View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Harvard University - DASH

DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD

Designing for Incentives: Better Information Sharing for Better Software Engineering

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters.

Citation	Klein, Mark, Gabriel A. Moreno, David C. Parkes, and Kurt Wallnau. Forthcoming. Designing for incentives: Better information sharing for better software engineering. Proceedings of FSE Workshop on the Future of Software Engineering: November 7-8, 2010, Santa Fe, New Mexico.
Accessed	February 18, 2015 9:38:09 PM EST
Citable Link	http://nrs.harvard.edu/urn-3:HUL.InstRepos:4434400
Terms of Use	This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

(Article begins on next page)

Designing for Incentives: Better Information Sharing for Better Software Engineering

Mark Klein
Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213
mk@sei.cmu.edu

Gabriel A. Moreno
Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213
gmoreno@sei.cmu.edu

David C. Parkes
Harvard SEAS,
Cambridge MA 02138
parkes@eecs.harvard.edu

Kurt Wallnau
Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213
kcw@sei.cmu.edu

ABSTRACT

Software-reliant systems permeate all aspects of modern society. The resulting interconnectedness and associated complexity has resulted in a proliferation of diverse stakeholders with conflicting goals. Thus, contemporary software engineering is plagued by incentive conflicts, in settling on design features, allocating resources during the development of products, and allocating computational resources at runtime. In this position paper, we describe some of these problems and outline a research agenda in bridging to the economic theory of mechanism design, which seeks to align incentives in multi-agent systems with private information and conflicting goals. The ultimate goal is to advance a principled methodology for the design of incentive-compatible approaches to manage the dynamic processes of software engineering.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Economics, human factors

1. INTRODUCTION

The practice of software engineering typically involves multiple communities of interest, and requires that delicate tradeoffs be struck involving resource allocation and design decisions, and involving compromises between various stakeholders. Software engineering is a very people-intensive discipline [4, 17]. Despite being also a highly technical discipline, most project failures in software engineering are not due to technology issues but to people issues such as poor project management [11], and conflicting stakeholder interests [7]. Members of these communities are quite naturally driven by numerous forms of self-interest; promotion, salary increases, profit sharing, recognition, and technical inquiry and discovery are several examples. Mechanism design (MD), which aims directly at promoting truthful information sharing for the purpose of coordinated decision making, is ideally suited to these types of issues. The challenge is to adapt the theoretical framing provided by MD to the practice of software engineering.

Much of software engineering has typically been done in a value-neutral way [3].[1] It is only recently, and in recognition that the ultimate goal of software engineering is to deliver value to different stakeholders, that a research agenda of *Value-Based Software Engineering* (VBSE) has been advanced [3, 2]. VBSE seeks to integrate value considerations into the software engineering practices.

But VBSE does not directly address the challenge of strategic behavior, and the assumptions that stakeholders will be able to cooperatively negotiate may break down in cases of stakeholders with conflicting interests [8]. For example, users may want better performance whereas other stakeholders may want to reduce the cost. Furthermore, stakeholders often have information that affects decisions that they care about [9], and there will then exist an incentive to misreport this information.

In this paper, we outline a number of the incentive problems that arise in this multi-faceted process of software engineering. Our aim is to make the connection with mechanism design, which seeks system-optimal outcomes through the careful alignment of incentives. Ultimately, we seek to advance the following challenge problem:

Develop incentive-aligned software engineering processes such that stakeholders will choose to participate in continual and truthful information sharing that will lead to decentralized but optimal decision making.

Mechanism design provides a toolkit for eliciting prefer-

[1]For instance, so-called "earned value analysis," a method for tracking project progress, is based on the cost of project resources consumed relative to schedule and not on the value of the current developed features to the customer.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

ence information from self-interested agents, in order to support useful system-wide decisions [10]. But it is no panacea. A significant research effort will be required by the scientific community if robust tools, based around the theories of MD, are to have an impact on software engineering. Progress may also require a significant re-engineering of development processes, for instance through the wider adoption of performance-based pay and through steps to sustain competition throughout development processes.

By way of illustration, we include a simple example of a method by which an incentive payment scheme can promote truthful information sharing about the resource requirements for different technical features in a product. Incentive considerations come into play because of joint constraints on the resource profile of a developed product and different teams jockeying to get their own best-of-breed techniques into the product.

Other than this, our main focus is given to explaining some of the research problems that exist in making progress on this grand challenge facing our field. These problems arise because of the complexity of managing the process of software engineering, which is inherently dynamic, uncertain, and with interconnected decisions.

1.1 Related Work

A small literature exists on market-based approaches to problems of design, both from the perspective of proposing conceptual frameworks for thinking about design and in terms of supporting a decision-making process. For example, Wellman [18] studies a computational market model for "design economies" in which design problems are formalized as problems of constrained optimization and solved via market-based algorithms. The computational market could in principle be extended to represent the preferences of multiple stakeholders, but was studied only in the context of a single customer and thus one set of preferences and avoiding the need to address any incentive challenges. Parunak et al. [15] introduced the *MarCon* algorithm for the support of a design process between multiple human product designers, but without providing any formal incentive properties. Both papers contribute the framing of preference elicitation and constrained optimization, but consider only static problems in which participants commit to a one-time decision about the design of an engineering artifact.

2. WHY WORRY ABOUT INCENTIVES?

Software engineering problems are traditionally situated within an organizational structure that consists of multiple communities of interest, whose interests are collected by a development organization, which in turn delegates the job of developing parts of the system to suborganizations. In what follows, we will often associate the communities of interest with *product consumers* (PCs), the development organization with *product managers* (PMs) and the suborganizations with *product producers* (PPs). We consider in turn some of the incentive conflicts that arise between competing suborganizations, competing communities of interest, and then taking both considerations together.

2.1 Competing Suborganizations

A single development group is tasked to develop a product. The group is led by a chief architect (PM) who needs to elicit technical information from the experts (PPs) who work for her. The chief architect is responsible for making optimal tradeoff decisions, considering the available resources (e.g., development hours, or competition for CPU cycles at run-time between different features) and the effect on the overall value of the product.

For example, one PP may be a security expert and another a video expert. In this case, the *feature set*, which is the set of possible features that could be included in the design, would comprise various levels of security and video quality. The CPU usage at run-time may be the constrained resource, with higher levels of security or higher video resolution requiring a larger share of CPU utilization in the designed product.

To model this, we associate the PM with a value function for the developed product that depends on the combination of feature levels, with $v_{\rm pm}(y) \geq 0$ denoting the value for different feature levels $y = (y_1, y_2)$, with $y_1 \in Y_1$ and $y_2 \in Y_2$ denoting the levels of features 1 and 2 respectively (and Y_1 and Y_2 denoting the possible feature levels). Let F denote the set of feasible resource allocations to features 1 and 2, so that $x = (x_1, x_2) \in F$ indicates the resource allocated to each of features 1 and 2.

For each PP, $i \in \{1, 2\}$, there is a *technology production function*, $y_i(x_i) \in Y_i$, which defines the feature level achievable given resource allocation x_i. Given this, the design objective is to allocate resources (and thus select a design) to maximize the total value of the product:

$$\arg \max_{(x_1,x_2)\in F} v_{\rm pm}(y(x)),$$

where $y(x) = (y_1(x_1), y_2(x_2))$. The PM must make a tradeoff across different feature levels in achieving a product that meets resource constraints while optimizing end value. The challenge is that we assume that the technology production function is private to the PPs.

One way in which incentive issues may arise is when each PP has an intrinsic value for the technical quality of work performed, and thus the feature level selected in the optimal design. Based on this, a PP might benefit from overrepresenting the resource requirement for a less interesting feature level in order to promote the selection of a more sophisticated feature level for the product. Alternatively, a PP might overstate the required resources in order to be given an easier task and to be able to shirk work.

For a related example in the context of software development across organizational boundaries, we can consider a contractor (PM), who is tasked to develop a product and subcontract to development organizations (PPs). Each subcontractor has private information about its expertise and costs, while the contractor wants to allocate budgetary resources to maximize the value of the final product.

2.2 Competing Communities of Interest

Turning to the question of different communities of interest, consider now a single product that is being developed by a firm (PM) with multiple end customers (PCs), each of whom wants a product with a specific set of features.

It stands to reason that each community of interest can stand to benefit by overstating the importance of certain features, while understating the importance of other features that will be requested in any case by another community. To model this, we can think about two PCs, $j \in \{1, 2\}$, each with a different value $v_j(y) \geq 0$ for feature levels $y = (y_1, y_2)$. Given a finite budget for any given product release,

the PM must select feature levels that maximize joint value while remaining on budget. Let B denote the set of feasible feature levels, such that the project will remain in budget. Given this, one design objective is to select feature levels that maximize the total value of the PCs:

$$\arg \max_{(y_1,y_2)\in B} v_1(y) + v_2(y)$$

In the case that the budget constraint is private, the PM may also face a strategic decision in deciding how forthcoming to be in revealing the possible tradeoffs.

For a related example within a single organization, we can consider a contractor (PM) who is developing a single product to serve multiple communities of interest (PCs) within the organization. Joint programs in the U.S. Department of Defense are examples of this situation. For example, the Joint Tactical Radio System (JTRS) has the mission of developing a family of interoperable software defined radios for the Joint Forces (Army, Marine Corps, Navy, and Air Force) [12]. The existence of diverse requirements is one reason why these projects are difficult to complete on budget and on time and with a feature set that is effective in meeting the needs of stakeholders [5].

2.3 Competition from Both Sides

In general, we see competition amongst *both* suborganizations and communities of interest in a single software engineering project. For example, a PM within one firm may consider the demand for features from each of multiple market segments while simultaneously handling the technical wishes of competing development teams. Similarly, a contractor may be trying to balance the competing requirements from upstream communities of interest and the competing capabilities and interests of downstream subcontractors. Now the incentive conflicts are between all parties, with the product manager or contractor needing to elicit missing information and reconcile the possibly misaligned requirements of many different stakeholders. An additional challenge is that a typical software engineering process is *dynamic*, with decisions made and refined continually as new information comes to light about technical difficulties and revised requirements of customers and other groups.

3. ILLUSTRATION: INCENTIVE DESIGN FOR COMPETING SUBORGANIZATIONS

Consider again the allocation of the fraction of a shared CPU resource for the use by competing video and security features. With more resource allocated to video, the PP representing the video technology can deploy a more sophisticated form of video processing technology.

Table 1 shows the PM's value for different possible video and security levels. Table 2 shows the production functions of each PP. The minimal resource requirements, as indicated in bold, are assumed to be known to the PM. But otherwise the technical data in regard to video and to security is private to the video and security PPs, respectively. The optimal design, denoted $y^* = (y_1^*, y_2^*)$ will allocate 0.5 CPU to each feature, providing *M-level* video and *512-level* security and a value for the PM of 85.

We will assume that it is possible to provide payments to each PP that depends on the value of the final product developed and the incremental value that is contributed by the PP.

Table 1: PM's value function

		security			
		128	**256**	**512**	**1024**
video	**L**	20	40	60	65
	M	45	65	85	90
	H	55	75	95	100

Table 2: PPs' technology production functions

CPU	*video*
0.3	L
0.5	M
0.7	H

CPU	*security*
0.1	128
0.3	256
0.5	512
0.7	1024

A first idea is to make payment $v_{\rm pm}(y^*) - v_{\rm pm}(y_i^0, y_{-i}^*)$ to each PP i, where y_i^0 is the minimal feature level associated with PP i. This payment is the amount by which the PM's value is greater under the optimal solution (which benefits from the expertise of PP i) than when y_i^* is substituted for the feature level achieved when i receives its minimal resource level. Notation y_{-i}^* simply means the feature level selected by the other PP (i.e., by the PP other than PP i.)

But this leads to an incentive problem whereby each PP will seek to contribute a higher-level of its own feature than is optimal overall, for example by overstating the resource requirements of simpler feature levels. When reporting true technology functions, the payments are 25 (=85-60) and 40 (=85-45) to the video and security PPs respectively. However, the video PP can misreport her production function, and claim that *M-level* video requires 0.6 CPU. In this case, the optimal allocation would be (0.7,0.3) CPU to video and security respectively, with a reduced total value for the PM of 75. On the other hand, this is effective for the video PP, who obtains a higher payment of 35 (=75-40).

There is a well-known solution to this problem. One can use the payment of the *Vickrey-Clarke-Groves* (VCG) mechanism [10] to align incentives by providing transfer:

$$v_{\rm pm}(y^*) - v_{\rm pm}(y_i^0, \hat{y}_{-i}),$$

where $\hat{y}_{-i}$ is the optimal feature level for the other PP given that y_i^0 is adopted for PP i. That is, rather than receiving payment equal to the additional value contributed while fixing the feature level associated with the other PPs, the payment is equal to the additional value for the design compared to the design that would be picked if the level associated with PP i was the minimal level.[2]

The effect on the example is that the payments, when truthful, are 20 (=85-65) and 30 (=85-55) to the video and security PPs respectively. Now, if the video PP misreports her production function as before, then she would be hurt by a reduced payment of 10 (=75-65).

4. THE RESEARCH CHALLENGE

[2]To see that this aligns incentives, notice that the second payment term $v_{\rm pm}(y_i^0, \hat{y}_{-i})$ is independent of anything reported by PP i, while the first term aligns agent i's incentives with maximizing the overall value $v_{\rm pm}(y)$ of the product. To achieve this, a PP i should report the true technology production function whatever the report by the other PP.

Making design and resource allocation decisions requires sharing information that is privately held by different stakeholders. For a developer, this information includes the current state of progress towards achieving functional and quality goals and beliefs about the probability of achieving those goals by different dates, and about the responsiveness of these estimates to additional resources or changes to the requirements. For a program manager, this includes the current state of resource allocation (e.g., developer time, fraction of CPU, etc.) and beliefs about the probability with which the availability of resources will change during the lifetime of the project, including the probability of competition for resources with other projects. For a customer, this includes the current estimate of expected value for different combinations of features and also beliefs about the probability with which these requirements will change as the product is developed and the customer's understanding of different features and actual needs is refined.

Success in applying the methods of MD to software engineering will require that we are able to craft simple ways to elicit this kind of information from different stakeholders. Success will also require the adoption of outcome-dependent payment schemes, along with new governance methods. Finally, success will require a new willingness to retain competition across different development organizations for the bulk of development, rather than having a single competitive tender that quickly whittles the field down to a single developer. This, in turn, will lead to an increased emphasis on open standards so that different organizations can fluidly switch in and out of a project.

In adopting MD to facilitate effective coordination in software engineering, we will also need progress on the following technical research problems:

Problem 1. *Handling PPs with intrinsic values while respecting payment norms in organizations.* For example, a PP could associate value with building best-of-breed components. The existence of private information on both sides of a market, for example with PPs and PMs both having value functions for different feature levels, takes the VCG mechanism off the table. This is because it may require payments from PPs to the PM, rather than in the other direction. But it seems unnatural for payments to be made in this direction, except perhaps when the PPs are independent contractors and looking for tasks in order to gain experience and/or reputation. For example, suppose that there are levels L and H for feature 1 and the values are (0,20), (5,5) and (15,10), for level L and H for each of the first PP, second PP and the PM respectively. The optimal decision is level H, with total value 35. In the VCG mechanism, the first PP would be asked to make a payment of 5 for its influence on this outcome. This is the smallest $z \geq 0$, for report $(0, z)$, that it could make and still have the choice be H instead of L.

Problem 2. *Handling complex value interdependencies between the features of multiple modules while retaining simple methods for expressing preferences.* Software engineering often has to deal with interdependencies between features provided by different components, where the value of a feature depends on the features provided by another component, and vice versa (e.g., authentication and authorization). This provides additional complexity because coordinated decision making is required across components, based for example on value statements by developers for the features provided by their own module and by other modules. It seems likely that the modularity that is essential in developing manageable and robust code bases is useful here; e.g., in allowing succinct representations of the valuations of different modules (since the value may only depend in a detailed way on the spec of a small number of other modules.) But it is clearly a major challenge to develop simple, yet expressive methods by which developers can share information about value interdependency between modules.

Problem 3. *Handling uncertainty and dynamics: enabling continual information sharing throughout the software engineering process.* Software engineering is a dynamic process. Decisions are made and continually refined as new information comes to light about requirements, technical difficulties, and competencies of core team members. In fact, an important consideration in estimating the value of a module is the extent to which it enables experimentation and creates the option of changing its internal details in the future [16]. Although traditionally developed for static systems with one time decision making, MD has been extended in recent years to promote continual information sharing and decision making in dynamic systems [13, 14, 6, 1]. On the other hand, there remain many outstanding research problems in extending these methods so that they scale to the complexity presented by software engineering processes. For example, methods that extend the VCG mechanism to dynamic contexts rely on correct probabilistic models (e.g., about how a customer's requirements might change, or how long a particular task might actually take to complete). These methods also rely on being able to compute an *optimal* decision policy within the confines of these probabilistic models. But the problems can quickly become intractable due to the curse of dimensionality that affects multi-faceted systems with the kinds of interactions between different modules that exist in software engineering. In addition, another source of value could be allowing team members to gain new skills through doing new things, or promoting the design of modules for reuse, so that a decision that considers only the needs of a current project can be overly myopic.

5. CONCLUSION

In this paper we have presented different incentive issues that arise, both within and across organizations, because of the competing interests of the multiple stakeholders involved in the process of software engineering. These problems will continue to increase in the future due to the ever increasing interactions between development organizations, consumers, and systems. We take as self-evident the commercial importance of developing effective methods by which to coordinate the process of software engineering, along with the obvious failings of current practice.

Mechanism design provides a framework and a set of results that, in theory at least, seem suitable to achieving continual, truthful sharing of information for the purpose of coordinated and adaptive decision making. But the research challenge that faces the community is large: not only will ultimate success require continued progress in the theory of mechanism design, but ultimate success will also depend crucially on the ability to develop productivity and decision-support tools that succeed without placing overly onerous requirements on developers, program managers, and customers in explaining the kinds of information that is currently held private and only selectively revealed.

6. REFERENCES

[1] D. Bergemann and J. Välimäki. Efficient dynamic auctions. Technical Report Cowles Foundation Discussion Paper No. 1584, Yale University, 2006.

[2] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, editors. *Value-Based Software Engineering*. Springer-Verlag, 2006.

[3] B. Boehm. Value-based software engineering. *ACM SIGSOFT Software Engineering Notes*, 28(2):3, 2003.

[4] B. Boehm and R. Ross. Theory-W software project management principles and examples. *IEEE Transactions on Software Engineering*, 15(7), 1989.

[5] M. M. Brown, R. M. Flowe, and S. P. Hamel. The acquisition of joint programs: The implications of interdependencies. *CrossTalk: The Journal of Defense Software Engineering*, pages 20–24, May 2007.

[6] R. Cavallo, D. C. Parkes, and S. Singh. Efficient mechanisms with dynamic populations and dynamic types. Technical report, Harvard University, 2009.

[7] T. DeMarco and T. Lister. Risk management during requirements. *IEEE Software*, 20(5), Sept.-Oct. 2003.

[8] P. Grünbacher, S. Köszeqi, and S. Biffl. Stakeholder value proposition elicitation and reconciliation. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, editors, *Value-Based Software Engineering*, pages 133–154. Springer-Verlag, 2006.

[9] M. Halling, S. Biffl, and P. Grünbacher. The role of valuation in value-based software engineering. In *6th International Workshop on Economics-Driven Software Engineering Research Proceedings*. IEE, 2004.

[10] M. O. Jackson. Mechanism theory. In U. Derigs, editor, *The Encyclopedia of Life Support Systems*. EOLSS Publishers, 2003.

[11] C. Jones. Software project management practices: Failure versus success. *CrossTalk: The Journal of Defense Software Engineering*, October 2004.

[12] R. North, N. Browne, and L. Schiavone. Joint Tactical Radio System - connecting the GIG to the tactical edge. In *IEEE Military Communications Conference, 2006. MILCOM 2006.*, Oct. 2006.

[13] D. C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[14] D. C. Parkes and S. Singh. An MDP-based approach to Online Mechanism Design. In *Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS'03)*, 2003.

[15] H. V. D. Parunak, A. Ward, and J. Sauter. The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM)*, pages 217–234, 1999.

[16] K. Sullivan, P. Chalasani, S. Jha, and V. Sazawal. Software design as an investment activity: A real options perspective. In L. Trigeorgis, editor, *Real Options and Business Strategy: Applications to Decision Making*, chapter 10. Risk Books, 1999.

[17] J. E. Tomakyo and O. Hazaan. *Human Aspects of Software Engineering*. Charles River Media, 2004.

[18] M. P. Wellman. A computational market model for distributed configuration design. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM)*, pages 125–133, 1995.