MIGUEL TORRES
@MM_TR

CARMEL HASSAN
@KARMEL

ANALIZANDO EXPERIENCIAS DE USUARIO CON JS

HI THERE!

MIGUEL TORRES

LEAD FRONTEND DEVELOPER
HTTPS://GITHUB.COM/MMTR/

CARMEL HASSAN

PRODUCT DESIGNER/UX LEAD
HTTP://CARMEL.ES
YES WE TECH, J ON THE BEACH, PYCONES CO-ORGANISER

WE WANT TO SHARE

LESSONS LEARNED ON **AUTOMATED USABILITY TESTING**
OF A LIVING DIGITAL PRODUCT BUILT WITH
JAVASCRIPT
UNDERSTANDABLE BY ANY MEMBER OF THE TEAM.

THE CONTEXT

AGILE METHODOLOGY

WEB APPLICATION

TECHNOLOGY

THE CONTEXT

AGILE METHODOLOGY

WEB APPLICATION

TECHNOLOGY

THE CONTEXT

AGILE METHODOLOGY

WEB APPLICATION

TECHNOLOGY

WHY

ANALYZING AND MEASURING USER EXPERIENCES
AUTOMATICALLY

1/3

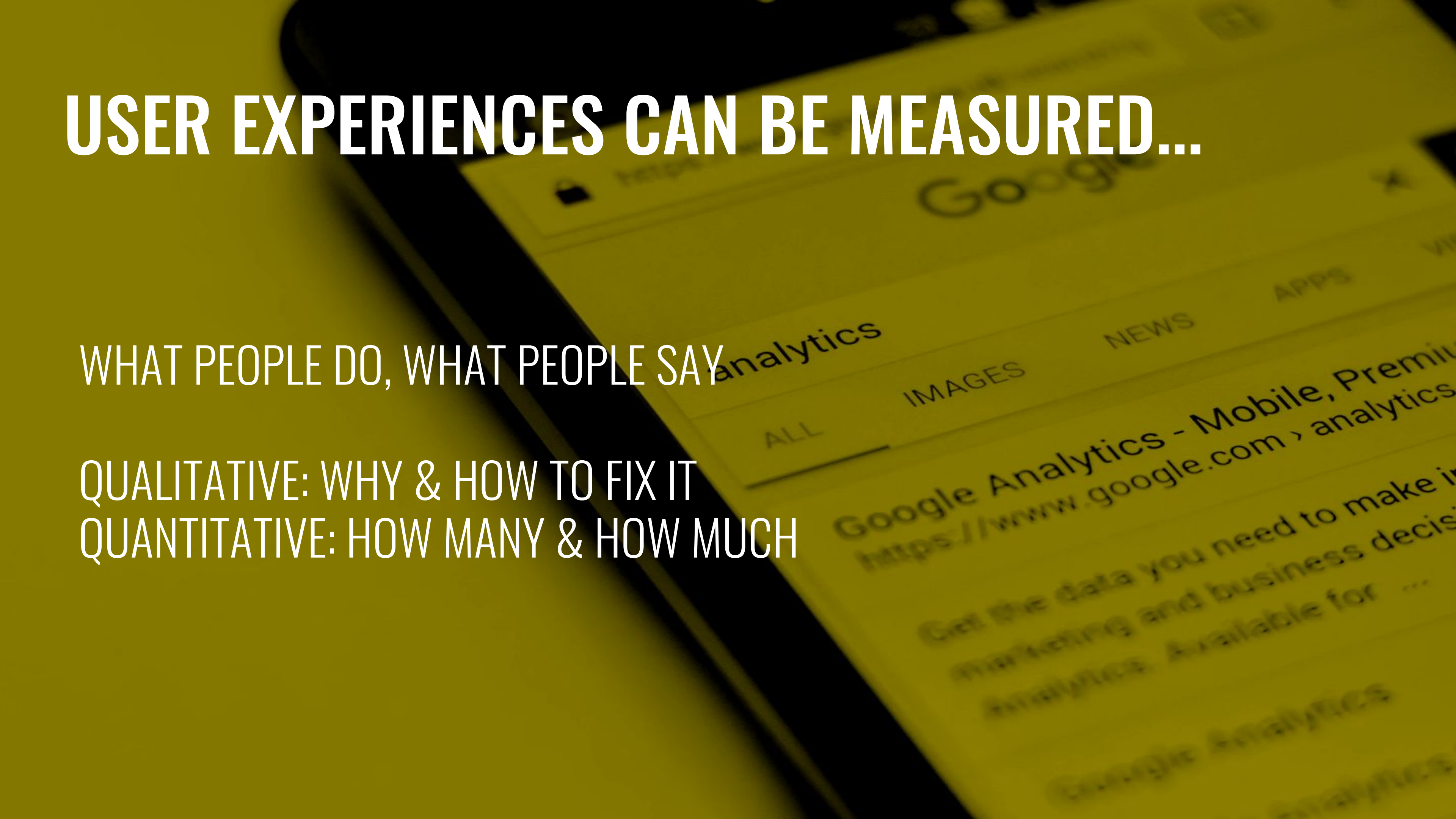
WHY

USING **REAL** AND **MASSIVE** DATA FROM USER INTERACTIONS

2/3

WHY

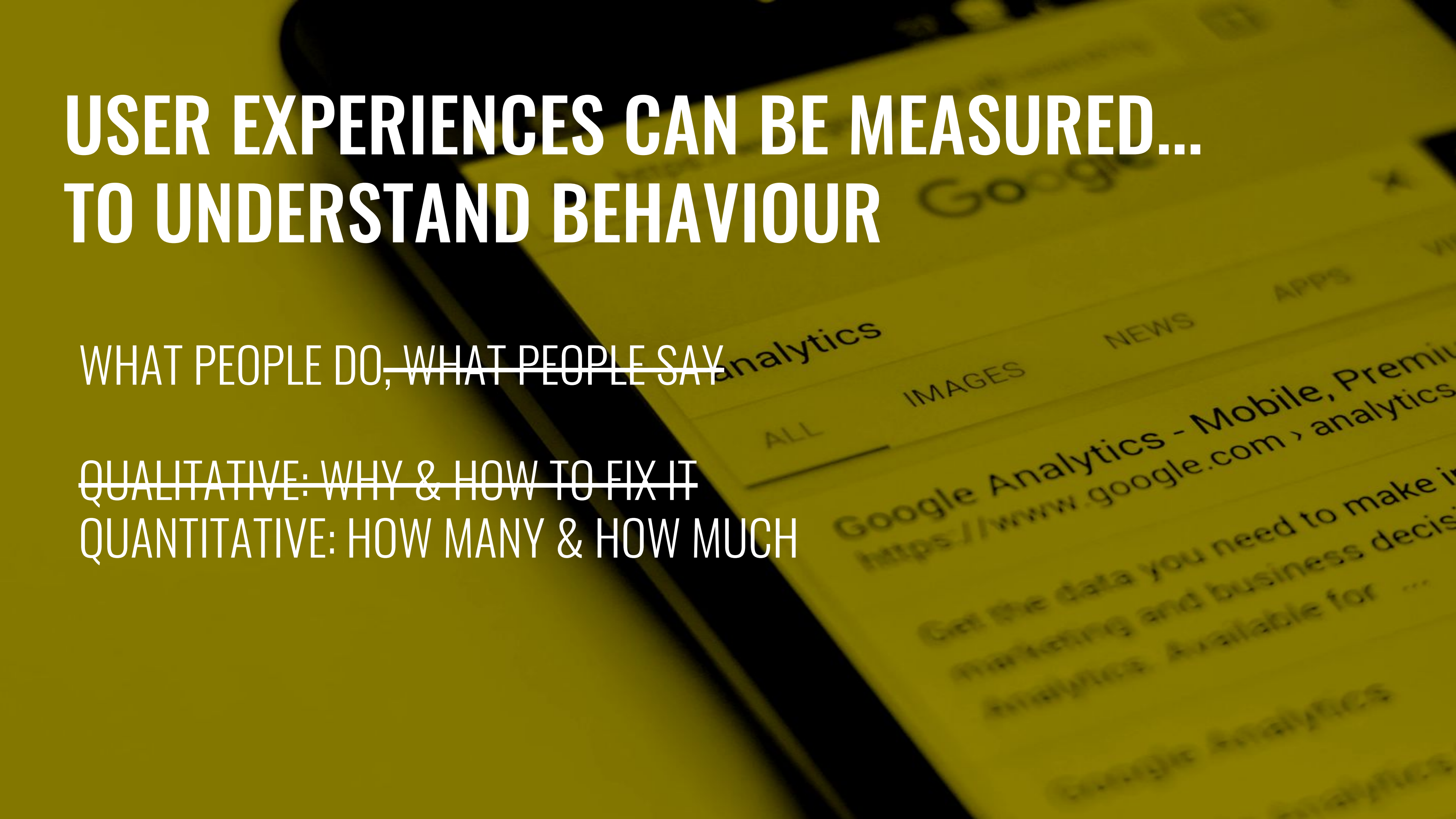
WE MAKE DESIGN AN **EXPERIMENTAL** PROCESS



USER EXPERIENCES CAN BE MEASURED…

WHAT PEOPLE DO, WHAT PEOPLE SAY

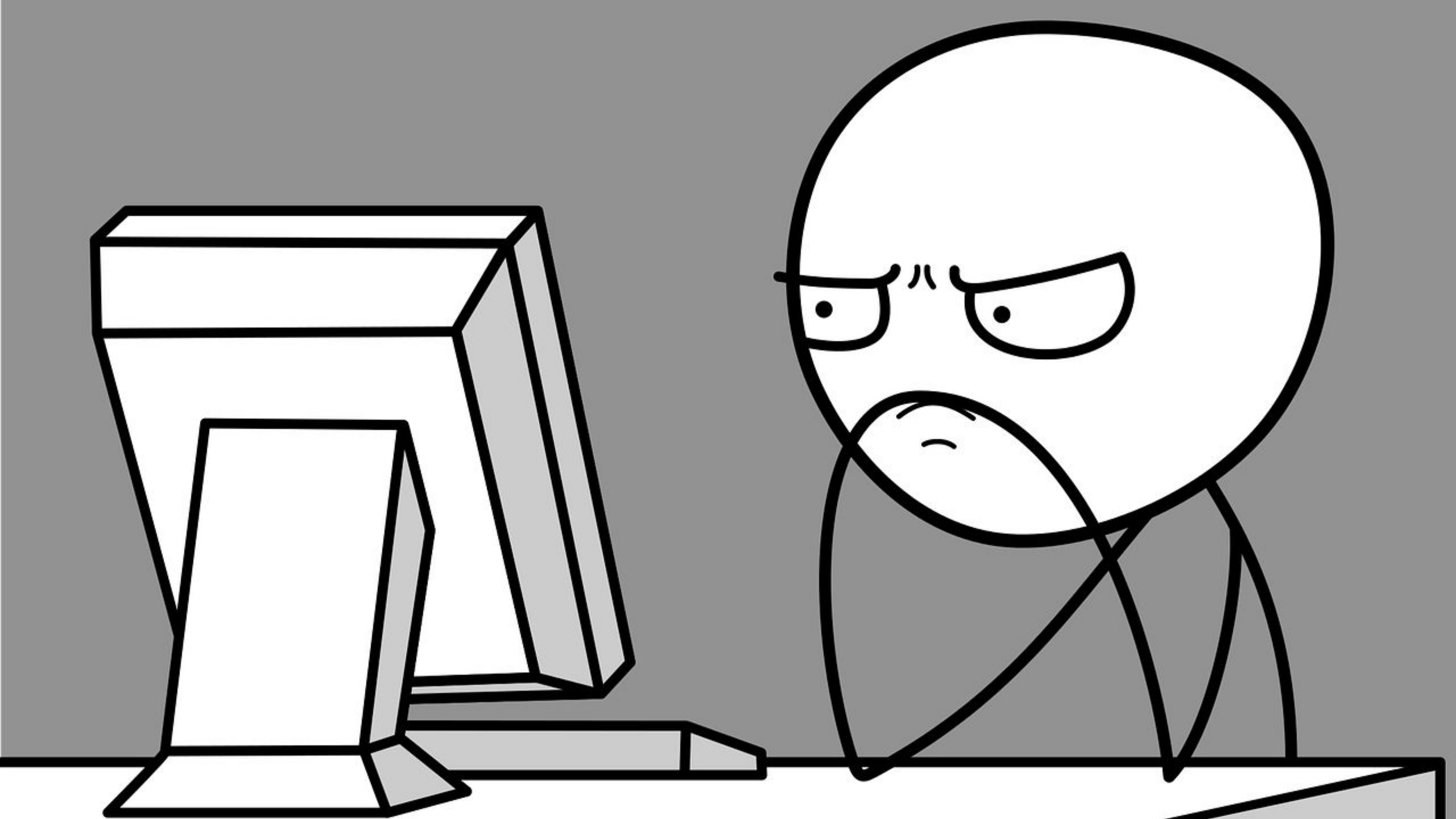
QUALITATIVE: WHY & HOW TO FIX IT
QUANTITATIVE: HOW MANY & HOW MUCH


analytics
ALL
IMAGES
NEWS
APPS
Google Analytics - Mobile, Premi
https://www.google.com › analytics

USER EXPERIENCES CAN BE MEASURED…
TO UNDERSTAND BEHAVIOUR

WHAT PEOPLE DO~~, WHAT PEOPLE SAY~~

~~QUALITATIVE: WHY & HOW TO FIX IT~~
QUANTITATIVE: HOW MANY & HOW MUCH

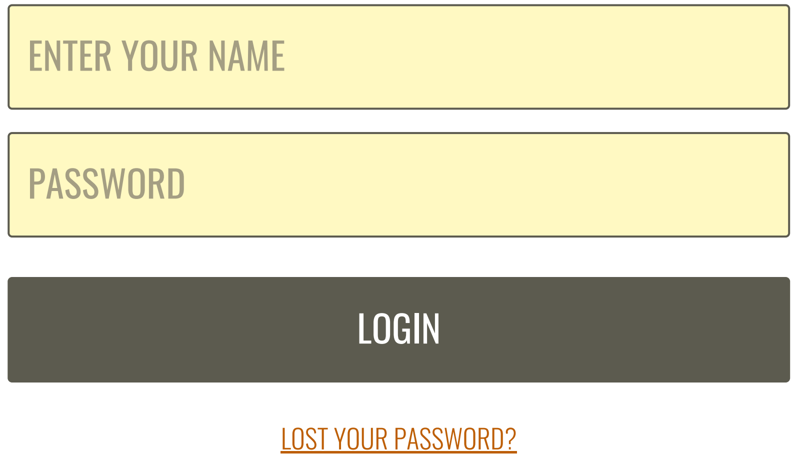


THE TASK CLASS

```
class Task {
  constructor(name) {
    this.name = name;
    this.result = 'In progress';
    this.effort = 0;
    this.errors = 0;
    this.time = 0;
  }
}
```

THE `Task` IS MODELED AS A CLASS WHERE EACH MEASUREMENT IS A PROPERTY

EXAMPLE



LET'S MEASURE THE USABILITY OF A LOGIN PAGE

EXAMPLE (HTML)

```html
<form>
  <input type="text" id="user">
  <input type="password" id="pass">
  <button type="button" id="login">Login</button>
  <a href="#" id="forgot">Lost your password?</a>
</form>
```

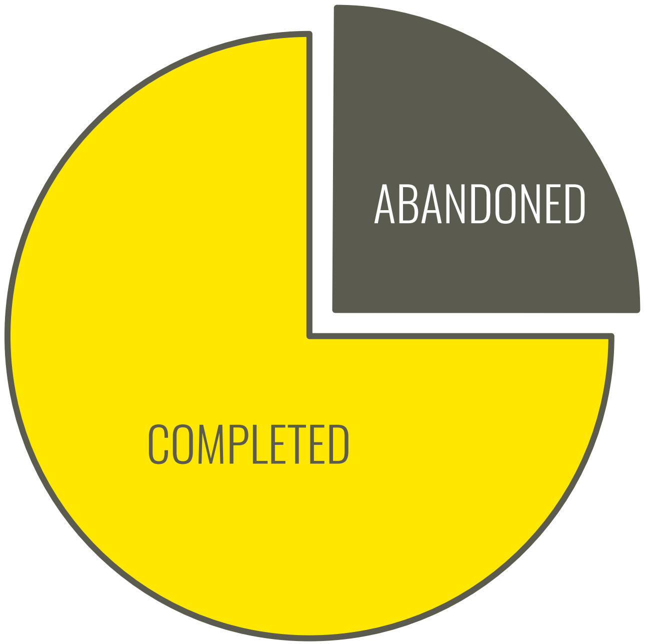
WE'RE USING SIMPLE HTML ELEMENTS

EXAMPLE (JS)

```
const task = new Task('Login');
console.log(task.name); // Login
console.log(task.result); // In progress
```

CREATING THE TASK

WHAT ARE WE MEASURING



RESULT

EFFICIENCY

ERRORS

TIME ON TASK

STORING THE RESULT

```
const IN_PROGRESS = 'In progress';
const COMPLETED = 'Completed';
const ABANDONED = 'Abandoned';

class Task {
  constructor(name) {
    // ...
    this.result = IN_PROGRESS;
    // ...
  }
  // ...
  complete() {
    this.result = COMPLETED;
  }
  abandon() {
    this.result = ABANDONED;
  }
}
```

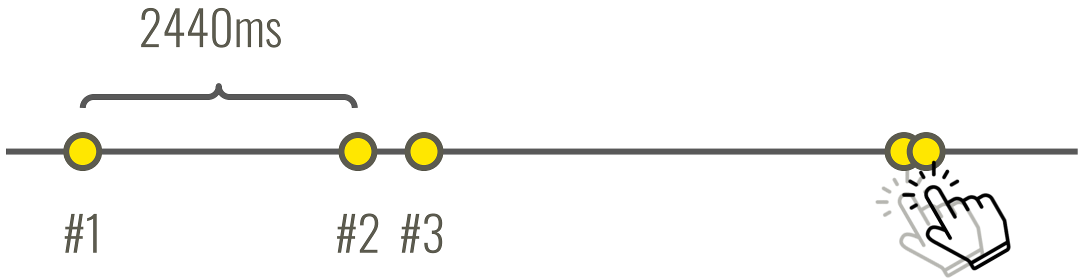
THE `complete` AND THE `abandon` METHODS SHOULD BE CALLED WHENEVER WE CONSIDER THE TASK HAS FINISHED

EXAMPLE (JS)

```js
const login = document.querySelector('#login');
const forgot = document.querySelector('#forgot');

login.addEventListener('click', () => {
  const user = document.querySelector('#user').value;
  const pass = document.querySelector('#pass').value;

  if (user && pass) {
    task.complete();
    console.log(task.result); // Completed
  }
}

forgot.addEventListener('click', () => {
  task.abandon();
  console.log(task.result); // Abandoned
}
```

STORING THE RESULT

WHAT ARE WE MEASURING



RESULT

EFFICIENCY

ERRORS

TIME ON TASK

MEASURING THE EFFICIENCY

```
class Task {
  // ...
  addInteraction() {
    this.effort += 1;
  }
}
```

THE `addInteraction` METHOD NEEDS TO BE EXECUTED EVERY TIME AN INTERACTION IS DONE (CLICK, FOCUS, ...)

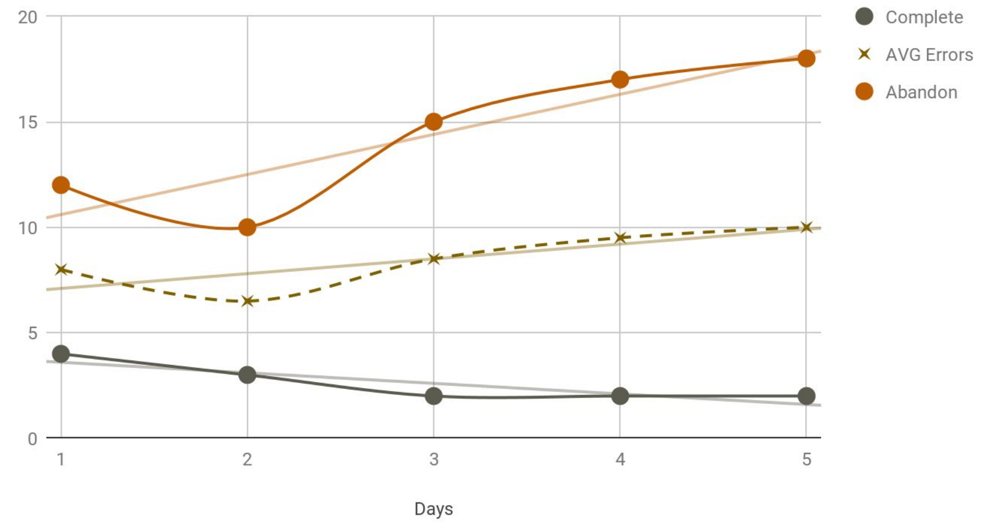
EXAMPLE (JS)

```
const inputs = document.querySelectorAll('input');
const login = document.querySelector('#login');
const forgot = document.querySelector('#forgot');

inputs.forEach((input) => {
  input.addEventListener('focus', () => {
    console.log(task.effort); // N
    task.addInteraction();
    console.log(task.effort); // N + 1
  }
});

login.addEventListener('click', () => {
  task.addInteraction();
});

forgot.addEventListener('click', () => {
  task.addInteraction();
});
```

MEASURING THE EFFICIENCY

WHAT ARE WE MEASURING

Login: average errors over time



RESULT

EFFICIENCY

ERRORS

TIME ON TASK

MEASURING THE ERRORS

```
class Task {
  // ...
  addError() {
    this.errors += 1;
  }
}
```

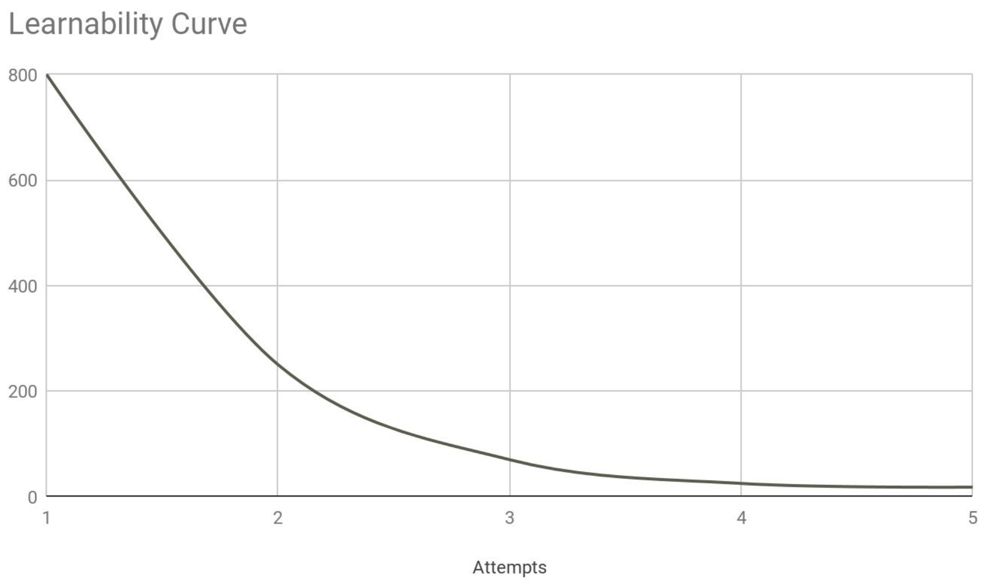
THE addError METHOD NEEDS TO BE EXECUTED EVERY TIME AN ERROR IS RAISED (INVALID FIELD, EXCEPTIONS, ...)

EXAMPLE (JS)

```
const login = document.querySelector('#login');

login.addEventListener('click', () => {
  const user = document.querySelector('#user').value;
  const pass = document.querySelector('#pass').value;

  if (!user || !pass) {
    console.log(task.errors); // N
    task.addError();
    console.log(task.errors); // N + 1
  }
}
```

MEASURING THE ERRORS

WHAT ARE WE MEASURING



RESULT

EFFICIENCY

ERRORS

TIME ON TASK

GETTING THE TIME ON TASK

```
class Task {
  constructor(name) {
    // ...
    this.start = new Date();
    this.end = null;
  }

  // ...

  get time() {
    return this.end - this.start;
  }
}
```

THE time PROPERTY IS REPLACED WITH start AND end PROPERTIES. NEW GETTER FOR RETURNING THE TIME SPENT

GETTING THE TIME ON TASK

```
class Task {

  // …

  finish(result) {
    this.result = result;
    this.end = new Date();
  }

  complete() {
    this.finish(COMPLETED);
  }

  abandon() {
    this.finish(ABANDONED);
  }
}
```

THE `end` PROPERTY IS SET WHEN THE TASK IS COMPLETED OR ABANDONED.

WHAT WE DO KNOW

TASK TIMINGS
ERRORS
EFFICIENCY
LOSTNESS
FRUSTRATION
RAGE
TASK COMPLETION RATE
EFFICIENCY BETWEEN VERSIONS

WHAT WE DON'T KNOW

USER ATTEMPT MEANING
EXTERNAL FACTORS
FAILED TASKS
LEVELS OF SUCCESS
USER ERRORS
COGNITIVE EFFORT

TRACKING METRICS

GOOGLE ANALYTICS

MOST **SIMPLE** WAY OF STORING AND ANALYSING DATA

TASK CAN BE TRACKED AS **EVENTS** WITH ANALYTICS.JS OR GTAG.JS

https://developers.google.com/analytics/devguides/collection/analyticsjs/events
https://developers.google.com/analytics/devguides/collection/gtagjs/events

TRACKING IT

```
class Task {
  // ...
  track() {
    gtag('event', 'timing_complete', {
      event_category: this.name,
      event_label: 'Time on task',
      value: this.time,
      name: this.result,
    });
    gtag('event', this.result, {
      event_category: this.name,
      event_label: 'Error',
      value: this.errors,
    });
    gtag('event', this.result, {
      event_category: this.name,
      event_label: 'Effort',
      value: this.effort,
    });
  }
}
```

THE `track` METHOD SENDS EVENTS TO GOOGLE ANALYTICS

TRACKING IT

```
class Task {
  // ...

  finish(status) {
    // ...
    this.track();
  }

  // ...
}
```

THE `track` METHOD IS CALLED WHEN A TASK FINISHES

HOW WAS THIS APPROACHED

JAVASCRIPT

KISS PRINCIPLE

ITERATING DEVELOPMENT

HOW WAS THIS APPROACHED

JAVASCRIPT

KISS PRINCIPLE

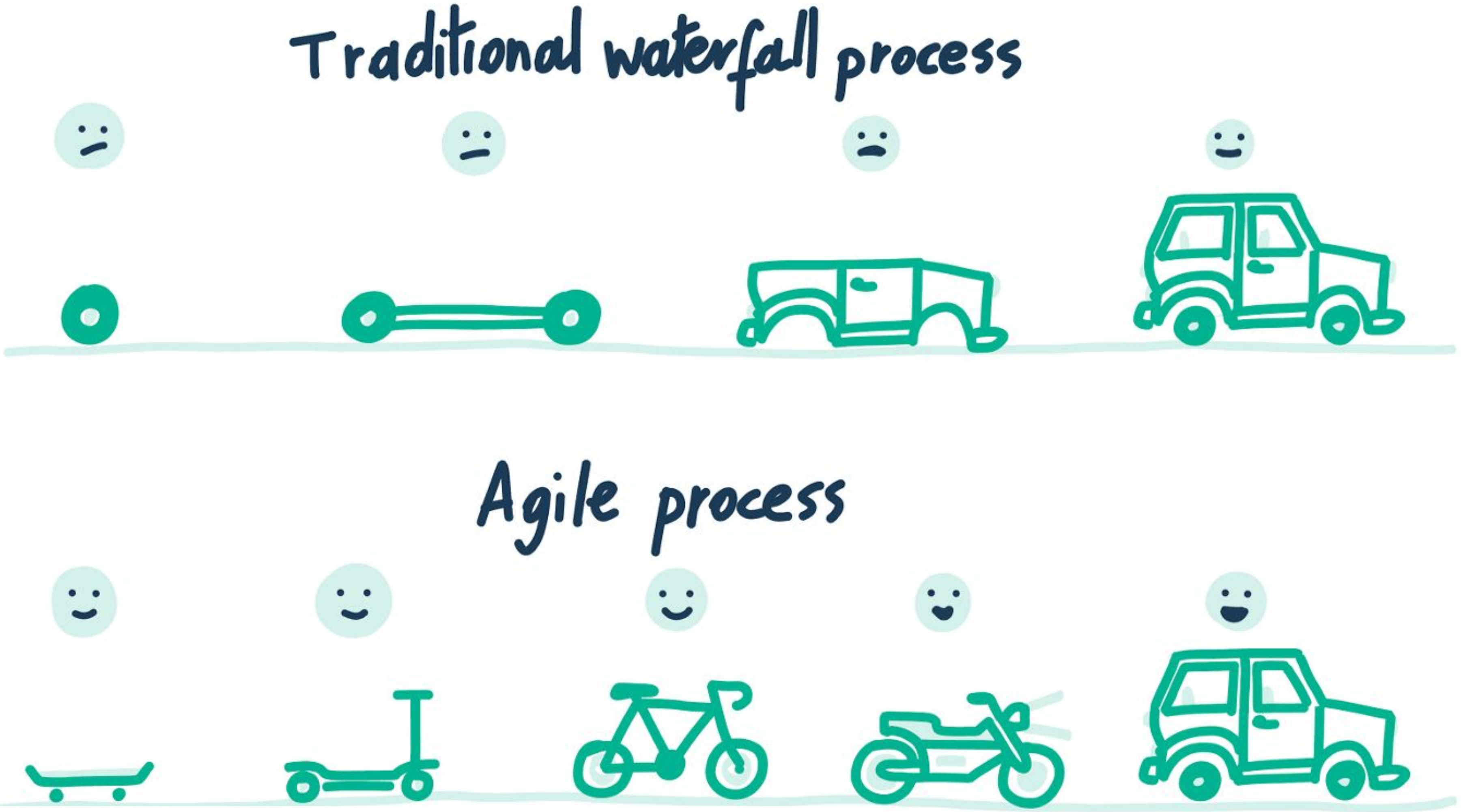
ITERATING DEVELOPMENT

HOW WAS THIS APPROACHED

JAVASCRIPT

KISS PRINCIPLE

ITERATING DEVELOPMENT



[Skateboard, Bike, Car concept]() by Andrew Wilkinson, Illustrated by [Frederik Vincx]()

HUHA.JS

HUHA.JS

HYPERACTIVE **U**SERS **H**INT **A**NALYSIS
JAVASCRIPT FRAMEWORK READY TO USE
OPEN SOURCE
CONTRIBUTORS

[HTTPS://GITHUB.COM/EBURY/HUHA](https://github.com/ebury/huha)

DEMO

CODE: HTTPS://GITHUB.COM/MMTR/OPENSOUTHCODE-SURVEY
LIVE: HTTPS://MMTR.GITHUB.IO/OPENSOUTHCODE-SURVEY/
RESULTS AT GOOGLE ANALYTICS

THANK YOU!

MIGUEL TORRES

LEAD FRONTEND DEVELOPER
HTTPS://GITHUB.COM/MMTR/

CARMEL HASSAN

PRODUCT DESIGNER/UX LEAD
HTTP://CARMEL.ES
YES WE TECH, J ON THE BEACH, PYCONES CO-ORGANISER