

Software Development: Cowboy or Samurai

Charmayne Cullom

University of Northern Colorado, Monfort College of Business, Greeley, CO 80639

Richard Cullom

Ridgetop Information Solutions

ABSTRACT

Much has been written about the failure of software development projects. The importance of good software to business cannot be underestimated. A major issue is quality of the software being written and produced. Over the years, the emphasis for improvement in software development has been to change the process. Unified Modeling Language is but one of a long line of changes in software development process starting with structured programming introduced in the seventies to object oriented programming in nineties. However, several software development experts have begun to assert that the problem may not be with the process employed but rather the human asset, namely the software developer. This paper discusses two extremes of software developer behaviors. These two ends of the spectrum are the cowboy, free of restrictions, and the Samurai, following a strict code of behavior. The paper first presents the characteristics of the cowboy and the Samurai then the paper compares the development environment within which each behavior is suitable. The paper concludes with a summary and industry observations.

INTRODUCTION

Much has been written about the failure of software development projects. In 2004, the Standish Group reported that an estimated 30 percent of all software projects are cancelled and that 60 percent of software projects are categorized as 'failures' by those initiating the projects.(The Economist, 2004) Moreover, on March 21, 2006, Microsoft announced that next upgrade to Windows, code name Vista, would be delayed until 2007. A new version of Windows was first expected to be delivered in 2005. The delayed release of a major development project containing an estimated 50 million lines of code illustrates that the software development industry is in trouble.(Hamm, 2006; Booker, 2006, Information Week, 2006; Kerstetter, 2003)

The importance of good software to business cannot be underestimated. The National Institute of Standards (NIST) in 2002 estimated that software errors cost the American economy over 59 billion dollars annually.(The Economist, 2004) Software is forever. One only has to look at the Y2K syndrome when businesses were concerned that software flaws for the date change would create major disruptions in business operations to understand the importance of software to business operations and continuity.

A major issue is quality of the software being written and produced. Jim Kerstetter (2003) writes "There's a troubling analogy to be made here to the fall of the American auto industry in the 1970s." Kerstetter further notes that quality assurance experts attempted to get the American auto industry to embrace rigid quality standards. While the American auto industry largely ignored the quality experts, Japan listened and built a powerhouse auto industry. However, while Bangalore, India appears to have embraced quality in software development, an industry wide survey of the software development engineers and programmers conducted by AccuRev in 2005 noted 74 percent of the respondents were concerned about software quality. (Business Wire, 2005) Thus, software development must do much more than merely embrace quality standards and processes if it is to revive customer support. Declining sales of ERP software (Kerstetter, 2003) as well as the typical network administrator mantra of not installing new software versions until at least service pack 1 is available reveal the need for software development to become a quality oriented endeavor.

Over the years, the emphasis for improvement in software development has been to change process. Unified Modeling Language is but one of a long line of changes in software development process starting with structured programming introduced in the seventies to object oriented programming in nineties. Steve McConnell (2004) discusses the prevalence of software development companies to seek 'silver bullets' to employ seeking enhanced productivity. He especially points out the alphabet soup of buzzwords and acronyms that prevail in the industry.

When a project gets behind schedule, the tendency is to look for the breakthrough methodology promising unbelievable gains in productivity.

However, several software development experts have begun to assert that the problem may not be with the process employed but rather the human asset, namely the software developer. (McConnell, 2004; VNU Computing, 2005) McConnell further notes that in the long run it is the type of programmer deployed for the project that has the ability to make a difference in the quality of the product. Cleveland Gibbon, technical director at Acknowledge Technologies, notes three things he deems essential to building good software. The number one item on his list is 'good people'. (VNU Computing, 2005) Ewusi-Mensah (1997) noted that one of the major critical issues in abandoned information systems development projects was a poor or inadequate project team. The Agile Manifesto clearly defines a reliance on the people involved in software development. Over ten years ago, James Bach (1995) wrote "Process is useful, but it is not central to successful software projects."

Several authors including Bach, Kling, Harper-Wood, and Wood have discussed the categorization and socialization of developers within information systems projects. Bach (1995) described the need for heroes while Harper-Wood and Wood (2005) build on a premise of roles. Harper-Wood and Wood build behaviors upon the opposites of objective versus subjective and structuralist versus humanist. In this regard, they provide a spectrum of behavior for negotiating change and structure with classifications of teacher, doctor, warrior, and emancipator. Briefly, the doctor is a technical expert with the teacher serving as a facilitator on the objective – subjective spectrum. The structuralist is depicted as a warrior being an agent for social progress and the emancipator being described in terms of change catalyst.

While each of these classifications is useful for the broad scope of project team behaviors, this paper is concerned with a strategy or behavior within the theme of the Agile Manifesto. The Agile Manifesto expresses a deeper theme than just software production moving into the concept of building a software community of "people who shared compatible goals and values based on mutual trust and respect, promoting collaborative, people-focused organizational models." The Agile Manifesto refines the human element in software development by "Build projects around motivated individuals, give them the environment and support they need, and trust them to get the job done." James Bach (1995) describes the situation directly, making the statement "The central issue is the human processor – the hero who steps up and solves the problems that lie between a need expressed and a need fulfilled." This paper discusses two extremes of software developer behaviors. These two ends of the spectrum are the cowboy, free of restrictions, and the Samurai, following a strict code of behavior. The paper first presents the characteristics of the cowboy and the Samurai then the paper compares the development environment within which each behavior is suitable. The paper concludes with a summary and industry observations.

THE COWBOY

The "cowboy programmer" is a metaphor for software development's rugged individual, the project hero who single-handedly saves the business from an embarrassing and very public software development failure. Steve McConnell (2004) describes the cowboy programming model in different terms. He refers to them as the commitment-oriented development with names like "hero-oriented development" and "individual empowerment" model. Bach (1995) on the other hand refers to the cowboy as pathological although not necessarily on purpose. But, regardless, the cowboy behavior is not sustainable over a long period of time either by the individual or the project team.

History of the Cowboy Programmer

The appearance of cowboy programmers coincides with the appearance of workstation and personal computing. While mainframe computers created organizational points of control, the workstations and personal computers that invaded business computing in the late 1970s made an end-run around these points of control. The mainframe community created a wall, a line of defenses designed to ensure proper software development methods were enforced. The mainframe environment was designed to protect corporate information and corporate finances through the careful control of software.

The advent of workstations and personal computers changed the software development landscape. And when the mainframe backlog became unbearable in the 1980s, within several organizations the authors were associated with the standard reply for any change was "18 months." As a result, business units turned more and more to personal computers for faster software turnaround. The business units did not care about corporate controls. Personally, business units were more concerned with using information and software to complete projects and assignments.

Mainframe-centric teams held to the gatekeeper mentality long after the invading army of personal computers invalidated their defensive positions.

Leading the wave of invaders was the cowboy programmer. The personal computer provided cowboy programmers with the freedom to develop software without the normal organizational constraints. Many organizations set standards of financial thresholds for acquiring computing equipment and software. Information users, many who had trained within the engineering and business computing environment, learned quickly how to get the equipment and software needed for jobs. Much of the data was downloaded never to be uploaded on the mainframe.

Cowboy programmers, sponsored by entrepreneurs, raced to stake claims to new business process markets. The personal computer represented a wide-open frontier, where new rules required new development styles and new beliefs. Winning, in a business sense, often meant first-to-market. In early startup days, these quick results mattered, and function was usually more important than form. This mentality carried out through the dot com boom and subsequent bust. Having been associated with several e-commerce startups, the cowboy is often sought out to spend the long hours away from family and friends to single handedly bring the project to fruition. Bach (1995) refers to this approach as the “big magic” model. He notes that the cowboy or “big magic” model drains the individual to the point of becoming a pathological environment.

In corporate terms, the cowboy represented the new breed, the developers who succeeded in spite of the organization. These new developers were at odds with the mainframe developers because the mainframe controlled access to the information the personal computer developers desperately needed. The difference in process maturity and development timeframes created conflicts in software development and in business processes. The personal computer revolution also changed the very nature of software development. The personal computer also revolutionized how companies conduct their business. More stylized user interfaces needed the talents of graphic artists and human factors consultants. As organizations struggled with the software development infighting, the business universe changed. Organizations attempted to define the appropriate controls for these new machines but lost control of corporate software development.

Corporations never fully regained control of personal computer software development, which meant that these organizations never gained control of the server-centric evolution of the 1990s or the Internet revolution of the new millennium. In essence, almost forty years after the invasion began; organizations are still building cowboy software.

About the Cowboy

The cowboy programmer is able to reach a goal without much direction, without much help, and without a stable working environment (cowboys create their own stability). Because cowboy programmers can work in sub-optimal environments, cowboy programmers mask gaps in requirements and design documentation. Cowboy programmers make up for the requirements and design gaps by making the hard decisions. The cowboy programming decisions may not be in the best interest of the organization, but, to the cowboy programmer, success is completion. Completion requires decision-making. Since the business has already abdicated control of the environment and the cowboy programmer has assumed control, the natural result is that cowboy programmers make the necessary corporate business decisions.

The cowboy programmer breaks the rules out of necessity since the rules are specifically designed to stop ad-hoc, succeed-at-any cost software development. For cowboy programmers, successful delivery is more important than conformity. For cowboy programmers, the ends justify the means.

The cowboy mentality fits the needs of business units, where the software development backlog creates underlying resentment for the software development division. The business unit finds an ally in the cowboy programmer, both want to build business software, and both have contempt for the managers in the software development division. The business unit accepts the cowboy programmer’s view, that the remaining staff is not capable enough to support the business unit.

Since cowboys deliver software in spite of management and since cowboys break corporate rules, the cowboy mentality creates a self-fulfilling prophecy; the corporation needs the cowboy to finish a project. The people the cowboy excluded from the project are not qualified to support the product. The product breaks or needs

enhancements. The cowboy must return to save the day again. The self-fulfilling prophecy reinforces the belief that management is not competent and other developers are not qualified. Cowboys look down upon the rest of the development community and the cowboy's development style reinforces this view. Software developed by cowboy programmers supports the cowboy's view that the organization would fail without cowboy programmers.

The Corporate Cost of the Cowboy

One belief of the mainframe community was that developers should not write code unless the organization was willing to support and maintain the software. This idea disappeared when the personal computer revolution overtook corporate software development divisions. Many organizations did not include maintenance and support costs in project budgets in order to have project budget be financially viable.

It is estimated that organizations currently spend roughly five to one for installing and fixing software over the cost of purchase. (Kerstetter, 2003) This maintenance factor represents a major cost to all organizations for the rugged individualist approach to software development.

Cowboy Organizations

Organizations develop cowboys internally through the combination of corporate culture and business process. First and foremost, organizational processes that reward results over truth are destined to drive cowboy behavior. If projects status reports are fictitious (the project is on schedule up until the week before the project is due, for example), the end of the project requires cowboy behavior to meet project extensions. Organizations that cannot tell the truth are the same organizations that have passive / aggressive behavior, where truth is often know but not told until it is late in the process in order to inflict the maximum corporate penalty on the offending business unit (when the quality assurance team recognizes a process problem but doesn't report the problem until the last day of testing is a classic example. The development team hoping the quality assurance team doesn't find a know flaw is another). Organizations that put off hard decisions also create cowboys by saving the hardest components until the very end. Hard decisions are hard for a reason; the decisions impact many different aspects of an application and business process. At a large telecommunications firm, a project was reported on time until the last week. The project assessment revealed the project would not be completed for at least another ninety days. Another project was scheduled to finish on time, but on closer examination the schedule revealed that each project participant would have to work between 80 and 100 hours per week to meet the project completion date.

Organizations thrive on cowboys. Organizations praise Herculean efforts. Organizations reward cowboys. It is ingrained in our business culture. Business managers want software projects delivered on time and under budget. Both the Agile Alliance and Beck (1995) believe that "highly motivated" individuals, heroes, are the core ingredients in successful software projects. Software success, however, is often measured as the delivery of software. Microsoft's Solution Framework even notes phasing and managing expectations as well as delivering with "known issues."

Organizations reward managers who inspire employees to work extended hours for the same rate, effectively reducing the development cost and saving the company money. The specific activity of getting more work for less money is what passes for leadership in middle management. Being a cowboy has its rewards; cowboys receive the highest praise and are the last to receive a pink slip.

While in the old West of the USA, the cowboy often is portrayed riding the range with the "white hat" pursuing the bad guys in reality the cowboy represents short term gains without concern for longer term success. In contrast, the Samurai reflect a highly disciplined and systematic approach to life and development. The Samurai are discussed in the next section.

THE SAMURAI

The Samurai warrior class of Japan has been portrayed in a variety of ways. Most often the warrior aspect is emphasized as the Samurai initially were employed as private armies for lords of feudal Japan. But when Japan entered a period of peace, education and the professions were pursued by the Samurai class. In fact, ultimately, the Samurai became the basis for the business class as many were absorbed Zaibatsu class. The Samurai utilized the Bushido code of conduct and brought this code to the business world. It is in this vein that the Samurai behavior or culture is proposed as a model for successful software development. The Samurai code is suggested for the class of technical warriors.

Some Aspects of Samurai Code of Behavior

According to the Bushido, there are three things which are essential to the Samurai's life. These are loyalty, duty, and valor. (Cleary, 1999) While the code speaks of many attributes for family and professional life, in reality the emphasis is upon loyalty and a disciplined approach to combat or problem solving. The concept of duty may easily be transformed into ethical behaviors for the modern technical warrior. To the technical Samurai, ethics trumps organization purpose and integrity over monetary gain.

Loyalty and duty provide the focus for much of the code. Loyalty in the modern world can extend beyond the family to the stakeholders of the software development project. The Samurai code may easily be applied to customer satisfaction. Additionally, the code would become the mantra for a longer term view of software including maintainability.

Since the code was written in 1730, obviously modern conveniences were not available. Thus, the instruction is not geared to the literal sense of wording but to the deeper meaning. An example is the description of horsemanship. The description of horsemanship calls for practice and versatility. The Samurai are instructed to learn how to value and judge the tools of their trade. Part of the code illustrates the need to be able to look beyond the exterior and determine true worth and capability. Using Cleary's translation (1999) of the Bushido, it becomes very clear that the Samurai are to study and gain an "extensive understanding of things".

While martial strategy is the focus of Musashi's (translation of 1974) five rings, the meaning of each ring brings to play mastering tools of military strategy and a system approach to problem solving. For the technical warrior, the message is clear. Become proficient in what you do and always be aware of your environment and competitors.

Samurai Practices for Software Development

Only the technical warrior has the capacity to change the state of software development. Therefore, an ideal developer would embrace the code of the Samurai for both the project and the organization to succeed. The technical warrior must have loyalty, discipline, duty, and ethics.

- *Loyalty*: Loyalty covers many aspects of software development including community, customer, and technology. The importance of the team is valued by the technical warrior. The value of a completed project is stressed as part of the loyalty to all stakeholders in the project.
- *Discipline*: As a Samurai code follower, the technical warrior pays attention to detail recognizing that details are important to quality of the end product. The technical warrior constantly works to improve the software for all stakeholders. Additionally, the technical warrior will always study to gain an "extensive understanding of things." (Cleary, 1999)
- *Duty*: The Samurai code requires a sense of responsibility for the completion of the project on time and within budget. The technical warrior should have a passion for the technology and have a sense of obligation to continue to learn and improve his/her skills in using the tools of the trade. Duty is to the project. Duty is to the software.
- *Ethics*: The technical warrior should have the capability to make bold decisions that respect the technology, organization, and most of all, the people including users of the system. Ethical decisions such as not building or releasing with known faults or security issues should form the basis for the technical warrior's behavior code.
- *Other skills and concepts*: As a Samurai technical warrior, the software developer should have a vision for the integration of systems. Caring how the application fits in the overall system of applications, the technical warrior focuses on connections, communications, and efficiency.

Business Reasons for the Samurai Technical Warrior

The attributes and behavior code of the technical warrior create a trust-based organization. Since businesses rely upon software to perform business rules, businesses must trust the creators of the software. In reality what the developer programs is the business rule. New software changes the business processes and the business culture. Therefore, when organizations want to improve software lifecycle efficiency, the Samurai technical warrior is useful in attaining those goals.

Organizations need long-term profitability. Long-term profitability is only sustained with the needs and goals of the stakeholders being satisfied. Since the Samurai technical warrior follows the code of loyalty and duty, organizations can depend upon software development projects that meet the needs of the stakeholders and are completed without shortcuts.

Recently, a company had to rehire a technical consultant to rewrite the flawed code he had been previously paid to perform. Such behavior would not be acceptable to a technical warrior. Employing unethical behavior and unacceptable best practices are not part of the culture and code.

The Samurai and Agile

While UML and the Unified Process attempt to force the cowboy to adhere to corporate policies, it is the Agile processes that truly require Samurai. It is possible to follow UML and still end up with a software development crunch that requires an organization to go cowboy. Agile processes, which focus on simplicity and human interaction, are not suited for the software development vacuum created by cowboys.

Agile methods focus on a continuous delivery of valuable software. Agile methods expect all participants to be engaged and focused. Agile methods stress teamwork and constant communication. Agile methods stress team effectiveness and constant improvement.

From the Agile Manifesto (Agile Manifesto Organization) “We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools; Working software over comprehensive documentation; Customer collaboration over contract negotiation; Responding to change over following a plan.” The Agile Manifesto Organization cites the need to be able to respond to changing requirements and attention to technical excellence.

All of the standards and code of conduct mentioned by the Agile Manifesto Organization are supported by the concept of a Samurai technical warrior. Differences between the two types of developers are discussed in the following section.

SAMURAI VERSUS COWBOY

The major difference between the samurai and the cowboy is found in the relationship between the developers and the organization. Cowboy programmers succeed in the narrowest sense, in the delivery of software. Samurai deliver in the broader sense, maintainable software with a group, not just an individual, capable of maintenance and support. The cowboy’s job ends when the software is accepted by the organization, the Samurai’s job ends when other Samurai are capable of long-term care for the software.

The cowboy focuses on self-improvement; the Samurai focuses on growing the community, growing more Samurai. The cowboy asks, “When is it acceptable?” The Samurai asks, “What are the consequences to the business?” Cowboys do not care for peer review and commentary. Samurai use peer review and commentary as part of the learning experience. For cowboys, knowledge transfer occurs when support personnel read the existing code. For samurai, knowledge transfer is a constant, starting with requirements and continuing through software support and maintenance.

Cowboys are loyal to themselves and to the technology. They form bonds and are loyal to customers – often using “us versus the corporation.” For cowboys it’s all about “me.” Cowboys are selfless in giving time and effort but are selfish on knowledge transfer. The cowboy’s discipline focuses on personal development efficiency. The cowboy expects the maintenance staff to understand the logic embedded in the system. The cowboy is an army of one. Cowboys hoard design information and source code. Many times the cowboy refuses to follow standards established by the team. (McConnell, 2004)

Organizations have two very distinct cowboy personalities, the software developer who “goes cowboy” and the “professional cowboy.” The developer who “goes cowboy” is the developer who reacts to corporate circumstances by taking charge and dismissing others when project failure seems imminent. They recognize the need for drastic measures in order to keep the project from failing. People who “go cowboy” have a personal need to see the project succeed that drives drastic and often counter-productive behavior. Conscientious employees “go cowboy” when project completion requires a Herculean effort and project teams have different levels of commitment. In these situations the team members who cannot accept failure will marginalize the team members who are perceived to not care.

Professional cowboys care about their hero status. Developers who go cowboy despise the status associated with project completion. Professional cowboys intentionally remove team members until they alone remain. Developers who go cowboy only remove employees as a last resort. An organization must determine whether or not it has created an environment which causes developers to go cowboy. Only if the organization understands its role in either hiring professional cowboys or forcing developers to go cowboy is critical to the successful transition from cowboy to Samurai.

Organizations with professional cowboys may have to restructure the entire development organization as well as transition management to a new software development process, just as Microsoft is doing in its operating system group. (Information Week, 2006) Organizations with developers who go cowboy may not have to hire and train new developers; they may only have to restructure software development processes. Organizations need to rethink the concepts of how to achieve loyalty and commitment to the project from the developers.

In the end, the cowboy is interested in each unique journey – the completion of each individual task. The Samurai believes each task is part of a larger corporate journey and the samurai constantly reviews the context of the journey and its impact on the organization. The samurai believes in joint-ownership, that everyone involved in the process must take responsibility for the organization's success.

CONCLUSION

People want to be part of something meaningful; the cowboy takes this away from the team by taking over. If this behavior continues, the net effect is the diminished drive and ability from other workers. The cowboy succeeds and the organization fails. McConnell (2004) labels these types of developers as “prima donna programming ball hogs.”

Organizations supporting cowboy programmers faced delivery of fixes and enhancements that never matched the original development. When budgets tightened, corporations retained the cowboy programmers. The authors have witnessed large corporations shut down multi-million projects only to see the project brought back expense to the organization. These organizations are now in the process of rewriting the applications, often for the third or fourth time. As a consultant on a large project for an energy company, the authors witnessed first hand the cost of rewriting software for the fourth time. The project finally succeeded after a decade of failed attempts using both internal resources and outside consultants.

At a large financial institution in the southeast United States, business units added budget for development but were not charged budget for maintaining software until development completed. The business unit paid the corporate development and support division through budget transfer, well below the actual cost of support. In several cases the corporate group had to hire new resources because they did not have the necessary technology skills needed to support the business unit software. The net effect of supporting the business unit software was an increase in the overall corporate software backlog.

Organizations depend on cowboys to cover for process and management missteps. The corporate penalty for the cowboy solution is the long-term viability of the organization. Organizations that do not rebuild software that is difficult to maintain will eventually lose their market to newer organizations that are not burdened with expensive, outdated software (IBM in the 1990s – losses to Microsoft, Microsoft's losses recently). According to McConnell (2004), organizations that utilize programming heroes “produce software that is alternately brilliant and erratic.” For example, on June 20, 2006, AirTran Airways introduced a new system into the airport hub in Atlanta. The completed software did not work as expected and over 1000 travelers spent up to 10 hours waiting for a flight.

The Samurai grows the organization's development staff – essentially to create more samurai. If the software developers go cowboy or if the software development division is all cowboy, the impact on the organization is the same; software that cannot be maintained chokes out new development by taking time, money, and resources.

If software development is to regain customer trust and loyalty, then the developers must change. Following a proven code of conduct and establishing the technical warrior is a starting point.

REFERENCES

- Agile Manifesto Organization. (). Retrieved April 29, 2006, from <http://agilemanifesto.org/>
- Bach, James (1995, March). Enough about process: what we need are heroes." *IEEE Software*, pp. 96-98.
- Booker, Ellis. (2006, April 3). Taking the long view of Vista. *B to B*, 91(4), 12. Retrieved 4/26/2006, from InfoTrac OneFile database (A144218250).
- Business Week. (2006, March 27). 37Signals: Programming at warp speed. *Business Week*, 3977, 72. Retrieved 4/24/2006, from InfoTrac OneFile database (A143588873).
- Business Wire. (2004, November). Executives should open minds to open source development techniques. *Business Wire*. Retrieved 4/15/2006, from LexisNexis Academic database.
- Business Wire. (2005, July). Survey cites distributed development, customization, and software quality as top software development challenges. *Business Wire*. Retrieved 4/10/2006, from LexisNexis Academic database.
- Clark, Timothy. J. (1999). *Success through quality*. Milwaukee, WI: ASQ Quality Press.
- Cleary, T. (1999). *Code of the Samurai: a modern translation of the Bushido Shoshinshu of Taira Shigesuke*. Boston: Tuttle Publishing.
- Ewusi-Mensah, Kweku. (1997, September). Critical issues in abandoned information systems development projects. *Communications of the ACM*, Vol. 40, No. 9, 74 – 80.
- Hamm, Steve. (2006, April 3). Microsoft's Slo-Mo scramble. *Business Week*, 3978, 36. Retrieved 4/24/2006, from InfoTrac OneFile database (A143883262).
- Information Week. (2006, March 27). Heads roll as Microsoft misses Vista target. *Information Week*. Retrieved 4/26/2006, from InfoTrac OneFile database (A143746762).
- Information Week. (2006, March 31). Game companies get down to business; being a genius with a great idea isn't enough. *Information Week*. Retrieved 4/24/2006, from InfoTrac OneFile database (A143935791).
- Kerstetter, Jim. (2003, June 23). Business software: Comes the revolution. *Business Week*, 3838, 82. Retrieved 4/24/2006, from InfoTrac OneFile database (A104012222).
- Krill, Paul. (6, June 10). Tying development to business. *ComputerWorld Canada*, 21(12) Retrieved 4/24/2006, from LexisNexis Academic database.
- Liebowitz, Jay (2002). Making knowledge management part of your human capital strategy: a look at a large government technical organization. *Journal of International Technology and Information Management*, 11(2), 1-8.
- McConnell, Steve (2004). *Professional software development*. Boston: Addison-Wesley.
- Musashi, Miyamoto. Translated by Victor Harris(1974). *Book of five rings*. Woodstock, New York: Overlook Press.
- Senge, Peter M. (1990). *Fifth discipline: the art & practice of the learning organization*. New York: Currency Doubleday.
- The Economist Newspapers Ltd. (2004, November). Managing complexity - Software development. *The Economist*. Retrieved 4/10/2006, from LexisNexis Academic database.
- Vnu Computing. (2005, September 22). Best practice: Improve your best practice. *VNU Computing*, 40. Retrieved 4/24/2006, from LexisNexis Academic database.
- Webster, B. F. (1995). *Art of 'ware: (Agile Manifesto Organization) Sun Tzu's classic work reinterpreted.* : M & T Books.
- Whitten, Dwayne (2004). Information systems service quality measurement: the evolution of the SERVQUAL instrument. *Journal of International Technology and Information Management*, 13(3), 181-192.
- Wood-Harper, Trevor and Bob Wood. (2005) Multiview as social informatics in action: past, present and future. *Information Technology and People*, Vol 18, No. 1. pp 26 -32.