Alone Together: Patterns of Collaboration in Free and Open Source Software Projects

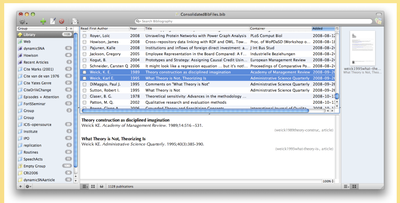
Ph.D. iSchool at Syracuse University **James Howison** Post-doc CMU Computer Science
http://james.howison.name

How do community-based FLOSS projects organize their work?

Participant Observation

BibDesk: A community-based reference manager for BibTeX, running on OS X.

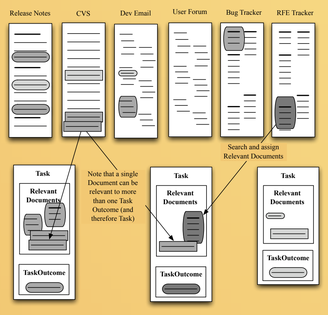
Four years participation: Experience suggests individual, layered work and deferral



Archival Study

Fire and Gaim: both community-based instant messaging clients, relatively successful in period studied

Organized all archives into Actions undertaken as part of Tasks (changes to shared outputs)

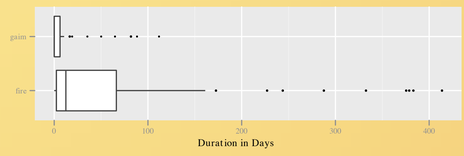


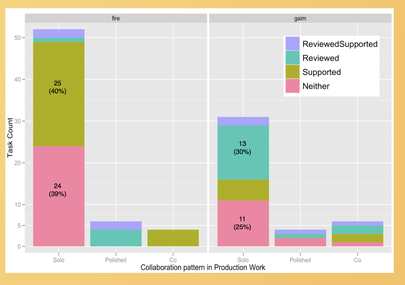
Coded Actions according to the type of contribution they make to the Task.

Classified the Tasks according to the number of participants undertaking different types of Actions

Code	Explanation and Example
Management Codes	
Management Work	Work done to organize other work. This includes planning, setting deadlines or announcing 'phases' like code/string freezes, assigning or rejecting tasks. This includes restructuring the infrastructure and accounting for work/credit. It also includes declaring bugs fixed, or Patches applied (closing Trackers) *A Call for Release and Creating a branch in SVN (eg gtk1-stable)*
Assigning Credit	Thanking people, *adjusting the Credits file etc.*
Review Codes	
Validation Work	Validating a coding technique, fix or approach (before or while it is being done)
Review Work	Work done to review other work, including checking in code written by others. This includes work that rejects patches etc.
Production Codes	
Core Production Work	Work that directly contributes to the project's outcomes; either through working application code, or through production of user interface elements (logos etc). *Implementing a feature (not successfully check in, since could be checked in on behalf of someone else)*
Polishing Production Work	Smaller changes that improve Core Production contributions (typos, integrations etc
Documentation Codes	
Documentation Work	Work that documents the code, application or activities. Includes pointers across Venues (eg in a Bug Tracker saying that a Patch has been submitted) *A ChangeLog/README only SVN commit.*
Planning Work	Work that documents one's own future activities (others is more likely to be management work)
Supporting Codes	Work that supports the work of others
Use Information Provision	providing information about the code base. Or use cases & RFEs and bug reports. These are often about conveying a private situation.
Code Information Provision	Includes providing suggestions about how to complete work (code examples or pseudo-code, if it compiles or is a patch against SVN then code Production Work). This includes a dev seeking more information from a peripheral member.
Testing Work	Testing application functionality. This includes requesting more information from users in bug reports.

Individual and Short Tasks, layered on each other, dominate





Classification of Tasks by number of unique participants doing production. Dominance of Solo Work is strongly evident.

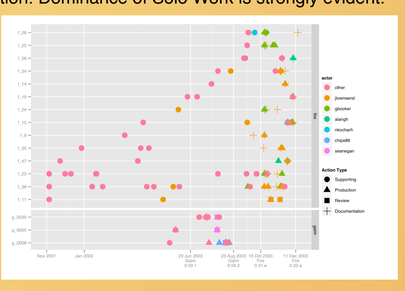
Complex work is often deferred until other tasks change the codebase to make them

June 2003 (Email)

I really want to use this, but the conditions have never quite been right - either I was waiting for … RSS+RDF (now looks like it'll never happen) or … an XML bibliographic file format … (could happen now, but I ran out of free time).

Jan 2007 (Email with patch):

It was much easier than I expected it to be because the existing groups code (and search groups code) was very easy to extend. Kudos - I wouldn't have tried it if so much hadn't already been solved well.
Thanks!



Tasks longer than 100 days show early supporting work, often requests or 'votes'. Production work is clustered; it begins when other work has made the task easier.

Why do they work this way?

A rational choice model of participant decisions

Background

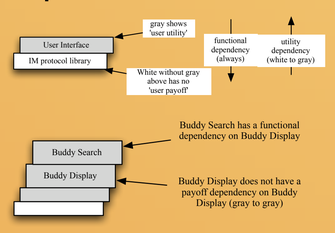
Research on motivations in FLOSS highlights the role of individual motivations, both extrinsic (e.g. working for useful software, scratching an itch) and intrinsic (e.g. learning or fun). The FLOSS environment has little, if any, claim on the time of participants; participants are volunteers (even if they are working for a firm, the project itself has little coercive ability). The model presented here is a simplification to explain the dominance of individual work and clarify the function of deferral. It is based on a rational choice model grounded in the expectancy-valance model of motivation. Only the simplest model, which makes assumptions that make the task hardest for the project is presented here. Many of these assumptions are eventually relaxed, allowing, for example, learning motivations to drive production without immediate payoff.

Assumptions

Table 7.1: Model Assumptions (✓shows which are later relaxed)

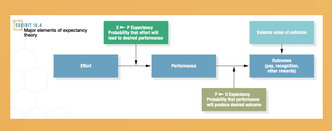
#		Assumption (*Justification*)
		Bounded Rationality
1	✓	Participants will only work for a utility payoff (*Parsimony*)
2	✓	Participants are motivated only by their own use of the software (*Parsi., Lit. and Exp.*)
3		Participants are good, but not perfect, judges of task complexity (*Parsimony and Exp.*)
4		Participants know the limitations of their judgement (expectations approach reality) (*Parsimony and Experience*)
		Other Assumptions
5	✓	All participants have the same set of skills and availability (*Parsimony*)
6	✓	Participants only know their free time for the next turn (*Parsimony and Experience*)
7	✓	There are no exogenous sources of code or solutions (*Parsimony*)
8	✓	Participants can build on existing layers without assistance from authors (*Experience*)
9		Contributions are always shared under an open source license (non-revokable, no royalties, allows derivative works) (*Parsimony, Lit. and Exp.*)

Graphical Notation



Decision Model

Expectancy-Valance model of motivation



Participants choose between working to realize utility from the application or spending their free time elsewhere.

The expected payoff of working depends on the expected utility of the outcome, conditioned by the expectation of successful completion of work.

$$E(B_{choice}) = E(U_{outcome}) \times E(P(e \rightarrow p)) \times E(P(p \rightarrow o))$$

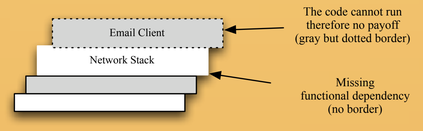
- $E(B_{choice})$: Expected payoff of working
- $E(U_{outcome})$: Expected utility of improving application
- $E(P(e \rightarrow p))$: Expected probability that effort will lead to performance (i.e. chance of writing code that works)
- $E(P(p \rightarrow o))$: Expected probability that performance will lead to payoff (i.e. chance that your code improves the application sufficiently to get payoff)

Solo Work:

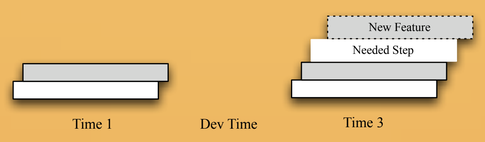
Individual, Layered, short and small work is possible

... but complex work is restricted

(Participants can only build a single block)



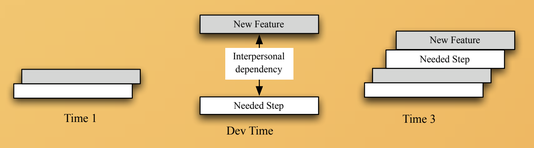
This re-states the problem of Collective Action



Complex problems like this are a common reason for working together; but what are the motivational impacts of co-work in the FLOSS context?

Two quite different solutions were observed in the empirical work above.

Solution 1: Co Work

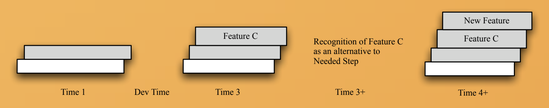


- Reliance on others to achieve a payoff makes your link from performance to outcome dependent on their link from effort to
- performance.
- In algebraic terms their e⟶p becomes your p⟶o
  Concurrent co-work also increases coordination costs, which can be
- modeled as an additional risk of failure at e⟶p
  Sequential co-work would avoid coordination costs, but introduces a payoff delay discount

Interdependent Collaboration is risky

Solution 2: Deferral as novel solution

- Complex work is deferred (accepted as desirable but not attempted.
- Other, less complex work is undertaken as individual work
- Participants eventually realize that the less complex work has made the complex work easier, and it can now be undertaken as individual work

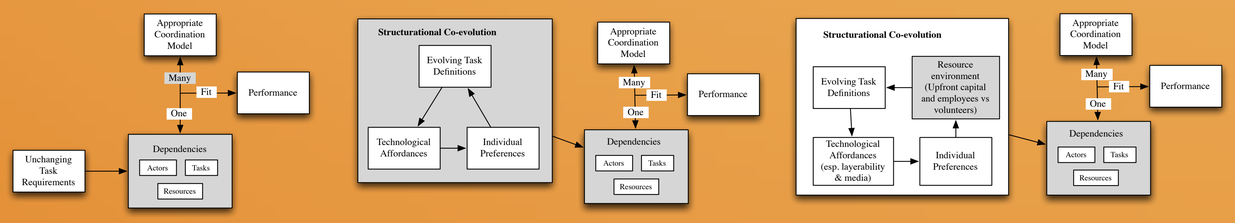


Deferral and a changing codebase, can make complex work easier

So What?

Theoretical Implications

There is a debate in the Management literature regarding the determinants of interdependency; does it flow from task requirements, or is it an emergent property? This work adds a focus on the resource environment, arguing that volunteer work without upfront capital, together with a layerable task undertaken through lean media articulates well with low interdependency work.



e.g. March and Simon (1958), Mintzberg (1979), Thompson (1967), Van de Ven et al (1976), Malone and Crowston (1994)

e.g. Shea and Guzzo (1989), Wageman (1995), Wageman and Gordon (2005), Rico and Cohen (2006)

Contribution of this dissertation

Patterns reflect an emergent alignment of motivations and resource context, technologies of production and communication.

Conditions for FLOSS collaboration

	Input
1	Ultra-low upfront investment
2	Individually motivating work
3	Past work is available, non-revokable and non-exhaustable
	Output
5	Instantiation is ultra-low cost and near-instant
6	Distribution is ultra-low cost and near-instant
	Process
7	Task can be approached in layers
8	Task is rewindable
9	Work and communications are observable
10	Communications support temporal mode switching

Attempted Adaptations

Wikipedia
Highly similar: layerable, rewindable work, low-upfront investment, past work is non-revokable and non-exhaustible, visible work and communication. Instantiation and Distribution are near-instant, but bandwidth costs can be high. Work appears largely individual.

Open Hardware
Effort to apply FLOSS principles to hardware. Hampered by slow instantiation and distribution costs. Work has focused on informational representations (e.g. blueprints) and informationalization (e.g. FPGAs).

Policy Advocacy
Some urge a FLOSS approach to democratic input, such as calls for comment on legislation. Hampered by low layerability and in-direct, delayed payoffs (payoff is in impact on process, not immediate work).

Commercial Software Development
Efforts to adapt FLOSS to internal, commercial environments (e.g. Inner Source) face issues with up-front investment and deadlines, undermining immediate payoff motivations and usefulness of deferral. Hybridization undermines some of the factors that make FLOSS work.

Adaptation is much more difficult than commonly acknowledged