Using Electronic Markets to Achieve Efficient Task Distribution

Ian Grigg Christopher C. Petro

28 February 1997

Abstract: The Internet was built using the efforts of a worldwide team of programmers that coordinated and competed through *laissez-faire* methods. Much of the effort was freely provided, or paid for by entities in a process that did not conform to normal commercial revenue-seeking or government regulatory behaviour. This points to major inefficiencies in the market for software. One inhibitor is the large search costs undertaken by managers to acquire new programmers.

On the other hand, there are inherent inefficiencies in the way in which much of the free Internet software is developed. Specifically, there is no efficient way for users to direct the efforts of developers, other than by contracting for entire projects. This often results in a mismatch between development and requirement, as user communities and developer communities are sufficiently culturally different to make communication non-perfect.

We propose a market-based solution that allows many users to each contribute small amounts to projects, and for the sum effect of these contributions to influence and direct the activities of programmers towards tasks that users demand. A range of solutions is presented, from a web billboard bounty market to trading exchange markets for digital financial instruments. Reputational effects, intermediaries and differentiation are considered.

Relying on the existance of efficient electronic payment mechanisms and the efficiency promised by new electronic markets (both web billboard and digital financial instrument forms), we submit that the markets proposed could make small tasks more readily directable over the Internet, and could significantly enhance the efficiency of certain classes of software development.

1. Introduction

The Internet and its nodal brother, Unix, were built by a worldwide team of cooperating programmers that generally achieved task distribution by means of laissez-faire competition. In some cases, dozens of competing commercial and non-commercial teams have worked on the same end-result project and have produced similar results, with popularity and distribution of each product determining eventual success (for example, mail front-ends).

Other components, especially internetworking protocols, have been produced using the Request For Comment process, a more formal method of coordination. Both RFCs and solutions can be submitted for consideration, and solutions can be trialled and pushed freely in the market.

The competing blend of software development processes has resulted in enourmous benefit to society; clearly, the Internet is unparalleled in its achievement. In terms of the efficient allocation of programming resources to achieve such a benefit, it would appear to have advantages over models, perhaps even raising new questions in the age-old debate over capitalism versus socialism as means of production [networks].

Dramatic claims aside, the Internet would at least appear to present *prima facie* evidence of inefficiencies in the market for commercial software development. Commercial project managers will be readily aware of some of these: difficulties in specifying and communicating projects plans and tasks, quality and lead time of resources, unavailability of appropriate tools.

Notwithstanding the apparent success of the Internet, there are some obvious inefficiencies in the way that tasks are distributed across the net:

- Small tasks suffer from inordinate costs in communication, selection and control of software, so much so that activity in small tasks is uncertain.
- Routine tasks suffer from communication problems and remuneration requirements that are generally much less than the direct cost of doing the work.
- Large tasks suffer from difficulty in group management and difficulty in mobilising sufficient enthusiasm.

This paper proposes solutions built on electronic markets, with the objective of reducing inefficiencies in the initiation of tasks by users and distribution to programmers. We make three very important assumptions during the development of our proposal:

1. Efficient Internet payment systems exist.
2. Strong identity methods exist.
3. Software development, in some sense, can be the subject of an electronic market.

The first two assumptions, payment systems and identity methods, are well covered elsewhere in the financial cryptography literature, and are assumed in this paper without further comment. The third assumption is not nearly so well addressed and hence we discuss it in a following section Can Software Development be Traded on a Market?

As a new and unproven concept, there is little existing work on the direct notion of electronic markets for tasks. Known work is documented in Existing Proposals. Then, with a confidence that some software can be managed in this fashion, we propose several forms of structuring markets in Market Designs.

2. Can Software Development be Traded on a Market?

Types of Software Development

Software development is a complex task. From the point of view of the project initiator and the resources available, it can however be refined into these choices:

- DIY - Do It Yourself. E.g., Spreadsheet programming.
- In-house programmer.
- New Hire
- Contractor
- Software Company
- Off-The-Shelf

In terms of the actual code that is produced, there are many criterion that apply:

- Specification: loose versus tight, narrow versus broad.
- Quality: concept --> prototype --> demo --> alpha -> production
- Support and documentation.
- Re-usability or Single Purpose.
- Efficiency of resources: prioritised as computing versus time versus manpower.
- Solution Set: broad or narrow, easily predicted or hard to pin down.
- Budget: large or small, fixed or variable, hidden or surfaced.
- Difficulty: Routine --> Research

The Costs of Software

If we had the patience, it would be instructive to model the cost of a project as a variable depending on some function of the above. In this model, there would be a range of areas where the cost, as a dependent variable, rises at a greater rate than the source variables.

It is not, however, our goal here to dwell on this model. Rather it suffices to demonstrate one area where the market for software development produces inefficient or sub-optimal results.

To do this, we need to develop a framework of costs [Firm]. Within any task, or venture that incurs costs, we can generally divide the costs into *production costs* and *coordination costs*. Production costs, in the case of software, might be the the salaries of programmers and designers, and refer to the process of physical production once all inputs are defined.

On the other hand, coordination costs refer to those costs that do not, in themselves, produce a line of code or eliminate a bug. These include a programmer's efforts at CV writing, scanning the jobs sections, attending interviews, improving skills, and billing for results. A software producer must also find and interview programmers, negotiate contracts and pay bills. [Internal&External].

Where the Market "Fails"

Let us consider a hypothetical project manager entering the contract market to purchase 1 months-worth of labour. We will call him Dave, for consistency with later sections.

One important component of coordination costs is known as search costs - Dave must invest significant resources in procuring the human resources: definition, promulgation, initial selection, interviews, final selection, loss risk, and finally initial training costs. These costs lead to an incentive to allow each cost incurred to be written off over as long a lifetime as possible; in effect, the length of relationship should be long, allowing the once-off cost to be offset against many productive months of programming. In general the risk of failure in selection represents a limit on this incentive, and for this reason, the market for contracted programming labour settles at an optimal contract of 3 months [3months].

In practice, the search costs make contracts of one month a marginal proposition, and anything less is infeasible. This would leave Dave at the mercy of other resources above, but for a one month task, new employment is quickly ruled out, off-the-shelf software is ruled out by assumption, and software companies would tend to prefer longer relationships, larger firms, and higher rates.

In the absence of inhouse resource - which we can generally assume is limited in its breadth and depth by definition, we are left with a hole in the solution set: how to organise a task that is too short to contract, with too low a budget to out-source, and beyond the indigenous capabilities.

At this stage one could comment that the above hole is small and insignificant, but this is easily debunked:

- Within the Unix, Perl and Shell environments, there are available some hundreds or thousands of modules that are of the order of one man-month of work.

- The average module in a software project is about a man-month, varying by a factor of a three or four. That is, it is rare for a contractor to be working on the same module for less than a week or more than, say four months [1month].

- Modular programming is oriented to producing re-usable modules (here, this includes object-oriented programming). By their nature, modules tend to be smaller rather than larger, highly defined rather than broad, and communicable rather than obscure.

- We may at last be about to witness the rise of *component-ware*, fuelled by the rise of the Internet as a communications medium and Java as a programming medium. Significantly, whilst the last decade or so has seen the domination of bundling, the alternative, construction using readily available building blocks, is thought to give users of software a solution that is closer to their ideal business models. Hence, it will at least theoretically increase users' surplus (captured benefit). [Lee&Kim96].

In fact, it is easier to conclude that buyers have left the marketplace, and that therefore there is an inefficiency in the marketplace. Without stopping to rigourously prove the existance or otherwise, we rush on to examine how to address the efficiency.

Where an electronic market might help

Bakos and others [Bakos91] identify that the introduction of an electronic market system can improve buyers' welfare by:

- the availability of enhanced price information leads to dramatically reduced sellers' prices to marginal costs,
- improved availability of product information leads to a gain in allocational efficiency due to a better match between the product and the buyer's ideal.
- other supporting roles such as providing settlement systems and contractual features.

By reducing the search costs, buyers can get a better deal, as sellers can no longer rely on resistance to further searching.

The Ideal Traded Software Project

Given the mostly disjoint nature of electronic markets and software projects, we need to identify the nature of the intersection. It is:

- Small rather than large. A small project lowers the risk of failure, eases the communication task, and lowers the cash requirements, which allows a less rigourous approach to decision making.
- Highly defineable, rather than open or vague. Up-front communication of the requirement is essential as there is less scope for the tacit communication that takes place in the physical workplace.
- Assumed to be based in a particular technology, rather than left as a decision for the developer. This reduces search costs by concentrating on sellers that are presumed competent in the area.

The ideal project is a component. As a piece of reusable code, it is small, highly defined as a library interface, and is almost always constrained to a rigourous technical environment.

We have proposed that this project is in demand by buyers. It is also possible to show that suppliers exist, by examining the various newsgroups and mailgroups for technologies. It remains to examine potential structures of marketplaces, and to assess whether there is merit in these structures.

3. Existing Proposals

The proposals acknowledge as initial influences the following:

- general discussions on the cypherpunks mailgroup,
- earlier work by Eric Hughes, and
- work done already by Systemics to produce Internet trading markets.

Completion Bonds

In a talk entitled *The Universal Piracy Network*, Eric Hughes introduced the notion of a market for task completion, where completers where beyond the reach of current legal procedures such as resort to courts [Hughes]. This raised the dilemma that a contract could be accepted but not completed. Hughes suggestion was to insert an intermediary that guarunteed completion of the contract, based on the intermediary's reputation with past completions. Thus, the conflated issue of completion and trust was neatly divided into its two parts and properly placed with a party that specialised in it.

Hughes' design incorporated the use of a *completion bond* that represented an underwriting on the developer completing the project, along the line of instruments used in the Hollywood Film industry. In the event of failure to complete, funds would be returned to contributors from the underwriter.

However, it is arguable whether the notion of payback on default is universally suitable. In many cases there will be insufficient enthusiasm for restarting the market in a given task, e.g., if buys are small and widely dispersed, unlike, perhaps, the Hollywood notion where default leads to immediate need for replacement. In other cases, where integrated project work is being distributed, there would appear to be a more coherent replacement requirement to be planned ahead rather than

invented after the fact [Detail].

Java Idea Futures

A nascent market in Java tools operating at the Java Repository. This market is based on the indicated interest, rather than firm committments, along the lines of the Robin Hanson's Idea Futures [Hanson].

At this stage of development, the market is of the billboard form, where prospective buyers and sellers meet by examining a common information area.

4. Market Design

In this section, we discuss several iterated designs for markets. We start from a minimal one with basic flaws, progress through attempts to address those flaws, and conclude with a sophisticated market that addresses the predicted serious flaws, at the expense of complexity.

A Bounty Market

We propose a market that effects a one-round game with payment of bounty to the fastest deliverer. A Proposer issues a project specification, users support the project by bidding in summed contributions, and programmers compete to provide the fastest solution, thus earning the current summed contribution.

In the first phase of Task Distribution, a Proposer, Alice, [Dramatis_Personae], constructs a contract that includes a specification of a task. That task might be, for example:

```
Language:   Java - class
Product:    A minimal HTML viewer
Features:   Handles P, A HREF, B, I, H and UL.
Other:      It should be extendable.  More features to follow.
Purpose:    For Help screens within Java applications.
Quality:    Code to be tested.
Delivery:   Uploaded to the FreeJava Server.
Ownership:  Freeware
```

This is a contract that can be presented to the exchange [Exchange], for listing as a task market. On acceptance (an administration detail for the exchange), several events occur:

- The contract is made available for browsing [browse].
- The contract is distributed to mailgroups according to some classification. For example, the above might have these keywords: Java, class, HTML, browser.
- A regime for value management (cash) is chosen [Issuer].
- The exchange opens a new market in this contract for trading.

In the next phase, users who will make use of the task deliverable enter the market. They place bids for the task, providing funds that go some way towards meeting the price required by one or more programmers.

Concurrently, programmers, Bob and Carol, work to meet the specification. The first one to upload the completed code to a stated server earns the bounty.

This method achieves the advantage of employing the programmer with the *quickest* offer to provide, and is thus an efficient method of allocation. However, its use of the speed metric causes these fundamental drawbacks:

- It encourages programmer races, and thus is likely to discourage large up-front investments of time, such as on larger projects. If Bob has a run of wins, then Carol will likely be bankrupt, or at least demoralized.
- A duplication of effort occurs, with all programmers producing to the same requirement [Bryce].
- It encourages minimal results. As the specification above involves uploading only, then that is the hurdle to leap. [Padgett].

Intermediaries Can Raise Quality of Code

Without there being a mechanism to assess relative merits of delivered products, the market, in whatever form, will result in inefficient results, as, in software projects, all offers and efforts are not of equal utility.

One way to assess results is to use a qualified judge such as Victor. He will check the code and make a pronouncement as to its acceptibility, which will remove the objection of minimal results. The bounty can then be paid out to Bob, or even to Carol, if she takes advantage of any delay imposed by Victor over Bob.

A variant of this is the completion bond proposed by Eric Hughes along the lines of film production. Walter, acting as an underwriter for a completion bond, would sign a contract that specified penalties in the event of failure of Bob to meet any contract or bounty awarded to him. Bob could now expect to demand more money of the market indicating the sum effect of Walter's reputation and his own (this is more useful once the programmers' race is removed, below).

This might, for example, recompense all buyers at the face value of their contributions, allowing them to redirect the funds elsewhere. This could work well where a defined deliverable component of a project fails, thus allowing the lead manager to simply restart with another task market, etc (although does nothing to recompense for lost time or critical path chaos).

A third variant is to employ a professional manager, Dave, to manage all aspects of the tasks, especially useful where there are multiple tasks to combine into a project. Then, Dave's key decisions are similar to user decisions.

Whilst different processes, the above three structures have two similarities. Firstly, they have similar solutions in the market place, discussed further below. Secondly, they raise the following issues:

- An individual decision is required for closing of the cycle, indicating a clear inefficiency in the market: How does Victor (or perhaps Walter or Dave) make his decision? Specifically, he will have to cope with multiple interests. Will his algorithm be based on:
  - Victor's cash flow?
  - Victor's own reputation (translates to long-term cash flow)?
  - Bob's reputation?
  - Bob's cash flow?
  - Fitness of the code to meet the specification?
  - Relationship with Alice, and her demands for further features?

  Now, in a perfect market, these interests are all aligned, and thus Victor's calculations take them all well into account. However, in the less-than-ideal world of the Internet, there is no particular guarantee of this, and this decision complexity will generate a cost.

- The notion of an independant decision *shifts the burden* of the users' problems to intermediaries, which carries an increase in transaction costs and risk due to the extra participant [Burden].

- This approach is less necessary given the lowered costs of communication with the Internet, whereby each programmer has a much heightened ability to advertise their own works. That is, a programmer can post the entirety of their completed works on the web, allowing remote code checks (something that is rather difficult under today's circumstances). We could also imagine ratings agencies, databases, search engines and mailgroups that facilitate the determination of suitability.

  Notwithstanding the infinite bandwidth of Internet advertising, it does little to aid the intermediary who has trouble turning information into decisions.

- Is there a distinction between Walter as underwriter, Victor as judge and Dave as manager?

  An underwriter would typically function remotely to the actual work, using an actuarial basis of risk, that relies mostly on public information. A judge tends to be brought in at the final stages, and goes through some process of immersion in the project. A manager works on a much more complex calculation, using a lot of private information and day-to-day, on-site, input into affairs (normally, this could be considered to be an employer or lead project manager).

> Now, at what point does Walter the underwriter or Victor the judge meld into Dave the (remote) manager? On what basis will these models represent the real specialist intermediaries on the Internet, given that distance from projects is no longer applicable? This is not to dispute the need for one over the other, just to wonder to what extent the equalisation of the barrier of distance makes these roles more similar.

Notwithstanding the above uncertainties, there may be a place for the existance of Walter, Victor or Dave, where they enjoy superior information or ability. The lesson to be learnt here is not whether their existance is appropriate or not, but that a system should not be designed to exclude the existance of these forms of risk reduction, and should also explicitly allow the users to bypass these mechanisms as efficiently desirable. The ability of the intermediary to enter the market, gain his own reputation, and perform a useful risk reduction service should be guarantee of the efficiency of the market in this respect.

Fortunately, there is a relatively simple solution to this requirement of efficient intermediation:

Victor (or Walter or Dave) acts directly in the market as a seller on Bob's behalf. The intermediary places the sell order and receives the contract acceptance, and then liases with Bob to have the task completed. This also has the advantage of allowing better control of the project by the simple expedient of phased delivery of the funds [Obligations].

A One Round Contract Market

In order to address the difficulty with the programmers' race, users can make a significant concession: they can award a contract up front. By providing certainty to the coders that their efforts will be rewarded, a better long term solution will be provided, and all of the "losers" will be released for other tasks.

In order to turn the bounty market into a contract market, the following occurs: Once Alice issues a specification, Bob and Carol can now offer prices to take on the task.

Programmers need to assess the nature of the task, and ascribe likely value to it. Once comfortable with the specification, and their analysies of worth, they submit offers to the market. These offers specify the price of taking on the task [Bids&Offers]. This results in the market now having a range of programmers' offers, at differing prices.

Users, those who desire the software, are tempted to place bids for the task, providing funds that go some way towards meeting the price required by one or more programmers.

The third phase occurs when the market clears: that is, the *total* of the bids sums to provide enough funds to meet the price of the lowest offer (let's assume that Bob submitted the lowest offer). When the market clears, the following occurs:

- The programmer's offer with the lowest price is accepted. The funds from the users' bids are then attached to an offer completion, which is sent to Bob. This message represents an acceptance of contract between the Bob and the users for supply of the software.
- Bid completions are sent to the users, with an indication of the winning bid.
- All other offers are ejected from the market and returned to their submitters (for example, Carol), and the market is closed.

This achieves the allocation of funds to the programmer with the *cheapest* offer to provide, and is thus an efficient method of allocation, but by using a different metric of efficiency to the bounty market. The contract market suffers from this fundamental drawback:

The solution presented by Bob may not be as good as the solution that might have been produced by another programmer. In the worst case, Bob may not complete his part of the deal by failing to meet the specification, or even reneging completely.

Differentiation Can Raise Quality of Code

Without there being a mechanism to judge relative merits at a level beyond price or time, both the contract and bounty markets will result in inefficient results, unless we can guarantee that all offers are of equal utility, according to their chosen metric. It is examination of the metric to which we now turn, in order to improve the efficiency of the market.

Which is better, speed or price? Whether *time to market* or *price-performance* is your preference is an individual question; to

one man’s meat, another’s poison.

By introducing meat and poison into the market, we challenge an assumption: that the offers are fungible in some sense. In software, there is little chance of fungibility being a common occurance. Indeed, marketing theory would have it so, as the poorer strategy is generally to compete on price (only one can win), and the better is to differentiate your offer into new territory.

So we need a market where the offerings are differentiated, but related around a core commonality of Alice’s original specification (see Appendix A for a discussion of market definitions). In order to be readily assessable by users, it should probably include some standard features, and room for extension. For example, the first standard feature would be price (already present in the above), and the second would be delivery time.

The third standard feature is the identity of the programmer. If Bob has submitted the cheapest and fastest bid, then his offer would dominate that of Carol in either of the two markets described earlier. However, if Bob has only 1 month of experience in Java, whilst Carol was part of the original language design team, there are bound to be some people unhappy with the way that Bob’s order always wins the bounty or contract.

Now, in general, it is hard for systems to measure intangibles, but plausible for users to do so. Or, at least, users do not hold back in attempting to measure these things. So in order to present users with a choice based on intangibles, we would need to do several things, as driven by this logic:

1. a major determinate of choice is intangible factors, and
2. a major intangible factor is the reputation of the programmer, and
3. the users must be able to identify the programmer behind the offer, and
4. they must be able to relate the identity to a body of information that defines reputation, and
5. notwithstanding the use of market clearing, the users must be able to prefer one offer over another.

Users, if presented with this arrangement, would then rationally choose to bid for only those programmers that they feel are likely to deliver, where *deliver* implies conforming to each user’s particular metric. And, with markets being markets, any breaches under this metric would cause users to resort primarily to *caveat emptor*, albeit that variant that is popular on the Internet [Caveat Emptor]. This approach then addresses the weakness of the contract market, above, by using reputation to apply a cost that is incurred by the programmer for supplying poor quality software.

A Differentiated Multiple-Round Market

In order to add differentiation into the market, we propose a market that allows programmers to present differing offers on the same task. The three differentiation paramaters of price, delivery time (after clearance), and identity will be included. To allow for flexibility and innovation, we further add a completely open text field.

Users can see the differing characteristics and can also determine who is making the offer [Identity]. Their bids will specify a list of offers that are applicable. They rationally apply a discount factor to every less than perfectly reputable seller, and they favour offers that best match their own requirements.

If we imagine Bob, Carol and Dave all active in the market, the following might represent a users buy screen:

	Price	Delivery date	Cash bids min - max	Free-form textual additional clauses
Bob	80	60	10 - 70	fast and furious
Carol	120	90	30 - 90	Tested to Destruction
Dave	150	85	50 - 110	Includes Documentation!

Each offer has against it funds that represent summed bids that are committed to just that single offer, and total bids if all switchable funds were put against it.

The Clearance process now has to check each sell to see if the sum total of funds applicable to just that sell will result in clearance. If so, it clears that sell order and advises all parties of a trade. Parties can then either cancel their orders or redirect funds. As there will be many times where it is the user’s interests to finance multiple versions, the market remains open with bidding against the remaining offers.

As each task is offered by a given name [Nyms], there is an ability to apply reputation by relating that tag to past efforts (mailgroups, Web advertising, etc). The problem of dissemination of reputational information is best handled outside of the strict trading regime of the electronic marketplace, due to the intangible, variant nature of the information.

It is still possible to assist in the collection of reputational information, by, for example, providing a home page in the PGP id field, and having the user's task market browser access this page (so as to check out written code).

Thus the strategy for programmers becomes one of competing on the differentiated factors. Reputation is something that can only be properly built up over repeated rounds of the market process, supplemented with appropriate web-based advertising. Note that such a multiple-round game presents interesting alternative to the use of an intermediary in order to improve efficiency of the market.

Market Feedback with Voting or Shares

The markets so far proposed do not do much more than distribute the tasks efficiently. Specifically, they leave open questions of ownership, partly in line with our notions of market forces determining the best way to finance, develop and exploit software, and also our preferences for free software in its myriads of forms.

However, markets are normally related strongly to ownership, in that they assume the concept of private property rights, and this section examines extensions necessary to support development, where the deliverable is valuable, in a revenue stream sense.

As in conventional markets, the bids and offers should be orders to trade cash against financial instruments. Then, in order to offer to supply, a programmer must offer a package of instruments that capture the ownership of the project. These instruments will be initially worthless before clearance, then becoming promises to share in whatever of value is to be delivered. On delivery of the project, they become simple shares, possibly earning royalties on sales. This complex pattern of value makes these instruments derivatives rather than simple shares, although the distinction is not important for our purposes [Derivatives].

When the market clears, on Clearance time C, the instruments are delivered to the bidders in the market completion. Users holding these instruments hold something of value, being a share in the future profits of the project, and thus it makes sense that they be tradable. For this purpose, the exchange can maintain an open market in these instruments for buying and selling (although, it is a different form of market to the task market, being more conventional).

Then, on project completion (C+t), the work is uploaded, at which point the market could initiate an advice to the owners of the shares. The futures convert themselves to shares in the deliverable, which as a valuable item will now deliver a revenue stream via, for example, a software shop on the Internet.

This approach has some interesting ramifications:

- Imagine Victor as an owner of shares. He can now download the software (perhaps for free as an additional right of ownership) and test it out. His viewpoint on the software, according to his own analysis of value can then be reflected in his decision to buy, hold or sell his shares. This has an effect on the price of the shares, still traded in an open market, which provides another useful input into the reputation of the programmer: Bob's success as a producer of quality product can be checked against the price of shares in his previous projects.
- Financial instruments could also be used for managing the free software results. In this case, on time C + t, the shares could be spent at a shop that provides Victor a chance to vote on the product. As the voting will reflect the amount of shares held, and thus how much he desired the software in the first place, this provides an interesting feedback loop for Bob.
- Walter, the underwriter, could now step into the market. In the event of default, repayment of funds could be effected by entering the market to buy out the owners' interests.
- As the ownership of the project is now a matter for the market place, Bob can enter the market to manipulate the price and attempt to improve his reputation. Rational users would of course realise this possibility, and therefore, discount the price to that extent.
- Bob can also enter the market to buy back all of the instruments at any time. If he achieves a takeover, which might be based on conditions in the original contract, then he would possibly find himself at liberty to adjust the terms of the contract. He could, in this way, buy himself out of the contract [Takeovers].

To add more flexibility, payments can be sent automatically by the market according to a formula in the sell offer of the programmer. This could allow for 3 phased payments: Clearance (C), Delivery (C+t), and post-support (e.g., C+t+s).

5. Concluding Remarks

Summary

By examining the cost structure of software development, we have identified where there are allocational inefficiencies. That is, for the short, well-defined projects known as modules or components, we can conclude that the existing market place has search costs that preclude buyers from meeting sellers efficiently.

In order to address this, we look at the ability of electronic market places to dramatically reduce the search costs. We propose several market microstructures that achieve a range of complexities and efficiencies.

Given the low transactional costs afforded by the Internet, and newly available trading technologies, it is possible to build a market-based solution that could significantly contribute to efficient development of Internet software.

Notwithstanding the brave predictions, it is easy to criticise the approach. As any programmer will affirm, software has a high proportion of tacit information and coordination requirements in it, challenging such assumptions as our highly defineable components [GHM]. This leads to a preference for partnership relationships rather than contractual relationships, something that the task market seeks to reduce [Partners].

Status and Some Findings

Systemics has a working internal market. It performs to all working specifications, but access is not user-friendly (Unix command line rather than web pages). The next phase of the project is to add end-user accessibility, there is an experimental contribution HTML method.

Beyond the obvious - that this project is only small on paper - the following are worthy of note:

Dynamic Instruments. The markets built by Systemics were designed on the assumption of each new tradeable item being carefully designed, vetted and mounted as a traded instrument. This followed the real-world markets where initial public offerings (IPOs) are events of at least three months in the making.

With the task market, this assumption is shattered. On clearance, an item of definate value is created, and this must all happen in real time. This heavily strained the design of Systemics' markets, and broke the original rational for us to enter into the project: that it was relatively efficient to add this feature in.

However, where economic reason falters, programmers carry on. The market is now redesigned and reworked to permit dynamic financial instruments. It remains to consider whether there are alternative uses for short term or cheap traded instruments.

Logistics *versus* Strategy. Sometime during the Gulf War of 1991, General Schwarzkoph said "Journalists talk about strategy. Generals talk about logistics."

The hard part of coding trading markets is not the algorithms to make the clearance decisions but the bulk cash management code that holds and disburses value. As a system, the applications are dominated by the error case, and especially, the need to maintain value in a reliable, consistent state regardless of circumstances.

Free Riders. There is a plausible inefficiency in the existance of *free riders*, however, we prefer to wait and see whether this represents an issue, on the basis that the Internet is already built on this premise.

Corporate Secrets. Some managers looking for a universal method have been surprised that in order to obtain task completion they have been required to specify the contents of their work plans. The loss of secrecy in activity would appear to be, at this stage, a cost of entering the market.

On a related front, a plausible attack has been suggested: that of using the market as a cover for industrial espionage, e.g., by listing a task that is most efficiently met by stealing and publishing a competitor's trade secrets. This is clearly possible both within this proposal and without, and would appear to reflect the efficiency of this method, rather than present a motive to adjust or defer operation of the market.

Appendix A: A Short Note on the Economics of Markets

Based on the framework described by Yannis Bakos in his 1991 paper on electronic marketplaces, "A Strategic Analysis of Electronic Marketplaces", we summarise some key elements that help our description of the markets developed in the body [Bakos91],

Definition

Bakos defines an electronic marketplace as:

> an interorganisational information system (IOS) that allows the participating buyers and sellers to exchange information about prices and product offerings.

Markets can fundamentally be divided into two forms: *commodity* markets and *differentiated* markets. This division occurs on nature of the product across sellers, with commodity markets characterised by products that are essentially identical, or *fungible*, regardless of the seller.

User Costs

The costs that a buyer in the market must cover include:

Search Costs. Where buyers have imperfect information on entering the marketplace, they must expend time and effort in improving their information to the point of decision. These costs, known as *search costs*, are key to the sustainability of economic rents, or higher-than-normal profits.

Transportation Costs. In the event of buying a non-ideal product in a differentiated market, the buyer faces a cost in *transporting* the product from its supplied state to its ideal state of usage.

Transaction Costs. In each transaction, there will be a fixed component involving the settlement of exchange of goods for currency. These can be formalised, for example, as the margin placed in credit card purchases, or they can be hidden, such as the cost of invoicing through the accounts department.

Switching Costs. There are fixed costs that an individual incurs in entering into any marketplace for the first time: learning how trades occur, where to search efficiently for information, etc. Once incurred, these become sunk costs, and the decision to change from one marketplace will incur these costs again, making switching a relatively expensive exercise in comparison to not switching.

Influences

Strategic considerations of electronic marketplaces include these elements:

- high fixed cost, including high capital investment in systems and the large sunk costs of development.
- large economies of scale, and some of scope.
- network externalities.

These costs determine the structures possible and the likely potential for economic rents.

Infrastructure

Electronic markets demand several elements:

- Communications: methods to permit buyers to search.
- Comparisons: standards for distribution of information that permit lesser or greater degrees of comparison.
- payment settlement mechanisms.
- Auditing, traceability, and other reliability tools.

Appendix B. Requirements of a Task Market

The cooperating parties have different objectives in participating in markets for task distribution. This section lists the parties and their objectives, to the

extent that we understand them. The chosen parties are not especially standardised, but proved to be useful based on experience with the design phases.

Most participants are interested in these requirements:

- Efficiency in time spent on conforming to the market architecture.
- Breadth and Depth of coverage - lots of different tasks and a strong competitive market in these areas.

Programmer

In participating in the market for tasks, the programmer is mostly interested in:

- Clear specifications.
- Clear completion milestones.
- A feedback mechanism of some form. this might take the form of voting on product, comment by peers or success in the marketplace.
- A structure of tasks that improves knowledge and/or the opportunity for "good" work in the future.
- Reward of some form. Whilst the most obvious form is money, all things are possible on the Internet [Maslow] Many programmers perform tasks for wider considerations, such as challenge, self respect, credibility, a sense of belonging and/or the gaining of experience [Freeware]
- (Excess) demand for interesting task completion.

User

Users have much simpler needs that can be expressed as solutions, solutions, and more solutions. There are some more specific interests:

- Security of knowing that a solution is on the way.
- Cost and Time limits.
- Proposal development. The User needs to push the model through a sanity checking phase.
- Implicational analysis. What does it all mean?
- (Excess) supply of professional task completers.

Proposer

The task proposer is harder to tie down in terms of their needs, as any of the above could propose a task (and indeed, it is hard to imagine any other party not listed here as proposing a task).

A programmer might trawl for new projects, a user could be planning a new strategy, or a project manager might have other agenda issues such as needing components. Which ever is the case, the needs of the proposer tend to be those of their fundamental activity, simply coupled with the ability to make a proposal.

Market

The market needs good operating characteristics that send its variable costs close to nothing. This means that all phases have to be automatic, and that support problems are not generated.

References

Version 0.6. Presented at Financial Cryptography 1997. To be jointly published in JIBC and Springer-Verlag. Original source is at http://www.systemics.com/docs/papers/task_market.html

Ian Grigg can be reached at iang@systemics.com . He is a founder of Systemics, Ltd, a developer of Internet Financial Systems software.

Christopher C. Petro can be reached at petro@suba.com .

Networks. Specifically, the Internet has outstripped efforts by major companies and governments in networking ventures. For example, the combined competitive efforts of AOL, MSN, Compuserve, and Prodigy have been subsumed, and Minitel in France is being overtaken.

The Firm. The nature of costs used here derives from *The Theory of the Firm*, a branch of microeconomics. Specific framework was extracted almost verbotim from Erik Brynjolfsson, Thomas W. Malone, Vijay Gurbaxani, Ajit Kambil, An Empirical Analysis of the Relationship Between Information Technology and Firm Size MIT Center for Coordination Science Working Paper 123

For reference, see especially Oliver Williamson, *Markets and Hierarchy: Analysis and Antitrust Implications*, Free Press, New York, 1975; and also Oliver Williamson *The Economic Institutions of Capitalism*, Free Press, New York, 1985. The Theory of the Firm was sparked by Coase, R. H., "The Nature of the

Firm", *Economica, New Series* (1937), Vol. IV, pp.386-405; reprinted in: Stigler, G. J., and Boulding, K. E., (eds.), *Readings in Price Theory* published by Richard D. Irwin, Inc., Chicago, 1952.

Internal&External. The theory goes on to divide these costs into *internal* and *external* coordination costs, although that distinction is not used here.

3 Months. It is often stated as *3 months* + where the plus signifies that the employer has an option for further work, but the contractor does not. Based on the author's experiences, try searching today's contractor positions at JobServe.

1 month. This is a personal observation based on the author's time as contractor. YMMV.

Lee&Kim 96. "A Game-Theoretic Analysis of the IT Paradigm Shift to Network Computing with Component-Ware", presented at *Workshop on Information Systems and Economics* , Bryngtae Lee (University of Arizona) and Beomsoo Kim (UT Austin).

Bakos 91. "A Strategic Analysis of Electronic Marketplaces," *MIS Quarterly*, Volume 15, No. 3, September 1991, pp. 295-310.

Hughes. Eric Hughes, *"The Universal Piracy Network,"* unpublished paper presented at DEFCON IV.

Detail. I am indebted to the notes of David Molnay and cypherpunk posts for what information I have on this proposal. The detail listed here could be wildly inaccurate.

Hereafter are some random queries: Is the completion bond on the component or the project? E.g., in the Hollywood film production cycle, is it on the actor who spits the dummy, or the film production that gets bucketed? If the bond is on the former, cost is likely to be difficult to manage due to intricacies (need to re-engage other actors, etc, for re-shoot, or lost time to market). If on the entire production, it is much more manageable in terms of cost, as it becomes built into the dicount factor. However, this implies that the investor can do it themselves (by diversification).

Hanson. Idea Futures is a market for betting on future events. It uses virtual money to generate odds which reflect the prediction of those who betted.

Maslow. The best source of inspiration here is still Maslow's Pyramid of Needs, IMHO.

Freeware. For example, Systemics publishes the Cryptix library. This library is freeware: essentially free for all-comers. In return for this open benefit, the authors are rewarded by bug reports and reputation, Systemics gains credibility, and all gain sense of belonging to the Internet community.

Dramatis Personae. We will use the cryptographers' friends to play our parts. Specifically, Alice, is a proposer of tasks, and Bob and Carol can be programmers. Dave will be a manager (responsible for the production of others). Victor is a verifier of other's claims. Walter is an intermediary who watches (and protects) the interests of others. Bruce Schneier, *Applied Cryptography*, 2nd edition. Users are, as always, users, and represent the buying mass of the market.

Exchange. Normally we will talk about a market, the markets and the market without distinction. Where a distinction is necessary, *a market* will trade one item, and an *exchange* will consist of many such markets. *The market*, in the global sense, refers to the sum total of all trading opportunities.

Browse. It is an architectural principle of the Systemics markets, at least, that the traders must have access to the exact contract that is traded. Hence, all contracts are made available for download.

Issuer. An Issuer is a manager of digital value. In this case, it is assumed that the Market will trade cash for contract items (although for some variants this might be overly complex). Regardless, the presence or otherwise of the Issuer does not effect the architecture described.

Bryce. Looking at the Debian group , a loosely organized band of Linux hackers, Bryce suggests that if programmers were to coordinate in off-market channels they could avoid losses such as duplication. There are two counter-arguments to this: firstly, that if programmers can coordinate in this fashion, then they might not benefit from a structured market in the first place, and secondly, that the coordination of resources by non-market means is not as scaleable as that achieved by markets. Post to e$@thumper.vmeng.com on Wed, 19 Mar 1996.

Padgett. Or, as Padgett warmly puts it, "good software takes time to write. Schlock does not." Post to dcsb@ai.mit.edu on Tue, 26 Nov 1996.

Burden. For an explanation of the *shifting the burden* see Peter Senge, *The Fifth Discipline*, Currency Doubleday, 1990.

Obligations. In some cases the intermediary could step in at a particular moment, rather than being a central part of the trade. For example, Walter could step into the market on default by Bob to meet their mutually promised obligations. This requires, however, more complexity in the way of out-of-band protocols such as email, or intervention by the Issuer, or open market operations, described later.

Bids&Offers. In markets, we talk about *bids to buy* and opposing *offers to sell*. In this type of market, we are really talking about *bids to enthuse* and *offers to hack*, but the flow is the same.

Caveat Emptor. The Internet uses *caveat emptor* as its natural law approach to everything. On top of that is imposed a *laissez-faire* approach to revenge that tends to be self-regulating: Frequently, reputations have been ruined by mistakes made, and perpetrators of crimes have been hounded by the mailgroups' kangaroo courts. Reputation is a powerful tool in the small and dynamic communities on the net.

Identity. Identity can be established using public key cryptography tools such as PGP. These techniques allow the secure building up of reputations over time, as is often done in newsgroups.

Nyms. Little we have discussed precludes or insists on the presence of psuedonyms (nyms), and we believe the decision to use them or not is orthogonal to the design of the market. One exception is the notion of code checks, although Lucky Green, in a post to e$ (03 March 97) suggests that there are certificate blinding protocols to preserve the essential anonymity.

Derivatives. For traders, these instruments are options on futures on shares. Working backwards, the shares are portions of ownership of the project assets, just like any project such as a company. Then, before delivery but after clearance, the items act as futures on the shares, as the programmer has contracted to deliver, with a time and place. Then, before clearance, the instruments are options on the future, with the option being "in the money" automatically in the event of clearance.

At least, that's what I think they are. YMMV. The problem with this view is that as options, they are valuable, and thus they should be traded. However, to simplify, we assume that they are valueless until clearance; this is clearly challengeable.

Takeovers. Indeed, it should be possible to write a contract that provides software wholey and solely under the control of the owners, thus presenting the possibility of innovative financing techniques to promote attacks on insecure software, a popular task on the net. When a toolkit is complete that enables users to attack badly protected communications, such as satellite transmissions, would the satellite provider feel compelled to buy out the owners? In an anonymous market this might be possible, although Internet programmers have generally been far too responsible to participate in damaging or immoral attacks (although all bets are off when they view the opponents as damaging or immoral).

GHM. A more formal model is the incomplete contracting framework of Grossman, Hart and Moore.

Grossman, S. and O. Hart, "The Costs and Benefits of Ownership: A Theory of Vertical and Lateral Integration." *Journal of Political Economy*, 24, 4, (1986).
Hart, O. and J. Moore, "Property Rights and the Nature of the Firm," *Journal of Political Economy*, 98, 4 (1990), 1119-1158.
Erik Brynjolfsson , " An Incomplete Contracts Theory of Information, Technology and Organization ," *Management Science*, revised June 1993.

Partners. Bakos and Brynjolfsson suggest that as transactions costs go down, encouraging markets, competition shifts to intangible quality aspects. This leads to more partnerships built on sharing the costs and benefits of investment in improvements in quality, which are otherwise difficult to tie down in a contractual arrangement. Their model suggests a natural counterbalance to the shift towards markets.

J. Yannis Bakos , Erik Brynjolfsson , " An Incomplete Contracts Theory of Information, Technology and Organization ," *Journal of Organizational Computing*, revised June 1993.