true

Contents

Trifork Handbook

You work at Trifork, and this is your handbook. The handbook details how we work together, which processes we have established, and the tools we prefer to work with. But before we get too much into details, you should first understand the purpose of this book.

Just like the world around us, our business is constantly changing, and the way we work should reflect that. This handbook is a living document and should be updated often. You should always question its contents, but try to follow the guidelines when there is no reason not to do so.

We have made handbook Open Source, and you can freely distribute it to anyone you see fit. We did this because we believe that be best way to work together is through transparency—The more our clients, team-mates, and potential employees know what to expect, the better.

If you think parts of the handbook are outdated, please do not hesitate to send us a pull request on GitHub.

What the handbook *is* The handbook is an introduction for people (both technical and non-technical) to the concepts we work with in our day-to-day business. It is also an introduction to our internal tools, e.g. time registration, calendar, etc. It describes the most common meeting types, and also acts as a handy look-up reference e.g. for a checklist when setting up a new project.

What the handbook *is not* The handbook is *not* about specific technical topics, e.g. recommended frameworks in Java or C#. It will not tell you what Ruby libraries to use, or how to configure PostgreSQL. Any language– or framework-specific guides should be added to our GitHub account separately from this document.

How we work

What it really boils down to, is that we believe that to produce good software, we have to empower our employees and our clients.

We have a culture where it is usually better to ask forgiveness than permission—because it gets the job done. This of course does not mean you should throw caution to the wind, but rather think for yourself. You do after all work here because you are an intelligent individual, right?

At Trifork, we are all about transparency. That means we don't hide negative stuff, we "call a spade a spade" and don't beat around the bush.

This means our clients should know when an error occurs, the level of maintainability and technical debt in the codebase, workarounds and general suboptimal stuff. No piece of software is ever perfect, and this fact is better faced head on. Having a client or project manager who is well-informed and in the loop, enables them to make the right decision, and that is always the best way to go.

Consulting

We make most of our money building products for our clients. In that sense, we are not your run-of-the-mill consultants, since we do very little on-site, six-month, "Body Shopping" assignments. We do most our work at our office, allowing us to be efficient and work, with the collective knowledge of our colleagues at our immediate disposal.

No matter the assignment, the goal is always to produce something great. We want to help our clients make the right choices. We are at the forefront of technology, usually knowing what is just over the hill, while usually, our clients are not.

This can be a huge benefit for our clients, and is one of the main reasons that they pick us to help them push the envelope. We are there to help them produce great software.

Our Own Products

TODO

Empowering Developers

At Trifork, we strongly believe that developers should be empowered to create software on the platform of their choice, using the tools of their choice, with which they are their most productive. We call this a Low Governance Environment. Having free hands to pick and choose fosters creativity, and ensures that we remain productive and at the forefront of technology. On the other hand, it also has a tendency to generate a lot of fragmentation and makes it harder to collaborate across teams. Hence, this Handbook.

If you want to use another technology than those found here, you are welcome to do so, but don't base your entire application—or any crucial components—on it. Instead, consider using trial technologies as non-central components and spikes, at least until their benefits are proven.

Planning

Project Process

1. Find and Eliminate Risk

- Defining and testing a set of hypotheses

This is important, in that it allows us to follow what we saw and learnt, rather than simply following assumptions or common beliefs about the customers.

- Doing Research

Information-Gathering / Interviews / Sketches / Artefacts

Cognitive vs. Emotional Empathy

Emotional

This is often the most radical in its effect on executives and as a source of insight.

This, combined with the Client's Hypothesis leads to insight.

2. Design

The question is “How do we build something fantastic?”

Innovation Workshop - Bring in customers. - Stakeholders.

We are now confident that we are solving the right problem; we set a limited amount of time and start to develop a solution.

3. Prototyping

Paper or HTML?

`HTML`

- Dislike Colours
- Slower
- Scales
- Working at a distance

  Paper
- Abstract
- Focus on features
- Quick

Work Pace

We work at a sustainable pace. This means that we do our best to stick to 40 hours a week, 8 hours per day, and plan our sprints this way. From time to time, it may be necessary to work more, or you may want to stay longer because you’re “in the Zone”, which is why we have flexible working hours.

Service Level Agreement

TODO (flg): I think you had some stuff that could fit here.

Morning Meeting

TODO

Task Tracking

Currently, we use JIRA for our project– and task-tracking. There is no such thing as a task outside of JIRA (i.e. that mail the project manager sent that asks if you could "just quickly do X" doesn't, in fact, exist and is only a figment of your imagination).

Sharing Knowledge

At Trifork, we dedicate time to learn, improve and evolve. We all want to become better at what we do, and luckily we are surrounded with intelligent people who can teach us stuff. Now, if we don't learn from each other, we end up making the same mistakes over and over, not even knowing that there are better ways to do it.

Everyone's aim should aim to spend at least 10% of their time teaching others. This is an investment, but one we cannot afford not to make! We need to pull everyone up to our high level of quality and professionalism.

Teaching can mean many things, and there are many ways to help others become better. First off, just doing pair programming is a very productive and fun learning experience. You could also write a blog entry, do a Tech Talk, contribute to this Handbook, write a language guide or framework guide for the company, or prepare a GOTO; Night.

In this section, we describe some of the ways we can help each other improve.

Tech Lunches

We keep track of upcoming Tech Lunches on a Trello board. People can suggest topics and vote for topics that they would like to hear about.

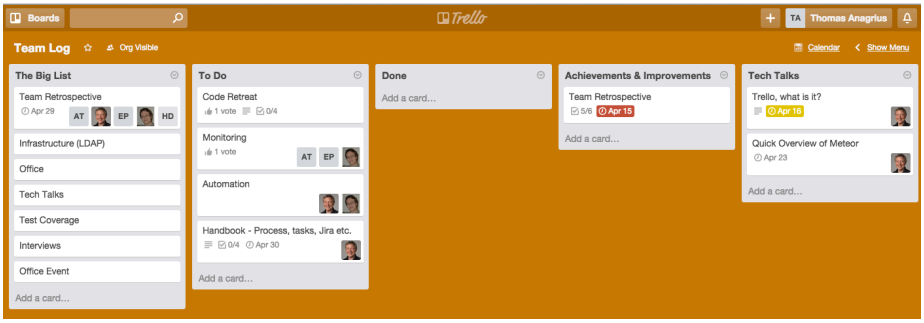

Figure 1: Trello Tech Lunches

TODO

Hacking Retreats

TODO

GOTO; Nights

TODO

Recruiting

Finding People

TODO

Interview Process

We keep track of our applicants in JIRA. When we receive an application, it is entered into JIRA so we can keep track of the progress.

Our Manager, Thomas, is responsible for the hiring process. He makes sure that we reply to everyone as soon as we can. We want to keep our standards high, and Thomas also leads most interviews and ensures that the applicants are up to the task.

Evaluating applications is a joint activity, and team members are asked to help take a look at the applicant's CV and any code samples supplied.

We either send them a kind rejection stating why they didn't make the cut, or invite them in for a meet-and-greet session. Either way, the will be moved to a new column in the JIRA board.

Meet-and-Greet Session The meet-and-greet is the first session, and is a chance for us and the applicants to asses each other. Will the applicant fit our culture? Does the applicant find the work we do interesting? And so on. We also discuss Trifork's way of working, and ask a few questions about the applicant's CV. This session will usually be done using Google Hangouts or Skype and last about 30 mins.

We also require that all applicants complete a personality test. The test is emailed to the applicant, is two pages long, and takes about 10 minutes to complete.

Technical Session The next step is to swing by the office and have a technical discussion with one or two people from our team. We have a prepared list of general questions about computer science, web development, iOS and Android, Enterprise Java, etc.

Trial at the Office The final step is for the applicant to spend a full day with the team. We will pay for any travel and hotel expenses that may apply. This will usually be a Friday and they will be pair programming with one of team members.

This way the applicant gets a real idea about what we work with and what the atmosphere is like at the office, and we get to see how the applicant deals with a real job situation.

Making an Offer Trifork's CEO, Jørn Larsen, has the final word about any new hire. We will send him the CV, grades and results of the personality test. If all parties agree that it is a good fit, the applicant will receive a contract for digital signing on HelloSign.

Once the contract is signed, we create a new employee account and our new colleague gets a new company email assigned. It will be comprised of their initials followed by `@trifork.com`.

Your First Day

One your first day, you are welcomed into the team. We assign a mentor to you who will help you get settled in. By now, you should have received a welcome email with your username and initial password—which you will have to change when you first log in.

You should spend your first day getting to know people, setting up your computer (if you have received one yet) and familiarizing yourself with our internal services like webmail, calendar, and time registration.

Your mentor is meant to help ease your first time at the company and help answer any questions you might have.

Your First Week

You should have been introduced to the first project that you are going to work on. You should also have made your first commit to this project—it doen't have have to be anything major. You should get a copy of the classic book *Extreme Programming* started reading it. It is a classic, and explains many of the methods that we value highly.

Management

Salary Review

Salary review is done on a annual basis, usually in August. We encourage our team members to know their market value so we can come to a fair arrangement that all parties can be happy with. It is in the interest of the company to pay people what they are worth. We don't want our employees to discover down the road that they have been under-paid.

Compensation

Compensation can be put together in many ways. You can suggest alternative solutions to just the usual monthly pay-cheque. Be inventive and come up with ideas that will suit your goals and life situation. People with kids tend to want more vacation, while others may be interested in training, or the company sending them to a conference that they are interested in.

Salary is usually paid out on the 25th of the month. Before Christmas, we pay out salaries a bit earlier.

Quarterly Review

We want to make sure you are happy and find your work interesting. Some things can only be said behind closed doors, between you and your manager.

Even though we have an "Open Door" policy and you should always come to the managing team and let them know if something is wrong, it is important that we also remember to talk on a regular basis. That way small issues don't become big problems.

Company Credit Card

If you have a lot of expenses, you can get a company credit card. We use Eurocard Gold for our employee cards. You will still have to enter the expenses in our Time Tracking System, but the money will not be transferred from your account for 60 days after your purchase. This allows our payroll team to reimburse you before the money is ever deducted. You should ask your manager for a Eurocard form if you wish to apply for a card. The card can also be used as a personal credit card in conjunction and has a few benefits, e.g. travel insurance.

Sales

Fixed-Price vs. Weekly

We prefer to avoid Fixed-Price contracts. They pit the client and us against each other right from the start of a project. You can end up arguing over "what it said in the initial contract" while the project has evolved. To be able to work in a flexible and agile manner, working on projects on a week-by-week model, allows both parties to focus on what they really want. It is impossible to know every facet of a project at the start, and there are always new ideas and realizations that happen along the way.

Contracts

Usually the kind of projects we take on are:

- Project Design: We help the client plan and evolve
- Nothing to v1: These projects usually
- Maintain an existing project in a transitional period
- Analysis Report, e.g. Code Quality Review, Agile Process Review
- Aid an existing dev team until they hire someone of their own

Morning Meeting

TODO

Developer Setup

Security

You should always enable full disk encryption on your machine, and protect it with a strong password. If you want to have a dual-booting machine, e.g. between Ubuntu and Mac OS X and only have a single physical disk, you may have issues with your Operating Systems' built-in encryptions not playing nice with each other. You can in this case look into alternative solutions like TrueCrypt.

We encrypt our disks to keep our data (and our customers' data) safe from prying eyes. Having your disk encrypted is the best way to ensure your data is not easily stolen.

Slack

For internal communications within the team, we use Slack, which is quicker and more efficient than email, and makes it easier to deal with file attachments, code snippets, notifications from 3rd-party APIs and so on.

Skype

We use Skype almost every day for communication, phone calls, screen and file sharing. You should set up a special company account. We use the naming convention `<initials>trifork` for our accounts — which makes it really easy to find others.

You can add Skype credit and put it into our system as an expense.

VPN

If you are not on the office network, and want access one of our servers such as code.trifork.com which are not in the DMZ, you will need to connect to our network with a VPN connection.

How you set up the connection will depend on your OS. You should set up an `IPSec` connection to:

```
Server: vpn.trifork.com
Username: <your initials>
Password: <should prompt every time you connect>
```

Our server requires that you put in a "shared secret", i.e. a group password. You can find the shared secret as well as several step-by-step guides for connecting by going to the Wall and searching for "VPN".

Email

We use an Exchange Server for our email. In your Welcome Email you will have received information on how to log into Confluence. There you can find information on how to set up your mail client. Just use the search function and search for "email".

When you send emails for work, you should always use your company email address. It is also a good idea to include a signature like the following:

> Firstname Lastname
> Trifork AB
> Phone: +46 000 0000

Figure 2: Trifork - Think Software

Email is probably your most important tool. It contains an enormous amount of data, e.g. project details, account info, attachments, etc. It is also one of your main resources for getting in contact with people. You will find yourself looking for contact information in other people's emails all the time, and you should return the courtesy and include yours.

Calendar

Just like with Email

Confluence (the Wall)

We use a Confluence server for storing internal information that cannot be publicly disclosed in the Handbook. You can find information about anything from WiFi credentials to VPN settings.

We also use Confluence as our internal Blog / News / Notice Boards.

Development

Project Checklist

- The project has up-to-date Documentation, preferably in Markdown. Is it up-to-date?
- The project has a job on a CI Server It should at least make sure the project can compile or a minimal sort of boot test.
- Be in a Git repository that has adequate security depending on the project, e.g. certificates and HTTP/S.
- The project has some sort of Task Tracking that is actively used.

Testing

We strongly encourage writing tests. Writing tests helps you and especially others to produce better and maintainable code. Every project should have automatic testing enabled. See continuous integration.

In an ideal world at least every central component, features of high business value or which have a high impact when failing are fully tested. In reality it seems to depend on the project pressure, the mood, experience and assertiveness of the current developers and project managers. But let us stay on the bright sight and let us tell some words about testing in general which hopefully change your point view :)

Testing ensures not only the correctness of your code - it facilitates the ongoing development and maintenance which is not necessarily done by the original developer (which might be YOU). Even if you assume that your code is error-free and will never be touched again especially in an agile development or project business you never know for sure. Testing is about proving that your code is testable! Nothing is as annoying and disappointing as to want to write tests only to find out that the code is not testable and writing tests would require a significant additional effort which is not arguable to project management. Who is responsible for that? We think that it's your responsibility to make sure that your code is testable! The following developer who will get in touch with your project (maybe in the near future) will thank you for an usable test foundation.

Writing good - clean, maintainable and extensible tests is as hard as any other part of creating quality software. You might even say writing good tests is harder because implementation details are changing - good tests last for much longer than their original target implementation. But knowing how to write 'good' tests especially which don't create an overhead when refactoring the target implementation and which are easy to understand and to extend is very hard and is a skill that is learned by study and practice and it takes time. To speed up the learning curve is doable in a very easy way: Just do it right from the beginning :) The major benefit of writing 'good' tests while you develop rather than after the fact is that it produces testable code.Testable Code tends to be better code because it is usually modular, has a clean and easy to use interface,and interacts only with few other parts of the code base…

Continuous Integration

TODO: CI Server

Continuous Delivery

All projects should be built on git push by a CI Use Jenkins over CircleCI over TeamCity. All projects should be buildable with single command You

are strongly encouraged to make a Make target called “ci-package” You are encouraged to tag all build versions in git. See semantic versioning.

Software Licenses

Software Licenses can be seem bit of a jungle to navigate at first. There are many different licenses, even many versions or variants of some licenses.

When doing client work, it is important that you know what kind licenses are approved by the client's legal department, if any. If a client does not have an existing policy on software licenses it is our responsibility to both educate them and make the right choices for them. This also holds for our own products and Open-Source libraries.

You can generally divide the most common licenses into two categories:

Proprietary License The publisher of the software retains ownership of the software, which is somehow licensed to the user. These are seldom used in our software, and can usually never be distributed as part of an open-source solution. They are usually paid copies, so that you will probably not include this type of component by accident.

Free Software License These licences require that any software, derivative of the software that incorporates a material under the license also be distributed under the same licence, in essence also making it Free Software.

For example, the GPL requires any derivative work to also be released according to the GPL.

This does not mean that you are obligated to distribute the code. E.g you may create company internal applications that use free software as long as you don't try to sell it.

Notable Examples: GPLv3, Copyleft

Open Source License TODO

Contributing to Open Source

We use Open Source software every day and on all our projects. We could not live without it. We therefore also encourage our developers to pay it forward and contribute to the open source community.

There are several ways to contribute: one is send patches to other people's projects. You can and should do this freely. Sometimes our projects yield useful

libraries and utilities that are worth sharing. You can release these on our GitHub account, but they have to be of high quality, have a good name, and especially good documentation—there is no point in sharing if no one knows how to use it!

If you have a piece of code that you think merits becoming open source, step one is to speak with your manager and get some time allocated to ensuring its quality. Then you should submit it to our GitHub account and link to it from our website.

Remember you are representing Trifork with your software. Copyright should remain Trifork's even though we are making it open source. We use the MIT License for our open source projects. It allows people to do what they want as long as they keep the Trifork attribution.

Git Repo Hosting

Trifork uses GitHub for both our Open Source projects and some of Closed Source projects. For most of our projects, we use our in-house code server at code.trifork.com. It is important that you make sure that the project's client is okay with hosting anything on GitHub before you create (even a private) repo on GitHub.

Logging

Use an image with logging like ELK. Use Rolling Logs

Use semantic logging patterns (LINK?) Avoid log and throw anti-pattern

Documentation

- README file containing:
- Brief project intro.
- Developer setup guide.
- Deployment guide.
- References to any further documentation.
- Architectural document

Top-tier components e.g. web services, databases, external systems, communication links.

- Technology decisions:

Choice of technologies to solve a problem must not be arbitrary.

But most often, it is. "Okay let's use Spring and Web Sockets… What was the project about again?" Choosing the right tool to solving a problem is crucial for the maintainability of a project.

Makefiles

Makefiles are a great was to document common tasks in your project. Make is ubiquitous on all *nix systems and it therefore a good way to install dependencies and set up a project.

When you take over a project from another developer, the `Makefile` is a key piece of documentation on how to get started and interact with the codebase.

Using an informal interface (i.e. a naming convention) for the `make` targets allows us to build up infrastructure and scripts around `Makefiles` that work across projects.

In many cases you can store `bash` or `python` scripts that contain the actual code to be executed from the make targets, but having them be executed from the `Makefile` serves as documentation for new developers. In many cases, you will even call external build systems like `grunt` or `mvn` from your `Makefile`. The makefile does not replace these tools, instead it ensures that we have a consistent way of interacting with the project, no matter if they are e.g. Java or Python.

The informal interface is described below.

Target Naming Convention Your project should normally contain the following targets. You may of course have many more.

`make setup`

The `setup` target bootstraps a project. It installs dependencies, sets up `git hooks`, creates databases, log files, etc. The idea is that whenever a new developer clones the project, they can run `make setup` and be ready to start developing.

It is of course not always possible to install all dependencies, and any additional setup steps must be documented in the `README.md` file.

`make run`

The `run` target compiles and starts a running instance of the project. This is not always feasible, e.g. for iOS projects it may not make sense, but in the cases where you are developing a web service or have some sort of `REPL` this is where you would do it. You will usually also make this the `all` (default) target of the `Makefile` so developers can just write `make` to get going.

`make test`

The `test` target is another important element. You can run your project's unit and/or integration tests.

`make ci`

The `ci` (as in Continuous Integration) target allows us to use a pre-configured job template on our CI Server. Here is an example of how the target will usually work:

1. clean
2. compile
3. test
4. create a deployable
5. create a git tag

Depending on your project you may not need or be able to do points 4 and 5.

You can check out our website's `Makefile` if you want an example.

`make deploy`

Again, depending on your project, having a `deploy` task may not make sense. In other cases it may be used to deploy to a staging or production server. Deployment should always be easy and painless. Having a `make` target for it ensures you doing it frequently, and "the next guy" actually maintaining it.

If you are doing continuous deployment, `deploy` could also be called from the `ci` target.

Web Development

TODO

Sending Email and SMS

TODO

Design & UX

TODO

Working with Designers

TODO

Working Together

Peer Review

Peer review is encouraged. Try to work in a peer review set-up as part of your team workflow. If you are working alone on a project, consider getting a review buddy and review each other's code across projects.

Consider mixing it up and do group refactoring sessions as well as a chance to learn from each other.

Review Checklist

- go through the Project Checklist from the Handbook and make sure the project conforms.
- Check the recommendations for the platform used in the project, and discuss and document if something diverges and is not already documented.
- Make sure that the project conforms to the platform's style guides. You should have a style guide for any language you work with. If you don't, make one and share it.

Pair Program

Pair programming is encouraged. It is great for knowledge sharing and catching bugs early. Find a balance with coding individually and in pairs.

Measuring

System Monitoring

Disk usage, memory etc. should in most cases be handled by operations. You should do monitoring using log aggregation. See logging. Teams are encouraged to have team dashboards, on a screen, with kibana or duckboard.

Log Aggregation

TODO

Feature Flags

TODO

Transparency

TODO

Time Tracking

We do our time reporting through the internal tool Tidsreg. For each project that we have, there should be a corresponding row were you input the amount of time (in hours) spent on that project per day.

Project categories Projects are grouped by `Customer -> Project Name -> Sub Category`. Working hours spent on tasks or general office duties not connected to a specific project should be filed under internal time ("Intern Tid"), e.g,

`Trifork AB -> Internal Time -> Non-billable`.

For internal time, please make sure you divide your hours into sub-tasks if you've spent a "significant" amount of time on different assignments. See the section "Custom Description" below for more info.

For client-facing projects it is very important that you report Billable time as billable. There should exist such a subcategory under each project name. For example:

`Trifork GmbH -> Wealth Analysis Tool -> Wealth Analysis Tool -> Analysis -> Billable`

failing to do so means that we as a company will not get compensated (by the customer) for the work you have done. Of course, equally important is that you not report non-billable time as billable.

See the next section for instructions on how to add/find the correct project to your Tidsreg.

Finding / Adding a Project to Tidsreg A new project can be added if it isn't listed in the Tidsreg table yet. On the top of the page there is a drop down-menu / search field that can be used to find the missing project (that is, as soon as the project manager has had time to create it). Use the "shuffle"-icon on the far right to toggle between the search field and drop down-menu functionality.

Typing e.g "Intern tid" or "Internal time" in the search field should give you a list of options to choose from.

Custom Description For all new project entries you can add a custom description through the link "edit voucher text" appearing immediately the project name.

This is useful especially when adding small "Internal Time"-tasks where you can more specifically define what you were doing, e.g "trifork.se website" or "GOTO night preparation" etc.

Long-term projects By default, projects are added on a per-week basis. This means that once a new week starts, you would have to go through the process of re-adding your projects to Tidsreg (as described in the previous section).

If instead you would like to have a project persist for a longer time, you can do so by clicking the little star icon that appears to the left of each project name. A filled star means that the project will "stick around" whereas an outlined star will make it dissappear by the end of the week.

Expenses

If you have expenses due to work, they will be covered by the company. You can use either your private debit card/credit card or your company credit to pay.

You must always enter your expenses in our Time Registration System. You will find expenses under the tab "Vouchers" at the top of the page.

The amount you enter should always be in your local currency (what your salary is paid out in). You should specify the amount that was listed in your bank statement to ensure that exchange rates are correct and that any conversion costs are included. Often, if you pay in a foreign currency the credit card company will add a fee.

Travelling (Booking)

Sometimes you may need to travel abroad either due to a company event, a project or a conference.

Normally, you will find the tickets and hotels yourself. This way you get a trip that suits you. We fly in coach, it is cheaper and we would much rather spend the money somewhere else, e.g. buying you a new phone. If you fly with SAS, we have a company membership code that will give you a discount. You can get the code from your manager.

For hotels, find something comfortable and reasonably priced and do yourself a favour and make sure breakfast is included.

You can either pay with your EuroCard or ask your manager to pay using the Corporate Credit Card. Paying with these cards gives you an additional travel insurance.

Travelling (Expenses)

While travelling, your expenses are covered by the company. Anything not related the assignment, say theatre tickets, sight-seeing trips or visits to the pub, you pay for yourself. But anything else is covered by the company.

Agile Processes

Every project team must decide how they work most efficiently. All activity should be based on the Agile principles of Scrum, Kanban, Extreme programming or similar. *But don't kill agility with agile processes.* In the end, agile development boils down to:

- do frequent deliveries (weekly/bi-weekly at least)
- be able to adapt to change (it will come)
- hold retrospectives to continuously improve the way you work together.
- get frequent feedback from your product owner and/or client

All these components are important to ensure a good project.

Team Retrospective

Every other week, the different development teams have a joined meeting. In Scrum terminology it is called "The Scrum of Scrums", but we think of it more as an Office Retrospective. The purpose of the meeting is to achieve common goals that may be too big for a single team. We also use the time to share experiences, e.g. what works and what doesn't, and generally try to improve the way we work together.

1. The meeting starts with any general messages from management. A facilitator is picked who will control the meeting, making sure we stay on track.
2. Each participant then goes up to the whiteboard, and on a time-line plots how their mood has been over past two weeks. Any significant events, e.g. a meet-up event or releases, are noted. This is a great way of gathering information for the subsequent discussions. No one should use more than a minute for this.

3. The team then needs a few topics to discuss. Some topics might have arisen from the time-line exercise, others may just be suggested. Any topic from previous meetings that are still relevant are put as candidates again. Using [Dot Voting] the team then agrees on 2 topics to discuss for the remainder of the meeting. All topics are written down in the Team Log for future reference.

4. The two topics are then treated using one of the many methods for processing a topic. There are many good resources online for creative was of processing a topic as well as many books, such as *Agile Retrospective*. One method that we have found especially good in the past is The 7 Hats Method.

5. The meeting should take no more than an hour in total and should result in a set of S.M.A.R.T. Goals that will make sure that our good intentions actually get realised.

Team Log

TODO

Legacy Code

Step by Step Guide If you inherit code, your first task is to really understand it. What does it do, what is architecture like, what are the interfaces, etc.

1. Have as many teaching/knowledge transfer sessions with the previous owner as possible.
2. Make your own judgement about the "quality".
3. Make sure you have a test safety net, before doing too much refactoring. LINK.
4. Don't fix what ain't broken.

The last point is probably the most tricky. Try no to be too hasty in passing judgement or applying your own aesthetics to a code base. Projects always have undocumented history, and if you change something that is seemingly 'stupid' or irrelevant, it could end up breaking the code or making the client upset.

In the end, it is also about protecting yourself; if you break something that isn't broken, you spend the time 'fixing' it and then re-fixing it, and life is just too short for that. Focus on adding value to the customer.

Deployment

- Use our pipelined images LINK
- Docker and Flocker
- Bash Scripts over Ansible over ChefSolo
- Ubuntu LTS over CentOS
- Use upstart over systemd over sysinit because we prefer Ubuntu.
- Use continuous deployment
- Use semantic versioning of you software