Development Methodology

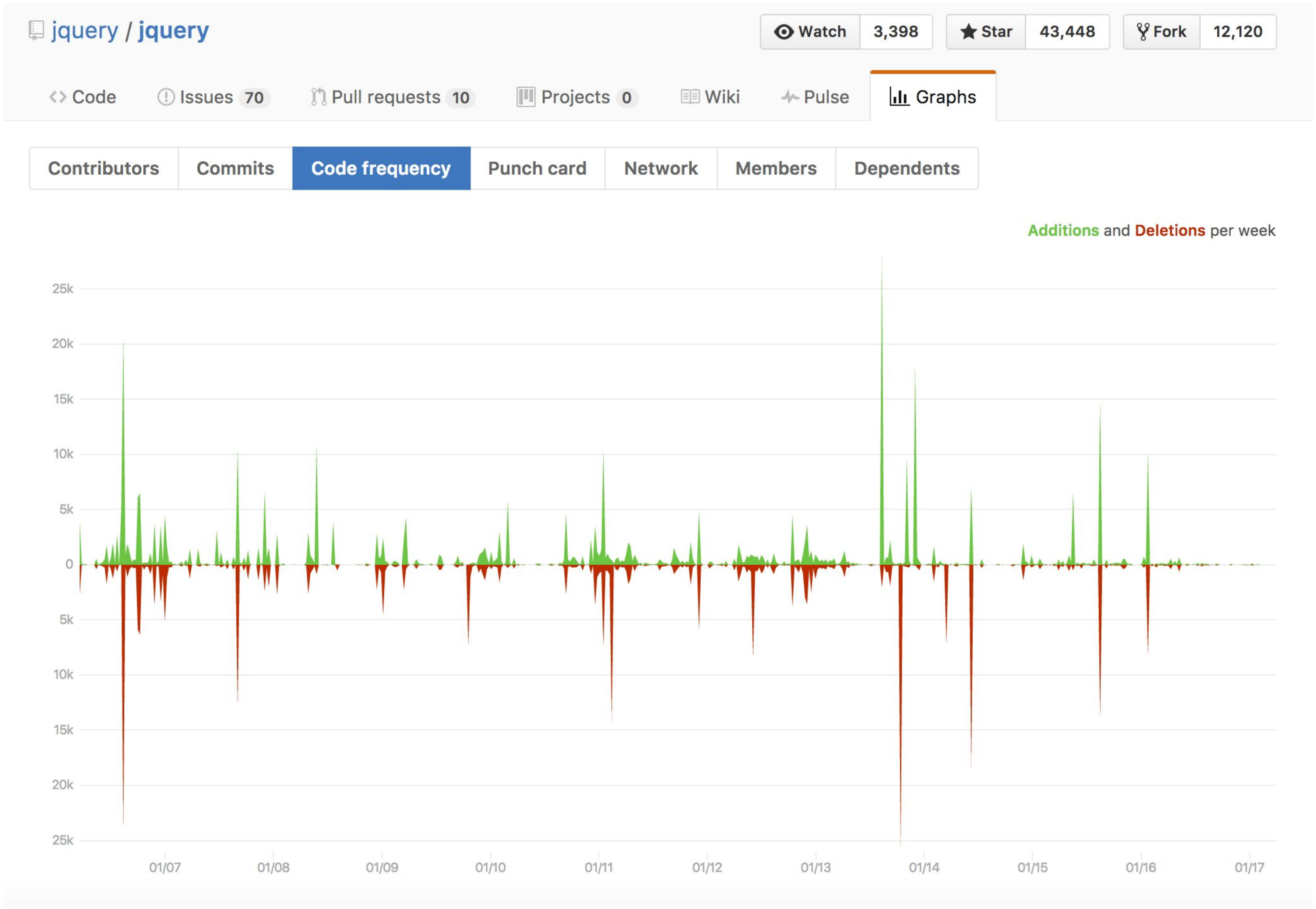
Aleksander Fabijan

Motivation

To be able to be **fast and reliable** (not to break things down while developing), we need to be **clever** in how we develop software.

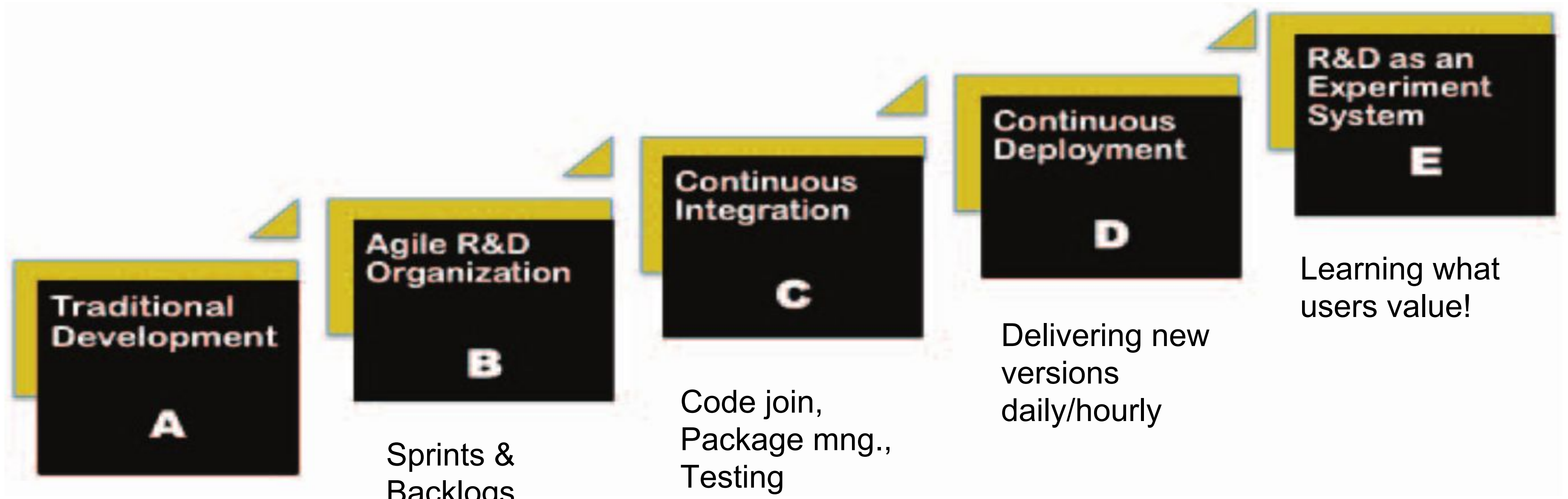
It is very **frustrating** when working code breaks and stops working.

Also, it is very **expensive** (in time) to share our code with colleagues, debug it, etc.



Jquery code changes over time.

Stairway to Heaven



Requirements first

Sprints & Backlogs

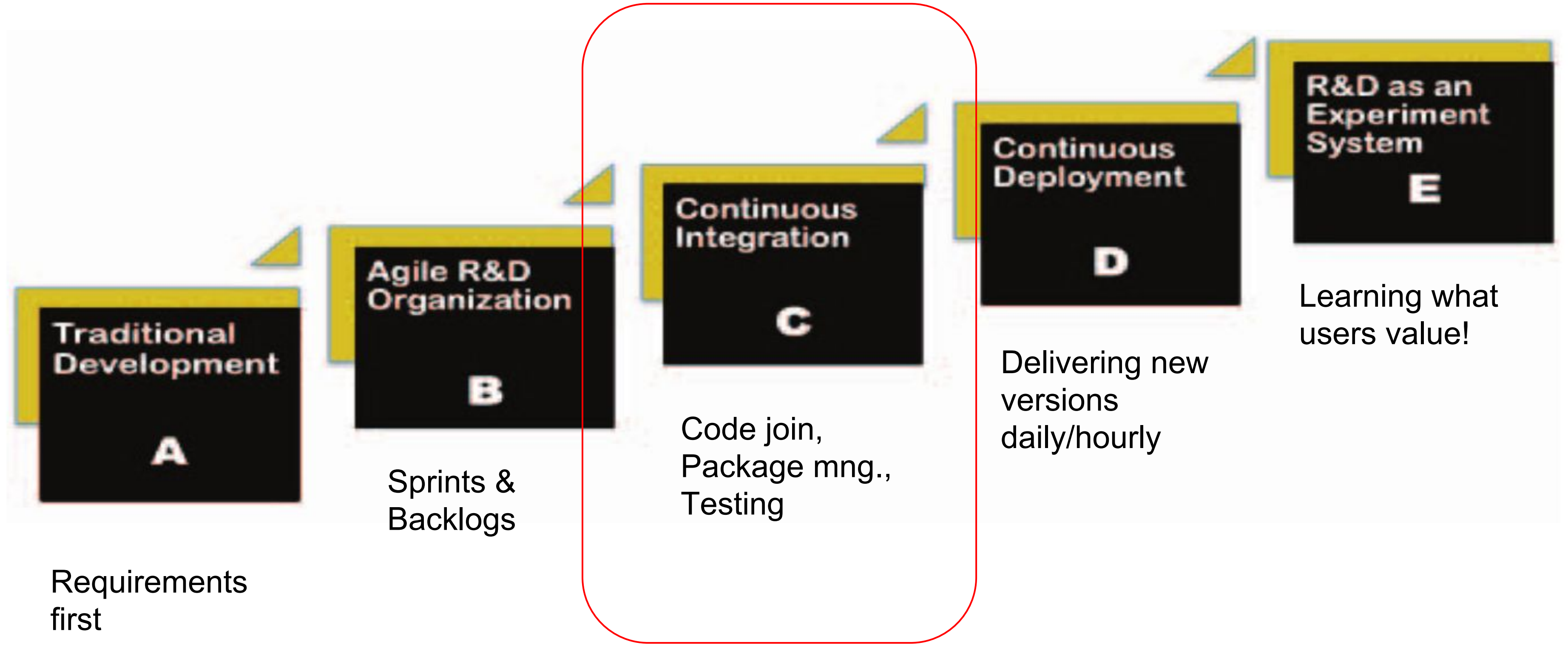
Code join, Package mng., Testing

Delivering new versions daily/hourly

Learning what users value!

Olsson, Helena Holmström, Hiva Alahyari, and Jan Bosch. "Climbing the" Stairway to Heaven"--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software." *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012.

Stairway to Heaven



Requirements first

Sprints & Backlogs

Code join, Package mng., Testing

Delivering new versions daily/hourly

Learning what users value!

Olsson, Helena Holmström, Hiva Alahyari, and Jan Bosch. "Climbing the" Stairway to Heaven"--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software." *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012.

Challenges with developing in Industry

1) **Changes to dependencies:**
   a) How do we speed up the download of the necessary dependencies?
   b) How do we make sure that our colleagues use the "right" version of the libraries?

2) **Changes to code:**
   a) How do we make sure that changed code works in the same way as before?

Answer to these Challenges

1) Package Management!

2) Automatic Testing!

1.

Package Management

The idea behing PM

Package managers **simplify** installing and updating project dependencies.

Browsing all the library websites, downloading and unpacking the archives, copying files into the projects — all of this is replaced with a few commands in the terminal.

Popular package Managers

1) NPM
2) Bower

With Bower each package is installed once, with NPM packages that need dependencies reinstall packages.

Installation of NPM and Bower

NPM:

- `https://nodejs.org/en/`

Bower:

- `npm install -g bower`

NPM

1) We initiate a new project in the terminal:

```
$ npm init
```

This creates a new project and stores all the relevant metadata describing the project in a file "package.json"

```
{
  "name": "tempdev",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Adding dependencies with NPM

```
npm install jquery --save
```

This will add it to the package.json

The library that we wish to add

Bower

The same things can be achieved with bower.

```
bower init
```

```
{
  "name": "tempdev",
  "description": "",
  "main": "index.js",
  "license": "ISC",
  "homepage": "",
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ]
}
```

Adding dependencies with Bower

```
bower install -- save jquery
```

This will add it to the bower.json

The library that we wish to add

Installing at a later time

```
bower install
```

That easy :)

2.

Testing

TDD (Test-Driven Development)

Instead of writing the program first, developers write the tests, see them fail, and write the program until the tests succeed.

Mocha test library for JS

Feature-rich JavaScript test framework running on Node.js and in the browser.

https://mochajs.org/

Installing mocha & should

Yes, you guessed it:

```
npm install mocha should —-save-dev
```

Writing our first test

```
// A simple javascript function
// that we will test later

exports.cleans = function(word)
{
    return word.toLowerCase();
}
```

An example test

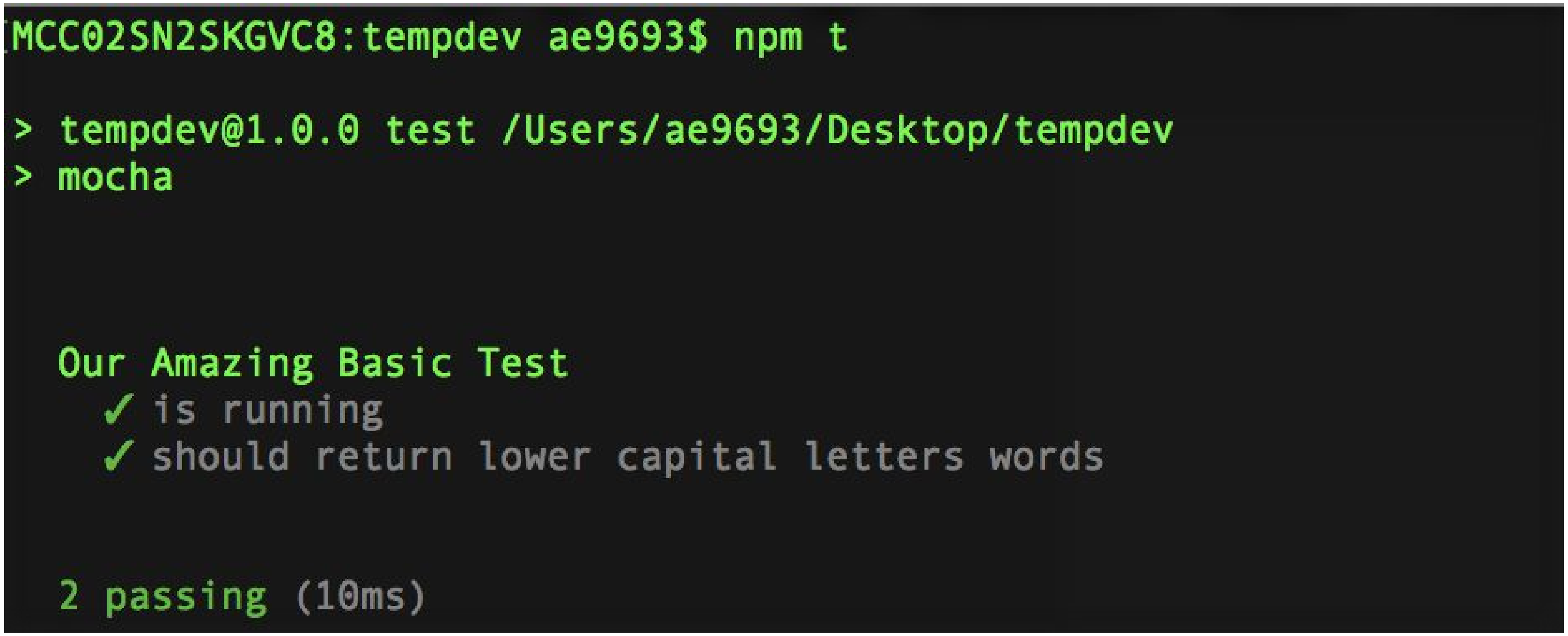
```
var should = require('should');
var mainF = require('../index');

describe('Our Amazing Basic Test', function() {
    it('is running', function() {
        true.should.equal(true);
    });

    it('should return lower capital letters words', function() {
        var inputWord = "TestWORD";
        var outputWord = "testword"
        mainF.cleans(inputWord).should.equal(outputWord);
    });
});
```

Running the test

mocha test

OR

Change package.json test to: mocha
and run: npm test



```
MCC02SN2SKGVC8:tempdev ae9693$ npm t

> tempdev@1.0.0 test /Users/ae9693/Desktop/tempdev
> mocha



  Our Amazing Basic Test
    ✓ is running
    ✓ should return lower capital letters words


  2 passing (10ms)
```

Tests have been ran!

Testing examples

```
var beverages = { tea: ['chai', 'matcha', 'oolong'] };
beverages.should.have.property('tea').with.length(3);
```

More testing Examples

```
// simple referencing
var obj = { foo: 'bar' };
expect(obj).to.have.property('foo');
expect(obj).to.have.property('foo', 'bar');
```

```
// deep referencing
var deepObj = {
    green: { tea: 'matcha' }
  , teas: [ 'chai', 'matcha', { tea: 'konacha' } ]
};

expect(deepObj).to.have.deep.property('green.tea', 'matcha');
expect(deepObj).to.have.deep.property('teas[1]', 'matcha');
expect(deepObj).to.have.deep.property('teas[2].tea', 'konacha');
```

3.
Experimentation

Learning what customers care about!

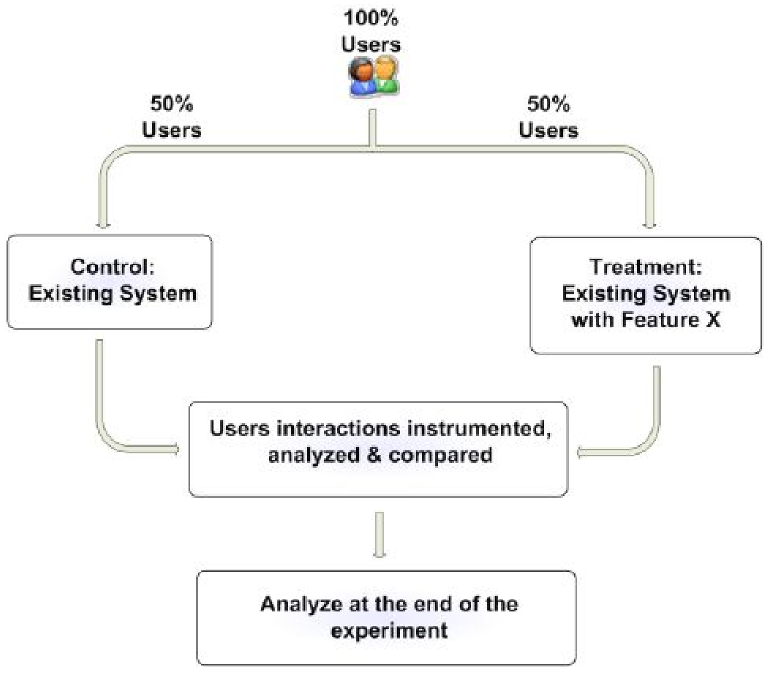
There are a lot of ideas on what our customers want.

HiPPOs have a looooot of ideas :).

HiPPO stands for "Highest Paid Person's Opinion"

A/B (Variant) Testing

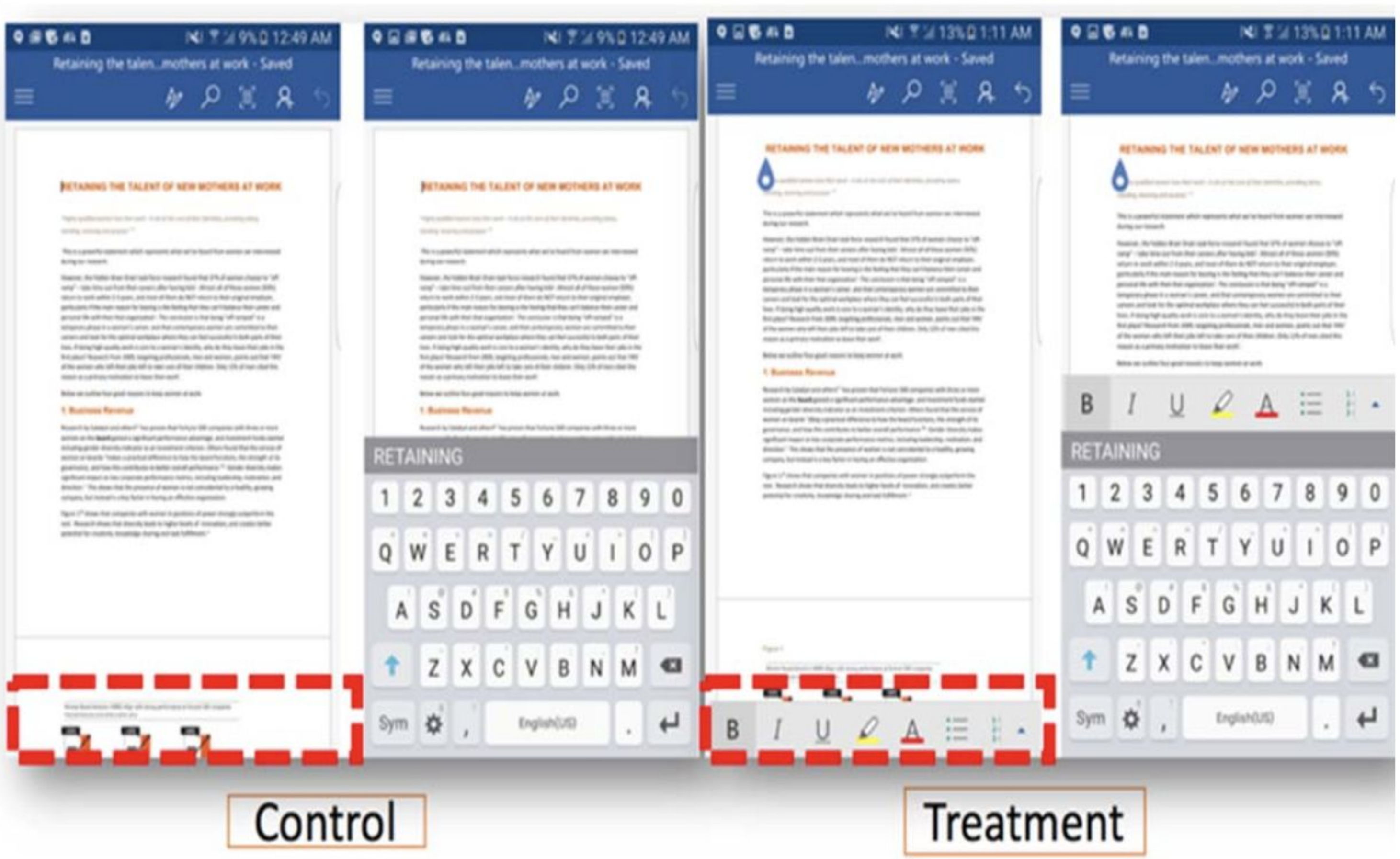
Using data to determine the “better” version of our feature/product.



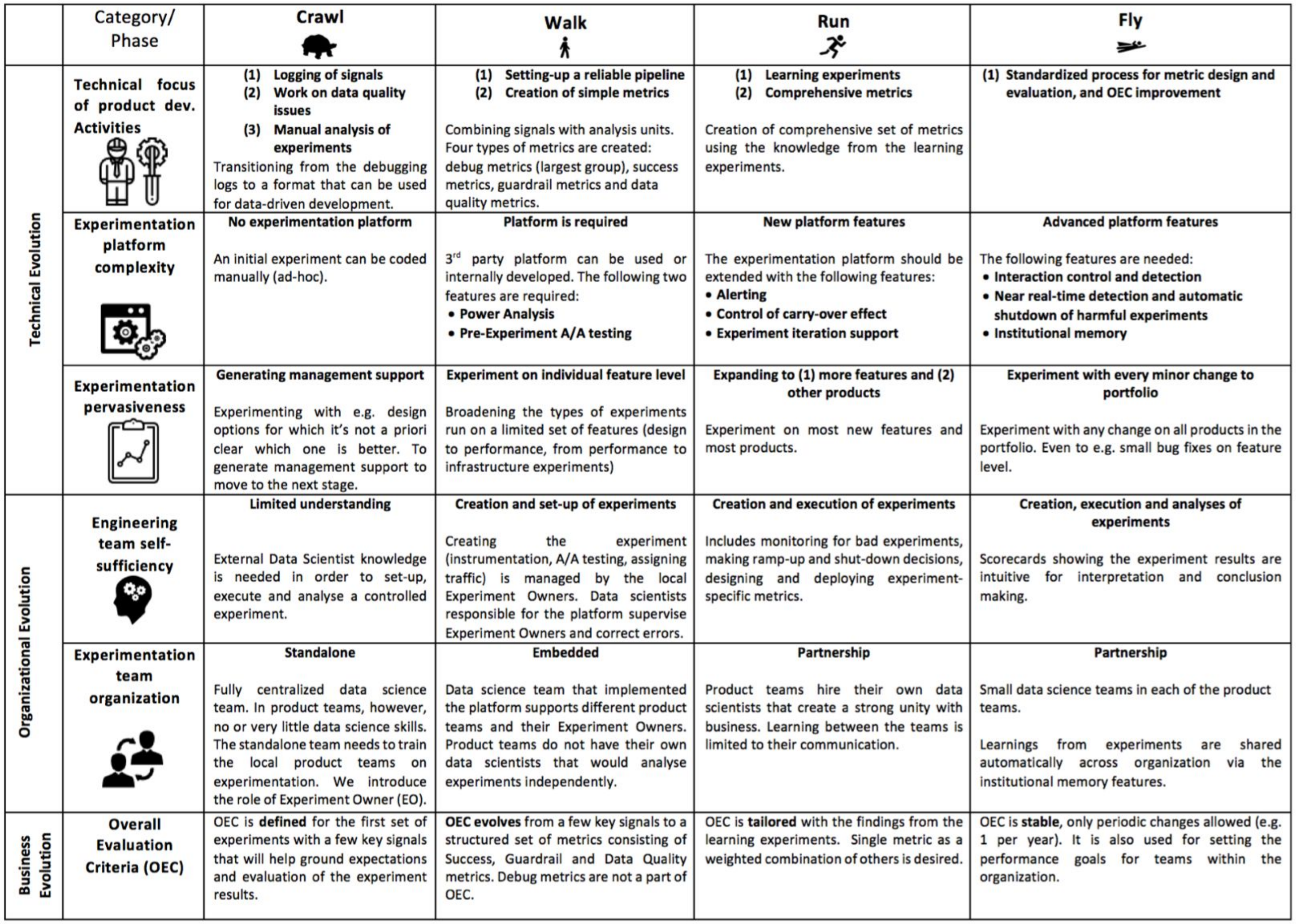
100% Users
50% Users
50% Users
Control: Existing System
Treatment: Existing System with Feature X
Users interactions instrumented, analyzed & compared
Analyze at the end of the experiment

Source: Ronny Kohavi, Microsoft

Example A/B test


Retaining the talen...mothers at work - Saved
RETAINING THE TALENT OF NEW MOTHERS AT WORK
RETAINING
Control
Treatment

The evolution of Cnt. Exp.



	Category/ Phase	Crawl	Walk	Run	Fly
Technical Evolution	Technical focus of product dev. Activities	**(1) Logging of signals (2) Work on data quality issues (3) Manual analysis of experiments** Transitioning from the debugging logs to a format that can be used for data-driven development.	**(1) Setting-up a reliable pipeline (2) Creation of simple metrics** Combining signals with analysis units. Four types of metrics are created: debug metrics (largest group), success metrics, guardrail metrics and data quality metrics.	**(1) Learning experiments (2) Comprehensive metrics** Creation of comprehensive set of metrics using the knowledge from the learning experiments.	**(1) Standardized process for metric design and evaluation, and OEC improvement**
Technical Evolution	Experimentation platform complexity	**No experimentation platform** An initial experiment can be coded manually (ad-hoc).	**Platform is required** 3rd party platform can be used or internally developed. The following two features are required: • **Power Analysis** • **Pre-Experiment A/A testing**	**New platform features** The experimentation platform should be extended with the following features: • **Alerting** • **Control of carry-over effect** • **Experiment iteration support**	**Advanced platform features** The following features are needed: • **Interaction control and detection** • **Near real-time detection and automatic shutdown of harmful experiments** • **Institutional memory**
Technical Evolution	Experimentation pervasiveness	**Generating management support** Experimenting with e.g. design options for which it's not a priori clear which one is better. To generate management support to move to the next stage.	**Experiment on individual feature level** Broadening the types of experiments run on a limited set of features (design to performance, from performance to infrastructure experiments)	**Expanding to (1) more features and (2) other products** Experiment on most new features and most products.	**Experiment with every minor change to portfolio** Experiment with any change on all products in the portfolio. Even to e.g. small bug fixes on feature level.
Organizational Evolution	Engineering team self-sufficiency	**Limited understanding** External Data Scientist knowledge is needed in order to set-up, execute and analyse a controlled experiment.	**Creation and set-up of experiments** Creating the experiment (instrumentation, A/A testing, assigning traffic) is managed by the local Experiment Owners. Data scientists responsible for the platform supervise Experiment Owners and correct errors.	**Creation and execution of experiments** Includes monitoring for bad experiments, making ramp-up and shut-down decisions, designing and deploying experiment-specific metrics.	**Creation, execution and analyses of experiments** Scorecards showing the experiment results are intuitive for interpretation and conclusion making.
Organizational Evolution	Experimentation team organization	**Standalone** Fully centralized data science team. In product teams, however, no or very little data science skills. The standalone team needs to train the local product teams on experimentation. We introduce the role of Experiment Owner (EO).	**Embedded** Data science team that implemented the platform supports different product teams and their Experiment Owners. Product teams do not have their own data scientists that would analyse experiments independently.	**Partnership** Product teams hire their own data scientists that create a strong unity with business. Learning between the teams is limited to their communication.	**Partnership** Small data science teams in each of the product teams. Learnings from experiments are shared automatically across organization via the institutional memory features.
Business Evolution	Overall Evaluation Criteria (OEC)	OEC is **defined** for the first set of experiments with a few key signals that will help ground expectations and evaluation of the experiment results.	**OEC evolves** from a few key signals to a structured set of metrics consisting of Success, Guardrail and Data Quality metrics. Debug metrics are not a part of OEC.	OEC is **tailored** with the findings from the learning experiments. Single metric as a weighted combination of others is desired.	OEC is **stable**, only periodic changes allowed (e.g. 1 per year). It is also used for setting the performance goals for teams within the organization.

Fabijan, Dmitri, Holmström Olsson, Bosh: http://goo.gl/k88d0b

4.

Resources

Summary

Use package managers for (1) fast downloading of dependencies and (2) standardising versions.

Write tests so you or your colleagues find bugs quickly.

Useful resources

https://medium.com/@dabit3/introduction-to-using-npm-as-a-build-tool-b41076f488b0#.jo91irkg1

http://chaijs.com/api/bdd/

https://codeforgeek.com/2016/04/continuous-integration-deployment-jenkins-node-js/

Paper on the evolution of Experimentation: http://goo.gl/k88d0b

Thanks!

You can find me at:

aleksander.fabijan@mah.se