International Journal of Computing and Business Research
(IJCBR)
ISSN (Online) : 2229-6166
VOLUME 5 ISSUE 1 JANUARY 2014

GROUPTHINK IN SOFTWARE ENGINEERING

Michael Scott Brown
Project Director, Software Engineering
University of Maryland University College

Abstract: This brief paper outlines research in software engineering that matches characteristics of GroupThink. GroupThink is an issue that occurs in groups that leads to incorrect decision-making. GroupThink could be a cause of software defects, project failure rates and other quality issues in software engineering. The root cause of GroupThink is group cohesion. But there are many benefits of group cohesion, so the elimination of group cohesion is not a solution. Much research has been done to determine how to prevent GroupThink without reducing group cohesion. Some solutions to the problem of GroupThink will also be presented. A number of these techniques are included in software engineering best practices, while others should be future research.

Keywords: Software Engineering, GroupThink, Peopleware,

1. Introduction

It is widely believed that the success or failure of a software project is greatly dependent upon the personnel working on it. Skilled personnel working together can produce reliable software within budget. But lack of the right combination of people can produce failure. This factor in building software, typically referred to as Peopleware, justifies research into how people behave in software engineering groups.

A number of studies have shown that a significant percentage of software engineering projects fail. Failure can be terminating the project, greatly exceeding budget or significantly reducing the feature

set. Project failure can cause enormous financial strain on organizations that spend money for software and do not receive the benefits of it. This is often referred to as the Software Crisis.

This paper describes a problem, called GroupThink (Janis, 1982; Janis, 1972), which occurs in many groups that make decisions. GroupThink is a phenomenon in which group cohesion can adversely affect the group's ability to make good decisions. Some authors have stated that software engineering groups suffer from GroupThink (Kontio, et al, 2004; Salinger, et al., 2008). But these papers do not go into detail GroupThink diagnosis techniques outlined by Janus (Janus, 1982). This paper will describe and analyze current research that supports that assertion.

Most research in GroupThink is in the form of case studies (Janus, 1982). These studies look at specific groups that made poor decisions and through analysis determine that they suffer from GroupThink. Some common examples are the Kennedy administration's decision to support the Bay of Big invasion; NASA decision to launch the Space Shuttle Challenger and the Nixon administration's decision to cover up Watergate. This paper takes a slightly different approach. Instead of focusing on a specific group and a single decision-making process, this paper focuses on a generic group, that group being software development teams. After decades of research in Peopleware we have learned a lot about how these teams function.

2. GroupThink

GroupThink was first introduced by Irving Janis in 1972 and is a failure that some groups exhibit that prevents them from making good decisions. Many factors can lead to GroupThink, however it is primarily an issue with group cohesion.

2.1. Group Cohesion

Group cohesion is a phenomenon that has been observed in groups from the beginning of the study of group dynamics. Although there is some differing of opinions, commonly group cohesion is thought of as "the degree to which the members of a group desire to remain in the group."(Cartwright & Zander, 1968) High group cohesion can produce many benefits including: the ability to retain members; the participation and loyalty of members; ownership of the product and the feeling of security by the members (Cartwright & Zander, 1968; DeMarco & Lister, 1987).

Group cohesion should be a goal of every software development project manager. Cohesion has many positive effects on software development and prominent members of the software community promote it as a way to increase development (Demarco & Lister, 1987). Cohesion seems logically to be a positive group characteristic. Cohesiveness, in any group, helps to reduced anxiety, heightened self-esteem, and retains members (Cartwright & Zander, 1968). With current software development being a high anxiety, high turnover occupation, group cohesion is thought to be a solution to many problems.

2.2. GroupThink

Groupthink is a term used to describe the "mode of thinking that people engage in when they are deeply involved in a cohesive in-group, when the members' striving for unanimity override their motivation to realistically appraise alternative courses of action." (Janus, 1972) Members of the group concentrate on maintaining group cohesion and avoid conflict. This phenomenon occurs all the time and is generally unnoticed. Members are generally unaware of Groupthink when it happens. Table 1 shows the primary conditions that breed GroupThink (Janus, 1972). It is easy to see that many software engineering groups exhibit these conditions.

Table 1 Condition that breed GroupThink

Conditions that Breed GroupThink
High group cohesion
Group has insulation from judgment by elements outside of the group while the group process is going on
Leader of the group promotes his/her own ideas

After examining the conditions that can cause GroupThink, Janus presents characteristics of GroupThink. By analyzing a group using these characteristics it can be determined if the group suffers from GroupThink. Table 2 shows these three characteristics.

Table 2 Characteristics of GroupThink

International Journal of Computing and Business Research
(IJCBR)
ISSN (Online) : 2229-6166
VOLUME 5 ISSUE 1 JANUARY 2014

Characteristics of GroupThink
Overestimations of the group
Closed-mindedness
Pressure toward uniformity

GroupThink is a condition that occurs in many groups. It is fundamentally caused by group cohesion, but since there are many benefits to cohesion methods are sot to prevent GroupThink without eliminating cohesion. There has been much research done on the subject and methods that prevent it. Section 5 of this paper will address some of these methods.

3. Existing Literature on GroupThink in Software Engineering

A number of publications reference GroupThink when discussing cultural or group dynamics in software engineering and information technology (Salinger, et al., 2008; Gallivan & Srite, 2005). Since any group has the potential to exhibit GroupThink, it is easy to conclude that software development group could fall victim to it. But these publications do not attempt to generalize the typical software engineering group and conclude that GroupThink is exhibited in them.

An exception to this is an article published by Schiano and Weiss (Schiano & Weiss, 2006). Schiano and Weiss assert that GroupThink was the cause of the Y2K crisis and is currently the reason that many organizations have poor Information Technology security.

The Schiano and Weiss (Schiano & Weiss, 2006) research draws correlations between existing research on the causes of the Y2K problem with the characteristics of GroupThink. It continues by drawing similar correlations between attitudes toward security and GroupThink. Unlike previous research in GroupThink Schiano and Weiss do not conclude that a specific group exhibited GroupThink, rather they generalized groups working in these two areas.

4. Methodology

This research paper uses the meta-analysis research methodology. In meta-analysis research results from different studies, normally on different topics, are combined. The combination of these studies produces some new conclusion.

This paper identifies and examines research that relate to Janus's characteristics of GroupThink. Multiple studies exist relating to each characteristic. By combining these studies we can conclude that GroupThink exists in typical software engineering teams.

5. GroupThink in Software Engineering

Before we can address methods to prevent GroupThink, we need to determine if GroupThink actually occurs in many software engineering groups. This can be done through a survey of existing literature in Peopleware.

A variety of researchers have studied the common characteristics of software development teams. These characteristics can be used to demonstrate that Groupthink causes many of the decision flaws in software development. The typical software development team is cohesive with respect to groups in general.

5.1. Overestimation

The first characteristic of GroupThink is overestimation of the group. Overestimation of the group's ability can be seen in existing Peopleware literature.

A common overestimation that teams exhibit deals with predicting when software will be completed. It is common knowledge that software engineers do a poor job of predicting completion dates (Yourdon, 1997). There are two ways to describe a project taking longer than estimated: the development team overestimated their ability or underestimated the complexity of the project. Both explanations are equivalent. A study by Michiel van Genuchten (Van Genuchten, 1991) shows that the two most common reasons that software projects overrun their schedule are that they spend more time on other work like maintenance and that they underestimated the complexity of the project (Van Genuchten, 1991). Overestimating their ability to do other work like maintenance caused 27% of the

projects to be late. Underestimating the complexity of the projects caused 20% of them to be late. These findings show that software developers overestimate their ability to do maintenance and develop software. Lederer (Lederer & Prasad, 1992) showed that 2/3 of all software projects overrun their estimation.

Recently, research has given a clearer picture of this overestimation. Jorgensen and Grimstad (Jorgensen & Grimstad, 2012) conducted research by contracting over 350 software developers working in industry. Developers were asked to estimate a short software development task. They were also given three questionnaires to measure different characteristics about the participants. The first questionnaire measured self-construal, which is if they place importance on interdependence or independence. The second questionnaire measured the subjects thinking style, holistic or analytic. The final question measured their need-for-cognition, which is if they enjoy problem solving. These are three common characteristics used in cultural studies.

Jorgensen and Grimstad (Jorgensen & Grimstad, 2012) found that individuals that score highly in interdependence test were more likely to give lower estimation for software development tasks. These results support previous research concluding that software developers do a poor job of estimation. Other literature supports the correlation between the desire for group cohesion and interdependence (Nisbett, 2003). Considering that developers that value interdependence also value group cohesion, this research strongly supports the correlation between GroupThink and Software Engineering.

Another interesting result of Jorgensen and Grimstad (Jorgensen & Grimstad, 2012) was that managers, developers with a lot of experience and developers with Master's degrees also give lower estimations. These individuals are considered leaders of the group. This finding is important because one of the solutions to prevent GroupThink that will be described later in this paper is that leaders do not give opinions.

Cerpa and Verner (Cerpa & Verner, 2009) conducted a study of failed software projects. Their survey looked at factors of failed software projects. They found that in 81.4% of failed projects the complexity of the project was under-estimated. This supports the assertion that many in software engineering overestimate their abilities.

These studies show a consistent overestimation of ability and underestimation of software development that span a number of decades. Even though it is commonly known that software development estimations are often very low, software developers have been unable to compensate for this.

5.2. Closed-mindedness

The second characteristic of GroupThink is closed-mindedness. In early stages of software development the development team proposes how to implement the requested features within time and budget. They develop a plan to do so. In many cases projects fail to fully implement requirements within time and budget constraints. However, much research has indicated that members of the software development team are some of the last to recognize this inevitable failure (Boehm & DeMarco, 1997).

The Cerpa and Verner (Cerpa & Verner, 2009) study that was mentioned in the previous section also found that in 75.7% of the projects risk management best practices were not followed. Risks were not adequately identified, controlled and managed. One observation of their study is that even in cases where project obviously failed team members would not admit that they failed (Cerpa & Verner, 2009). In the minds of the team members on these projects they were successful.

5.3. Pressure Toward Uniformity

There are a number of ways that software development groups exhibit internal pressure toward uniformity. There are many ways to build software and ultimately the group must decide on one method. Janus writes there are two ways that this pressure can be identified: self-censorship and pressure of group members to agree.

Self-censorship is seen when software development efforts fall behind schedule. Often setbacks are not reported to management when first realized (McConnel, 1996).

Pressure on group members, is seen in many phases of development. There is pressure on members of a development team to not take on critical roles (Carr, 1997). Management rewards members that agree with group ideas and punishes those who point out risks.

6. Methods to Address GroupThink

Research in GroupThink has developed a number of methods to prevent GroupThink without eliminating group cohesion. The software engineering community has adopted some of these methods. This also strengthens the argument that software engineering teams suffer from GroupThink. Not only do the causes of GroupThink exist in software engineering, but also the solutions to GroupThink increase quality of software.

6.1. Assign Role of Critic

One method to prevent GroupThink is to assign a role of critic. Many people have difficulties bringing up issues with someone's idea. This is viewed as too confrontational. Assigning a role of critic reduces this stress. The critic must come up with potential issues with the idea. That is their role. Most software development methods have included this solution in the form of design and code reviews. Much research has indicated that reviews of this type are beneficial (Constantine, 1995; Kemerer & Paulk, 2009).

6.2. Leaders Avoid Making Suggestions

Researchers that study GroupThink believe that groups may come to better decisions if leaders of the group avoid making suggestions (Janus, 1972). When leaders put forth suggestions other group members are too quick to agree with them. This prevents the group from fully considering all possible approaches. Little if any research has been done on this in the software engineering field. This could be a possible area of future research.

6.3. Independent Evaluation

A third method to prevent GroupThink is independent evaluation. This is also used in software engineering. The field of Independent Verification and Validation (IV&V) is based upon this

philosophy. IV&V is used by many organizations including NASA. There is a variety of research indicating that IV&V reduces costs and increases quality (Akella & Rao, 2011).

6.4. Make Suggestions Anonymous

There may be another way to prevent GroupThink without eliminating input from the group leaders. If input from the group is anonymous group's members do not automatically support the opinions of their superiors. They do not know what input came from the group leaders. Research suggests that this can improve group decision-making (Kontio, et al., 2004).

7. Conclusion

This research has multiple contributions. The obvious one is that it concludes that typical software engineering groups exhibit GroupThink.

Understanding the correlations between software engineering and Groupthink provides a framework for research. Research describing problems that occur in software engineering and best practices fit nicely into the GroupThink framework. They are not disjoint research; rather there are relationships between them. They all address issues with group cohesion.

There are some methods of addressing GroupThink that could be applied to software engineering. One solution that Janus suggests is having a leader that does not give opinions. This removes the desire of people to please their superiors by agreeing with them. Currently there has been no research in software engineering on this approach. This could be future research.
Future research in GroupThink should be applied to software engineering. If evidence suggests that GroupThink occurs in software engineering, then future research in GroupThink should be applied to software engineering.

This paper outlines current research in Peopleware that support the notion that GroupThink occurs in software engineering groups. It outlines many solutions to GroupThink. Some are part of current best practices, while others could be future research. Preventing GroupThink while not eliminating group cohesion is very important for software engineering.

References

[1] Janis, I (1982) Groupthink: Psychological Studies of Policy Decisions and Fiascoes, Houghton Mifflin Co., Boston, Massachusetts.

[2] Janis, I (1972) Victims of Groupthink: A Psychological Study of Foreign-policy Decisions and Fiascoes, Houghton Mifflin Co., Atlanta, Georgia.

[3] Kontio, J, et al. (2004) "Using Focus Group Method in Software Engineering: Obtaining Practitioner and User Experience", In *Proceedings of the International Symposium on Empirical Software Engineering,* pages 271-280, Finland.

[4] Salinger, S, et al. (2008) "A Coding Scheme Development Methodology using Grounded Theory for Qualitative Analysis of Pair Programming", *An Interdisciplinary Journal on Humans in ICT Environments*, Vol. 4, No. 1, pp. 9-25.

[5] Esser, J (1998) "Alive and Well after 25 Years: A Review of Groupthink Research", *Organizational Behavior and Human Decision Processes*, Vol. 73, No. 2/3, pp. 116-141.

[6] Cartwright, D, Zander, A. (1968) *Group Dynamics: Research and Theory*, Haprer and Row, New York, New York.

[7] DeMarco, T & Lister, T (1987) *Peopleware: Productive Projects and Teams*, Dorest House Publishing Co., New York, New York.

[8] Gallivan, M & Srite, M (2005) "Information Technology and Culture: Identifying Fragmentary and Holistic Perspectives of Culture", *Information and Organization*, Vol. 15, pp. 295-338.

[9] Schiano, W & Weiss, J (2006) "Y2K all over again: How groupthink permeates IS and compromises security", *Business Horizons*, Vol. 49, pp. 115-125.

[10] Yourdon, E (1997) Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, Prentice Hall, Upper Saddle River, New Jersey.

[11] Van Genuchten, M (1991) "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development", *IEEE Transactions on Software Engineering*, Vol. 17, No. 6, pp. 582-590.

[12] Lederer, A & Prasad, J (1992) "Nine Management Guidelines for Better Cost Estimating", *Communications of the ACM*, Vol. 35, No. 2, pp. 51-59.

[13] Jorgensen, M & Grimstad, S. (2012) "Software Development Estimation Biases: The Role of Interdependence", *IEEE Transactions On Software Engineering*, Vol. 38, No. 3, pp. 677-693.

[14] Nisbett, R (2003) The Geography of Thought: How Asians and Westerners Think Differently, Free Press, New York, New York.

[15] Cerpa, N & Verner, J (2009) "Why Did Your Project Fail?", *Communications of the ACM*, Vol. 52, No. 12, pp. 130-134.

[16] Boehm, B & DeMarco, T. Software Risk Management, *IEEE Software*, Vol. 17, No, 19, pp. 17-19.

[17] McConnel, S (1996) *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, Redmond, Washington.

[18] Carr M (1997) "Risk Management May Not Be for Everyone", *IEEE Software*, Vol. 14, No. 3, pp. 21-24.

[19] Constantine, L (1995) *Constantine on Peopleware,* Yourdon Press, Englewood Cliffs, New Jersey.

[20] Kemerer, C & Paulk, M (2009) "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data", *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, pp. 534-550.

[21] Akella, P & Rao, K (2011) "Effective Independent Quality Assessment using IV&V", *International Journal of Computer Science & Information Technology,* Vol. 3, No. 3, pp. 188-198.