International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012

DOI: 10.7321/jscse.v2.n7.1

Managing Software Development Risk with the OODA Loop

[1]Nathaniel D. Amsden, [2 *]Narasimha K. Shashidhar
[1,2]*Department of Computer Science, Sam Houston State University, Huntsville, TX, USA*
[1]*nate@thenatrix.net*, [2]*karpoor@shsu.edu*

Abstract. **Software development projects are subject to risks like any other project. These risks must be managed in order for the project to succeed. Current frameworks and models for risk identification, assessment and management are static and unchanging. They lack feedback capability and cannot adapt to future changes in risk events The OODA (Observe, Orient, Decide, Act) Loop, developed during the Korean War by fighter pilot Colonel John Boyd, is a dynamic risk management framework that has built in feedback methods and readily adapts to future changes. It can be successfully employed by development teams as an effective risk management framework, helping projects come in on time and on budget.**

Keywords: *OODA Loop, risk management, software development*

**Corresponding address:*

Narasimha K. Shashidhar,
Department of Computer Science, Sam Houston State University,
Huntsville, TX, USA
Email: nate@thenatrix.net

1. Introduction

Software development projects are subject to risks like any other project. Software development is subject to unique risks which can be mitigated through effective risk management techniques. Risks are unavoidable and must be managed. Successfully managing risks assists developers in completing the project on time and on budget. Strategies selected to manage risk may result in a better product than originally anticipated. Identifying, analyzing, tracking, and managing software risk aids crucial decision making including release readiness.

During the Korean War, fighter pilot Colonel John Boyd developed a series of four steps he noticed fighter pilots followed during air to air combat. These steps, observe, orient, decide, act, became known as the OODA Loop. Col. Boyd went on to become a superb fighter pilot and Pentagon strategist.

Current risk management frameworks are static and unchanging. They lack feedback capability and cannot adapt to future changes in risk events. The OODA Loop is a dynamic risk management framework with built in feedback methods and readily adapts to future changes. Software development teams can employ the OODA Loop to manage risks affecting their projects, thereby reducing risk.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012

DOI: 10.7321/jscse.v2.n7.1

2. Literature review

Risks plague projects. Software development projects are not immune to risks. Risk management strategies are crucial to identifying, tracking and reducing risks. A 1995 study of Department of Defense (DoD) software spending showed only 2% of software was able to be used as delivered. 75% was either never used or cancelled prior to delivery. In total, $35.7 billion was spent on software [1]. Much research involves surveying current software developers and program manager professionals. Researchers cite Barry W. Boehm's work often while discussing software risk management research. The studies result in similar, but different, strategies to identify, track and reduce risk.

While reviewing past research, similar components of risk were identified. These risk components include scheduling and timing risks, system functionality risks, subcontracting risks, requirement management risks, resource usage and performance risks, and personnel management risks [2]. A lack of knowledge of software suppliers adds an increased level of risk. This poses a large problem to the DoD, who does very little of its own software development and instead contracts it out [3] [4]. Risks stem from changing requirements, lack of skills, fault technologies, gold plating and an unrealistic project schedule. Gold plating refers to developers developing a requirement beyond the threshold/objective to make it better [5]. As companies grow, development teams may be spread across a building, the country or even the world [6]. Distributed development teams add risk to software projects, based simply on the fact that they are not in a centralized location.

Two main methods of risk identification are used today. These are checklists and group effort, or brainstorming [5]. Effective requirements engineering aids risk identification. Eliciting requirements from stakeholders is important, but even more important is continuing to ask why a requirement should be as requested. Analyze and specify requirements to eliminate IT redundancy and avoid unnecessary risks [7]. Requirements elicitation, analysis, documentation, verification, review, approval, configuration control and traceability should be incorporated into sound risk management processes and procedures [1].

Starting early to identify and plan for risks reduces risks later in the development cycle. Cleanroom software development and software inspections focus on avoiding risks before introducing them into the project [8]. If risks are avoided to begin with, time, money and effort are not wasted during the development process to mitigate the risks. One study suggested that an increased identification of risks led to an over-confidence and over-optimism in estimating software development efforts [9]. More research with better scenarios is required to provide more data with which to better prove their argument. Even though development teams may be skilled to implement risk management techniques in software projects, they do not always do so. Company history, structure, processes and reward systems can facilitate or inhibit risk management processes [10].

Conceptualizing requirements has been a popular method of tracking requirements, identifying and managing risks. Project requirements are the greatest source of risk [7]. Various model-based requirement management approaches exist to better identify, track and manage requirements. Models are often not used. When they are, models play a secondary role. In the past, models weren't formally connected to software. Thus there was no means to ensure programmers followed design decisions captured in a model [11].

Project characteristics, the project risk management team, risk identification approaches, and project quality all contribute to and affect the level of project risk. Assessing the impact of project risk simultaneously with the impact of residual performance risk can provide project managers with a better understanding of the effectiveness and adequacy of their risk management techniques [12]. No matter how implemented, risk management capabilities play important roles in managing software projects. However, the conceptualization and development of risk management theories lags the requirements of practice. Risk management practice lags the understandings of risk management found in research studies [13].

Current frameworks and models for risk identification, assessment and management are static and unchanging. They lack feedback capability and cannot adapt to future changes in risk events. Dynamic risk management frameworks provide futuristic assessments of risk events [14]. Dynamic risk management frameworks, coupled with static models, can enhance project success.

Software development projects benefit greatly from model-based requirements engineering. Identifying, assessing, analyzing, verifying, tracking, and managing requirements reduce risk to software projects. The earlier risks are identified and managed in the projects, the less costly they are to fix should they ever become a problem.

Very little research has been conducted relating the Observe-Orient-Decide-Act (OODA) loop to risk management in the software development process. This work talks mainly to agile software development and is found primarily on personal blogs. The level of detail is relatively low.

Steve Adolph relates the OODA Loop to agile software development in [15]. He argues agility depends on the tempo at which we iterate through the OODA Loop. This speed depends on culture, not methodologies or tools. This paper is primarily an introduction to the OODA Loop and agile software development. It briefly outlines how the OODA Loop fits in with the notion of agile software development and proposes research opportunities. It does not detail any conducted research or present information about risk management.

Dr. David Ullman discusses how the OODA Loop has been applied to business and product development in [16]. He specifically explains how business and product development teams get stuck on the D and action never occurs. He prescribes guidelines to unstick the OODA Loop so decisions can be made and action taken.

3. Colonel John Boyd's OODA loop

3.1. Col. John Boyd

Colonel John Boyd (1927-1997) was a United States Air Force fighter pilot and brilliant military strategist. During the Korean War, Boyd observed a cycle of four actions pilots took during combat. He named these actions the OODA Loop. Pilots who cycle through the OODA Loop faster than others dominate dogfights. Their speed through the OODA Loop forces opposing pilots to constantly re-observe and re-orient themselves. This prevents them from making decisions and acting to gain the upper hand.

Col. Boyd mastered the OODA Loop and received the nickname "40 Second Boyd." He bet pilots forty dollars that he could defeat them in a dogfight in less than forty seconds. He never lost. After his Air Force career, he worked for the Pentagon and developed strategies for the invasion of Iraq in the first Gulf War.

3.2. The OODA loop

The OODA Loop is composed of four steps: observe, orient, decide, and act, as shown in figure one below.

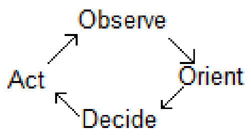

Figure 1. The OODA loop

Development teams cycle through these steps repeatedly. Observation deals with collecting data about the situation and surroundings. Orientation is the analysis of that data to form a mental perspective. In the decision phase, a specific course of action is chosen based upon the gathered and

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.1

analyzed data. Action is the physical act of executing the decision. The results of the action should be observed, and the cycle repeats until no longer needed.

Although created for fighter pilots and air-to-air combat, the OODA Loop applies to risk management for software development. Just as pilots apply the OODA Loop to manage risk in combat, that is, to not get shot down, stakeholders, project managers, and developers can apply the OODA Loop so software projects don't crash and burn. The OODA Loop can assist in managing scheduling and timing risks, system functionality risks, subcontracting risks, requirement management risks, resource usage and performance risks, and personnel management risks.

4. The OODA loop and software development risk management

4.1. Observe

The first step to risk management is to identify, or observe, risks. Failing to identify risks can drastically harm software projects. Four factors influence observations in the OODA Loop as shown in figure two below. These four factors include outside information, unfolding circumstances, unfolding interaction with the environment, and implicit guidance and control. These factors are external to the loop. Combined, they assist developers and project managers with risk identification.

Outside information is required for effective risk management. Software developers must absolutely receive information from stakeholders. Eliciting requirements from stakeholders is time consuming. Care should be taken early to identify all classes of stakeholders from all involved organizations. Missing a stakeholder or missing a requirement can complicate development.

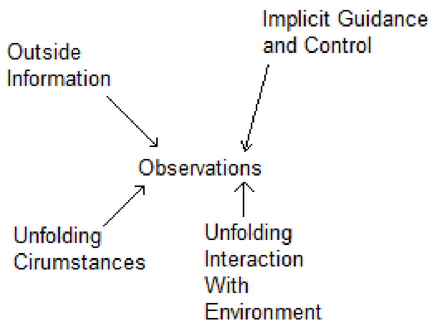
Outside Information
Implicit Guidance and Control
Observations
Unfolding Cirumstances
Unfolding Interaction With Environment

Figure 2. The observe step of the OODA loop

As development progresses, whether in the requirement identification, design, development, or testing phase, unfolding circumstances change the risk posture. How does risk change as software components are developed? Generally risk is reduced, but do previously unidentified risk sources appear? Do stakeholders identify new requirements or not like how a requirement was implemented? Requirements creep is a major risk to software projects. Failing to capture requirements during the requirements identification phase is one source of requirements creep. The cost and effort to integrate new requirements increases as development progresses. Even more costly is fixing bugs as development progresses. Finding bugs early in development and fixing them is much cheaper than finding and fixing bugs late in development. Apply the OODA Loop on a small scale when new requirements or even coding bugs are identified. What is the bug? How does it affect the software? How do we fix it? Implement the fix!

Unfolding interaction with the environment inputs into observation. As development progresses and components completed, development teams must ensure each component works as intended. There may be unexpected consequences to components it interfaces with. Careful planning and design can mitigate these side effects. Interaction with the environment is also critical in developing software for a system of systems. Requirements must capture how the systems interact with each other and with the software. Risks in system of systems are more numerous. Each system and interface adds additional risks to be tracked and managed.

Implicit guidance and control feeds the OODA Loop at each stage. It is especially crucial during observation, as this is when direct orders, key performance parameters, laws, regulations, and best practices must be identified and planned for. It is here where scheduling and timing risks are first identified. Stakeholders outline the desired timeframe for delivery and any key milestones that must be met. Specific guidance may determine how specific risks are to be managed.

4.2. Orient

Information gleaned from observation feeds forward into the second step of the OODA Loop, orient. Orientation aligns observed information into a well-defined, logical manner from which decisions can be more readily made. During this stage, risks must be assessed based on probability of occurrence and the potential impact. Numerous risks can be ranked based on calculated composite risk indices. The higher the composite risk index, the more severe the risk is. Col. Boyd identified five factors contributing to how pilots oriented themselves based on observed information. These five factors include cultural traditions, new information, analysis and synthesis, previous experiences, and genetic heritage. Software development teams orient themselves and their projects according to these five factors. Figure three below presents the relationships between the five factors.

Requirements, systems, data, and circumstances change. This leads to new information the team can use to identify and orient the project to manage risks. This factor is actually a mini observe step built into the orientation step. It is a reminder for teams to constantly absorb new information and watch for unanticipated risks.

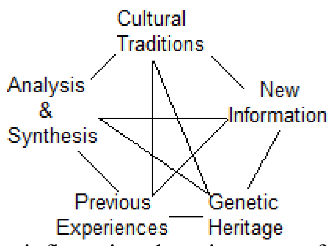

Figure 3. Factors influencing the orient step of the OODA loop

Analysis and Synthesis is a no-brainer. Information and observations are useless without analysis. Analyzing identified risks allows teams to determine appropriate and effective risk management techniques. Software can be analyzed for functionality, bugs, and completeness. It can be synthesized and tested.

We argue that for risk management of software development projects, the three factors cultural traditions, genetic heritage, and previous experiences are very similar to each other. Yet there is a fine distinction between the three.

Cultural traditions influence software development teams. In the scope of this paper, cultural traditions refers not to the social upbringing of team members, but rather to the culture and traditions of the organization. The team or organization may have a preference for one software development or requirements model. Genetic heritage for a software project describes how developers and stakeholders have learned to manage projects and risks. Every person has had different training and education. Development teams and project managers rely on previous experiences to identify and manage risks in current projects. Having successfully (or unsuccessfully!) completed past projects brings experience to the team. Risks that affected past projects may affect current projects. Team members use experience from those projects to understand how to track and mitigate current risks.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.1

4.3. Decide

Once risks have been identified and analyzed, and oriented to project goals, the development team must choose a risk management strategy. Can the risk be avoided? Can the team transfer the risk elsewhere? If not, should the team mitigate the risk or accept it? Seek management inputs. This is where additional implicit guidance and control occurs. Develop the risk action plans and contingency plans in case a risk turns into a problem.

Requirements engineering is also a key part of risk management. The ability to identify, analyze, monitor, and track requirements and project status through the development lifecycle can greatly decrease risk. Deciding how to manage and implement requirements greatly reduces risk. It is much easier to plan for and integrate requirements at the beginning of the software development cycle. As software development progresses, cost and effort to implement new requirements, makes changes, and fix bugs increases.

Feedback from decisions flows back to the observation step. The risk management strategy chosen may affect the project schedule or budget. It may change how the software works entirely, even though it might accomplish the same function. The benefit of effectively managing requirements is that feedback occurs quickly when decisions are made, allowing quick iteration through the OODA Loop.

4.4. Act

Implement strategies! Manage risks! As soon as decisions have been made and risk management strategies chosen, go forth and execute. Although risk management is not necessarily a quick process, feedback as risks are managed can be quickly fed back into the observation stage. As development progresses, how the project interacts with the environment must be considered. How does the software (components) work operationally? Does it work as intended? Is further development needed to tweak the software or correct bugs? Ensure stakeholders are happy with the progress and results. If not, they may require additional tweaks or features. This adds additional risk, as requirements begin changing and requirement creep sets in.

Risks need to be tracked and the results of the risk management strategies recorded and shared. Team members and stakeholders need to know the status of their project. If the risk is not reduced as anticipated, another iteration of the OODA Loop is required.

4.5. The loop

The four steps and factors influencing them are shown in figure four below. The OODA Loop is not a once-through framework for risk management. It is to be applied repetitively throughout the entire software development cycle. Risks to project development do not go away until project completion (of course new and different risks then appear!). Teams must not get stuck observing and orienting themselves to risks and development status. Teams must decide and act on observed information.

Feedback from the decision and action stages flows back to the observation stage. Continually observe the results of team decisions and actions. Risks rarely go away by themselves. Repetitive iterations of the OODA Loop will reduce software project risks and increase the likelihood of completion on time and on budget.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.1

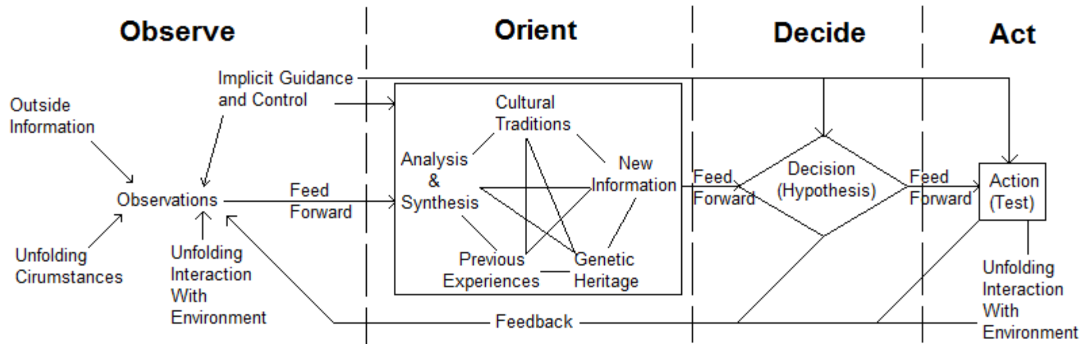

Figure 4. The full OODA loop as described by Col. John Boyd

5. Conclusions and future work

The OODA Loop is a tool for effective risk management. Software projects, like all projects, have risk. Software project teams can use the OODA Loop as a risk management framework. Each step helps developers and project managers identify, track, and manage risks. Due to the cyclic nature of the OODA Loop, multiple iterations can be applied as the project and as risks evolve over time. Successful implementation of the OODA Loop can assist project managers in bringing their projects in on time and on budget.

In the future, we plan to use the OODA Loop as a risk management framework for a software project. We will test its effectiveness over the course of the project. Each identified risk will be tracked and all observations, decisions, and actions will be recorded. We shall report on our ability to implement the OODA Loop, and its effectiveness as a risk management framework.

6. References

[1] T. R. Leishman and D. A. Cook, "Requirements Risk Can Drown Software Projects," Journal of the Quality Assurance Institute, vol. 17, no. 2, pp. 21-26, 2003.
[2] J. Ropponen and K. Lyytinen, " Components of software development risk: How to address them? A project manager survey," IEEE Transactions on Software Engineering, vol. 26, no. 2, pp. 98-112, 2000.
[3] K. V. Schinasi, "Defense Acquisitions: Knowledge of Software Suppliers Needed to Manage Risks: GAO-04-678.," United States General Accounting Office, Washington D.C., 2004.
[4] M. L. Polydys and S. Wisseman, "Software Assurance (SwA) in Acquisition: Mitigating Risks to the Enterprise," [Online]. Available: https://buildsecurityin.us-cert.gov/swa/downloads/SwAin Acquisition%20MitigatingRisks%20to%20Enterprise.pdf, 2008.
[5] G. N. K. Suresh Babu and S. K. Srivatsa, "Increasing Success of Software Projects through Minimizing Risks," International Journal of Research & Reviews in Software Engineering, vol. 1, no. 1, pp. 18-22, 2011.
[6] M. Mohtashami, T. Marlowe and V. Kirova, "Risk management for collaborative software development," Information Systems Management, vol. 23, no. 4, pp. 20-30, 2006.
[7] Borland Software Corporation, "Mitigating Risk With Effective Requirements Engineering,"[Online]. Available: http://www.borland.com/resources/en/pdf/white_papers/ mitigating_risk_with_effective_requirements_engineering.pdf, 2005.
[8] T. R. Adler, J. G. Leonard and R. K. Nordgren, "Improving risk management: moving from risk elimination to risk avoidance," Information and Software Technology, vol. 41, no. 1, pp. 29-34, 1999.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.1

[9] M. Jørgensen, "Identification of more risks can lead to increased over-optimism of and over-confidence in software development effort estimates," Information and Software Technology, vol. 52, no. 5, pp. 505-516, 2010.
[10] J. Stoddard and Y. H. Kwak, "Project Risk Management: Lessons Learned from Software Development Environment," Technovation, vol. 24, no. 11, pp. 915-920, 2004.
[11] B. Selic, "The Pragmatics of Model-Driven Development," IEEE Software, vol. 20, no. 5, pp. 19-25, 2003.
[12] M. Uzzafer, "A New Dimension in Software Risk Managment," World Acadamey of Science, Engineering and Technology, vol. 64, pp. 340-344, 2010.
[13] P. L. Bannerman, "Risk and risk management in software products: a reassessment," The Journal of Systems & Software, vol. 81, no. 12, pp. 2118-2133, 2008.
[14] L. Sarigiannidis and P. D. Chatzoglou, "Software Development Project Risk Management: A New Conceptual Framework," Journal of Software Engineering & Applications, vol. 4, no. 5, pp. 293-305, 2011.
[15] S. Adolph, "What lessons can the agile community learn from a maverick fighter pilot?," In Proceedings of the Agile Conference, Vancouver, BC, pp. 99-104, 2006.
[16] D.G. Ullman, "'OO-OO-OO!' The sound of a broken OODA Loop," Crosstalk: Journal of Defense Software Engineering, vol. 20, no. 4, pp. 22-25, 2007.