# Welcome to the IP Applications team!

This guide is an introduction to the work we do, and how we have done it so far. We want to give you an idea what to expect, maybe reduce some nerves, and to give you some terms to google if you feel like preparing. You won’t be tested on this stuff or anything like that. Once you arrive in the team we will be running plenty of training sessions to help get you up to speed, so don’t worry!

At the moment we are two teams. One team generally gets work from BT’s internal network, which is used by BT to run itself. The other gets work from the external network, which is used by the majority of the UK to access the internet every day.

Both teams place a big emphasis on continuous learning. We run events every week, such as the ‘software designery’, where we discuss programming techniques, and ‘brown bags’, where one of us gives a talk on a subject that interests us. Sometimes that’s a topic relevant to our current work, and sometimes it’s not. We also hold Innovation Days: three days roughly every six weeks where we work on something interesting that we feel will benefit the team.

We are all learning, so if you have questions, if you don’t understand something or you don’t think something works, do speak up and ask!

# Who will use your software?

At the moment, we are in high demand by the rest of BT, and our customers have lots of different ideas for software we could write. Here is an utterly non-exhaustive list of a few applications to get you thinking:

1. Network operations engineers
   1. These guys monitor the network and fix any problems as quickly as possible.
   2. They generally need programs that detect changes or faults on the network.
2. Network designers
   1. They design extra capacity and other upgrades for the network.
   2. They also forecast network usage to prepare for the future.
3. Managers
   1. They need programs that monitor the network performance, e.g:
      1. Failure rates
      2. Traffic
      3. Data loss
   2. To monitor team performance.

# What kind of software will you be building?

We write the programs other people in BT need to do their job, so our choice of tools is flexible: we will use whatever helps us get the job done. As a result, if you think there is a tool out there we should use, let us know!

To give you an idea what to expect, in the past we have written web applications that are used by BT internally. These applications run on our servers, and are accessed by our customers (other BT employees) through their browser.

Our applications are usually made up of the following parts (the tools we use are in brackets):

Our data is usually stored in a database (e.g MySQL and Postgres)

A ruby server handles browser requests, often referring to the database (e.g Sinatra, Rails)

We use Javascript + HTML to build a UI in the browser (e.g jQuery, Bootstrap)

Our server’s apps are organised using the Model-View-Controller pattern. You can get a good idea what this means by following the Getting Started in Rails tutorial:

<http://guides.rubyonrails.org/getting_started.html>

# What tools do we use?

### http://www.automatedtestinginstitute.com/home/images/Logos/tools/cucumber2.pngCucumber

We use a Ruby framework called Cucumber for our integration tests and acceptance tests. These are tests that check the system works as a whole, and are derived from the customer’s requirements. If these tests pass, that means the system has the features the customer needs.

Cucumber tests follow this pattern:

Given (some setup, maybe give the database some dummy values)

When (specify an action to be performed)

Then (a check that something has happened)

In our projects, the cucumber tests are in the features/ folder. The features themselves are written in English, and each step is defined in a Ruby file under features/step\_definitions. Settings for cucumber are kept in an env.rb file, in features/support.

<http://cukes.info/>

### http://jlynch.co/assets/skills-rspec-dd0946616833a3b4bf7f9aff6372f378.pngRspec: ruby specs

Rspec is used to write unit tests: a specification of the behaviour of each of the objects that make up our systems. Test setup for rspec is in spec/spec\_helper.rb.

<http://rspec.info/>

### https://senchamarket-images-production.s3.amazonaws.com/uploads/screenshot/file/251/big_jasmine_logo.pngJasmine: javascript specs

While the vast majority of our code is written in Ruby, we use Javascript when we need code to run in the browser. When we write any non-trivial Javascript, we use Jasmine to specify that code’s behaviour. Jasmine is very similar to rspec, but it is implemented in Javascript.

<http://jasmine.github.io/>



### Ubuntu VM

Most of us develop code in an Ubuntu virtual machine. Some of us have Macs, but all development is done in some variant of Linux. The virtual machine is run in VirtualBox. We’ll have a VM with a basic setup ready for you when you get here.

### http://www.elfnet.org/wp-content/uploads/2011/10/Vim_logo.pngVim

We edit our code in Vim. We’ll help you out with Vim when you get here, but there are some hotkeys at the back of this guide to get you started. The greatest strength of Vim is the way you can mix the commands you know into new combinations. It is also extremely customisable. We have a team vim config that contains helpful shortcuts and plugins. You’re very welcome to use this when you get here, or of course you can roll your own!

This game may help teach you the basics of vim:

<http://vim-adventures.com/>

If you have vim on your own machine, you can type ‘vimtutor’ from the command line to start a tutorial that will get you up to speed.

### http://cscrunch.com/sites/default/files/imce/u2/Git-Icon-1788C.pngGit: version control

Whenever we make a change or addition to our code, we record the change in a git commit. These commits form a chain of changes to the code. We can use this as a history, reverting changes that went wrong, and recording what we have done in the past.

Git also allows us to create branches. New features go on a branch, and are merged back into the master branch once they are completed and we are confident the feature works.

CodeSchool have a good introduction to git:

<https://try.github.io/levels/1/challenges/1>

# How do we work?

This is roughly how we get from a customer’s idea to a finished application. This workflow is open for discussion: we have regular ‘retrospectives’ where we talk about what has worked in the past and what we could change.

1. Discussions with the customer(s)
   1. We involve the people we are making the software for as often as possible
   2. We find out what they need the most, and implement the smallest possible thing that works.
2. Pairing
   1. We split the customer requirements into sections called stories.
   2. We work on each story in pairs – one person types, the other acts as a co-pilot.
   3. The pairs change every day – everyone works with everyone else at some point
   4. We write the code for each story on a separate git branch, and submit a merge request once the story is done.
   5. The merge requests allow us to review other each other’s code so we learn from each other.
3. Iterate over 1 and 2 until the customer has the software they need.

We have two servers, which both run a different version of the software we are writing:

1. Edge Server
   1. Has our latest output that passes all our tests.
   2. This is shown to our customers to demo new features, but is not relied on by them.
2. Production Server
   1. The version relied on by our customers
   2. Once we have tested an edge version and are happy with it we push directly to our production server and let out customers know.

# Vim Cheatsheet

Vim starts in command mode, which makes sense once you know the commands. If you have vim to hand, try out some of these commands:

i - enter insert mode. In insert mode, the letter keys enter text, rather than triggering commands.

esc - return to command mode. Your keys now trigger commands.

o - start a new line, and enter insert mode.

cw - change word. Delete characters from the cursor to the word’s end, and enter insert mode.

dw - delete word. Delete characters from the cursor to the word’s end. When you delete something in vim, it is also copied to the internal clipboard.

. - repeat the previous action (extremely useful)

u - undo

r - redo

v - enter visual mode. Start selecting text from the cursor location.

y -yank. Copies the current selection to vim’s internal clipboard. ‘y’ can be combined with ‘w’ to yank a word.

p - paste. Paste the current selection.

dd - delete the current line.

yy - yank the current line.

w - move the cursor to the beginning of the next word.

b - move the cursor to the start of the last word.

shift + [ or shift + ] - move the cursor up or down one paragraph.

ctf - (like change word) change to the next occurrence of the ‘f’ character. ‘f’ can be by any other character you want to change to, e.g ct” changes to the next quote. ‘t’ will also work with ‘d’ and ‘y’, e.g dt” or yt”.