Manifesto, management and money

The future of collaborative software development.

Peter Fox[1], January 2007

Updated Nov '08

Summary

This paper starts from the premise that the software project paradigm composed of a team of volunteers, as for example used by many open-source developments is basically A Good Thing, but (a) in the main is not being done very well (b) lacks a sound reward structure and (c) lacks an infrastructure to facilitate the cross-fertilization of ideas, competitive approaches and fluid application of skills.

So what can be done to channel bright ideas, enthusiasm and technical prowess into top quality, top value products?

1 Separation of the 'creation of bright ideas' from the 'production of quality software'.
2 Improving the standard of collaborative teams.

The first is tackled by 'Manifestos' (equivalent perhaps to a script for a play or film) that set out what the idea is and why it is worth working on and some details about the style of production. These are then published for criticism and adoption by the developer community. The second is dealt with by simple analysis of how we expect a collaborative team to be structured and perform. The term 'Volunteer Company' (where volunteer doesn't necessarily mean unpaid) is used to describe a group which probably has an existence separate from a specific project but may specialise in some way. The key idea is that the 'expensive' process of becoming an effective team doesn't get repeated for each project and the emphasis changes subtly from features of the product to competence and application of team members.

The other matter, that of getting paid, earning money, developing a career and respect among one's peers often gets fudged by filing under 'too difficult and too boring'. It turns out that we already have a thriving, 500 year-old, working model to deal with this - the theatre. The analogies between the performing arts and software development are examined to show how Volunteer Companies fall neatly into place as a very important and socially valuable enterprise, worth supporting financially.

When these are used together, today's 'buskers' and 'ants and twig' methods can be converted into tomorrow's respectable, organised, properly rewarded and socially desirable activity.

There are more details of how to construct manifestos and volunteer companies in the appendices.

1 Introduction

The urge to program

I am not getting paid to write this. I'm doing it because it is (a) more interesting that 'real work' and (b) I believe it is a valuable contribution to the Lives Of Men; that is something Worth Doing.[2] Some people start blogging for campaigning reasons or boredom or a desire to experiment with the technology of expressing ideas. Programmers are a similar but slightly different breed: They get a buzz from making useful things and fixing 'broken' things. For example I wanted a calendar that would show the phase and visibility of the moon.[3] Since I wanted a diary anyway I wrote my own to suit my own tastes. Now where did I get the necessary astronomical code from? Somebody who had generously made their code available on the Internet. Where did they get their astronomical calculations on which to base their code from? From books which are the concrete expression of the scientific knowledge base, which in turn are layered upon previous knowledge many times over.

From this we can see that making knowledge and programs available is a virtuous circle: Now any astronomer will be able to use my program to make a date for moonwatching.... ...If I make it available.

The 'use, enjoy, benefit' cycle.

If someone steals your property we call that theft and look for a legal remedy, preferably imposed by a society which considers this to be abhorrent as a matter of public policy. But what if you give something away to somebody who sells it? That can leave a very nasty taste in the mouth. *But nobody forced you to give it away*. Well actually this isn't quite true there are three situations where you are forced to give it away:

1 If you patent an idea : In return for official recognition of ownership and rights attached to that ownership you must make the complete method public.[4]
2 If in order to live you have to disclose all the details. I could go to your poetry reading with a tape recorder and print your words in my anthology.
3 If you are working in an environment where disclosure, peer review and free exchange of ideas is accepted as the norm.

As we saw above, there are great benefits in sharing ideas, formulas and code, so even though there is no *forcing* it is highly *desirable* to keep enriching the knowledge-base and code-base. The more there is the greater the benefits that should result : All for the Good Of Mankind. This method has been fundamental to the rapid evolution of the Internet's protocols and implementations. Various licensing schemes such as GPL and Creative Commons have evolved to encourage contributors to keep enlarging the knowledge and method universe. There are two principle issues which are not always compatible:

- Once freely available-always freely available, and
- Restrictions on exploitation

The principle has been established that it is possible to contribute without being 'ripped-off' but some practical details remain unsatisfactory. For most people, 'reward' remains the warm glow of knowing that somebody, somewhere, sometime might benefit from their work.

We have gone on this diversion in order to set the scene in which some of the most creative and talented people on the planet

---

1 Peter Fox is a freelance business consultant living and working in Essex, England

2 As I write this a neighbour is coaching the lads football team and another lady is cooking charity breakfasts.

3 Nice to know if you're cycling through the countryside at night.

4 In the UK Software and mathematical formulas are not patentable - (Let's hope it stays that way).

contribute for a very uncertain reward. This issue will be dealt with in section four. However, and this is important, FOSS is not the only model for collaborative development. If the goals of building better quality software are met then there should be plenty of scope for making a reasonable charge for something that does what it says, and is reliable and supported. Or to put it another way: Someone itching to program stands a better chance of getting some reward if they are part of a professional team who value their reputation than 'going alone' and simply not having the resources to cover all the bases.[5] There is no fundamental reason why a commercial operation looking for particular talent shouldn't advertise providing of course that the terms of participation are made clear.

Underachievement

If you skip along to Sourceforge[6] you'll see hundreds of projects that started with one developer, ended with one developer and managed to release something that might charitably be called 'proof of concept' then died. Let's applaud the naïve enthusiast for 'having a go' - Already they have achieved more than the 'why bother' masses.

Some projects get further, typically having been matured before being released, or having limited objectives - There can be lots of useful stuff here, but now, without a life-force, these projects have stagnated and perhaps become broken with the passage of time.[7] [8] So the project activity has ceased although the results may be useable if not well supported.

A few projects start with or acquire a nucleus of developers who continuously improve, repair and enrich their baby. With time the focus of the project may change, factions might break-away to concentrate on matters of private interest and so on. These are living entities fed by dedicated slog during the midnight geek-hours. (This is not to suggest that Use-Enjoy-Benefit projects are particularly susceptible to organisational disruption, after all traditional commercial outfits can cease trading or get taken over and drowned - and when that happens the consequences are often terminal which isn't the case with OSS.)

The adage 'you get what you pay for' is not true as far as this field of endeavour is concerned. Unuseable, badly supported, limited life-span, corrupting and simply non-functional software can be acquired for free or commercially. Of course, like any other product there is more to the quality of software than 'does it work when taken out of the box'. Here some of the quality issues are 'in the eye of the beholder' and others can be classified as fairly fundamental value-for-money issues. For example a particular feature or 'cool interface' may be a specific customer requirement; while following functional specifications accurately, freedom from critical bugs, adequate documentation and adapting the program to deal with changing circumstances over an appropriate lifespan are inherent in software quality. These are fairly vague terms when used in general, but anyone who has started enthusiastically and full of confidence with a tool only to discover a rake of intractable shortcomings will be understand how flaws, and inability to fix them, reduce the edifice of fine ideas to a heap of practical rubble.

It is worth repeating that poor quality software (whatever your particular 'definition' of quality is) is by no means the preserve of hobbyist volunteers. Some one-man-bands have produced spectacularly excellent software, while some large organisations who should know better start by releasing an awful product then compound the problem by lack of support. In the next section I will be asking: What is a good method to ensure that all bases are covered when it comes to producing quality results - with particular emphasis on how cooperating volunteers should organise themselves.

Collaborative doesn't have to be FOSS

As we'll see in section 4 there may be a way for good developers to earn good money, but the overwhelming temptation is to use a low-overhead, low-revenue model and just get satisfaction from doing something interesting and hopefully useful.

At this point we're stuck for a descriptive term that is not necessarily tied to 'free' or 'open source'. I suggest *Volunteer Company*. (Where 'volunteer' may or may not mean unpaid.) Now we can look at how such a collaboration might be put together.

---

5 There are some magnificent one-man efforts and all credit to their developers, but I wonder if they might not be better with a couple of assistants.

6 The best known home of collaborative OSS projects. www.sourceforge.net

7 There is a whimsical term "bit rot" which describes how programs that once worked now fail. This is very common as operating environments and cooperating programs evolve.

8 You might be tempted to say that by this definition Beethoven's fifth symphony has 'stagnated'. The difference is that there are living experts who study it and adapt it for current day presentation.

2 Creating bright ideas - The manifesto

In this part we'll describe a foundation method for constructing collaborative software projects. It is not fiddly, bureaucratic or difficult and can be used by one-man-bands. In fact it is simply a matter of establishing the goals and filling roles. In appendix D we'll suggest a possible organisational structure - although this is not really part of the template.

Before starting : The manifesto

Which is the best system for harnessing distributed developer efforts:

- Thousands of : "I could code this...I tried...but now need help"
  - *The world needs another font management program.*
  - *I've written a framework - sorry about the documentation*

or

- Tens of : Larger applications, standards or protocols, addressing issues and exploring possibilities.
  - *Isn't it time the flaws in SMTP were addressed.*
  - *I can see how the system would work and how it fills a genuine need - here is the demonstrator.*

Clearly the latter, the difference being that the 'workers unite *before* marching on the capital'. This is the Achilles heel of free-range software. The inability to collect a team of energetic but tractable workers able to coordinate their efforts to deliver something approaching the original vision.

It seems to me that we need two things to deal with this:

1 Visionaries to specify why and how and what they want to achieve

2 A soap-box corner where interested parties can congregate

A 'manifesto' is a method to deliver the first of these.

From original manifesto...

The original manifesto is likely to be the encapsulation of a vision, quite likely of one person. The process starts with an idea promoted in the form of a paper which includes:

- Reasons why the a project like this is worthwhile (Including need, opportunity, or simply new exploration.)
- How the project might be implemented (Including organisation and technology)

... to working blueprint

Which is clarified and firmed-up by a collaborative effort in order to determine

- Exact scope (Perhaps in stages and/or modules)
- Prove the validity of the concept and availability of skills. (Due to the nature of the beast these have to be dealt with together.)

And also develop an implementation plan

- Importantly, what development model to use.
- What intellectual property/licence to use
- What technologies will be exploited...
- ...And developed or applied in novel ways (Possibly leading to an IP agenda)

And also develop an exploitation plan

- What will be delivered
- How it will extend or replace existing applications...
- ...or open new fields of application of computerisation

And (management issues)

- Success factors
- Failure factors
- Resource investment requirements
- Probable rewards

Of course the size and detail of the manifesto would be appropriate to the size and nature of the project. If all you're trying to do is rewrite your home-grown utility to make it available across more platforms with professional documentation and support then the manifesto should have few unknowns and be simple to write.

There's no need for grand mega-project objectives. It might be "*I want an add-on to Firefox that will do Foo but the implementation has me baffled. I'd like to learn but need a mentor who has done this sort of thing before. This might lead to Bar and Buz*"

Appendices A and B go into manifestos in more detail.

Implications of starting with a manifesto

The urge to code is very strong. Later I'll describe how this might be usefully channelled in the early stages, but for the time being, an ounce of preparation and method clarification is worth a ton of perspiration and mission-creep.

The person or people with the vision must be literate, able to structure a paper and able to explain the significant points. Of course this implies they must be able to identify the significant points and tie them together sensibly. Flowing literary prowess isn't required - but who wants to work with a confused non-communicator?

A first, private review might be useful to explore vague issues and clarify content. This is the place where one would hope awkward questions such as "Why is your wheel so much better than all the others already invented?" can be asked. Where there is already a group involved this is less of an issue, but a one-man-band should definitely get the rough edges rounded and cracks filled in before bringing it to the attention of the World and his wife. There is the difficulty of finding people who are detached enough to be objective while having the necessary technical background to perform the delicate task of reining in and re-directing rampant enthusiasm. (My favourite model is where a student asks their professor for guidance but I can't see how that would work for most people here.)

Publication

The next stage is publication. In a non-Internet world this would mean passing the editorial review panel of 'Manifesto monthly' then appearing in print. Of course this would now be anachronistic but the need to publish in a suitable place and the need to maintain some minimum editorial standards remains. For example it would need to be clear to a reader what the working methods and rewards would be so that they are not lured into contributing to a project under false pretences.

An obvious place to start is a SlashDot-ish forum where manifestos can be listed, categorised, perused, criticised and, with any luck, interest potential collaborators.

Typically there might be disinterested suggestions such as for example "Have you seen such and such" and "that would be really great if it interfaced with ...". OK that's good food for thought and might lead to a revision or even withdrawal of the manifesto. Hopefully there will be potential users, technically knowledgeable people who show an interest in supporting in a small way from the edges. Naturally the big prizes are developers willing to join in with the various labouring jobs. (As we'll see that's not just programming.)

Limbo...

At this stage the project should be beginning to gain momentum but there will still be uncertainties and changes. The potential team members have yet to fully commit themselves and the scope and development plan of the project are being revisited as

fresh ideas, experience and competence gather round. At some stage this pussyfooting has to stop and the 'Brothers sign the pledge'. This is made more difficult if bonding has to happen using only the Internet.

...to starting gate

At what stage does a suggested plan become a live project? Good question. Possibly by decree! Presumably some conditions have been met which in the view of the promoters and putative participants makes a full scale effort a viable proposition. The minimum list of criteria should be:

- All failure factors have been addressed.[9] For example all technologies to be used are sufficiently well understood and essential resources are available.
- As a result it has been ascertained that the project is feasible.
- The team has adequate skills, cooperation, flexibility and willingness to try.
- The objectives are comprehensively described and clearly set out
- The methods (eg development process, management structure, milestones) are agreed.
- Management is on a realistic footing.

This sounds like a tall order, and of course many people prefer to press on regardless considering this sort of thing to be only for *suits*. There is a practical way to achieve these goals:

Shake-down

Have a mini-project with the aim of proving feasibility, getting people familiar with development methods, each other and exploring the trickier aspects of the project in prototype form. All this work will pay off later, and of course prototype materials can be re-used, or used to show definitively (that is for discussions based on fact rather than hunch) why 'easy' method A won't cut the mustard and 'tricky' method B will have to be used.

I'm a great believer in prototyping - *provided* the reasons for doing it are understood and the line between lash-up and production is clearly understood. Amongst other things a prototype teaches you what you need to know about the system design (top-down approach) and the nitty-gritty technology (bottom-up approach). These don't need to be joined up very well in the middle or packaged very well on the outside. Now you have a much clearer idea of the *best way* to create a system for real and how long it will take. Suppose for example that your project involves something that might consume a lot of resources or take a lot of time - but nobody really knows how much and whether it will be a problem in practice, and if so what might need to be done to mitigate the negative effects or use a different approach. Build a prototype of the unit, put it on a test rig and do some experiments. Far better to find out before serious construction work starts than deliver a turkey or delay the main project by swapping-out an important engine.

The shake-down is just a bit of muscle flexing. Some of it may have been done before as the original promoters might have produced a proof-of-concept or something else to use as a template. Even in these circumstances it is worthwhile having some pre-development activity if only to 'get all the pieces on the board'.

[9] Failure factor : Something that could scupper the project and need to see coming in order to take avoiding action.

3 Developing quality software - Volunteer companies

Development process

Phew! All that but still nothing much to show for it. Actually of course there are three important things:

1 An agreed plan :
   - In a collaborative environment members have to 'sign-up' to the plan, you can't demand they work on Foo or else they won't get paid.
   - The order in which the parts will be made, what they have to fit in with and why they're important is clear to everyone.

2 A team environment :
   - This takes time to develop when a bunch of strangers come together
   - Remote communications are more tricky as they lack the social 'glue' that comes from face-to-face meetings.
   - A form of management. Mad inventors may not make the best leaders! The job of getting anything from a bunch of egotistical and dogmatic programmers spread around the globe is like herding cats! Lack of experience might tend to lead to the 'muddling through' style of project management - a fine, low overhead, system *provided* failure factors are ruthlessly sought out and squashed.

3 A development method
   - Common or compatible tools
   - Common standards
   - Shared understanding of process
   - Shared understanding of roles

   (More about this below)

These things are much more significant in a free-range collaborative project as the team and it's methods of work are being built from scratch. In a traditional commercial environment the framework of standards, tools, management hierarchy will be, or should be, already in place.

Roles

Of course there are a variety of jobs necessary to the production of quality software. The length of following table might come as a surprise, it did to me, and it isn't complete. There are plenty of people who are able to wear all these hats while some are good at or temperamentally suited to, or only have the time for, just one. Note : This is *not* a one role per person listing.

Analyst/Researcher	Investigator. Looking at requirements and searching for available resources.
Artist	Graphic technician and design.
Assembler	Technician in charge of build process
Asset manager	Administrator for all assets including keeping records of ownership and licencing. Might include human resources.
Contributions administrator	Monitoring and recording team contributions. Including checking compliance with standards, quality, reliability etc.
Designer	There are various flavours of designer. The common element is being able to see how to deliver a result using the available technology.

Exerciser	Harness-maker who makes sure the product and components deliver what they're supposed to.
Finance administration	Person with a certain mentality; able to follow through accounting chores, demonstrate transparency and keep meticulous records.
Gaffer	Hardware/networking infrastructure support.
Lawyer	Technician, researcher and possibly negotiator.
Leader	A good communicator with good understanding of 'what matters' and attention to detail.
Librarian	File administration and retrieval technician.
Packager	A person who creates a finished product and is responsible for its reliable delivery and installation.
Programmer (High level)	A coder who is good at or temperamentally suited to top-down programming. Scheming what the main bits are and how they interact.
Programmer (Low level)	A coder who is good at, or temperamentally suited to bottom-up programming. Typically with specialist knowledge of the technical realms covered by the project.
Promoter/ Publisher	An irrepressible enthusiast able to get across why people should be interested in making, testing, buying or trying the product. Knowledge of the market structure and conditions is very useful.
Prototyper	A skilled developer who can hack together something that 'works on the bench'.
Technical author	A writer able to communicate specific ideas and skills to specific audiences with the technical prowess to use any appropriate media.
Tester	A devious cove who investigates robustness and susceptibility to failure. This will include 'peripheral' aspects such as user documentation.
User contact	An ambassador and conduit between the technical team and users. They might arrange user trials and publish potted feedback to the developers.
Visionary	Original creator of the manifesto

Some shopping list! Manifesto writers need to have some idea of which will be the critical roles in the way they envisage the development. Some projects might need a horde of programmers while others share most of the tasks between team members with a recurring agenda item. Almost inevitably there will be overlaps and joint efforts as team members put in their two-pennyworth. (Regarding this last matter, 'advice' could be anything from interference to mentoring.)

Some people have a natural knack. These are difficult to discover, particularly without face to face meetings. If you find such a talent then make a role for it. They may be good at cadging resources or seeing through confusion and spotting the key issue when others are milling around, or simply proof-reading. (If you *think* you have found a useful talent then nurture and cosset it. You might be able to get another Technical Author 'off the shelf' but where can you get someone who produces beautifully illustrated quick-start guides which demand attention?)

Free-range collaboration can be a good opportunity to learn what happens 'on the other side of the fence', and learn new things, and try new methods because of the less formal demarcation and greater emphasis on team effort than might be found in a traditional software factory ...

... But it carries the risk of having an entrenched and useless person occupying an important role. (All roles are important.) Being able to test a CV out before 'employing' them is not possible - but it is practical to ask for a brief background and use the shake-down to discover their weaknesses.

Development sequence

Here is my standard development template. The numbers indicate definite stages. (More in appendix C) (Dots fors5 and 7 indicate lots happening at once or order isn't important within section.)

1 Appreciate the task. Get a gut-feel for (a) What (b) How (c) Amount of work
2 Top-down design (Evolving from "what will it do")
3 Bottom up design (Evolving "how will it work")
4 Proof of concept and prototyping
- Plan production
- Finalise design and review estimates of amount of work
- Check with users and confirm 'market'

6 Finalise development environment
- Write user documentation
- Write code. NB Possibly in stages.
- Get code to work (Exercise)
- Check code works (Test)

8 Build finished package of deliverables
9 Consolidate development documentation and review

There is more to the software life cycle after it goes out of the door, but for now let's simply call that 'post-delivery support' and forget about it. (There's a good reason for this split. During development developers are working to satisfy their own goals, but when users start complaining they have to submit to somebody else's demands. This can be a big strain especially where high performance head-down programmers are concerned.)

Notice that stages 1 to 6 these are 'training for the real thing'. As discussed above this is ideally suited to the shake-down phase. This isn't to say that the design stages are merely play-acting - In fact stages 1,2 and 3 are an opportunity for team discussions that serve the purpose of educating the team members and allowing them to get to know each other. If you like 1,2 and 3 are 'meetings round a drawing board'.

Stages 4, 5 and 6 have a different character. Let's describe it as "Each person to prepare for their role as leader in their specialisation with relevant bits warmed-up and ready".

Stage 7 is head-down development. In a collaborative project there needs to be some central management of this process. In a Volunteer Company there needs to be communication between coders and the rest of the team throughout in order that the 'peripheral' stage-8ers can get on with their jobs with the least practical time lag.

Stage 9 is an audit for three purposes:

- clarifying what lessons have been learnt, what mistakes were made, what worked, what didn't, who shone and who stuttered
- making sure the internal administration and project documentation is correct.
- considering "where do we go from here". (Including the support that we conveniently forgot about earlier.)

Obviously this is a gross simplification of a real-life development but from simplification comes

- Understanding and
- Control.

Management blues

For what it's worth I used to believe that leaders were born and I wasn't born to be a leader. I was wrong.[10] The reason people find it difficult is the same reason they'd find it difficult to ride a horse without saddle and reins. A management method of some sort is required before the force of the horse can be harnessed. (And of course a lesson or two and some practice.)

- A leader holds the reins
- A management system is used as a harness

The harness for a horse does two things:

- Sends controls to the head
- Takes the pulling force from the animal to the cart

A leader should *understand* and *adjust* how the whole horse/harness/cart system works but on the road only uses the reins to give instructions to the horse's head *and lets the horse do the rest*. A badly trained horse or poorly adjusted harness will be a nightmare.

Is it surprising that most of the 'doers' would rather be getting on with useful work than wasting time in meetings and writing documentation that nobody will ever read? The ennui of committees, trying to explain to the dim-and-entrenched and being so much more knowledgeable than the others but only having one voice in a 'democracy' is enough to put off anybody.

This circle must be squared! Volunteer Companies need management - management by the people with the most 'fizz'.

Management model

For a start let's assume that the 'management' is a bit vague about what it should be doing. That's easy: Provide some templates and checklists that require answers, structured analysis and a sound basis for team discussions. That's what this article is all about.

Now how about command and control? Plenty of options from "It's my party, I'm the boss, accept it or leave" to "join the everyone's equal coder's commune, we can't wait for your opinions". Umm... Something more in the middle, something that recognises the value of contributions and the value of letting people get on with their personal micro-missions without interference will probably be more suitable to a Volunteer Company.

Of particular importance to the Volunteer Company is distinguishing between the team building, project definition stage and the noses-to-the-grindstone stage. In an organisation that evolves over a longer period these phases are less distinct and have a traditional leader-committee-followers structure that may have evolved over years together with objectives and methods that are already in place. In essence a VC must stick a 'build the team' task in front of it's other plans.

Trierarchy

Something that I've noticed is that a group of three works better than one of two or four. In appendix D I've described a possible organisational structure that uses this concept. In short, members work in groups of three with one link 'up' and two 'down' although 'up' and 'down' are not necessarily absolute. The purpose of this is to *optimise the setting and delivery of goals during the production phase of a project*, freeing members to be clear what part they have to play and get on with it. This is *not* suitable for early days sorting out in the team building phase where larger forum scope is required in order for members to weigh up the wider picture.

Management conclusion

It should now be abundantly clear that Volunteer Companies will have a distinctive lifecycle:

Seed	Manifesto
Shoot	All energy goes into growing strong quickly
Plant	Effort now goes into fruit-bearing

The management of each stage is different:

Seed	Proselytiser on a soap box
Shoot	Mêlée that sorts itself out by technical roles and into...
Plant	... an organisation tuned for creating and assembling components

The two objectives of the shake-down should now be clear : To clarify the technical objectives and to establish a working organisation.

There may be other ways of arriving at the 'Plant' stage without going through a shake-down but the objectives remain just as important.

As we'll see in the next section, there is no need for the plant to die after a single flowering. It is a much more efficient and effective process if a VC doesn't need to go through the team building stages all over again. "That was fun - Let's do another"

By the way, notice that this is nothing to do with open source or working for no pay. This is about making efficient use of talent to deliver quality software.

---

[10] I've now graduated to leading from behind, by example and subversion. Programmers ought to make good leaders : They can see how to make things work, they can draw up and follow a plan, they can see through walls, they work towards clear objectives, they are always asking "What could possibly go wrong" and are basically optimistic. OK some lack the necessary social skills - but even then, being technically superior in a technical environment goes a long way.

4 A model for financial and technical development

Making a living

Which is more 'valuable' : Firefox or Nelson's column; Sendmail or the Mona Lisa; Shields Up[11] or a free concert in the park?

The value to society of 'art' is already established. There are grants, scholarships, bequests, festivals, and sponsorship. Then there is a large commercial arm of the arts from mega-corporations to writers and buskers. I believe that a similar approach to the 'programming arts' would be extremely beneficial.

Firstly it would remove the necessity "to be commercial". There are probably thousands of unpublished novels that are far better than the branded pot-boilers and hundreds of biographies that are far more interesting than 'auto' biographies of celebrities. The same applies in software: "What we want is something to steal another's market not make our own." not to mention the infamous "I think there is a world market for maybe five computers."[12] (Now if *I* was handing out money I'd want some expectation of quality - hence the previous sections.)

Secondly, the people who are intimately involved with technology are often the ones who can see how it can be improved. Also they're probably the ones who are best qualified to make the changes. Supporting these innovators helps build a culture of continuous improvement and challenge. It is interesting to note the technological developments in the spam wars. Currently there is more spam traffic than there was total traffic a couple of years ago[13] and everyone pays for it. So far society hasn't been very forthcoming in giving proper support and nourishment to the good guys.

Thirdly, unless there is a thriving and competent source of variety there can be an awful stagnation and entrenchment and exploitation through monopoly.

- PGP was a classic example of how 'official' restrictions 'for our own good' can be challenged.
- Independent developers can reduce dangerous dependence on mono-cultures. Linux and Open office for example.
- Ogg is an example of an independent development that avoids a potential patent stranglehold on streaming formats.
- Whilst some might say that having two main graphic desktop systems for *nix (Gnome and KDE) introduces unnecessary confusion, most people agree that the competition is a good stimulus to further development and choice is a good thing anyway.

Fourthly, this can be a very low cost operation giving very good value for money.

- Highly skilled and knowledgeable people contributing very effectively
- There are potentially many more testers and proto-users available to guide progress from the 'Can we get it to work' stage to the 'It works' stage to the 'This is really good' stage. Developments that work on fixed budgets and timescales can't go through these iterations.
- Ab-initio developments can establish standards that people actually use without having to engage in patent-wars, royalties and proprietary stitch-ups.

Fifthly, amateur, low-budget, projects can be a good training ground for developing developers. In depth technical knowledge, skills and practice working in a disciplined team are very valuable assets and continually need replacing. Any society that expects to thrive without bright and experienced software developers is kidding itself. Traditionally this function might be done by universities and research institutions - Long may they continue to do so, but there are plenty of specialists who for one reason or another will never visit such an institution. (It would be nice if they could reach-out to non-students and non-researchers.)

Sixthly, if a lot of software (or content) is 'already paid for' it means more users can access it. This is the ethos of public service broadcasting and equal opportunities.

Seventhly, governments and large organisations have been sponsoring targeted software development, experimental research and training for a long time already. So this isn't a shockingly new idea anyway. The principle of supporting software development and standards is already established and the value is understood. What has yet to be recognised is that there's a new paradigm emerging with plenty of *potential* (and a few existing notable examples) to out-perform the traditional methods.

All the world's a stage

What if collaborative developers were theatre companies? Individual 'donation-ware' developers might be equivalent to buskers and poets. The large software companies might be equivalent to the film studios and music corporations. There's plenty of meat on this bone - some interesting similarities - and differences in 'how the business works'.

Scale and variety

How much like an amateur dramatic company is a volunteer software collaboration? Possibly quite a lot. The amount of effort by individuals might be broadly similar; the writer (in the software analog the manifesto author) may not be involved; there is an element of direction - that is getting the best out of the participants to produce a particular vision; there is an element of production - that is organising the nuts and bolts and selling the seats; results can be very variable - often superb and well worth the ticket; there is the element of casting and needing certain technical skills; and there is a clear understanding that although the whole thing is being done as a team, in the first instance members are responsible for their clearly defined role or job. If amateur dramatics can be such fun then perhaps the same pleasure from camaraderie can be developed by a good software group president.

As well as the amateur companies there are small professional companies that tour and build up loyal followings. Often these survive by grants, guarantees on the income side and an infrastructure of goodwill and part-time employment for trusted, versatile technicians on the expenditure side.

Large permanent professional companies usually rely on substantial state funding. One of the justifications for this funding is the need to have really top class work across the board. As identified above, universities and research organisations might be the equivalent software analogue.

---

11 A free on-line security test from Gibson Research Corp. www.grc.com

12 Thomas Watson, chairman of IBM in 1943. Another classic : "This 'telephone' has too many shortcomings to be seriously considered as a means of communication." Western Union internal memo, 1876.

13 By a back of the envelope calculation.

Single-show major productions are on a different footing. They start with a 'script' which is developed into a production plan with stars pencilled-in which gives a sum of money that needs to be raised from private backers in order to proceed. If lots of tickets get purchased then the backers profit, if not they loose. This model has been the traditional one in the software industry. Perhaps it is only suited to the larger, more complex productions.

What can we learn?

As Shakespear's famous "All the world's a stage" soliloquy suggests we compare life to a play so let us now reflect on how true or insightful comparing software development to theatrical production really is.

Large productions rely on large financing and ticket sales. Selling tickets is of course analogous to selling licences. Ticket sales mean promotion and a 'finished product' that can be passed through a distribution chain according to an established financial model.[14]

The many thousands of amateur companies only look to recoup their costs. Interestingly they charge a *range* of up-front ticket prices and don't use the 'if you've enjoyed this then put some coins in the box' method. Suppose you've found some really good software - how much should you donate? Nobody knows. There isn't an 'about right' socially acceptable figure. Not a lot of sleep is lost ignoring shareware authors - we all like something for nothing. There is no framework, no box office and nobody to check your tickets. So are we stuck with software that's "worth every penny"? One model which has been used is to give away the software but sell the bits that go with it, productivity tools and support. This is like having a free cabaret but being over-charged for drinks.

Reviews play an important part in developing theatrical and literary careers. Software reviews (where they exist) look at comparative features not the value of the key contributors. "Vera Smith's cumbersome UI is rescued by the superlative help system we've come to expect from Jim Scoggins". They tend to look at early versions which is equivalent to reviewing a rehearsal. In the performing arts there are freelance performers, permanent technicians and commissioned writers and specialists. The same thing could exist in software if only you knew who were at the top of their profession, had a track record, and were available locally at a reasonable fee.[15]

In the performing world there is commentary and news about who's doing what by people who know the business. Journalists are tasked with being spotters of trends and flaws on behalf of the 'man at the console', are useful conduits for information, and can apply pressure to otherwise unapproachable organisations to be 'user friendly'. This world-of-hype works. It creates and maintain an interest in the film and theatre industry. From the manifesto to the release of my next bytebuster "Indoor Hanglider III" it's good for people to know I'm looking for talent when recruiting, and to become excited by 'inside' progress reports, and scan the reviews when it gets released. If I can't be bothered to do a little bit of public relations work then why should anyone else be bothered about it? The average person needs some filtering and sorting to be able to access what matters to them. Just one example might be a review of new manifestos, roughly equivalent to book reviews. Obviously this requires a reviewing infrastructure.

Some outfits such as folk, musical and dance troupes get paid to perform at festivals. The festivals are typically civic subsidised cultural events. The software equivalent might be an annual delivery of projects at a particular trade show or government sponsorship for groups that adapt foreign-language programs for local use. In this last case it shouldn't be too difficult for a government to see the value of maintaining a skill-base capable of doing this work. Note that these involve semi-permanent groups with particular specialisations.

With large one-off productions such as films a company is collected specially to do the job. Is this the best way to organise small software development projects? Probably not, it seems silly to have to go to all the trouble of building up a team then let all that effort get dispersed at the end of the project. Smaller theatrical productions tend to be the preserve of small permanent companies who tour and adapt or hire extra bits as required. Software development could use this model. "Following on from this year's yacht design workbench, next year's production will be a boat owner's toolkit".

---

14 Interestingly even with large software productions we are seeing a shift from the 'movie model' (finish and move on to something different) to the 'serial model' with service packs, and security updates.

15 We'll know when the world accepts developing software is 'cool' when there's a programmer's Louella Parsons.

5 Conclusion

Note : 'Software development' is used here as shorthand for 'Information technology development'. For example this could cover standards, robotics, training materials and hardware packages. Participants need not be individuals, for example there are plenty of examples of standards and technical working groups staffed by big corporations.

The need for more collaboration

We've seen how the current collaborative groups tend to be (though once again I take my hat off to the few that shine) too amateurish in getting to grips with the establishment of a viable group and covering all the bases required for quality software. One-man-bands proliferate because of the effort required to recruit, indoctrinate and manage contributors.

Skills and in-depth technical knowledge need nurturing. A useful environment for this is as part of an 'amateur' group where people can develop their techniques and broaden their experience. The Amateur dramatic analogue shows us that this can be done for fun as a hobby with surprisingly good results.

Increasing the available man-hours by collaboration means larger projects can be successfully completed. Increasing the available skills through collaboration means all aspects of the project can be produced to a good quality standard.

The possibility of being paid

Earning a living depends on being able to obtain real money from somewhere. This will remain a difficult and irritating necessity but it will be *impossible* unless results of good quality are produced. The more professionals that can be employed the more everyone will raise their game.

"Volunteer" in VC doesn't necessarily mean 'unpaid', but in the nature of things, regular pay-checks might be unlikely. Co-ops exist around the world in many fields which might provide suitable models. The VC 'commune' model could evolve into the VC 'corporation' model where commercial exploitation is the primary objective.

It is possible to visualise situations where specific units of work created by a contributor are re-sold. In this case the VC might (for example) give 50% of the profit to the contributor, with a sliding scale for how much team-effort has gone into wrapping the work up to commercial standard.

The efficient system

It seems silly to go to the effort of creating a viable and effective software development team just for a single project. A better method might be the Volunteer Company which sets out to do work in a certain field rather than produce a specific product. For example 'VC-New paradigm email' might work on modules used to build clients, authorisation for organisations, simple digital signatures and many other practical steps in the evolution of email. At some stage it might be able to sell-off some of it's intellectual property and for incorporation into 3rd party products. One of the reasons it would be able to do this is that it would have a number of people in the project tasked with communicating with other interested parties.

The independent Jack-of-all-trades ethos may work for micro-projects but successful authors concentrate on writing while publishers do the production and promotion work. Each does what they're good at. If the creation of software ideas is not intimately tied to the production process this provides some useful flexibility.

- Manifestos become free-standing and possibly commodities in their own right. A bit like plays and film scripts.
- Development teams are free to choose which ones they think are viable, worthwhile, suited to their field or technical abilities.

This can lead to authors being commissioned, technologists being hired to troubleshoot and designers being asked to submit designs. All-in-all a much more fluid system for getting the right skills used for the right thing.

Epilogue

So that's the vision. A framework for setting up collaborative software development efforts in order to boost quality, and a framework based on the performing arts model for nurturing talent, giving simple pleasure to many, offering paid employment and performing a useful civic service.

I have provided some checklists in the appendices to assist people who want to have a go at writing manifestos and organising VCs. I would suggest that these would be useful to anyone contemplating a project, even if it is only a 'couple of man-days'. These are an attempt to remove the 'Continually evolving mission', 'group management is too scary' and 'factoring-in quality sounds tedious, difficult and basically unnecessary in this case' barriers to quality team efforts.

Pre-publishing review and publishing of Manifestos requires some Internet and technical infrastructure. This will have to evolve. My suggestion if for there to be a 'Manifestival' every three months, perhaps each with a different sponsor and flavour, at which manifestos are on public display for a few days. This generates a 'market' atmosphere where all interested parties know where to be to see what's on offer and who is interested.

The VC model can succeed without financial support, but civic-minded bodies should soon be experimenting with sponsorship, training and using them as a socially beneficial resource. This will necessarily be tentative at first until VCs show what they are capable of.

In order for VCs to inspire confidence in their usefulness there will need to be some infrastructure for evaluating the quality and spin-off value of previous projects. This is important but at the moment it is too early to say what forms this might take.

What to do now

1. Establish a Manifesto/VC net meeting place
2. Encourage the publishing of some manifestos and discuss the IP issues.
3. Encourage the formation of some technical area specific VCs (as opposed to project-specific ones)
4. Seek donors and create a distribution method to dangle carrots or otherwise encourage VCs or commission manifestos.
5. Polish and expand on the practicalities and benefits for all concerned, including promoting this as extremely good value for money for large funders who are concerned to develop skills and locally useful results.

As I don't work for a wealthy foundation and don't know any, these matters are now handed over to the reader.

6 Appendices

A Manifestos
B Example manifesto
C Tips for timid team leaders
D Trierarchy

Appendix A
Manifestos

1 Purpose
2 Components
3 Simple assessment criteria

1

A manifesto is a statement of vision sufficient to attract potential collaborators. It describes what it is intended to produce and the value of doing it. Then some thoughts on the mechanics of production; including what work can be re-used as a start point, what special skills or knowledge would be useful and any pitfalls or possible bonuses.

This is then published in a suitable place where potential collaborators or developers can assess it from their own perspective, suggest alterations and indicate an interest.

Converting a manifesto into a complete project plan including the scheme of collaboration being used is a job for the production team in conjunction with the manifesto author. To make this simple the author should indicate the sort of licencing scheme they prefer.

2

There are three basic parts to a Manifesto:

- What it is proposed should be done.
- Reasons why the a project like this is worthwhile (Including need, opportunity, or simply new exploration.)
- How the project might be implemented (Including organisation and technology)

The level of detail will vary according to the size of the project and anticipated technical difficulties. It is possible to visualise a manifesto as compact as:[16]

Task : Translate the UI of program Foo written for Bar speakers into language Buz using native units. There are 12 screens, 40 database fields, 50 help screens, a manual and 3 main routines involving scientific calculation.
Also : New work : Produce a cardboard ready-reckoner.
Why : Our farmers waste an estimated $4,000,000 each year on mis-applied fertiliser.
How : 1 - Check scientific basis is applicable for our climate/soils.
2 - Persuade original author to let VC <u>generically</u> internationalize program
3 - Use Bar-Buz translators (National Agric. Inst.?) to change texts
4 - Validate texts in the field
5 - Validate changed calculations
Notes • It might be possible to create the ready reckoner in time for the upcoming season. (This might be a good shake-down task.)
• Informally, the original owner will 'free' the software on the understanding that the internationalizable version will also be 'free'.

This lists 'What' and 'How', or if you like Objectives and Methods. Notice that while the manifesto author is fairly clued-up about the 'How' their analysis is at a high level and doesn't, for example, specify how many man-hours or go into the complete skills roster. These matters will be developed by the people who are actually going to do the work who might be expected to have a better grasp of their resources and practicalities of managing them. Obviously this is hardly enough to build a proper project plan on and there will need to be much more detail, discussion and fact-finding when the VC has agreed that this is a project it would like to get involved with *in principle*.

Notice that the author has clearly indicated the multi-disciplinary nature of this project. In a sense all projects will require a bunch of different skills, but this one in particular has quite a large non-computing specialist level.

In this example we don't know anything about the author they might be the student who developed the program as part of a university course and now sees wider potential in another country, or a government official looking for the right skills for a properly funded project, or a software developer desperate for assistance from the Real World.

The *How details* are in the 'to be determined' section
a Exact scope (Perhaps in stages and/or modules)
b Proving the validity of the concept and availability of skills.

At the same time the *implementation plan* will be being firmed-up
a What (software) development model to use.
b What intellectual property/licence to use
c What technologies will be exploited...
d ...And developed or applied in novel ways
In the example the author has hinted at the last three but may not be able to specify in any more detail. The experts in the VC may have their proven software development procedures so this aspect might be best left to them.

The third part that requires more detail is an *exploitation plan*
a What will be delivered
b How it will extend or replace existing applications...
c ...or open new fields of application of computerisation and management issues
d Success factors
e Failure factors
f Resource investment requirements
g Probable rewards
Our example author has sketched these out but not gone into any detail. Though they've talked about translation and the need to check the technical translation 'works in the field' they haven't explicitly said that "This project is a non-starter if we can't do the technical translations." If they had done so then they might have concluded "With potential savings of say $1,000,000 to the nation in the first year, getting this project completed and released before the end of the next rainy season should be a priority - The tight timescale suggests professional talent should be made available as a matter of urgency."

If we were reviewing this example manifesto we'd have some things to say about lack of detail in this section. For example all we know for sure is there will be a cardboard ready reckoner. No idea what form the software will take, how many copies, what the distribution channel will be and so on. Most of these details should be specified up-front by the manifesto author.

[16] At the other extreme is a 76 page design I wrote for a National Pothole reporting system.

3

Simple assessment criteria as a checklist for authors to avoid this last problem and for reviewers to categorise their opinions form a framework for self-evaluation and peer-evaluation. Here is a suggested list:

a Does the document conform to the basic Manifesto structure as described above?[17]
b Is the 'What the project should achieve' section well written so that a reader can grasp the essential concepts and overall structure?
c Does this demonstrate a good grasp of the current state of the art?
d Does the 'Why the project is worthwhile' section explain clearly the benefits?
e In the opinion of the reviewer, are these benefits really as valuable as suggested?
f Has the author suggested a reasonable 'How to go about this project'? Too vague or too prescriptive? Too ambitious or too narrow-minded?
g Has the author made a reasonable stab at an implementation plan or indicated what the issues and unknowns are?
h For each of a-g in the exploitation plan is there enough detail and realism? Are there items that have been left out?

A shorthand scheme might be

- Ambition — Already been done ... through to ... far too much to expect.
- Tech. difficulty — Practical and realistic? Too optimistic or easy?
- Usefulness — Benefits likely to be seen in practice or just hype.
- Clarity of participation model — There may be constraints at the manifesto stage on the way the software is to be developed, released and exploited. Eventually these become more significant for individuals deciding whether to participate in a VC.
- How well are success/failure factors identified

Appendix B
Example manifesto

Project name : AX

Author : Peter Fox
peter=inventor@PeterFox.ukfsn.org

Date : 27th December 2006

Status : First release.[18]

Result : A simple method of obtaining accents or unusual characters from a plain QWERTY keyboard.

Job to do : Reimplement a proven but very old home-grown utility for as many alternative operating systems as people see fit.

Basic idea : Repeatedly pressing a key cycles through alternative characters. In simple terms:

Press e	Result : e
Press e again	Result : backspace é
Press e again	Result : backspace ê
Press e again	Result : backspace è
Press e again	Result : backspace e (ie cycled back to unadorned character)

The cycles of alternative characters is user-programmable and not limited to multi-lingual characters, so for example c-©-c or $-£-€-$. In practice this works very smoothly and quickly as say 'cc' soon becomes automatic for somebody who uses the copyright symbol quite a bit. There is no hunting for keys or trying to remember which shift is required. And if you go 'too far' or need the unornamented character you can just cycle to the beginning. This also works if for example you wanted to *later* change é to è.

Benefits : This is a very simple system for people to use whether for occasional special characters or frequent use in accented languages. In particular, a plain keyboard can be enabled in whatever way the user wants.

Original implementation : The original implementation used a macro-key recording feature available in Windows 3 (I told you it was old!) that trapped target key presses and passed them to a Visual Basic routine for replacement by backspace then 'next character'. The 'current' character was determined by examining the character to the left of the cursor and cutting it into the clipboard. So in fact it is not really a 'two presses of key' method but a 'can we modify the last character on the screen' system. It worked!

New implementation : Whilst the original implementation proved the utility of the concept; it was crude, relied on obscure Windows features and clobbered the clipboard. My guess is that it will be difficult to implement a one-program-fits-all-OS because of the low-level operations involved, however it should be possible to provide a high-level specification, standard UI and standard configuration format to allow country-specific files to be preset and distributed regardless of OS.

Developer skills required :

- Bottom-up, OS-specific programming for what should be standard calls to keyboard and screen.
- General design and coding for UI and configuration file format
- Multi-lingual documentation

17 **And one presumes, some formalities related to indexing and key word searching as developed in any manifesto cataloguing scheme.**

18 September 2008 update: One day, in an attempt to avoid real work, I implemented this for Windows and is now available free.

- NB The manifesto author will NOT be providing project management.

Constraints :
- Idea : Released to the public domain.
- Source code of original implementation : VB program files (very old version) and user documentation in Windows help format available free from Peter Fox.
- Name : Copyright Peter Fox (Available free to any viable project team.)

Implementation plan :
(a) Software development model
  1 Briefing of principles : Nothing difficult
  2 Prototypes for each operating system
  3 Standardised design specification and UI agreed
  4 Production quality coding and user documentation
  5 Packaged for stand-alone release
  6 Offered to o/s providers
(b) Intellectual property/licence to use
  As developers see fit. (There is no reason inherent reason this shouldn't be a closed source project.)[19]
(c) Technology base
  - This relies almost entirely on (presumably documented) operating system calls.
  - There could be issues with character encoding systems that are not one byte=one character
(d) Technology development
  The principle of what to do when combinations of character to the left of the cursor and certain key press occurs might be extensible. For example an underscore after a full stop might command applying a style to the whole preceding sentence or decimal aligning a number. Whilst 'certain special actions only work in special circumstances' might be confusing for the novice and irrelevant for people who don't do 'that sort of thing', magic keys, especially where there might be a dearth of keys such as a mobile phone, might be really handy short-cuts for the type of person that likes to make use of them.

Exploitation plan :
(a) Deliverables
  - Single (open) standard for storing configuration data
  - Stand alone 'this will tweak your o/s to do the necessary trapping of events' utility. One per operating system (or o/s variant if required). With user documentation etc.
  - Developer-level documentation to allow 3rd parties to internationalise the UI without any further involvement by the original developers.
  - 'Internal' documentation (could be closed even if generic program is not charged for) describing how the technology might be incorporated into specialist applications.
(b) It is believed that this is a completely new approach to providing accents and symbols. By gluing it to the operating system this makes it available to all applications.
(c) -
(d) Success will depend on
  - Good design : Simple UI, ease of use and configuration
  - Good grasp of details of o/s calls (including variants and wrinkles)
(e) Failure could arise from
  - Complexities of o/s causing some operations to fail in the field. (Prompt for good test plan.)
  - Difficulties with multi-byte encoding
  - Lack of globally available o/s calls for keyboard event trapping and cut-and-paste.
  - Applications that refuse to let the o/s get hold of necessary key presses
(f) The development resources required would be quite limited but specific.
  - Good designer who understands the importance of use and create a simple architecture that makes internationalisation simple.
  - 'Expert' programmer who are familiar with system-level o/s calls...
  - ... one per o/s should be enough (This is not a large job)
  - Experienced packager who can create o/s (and variant) specific versions in a simple or automated installation method. (Features may need to be switched on/off according to o/s variant and specific other applications being used.)
  - Test manager with access to a wide selection of o/s and variants either directly or via a network of tame testers.

  Note : The software would presumably be developed using a portable system-level c-ish language. It makes sense for this to be standardised across the whole project.
(g) For the general public it may be too difficult to charge for the finished generic program and make a profit worth bothering about. (See also the notes below.)
  - It may be possible to sell finished programs to large institutions in bulk pre-configured for their specific purposes.
  - It may be possible to sell the proven-in-practice code to o/s manufacturers so that can build it in seamlessly to their products.

Notes :
- It might be easy for clones to incorporate parasitic malware. For this reason it would be a Good Thing if the end-user generic deliverables were completely free and easy to adapt for foreign language use.
- Although this has been written as an illustration of a small Manifesto for the purpose of showing what such a document might look like, it is 'on offer'...
- ... But until the global Manifesto cataloguing infrastructure exists it isn't possible to tie this to a discussion and team recruitment forum.

[19] Although the author suggests that making the source available will act to improve its quality because of the knowledge that it will be examined by 'hackers' who like working at this level.

Appendix C
Tips for timid team leaders

Organising a collaborative project will strike fear and loathing into the hearts of many highly qualified, motivated and competent software developers. Well yes it would do - it's like finding yourself in a strange country: There's a new language to learn and different ways of doing things.

Here is a survival guide. The effort is well worth it. Firstly you achieve a lot more as a team, and secondly you rapidly become tuned in to what makes teams work both as a leader and what makes a good cog-in-the-system. Most experts like to do both : Contribute to policy, coordination and oiling the wheels; and also to be have a private domain where they can do really useful work without interference.

How to start building a team

Easy : Be a *team builder*. A team builder is someone who can convince others to work together towards one objective. The team builder has to communicate, that is educate and sell, what it is they're trying to do together.

1 It would be a great idea if ...
2 Who's up for it?

You won't get anywhere it if you can't explain what you're trying to achieve.

You must also be clear in your mind what the success and failure factors are and have a back of the envelope 'what to do - how to do it' plan. For example if you're planning a skiing party there are probably serious time constraints on getting essentials organised. What are the constraints and what are the essentials? These may need discussion but by having thought about the key aspects of the project in more depth than the others you can demonstrate your superiority in the fitness-to-lead stakes. Also this is good ammunition when you have to head-off daft or disruptive suggestions or force a decision.

Then for example (in this order)

- Come on Charlie - You'll be ace doing the twiddly-bits
- Come on Beryl - Charlie is doing the tricky bits
- Come on Andrew - The others will be relying on you
- Come on Dawn - You'll see the others in action and pick up lots of useful stuff
- Come on Eric - A couple of days work and then we can sell at a big profit

Different members may have different motivations but all are working towards the same goal. It is your job to find what 'buttons to press' to get them to join. *That's the hard bit over with.*

Immediately a candidate is 'sort of' recruited you should give them some task involving mental planning. Normally you'd ask them about how they would and the team should go about something that they might be interested in.

- I'm glad you're here as I'm worried about how we organise the publicity (and you're person who knows about these things and can give us advice.)
- It looks like you'll be the best person to look after the money. It might be worth having a peep at 'Bookkeeping for beginners'.
- I don't know if you've any ideas where we can find ...

Software developer's role checklist

There's more to developing software than sitting at a console and writing a program that 'works'. Any serious project will need the application of all sorts of skills. Whether you're the captain on the bridge or a stoker in the engine room you ought to be familiar with the following list in general and how it is put into practice for your particular project.

There is no one person to one role suggested here. Typically all projects will use these roles but they might be carried out by one person or a horde.

Analyst/ Researcher	Investigator. Looking at requirements and available resources.
Artist	Graphic technician and design.
Assembler	Technician in charge of build process.
Asset manager	Administrator for all assets including keeping records of ownership and licencing. Might include human resources.
Contributions administrator	Monitoring and recording team contributions. Including checking compliance with standards, quality reliability etc.
Designer	There are various flavours of designer. The common element is being able to see how to deliver a result using the available technology.
Exerciser	Harness-maker who makes sure the product and components deliver what they're supposed to.
Gaffer	Hardware/networking infrastructure support.
Lawyer	Technician, researcher and possibly negotiator.
Leader	A good communicator with good understanding of 'what matters' and attention to detail.
Librarian	File administration and information retrieval technician.
Packager	A person who creates a finished product and is responsible for its reliable delivery and installation.
Finance administration	Person with a certain mentality; able to follow through accounting chores, demonstrate transparency and keep meticulous records.
Programmer (High level)	A coder who is good at or temperamentally suited to top-down programming. Scheming what the main bits are and how they interact.
Programmer (Low level)	A coder who is good at, or temperamentally suited to bottom-up programming. Typically with specialist knowledge of the technical realms covered by the project.
Promoter/ Publisher	An irrepressible enthusiast able to get across why people should be interested in buying or trying the product. Knowledge of the market structure and conditions is very useful.
Prototyper	A skilled developer who can hack together something that 'works on the bench'.
Technical author	A writer able to communicate specific ideas and skills to specific audiences with the technical prowess to use any appropriate media.

Tester	A devious cove who investigates robustness and susceptibility to failure. This will include 'peripheral' aspects such as user documentation.
User contact	An ambassador and conduit between the technical team and users. They might arrange user trials and publish potted feedback to the developers.
Visionary	Original creator of the manifesto

Notes

- This is not a definitive list. It will evolve with experience, specific requirements and luck in finding geniuses in a particular field.
- The assembler's role is to physically construct deliverables. This is an internal technical role that supports the rest of the team. The packager's role is to develop, maintain and validate the deliverable packages. This is part design, part user liaison, and obviously needs to specify aspects of assembly.
- What is an 'asset'? Classically Men-Money-Machinery-Materials. This probably needs careful thought in each instance - to deal with IP for example.
- Contributions need to be *administered* and *managed*. The management might involve quality feedback and standards compliance. This is a job for a mature person who can be relied on to administer while getting the best out of contributors.
- The exerciser may build harnesses for internal use, but can usually, very usefully, combine these with examples, tutorials and 'test'[20] materials for users.
- Programmers are funny creatures who can be superb at some aspects and miserably incompetent ,or just miserable, at others. The split I've used between those that work in bold strokes with a large canvas from the general to the specific and those that are fanatical about microscopic details and hidden subtleties may not be the best way to allocate your available talent. In any event you need to find out the strengths and preferences of your programmers to be able to make the most of them.
- The exerciser supports developers trying to get the system to work. The tester tries to break the system. These are quite different outlooks although of course both will liaise closely with the programmers.

Developing competence and confidence

It is only after starting a job that people find out what it's *really* all about. When a team starts work members find out about each other's knowledge, skills, trustworthiness, reliability as well. One thing is guaranteed - Their standards and methods of working will not be the same as yours. So you learn to work together sharing tasks as appropriate. The thing that makes a good team is that, as well as getting the job done, everyone helps other team members get better. "Here, let me show you", "George has written a procedure which checks you're working on the latest version", "Please use the official bug reporting system" and "As we've had a rash of easily preventable coding bugs we're going to hold a visual bug-spotting contest on Tuesday for all coders to have a go at."

If you're used to being a lone developer then you'll probably think that spending time 'bringing the others up to your standard' (as you see it) is a diversion. Well in one sense it is - a useful and productive one that soon raises everyone's game. You know when it's working because bright ideas and smart work and awkward questions are 'coming up from below'. Talent and competence is always short supply, and the best sort is that which you've helped to develop yourself.

There are some people, possibly 10%, who are never going to get the hang of being in a team. Side-line them, kick them out or find some harmless niche activity. Most however are only in need of a little guidance:

- When something is risky be ready to catch a problem early on.
- If things go wrong identify the problem at the source and if there's a person at the source go through how to deal with it with them.
- Many people are afraid of taking necessary bold steps because they don't have confidence. A team leader will inspire confidence. (Ask yourself what steps someone could take to make *you* more confident) Encouragement, good tools, good briefing, being told how others have confidence in you, technical and moral support, and frequent "you're doing well" feedback.

Confidence breeds competence - Build some today!

Teams feed on achievement. When there isn't much to show remind members of the worth of their individual contributions, or arrange some extra-curricular activity where people reinforce their togetherness.

Early days - shake-down.

Teams with simple physical tasks and no technical difficulties or high level of skill required can become efficient quite quickly - a couple of hours. But each of the following doubles the difficulty

- Technical ability and skill required
- A wide mix of roles
- Intermittent interaction
- Opaque interaction
- Abstract activity
- Inexperienced management
- Participants who haven't fully 'signed-up' to the team ethos
- Uncertainty about what tasks are (and who they 'belong to')
- People drifting in and out

For a collaborative software effort mediated via the Internet all of these alarm bells are quite likely going to be ringing at once. Run-away-and-hide sounds like a smart move!

Practical team building

Fortunately there is an answer.
Step one : Build the team
Step two : Apply the team to the task

We've already looked at team building in the abstract. Here's the practical. Sports teams have friendly pre-season matches to get up to speed, fresh military units go on training trips and your fresh team needs the same experience. (In fact often you're *building the whole system from scratch* whereas the sports team and military unit will have their rules, equipment and systems provided.)

[20] Check to see everything is working after installation etc. Not thorough testing.

1 Collect the team
2 Make sure everyone knows their primary role
3 Brief for **trial run**...
... clear but limited overall and specific objectives
... get up to speed on your bit
... and shout if anything is not working right
4 Confirm everyone is ready and knows what their objectives are
5 Off you go
6 Make sure people that should be communicating are happy
7 Have a finish point
8 At the end ask for suggestions for improvement ...
... and if possible get a respected outsider to give a review
... and pat on the back.

In software development you might aim for a general proof of concept or a simple but related project that is a taster for more serious work to come.

Project planning
There are two main things to bear in mind here:
- An orderly sequence of steps
- Ways of measuring progress

One way to generate a plan of work is to start at a top level (let's call these phases) and work down in more detail (steps). Each phase or step will have some objective that can be reviewed. For example we've just been looking at the shake-down phase where the objective is to build the competence of the team and its members so everyone is confident they can do a good job on the main project. Existing teams may not have a shake-down phase, instead a familiarisation with project objectives phase which could last just minutes. One possible sequence is:

1 Recruit. Refine objectives and methods
Objective : All necessary resources vouched for. Members committed and understand what's being attempted and how they will play their part.

2 Shake-down
Objective : Raise the general efficiency of the team. Ensure all members are confident they can do a good job and are not having problems. Clarify and contain any outstanding technical issues. Review the main plan's objectives and methods.

3 Production
Objective : To produce according to the main plan.

4 Tidy-up
Objective : To establish all the necessary resources for support and continuing exploitation either in-house or by 3rd parties.

Obviously the production phase is going to contain a loads of steps many of which will not be synchronised. Each role may have a separate agenda with a certain amount of 'this has to be ready before we can do that'. Your planning job is to make sure that each step has a clear objective. Often the people performing the role will be able to fill in the steps and details and of course estimate the timescales.

As mentioned above, teams thrive on achievement. Therefore the good planner will find some milestones that indicate progress that many people will have been working towards. For example the UI design might be 'released for discussion' and later 'approved'.

(By the way, a good team leader can manufacture 'progress' *after the event* almost at will. These might be general such as 'we now have six completed modules' or specific 'The testers have passed the Foo module for full RFC9999 compliance' or personal 'Sarah rebuilt the server over the weekend'. The *planner's* job is to provide objectives to work *towards*.)

"Objective" appears a lot here. The key thing is that an objective is measurable. (Whereas an aim is an ambition that might mean different things to different people.) A typical task for an exerciser is to provide some sample data. The objective will be to provide a certain amount for a specified purpose (by a certain date). Perhaps the A-module stuff this month and the B-module stuff next month. Team working practices or standards will dictate what documentation or other notes are supplied in addition to the raw sample data. In my opinion it is a really good idea for the beneficiary of the work to check it out *as soon as possible* and give feedback. "Thanks, that's just what I wanted" or "It isn't doing foo when put through the bar-mangle". The smaller the tasks the easier it is to focus-in on them and keep discussions to the point.

Software development template
This is my standard development sequence.

1 Appreciate the task. Get a gut-feel for (a) What (b) How (c) Amount of work
2 Top-down design (Evolving from "what will it do")
3 Bottom up design (Evolving "how will it work")
4 Proof of concept and prototyping
- Plan production
- Finalise design and review estimates of amount of work
- Check with users and confirm 'market'

6 Finalise development environment
- Write user documentation
- Write code. NB Possibly in stages.
- Get code to work (Exercise)
- Check code works (Test)

8 Build finished package of deliverables
9 Consolidate development documentation and review

Steps 5 and 7 are a mixture of things. It doesn't address the complete software lifecycle. This is applicable to almost any construction task whether building a server or a user manual. It applies to the whole project in a general sense but is more geared to the development of individual elements.

There are two good reasons for having a template such as this:
- Coding (or writing or soldering or translating) is *part* of the job and it isn't at the beginning.
- Understanding the structure and semantics helps everyone in the team understand what others are working on at any moment. "When you've got your installation instructions written I can try them out on my mum's old Mac. Do you want me to exercise it or test it? If test then what's the procedure?"

Finally
Once you've been part of a buzzing team you'll soon become frustrated by the lumpen alternatives. When you have the vision and the confidence but your boss is a feeble fumbler it might be time to move on to other things - they will soon resent your command of the situation and intuitive team building behind their back.

Appendix D
Trierarchy

This is a possible management structure for command and control. It designed particularly

- for people who don't like committees or 'noisy' forums
- to give a definite management structure ...
- ... with the camaraderie of micro-teams.
- to be scaleable
- to minimise administration channels
- to support organic growth

It is not ideal for initially discovering who is who in the early days of getting a team together from a large pool of recruits, although as the structure forms it will help members 'find their niche'.

Teams of three

I have noticed teams of three work better than teams of two or four. There is no reason why all three members should be 'equal', in fact the ease with which one can brief two close partners, or can speak up 'against' with 'one third of the votes' means that the team can share while at the same time not jumping to reckless conclusions. For example the person who is naturally disposed to fine detail will complement two that aren't really interested in details without skewing the group's activities so they get bogged down. Working with only two colleagues means relationships are quickly and firmly built.

Linking together

How can we use these micro-teams together to form a larger organisation?

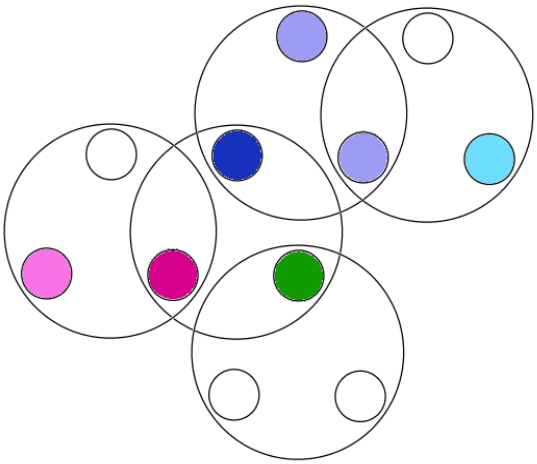

Figure 1

For sake of illustration let's suppose that three friends (Red, Blue and Green) decided to start a project. This makes a cozy team of three.[21] Now Blue recruits two helpers for 'her side of the business' which means Blue is now part of two groups of three. We might think of Blue as the 'senior' member of the all-blue group (but as we'll see in a moment this may not be the case) ie. the one that has all the project knowledge and who delegates and educates. In total Blue has five colleagues. In the 'blue team' success breeds success and another person is recruited. There are two ways of dealing with this:

1 As shown in the figure: Allow slim teams of two.
2 Allow fat teams of up to six.

Which to use probably depends on circumstances and preferences.

Looking at figure 1 without the explanation just given one might think that top-of-the-tree is the blue team. It certainly seems to be the powerhouse of this project. Which is the correct view? For command and control matters the commanders are at the top of the tree so the 'correct' view in this example is that the Red-Blue-Green team is the ultimate controlling and coordinating force. Where a trierarchy has evolved the more experienced members are going to be the ones nearer the centre and so they're in the best position to take decisions based on the history, group experience and culture of the company.

What is the individual member's view? Typically they belong to one micro-team as a helper and another as a leader. Ideally each team would have two other members giving a total of four immediate colleagues. A small enough number to get to know given the restrictions of remote working. Everyone can be on first name terms and have some idea of the personal qualities of their colleagues and other pressures on them. Obviously they will be well acquainted with what their team work schedule and objectives are.

There is no reason why somebody in one branch shouldn't be dealing directly with someone in another. For example suppose the top-level threesome split into Finance, Coding and Other then 'Coding/Quality/Exercise/ImageProcessing' might want to work closely with 'Other/Deliverables/HelpDocs/Tutorials'. However it seems reasonable to me for a micro-team to be informed when an individual makes an 'outside contact'. This reinforces team collective responsibility. This would also mean that work gets peer-reviewed before being 'released' and 'doing work on behalf of another team' gets approved by a team not just an individual.

[21] Tream?