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 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. The handbook is a living document and should be updated often. You should always question its contains but try to follow the guide lines when there is not reason not to do so.

We have made handbook Open-Source and you can freely distributed to your friends and our clients. The reason we did this was that 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If you think parts of the handbook is outdated please do not hesitate to send us a pull-request on GitHub.

What the handbook IS The handbook is an introduction to people (both technical and non-technical) to concepts we work with in our day-to-day business. It also an introduction to our internal tools, e.g. for time registration and calendar. We describe the most common meeting types and it also meant act 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 that permission — because it gets the job done. That 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 client's should know when an error occurs, the level maintainability and technical debt in the codebase, workaround 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 that is well-informed and in the loop, enables them to make the right decision and 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 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 on the forefront of technology, usually knowing what is just over the hill, while our clients are not.

This can be a huge benefit for our client's, 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

In Trifork we strongly believe that developers should be empowered to create software in on their platform of their choice, using the 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 an 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 hard to collaborate across teams. Therefore this Handbook.

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

Planning

Project Process

1. Find and Eliminate Risk

- Defining and testing a set hypotheses.

This was important in that it allowed 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 / Artifacts 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 lead 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 Colors - Slower - Scales - WOrking at a distance

Paper - Abstract - Focus on features - Quick

Work Pace

We work at a sustainable pace. This means that we

Service Level Agreement

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

Morning Meeting

TODO

Task Tracking

TODO: JIRA, Trello, etc.

Knowledge Sharing

Tech Lunches

TODO

Hacking Retreats

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 applications 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 a interesting, and so on. We also discuss Trifork's way of working and ask a few questions about the applicant's resume. 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. It is two pages long and takes about 10 mins. 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, as well as one for Web Development, iOS and Android and one for 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 you 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 Resume, Grades and Results of the Personality Test. If all parties agree that it is a good fit. You will receive a contract for digital signing on HelloSign.

Once the contract is signed we create a new employee account and you get a new company email assigned. It will be comprised of your initials followed by `@trifork.com`.

Your First Day

One your first day you are welcomed into the team. We assign a mentor to you how 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" and gotten 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 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 paycheck. 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 not making Fixed-Price contracts. It puts 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 to focus on what they really want. It is impossible to know every facet of a project and there are always new idea 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 their 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 System's build-in encryption not playing nice with others. You can in this case look into alternative solutions like TrueCrypt.

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

Slack

TODO

Skype

TODO

VPN

If you are

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 setup 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 1: Trifork - Think Software

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

Calendar

Just like with Email

Confluence

We use a Confluence server for storing internal information that cannot be public 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 is 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

Tests are first class citizens We strongly encourage writing tests. Writing tests helps you produce better code. Every project should have automatic tests. See continuous integration.

Test when it is appropriate, and test only what needs testing. If you are new to testing you should adopt Test Driven Development or a variation there of e.g. BDD.

Writing good tests is as hard as any other part of creating quality software. Having lots of tests does not mean your software is great. Testing the your software can help make your software stable. Knowing what to test and what may not need testing is a skill that is learned by study and practice and it takes time. Here are a few good resources with general concepts:

- http://sturgill.github.io/2013/04/15/tests-are-overhyped/
- https://news.ycombinator.com/item?id=5554600

The major benefit of writing 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 (low coupling).

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.

GitHub

We share our Open-Source project's on GitHub. We are strong believers in paying it forward to the Open-Source community.

Anytime your make a useful component that others may find useful as well, go ahead and share it on GitHub. Remember that you are representing our company and should keep the standard high, documentation good, and code clean and well tested.

You can get your existing GitHub account associated with Trifork's GitHub organization. That allows you to create repositories under Trifork's organization page. We want to keep the list of repositories clean and up-to-date, so you should talk to someone and make sure that your stuff is ready before you publish it to the masses.

You are free to create *closed repo's* as needed, but we tend to keep most repositories on our in-house git servers.

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.

Generally you can divide the most common licenses into two categories:

Proprietary 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 you will probably not include this type of component by accident.

Free Software These licences are can 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 the GPL license 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 TODO

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. WebServices, databases, external systems, communication links.

- Tech decision:

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

But most often 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 start and interact with the codebase.

Using an informal interface (i.e. a naming convention) for the `make` targets it 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 `Maven` from your `Makefile`. The

makefile does not replace these tools, instead it ensures that we have a consistent way of interacting with 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 bootstrap a project. It installs dependencies, sets up `git hooks`, creates databases, log files, etc. The idea is that when ever a new developer clones the project, he can run `make setup` and be ready to start developing.

This 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 project. This is not always doable 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. Usually you will 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 projects 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 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 that you doing frequently and "the next guy" actually maintain 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 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 platforms style guides. You should a styleguide 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

TODO

Hours TODO

Traveling (Booking)

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

Normally you will find the tickets and hotels yourself. This way out get a trip that suits you. We fly in coach, it is cheaper and we would much rather spend the money someway else, e.i. 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 you 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 you Eurocard or ask you manager to pay using the Corporate Credit Card. Paying with these cards gives you an additional travel insurance.

Expenses

You can use your company credit card to delay the expenses being withdrawn from your account until the company has reimbursed you.

While traveling you expenses are covered by the company. Anything not related the assignment, say Theatre Tickets, Sightseeing Trips or visits to the pub, you pay for yourself. But anything else is covered by the company.

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.

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. Then each participant goes up to the whiteboard and on a time-line plots 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, like "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.

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 must tricky. Try no to be too hasty in passing judgement or applying your own esthetics to a code base. Projects always have undocumented history and if you change something that is seemingly ‘stupid’ or irrelevant 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