of week etc in the csv report.
- Validation in a check box type question, when the other is not checked.
- For Q26., added [\:] in preg_replace() while generation csv output.
- For radio type questions, encoding changed to "1=yes and 0=no" for all questions which have yes/no as options.
- Corrected few reportoptions in questions table, as few entries were not according to codebook.

- In Q56, item checked in Q41 appears as part of question. So if we leave Q41 unanswered(since it is not compulsory question) then we get blank spaces in Q56.
- Corrected the problem which happens after changing encoding of radio type questions. Problem was that even the compulsory questions could be skipped and some input was not recorded in the database.
- Administrator Login problem solved. After making few changes to Cake file and project file, the administrator login works fine.

Q. In the development life cycle of the project, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?

Ans. Requirements were given by the customer through emails and also we had a weekly status call where we discussed the work done in previous week and next week's deliverables. Also any change in requirements was conveyed to us by customer. The design of website was already setup. It used Cake PHP framework which followed MVC(Model View Controller) design pattern which we also followed. We used WAMP to host webpage locally during the development phase. All code changes were made in existing PHP files in project. The project was delivered on the exact deadline which was 10th Jan, where we hosted the website on a temporary server and asked the client to perform Beta testing. The client feedback was good and he was satisfied with the output generated.

Q. As a single developer, what challenges did you face in the course of the project? For instance, learning new technology, problems related to programming/deployment, interaction with the customer/project manager, etc. How did you resolve these problems?

Ans. First major task was to get acquainted with the CakePHP framework. It took quite a lot of time to study the framework so that whatever changes had to be made to the code should be in right place so that normal working of site shouldn't get hampered. Second problem was "login " issue where we were not able to login into website as administrator, which we had to so as to generate reports. This problem got solved by changing the code in CakePHP framework files and not any of the project file. As regards to customer interaction it was quite smooth sailing all through. Final problem which is yet unsolved is that we have not been able to login to Yale server in order to deploy our website. We are in process of getting the required permission.

Q. What was the duration of the project? How did you keep up the motivation to keep working? Were there any other non-technical challenges you faced? If yes, how did you overcome them?

Ans. The duration of project was approximately 2 months time. But major development work started in 2nd week of December and finished on 10th January 2011. Main motivation to work on this project was to get a industry kind of work exposure where you have customer deploying your project and you have deliverables and customer expects you to provide best quality solution. This was a learning curve which cannot be learnt in normal course project. There were no specific non technical challenges except one where

previous developer had left abruptly and it took time to communicate with him to understand what code changes he had made.

Q. Any other thoughts about the project and the process?

Ans. The project is excellent one for the researcher's work and provided some live project experience to me as a developer. The process of completing this project with the advisor and his experience and inputs contributed a lot in making of this project.

A.1.2 Questionnaire_1_Participant_2

Q. Can you give an overview of the project?

Ans. The primary aim of this study is to increase our understanding of care-seeking behavior surrounding acute coronary syndromes [ACS] through the application and testing of an integrated self-regulatory care-seeking model [ISCM] and the utilization of a world wide web [WWW] based ISCM survey instrument. At present, care-seeking delay among individuals stricken with ACS [acute myocardial infarction [AMI], unstable angina or sudden cardiac death] prevents many patients from obtaining the full therapeutic benefit of hospital based thrombolytic and/or mechanical reperfusion therapy to reduce the morbid and mortal consequences of acute cardiovascular disease [CVD]. Results from prior studies of ACS care-seeking delay generally rely on measures of time duration from acute symptom onset [ASO] to hospital emergency department [ED] arrival and measures of social demographic and clinical variables, extracted from emergency medical system [EMS] logs and/or ED charts. By relying on such limited

data sources, these studies have failed to take into consideration the complexity of the social and behavioral processes by which individuals make decisions to seek medical care for symptoms of ACS.

By applying and testing an integrated self-regulatory care-seeking model [ISCM] to ACS care-seeking, it will be possible to delineate decision points and situations and circumstances that are critical to producing patterns of care-seeking that are efficient and expeditious or are protracted and delayed. The care-seeking process in the ISCM is viewed in terms of five analytic care-seeking phases from warning or premonitory symptoms to ED arrival. By noting the presence or absence of three central phases, it is possible to distinguish patterns of care-seeking and to record the duration of each care-seeking phase and total duration of each care-seeking path to the ED, see Figure 1. The ISCM assumes that coping behaviors emerge over time and initially care-seeking behaviors are guided by demographic and structural factors. As the self-regulatory and coping processes emerge over the course of the ACS episode, the influence of these factors diminishes and emergent ACS evaluations, advice from others and health care providers, emerging symptoms and emotions, come to dominate the care-seeking process and directly determine the duration of ACS care-seeking. The study will test and explore the following hypotheses:

1] Commonly used demographic variables, such as, age, sex, education and race will prove to be only one component, along with social situational and circumstantial, cognitive and emotional variables, that significantly influence the course and duration of ACS care-seeking.

2] The ISCM will show that decision process variables associated with the care-seeking phases, the Self, Lay and Health Care Provider Phases, as shown in the Figure, are important predictors of time from symptoms onset to ED arrival and in the prediction of the length of time spent in each care-seeking phase.

3] The ISCM will show differences in the total duration of the 5 most frequently occurring care-seeking paths to the ED shown in the Figure, Care-Seeking Paths II, IV, V, VI and VII and will explain most of the differences by the variables in Hypotheses 1 and 2.

4] Conditional models of subjects who use key care-seeking phases, namely the Self, Lay and Health Care Provider Phases, will identify unique predictors of time in each phase.

5] Significant interaction effects [model coefficient differences] will be found for important risk variables and socially important gender, age and racial groups. For example, African-Americans will show differences in model coefficients from Non-African-Americans.

The proposed study utilizes a twice tested pilot ISCM survey instrument placed on two secure dedicated web servers to collect patient care-seeking data. The target sample size for this study is 2,314 patients who have experienced an ACS episode. Subjects will be recruited using: 1] extensive marketing of the study URL on websites of interest to cardiac patients and oriented toward cardiac health and ACS risks, such as, diabetes or hypertension; 2] placement of study information on discussion boards and chatrooms with a cardiac health focus; 3] emailing of a study flyer PDF for posting in public libraries, senior citizen centers, African-American churches, county and municipal health clinics, and HMO web sites; 4] application of a viral marketing strategy; and 5] use of

selected print media, for example, AARP publications. Marketing of the study site URL will emphasize the recruitment of women, the elderly and minorities. In the data collection phase of the study, the proportions of myocardial infarction and angina pectoris subjects falling into each cell of the cross classification of age, sex and race will be derived from NHANES 1999-2000 and 2001-2002 studies. These estimates of the population proportions in these cells will be used to determine quotas for each of the cells in an attempt to ensure all risk groups are adequately represented. We will over sample African-Americans. Statistical models will be used to estimate the effects of variables that characterize the decision process on the time delay until ED arrival. Time required to move from one phase of the decision process to the next will also be modeled, as will the different paths chosen by the subjects. Models will also include patient characteristics and other circumstantial and situational variables in the decision process.

Q. There were three explicit stages in the project life cycle: 1) as a Capstone project 2) with a previous developer 3) when this micro team took over the project. Can you describe -

Who were the persons involved in each of the stages, and in what capacity?

Ans. An OSU IT undergraduate, and a group of students from one of the professor's classes

Q. How did the stages start and end?

Ans. The IT undergraduate was in class where I was client for developing the website system design, he volunteered to continue working on project, professor's class did preliminary programming of website, undergrad and I received a summer to have him participate in summer program to complete work on the website, website was up and running as their university for pilot study, website was brought to my workplace after funding was received from NIH, and then returned to their university for final complete of website and launching of study.

Q. What work was done in each?

Ans. I do not know exactly what was done at each stage in a technical sense. All I know is that at each stage the website has become more functional.

Q. Were there any other stages before these too? If yes, can you describe them?

Ans. No other stages

Q. How were the stages different from each other?

Ans. Each stage added some incremental functionality to the website from system design to functional website. Again, technically I do not what it was or what one calls the items that were added. Even the previous developer added functionality.

Q. In each of the stages, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?

Ans. You may not be asking the right person here as I do not know in a technical sense. I do not really know what "requirements collected" are. Unless you mean what did the

website need to do. In the beginning in the system design class I described what I wanted to do and the students described what they proposed and I evaluated their proposals to make sure they system could do what I needed. Then it was a matter of building system, consulting with students or professor on what was needed, and what they had done and seeing if work met needs and whether needs could be built and the compromises necessary to get to the completed website.

Q. Any other thoughts about the project and the process?

Ans. I probably should have returned to project back to university sooner than I did and even hosted the website there rather than my workplace.

A.1.3 Questionnaire_1_Participant_3

Q. Can you give an overview of the project?

Ans. The Complex Flow Analysis project was the name given to a methodology being developed by the research center. The center was building a practice in which they coordinated with various companies to build up by-product synergy networks (basically, groups of companies willing to work with one another to redirect their waste streams by giving or taking waste from each other and using it as an alternative fuel, recycled materials, etc.). The goal of these networks was to both reduce costs and their environmental footprint. At the time I began working with them, they were primarily working with a group of companies that had already agreed to work together (in Kansas) and were trying to establish and similar network of companies locally in Ohio. In addition to getting the companies connected to one another, the center acted as a mediator that could determine how to redirect the waste streams and make proposals based on their

calculations (if a single company did this they might want to do things that were in their best interest rather than in the best interests of all the companies). Getting all the data from each company, doing the mathematical optimizations, communicating proposals (cost/benefit of changes to each company), took a lot of work so the Complex Flow Analysis software was intended to be a tool that would make it easier.

The mathematical analysis that the center did as part of their analysis was often difficult to communicate to clients and they used custom spreadsheets and drawings to share data with their clients and communicate the result. The software was designed to allow clients to be more engaged in the modeling and optimization part of the process without having to know all the details. They also hoped it would allow them to standardize data collection & share results more easily without the needs for custom spreadsheet, etc. I developed a software package based on Eclipse RCP that could organize "projects" and associated data (models, solutions, etc.). A large part of it was a GUI that allowed networks to be created, saved, updated, viewed, etc. via drag/drop operations. Material flows in these networks could be optimized (by a user definable set of criteria) with a single click. This also helped avoid the possibility of errors because we found that even when expert graduate students tried to formulate these networks by hand they often made errors.

Complex Flow Analysis was eventually linked with another tool, that I developed at the same time with this group.

Q. Can you describe -

Q1. Who were the persons involved in the project, and in what capacity?

Ans. Me – sole developer

2 grad. Students @ research center – They worked with the clients to model the by-product synergy networks, did the data analysis, collected data, etc. They were also the primary users of the software. The clients used it as well, but the grad. Students were usually there to answer questions, help them do the modeling in the tools, etc.

2 directors @ research center – Also worked with clients and set the overall vision for the group.

Q2. How did the project start and end?

Ans. It started when my professor asked me to do it as a source of funding and ran approximately 2 years. However, I did not work on it exclusively during that period. I also did other tools and integration, website, and some other small side projects.

Q3. What kind of work was done in it?

Ans. I developed the software and participated in the groups meetings as often as I could.

In the development life cycle of the project, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?

For the majority of the project the requirements were handled much like a top 10 list. We had group meetings were we (2 grad students + 2 directors + me) would

talk about what needed to be implemented next. I would maintain this list of the top requirements and share it. I also kept a private backlog, which I occasionally used to start up discussions. I would typically demo the software to the group in later meters for feedback/acceptance. I had very little interaction with the center's clients that used the tool. Although the grad students were really the "power" users of the tool (because it helped them do their work, which was much more intensive than anything the clients did with the tool), so I got lots of feedback from them during these meetings. We rarely had deadlines, although I would provide rough estimates. For the one component that we built (the solver part), I followed a slightly different approach. I had one of the grad students write (1) mathematical formulations of what the software was supposed to do and (2) create test cases which covered this formulation. I used these implement & test the solver component. In addition, the grad students would occasionally check the results Complex Flow Analysis generated against solutions worked out in other software (it was important that we did not give inaccurate results to clients). I implemented it at home by myself – I usually worked in full day blocks – so I did not mix it with other work and worked on Complex Flow Analysis one or two days a week.

Q. As a single developer, what challenges did you face in the course of the project? For instance, learning new technology, problems related to programming/deployment, interaction with the customer/project manager, etc. How did you resolve these problems?

Ans. It was difficult to reach some of the users because of their distance (Kansas) and because I had limited time. Weekly group meetings + a few development hours / week

didn't leave much more time to get feedback from everyone that I would have liked to. The scope of this project was sort of large (i.e. they wanted a lot built, had tons of ideas, and were aware it would take awhile). This was a concern at first, but as we got in the habit of weekly meetings w/ visible progress each week and maintaining our top 10 list we were able to estimate progress and set expectations reasonably well.

Q. What was the duration of the project? How did you keep up the motivation to keep working? Were there any other non-technical challenges you faced? If yes, how did you overcome them?

Ans. Roughly 2 years. It paid the bills and the problem domain was pretty interesting. It was sometimes difficult to balance my own graduate schoolwork with this. Ironically, oftentimes the problem was to force myself to stop working on Complex Flow Analysis (which was relatively enjoyable, visible and incremental progress, etc.) and do my own research related things (which while usually enjoyable, did not have the clear milestones, easily visible progress, etc.)

Q. Any other thoughts about the project and the process?

Ans. My process was pretty organic. I would say it's largely a result of how the people at the center for resilience wanted to work. I basically tried to fit what I needed to do into their weekly meetings, which included discussion of much more than just the software. The better I understood them, the better I could integrate. I avoided asking them to do very much "development" stuff, like looking over use cases, etc – which was how a previous developer had approached the project. Of course, I got the same information I just didn't use those technical terms and put it in a form that was more natural to them.

A.1.4 Questionnaire_1_Participant_4

Q. Can you give an overview of the project?

Ans. The project is cloud service that provides data storage in the cloud that is fed to and from a wireless sensor network.

Q. Can you describe -

Q. Who were the persons involved in the project, and in what capacity?

Ans. Previous developer - worked on the design of what cloud server to use, SpringMVC framework for development and provided the foundation for further work.

Customer 1- Guidance on design, implementation and project management.

Customer 2- Help on security aspect of the cloud service.

2 grad students – Help on building code regarding secure communication on embedded systems.

1 employee – Worked on the infrastructure required for development such as SVN access, email access. He also helped in design decisions for data storage aspect of the project.

Q. How did the project start and end?

Ans. It is an ongoing project and not sure on how it started.

Q. What kind of work was done in it?

Ans. When I began, I continued with where previous developer stopped. There was Spring MVC framework implemented and deployed on the Google App Engine and a demo code was provided with communication of the server and the client.

Q. In the development life cycle of the project, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?

Ans. Requirements were collected by thinking of various applications that can use Cloud Sensor and how and what type of data could be stored. For each application and scenario, the users involved were thought of and how communication amongst the various users can be made secure. Design decisions were made by the most feasible and best implementation that can be provided.

Implementation – Code was regularly checked in SVN when any changes were made. A sample code for a similar shorter example would be developed for cases when implementation is not sure of or when working on new technology and then carried over to Cloud Sensor.

Q. As a single developer, what challenges did you face in the course of the project? For instance, learning new technology, problems related to programming/deployment, interaction with the customer/project manager, etc. How did you resolve these problems?

Ans. Communication with project manager in terms of meeting timings and conflict with schedule. This was resolved by working on a fixed schedule for meeting and talking with team members whenever possible online.

Q. What was the duration of the project? How did you keep up the motivation to keep working? Were there any other non-technical challenges you faced? If yes, how did you overcome them?

Ans. It is an ingoing project. The data storage part of the project took a quarter to complete. The motivation was kept by talking to team members on the project I am working on and having regular status update meetings. It took quite some time to understand the project and its requirement as well as the technologies being used.

Q. Any other thoughts about the project and the process?

Ans. No.

A.1.5 Questionnaire_1_Participant_5

Q. Can you give an overview of the project?

Ans. The project is the development of a survey system for cardiac patents. The system was written by another company using CakePHP. Our job was to verify the system and make the changes as specified by the sponsor.

Q. Can you describe -

Q. Who were the persons involved in the project, and in what capacity?

Ans. Primary technical person.

Minor technical and supervisor.

Technical advisor and project manager.

Q. How did the project start and end?

Ans. The project was languishing and I got called in on it to make certain that it got adequate attention. How it originally started I do not know.

Q. What kind of work was done in it?

Ans. The system was reviewed to make certain that the input and output matched the code book. The output was examined to insure it was accurate.

Q. In the development life cycle of the project, can you describe what process was followed? For instance, how were requirements collected, how did the design come up, how did implementation take place, etc?

Ans. The system requirements, design and implementation were completed when we received the project. Our job was to verify, support and troubleshoot.

Q. As a technical advisor, what challenges did you face in the course of the project? How did you resolve these problems?

Ans.

- When we received the project is was behind and had been neglected. The customer felt that we were neglecting it. We communicated with the customer what we were doing and gave him a realist commitment for completion as well as level of effort.
- Keeping the project scope to the committed tasks. When people tried to add tasks, I would remind them that we wanted to first complete the stuff we committed to.

- Lack of familiarity with the frame work. Spent a lot of time reading the documentation and researching the framework.

Q. Were there any other non-technical challenges you faced? If yes, how did you overcome them?

Ans. No.

Q. Any other thoughts about the project and the process?

Ans. -

A.2 Responses to Questionnaire 2:

A.2.1 Questionnaire_2_Participant_1

Q. How did you come up with the development process? Was the process similar to development processes of other projects you worked on? Or was it shaped to suit the client's style of working? If the latter, can you give an example of the client's style of work and how it shaped your process?

Ans. The development process for project was pretty similar to ones which I had worked on which involved some customer interaction and weekly iterations.

Here we had a weekly developer's call with the client where we basically update the client about the progress made in past week and our plan of action for coming week. We also tried to get customer consent on the work completed in previous week. Any changes in requirements by customer were also taken down in call. Any other communication was through mails, where in we updated the client about our status and he replied back explaining some requirement in detail which would have been discussed in weekly call.

After the development work was done, we uploaded the website on our server to perform alpha testing and also asked the customer to check the website himself and give his consent on the work done.

Q. While developing different features/components in the project, for the development of each component was the same process used? Or did the implicit nature of a component/feature warrant a different kind of process? E.g., one component might have

been suited best for a Test Driven Development approach, so you created the test cases before implementing, while another followed the traditional approach.

Ans. My project was basically a website developed in CakePHP framework. There were different modules which were divided according to MVC architecture of website. Each module had a different functionality but the development process for each module was similar. I followed a traditional approach while developing the modules ie. coding the module and unit testing it each time.

Q. Did you have to learn new technology for implementing the project? If yes, how much time did it take? What did you do when you were stuck on something? For e.g., did you go to someone who was an expert, or used any other resource?

Ans. I had to understand the CakePHP framework which was used to develop the website. This is the only thing which I needed to learn along with my mentor. It took my approximately a week to learn the new technology.

But this learning was not sufficient as for some subsequent bug fixes I had to refer back to documentation of the CakePHP framework. When I was stuck at a "login" problem I worked with my supervisor on the bug and fixed it with his help by making some change to original cake php files.

Q. When the project was passed on to you, did you have any difficulties in the knowledge transfer process? Was the earlier developer available easily to clarify doubts?

Ans. When the project was passed on to me, the previous developer had left the project abruptly and without making a document of whatever changes he had made to the code.

It was difficult at beginning trying to understand the changes made and work upon them to fix bugs.

The developer was not so easily available to clarify doubts, but eventually I had a chat with him and did some amount of knowledge transfer. But it was not sufficient amount as the previous developer had not worked sufficient amount of time on project.

Q. Was the learning curve and development process shortened due to the presence of a technical expert in your team?

Ans. The learning curve and development process in project dependent upon understanding CakePHP framework and incorporate changes accordingly.

The learning curve was not so much shortened due to presence of a technical expert in my team, as the expert himself was new to this framework. It took time to understand the MVC architecture and make changes to appropriate pages.

A.2.2 Questionnaire_2_Participant_3

Q. Did you have to learn new technology for implementing the project? If yes, how much time did it take? What did you do when you were stuck on something? For e.g., did you go to someone who was an expert, or used any other resource? If you did go to an expert, was the learning curve and development process shortened due to the presence of this expert?

Ans. I had to learn the basics of Eclipse RCP (how to build & package an Eclipse based product, plug-in architecture, etc.) and a number of key Eclipse Plug-ins (GEF, JFaces, ect.). Also had to select and learn a few other COTS packages (I can't remember what

they are anymore – one was for XML serialization, another for encryption, etc.) It's hard to say how much time I spent learning anything because I never devoted any time to learning (except maybe a week at the beginning when I did some research on Eclipse RCP and the GEF framework to see if it was a good fit for the project). I usually just read the manuals, examples, etc. as I developed and reworked things if I initially messed up. I never went to anyone for help, except for Google. Many of the major frameworks (Eclipse RCP, GEF) had dedicated mailings lists and developer forums. I rarely actively participated (asked questions myself, etc) but I did often go to them first instead of heading straight to Google.

Q. Can you specify how many and what kind of work products were created in the entire process? You could list all the work products along with their contents. (Note - if it is possible to provide any of the work products, it will help me a lot; if not, a detailed description of what each one was for and its contents will also work.)

Ans. Project Wiki (username: ***, password: ***) Wiki URL: ***

We mostly just used the "Builds" Page. I also used the tickets for my own purposes (backlog, bugs, etc.)

I also got what were sort of specifications and tests cases from one of the grad students on the team. I'll forward you one example, I can find more if you'd like. They typically contained a high level description of what functionality the tool needed to support, or examples of input/output, and/or wireframe UI's to show what the data requirements were. These were rarely ever 100% correct – we typically used them as a starting point for discussion.

A.2.1 Questionnaire_2_Participant_4

Q. What did your development process look like? Were there clearly separable phases like a requirements collection phase, analysis, design, implementation phases, etc? If not, what was the process? Was a waterfall model followed (with all requirements known at the beginning)? If not, how did you manage changing requirements?

Ans. The process was not structured. The requirements would change in phases wherein the requirements of one phase would depend on the previous phase. For example, when implementing security on a web API, we would go ahead with an initial basic algorithm and then develop it further on based on implementations present or those that can be developed in a short period of time.

Q. How did you come up with the development process? Was the process similar to development processes of other projects you worked on? Or was it shaped to suit the client's style of working? If the latter, can you give an example of the client's style of work and how it shaped your process?

Ans. The project is not based on any particular client. The development process Is different from other projects I have worked on as the designing is done by the developer itself .

Q. While developing different features/components in the project, for the development of each component was the same process used? Or did the implicit nature of a component/feature warrant a different kind of process? E.g., one component might have been suited best for a Test Driven Development approach, so you created the test cases before implementing, while another followed the traditional approach.

Ans. Different components had different processes used as data storage component depended on creating test cases and deciding what kind of data and how data is stored in the server. The security component depends on technologies available and what basic implementation can be done after which the component will be tested.

Q. Did you have to learn new technology for implementing the project? If yes, how much time did it take? What did you do when you were stuck on something? For e.g., did you go to someone who was an expert, or used any other resource? If you went to an expert, would you say that the learning curve and development process was shortened due to the presence of this technical expert in your team?

Ans. Yes. I was initially stuck in the setup of the system. I took help of the previous developer incharge as well as knowledge sessions and documents created. The learning curve was shortened due to the expert's input. It took a couple of weeks for the setup to be done. It took around a month to understand the security aspect of the project and due to which understanding the requirements was delayed.

Q. When the project was passed on to you, did you have any difficulties in the knowledge transfer process? Was the earlier developer available easily to clarify doubts?

Ans. No difficulties involved. The earlier developer was available easily to clarify doubts.

Q. Did you create any kind of a document to record your progress? For example, say, a Word file that would contain all the requirements, bugs, completed tasks, etc?

Ans. Progress was recorded by regular status updates to management. Documents were created and at times minutes were recorded.

Q. What kind of collaborative workspace did you and your customers and/or team members use to communicate?

Ans. We used MSN Messenger for communication and screen sharing for demos. Conference calls were setup on a regular basis. SVN was used to share code.

Q. Can you specify how many and what kind of work products were created in the entire process? For instance, the source code would be one; the requirements document another, and so on. You can list any document that was created along with what its contents were in the process. (Note - if it is possible for you to provide the document, it will help me a lot; if not, a detailed description of what the document was for and its contents will also work.)

Ans. Source code in SVN, Requirements document, design document. Design document would have various versions based on updates from management.

Q. How frequently do you have meetings with the customer? Is this frequency sufficient, or do you need more frequent or less frequent meetings? When certain issues are discussed in a meeting, are they addressed completely during the time leading up to the next meeting? Or is this period too short to address the issues or too long such that you have free time on your hands?

Ans. Meetings with management are 3 times a week. Frequency is sufficient. If any issue is not addressed in time specified , further meetings are arranged.

A.2.2 Questionnaire_2_Participant_5

Q. How did you come up with the development process? Was the process similar to development processes of other projects you worked on? Or was it shaped to suit the client's style of working? If the latter, can you give an example of the client's style of work and how it shaped your process?

Ans.

The process was adhoc. I was brought in because nothing was happening on the project. My job, as I saw it, was to find out why nothing was happening and to facilitate successful completion of the project. It became apparent that the project did not have a clear set of tasks. So I made that the first task. Second, I forced the programmer to commit to the effort. Third, when the customer showed signs of unhappiness with our effort, I pushed for a due date that would meet his needs, yet be realistic given the programmer's class work load.

Yes I have used this process before in similar circumstance. Throughout my career, I have been brought in to troubleshoot a project in trouble and to help organize it into a successful process.

The client was not skilled technically in software development. He did not have a "style of working" that he presented to us. He simply wanted a successful project and he did not care how it got accomplished. So maybe that was his style of working?

Q. While developing different features/components in the project, for the development of each component was the same process used? Or did the implicit nature of a

component/feature warrant a different kind of process? E.g., one component might have been suited best for a Test Driven Development approach, so you created the test cases before implementing, while another followed the traditional approach.

Ans.

The project came to us with the client being suspicious of the software. He listed several areas that he felt needed to be confirmed that they worked correctly and that the output was correct. So the project was test driven. The customer had already specified a rough set of test cases that guided our testing scenarios. So yes the same process was used throughout the project.

Q. When the project was passed on to you, did you have any difficulties in the knowledge transfer process? Was the earlier developer available easily to clarify doubts?

Ans.

Yes we had difficulties in the knowledge transfer process. The original developer was available by email and responded to the first email. Subsequent emails he was less helpful. Possibly this was because he did not know or remember the answer and was not motivated to find it.

Q. Can you specify how many and what kind of work products were created in the entire process? For instance, the source code would be one, the Trac page would be another, and so on. You can list any document that was created along with what its contents were in the process. (Note - if it is possible for you to provide the document, it will help me a

lot; if not, a detailed description of what the document was for and its contents will also work.)

Ans.

The main work product was to be a “How To’s” manual for the developers. I have not seen it, so I do not know if it was completed. The client was only interested in one work product, the web site. The client had initially created a booklet detailing the content of the survey and the next question to be presented given a particular answer.

The “How to’s” programmer’s manual was to list how to enter new questions and what the special symbolic coding meant. Basically it was what we had to learn to test and modify this software so in the future it can be handed off successfully to the next person.

A.3 Responses to Questionnaire 3:

A.3.1 Questionnaire_3_Participant_1

1. How essential was the project to the customer's business process? Rate on a scale of 1 to 5, with 1 being not very essential and 5 being extremely essential (the business could not proceed without the success of the project)

Ans: 4 (this is relating to his research work and hence very important for customer)

2. Did that affect how you approached the project? For e.g., did you stress more on getting requirements right at the cost of time spent, or spend more time getting the design perfect, or just jumped into implementation to show results? (1-2 lines is sufficient)

Ans: Got the requirements right at cost of time spent and just jumped onto implementing them immediately. The customer expected quick results.

3. This is a three-part question to get an approximate measure of software size:

a. What were the approximate number of features that needed to be fixed/verified in the project?

b. What were the approximate number of components that were implemented/touched in the project?

c. What is the average number of lines of code written per component?

Ans: a. 3

b. 4-5

c. 1000

A.3.1 Questionnaire_3_Participant_3

1. How essential was the project to the customer's business process? Rate on a scale of 1 to 5, with 1 being not very essential and 5 being extremely essential (the business could not proceed without the success of the project)

Ans. Probably around a 2. The software really did two things - help streamline their internal process (but they could do it w/ out) - and to improve their clients experiences, but again with more manual effort they could do this w/ out the tool as well. It would probably be higher if they had more concurrent clients (i.e. then the manual effort would have been too much), but given their workload at the time I'd say it was about a 2. Although project was also sort of an advertising piece (i.e. hey look we have this neat tool, etc.), but still probably not essential.

2. Did that affect how you approached the project? For e.g., did you stress more on getting requirements right at the cost of time spent, or spend more time getting the design perfect, or just jumped into implementation to show results? (1-2 lines is sufficient)

Ans. Well, project was viewed by the sponsors as a long term project to meet the needs of the group. Those needs were not so clear so spending a lot of time on requirements didn't make much sense (i.e. they were trying to engage more clients, each client wanted to do things slightly different, etc.). Since they could engage their clients without the tool (i.e. question 1) we didn't mind trying things out, experimenting, and seeing how they worked out. Throwing things away was acceptable (I threw plenty of big features away after implementing them, or used them as a first iteration and completely redesigned them).

3. This is a three-part question to get an approximate measure of software size:

a. What were the approximate number of features that needed to be implemented in the project?

b. What were the approximate number of components that were implemented in the project?

c. What is the average number of lines of code written per component?

Ans. I honestly have no idea how to answer this, other than just randomly guessing. Sorry! First, it was a long time ago so I don't really recall much of the low level details. I'd have to go through the code to figure this out and it would take a long time (I can give you access to it if you want to work it out). Second, even if I did "feature" and "component" are pretty subjective and open for interpretation.

A.3.1 Questionnaire_3_Participant_4

1. How essential was the project to the customer's business process? Rate on a scale of 1 to 5, with 1 being not very essential and 5 being extremely essential (the business could not proceed without the success of the project)
4

2. Did that affect how you approached the project? For e.g., did you stress more on getting requirements right at the cost of time spent, or spend more time getting the design perfect, or just jumped into implementation to show results? (1-2 lines is sufficient)
Yes. There was more stress on getting the requirements right and the design process as

well. The product to the customer is a framework to use for their software development.

3. This is a three-part question to get an approximate measure of software size:
a. What were the approximate number of features that needed to be implemented in the project? I think 4-5.
b. What were the approximate number of components that were implemented in the project? 2
c. What is the average number of lines of code written per component? 3000 - 5000 lines of code.

A.3.1 Questionnaire_3_Participant_5

1. How essential was the project to the customer's business process? Rate on a scale of 1 to 5, with 1 being not very essential and 5 being extremely essential (the business could not proceed without the success of the project)

Ans. Well I don't know how to quantify this for this project. It was a research project that revolved around this software. Without the software the research could not be completed. in that regard it was a 5.

However, he has other projects going on probably and other duties. So he could shift his focus to those if this project fell through and still work. In that regard it would be a 1.

Plus, the "business" of Yale Medical is huge compared to this one little project. So in that regard it is a 1.

2. Did that affect how you approached the project? For e.g., did you stress more on getting requirements right at the cost of time spent, or spend more time getting the design perfect, or just jumped into implementation to show results? (1-2 lines is sufficient)

Ans. Given that we were batting cleanup (others began and "finished" the project and we were brought in to fix the problems and unimplemented stuff) we did not spend anytime on requirements as such. We did spend time talking to the customer to verify our understanding of the specifications and requirements others had compiled.

3. This is a three-part question to get an approximate measure of software size:

a. What were the approximate number of features that needed to be fixed/verified in the project?

Ans. About a half a dozen need to be tested. And three or four needed to be fixed including two that the customer did not know about.

b. What were the approximate number of components that were implemented/touched in the project?

Ans. Two components. Most of the fixes were made by using the administrator interface to correct the setup data.

c. What is the average number of lines of code written per component?

Ans. Four lines of code to fix the problem. Maybe 40 to develop tools to identify the problems.