master-thesis-experiments

Kevin Hernández Rostrán

2023-11-10

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

1 Introducción

El artículo “Un método para determinar el peso de las contribuciones de los programadores para medir el riesgo asociado a su salida de un proyecto” aborda el problema de la rotación de programadores en proyectos de software. La rotación de programadores puede tener un impacto negativo significativo en un proyecto, ya que puede provocar la pérdida de conocimiento crítico, la disminución de la productividad y el aumento de los costes.

Los métodos de gestión de proyectos por lo general no consideran el grado y la complejidad de los aportes que realizan los desarrolladores y el riesgo que tiene asociada su eventual salida de un proyecto. Por ello, se requiere de métodos que permitan conocer la complejidad de los cambios que realizan, y permitan medir la concentración de conocimiento y el nivel de dependencia que se tiene de cada programador durante el desarrollo y mantenimiento del software.

El artículo propone un método para medir el riesgo asociado a la salida de un programador de un proyecto de software. El método se basa en la hipótesis de que el grado de conocimiento de un programador sobre un sistema está relacionado con la complejidad de los cambios que realiza.

2 Estructura/dependencias

Este proyecto depende de:

- GAST
- Mapper
- Analyzer

Para minar se utiliza:

- repositoryminer

3 Experimentos

3.1 Visualizaciones

3.1.1 Complejidad única

3.2 Apéndice

3.2.1 Variables y config

```
changes = selectedRepository[0].changes
```

```
mutable footerdata = null;
```

```
unique_cx_metrics = {
  const grouped_data = d3.hierarchy(
    changes_with_dev_key
      .groupby("_package", "_class", "method")
      .orderby(aq.desc("date"))
      .derive({ last: aq.rolling((d) => op.first_value(d.cur_m_cx)) })
      .orderby("date")
      .groupby("_package", "_class", "method", "last")
      .pivot({ key: (d) => d.dev_key }, { value: (d) => op.any(d.last) || 0 })
      .groupby("_package", "_class")
      .objects({ grouped: "map" })
  );

  const tree_summary_data = grouped_data.copy().eachAfter((d) => {
    devs_by_email.forEach((k) => {
      d[k] = d.children ? _.sumBy(d.children, k) : d.data[k] || 0;
    });
```

```
//| panel: sidebar

viewof selectedRepository = Inputs.select(
  d3.group(repos, (repo) => repo.repository.name),
  { label: "Elegir repositorio" }
)

stadistics = {

  const data2 = footerdata.filter(v => v.title === "Complejidad única")

  const df22 = aq.from(data2).fold(aq.not('title', 'cx'), { as: ['dev', 'metric

  return vl.markBar()
            .data(df22)
            .encode(
              vl.y().fieldN('dev'),
              vl.x().fieldQ('metric')
            )
            .width(160)
            .height(400)
            .render()
}

viewof riskPercentile = Inputs.range([0, 100], {step: 1, label: "Percentil de ri
```

```
//| panel: fill
t = {
  const holderEl = document.createElement("div");

  // tag ids
  const header_container_id = "unique_header";
  const body_container_id = "unique_body";
  const footer_container_id = "unique_footer";

  // create
  const header_table = create_header(unique_cx_metrics, header_container_id);
  const body_table = create_body(unique_cx_metrics, body_container_id);
  const footer_table = create_footer(unique_cx_metrics, footer_container_id);

  // append to holder
  holderEl.appendChild(header_table.element);
  holderEl.appendChild(body_table.element);
  holderEl.appendChild(footer_table.element);
```



```
  // helper to support scrolling
  header_table.on("scrollHorizontal", function (left) {
    document.querySelector(
      `#${body_container_id} .tabulator-tableholder`
    ).scrollLeft = left;
    document.querySelector(
      `#${footer_container_id} .tabulator-tableholder`
```

```
  d["cx"] = d.children ? _.sumBy(d.children, "cx") : d.data["last"] || 0;

  d.title = d.data[0] || d.data.method;
});

const { devs_with_zero, devs_with_non_zero } = _.chain(devs_by_email)
  .partition((key) => _.every(tree_summary_data.children, (d) => d[key] == 0))
  .map(_.sortBy)
  .thru((v) => ({ devs_with_zero: v[0], devs_with_non_zero: v[1] }))
  .value();

const picked = _.concat(devs_with_non_zero, ["cx"]);

const total_cx_data = _.pick(tree_summary_data, picked);

const percentage_cx_data = _.transform(
  total_cx_data,
  (result, value, key) => {
    result[key] = +((value * 100) / total_cx_data["cx"]).toFixed(2);
  },
  {}
);

const unique_cx_data = calculate_unique_cx_summary(
  tree_summary_data.children,
  devs_with_zero,
  devs_with_non_zero
);

const percentage_unique_cx_data = _.transform(
  unique_cx_data,
  (result, value, key) => {
    result[key] = +((value * 100) / total_cx_data["cx"]).toFixed(2);
  },
  {}
);

const knowledge_remaining = calculate_knowledge_remaining(tree_summary_data.children,devs_wi
const knowledge_remaining_percentage = 100 - (+((knowledge_remaining * 100) / total_cx_data[
```

```
  return {
    grouped_data,
    tree_summary_data,
    devs_with_zero,
    devs_with_non_zero,
    total_cx_data,
    percentage_cx_data,
    unique_cx_data,
    percentage_unique_cx_data,
    knowledge_remaining,
    knowledge_remaining_percentage
  };
}
```

3.2.1.1 Limpiar devs

```
changes_with_dev_key = {
  const changes_with_dev_email_splitted = aq.from(changes)
    .derive({ splitted: (d) => op.split(d.dev_email, "@") })
    .derive({ username: (d) => op.lower(d.splitted[0]) }) // normalize to lower
    .derive({ domain: (d) => op.lower(d.splitted[1]) }) // normalize to lower
    .derive({
      new_email: (d) => d.username + (d.domain ? "@" + d.domain : "")
    });

  const changes_with_dev_email_splitted_objects =
    changes_with_dev_email_splitted.objects();

  const use_username =
    _.chain(changes_with_dev_email_splitted_objects.map((d) => d.username))
      .uniq()
      .size()
      .value() ==
    _.chain(changes_with_dev_email_splitted_objects.map((d) => d.new_email))
      .uniq()
      .size()
      .value();
```

```
  return aq.from(changes_with_dev_email_splitted)
    .derive({
      dev_key: aq.escape((d) => (use_username ? d.username : d.new_email))
    })
    //.view();
}
```

```
devs_by_email = changes_with_dev_key
 .dedupe('dev_key')
 .select('dev_key')
 .filter(d => d.dev_key !== "noreply@github.com" && d.dev_key !== "noreply") // filter noreply
 .orderby('dev_key')
 .array('dev_key')
```

3.2.2 Tabulator

```
Tabulator = require("tabulator-tables@5.5.2")
```

3.2.3 Helper Functions

```
create_header = (metrics, containerId) => {
  const data = _.chain(metrics.devs_with_non_zero)
    .reduce((p, v) => ({ ...p, [v]: true }), {})
    .thru((v) => [v])
    .value();

  const container = document.createElement("DIV");
  container.id = containerId;
  const table = new Tabulator(container, {
    data: data,
    layout: "fitDataStretch",
    //height: 260,
    headerSort: false,
    nestedFieldSeparator: false,
    columns: [
      {
```

```
        title: "Software Item",
        headerSort: false,
        width: 200,
        responsive: 0,
        frozen: true
      },
      ...metrics.devs_with_non_zero.map((d) => ({
        title: d,
        field: d,
        hozAlign: "center",
        headerSort: false,
        editor: true,
        width: 50,
        headerVertical: "flip",
        formatter: "tickCross"
      })),
      { title: "Complejidad", width: 150, headerSort: false }
    ]
  });
  return table;
}
```

```
create_body = (metrics, containerId) => {
  const data = _.cloneDeep(metrics.tree_summary_data).children;
  //const footerElement = footer_table.element;
  const container = document.createElement("DIV");
  container.id = containerId;

  const table = new Tabulator(container, {
    data,
    dataTree: true,
    dataTreeChildField: "children",
    height: 350,
    headerVisible: false,
    layout: "fitDataStretch",
    nestedFieldSeparator: false,
    columns: [
      {
        title: "Software Item",
        field: "title",
```

```
        width: 200,
        responsive: 0,
        frozen: true
      },

      ...metrics.devs_with_non_zero.map((d) => ({
        ["title"]: d,
        ["field"]: d,
        width: 50,
        headerVertical: "flip"
      })),
      { title: "Complejidad", field: "cx", width: 150 }
    ],
    //footerElement: footerElement
  });

  return table;
}
```

```
create_footer = (metrics, containerId) => {
  const data = [
    {
      title: "Complejidad total",
      ...metrics.total_cx_data
    },
    {
      title: "Complejidad total (%)",
      ...metrics.percentage_cx_data
    },
    {
      title: "Complejidad única",
      ...metrics.unique_cx_data
    },
    {
      title: "Complejidad única (%)",
      ...metrics.percentage_unique_cx_data
    },
    {
      title: "Conocimiento restante",
      cx: metrics.knowledge_remaining_percentage
```

```
  }
];

const container = document.createElement("DIV");
container.className = "tabulator-footer";
container.id = containerId;
const table = new Tabulator(container, {
  data,
  //height: 150,
  layout: "fitDataStretch",
  headerVisible: false,
  nestedFieldSeparator: false,
  cssClass: "tabulator-footer",
  rowFormatter: function (row) {
    const row_data = row.getData();

    if (row_data.title == "Conocimiento restante") {
      if(row_data.cx < riskPercentile){
        row.getElement().style.backgroundColor = "#ff0000";
        row.getElement().style.color = "#fff";
        row.getElement().style.fontWeight = "bold";
      }
    }
  },
  columns: [
    {
      title: "Software Item",
      field: "title",
      width: 200,
      responsive: 0,
      frozen: true
    },
    ...metrics.devs_with_non_zero.map((d) => ({
      ["title"]: d,
      ["field"]: d,
      width: 50,
      headerVertical: "flip"
    })),
    { title: "Complejidad", field: "cx", width: 150 }
  ]
```

```
  });

  return table;
}
```

```
calculate_unique_cx_summary = (data, devs_with_zero, devs_with_non_zero) => {
  const picked = _.concat(devs_with_non_zero);

  let map = devs_with_non_zero.reduce((p, c) => {
    return { ...p, [c]: 0 };
  }, {});

  //let cx = 0;
  data.forEach((p, p_index) => {
    p.children.forEach((c, c_index) => {
      c.children.forEach((m, m_index) => {
        //console.log(m)
        const element = _.pick(m, picked);
        const values = Object.values(element);
        //console.log(element)
        //console.log(values)

        const max = _.max(values);
        const sum = _.sum(values);
        const can = max == sum;
        if (can && max != 0) {
          //console.log('max',max, 'sum' , sum)
          for (const [key, value] of Object.entries(element)) {
            if (value == max) {
              //console.log(key, value)
              map[key] += value;
              //console.log(map)
              //cx = value;
              break;
            }
          }
        }
      });
    });
  });
```

```
  //map['cx'] = cx;

  return map;
}
```

```
calculate_knowledge_remaining = (data, devs_with_non_zero) => {
  const picked = _.concat(devs_with_non_zero);

  let result = 0;

  data.forEach((p, p_index) => {
    p.children.forEach((c, c_index) => {
      c.children.forEach((m, m_index) => {
        const element = _.pick(m, picked);
        const values = Object.values(element);

        const all_zeros = _.every(values, d => d == 0);

        if (all_zeros) {
          result += m['cx']
        }
      });
    });
  });

  return result;
}
```

3.2.4 Fetchers

```
repos = {
  const jgit_cookbook = {
    repository: { name: "java.jgit_cookbook" },
    changes: await FileAttachment("./../../data/java/changes_jgit_cookbook.json").json()
  };

  const uCrop = {
    repository: { name: "java.uCrop" },
```

```
  changes: await FileAttachment("./../../data/java/changes_uCrop.json").json()
};

const java_jwt = {
  repository: { name: "java.java_jwt" },
  changes: await FileAttachment("./../../data/java/changes_java_jwt.json").json()
};

const mapstruct = {
  repository: { name: "java.mapstruct" },
  changes: await FileAttachment("./../../data/java/changes_mapstruct.json").json()
};

const modelmapper = {
  repository: { name: "java.modelmapper" },
  changes: await FileAttachment("./../../data/java/changes_modelmapper.json").json()
};

const javacv = {
  repository: { name: "java.javacv" },
  changes: await FileAttachment("./../../data/java/changes_javacv.json").json()
};

const spatial4j = {
  repository: { name: "java.spatial4j" },
  changes: await FileAttachment("./../../data/java/changes_spatial4j.json").json()
};

const google_http_java_client = {
  repository: { name: "java.google_http_java_client" },
  changes: await FileAttachment("./../../data/java/changes_google_http_java_client.json").jso
};

const jadx = {
  repository: { name: "java.jadx" },
  changes: await FileAttachment("./../../data/java/changes_jadx.json").json()
};


return [
```

```
    jgit_cookbook,
    uCrop,
    java_jwt,
    mapstruct,
    modelmapper,
    javacv,
    spatial4j,
    google_http_java_client,
    jadx
  ];
}
```

3.2.5 Imports

```
d3 = require("d3@7")
```

3.2.6 Notebook Imports

```
import { aq, op } from '@uwdata/arquero'
```

References