Information Research, Vol. 7 No. 3, April 2002,

The role of motivation and risk behaviour in software development success

Kenneth R. Walsh and Helmut Schneider
Information Systems and Decision Sciences Department
Louisiana State University
Baton Rouge, Louisiana, USA

Abstract

Although, significant progress has been made in software development methodologies, software project failures continue to exist. Previous work on risk management has found several risk factors and has developed methods for avoiding those factors from causing failures of software projects. However, software development remains a risky undertaking where decisions must be made without complete information. Another approach to risk management is to concentrate on those making decisions as agents of an organization rather than just the methods they use. We propose that the behaviour of decision makers affected by risk propensity and motivation is critical to the outcome of a software project. This paper discusses individual risk behaviour and applies agency theory to manage the behaviour of individuals in the context of software development. The implication is that the goals of individuals and their propensity to take risks may have a significant impact on project success not addressed in previous research. We apply agency theory to software development. Agency theory is a management theory that puts the principle agent dilemma at center stage and is a way to understand the software development process and how to improve it.

Introduction

Although the technology of software and software development has changed dramatically, and the number of software packages acquired by organizations has increased significantly, problems with the implementation of information systems remains common. In the late seventies and early eighties the implementation of a management information system was considered fraught with uncertainty (Alter and Ginzberg, 1978) and it was the exception rather than the rule for an organization to develop an information system within budget and according to schedule (Zmud, 1980). Although modern software practices (Zmud, 1980) have benefited the management of software development over the last 20 years, the failure rate of new implementation remains high. Identifying risk factors and probabilities associated with the risks (Boehm, 1991, Alter and Ginzberg, 1978, Barki, *et al.*, 1993) has been viewed as a step to reduce risk. Quantitative and qualitative models have been developed (Bennett *et al.*, 1996, Barki *et al.*, 1993) to measure the magnitude of risks. Risk management of software projects has been the focus of much research over the last 20 years. Risk assessment and risk control (Boehm, 1991) are the two main steps in managing project risk for software development. Risk assessment involves risk identification, risk analysis and risk prioritization. Risk control involves risk management planning, risk resolution and risk monitoring. Assignment of probabilities (Boehm, 1991, Bennett *et al.*, 1996) is used to evaluate the likelihood of the occurrence of hazards. Combining estimates of loss magnitude with failure probabilities (Boehm, 1991, Bennett *et al.*, 1996) is the usual way of evaluating risk situations.

Several factors have been identified (Boehm, 1991, Alter and Ginzberg, 1978 and Barki *et al.*, 1993) that threaten successful software implementation. Boehm gives ten factors, Alter and Ginzberg identify eight factors and Barki *et al.*, conducted a literature review that identified thirty-five risk factors. For instance, lack of management commitment is often cited for project failure (Newman 1996).

Software development methodologies attempt to reduce risk by gathering information and using structured processes. The implicit assumption is that by following a good methodology and identifying risk factors, software risk is reduced enough to avoid failure. However, a continuous stream of software failures may be a hint that there are risks that cannot be overcome by traditional approaches to software development. One key factor is the behaviour of decision makers using any methodology. Markus (1983) showed the importance of power and politics on software success, demonstrating how an individual's motivation can have a dramatic impact on the success of a systems implementation. Introducing a better software development methodology into such a situation would not likely have change the results. By contast, changes in personnel motivation had on overwhelming impact on success.

This article concentrates on the individual decision maker, his/her motivation, his/her willingness to take risks and how this behaviour is affected by relationships or contracts between employees and owners of an enterprise. The problem is framed as a principal agent problem where the owners of the organization, the principals, desire effective software development, and the software development team members, the agents, are asked to perform. Agency Theory is a management theory that puts the principle agent dilemma at center stage and may be a good way to understand the software development process and how to improve it. It has been applied to diverse management issues that have in common an agent who may have different interests than the principle being represented. Often the theory has been applied to CEO compensation, but it has recently been applied in diverse areas such as advertising agency contracting (Spake *et al.*, 1999) and franchising (Brickley, 1999). This paper applies agency theory to the problem of software failure with the assumption that software development has inherent risks. The paper is organized as follows. The following section introduces our research model. The next section deals with the risk propensity of individuals, followed by a section that describes how Agency Theory can be used to understand the motivational aspects of the individual decision makers. Eisenhardt's (1989) propositions will be applied to software development. A conclusion proposes several research directions based on the propositions in this paper.

Risk and Motivation

Software development requires members of the software development team to make a number of decisions including choices of technology, software development methods, usage of prototyping and inclusion of requirements. In order to make the best decisions individuals need to know how to make the best decisions and they must be motivated to make those decisions in the best interest of the organization. Much current research implicitly assumes that members of the project team are making decisions in the best interest of the organization and improvements to the software development process will come through development and implementation of better methodologies. However, putting better methodologies in the hands of individuals that are motivated by their own self-interest can have a detrimental effect. If individuals do not use their abilities in the best interest of the organization, software failure can be expected even with well-developed methodologies. The four authoritative approaches, Boehm's (1991), Davis's (1982), McFarlan's (1981), and Alter's and Ginsberg's (1978), have dominated discussions of risk management (Lyytinen *et al.*, 1998). Traditional approaches to manage risk consist of identifying risk and finding ways of controlling the risk. Applying modern software practices (Zmud, 1980), using good requirement analysis, version development and evolutionary development can help to reduce common known risks. However, none of these methods addresses the inherent nature of software development risk and the human nature of risk propensity. The following paragraphs discuss some of the methodologies and their shortcomings.

Classical decision theories that define risk as the variance between possible outcomes and model the decision process based on expected value of decisions do not accurately capture the real behaviour of management decision makers (March and Shapira, 1987). It is more accurate to describe them as being attentive to the magnitude of loss and seeking to avoid large losses (March and Shapira, 1987). With this in mind, large losses as perceived by a manager or another person on a software development team must be considered in light of the individuals' attention to individual vs. organizational goals and the decision maker's propensity to take risk.

Lyytinen *et al.*, (1998) have argued that a socio-technical approach considering actors, structure, task, technology, and their interactions integrates previous research on software development risk management and provides a richer description of the real world system. Although each is important and all must be in balance for best organizational function, the actors are quite central in the sense that they are making decisions both at the software development project level and at the organizational level, shaping their work tasks, structures, and technology. Further, in a professional environment, a number of types of actors, many of whom are not managers, are making "managerial decisions."

Always operating without complete information, managers and others involved in a software development project must make satisfying decisions. A problem in any organization is that managers may not always make decisions in the best interest of the organization, but rather may make them in their own interest. Project leaders, systems analysts, programmers, end-user, and others involved with software development may have different interests than the organization in which they work and different propensities for risk.

Figure 1 shows our model of how individual risk propensity and motivation effects project success or failure. Propensity to take risk is mitigated by control methodologies that are intended to prevent individuals from taking unwarranted risk. The type of contract or reward system mitigates the motivation. However, control methodologies may also affect the motivation of individuals and the rewards will affect propensity to take risks.

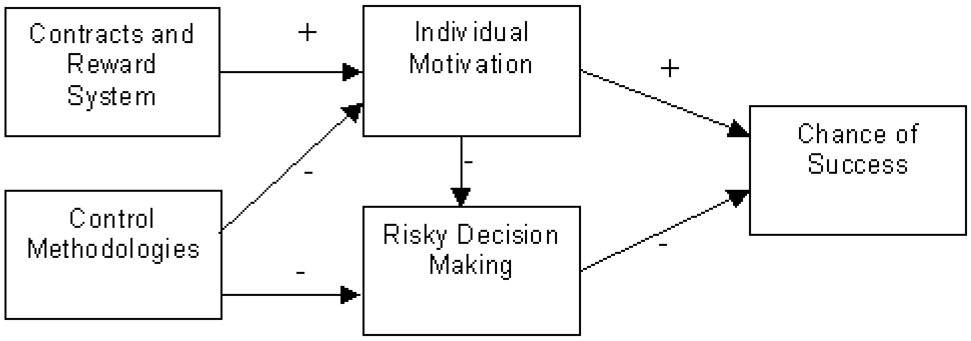

Figure 1: Model of individual behaviour on Failure/Success

Figure 1 shows a model of how individual decision making and motivation can effect the chance of a successful project. The model shows that risk decision making has a negative effect on the chance of project success. Traditional approaches to risk management have focused on controls and procedures that guide individuals away from risky situations or guide them in managing appropriate but risky choices. By reducing decisions to take bad risks, the chance of project success should be improved. However, if those controls have a negative effect on individual motivation, reducing either an individuals overall motivation or reducing the alignment of their interest with organizational goals, risky decision making could be increased. However, the reward system can be used to maintain motivation, allowing controls to be used effectively.

The next section will deal with the propensity of individuals to take risks and how it affects behaviour while the motivational aspect is discussed in the following section.

Propensity of Individuals to Take Risks

Risk analysis usually assumes that there is a constant risk for failure of software projects and that probabilities can be assigned to these failures. Methods are then suggested to reduce the risk of these potential failures. It is commonly believed that reducing uncertainty will reduce the risk of failure (Zmud, 1980). One of the factors which contribute to an increase in failure of software projects and which are not widely discussed in the literature is the propensity of people to take risks. Individuals confronted with the same risk often act very differently. One explanation is that different people have different utility functions. People may be considered risk neutral, risk averse and risk taking depending on their utility function. Consider three situations: a person has three choices, [1] receiving $50, [2] receiving $200 with probability 0.5 and losing $100 with probability 0.5, [3] receiving $ 2,000,000 with probability 0.0001 and losing $50 with probability 0.9999. The risk averse person would choose the $50 with certainty, while a risk taker would prefer the bet [3]. The risk neutral person would consider all bets essentially the same, as the expectation is about the same, namely $50. Confronted with a risk, individuals will balance the uncertain rewards of actions against the potential losses. Although, everyone has a propensity to take risks this propensity varies from one individual to another and is influenced by the potential rewards of risk taking, the perceptions of risk and the experience of failure or success. In general, the more risks a rational individual takes, the greater, on average, will be both the rewards and the losses he/she incurs. Adams (1999) presents this balancing act in a so-called risk thermostat (Figure 2). Rewards are influencing the propensity of a person to take risk as do failure and losses influence the perceived danger.

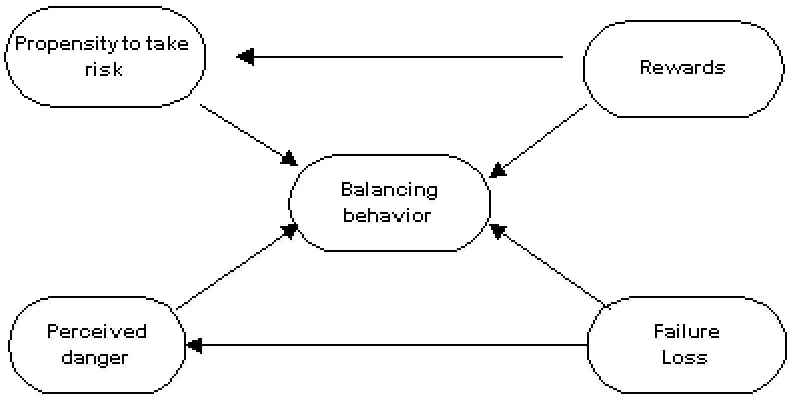

Figure 2: Risk Thermostat (Adams, 1999)

Thus a software project manger who has experienced failures may take less risk than a project manager who has little experience. Individual risk-taking decisions represent a balancing act in which perceptions of danger are weighed against propensity to take risks and the potential rewards and failure resulting in losses.

Every day, every individual must make decisions from driving to work, eating certain foods to investing in retirement to name a few. Every such decision involves balancing the rewards against uncertain losses. Individuals involved in software projects from managers to programmers perform the same balancing act. For instance, a manager will balance the reward of having a system in a short time against the risk of failure of a rushed project, or the promised rewards of a "comprehensive solution" against the risk of failure because of project complexity.

All risk models assume that there is some probability of failure. However, the estimation of probabilities is often difficult and subjective (Barki *et al.*, 1993). Probabilities are well defined in the mathematical literature. The following approaches are accepted, the classical method that derives probabilities from a model, the frequency based probabilities and subjective probabilities. The subjective probabilities however have the disadvantage of being not verifiable. Different individuals may derive different probabilities. Boehm (1991) for instance recommends using three intervals for the probabilities. Adams (1999), however, points out the scientific elusiveness of risk assessment. "*The clouds do not respond to what the weather forecaster say about them. People do respond to information about risks, and thereby change them.*" (Adams, 1999, p. 49). For instance, wearing a seat belt while driving a car unquestionable reduces the risk of a car crash being fatal. However, "*...as people perceive themselves as safer or better equipped against a danger, they are more likely to take more risks.* " (Adams, 1999). Hence, the probabilities for other risks are changing. For instance, the probabilities for injury and property damage crashes may actually increase because drivers take more risks. Similarly, individuals involved in software projects may change their propensity to take risks based on changes in perceived risk of failure. As more and more software development techniques are used and technology advances individuals may take more risk and thus change the probabilities of failure. Overconfidence in a new technology and the subsequent beliefe of invincibility may invite catastrophic failures.

To the extent that software development risks are analyzed and researched, it will alter the behaviour of people and thus change the risk because people will act upon this new information. For instance, just as more highway signs of dangers will make the motorist drive more carefully, established methodologies and risk management in software development will make failure less likely. However, people will also expect that all significant danger will be "signposted", thus increasing the potential risk of failure. Just as reduced perceived danger increases the risk taking, increased perceived danger may reduce propensity to take risk. For example, many dangerous roads have good accident records because people know that they are dangerous. In the same way known risks awareness in software development may reduce their probability of occurring.

Much research has focused on using methodologies to avoid failures by directly or indirectly controlling the risk behaviour of individuals. We may consider them as the "traffic laws" of software development. However, as crash statistics show over 80% of driver fatalities involve excessive risks taking such as alcohol, speed and not wearing seat belt. All of these factors are addressed by laws. Hence, individual behaviour not external factors are most often to blame. Making more laws (methodologies) may not decrease failure. Any improvement has to address the motivation of the individuals. In the following sections we discuss some of the common methodologies used in software development intended to reduce risk of project failure.

Requirements Analysis

Much emphasis has been placed in the information systems literature on developing complete requirements. Therefore project leaders often believe that gathering complete and consistent requirements can specify a system well enough that risks are brought into check. “Correct and complete information requirements are key ingredients in planning organizational information systems…” (Davis, 1982, p.4). However, correct and complete requirements are difficult for users to specify systems because of the complexities of systems and limitations in human information processing capabilities (Davis, 1982). Using a contingency approach based on uncertainty, a strategy for selecting a requirements determination methodology can be made (Davis, 1982). Although the technique describe can improve requirements elicitation, it improves the quality of the requirements that a user specifies and it ignores the problem that the user may not have the same interests or goals as the organization. Relying on complete requirement analysis may actually contribute to failure because of overconfidence and because of ignoring risks lurking to foil developers’ plans. Confidence in new technology and use of traditional methods may result in failure. For instance, implementations of enterprise resource planning (ERP) projects most often require dramatic redesigns of business processes. ERP projects may fail because they are solely based on a requirement analysis and the implementation team tries to adapt the ERP software to these “required” business processes ignoring what the software can do and cannot do.

We do not propose to ignore the importance of requirement analysis, however, overemphasis on requirement analysis may create a false security in the software developer which may contribute to increased risk.

Prototyping

Prototyping has been proposed as a method for reducing risks. It often cited as a way to develop better requirements. Prototypes in IS are often used differently than prototypes in other engineering disciplines. Prototypes in information systems development often show of proof of concept. In other words they show that a scaled down system can work. Users can view the concept and suggest changes thus improving requirements. However, this may not necessarily reduce risk of failure of the whole project. Since the system is a proof of concept and leaves out certain technical aspect, users may view the project as being less risky than it is. Conversely, many types of prototyping in engineering have the opposite goal. Rather than showing that a system works, they show the limitations of a system. In this concept engineers may stress a system until it breaks thus discovering the range of situations where the system can be expected to work. If that range is considered as a constraint in product development, risk can be reduced. Prototyping therefore, theoretically, can reduce or increase the risk of systems development failures. Only when risks are identified and prototypes are design in a way that will yield new information reducing such risk can they be helpful. Otherwise they may be contributing to increased risk.

Chief Programmer

The "chief programmer" (Zmud, 1980) is seen as a way to manage large software projects and to reduce project failures. However, this concept may have less value for an ERP project. A chief programmer may push a technical solution rather than a business solution. The responsibility for ERP projects rest with the line management because a successful implementation requires reengineering of business processes that are the responsibility of the respective managers. However, line managers often lack the understanding of information technology, and may take higher risks than appropriate.

Individuals have different levels of propensity to take risk and their decision can be seen as balancing behaviour between perceived danger of failure and rewards. Just as traffic laws have made roads safer, software development methodologies have made project failures less likely. However, fatal traffic accidents continue to occur, so do software failures. Much of this is due to individual risk taking in both cases. Increasing the emphasis on control methodologies to change risk behaviour is unlikely to be successful just as an increase in advertising the advantages of wearing a seatbelt may not change individual behaviour. Any method to further influence the risk behaviour of individuals has to address the motivation of individuals. The following section discusses the motivational aspect and introduces Agency Theory to affect individual behaviour.

Management and Motivation

> Agency theory is directed at the ubiquitous agency relationship, in which one party (the principal) delegates work to another (the agent), who performs that work (Eisenhardt 1989: 58).

In the arena of oftware development, owners of a firm would want software development projects to be successful. Agency theory addresses two problems; the first is the agent may have different goals than the principal and the second is that it is difficult or costly for the principle to monitor the agent. In the case of software development, the project leader and others become agents for the principals of the organization when they make decisions about a software development project. The agents make decisions about requirements, technology platforms, scope, expected profitability and potential risk of projects. It is quite difficult for owners or upper managers to judge the quality of those decisions because they do not have the information and cannot take the time to make a judgment on every decision the agents are making. It would not be surprising if a motivated software developer had a different preference towards using a bleeding edge technology solution than did senior managers. It would also not be surprising if that same developer had a different estimate of the risk of using such technology.

Agency theory assumes that agents are risk averse relative to the principle (Eisenhardt 1989). Since principles, usually identified as stockholders, can diversify ownership they are not averse to risk in any one particular holding. An agent, however, is likely to have only one significant form of employment and is therefore assumed to be risk averse relative to the principle. However, care must be taken when applying this assumption to agents involved in a software development project such as a systems analyst or an information systems manager. If the project is a failure, the individual analyst may not be affected much. Analysts knowing this may see certain decisions as risky for the organization, but with limited personal risk. Therefore, an analyst can be considered risk averse relative to the principal in their reliance on the source of income. However, this type of risk aversion may or may not lead to decisions that are of low risk to the project or the organization. One example of this may be in the choice of a new technology where people that know how to use the technology are in great demand. This new technology may be risky and a bad choice for an organization that has never used the technology. However, it may be a low risk decision to use the technology for an individual analyst who may increase personal employability and may not receive blame if the project fails.

Agency theory often only considers a single principle agent relationship such as that between stockholders and a CEO. Some work looks more broadly at organizational ownership at different levels such as CEO, management, and employee (Welbourne 1999). However, the software development team may be made up of several members who may be several management layers removed from the CEO. This leads to a complex web of relationships, each of which is similar to the classic principal-agent relationship.

In a software development project, each project member is an agent of the owners of the organization in similar fashion as the CEO in classical Agency Theory research. However, because people work in a social and political environment and do not generally report to the principals directly, they can be said to have a principal/ agent relationship with each other and management. In this sense, each person plays the role of a principal when they delegate or manage and they play the role of an agent when they take action to satisfy or not satisfy another person's goals.

Because of the number of agents positioned to affect project outcome, it may be useful to introduce the concept of agent type. In a software development project, agent types often discussed in the literature include project leaders, systems analysts, programmers, end-user, and mangers. These agent types may exhibit some characteristics in common with each other and may have characteristics unique to the type. Either way, empirical research on Agency Theory and software development should include this distinction.

Beath and Orlikowski (1994) note that end-users must sign off on specifications developed by systems analysts and take responsibility for project outcomes. They note a conflict between the analysts being responsible for many of the decisions while the end-user takes responsibility for project outcome. In this relationship, the end-user is the principal and the agent is the systems analyst. Agency theory predicts a similar misalignment of relationship. Beath and Orlokowski (1994) suggest that the software engineering methodology can create an unhealthy dominant role in the systems analyst. The agency perspective reframes the role with the end-user as principal and the systems analyst as the agent with inherent uncertainty in the relationship. In a principal agent relationship, an assumption is that the agent has more knowledge.

Although a number of actors can be identified, four major archetypes define many of the players. The **management actor** type represents the manager as the major steward of organizational goals who is closest to the principals of the organization. Managers can be seen as agents for the principals of the organization but also as a principal in relationships with **systems analysts** and **end-users** setting up two principal agent types of relationships. Generally there is also collaboration between end-users and systems analyst where end-users need to provide requirements to systems analysts who use those requirements to develop a system to be used by the end-user. This is another relationship resembling the principal agent model where the systems analyst is an agent for the end-user. Figure 3 summaries these four major actor types and show their principal agent relationship. Because these entities represent types, rather than individual actors, their instantiation in a real organization may be through several individuals with some individuals acting as more than one type. For example, there is often more than one systems analyst and a project leader may at one time be acting as a management type in relationship to systems analysts and at another time be acting as systems analysts in relationship to management or end-user types.

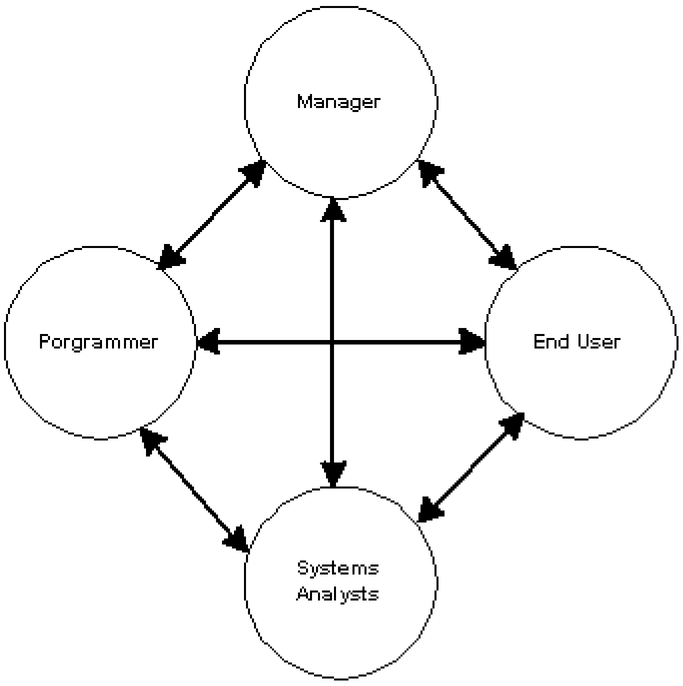

Figure 3. Agency Influence Model

Figure 4 shows the causal relationship implied by agency theory in relationship to that implied by traditional software development research. If we frame the process of creating successful projects as one in which project will be successful when those involved make the best decisions possible for how to undertake the projects. The upper row of Figure 4 describes a simplified version of the causal theory that underlies much research on software development methods. It says that if we develop better methods, the team will work better, and there will be greater project success. However, as we have argued, an unmotivated team may not use software methods effectively. Figure 4. describes high quality decision making has the *product* of knowledge of methods and motivation to use them. Agency Theory implies that monitoring and contracting are important influences on motivation.

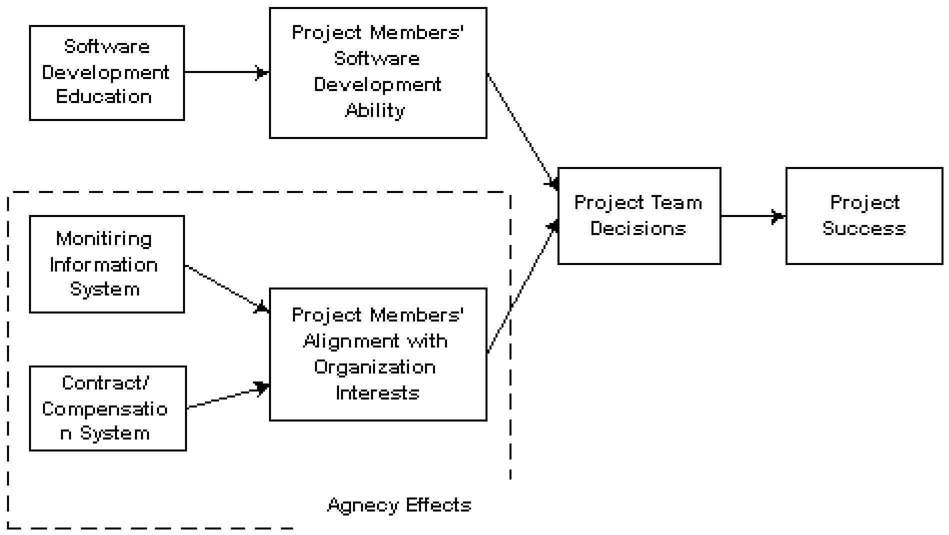
Software Development Education
Project Members' Software Development Ability
Project Team Decisions
Project Success
Monitiring Information System
Contract/ Compensatio n System
Project Members' Alignment with Organization Interests
Agnecy Effects

Figure 4. Project Success Model

The causal relationship described in Figure 4 is in contrast to the causal structure implied by most recent research on software development. Past work has focused on the skills and techniques needed to improve software development success, but has not addressed individuals' motivation to use them. Although skills are a critical part of improving software development success, a lack of motivation to apply those skills in the best interest of the organization can negate any skill effect.

The Workshop on IT Risk Management (held at ICIS 1998) developed a four level IT Risk Management Model that included strategy, planning, management, and operational levels of risk management. Using the four level lens we can apply agency theory at each level enriching our understanding of the rich web of relationships and motivations surrounding the software development process.

At the strategy and planning levels, agency theory can be applied in a similar way as it has been often applied to the CEO management issue. These levels apply broadly across several software development projects and can b addressed at a macro level. At these levels, organizational policy on employee ownership, outsourcing, and contracting are relevant. However, at the management and operational levels, a more novel a version of agency theory needs to be developed.

The next section adapts Eisenhardt's (1989) propositions to the software development function.

Eisenhardt's Propositions

Eisenhardt (1989) developed ten propositions of Agency Theory. These propositions suggest under what circumstances behaviour-based vs. outcome-based contracting should be used. This implies, in information systems development, the form of contracting that minimizes the agency and monitoring costs would be most effective in software development projects. Extending the idea, proper contracting and monitoring should also reduce the chance of software development failure. A first step in an agency theory software development research stream would be to test the relevance of these propositions. If they are found to be relevant, then implications of the theory could be applied to improving software development. The following is a list of the propositions and their implications for software development.

> **Proposition 1**: When the contract between the principal and agent is outcome based, the agent is more likely to behave in the interest of the principal. (Eisenhardt, 1989: 60)

Proposition 1 can apply to software development in terms of the contracts with the various agent types. It addresses the motivation of the of the software developer which are increased by owning part of the success. When contracts with agents in the software development process are outcome based, the agents should make decisions in the interest of the principal. This implies that when the software developers and managers have ownership of the outcome, then surprising software failures should be lower. For instance, if the IT department were a type profit center which is rewarded for success rather than for work performed, software failures may be lower. This may also imply that some types of outsourcing might be effective when the outsourced agents are working under an outcome based contract.

> **Proposition 2**: When the principal has information to verify agent behaviour, the agent is more likely to behave in the interests of the principal. (Eisenhardt, 1989: 60)

Proposition 2 implies that when the principal has information on the behaviour of the agent, then the agent will behave in the interest of the principal because the principal will observe the behaviour and reward appropriately. Although this proposition should hold for software development, behaviour of the software development team can be quite difficult to observe unambiguously. For example, a user representative may specify requirements that make the job of that individual easier, but are detrimental to the larger business process. This would only be observable by a principal that knew the implications of the requirement and was aware of the decision to include the requirement.

In general, an organization with a defined software development methodology including definition of responsibilities of the development team positions will have better information on behaviour and should experience less software failure.

Proposition 3: Information systems are positively related to behaviour-based contracts and negatively related to outcome-based contracts. (Eisenhardt, 1989: 61)

Note that Eisenhardt uses the term "information systems" as a generic description of systems gathering information about employees. Proposition 3 implies that increased monitoring of project members behaviour is used when contracts are behaviourally based than outcome based. This could be observed in the relationships an organization has with contract programmers that can cover most of the range from hourly-based [behavioural] to project-based [outcome].

One interesting implication related to Proposition 3 is that when outcome-based contracting is used, the need to require certain control methodologies is reduced. Closely monitoring requirements analysis may not be necessary when the development team is compensated with an outcome-based contract or when they have ownership in the organization. Proposition 3 implies that organizations using outcome based contract will make less use of standard software development policies since one role of a software development policy is to control behaviour that is less needed if outcome based contracting is used.

Proposition 4: Outcome uncertainty is positively related to behaviour-based contracts and negatively related to outcome-based contracts. (Eisenhardt, 1989: 61)

When outcome uncertainty is high, it is difficult to transfer the risk to the agent, as an outcome-based contract would do. Careful definition of outcome must be done for this to be measured empirically. If outcome were defined as a system implemented and providing the planned benefit, then risk may be relatively high. If outcome is defined as developing a program module that meets specification, then risk may low. Therefore, outcome-based contracts should be expected to be seen to a greater extent for well-defined deliverables that may be only part of a full project.

Proposition 5: The risk aversion of the agent is positively related to behaviour-based contracts and negatively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Proposition 5 implies that the risk aversion level of agents on a software development team will be positively related to behaviour-based contracts. Outcome-based contracts transfer risk to the agent. If an agent is risk averse then the cost of transferring risk to the agent is relatively high, favouring behaviour-based contract.

Proposition 6: The risk aversion of the principal is negatively related to behaviour-base contracts and positively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Conversely, Propositions 6 implies the risk aversion of the principal favours transferring risk to the agent through outcome-based contracts.

Proposition 7: The goal conflict between principal and agent is negatively related to behaviour-based contracts and positively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Proposition 7 implies that when the goals of the principal and the agent diverge there should be a tendency to use outcome-based contracts. In software development using contractors, it would follow that the interests of the contractor and the contracting organization may well diverge and outcome-based contracts should be more effective. When software development is undertaken within the firm, goal conflict may still be high, but in this case behavioural contracts are common suggesting that internal software development may have a lower success rate than external development.

Proposition 8: Task programmability is positively related to behaviour-based contracts and negatively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Proposition 8 implies that the more programmable the task the greater propensity for using behaviour-based contracts. In software development, the use of standardized software development procedure makes tasks more programmable. Organization with mature software development models should favour behaviour-based contracts. However firms tackling novel applications of software that have immature development models may be successful with outcome-based contracts.

Proposition 9: Outcome measurability is negatively related to behaviour-based contracts and positively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Outcome measurability is the ease of which an outcome can be measured in a timely manner. In the case of a short-term small-scope project, outcome measurability may be relatively high. On a large project, outcome measurability may be relatively high and results may not be measurable in a timely manner.

> **Proposition 10**: The length of the agency relationship is positively related to behaviour-based contracts and negatively related to outcome-based contracts. (Eisenhardt, 1989: 62)

Proposition 10 implies that principal learns about agents over time and can therefore better assess behaviour reducing the need for outcome-based contracts.

Eisenhardt's propositions shed new light on the software development process and its success or failure. These propositions address the motivation of the developer and manager of software projects. The likelihood of success of software projects may be increased when the developers and managers have some ownership of the outcome. This may be achieved through outcome based contracts, type of outsourcing or type of profit center. Having the three parties, management, systems analyst, and user, shown in Figure 3, share responsibility for outcome rather than for work done can be expected to reduce failures in software projects. In any case the propositions suggest that empirical research should be done to determine the effect of motivation in general and outcome-based contracts specifically on software projects' success.

Conclusion

Much research in software development has focused on risk management identifying risk factors and developing methodologies to prevent these factors from resulting in software project failures. However, despite the many advances in software development methodologies, software failures still occur and may even have increased. One issue not addressed in the information systems literature very much is that people often insist that they want to be their own risk managers and thus balance their individual risks and rewards. Motivations and self-interest may also contribute to software project failures. Thus, modern software development methodologies by themselves do not guarantee project success. Agency theory provides a framework for understanding the relationship type of contracts and project success. Empirical research needs to be conducted into outcome based software contracts versus behaviour-based contracts in information technology development. Specifically the following hypotheses derived from the propositions are of interest. [1] Companies where employees have outcome-based contracts have more software project successes than companies where contracts are based on behaviour. [2] Companies with outcome-based contracts use fewer software project controls than companies with behavioural contracts. [3] In companies with outcome-based contracts employees create a better balance between risks and rewards in software development than in companies with behavioural contracts.

References

- Adams, J. (1999) "Cars, cholera, and cows, the management of risk and uncertainty." *Policy Analysis*, No.335. Washington, DC: Cato Institute.
- Alter, S. and Ginzberg, M. (1978) "Managing uncertainty in MIS implementation," *Sloan Management Review*, **20**(1), 23-31.
- Barki, H., Riverd, S., and Talbot, J. (1993) "Toward an assessment of software development risk," *Journal of Management Information Systems*, **10**(2), 203-225.
- Beath, C.M., & Orlikowski, W.J. (1994) "The contradictory structure of systems development methodologies: deconstructing the IS-user relationship in information engineering." *Information Systems Research*, **5**(4), 350-377.
- Bennett, J.C., Bohoris, G.A., Aspinwall, E.M., & Hall, R.C. (1996) "Risk analysis techniques and their application to software development," *European Journal of Operational Research*, **95**(3), 467-475.
- Boehm, B.W. (1991) "Software risk management: principles and practices." *IEEE Software*, **8**(1), 32-41.
- Brickley, J.A. (1999) "Incentive conflicts and contractual restraints: evidence from franchising," *Journal of Law and Economics*, **42**(2), 745-774.
- Davis, G.B. (1982) "Strategies for information requirements determination," *IBM Systems Journal*, **21**(1), 4-30.
- Eisenhardt, K.M. (1989) "Agency theory: an assessment and review," *Academy of Management Review*, **14**(1), 57-74.

- Lyytinen, K., Mathiassen, L. & Ropponen, J. (1998) "Attention shaping and software risk: a categorical analysis of four classical risk management approaches," *Information Systems Research*, **9**(3), 233-255.
- March, J.G. & Shapira, Z. (1987) "Managerial perspectives on risk and risk taking," *Management Science*, **33**(11), 1404-1418.
- Markus, M.L. (1983) "Power, politics and MIS implementation," *Communications of the ACM*, **26**(6), 430-444.
- McFarlan, F.W. (1981) "The portfolio approach to information systems," *Harvard Business Review*, **59**(5), 142-150.
- Newman, M. (1996) "Determinants of commitment to information systems development: a longitudinal investigation," *MIS Quarterly*, **20**(1), 23-54.
- Spake, D.F., D'Souza, G., Morgan, R.M., & Crutchfield, T.N. (1999) "Advertising agency compensation: an agency theory explanation," *Journal of Advertising*, **28**(1), 53-72.
- Welbourne, T.M., and Cyr, L.A. (1999) "Using ownership as an incentive: Does the "too many chiefs" rule apply in entrepreneurial firms?" *Group Organization Management*, **24**(4), 438-460.
- Zmud, R. W. (1980) "Management of large software development efforts," *MIS Quarterly*, **4**(1), 45-55.

How to cite this paper:

Walsh, K.R., & Schneider, H. (2002) "The role of motivation and risk behaviour in software development success" *Information Research*, **7** (3) [Available at http://InformationR.net/ir/7-3/paper129.html]

Check for citations, using Google Scholar

Contents 26160 **Web Counter** **Home**