Vol.2021-SE-209 No.3
2021/11/25

定量的指標に基づいた フロントエンドフレームワーク選定補助システムの提案

清川航一 [1,a)]　金群 [1,b)]

概要：フロントエンドフレームワークは Web アプリケーション開発において複雑な UI を効率よく作るために需要が高まっている．しかし，進化が早く，種類も多いため人力で比較するのは困難である．先行研究によって選定の際に開発者が重要視する指標がいくつか明らかになっている．本研究では，そうした指標を GitHub などからデータを収集して定量的に計算し，開発者に提示して選定を補助するシステムを構築した．このシステムは Web アプリケーションとして提供され，ユーザーは各指標に対して重みを入力することで，それぞれの指標をどれくらい重要視するかを指定できる．本稿では，用いる指標，システムの構成とその評価方法について述べる．

キーワード：技術選定，フロントエンドフレームワーク，半構造化インタビュー

1. はじめに

フロントエンドフレームワークは Web アプリケーションのユーザーインターフェースを効率よく作成するためのツールである．フロントエンドフレームワークは種類が多く，進化も早い．選定の際には，多くの選択肢の中からプロジェクトの要件やメンバーのスキルに応じて適切なものを見つける必要があり，開発者が人力で意思決定を行うには負担がとても大きい[1]．また，一度選定を行って開発を進めた場合，あとから別のフレームワークへ移行するのはとても工数がかかるため慎重に選定を行う必要がある．

フレームワークの選定には，定量的なデータに基づいて判断できるものと，定性的に判断せざるを得ないものに分けられる．例えば，フーレムワークの人気や開発の活発さといった指標はダウンロード数やコミット数といった定量的なデータから判断可能である．一方で「プロジェクトの要件にフレームワークの特性があっているか」といった視点は定性的な判断を行う必要がある．

本研究では，先行研究によって示されたいくつかの指標のうち，データから定量的に導出可能と考えられる指標について，データを収集・計算してユーザに提示することで選定を補助するシステムを構築し，開発者の負担軽減を試みた．

2. 関連研究

2.1 サードパーティライブラリの選定における指標

Enrique らはサードパーティ製ライブラリの選定プロセスに影響を与える要因について研究を行った[1]. 11 の異なる業界から 16 人の開発者に対してインタビューを行い，またライブラリ選定に関わる 115 人の開発者に対してアンケート調査を行った．その結果，ライブラリの選定要因として，大きく分けて技術的要因・人的要因・経済的要因の 3 種類があることを明らかにした．このうち人的要因と経済的要因については定性的な要素がほとんどを占める．一方で技術的要因については機能性(サイズや複雑度, 目的に合致しているか)・品質(使いやすさ，セキュリティ，パフォーマンス, テストのカバレッジ)・リリース(メンテナンスがされているか，成熟度と安定感，開発の活発さ)，コミュニティ(人気, 情報共有の活発さ)などが挙げられており，データをもとに定量的に判断できる可能性がある．

2.2 特定のフレームワーク同士の比較

Kaluza らは Angular, React, Vue の 3 つのフロントエンドフレームワークについて，構文・習得難易度・性能を比較した[2]．この研究は特定のフレームワークの，特定のバージョンを比較したものである．ライブラリの新しいバージョンが出ると比較結果は古くなってしまう．

2.3 既存のライブラリ比較システム

npm は JavaScript 用のライブラリを保存するレジストリである．ライブラリの公開，ダウンロードを行うことができる．Npms は npm に公開されたライブラリの比較ツールである[3]．npm からダウンロード数などのデータを，GitHub からソースコード，コミット情報などのデータを取得し，品質,・人気・ メンテナンスの 3 指標でライブラリを点数化している．また以下の計算式でトータルスコアを算出している．

$$\text{トータルスコア} = \text{品質} \times 0.3 + \text{人気} \times 0.35 + \text{メンテナンス} \times 0.35$$

式に登場する 0.3, 0.35 といった係数は Npms の作者によって恣意的に決められた値である．システムを利用者によって，どの指標を重要視するかに違いがあると考えられるが, Npms ではその違いに対処することができない．システムのユーザーインターフェースは，検索欄にライブラリ名やライブラリに登録されたキーワードを入力し，部分一致で検索する仕様である．キーワードはライブラリの作成者

1　早稲田大学 人間科学研究科
Graduate School of Human Sciences, Waseda University
a)　koichi19937@toki.waseda.jp
b)　jin@waseda.jp

が自由に設定できるものであり，統一された基準が存在しない．そのため，Npms では「フロントエンドフレームワーク同士」といった同じカテゴリのライブラリ同士を比較することが難しい．

Orion は独自のクエリ言語によるソフトウェアの検索システムである[5]．ライブラリの選定だけでなく，再利用可能なコードを探すこと，開発に貢献できるオープンソースプロジェクトを探すことも想定されている．このシステムでは，独自のクエリ言語を用いて「10 人以下で開発された，10000 万行以上のコードを含み，バグ修正率 80%以上のプロジェクト」といった検索を行うことができる．検索の自由度が高いため，ユーザーの細かい要望に対応できる可能性がある一方で，独自のクエリ言語の学習にコストがかかる．また，ひとつの指標を計算するのに複雑なクエリをユーザーが考える手間がある．

3. 提案手法

本研究で提案するシステムは，Kaluza ら[2]の行った特定のバージョンの比較と異なり，システム利用時点におけるフロントエンドフレームワークの最新データをもとに比較できるように設計した．また，Npms[3]と異なり，どの指標をどれだけ重要視するかをシステムのユーザーが設定できるようにした．さらに，フロントエンドフレームワークにあまり詳しくない開発者も利用することを想定している．

3.1 指標

本研究では，先行研究[1][6][7]に基づき，以下の指標を用いた．

表 1. 扱う指標と使用するデータ

指標	使用データ
開発の活発さ	Git のコミット
情報共有の活発さ	GitHub の issue, Stack Overflow の質問
既存技術との API の類似度	RealWorld プロジェクトのコード
メンテナンス	Git のコミット，issue
成熟度	バージョンの公開履歴
人気	GitHub のスター数，npm のダウンロード数

それぞれの指標に共通して,「新しいデータほど価値を高める」という方針をたてた．そのために，スコアの計算の際には，経過日数が大きいほど値が小さくなるように

$$score_{age}(d) = \frac{D}{D+d}$$

という関数を用いる（d はそのデータにおける経過日数を表し, D は定数である）．これは当日のデータに対して D 日前のデータの価値が半分になる．本研究では$D = 30$ とした．また，取得するデータは直近 1 年に限定した．これは，取得する年数を増やすと，公開日の古いフレームワークが有利になりすぎるためである．また，それぞれの指標は 0~1 の範囲に正規化してからユーザーに提示する．以下で，それぞれの指標について解説を行う．

3.1.1 開発の活発さ

新しい機能の追加, バグの修正などが積極的に行われているかを表す．これらが積極的に行われているフロントエンドフレームワークを選択することで，フロントエンド開発者がフレームワーク利用中にフレームワークの機能不足やバグに気づいて報告した際に，機能の追加やバグの修正を比較的短期間で行ってもらえる確率が高まり，よりフロントエンドの開発効率を高めることが期待される．フレームワーク開発者は，ソースコードの変更をコミット意味のある単位にまとめて保存する．したがって，変更行数の多いコミットが多くなされているほど開発が活発であると考えられる．これは先行研究[3]でも同様に考えられている．また，コミットは直近のものほど価値を高く設定する．これらを踏まえて，開発の活発さは式(1)のように計算した．

$$score_{dev} = \sum^{直近1年のコミット} (変更行数) \times score_{age}(d_{commit}) \quad (1)$$

(ここでd_{commit}はコミットされてからの経過日数を表す.)

3.1.2 情報共有の活発さ

エラーメッセージや実現したい動作をキーワードにして検索した際に，解決策となる情報が得やすいということが重要視される．情報が得やすいことにより，そのフレームワークの深い知識がなくても課題を短い時間で解決することが容易になる．情報の得やすさを支えるのは，フレームワーク利用者による情報共有である．例えば，GitHub 上での issue の投稿(バグの報告や使い方に関する質問など), Stack Overflow などの技術質問サイトへの質問とそれに関する回答などが挙げられる．issue についてはコメントの文字数が多いほど価値を高め，技術質問サイトへの質問については回答数が多いほど価値を高める. これらを踏まえて，式(2)のように計算した．

$$score_{share} = \sum^{issueコメント} (コメント文字数) \times score_{age}(d_{comment}) + \sum^{直近1年の質問} (回答数) \times score_{age}(d_{question}) \quad (2)$$

（ここで，$d_{comment}$はコメントされてからの経過日数，$d_{question}$は質問が作成されてからの経過日数を表す）

3.1.3 既存技術との API の類似度

基本的に開発者が使用経験のないフレームワークは選定の対象にしづらい．これは新たに学習するのにコストがかかるためである．選定から開発開始までの期間が短い場合は学習する時間が限られるため，特に選定対象から外れや

すい．しかし，開発者が使用経験のないフレームワークであっても，その開発者が習得済みの技術と API が似ている場合は，学習にかかるコストが削減できるため，選定の対象になりえる．ここで，API とはコンポーネントの定義方法や状態変数の定義方法などを指す．

図 1,2 は Vue と Svelte という 2 つのフレームワークで API が似ている例である．ボタンを押すとカウントが 1 増える簡単なアプリを実装している．どちらも一つのファイルでコンポーネントを定義し，UI を表す template 部分と，機能を表す script 部分に分かれている．この場合，Vue を取得している開発者が Svelte を習得することは難しくないため，Svelte も選定の対象になる．

```
<template>
  <div>
    <span>{count}</span>
    <button>+1</button>
  </div>
</template>

<script>
export default {
  data: () => ({
    count: 0
  }),
  methods: {
    increment() {
      this.count++
    }
  }
}
</script>
```

図 1 Vue でカウンターを実装した例

```
<script>
  let count = 0
  function increment() {
    count++
  }
</script>

<div>
  <span>{count}</span>
  <button>+1</button>
</div>
```

図 2　Svelte でカウンターを実装した例

本研究では RealWorld オープンソース[4]のソースコードを用いて API の類似度を計算した．RealWorld は仕様の定まったブログアプリを，様々なフレームワークを用いて実装するオープンソースプロジェクトであり，実用性のあるデモアプリを提供することを目的としている．フレームワーク間のコードの比較にも適している．本研究では，ログインページのソースコードを比較に用いた．これは，記事一覧ページなどは，フレームワークによってコンポーネント分割の粒度，バックエンドとの通信方法，状態保持の方法が異なり，API の違い以外の差が大きすぎるためである．ログインページについては十分シンプルであるため，API の違いがメインとなり，比較に適する．ログインページのソースコードについて，単語の出現回数によってベクトル化し，コサイン類似度を用いて計算を行った．具体的には，ソースコード A と B を比較する際，以下のアルゴリズムで計算した．

1. 前処理として，A, B からストップワード(セミコロン，インデント，改行)を削除する
2. A, B に出現する単語をひとつの集合にまとめる
3. 2 の集合を辞書順にソートしてリストを作る
4. 3 のリストの順番に単語の出現回数をカウントし，ベクトルにする

単語の出現回数によってベクトル化したのは，「同じようなコードでも，出現する順番が違うだけで類似度が低くなる」という問題を回避するためである．図 1 と 2 のソースコードを例にあげると，API は類似しているものの template タグと script タグの順番が異なる．

3.1.4 メンテナンス

本研究では，既存の比較システム[3]を参考に，メンテナンスを「フロントエンドフレームワークを開発しているチームのメンバーが，バグの修正や仕様の質問に対して対処している度合い」と定義する．例えば，フロントエンドフレームワーク利用者がバグと思しき挙動を見つけた際に issue として報告を行う．これに対して開発チームのメンバーが 1.その挙動がバグであるのか　2.バグであるならばどれほどの優先度で修正すべきか 3.バグでないならどのように解決するかなどを回答する．こうした回答が滞ることで，フレームワークは利用者のフィードバックを反映できなくなるという問題が発生する．

「メンテナンスがされている状態」は issue が早く解決されること，また開発メンバーによる issue へのコメントが多くされることなどが考えられる．逆に issue が解決されずに放置されているほど，メンテナンスがされていないと言える．とくに，多くのコメントがなされている issue は，フレームワークの多くが同様の問題に困っているということであり，放置されたときの悪影響が大きい．これらを踏まえて，「メンテナンスがされているか」のスコアを

1. issue の早期解決度合い
2. 開発メンバーの issue への参加度合い
3. issue の放置度合い

の 3 つのサブスコアにわけて考えた．

1. issue の早期解決度合いについては，issue が open されてから close されるまでの日数が短いほど値が大きくなり，また新しい issue ほど値が大きくなるようにした．

$$subscore_{close\_speed} = \sum^{\text{直近1年にcloseされたissue}} score_{age}(d_{elapsed}) \times score_{age}(d_{close}) \quad (3)$$

(ここで，$d_{elapsed}$は issue が open されてから close されるま

での日数，d_{close}は issue が close されてからの経過日数を表す．)

2. 開発メンバーの issue への参加度合いについては，開発メンバーが issue で投稿したコメントについて，コメントの文字数が多いほど，またコメントが新しいほど値が大きくなるようにした．

$$subscore_{comment} = \sum_{\text{開発メンバーによるissueコメント}} (\text{コメント文字数}) \times score_{age}(d_{comment}) \quad (4)$$

3. issue の放置度合いについては，多くのコメントがついている issue を長く放置しているほど値が大きくなるようにした．この値については 1,2 とは異なり，値が小さいほど好ましい．

$$subscore_{not\_closed} = \sum_{\substack{\text{直近1年にopenされ，} \\ \text{closeされていないissue}}} (\text{issueのコメント文字数}) \times \frac{1}{score_{age}(d_{open})} \quad (5)$$

(d_{open}は issue が open されてからの経過日数を表す．)

そして，3 つのサブスコアそれぞれを 0~1 の範囲に収まるように正規化した後，最終的なメンテナンススコアを式(6)のように計算した．ここで，issue の放置度合い($subscore_{not\_closed}$)だけは値が小さいほど好ましいため，マイナスしていることに注意されたい．

$$score_{maint} = N(subscore_{close\_speed}) + N(subscore_{comment}) - N(subscore_{not\_closed}) \quad (6)$$

(ここで，Nは 0~1 の範囲に正規化を行う関数である．)

3.1.5 成熟度

フロントエンドフレームワークは開発当初において，活発に変更が行われる．この変更にはそれまでとの互換性がなくなるような破壊的変更を含む．こうした破壊的な変更が行われている状態では，フレームワーク利用者は安定して開発を行うことができない．そのため，開発がしばらく進み，どれくらい破壊的な変更が少なくなったかを成熟度という指標を用いて表す．成熟度の高いフレームワークはより仕様が安定した，バグの少ないものであり，選定するとフレームワークに関する問題に直面する確率が低くなったり，新しいバージョンへの移行作業が楽になったりするというメリットがある．

図 3 はバージョンの公開履歴の時系列データによって成熟度を分類したものである．図の丸印はバージョンが公開されたイベントを表す．図 3(a)は新しいバージョンが活発に公開されていた時期を過ぎ，仕様が固まってきてバージョンの公開間隔が徐々に空いている．これは安定している状態と言える．図 3(b)は直近で活発にバージョンの公開が行われていて，安定していない状態である．以上 2 つから，「古いバージョンが多いほど成熟している」と考えられそうであるが，例外として図 3(c)を示す．これは，以前にバージョンの公開が活発に行われていたものの，開発が途中で停止している．開発が再開した場合，破壊的な変更が発生する可能性があり，成熟しているとは言い切れない．

(a) 成熟度の高いライブラリのバージョン公開履歴

(b) 成熟度の低いライブラリのバージョン公開履歴 1

(c) 成熟度の低いライブラリのバージョン公開履歴 2

図 3 バージョン公開履歴と成熟度

以上を踏まえ，成熟度は式(6)のように最新バージョンとの公開日の日数差をもとに計算した．具体的には，最新バージョンとの日数の差が大きいほど点数が高くなるようにし，足し上げた．

$$score_{maturity} = \sum_{\text{公開済みのバージョン}} \frac{1}{score_{age}(d_{last\_version})} \quad (6)$$

(ここで，$d_{last_version}$は，あるバージョンと最新バージョンの公開日の日数差である．また，$score_{age}$は日数が大きいほど値が小さくなる関数であることに注意されたい．式(6)ではその逆数をとっているため，日数が大きいほど値が大きくなる．)

ただし，公開済みのバージョンの中には実験的に公開したもの，プレリリース版なども存在する．1.0.0-exp1, 1.0.0-alpha1 などが一例である．そういったバージョンは実験的バージョンとして除外して考えた．なぜなら，フロントエンドフレームワークの開発チームによって，どれくらいの頻度で実験的バージョンを公開するのか方針が異なり，こうした方針の違いは成熟度とは切り離すべきであると考えられる．ここで，多くのフロントエンドフレームワークが採用している Semantic Versioning[a]において，0.1.0 のようにメジャーバージョンが 0 のものは開発段階にあり，安定していないとされているが，本研究ではこうしたバージョンについてもフレームワークの仕様を固めるのに寄与すると考え，除外せずに含めた．

3.1.6 人気度

活発にダウンロードされ，多くのユーザーに注目されているフロントエンドフレームワークは，フレームワーク利用

a https://semver.org/

者のフィードバックが多く得られ，より良い改善が行われる可能性がある．ただし，こうした人気があるフレームワークでも，開発チームのリソース不足でフィードバックを捌き切ることができなかったり，開発が滞っていたりするものもある．そのため，他の指標での評価も重要となる．

Npms[2]での計算方法にならい，本研究ではフレームワークのレジストリである npm のダウンロード数, 及び GitHub でのスター数が多いほど人気度が高いと考えた．また「ダウンロードされる」「スターがつけられる」というイベントは直近のものほど価値が高くなるようにした．

ここで，ダウンロード数については，人気が高まるにつれて指数関数的に増加する傾向がある．まず，人気が高まるにつれて, フレームワークの周辺ライブラリが増加したり，そのフレームワークを導入するプロジェクトが増えたりする．ここで，ライブラリをダウンロードするのは，開発者だけでないことに注意したい．例えば，継続インテグレーションと呼ばれる手法によって自動テストが実行される際にもダウンロードは行われる．つまり，フレームワークの人気が高まるにつれて，こうした開発者以外のダウンロードが加速的に増加する．それらを踏まえて，ダウンロードは対数尺度によって評価する必要がある．以上より，人気度は以下のように計算した．

$$score_{popularity} = \sum_{d=0}^{365} (C_{star}(d) + \log C_{download}(d)) \times score_{age}(d) \quad (7)$$

(ここで，$C_{star}(d)$は d 日前の 24 時間におけるスター数を，$C_{download}(d)$は d 日前の 24 時間におけるダウンロード数を表す．)

3.2 対象となるフロントエンドフレームワーク取得方法

本研究では RealWorld プロジェクト[4]で実装されている例のうち，プログラミング言語に JavaScript が使われているものを取得した[b]. その中で,「同じフレームワークで，バンドルツールだけが異なる」といったパターンを排除した. 結果として, 2021/10 時点で次の 17 個が対象となった.

Angular, Apprun, Aurelia, Dojo, Ember.js, Hyperapp, Imba, Neo, Owl, Preact, React, Riot, San, Solid, Stencil, Svelte, Vue

3.3 選定補助システムの概要

図 4 に選定補助システムの概略図を示す．提案システムは，Cron プログラムによって，定期的に GitHub へリクエストを行って新しいデータを取得し，データベースに保存する．さらに，指標を再計算して，データベースに計算結果を保存する．この計算結果は Backend プログラムを経由してシステムの UI に表示される．

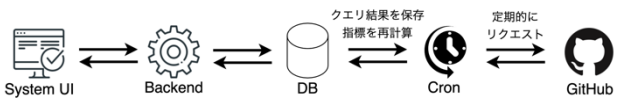

図 4 選定補助システムの概略図

データの取得, 指標の計算にはある程度の時間がかかるため，計算結果をデータベースに保存しておくことで，システム利用者の待ち時間を軽減し，ユーザビリティの向上を図った．将来，新しいフレームワークが登場した際には，システム運営者がデータベースのフレームワークテーブルに「フレームワークの名前・リポジトリ・npm での公開名・RealWorld プロジェクトへのリンク」を保存するだけで良い．これにより，Cron プログラムが実行される際に自動で新しいフレームワークのデータを取得し，指標を計算する設計となっている．

図 5 に提案システムの UI を示す．提案システムは Web アプリケーションとして提供しており[c]，各フレームワークの評価指標値と重み付けしたスコアをテーブルに表示する. テーブルの下には, それぞれの指標に対する重みを 0~1 の範囲で設定する入力欄が配置されている．この重みを設定することで，利用者は「どの指標をどれだけ重要視するか」を指定することができる．テーブルの右端にある「重み付けスコア」のカラムはユーザーが入力した重みに基づいて, 各フレームワークの値がリアルタイムに計算される．また，それぞれのカラムをクリックすることで，昇順および降順でのソートが可能である.「類似度の比較を行うフレームワーク」セクションは，フレームワーク同士の API の類似度の指標に関する機能である．このセクションに配置された選択欄で比較対象となるフレームワークをいくつか指定することで，表にカラムが追加され，その下には重みの入力欄が追加される．比較対象としては以下のようなユースケースを想定している．

- 習得済みのフレームワークを指定し，なるべく学習コストが少ないものを探す
- すでに導入しているフレームワークを指定し，なるべく移行コストが低いものを探す

図 6 は比較対象として react というフレームワークを指定した様子である．テーブルに「react との類似度」というカラムが追加され, 各フレームワークと react との類似度が表示されるようになる.「重みのプリセット」セクションに配置されたボタンをクリックすると，事前に用意した重みが自動で設定される．例えば「初心者向け」というボタンをクリックすると，情報共有の活発さや人気度といった指標の重みを大きく設定する．これにより，重みを細かく指定するという手間を省ける．図 7 は「安定性重視」のプリセットを選択して重みの値を自動設定し，重み付けスコアのカラムを降順でソートした様子である．重み付けスコアが高く，上に表示されているものほどユーザーの要望に合う可能性が高い．そのため，ユーザーは上位いくつかについて，フレームワークの仕様が開発要件にマッチするかなど

b https://codebase.show/projects/realworld?language=javascript
c https://tech-choice.web.app/ にて提供している

の定性的な判断を行えば良い．その際，フレームワーク名をクリックすることで，そのフレームワークの公式ドキュメントへアクセスすることができる．

図 5 提案システムの UI

図 6 類似度の比較対象を選択した様子

図 7 重み付けスコアで並び替えた様子

システムの開発環境については以下の通りである．

- 言語：TypeScript
- フロントエンドフレームワーク：Svelte
- バックエンドフレームワーク：Fastify
- ORM：Prisma

4. 選定補助システムの評価方法

提案システムはフロントエンド開発経験のある開発者数名に使用してもらい，半構造化インタビューを行うことで評価を行う．システムを使用するシナリオとしては，以下を想定している．

> RealWorld の仕様[d]で定められたブログアプリを新規開発するときに，どのフレームワークを使うかを開発者である被験者が選定する．その際，提案システムを補助として用いる．

使用時には，発話思考法[8]に基づき，被験者には考えていることをなるべく声に出してもらう．

システム操作後には，半構造化インタビューを行う．事前に用意した質問に加え，回答にあわせてより深堀りして質問を行う．事前に用意する質問内容としては，稲垣ら[9]のソフトウェアシステム評価を拡張し，またソフトウェア品質の国際規格である SQuaRE[10]の品質特性を参考に，大きく分けて有効性・使用性・実用性・データの正確性の観点から以下のように考えた．

有効性

・フレームワーク同士を比較することができたか

・6 つの指標はフレームワークを評価するのに効果的か

使用性

・システムの使い方は理解しやすかったか

・重みの入力，プリセットボタンなどの要素は使いやすかったか

実用性

・実際の選定の際に使えそうか

データの正確性

・各フレームワークの評価値は適切か

・重みのプリセットの値は適切か

5. おわりに

本研究では，フロントエンドフレームワークの選定における困難を軽減するために，先行研究によって示されたいくつかの指標のうち，定量的なデータから計算可能なものについて，データを収集・計算し開発者に提示する選定補助システムを構築した．

今後の展望として，被験者を集めて評価実験を行うこと，各指標の計算方法をさらに検討すること，提示する指標の種類を増やすことなどが挙げられる．

d https://gothinkster.github.io/realworld/docs/specs/frontend-specs/templates

参考文献

[1] E. Larios Vargas, M. Aniche, C. Treude, M.Bruntink, and G. Gousios. Selecting Third-Party Libraries: The Practitioners' Perspective. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 245-256.

[2] M. Kaluza, K. Troskot, B. Vukelic. Comparison of front-end frameworks for web applications development. *Zbornik Veleucilista u Rijeci*, 2018, vol. 6, no.1, pp. 261-282.

[3] A. Abdellatif and Y. Zeng and M. Elshafei and E. Shihab and W. Shang. Simplifying the Search of npm Packages. *Information and Software Technology,* 2020, vol. 126.

[4] F. Bissyande, Tegawende, T. Ferdian, L. David, J. Lingxiao and R.Laurent. Orion: a software project search engine with integrated diverse software artifacts. *18th international conference on engineering of complex computer systems,* 2013, pp. 242-245.

[5] Yiyi Sun. A Real-World SPA. In: Practical Application Development with AppRun. *Professional and Applied Computing*, IEEE, 2019, pp 219-246.

[6] A. Gizas, S. Christodoulou and T. Papatheodorou. Comparative Evaluation of Javascript Frameworks. *Proceedings of the 21st International Conference on World Wide Web, Association for Computing Machinery,* 2012, pp.513-514.

[7] D. Graziotin and P. Abrahamsson. Making sense out of a jungle of JavaScript frameworks. International Conference on Product Focused Software Process Improvement, 2013, pp. 334-337.

[8] K.A. Ericsson and H.A. Simon. Protocol analysis: Verbal reports as data. *The MIT Press*, 1984.

[9] 稲垣成哲，舟生日出男，山口悦司．再構成型コンセプトマップ作成ソフトウェアの開発と評価．科学教育研究, 2001, vol.25, no.5, pp. 304-315

[10] 込山俊，博東基衛．システムおよびソフトウェア品質の国際的な基準の確立―日本主導の国際標準化への取組み―．デジタルプラクティス, 2019, vol. 10, no. 1, pp. 62-73

付録

提案システムのソースコードは次のリポジトリで公開している
https://github.com/KoichiKiyokawa/tech-choice