
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ


# AUTOMATION OF SOFTWARE CORRECTION RELEASE PROCESS FOR OPENSCAPE 4000

AUTOMATIZACE PROCESU VYDÁVÁNÍ SOFTWAROVÝCH OPRAV PRO OPENSCAPE 4000

MASTER'S THESIS
DIPLOMOVÁ PRÁCE


AUTHOR
AUTOR PRÁCE

Bc. SERGII KHUNOVYCH

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. RADEK KOČÍ, Ph.D.



BRNO 2017

**Brno University of Technology - Faculty of Information Technology**

Department of Intelligent Systems | Academic year 2016/2017

# Master's Thesis Specification

For: **Khunovych Sergii, Bc.**

Branch of study: Intelligent Systems

Title: **Automation of Software Correction Release Process for OpenScape 4000**

Category: Software Engineering

Instructions for project work:

1. Analyse current process of software correction release (hotfixes) for the OpenScape 4000 process and discuss the suggested process changes.
2. Design a suitable tool, which will automate the process of hotfixes or at least support the necessary manual steps effectively. The tool should fulfill in particular the following requirements. Shorten the time to release the software correction, minimize the necessity to enter data by hand, start hotfix production automatically, and provide reports about hotfixes and planned corrections.
3. Implement the tool and integrate it with the existing project management tools (JIRA, Confluence) and Continuous Integration tool (Jenkins).
4. Provide technical documentation for the tool and teach the staff how to use the tool in form of a training.
5. Evaluate the benefits of the tool and its implementation.

Basic references:


- J. F. Smart, Jenkins: The Definitive Guide, O'Reilly Media, 2011
- Extend Jenkins, https://wiki.jenkins-ci.org/display/JENKINS/Extend+Jenkins, Oct. 2016
- Atlassian Developers, Jira aand Confluence. https://developer.atlassian.com /index.html, dostupné říjen 2016.


Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Kočí Radek, Ing., Ph.D.**, DITS FIT BUT

Beginning of work: November 1, 2016

Date of delivery: May 24, 2017


L.S.

Petr Hanáček

*Associate Professor and Head of Department*

# Abstract

This thesis describes the problematic of software correction release in OpenScape 4000 ecosystem, analyze actual way of collecting correction information and producing new Hotfixes. Together with that were discussed ways how to eliminate most manual steps and design new tool, with better UX, for needed processes. The result is a new implemented web based application, which called Hotfix Portal, with authentication and different views for different roles of users.



# Abstrakt

Tato práce popisuje problematiku vydávání softwarových oprav v ekosystému OpenScape 4000, analyzuje aktuální způsob sběru korekčních informací a vytváření nových Hotfixu. Společně s tím byly diskutovány způsoby, jak eliminovat většinu manuálních kroků a navrhnout nový nástroj s lepším UX pro potřebné procesy. Výsledkem je nově implementovaná webová aplikace, která se nazývá Hotfix Portal s autentizace a různými pohledy pro různé role uživatelů.



# Keywords

Release management, Product management, Hotfix, OpenScape4000, Jira, Meteor, MongoDB, ReactJS, Automation, UX.



# Klíčová slova

Řizení vydání, Řizení  productů, Hotfix, OpenScape4000, Jira, Meteor, MongoDB, ReactJS, Automatizace, UX.



# Citation

Sergii Khunovych: Automation of Software Correction Release Process for OpenScape 4000, Brno, Faculty of Information Technology, Brno University of Technology, 2017

# Automation of Software Correction Release process for OpenScape 4000

## Declaration

I declare that I have worked out this diploma thesis independently under supervision of Ing. Radek Kočí, Ph. D. Further information was provided by Ing. Peter Jelen from the company IXPERTA s.r.o. I have listed all literature and publications from which information was received.

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph. D. Další informace mi poskytl pan Ing. Peter Jelen, ze společností IXPERTA s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Sergii Khunovych
24.05.2017


## Acknowledgment

I would like to thank the supervisor of this work Ing. Radek Kočí, Ph. D. and Ing. Peter Jelen for their help in creating this work.

## Poděkování

Chtěl bych poděkovat vedoucímu této práce panu Ing. Radkovi Kočímu, Ph. D. a panu Ing. Petrovi Jelenovi za jich pomoc při vytváření této práce.

# Contents

# 1 Introduction

OpenScape 4000 (OS4K) is hybrid TDM/IP-PBX platform for enterprise voice communication. As for all enterprise systems it needs get all parts sustainable for long period of time and fixable in short period of time. Because of this the release process are divided into Major, Minor and Fix releases and for the last one Hotfix releases. Also are two types of product with synchronized versioning and releasing – OpenScape 4000 Assistant and OpenScape 4000 Manager. Assistant – is dedicated software for service management one OS4K system. Unlike an Assistant, the Manager – is a master for slave Assistants and it is possible to manage all system from one place. In OpenScape 4000 ecosystem always exists few sustainable releases for which ones we need to produce Hotfixes. Right now exist 2 sustainable versions (V7 R2.24.0 and V8 R0.14.0).

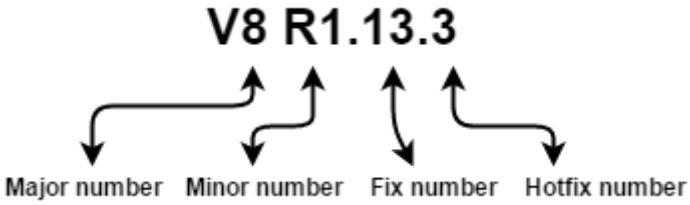



*Image №1 – Release numbers in OpenScape 4000*

## 1.1 Motivation

Given the fact that Hotfixes released only for sustainable Fix releases, each of them are the set of rpms or how they called components. Producing and releasing of Hotfixes for enterprise system – is hard and volume teamwork of developers, testers and product coordinators, because of needs to collect all fixed issues, producing only changed components of the system and testing of produced Hotfix. In OpenScape 4000, Hotfix – is tarball of accumulated changed components starting from Hotfix 1. So, for example, Hotfix 3 will contain all changed components since Hotfix 1 for current Fix release. Common Hotfix releasing workflow shown on image №2: “Common Hotfix releasing workflow”.

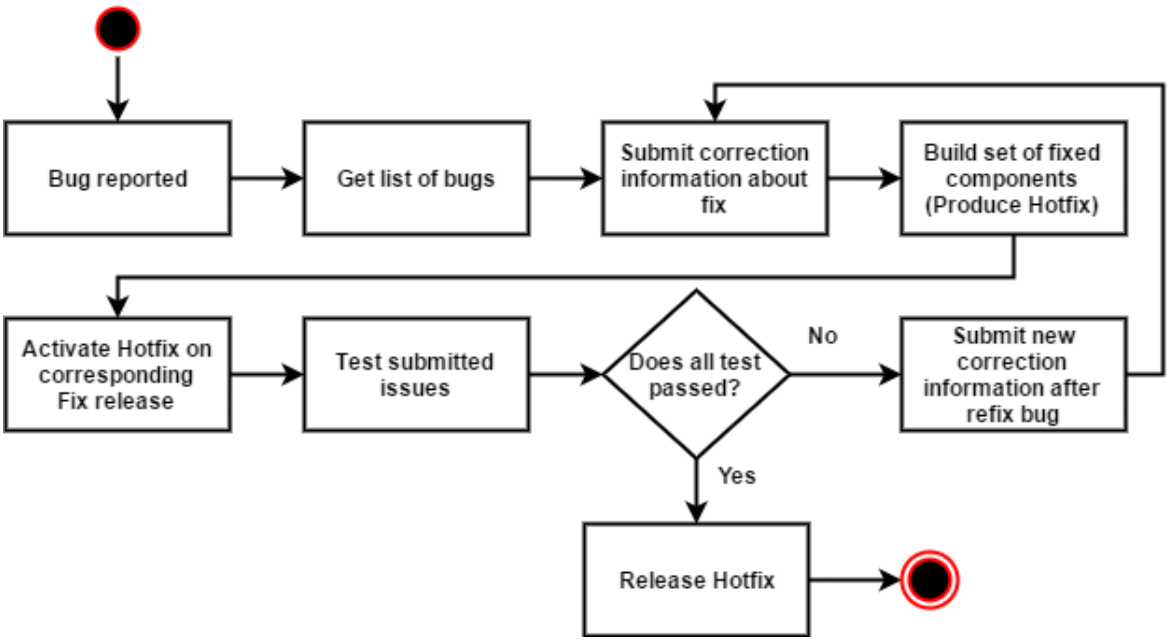



*Image №2 – Common Hotfix releasing workflow*

Also the current solution of Hotfix production and release called “Hotfix portal” requires too many manual steps from users, has security issues, because of old operating system, does not have integration with project management tools, as Jira, and has bad UX.

## 1.2 Goal and structure of work

This thesis contains two main goals. The first one is to analyze current workflow of Hotfix production and release, highlight weak steps and bad UX. Based on it design new software tool, which will automate or support manual steps effectively. The second goal is to implement designed tool with better UX and integrate it with existing management tool, as Jira.

To achieve the set goals work was organized as follow. The first part describes current Hotfix workflow separated by user views – developer view, tester view and product coordinator view. Analyzed and highlighted weak parts of currents solution in this section. The second part contains proposals of business process with possibilities to improve current release management. The third part is devoted to description of used and studied technologies, such as Meteor, MongoDB and ReactJS. The fourth part contains detailed design and implementation of new application for release management process.

The final part of thesis evaluates the benefits of the tool and describes further improvements and next steps after integrating it into OpenScape 4000 production and release ecosystem.

# 2 Problems and analysis

As was said in introduction current Hotfix releasing tool called Hotfix portal has many different problems. Such as many manual steps from all participants of releasing process, bad user experience design, missing integration with project management tool Jira and has security issues. A number of presentations of work with the current tool was conducted to identify and analyze the main problems and limitations. After it, was decided to divide all the work into three types of activity with this tool – from the point of view of developer, tester and product coordinator.

## 2.1 Developer view

Every developer, after the issue was fixed, have to submit correction information (CIT). For this proposes is used Hotfix portal page showed on image №3: “Submitting correction information in current Hotfix portal”.

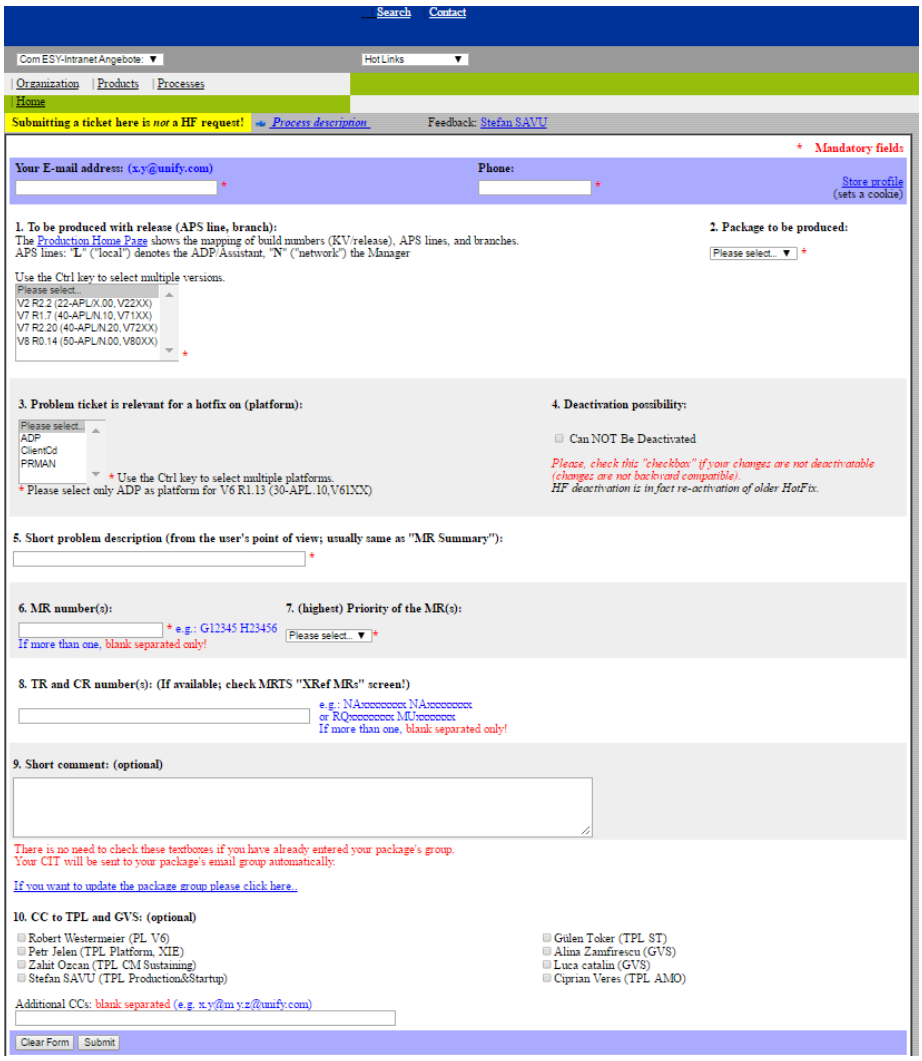



*Image №3 – Submitting correction information in current Hotfix portal*

As seen from image №3: “Submitting correction information in current Hotfix portal”, each developer has to perform many manual steps and together with that requires knowledge of archaic names of products and versions. This is the only page accessible for them as well and this leads to impossible view all submitted correction information and current statuses of Hotfixes.

To summarize, the main problems of developer workflow in Hotfix Portal are:

- Every submitting correction information requires many common manual inputs, such as email address, telephone number and additional email list for notifications.
- Requires knowing of archaic platform and version names.
- Does not provide list of yours submitted correction information.
- Does not provide list of Hotfixes.
- Has old-fashioned and unclear UX design.

## 2.2 Tester view

In Hotfix produce and release management testers has a task to test all issues included into current Hotfix. The issue must have corresponding correction information ticket (CIT) and this CIT must be included to current Hotfix. When product coordinator take the necessary steps, such as building and starting up Hotfix on live system, all participants will get an notification email and hyperlink to the webpage where everyone, not only testers, can write test result in format PASS or FAIL and short comment.

For each Hotfix has new link with testing table and from image №4: “Webpage for writing Hotfix test results” seen, the header of page contents release number, in example HF 10 for Assistant Fix release V7 R2.20. Then goes commit changes button, description how to write test results and the table with issues to test itself. First column contains archaic and hard-remembered CIT number of issue to test, second – short description. After that goes issue number in obsolete issue tracker system. The column called “Produce” contains very important component name, where fix was provided.

**Release: V7 R02.20 HF: 10 (Delta) Platform: Assistant**

Commit changes

If you write PASS or FAIL (literally) in the test result field, it will be colored accordingly.

| FLID | Summary | MRnr | Prio | Originator | TRnr | Prio | Customer | Produce | Tester:PASS\|FAIL(Reason) |
|---|---|---|---|---|---|---|---|---|---|
| UAS0025447 | NA15291166-LMT daily transfer hangs | I57326 | 2 | | | | | ASmpcid | |
| UAS0025466 | Sesion max count = 0 possible | I57390 | 3 | | | | | AShttp | |
| UAS0025489 | TSDM:Support of SLMU boards | I57513 | 2 | | | | | AStsdm | |
| UAS0025491 | NA15300218-engr user can not access Platform | I57355 | 2 | | NA15300218 | | | ASsecm | |
| UAS0025505 | NA15293475-Call to External Template generate empty report | I57321 | 3 | | NA15293475 | | | ASpm | |
| UAS0025516 | AScm: Update of online help for Assistant. | I57562 | 3 | | | | | AScm | |
| UAS0025521 | HF9 V7 R2 20_Tomcat's memory is already corrupted | I57568 | 2 | | | | | AShttp | |
| UAS0025527 | Scrollbar vertical on LAP2 page not needed | I57576 | 4 | | | | | ASlap2 | |
| UAS0025535 | COL Java Exception Assistant V7R2.20 HF9 | I57495 | 3 | | | | | ASsysm | |
| UAS0025538 | I57313-NA15279343 Uknown value for system number in LAP2 | I57313 | 3 | | NA15279343 | | | AScomedit | |
| UAS0025541 | NA15319767 there is an unexpected long delay of approx 6 sec | I57610 | 2 | | NA15319767 | | | ASxie | |

If you need a list of the MRs only:
click here to expand

*Image №4 – Webpage for writing Hotfix test results*

If summarize, as in developer view, the main problems of tester workflow are:

- Test Hotfix pages are available for everyone via link in notification mail.
- All test Hotfix pages are separated from each other.
- UX design for submitting result.
- Lack of information about issues in test table.
- Does not provide information about tester of CIT.
- Does not provide list of Hotfixes.
- Provide possibility to write test result for anyone.

## 2.3 Product coordinator view

The main role and work in all process of releasing Hotfixes have product coordinator. The set of his responsibilities are from defining new product fix versions to build needed components via special script on build server.

During analyzing product coordinator workflow were discovered lack of automation in his work, plenty operations and steps were handled manual. Together with that were found many archaic and historical parameters in producing steps. As well as in previous views webpages had UX design problems.

First of all, product coordinator has to define new sustainable product fix version. For this case in current implementation does not provided any graphical interface. All changes handled manually directly into database tables, which used by current tool. When changes added developers can submit correction information for fixed issues and product coordinator can define new Hotfix for created version.

When it is time to release a new Hotfix first step is necessary to proceed – define. For this purpose product coordinator use Hotfix portal page shown on image №5: “Parts of define a new Hotfix page” and as seen on it needed to select version, release number in some archaic format and manually enter new HF number.

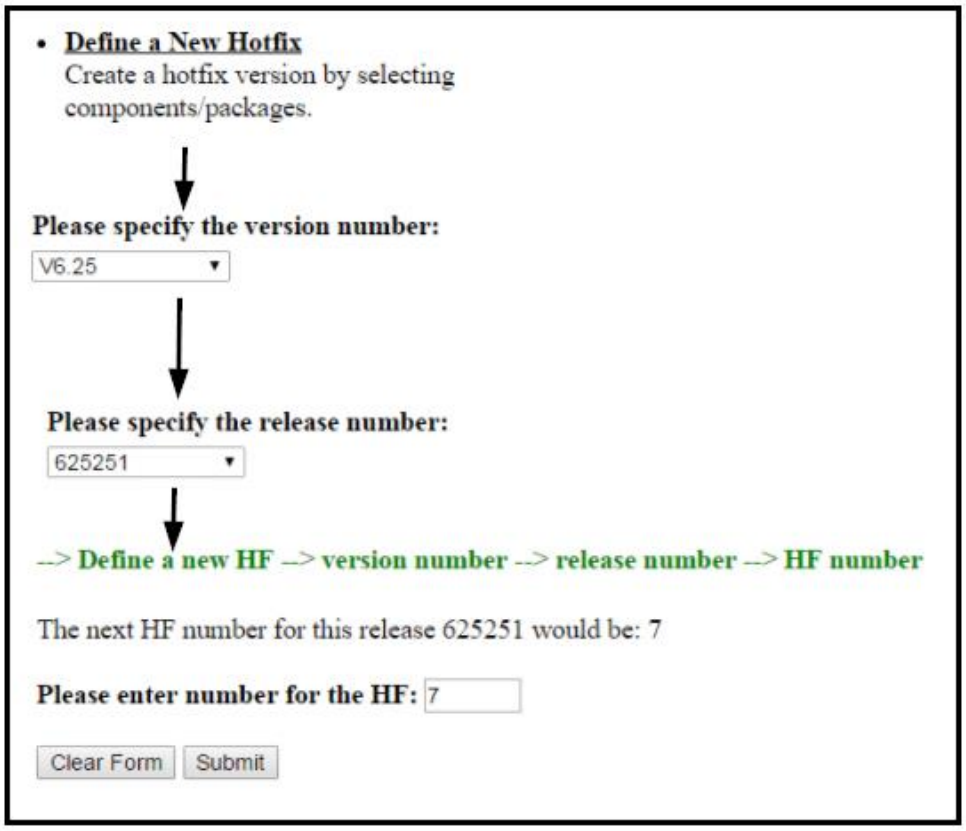
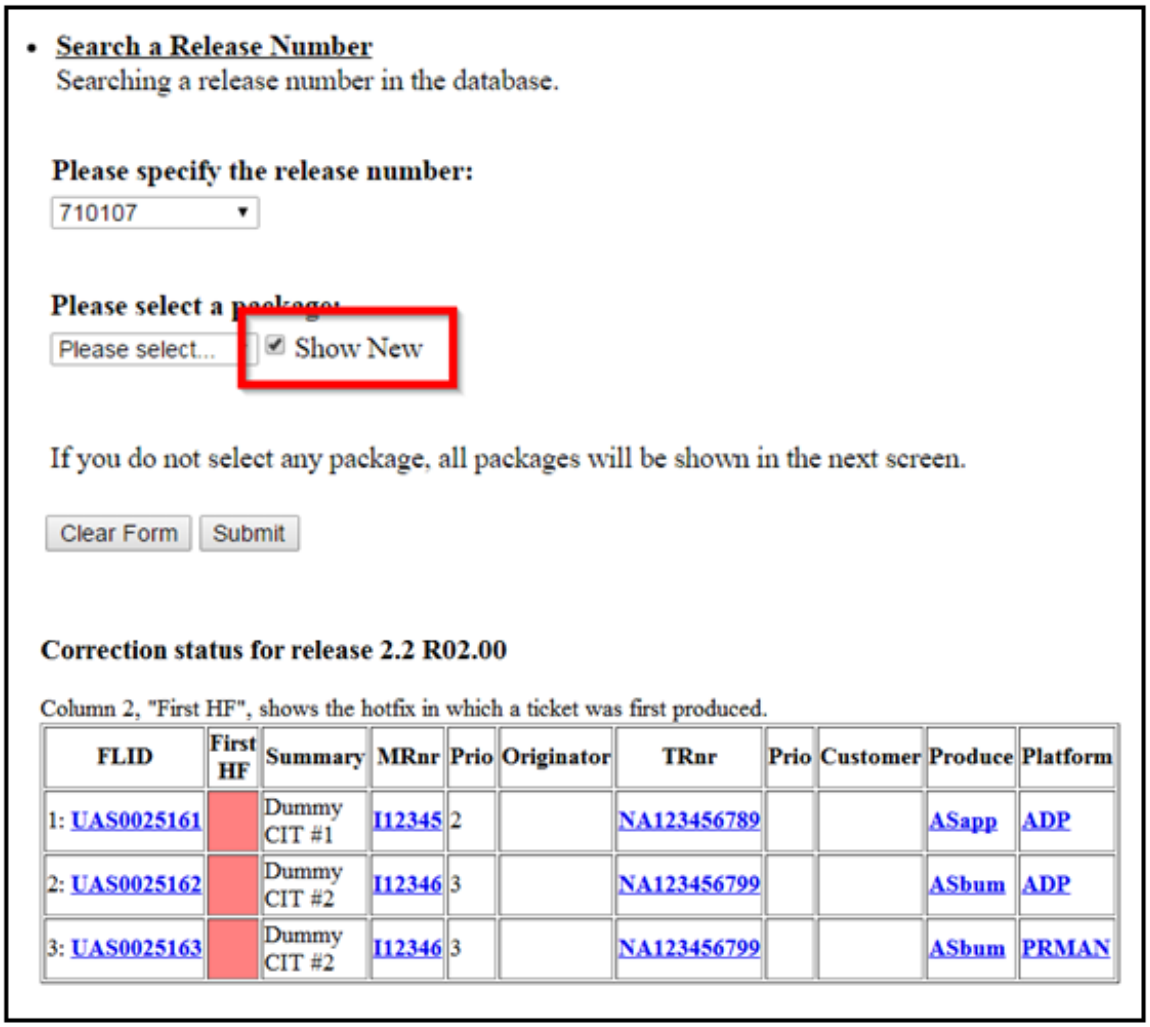



*Image №5 – Parts of define a new Hotfix page*

The next step in releasing – request. And again product coordinator uses Hotfix portal. On image №6: “Parts of request Hotfix page” seen workflow for requesting: select release number, for all packages does not select any and check “Show new” and after submit button clicked an table with correction information tickets generated to be included into current HF. This list of CITs are manually copied to xls file and send to all participants via notification mail.



- **Search a Release Number**
  Searching a release number in the database.

**Please specify the release number:**

710107

**Please select a package:**

Please select... Show New

If you do not select any package, all packages will be shown in the next screen.

Clear Form Submit

**Correction status for release 2.2 R02.00**

Column 2, "First HF", shows the hotfix in which a ticket was first produced.

| FLID | First HF | Summary | MRnr | Prio | Originator | TRnr | Prio | Customer | Produce | Platform |
|---|---|---|---|---|---|---|---|---|---|---|
| 1: UAS0025161 | | Dummy CIT #1 | I12345 | 2 | | NA123456789 | | | ASapp | ADP |
| 2: UAS0025162 | | Dummy CIT #2 | I12346 | 3 | | NA123456799 | | | ASbum | ADP |
| 3: UAS0025163 | | Dummy CIT #2 | I12346 | 3 | | NA123456799 | | | ASbum | PRMAN |

*Image №6 – Parts of request Hotfix page*

After this product coordinator start compilation of changed components via ssh connection to build server. The result Hotfix tarball are copied into special intranet download location. Together with it product coordinator initiate next step of releasing – produced. This means that for all participants will send notification mail with header “STEP 3”, which corresponds to produced status, and subject in following format – “Hotfix: ADP-MGR-22R02.00.001-ASapp-ASbum Several Corrections”, ADP and MGR – are archaic names for Assistant and Manager products , 22R02.00.001 – means V2 R2.01 and then follows names of components produced with HF. Also, needed to set issues in product management tool Jira to solved status. This is done manual.

Next step is – start up Hotfix on live system. If startup was successful product coordinator initiate next notification mail with link for downloading HF and link to page were testers can write test results.

If some issue did not passed the test reproduction initiate needed. In this case after new fix of it developer need to submit new correction information and product coordinator have to add it to corresponding Hotfix via steps shown on image №7: “Parts of adding CIT for reproduction Hotfix portal page”. And as seen on this image to add new CIT product coordinator need make following manual steps: enter HF number, select component where fix are and finally add it.

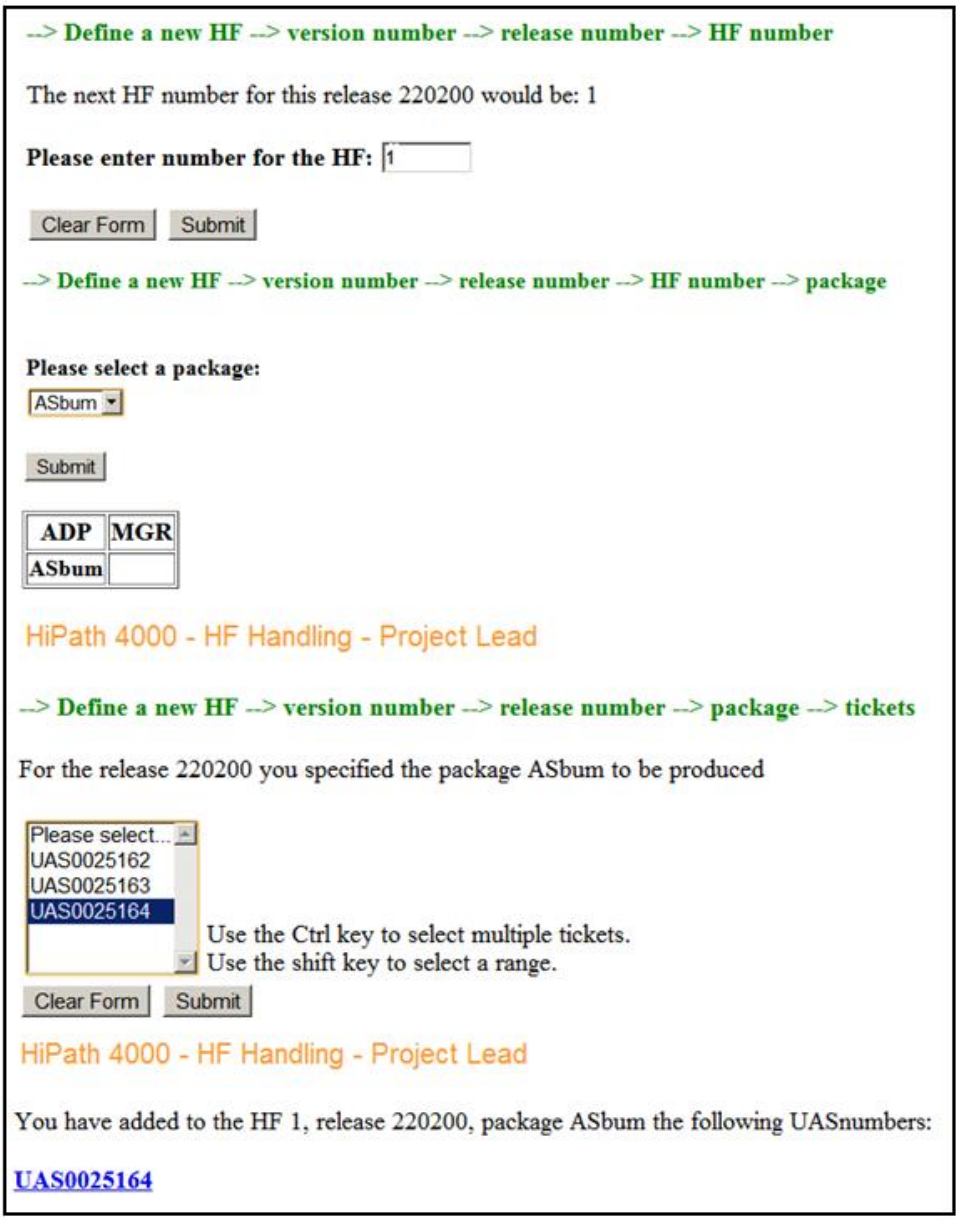



*Image №7 – Parts of adding CIT for reproduction Hotfix portal page*

After all this steps, also needed to submit reproduction request via specific page on Hotfix portal, where again needed manual specify number, check for which product this applies (Assistant or Manager) and submit decision. Notification mail will be send to all participants with subject and header

at following format: “Hotfix: ADP-22R02.00.001-ASapp-ASbum”, “===[ADP REPRODUCTION REQUESTED]=== ”, what means that for Assistant V2 R2.0.1 reproduction was requested.

Next steps will be the same as after build HF – produced with mail notification and the same for startup.

After all issues passed tests the next step are initiated – tested. This means that the final list of issues included into HF need to create. As seen from image №8: “Creating list of issues added to HF page” this leads to many manual steps, such as: selecting release number, product name, HF number, specifying from to list of CITs and choosing show only delta, because of said before that each HF includes accumulate fixes since first one. Also manual deleting of failed and duplicate CITs from this list needed.


*Image №8 – Creating list of issues added to HF page*

And after all of above, the last step is released. For change HF status to that product coordinator again, same as in previous steps, have to do several manual operations (choose release, HF number and so on) and submit his decision. After that mail notification sends to all participants with info about released HF, links to Hotfix Portal, where list of included issues can be found and, of course, link for download it.

And now, when analyze are finished. To summarize, the main problems of product coordinator workflow in Hotfix Portal are:

- Graphical interface for adding new sustainable product versions.
- Manual operations while defining new Hotfix.
- UX design and manual sending issue list while requesting Hotfix.
- Manual generating list of components to be produced on build machine.
- Archaic mail notification format.
- Manual setting issues in Jira to solved status after HF produced.
- Many same manual operations in reproduction needed step.
- And again manual operations in tested and released steps.

# 3 Business process proposal

An important part of this work was based on studied and analyzed current workflow propose the ways how to improve and automate Hotfix release management. To reduce time needed to pass the all bureaucracy management steps, to facilitate work of all participants of this process and eliminate manual steps to the minimum. Given the fact, that in release process several user roles take part we need to develop and improve interaction between them considering current problems. Also, we need to consider all management and product eliminations during proposal of future business process.

If generalize, Hotfix release process covers the following sequence of steps. In the first step management defining date and the list of issues, to be fixed, for the future Hotfix release. Based on that, developers start investigate into the issues problems and fix them. After, product coordinator collect fixed issues and start producing new Hotfix and inform about result of it. Based on this, testers begin to check included fixes and if everything works fine allow to provide Hotfix to customers. If not, the fix, failed during tests, need to be refix and whereupon reproduced. Also it necessary to take into account that all mentioned above actions require to notifications to the all participants in Hotfix release process.

In view of the above, the following solutions were analyzed and weighted their pros and cons – implementation of Jira plugin, updating current application, implementation of new web based application.

## 3.1 Jira plugin

One of the possibility to improve release management process is to implement Jira plugin solution. This implementation would eliminate needs in integration issue tracking system and release management system. Together with it, there will be no need to solve notifications of all participants about changes or progress in Hotfix producing, because of accessibility of information for all.

For definition lists of issues to be included into specified Hotfix would be used labeling system of Jira. At the time when all issues are solved by developers – the automatic build of the Hotfix begins. After that product coordinator can take produced tarball and started up it in system for testing. In that moment, testers should start their work and changed status of corresponding issues to pass or failed. Based on this, in the moment when all tests are succeed – the automatic release of the Hotfix are made.

Unfortunately, due to the fact that Jira is a paid commercial product and any changes or customer specific improvements are also paid – this implementation is not suitable for solving our goals in current period of time.

Summarizing all of the above, this solution has the following pros and cons:

- Pros: one tool for all processes, high level of automation, notification possibility by default.
- Cons: paid changes in commercial product.

## 3.2 Updating current solution

One more variant to solve pressing problems is to update, currently used and described at chapter 2:“Problems and analysis”, application. Given the fact, that it was implemented in the mid-nineties and after that no further improvements were made, it means that all parts of solution should be upgraded from out of date operation system, where application are running, to older version of programming language (PHP) and database (MySQL).

Also, need to consider to update UX and graphical interface, which leads to the need for changes and addition on the server side. One more thing to be solved is integration with the Jira product management tool. In fact, all parts of application would be updated, rewritten or modified to meet the goals of this work.

So, if summarize, the question arises – does the benefits of such a solution cover the invested time and effort? And also this solution will be deprived of the choice of the programming language, the used platform and database.

## 3.3 New web based application

The last one way and the chosen one to improve Hotfix release management process – is to develop new real-time web based application and integrate it with Jira product management tool, which will substitute current solution.

This implementation would provide for all participants minimize needed manual steps in their daily work. Also, it will have better UX design and graphical interface. Manual providing of user specific information, such as email address, name and telephone number, would be eliminated by introducing authentication. The graphical interface would be divided by user roles workflow to developer, tester and product coordinator view. Each view will be interacting only with allowed set of data.

For developer view should be provided form for submitting correction information for fixed issues with easier way to do it. For this purposes, will be integrated Jira API connector with possibility to retrieve needed data corresponding specified issue. Also, the way to choose relevant version information, component list and notification mail subscribers should be implemented in multi selection way. As one more of improvement would be possibility to view list of previously submitted correction information (CITs) and list of Hotfixes.

The main and most important improvement for tester view would be ability to have all issues for test on one place. Also, setting result of the test would be by select it by click. The same as in developer view would be provided read only overview about all Hotfixes.

For product coordinator view should be provided new possibilities to define new supported versions and new Hotfix, to add list of correction information for specified Hotfix and changing statuses of Hotfixes. It will be achieved by minimizing manual steps and automation, for instance if some test failed – corresponding Hotfix will automatic change the status to "Reproduction requested". The list of components for producing will be provided by click. Mail notifications about changing statuses would be more informative.

Also, application should allow to initially configure needed values, such as users with attached roles, groups of notification mails subscribers, list of components and of course credentials for Jira API.

Based on foregoing, to implement such an application, it was decided to use the following stack of technologies: JavaScript programming language, MongoDB, as a data store, Meteor, as platform for developing, and ReactJS for creating graphical interface. The above technologies are described in the following chapter 4: "Used technologies".

# 4 Used technologies

In this part of thesis will be provided information about technologies used in implementation of new Hotfix Portal tool.

## 4.1 Meteor

Meteor is a full-stack JavaScript platform, based on Node JS, for developing modern web and mobile applications [1]. If summarize, the main advantages are:

- Possibility to develop in one language, JavaScript, for server-side and client-side (web, browser, mobile devise).
- Using data on the wire, means that the server sends data and the client processes it.
- Own ecosystem of packages called Atmosphere JS.
- Automatically propagating data changes to client via Distributed Data Protocol, without requiring any synchronization code.

In the moment when Meteor was installed, project could be created with following command: `meteor create myapp`, this would create directory called by name of application and contains initial directory hierarchy and files that needs by project. Going to the folder, need to install Meteor's build-in npm dependencies using command `meteor npm install`. After this, run new application on localhost (`meteor start`). Server will run on http://localhost:3000/ and Meteor support hot reload, which means that when some changes did, it will restart server and update GUI according to the changes. To add needed Meteor or npm packages to application use `meteor add <package_name>` and `meteor npm intall <package_name>` respectively.

For using ReactJS as view library adding npm packages needed, such as react and react-dom. To provide data from DB or server-side to client-side ReactJS components needed to use Meteor package `react-meteor-data`, which allows to create data-container to communicate with backend on one side and provide this data to frontend on another side [2]. Prerequisite to use this package is installed npm package `react-addons-pure-render-mixin`. Detailed description of using and developing ReactJS components are in 3.3 "ReactJS" part of this thesis.

Meteor gives possibility to build multi-user application with no need to develop own solution. For this purposes just needed to add new packages `accounts-ui` and `accounts-password`. Given the fact, that this packages provide UI component in internal templating technology, called Blaze, to use in a React component needs to wrap it. In data-container, described above, exist possibility to check if user is logged in and get information about him. For this purposes just needed to call internal method `Meteor.user()` and response will contain info about current user. This trick also used on server-side

to validate if calling secure parts of application does under logged in user. If not, throw not authorized error.

To extend accounting management could be used `alanning:roles` Meteor package, which lets to attach roles with different permissions to a user. After roles are defined and attached to control if user is in some role the following method used: `Roles.userIsInRole(userId, nameOfRole)`[3].

To provide secure client-server interaction in Meteor needed to define methods with allowed operations by roles. Methods - are Meteor's remote procedure call (RPC) system, used to save user input events and data that come from the client [4]. On image №9:"Define Meteor's method" shown an example of defining method for inserting, let's say, new task to Mongo collection with checking userId in and role.

```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';
import { check } from 'meteor/check';
import { Roles } from 'meteor/alanning:roles';

export const Tasks = new Mongo.Collection('tasks');

Meteor.methods({
  'task.insert'(txt)
  {
    check(txt, String);

    if(! Meteor.userId) {
      throw new Meteor.Error('not-authorized');
    }
    if(! Roles.userIsInRole(this.userId, ['role'])) {
      throw new Meteor.Error('not-permitted');
    }
    Tasks.insert({
      text: txt,
      createdAt: new Date(),
      owner: Meteor.userId(),
      username: Meteor.user().username
    });
  }
});
```

*Image №9 – Define Meteor's method*

Also Meteor gives ability to send emails, for this purposes it provides package `email`. Before actually send an email, needed to provide access to email provider, specifically needs to sign up for an SMTP service that can delegate our email for sending [5]. In order to send email in application, needed to set `MAIL_URL` environment variable. After this is possible to send an email with calling `Email.send()` method with following properties:

- Required parameters
    - to – email address or an array of string with email addresses.

- from – the email address email being sent from.
- subject – the subject field added to email.
- text – plain text of email.

- Additional parameters
  - cc – array of string with email addresses to copy email to.
  - replyTo – string or array of strings with email addresses that can be set as the reply to address for the recipient.
  - html – string, which contains HTML that can be rendered in the body of email.

## 4.2 MongoDB

MongoDB – is open-source cross-platform document oriented database that provides, high performance, high availability, and easy scalability. It works on concept of collection and document [6]. In MongoDB world, database is physical container for collections, each gets its own set of files on file system. Collection – is the equivalent of an SQL based table, it is a set of MongoDB documents. They do not have schema and documents inside it can have different fields, but typically, all documents in a collection are of related purpose. Document – is analogue to JSON object of key-value pairs, but stored in the database in a more type-rich format called binary JSON or BSON.

| Relational DB | MongoDB |
|---|---|
| Database | Database |
| Table | Collection |
| Row | Document |
| Index | Index |
| JOIN | Reference or embedded document |

*Table №1 – Relationship of relational DB with MongoDB [7]*

MongoDB provides a unique id for every document. This id is 12 bytes hexadecimal number, which guarantee uniqueness of every document. First 4 bytes are for the current timestamp, next 3 bytes are machine id, next 2 bytes for process id of MongoDB server and last 3 bytes are simple incremental value.

MongoDB provides various official supported drivers in following programing languages: C, C++, C#, Java, JavaScript, Python, etc. Given the fact that in this work used JavaScript platform Meteor with native supporting of MongoDB, let's describe possibilities and main functions of JavaScript MeteorDB driver.

To store and manipulate with some data we need to create a collection and keeps a reference to it with possibility to use it in needed place. To do this in JavaScript just write the following statement: `export const MyNewCollection = new Mongo.Collection('myNewCollection')`. This statement

returns a reference to object with methods to insert documents into collection, update their properties, remove them and to find the documents in the current collection that corresponds search criteria.

To retrieve the documents from a specified collection the following function used: `MyNewCollection.find(object)`, where object – is a search criteria JSON object. The function will return an array of JSON objects corresponds to search terms or if we leave function empty, it will retrieve all of the documents from the collection. To retrieve the date in a human-readable format use the same function, but attach a `fetch` function to the end of it: `MyNewCollection.find({}).fetch()`. Also if needed to return only one document from a collection should use function `MyNewCollection.findOne()`, with specified search criteria.

To store a new document into collection used function `MyNewCollection.insert(object)`, where object – is a JSON object to be inserted.

To update existing document in collection used function `MyNewCollection.update(_id, {$set: })`, where the first argument is id of document to be updated, and the second one passed $set operator and this allows to modify the value of a field or multiple fields.

To remove document from collection used function `MyNewCollection.remove(_id)`. If instead of id place an empty object ({}), then all documents from current collection would be removed.

MongoDB also has his own console interface to query and update data as well as perform administrative operations. To start Meteor MongoDB shell call command `meteor mongo` in root folder of application. In table №2: “MongoDB shell commands” describes main commands of this interface.

| Command | Description |
|---|---|
| use dbName | Switch to database called dbName |
| show collections | Check created collections in current database |
| db.createCollection(“newCollection”) | Creates new collection |
| db.newCollection.insert({“field”: “value”}) | Insert new document into collection |
| db.newCollection.find() | Retrieves all documents in collection |
| db.newCollection.drop() | Drop collection from the database |
| db.newCollection.update(id, {$set: {“field”: “value”}}) | Update document fields mentioned in second parameter by document id in first parameter |
| db.newCollection.remove(id) | Delete document by id from collection |

*Table №2 – MongoDB shell commands*

# 4.3 ReactJS

React is front end library, which used for handling view layer for web applications. It allows to crate reusable UI components which presents data that changes over time. Lots of people use React as the V

in MVC (Model-View-Controller). It abstracts away the DOM from you, giving a simpler programming model and better performance [10].

If summarize the features and advantages of React are:

- JSX – is JavaScript syntax extension. It is hardly recommended to use it in developing React applications.
- Components – let you split the UI into independent, reusable pieces. They are like JavaScript function – accept some input and return elements describing what should be drawn on the screen. Also they improve readability and this is helps to maintain bigger applications.
- One-way data flow, which makes easier understand application.
- Virtual DOM – is JavaScript object and this improve applications performance since it is faster than the regular browser DOM.
- Can be used with other framework. In case of this thesis, with Meteor.

React uses JSX for templating purposes instead of regular JavaScript. It neither a string nor HTML, however it looks like regular HTML in most cases. JSX is faster because of optimization while compiling code, it is type-safe and the errors can be detected during compilation. The example of JSX file syntax, where returning value are HTML element div, shown on image №10:"File with JSX syntax". Also it is allows to embed any JavaScript expression by wrapping in curly braces.

```
import React from 'react';

class App extends React.Component {
   returnHelloWorld() {
      return “Hello World!”;
   }
   render() {
      return (
         <div>
            {returnHelloWorld()}
         </div>
      );
   }
}
```

*Image №10 – File with JSX syntax*

Component – is an independent and reusable piece of UI, as an input they accepts, from parent component, props and as an return gives back React element to render on the screen. There are two types of them – functional, when Component defined as function, and class Component, when it defines as class. Component names should always start from capital letter, because native DOM element names starts from lowercase letters.

To show developed UI on web page, we need to import file with root component to main JavaScript file and call there React render method with first parameter name of root component and

the second one is an html element with specified id where to place it: `ReactDOM.render(<RootComponent />,document.getElementById('root'))`.

As was said, props are passed from parent components and cannot be changed, they are should be immutable. That's why, for making dynamic updates in UI used state, which allows to change components output over time in response to user action, server responses, etc. For this proposal, in component's container should be initialized state. An example of using props and state shown on image №11: "Components with props and state", on it seen that root component has state with names of the tasks and propagate it to the child components via their props, so in the moment when one of names would be changed, only corresponding child component will be re rendered.

```
import React from 'react';

class App extends React.Component {
   constructor() {
      this.state = {
        taskName1 = "Task 1",
        taskName2 = "Task 2"
      }
   }
   render() {
      return (
       <div>
         <Task name={this.state.taskName1}/>
         <Task name={this.state.taskName2}/>
       </div>
      );
   }
}

class Task extends React.Component {
   render() {
      return (
         <p>{this.props.name}</p>
      );
   }
}
```

*Image №11 – Component with props and state*

# 4.4 Atlassian Jira API

Jira is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking, and project management functions [12]. It provides both Java APIs and REST APIs that can be used to cooperate with Jira programmatically.

REST APIs provide access to resources (data entities) via URI paths. To use a REST API, your application will make an HTTP request and parse the response. Your methods will be the standard

HTTP methods like GET, PUT, POST and DELETE. REST APIs operate over HTTP(s) making it easy to use with any programming language or framework. The input and output format for the JIRA REST APIs is JSON [13].

Using this API it is possible to retrieve an issue data, create issues, edit and update. For instance, to retrieve specific issue data you need to send GET request in following format: `http://hostname/rest/api/2/issue/{issueKey}` and as response you get needed information in JSON format.

Also exists program drivers, which provides an object oriented wrapper for Jira Rest API. One of them are used in this work. It is NodeJS module called jira-client. After adding this module to application you need to initialize connection to Jira instance, providing used protocol, host address and authentication credentials. Using this driver it is possible to interact with needed data in allowed way, for instance to find an issue called following method and in promise get needed information: `jira.findIssue(issueNumber).then(function(issue){console.log(issue);})`.

It is possible to add new issue to Jira via method `addnewIssue(issue: object)` or delete it via `deleteIssue(issueId)`. Given the fact, that Jira has her own query language called JQL (Jira query language), you can also search issues or users by search criteria by passing it to method `searchJira(searchCriteriaString)`. For update an issue you need to call the following method: `updateIssue(issueId, issueUpdateObject)`, and as the first parameter specify issue id and as the second one an object with updating data.

# 5 Implementation

This part of thesis will describe the way tool would be implemented and the UX interface for it. Assumed that graphical interface of implemented web application are only UX and functional. As it was described it previously, the main goal of this project is to design and implement a batter way of working with Hotfix releasing management for all participants, never create beautiful user interface.

The graphical part of new Hotfix Portal would be divided by user roles: developer, tester and product coordinator. Each role will receive permission to view and modify only data related to their work responsibilities.

## 5.1 Database design

For storing and collecting purposes would be used MongoDB collections. Database will consist of eight collections – users, roles, versions, statuses, mail_list, components, cits and hfs. Assumed that MongoDB collection is schema-less JSON array of objects with internally generated unique ids. Description of each collection shown in table №3: “MongoDB collections description”. Collections roles, users, versions, mail_list, components and statuses would be filled on initial application running with data from configure files edited by needs. The description of these files are at chapter 5.4: “Configuration” of this work.

Data in cits and hfs collections are the most important and will contain information about submitted correction information for fixed issues and Hotfixes information respectively.

Collections users and roles serve for authentication purposes. Mail_list collection – is defined and used as store of mail groups for notification. The last three collections – versions, statuses and components – are used as a source of data for select box graphical element, which is described at chapter 5.3: “UX and GUI implementation” of this thesis.

| Collection name | Attribute | Description |
|---|---|---|
| users | email | User email |
| | password | Saved bcrypted password |
| | role | Developer, tester or product coordinator |
| | createdAt | Creating timestamp |
| roles | name | Names of defined roles |
| statuses | value | Name of status used as attribute at hfs collection |
| | label | Status name for GUI |
| | nextStatuses | Set of allowed next statuses for current status |

*Table №3 – MongoDB collections description*

| Collection name | Attribute | Description |
|---|---|---|
| versions | value | Internal value merged product and version |
| | label | Version name for GUI |
| | product | Product name (Assistant or Manager) |
| | version | Version in internal format (example: v7_r2.14) |
| cits | citNo | Incremental CIT number, starts from 1 |
| | issueNo | Issue number in Jira |
| | owner | User id by whom created |
| | priority | Priority of Jira issue |
| | description | Short description of Jira issue |
| | ticketNo | Customer ticket number |
| | comment | Developer comment |
| | deactivable | True\|\|False, if this fix are deactivatable |
| | version | Version where fix done |
| | product | Product where fix done |
| | components | List of components where fix done |
| | createdAt | Creating timestamp |
| | hfId | Assigning HF id |
| | email | Developer email |
| | test | passed\|\|failed test result |
| | testedBy | Tester email |
| hfs | hfNumber | Incremental HF number, from 1, for current product and version |
| | modifiedAt | Timestamp of last modifying |
| | version | Version from versions collection |
| | product | Product from versions collection |
| | status | Current status of HF |
| mail_list | value | Array of mail strings |
| | label | Group name for GUI |
| components | value | Internal name of component |
| | label | Name of component for GUI |

*Continuation of table №3 – MongoDB collections description*

## 5.2 Application implementation

Hotfix Portal would be real-time web based application running on Meteor engine, using as database MongoDB and GUI interface implemented in ReactJS framework. Functionality would be divided, as was said, by user roles in Hotfix release management – developer, tester and product coordinator. For this proposes, user accounting management will be used from accounts-ui, accounts-passwords and roles packages for Meteor engine. Based on analysis, workflow and summarized problems use-cases was defined – developer workflow on image №12: “Developer use-case”, tester workflow on image №13: “Tester use-case” and finally product coordinator workflow on image №14: “Product coordinator use-case” respectively.

Next step would be to eliminate summarized problems for each use-case detected and described at chapters 2.1 Developer view, 2.2 Tester view and 2.3 Product coordinator view of this work.

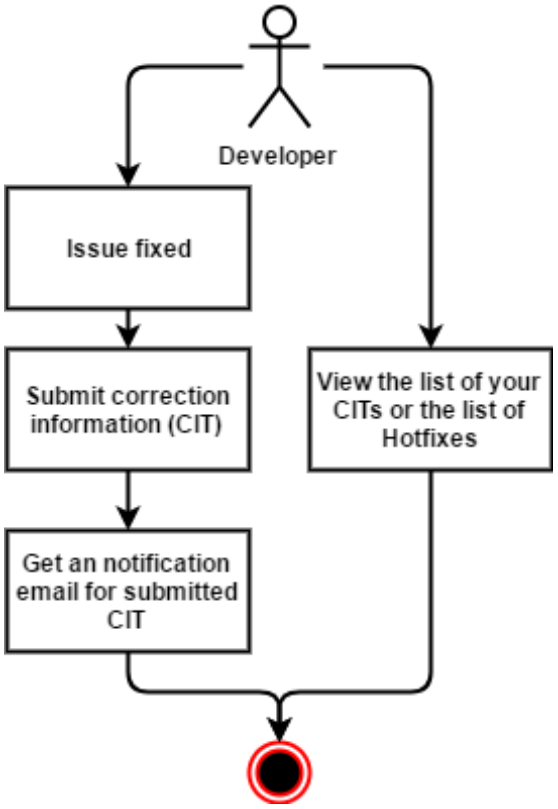



*Image №12 – Developer use-case*

Eliminated problems in developer view with description:

- Manual input of email address and telephone number would be backed off by accounting management.
- Email list for notifications about submitted CIT solved via mail groups, such as developer, testers, management.
- Automatically update priority and description after Jira issue number is placed.

- Archaic platform and version names changed by version labels from versions collection described at 4.1 Database design.
- New tab would be implemented with displayed “My CITs”.
- New tab would be implemented for read only viewing list of Hotfixes.

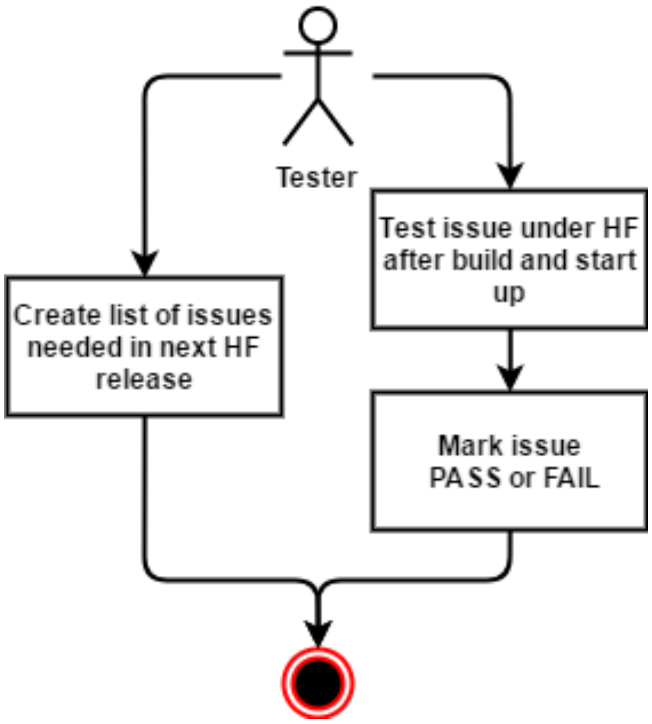



*Image №13 – Tester use-case*

Eliminated problems in tester view with description:

- List of all Hotfixes ready to testing would be available for all testers via their view of Hotfix Portal.
- Redefined information described for each CIT to test.
- Submitting test result via select box and automatically modifying CIT with information about tester.
- New tab would be implemented for read only viewing list of Hotfixes.
- Roles in accounting management eliminate this possibility.

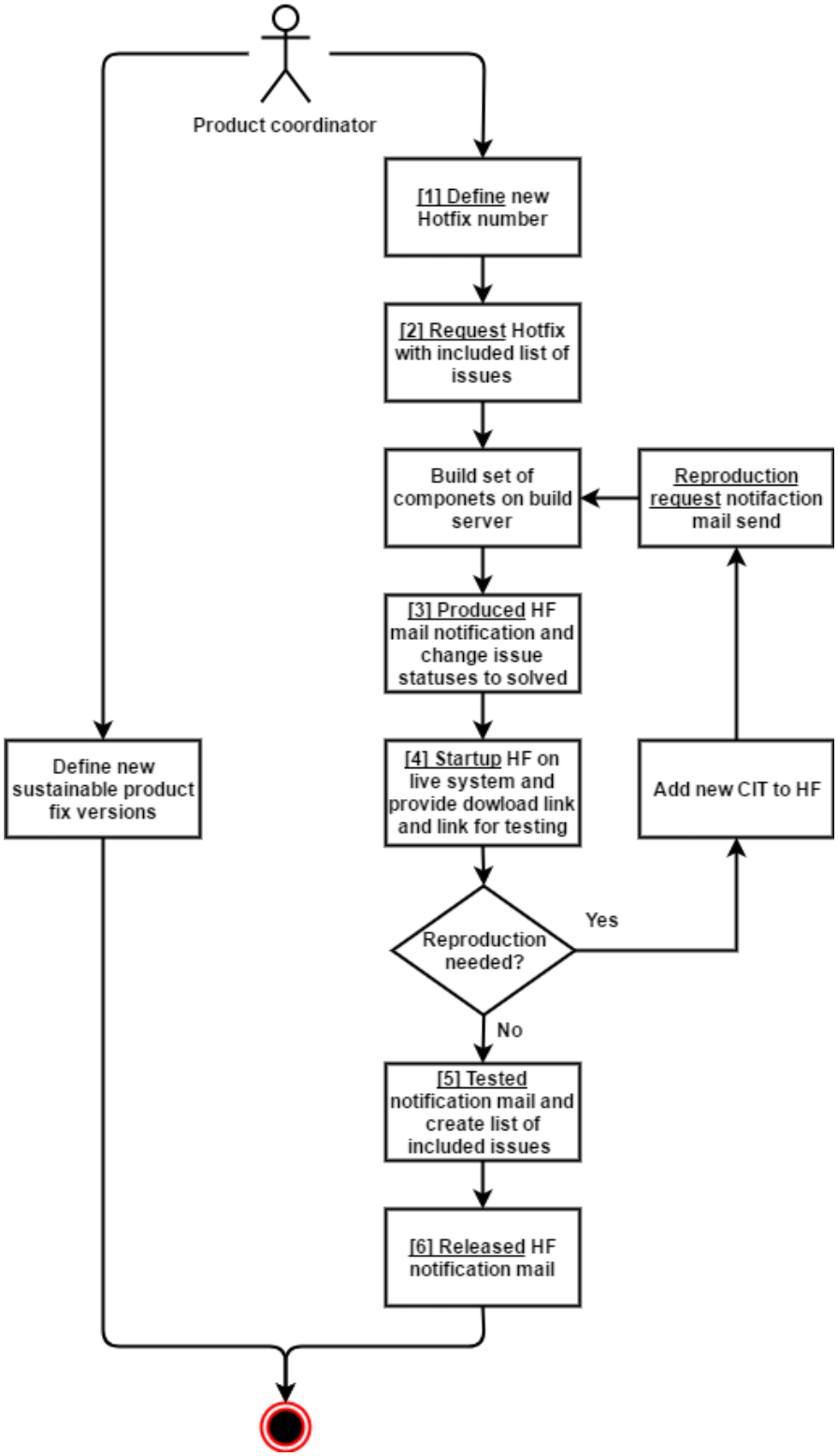



*Image №14 – Product coordinator use-case*

Eliminated problems in product coordinator view with description:

- Graphical interface for adding new sustainable product versions would be implemented.
- Define new Hotfix by product and version selection.
- Adding unassigned CITs to requested Hotfix via multiple selection by product and version.

- Generate list of components to be produced automatically.
- Change Hotfix status via select box with allowed next statuses from statuses collection described at 4.1 Database design.
- Automatically set Jira issues to solved state after corresponding Hotfix in produced status.
- Automatically request reproduction if some test failed.
- List of issues under tested and released Hotfix automatically generated.
- New notification mails.

## 5.3 UX and GUI implementation

As a part of new Hotfix Portal was UX design improving and graphical interface implementation in ReactJS framework. At this part, would be shown images with new functional design. As a main element of graphical interface was decided to choose a table element, because it is nice to browse and map each correction information to table row or Hotfix information to table row.

The developer view consists of three tabs – tab for add new correction information, tab with read only list of his previously submitted CITs and tab with read only list of Hotfixes. The last two tabs serve for review purposes. On image №15: “Developer submit CIT form” seen new UX and graphical interface for adding correction information ticket. When developer input complete Jira issue number – priority and description will be fielded automatically via retrieving Jira issue information. Need to mention also that by default mail notification will be send to all mail groups, if needed, developer can delete unnecessary groups or via field “Additional mails” add further mails.


*Image №15 – Developer submit CIT form*

On image №16: “Developer select versions” and on image №16: “Developer select components” respectively seen implementation of new version and components selection, which will minimize needed manual steps.

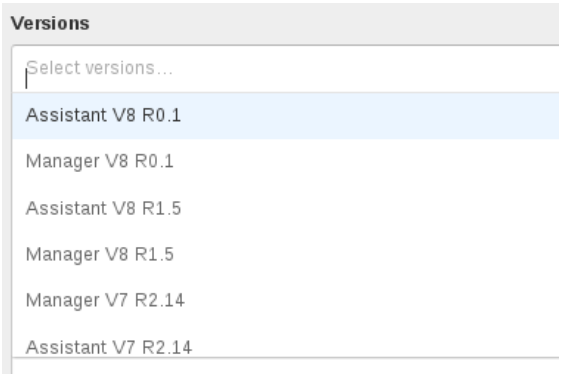



*Image №16 – Developer select versions*

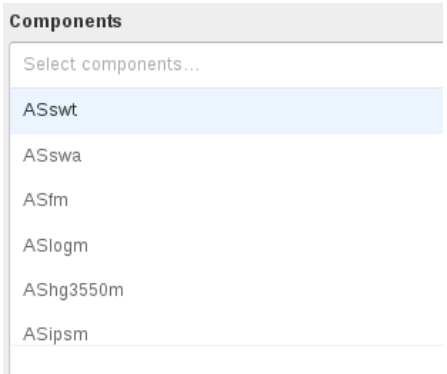



*Image №17 – Developer select components*

Under second tab developer could see his CIT divided to unassigned section and assigned to specific Hotfix section. Each column provide the most important information about it, such as correction information number, version and product information, components list, owner, creation time and corresponding Jira issue number. For further is possible to view detail information about each CIT, example of this shown on image №18: “Developer My CITs tab”, where user can find description, comment, priority, deactivatable possibility and test result, if placed.

| | | | | | | |
|---|---|---|---|---|---|---|
| + | CIT-44 | Manager V8 R0.1 | swt | sergii@ixperta.com | 5/13/2017, 3:25:18 PM | OSFOURK-4985 |
| + | CIT-45 | Assistant V8 R1.5 | swa | sergii@ixperta.com | 5/13/2017, 3:30:01 PM | OSFOURK-4986 |
| − | CIT-46 | Assistant V8 R0.1 | swa | sergii@ixperta.com | 5/13/2017, 3:41:17 PM | OSFOURK-4988 |
| **Description:** | When you login via ssh or GUI into Manager there is a message that the /var partition is full | | | | | |
| **Comment:** | | | | | | |
| **Priority:** | P3 | | | | | |
| **Deactivable:** | True | | | | | |
| + | CIT-47 | Assistant V8 R0.1 | AShg3550m | sergii@ixperta.com | 5/15/2017, 1:56:23 PM | OSFOURK-3333 |
| **HF 1 Manager V8 R0.1** | | | | | | |
| + | CIT-25 | Manager V8 R0.1 | fm | sergii@ixperta.com | 4/8/2017, 4:11:06 PM | OFSOURK-77777 |
| + | CIT-10 | Manager V8 R0.1 | swa,fm | sergii@ixperta.com | 4/6/2017, 9:14:20 AM | OSFOURK-456 |
| + | CIT-13 | Manager V8 R0.1 | swa | sergii@ixperta.com | 4/6/2017, 10:00:39 PM | OSFOURK-12345 |

*Image №18 – Developer My CITs tab*

New product coordinator view will contains two tabs. The first one is with list of Hotfixes, button to define new one, button to create new supported version, possibility to change status via select box with allowed next statuses, button produce for requested and reproduction requested statuses. The Hotfixes will be divided into sections by version and each row would contain the following information: Hotfix number, product, status, with possibility to change it via select box with allowed next statuses, and last modified time. The additionally provided information would be the list of attached corrections. The example of this tab shown on image №19: “Product coordinator Hotfixes tab”.

Define new HF | Define new supported version

| | HF number | Product | Status | Last modify |
|---|---|---|---|---|
| **V8 R0.1** | | | | |
| + | HF 1 | Assistant | **Released** | 3/20/2017, 8:24:45 PM |
| + | HF 2 | Assistant | **Released** | 4/8/2017, 7:40:32 PM |
| − | HF 1 | Manager | Started up/In test | 4/27/2017, 9:42:11 AM |

| | | | | | |
|---|---|---|---|---|---|
| CIT-25 | OFSOURK-77777 | fm | sergii@ixperta.com | | |
| CIT-7 | OSFORK-3423423 | hg3550m | franta@ixperta.com | Test passed | Tested by alina@unify.com |
| CIT-10 | OSFOURK-456 | swa,fm | sergii@ixperta.com | Test passed | Tested by alina@unify.com |
| CIT-13 | OSFOURK-12345 | swa | sergii@ixperta.com | | |
| CIT-15 | OSFOURK-123123 | hg3550m | sergii@ixperta.com | | |
| CIT-17 | OSFOURK-345345 | ipsm | sergii@ixperta.com | | |
| CIT-21 | OSFOURK-66666 | logm | sergii@ixperta.com | | |
| CIT-23 | OSFOURK-77777 | fm | sergii@ixperta.com | | |
| CIT-3 | OSFOURK-345435 | swt,swa,fm | sergii@ixperta.com | Test failed | Tested by alina@unify.com |
| CIT-29 | OSFOURK-3454563 | swa | sergii@ixperta.com | | |
| CIT-31 | OSFOURK-1232343 | fm | sergii@ixperta.com | | |
| CIT-37 | OSFOURK-789789 | swt | sergii@ixperta.com | | |
| CIT-39 | I55555 | swa | sergii@ixperta.com | | |

*Image №19 – Product coordinator Hotfixes tab*

To define new Hotfix product coordinator should click button “Define new HF” placed, under the first tab, and after this form with version and product select box will be shown, as on image №20: “Product coordinator define new HF under selected version”.

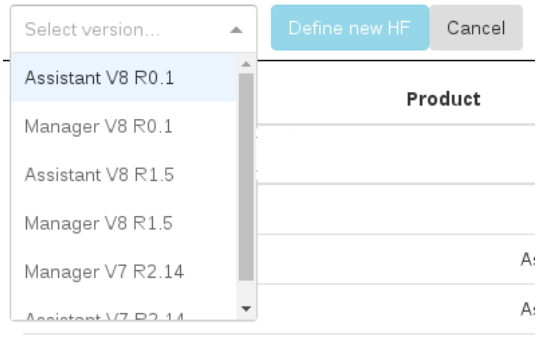



*Image №20 – Product coordinator define new HF under selected version*

As was said above, on the first tab also placed functionality for defining new supported versions and products. After click corresponding button the dialog for placing new supported version will be shown. The example is on image №21: “Product coordinator modal dialog for defining new supported version”.

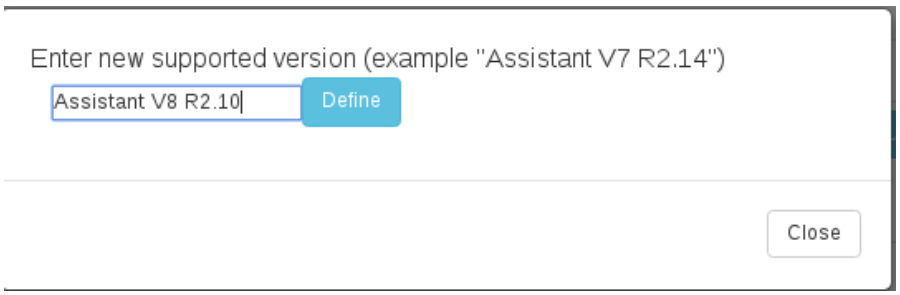
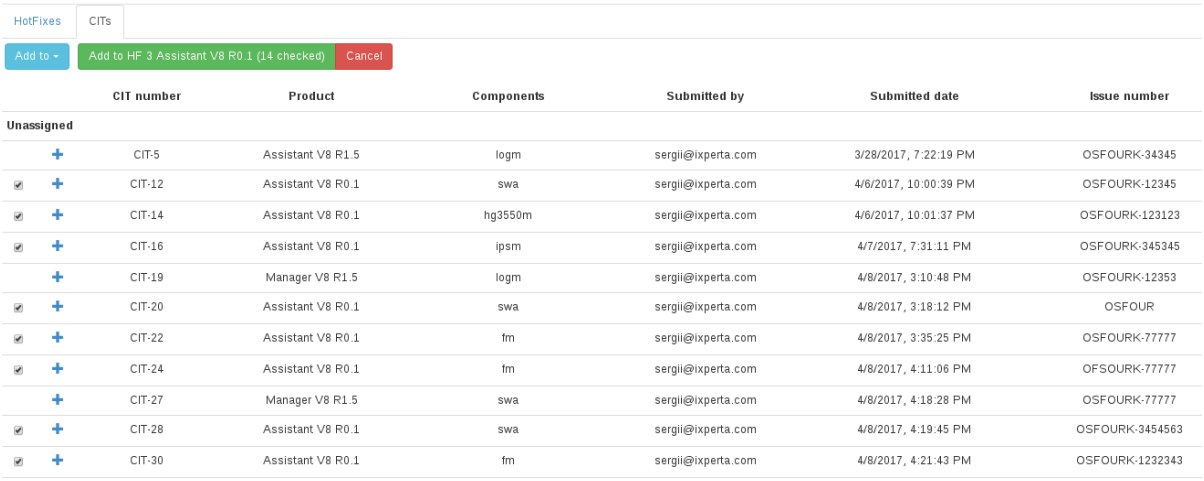



*Image №21 – Product coordinator modal dialog for defining new supported version*

To obtain list of components to be produced, for needed Hotfix in one of the following statuses: “Requested” or “Reproduction requested”, provided button “Produce”, which will show needed components in modal dialog, as on image №22: “Product coordinator components list to produce”.


*Image №22 – Product coordinator components list to produce*

The second tab of product coordinator view, provides list of all CITs divided to unassigned and assigned sections – is the same as at developer view, but with possibility to assign unassigned CITs to selected Hotfix, as shown on image №23: “Product coordinator add CITs to specified Hotfix”.




| | CIT number | Product | Components | Submitted by | Submitted date | Issue number |
|---|---|---|---|---|---|---|
| Unassigned | | | | | | |
| + | CIT-5 | Assistant V8 R1.5 | logm | sergii@ixperta.com | 3/28/2017, 7:22:19 PM | OSFOURK-34345 |
| ☑ + | CIT-12 | Assistant V8 R0.1 | swa | sergii@ixperta.com | 4/6/2017, 10:00:39 PM | OSFOURK-12345 |
| ☑ + | CIT-14 | Assistant V8 R0.1 | hg3550m | sergii@ixperta.com | 4/6/2017, 10:01:37 PM | OSFOURK-123123 |
| ☑ + | CIT-16 | Assistant V8 R0.1 | ipsm | sergii@ixperta.com | 4/7/2017, 7:31:11 PM | OSFOURK-345345 |
| + | CIT-19 | Manager V8 R1.5 | logm | sergii@ixperta.com | 4/8/2017, 3:10:48 PM | OSFOURK-12353 |
| ☑ + | CIT-20 | Assistant V8 R0.1 | swa | sergii@ixperta.com | 4/8/2017, 3:18:12 PM | OSFOUR |
| ☑ + | CIT-22 | Assistant V8 R0.1 | fm | sergii@ixperta.com | 4/8/2017, 3:35:25 PM | OSFOURK-77777 |
| ☑ + | CIT-24 | Assistant V8 R0.1 | fm | sergii@ixperta.com | 4/8/2017, 4:11:06 PM | OFSOURK-77777 |
| + | CIT-27 | Manager V8 R1.5 | swa | sergii@ixperta.com | 4/8/2017, 4:18:28 PM | OSFOURK-77777 |
| ☑ + | CIT-28 | Assistant V8 R0.1 | swa | sergii@ixperta.com | 4/8/2017, 4:19:45 PM | OSFOURK-3454563 |
| ☑ + | CIT-30 | Assistant V8 R0.1 | fm | sergii@ixperta.com | 4/8/2017, 4:21:43 PM | OSFOURK-1232343 |

*Image №23 – Product coordinator add CITs to specified Hotfix*

New tester view also contains two tabs. The one of them, is the same to the developer's view read-only Hotfix list. But another tab provides the list of correction information tickets, sectioned by Hotfixes, ready for test. This list looks the same table with CITs, but has two tester specific columns with information about tester and test result. In the moment, when the time for placing result comes, user can choose it from select box, it is also possible to change previously selected result, if needed. The columns described above are shown on image №24: "Columns for fill test results" and corresponding result selection on image №25: "Test result select box".

| Tested By | Test Result |
|---|---|
| | |
| | |
| alina@unify.com | Failed |
| alina@unify.com | Passed |
| alina@unify.com | Passed |
| Waiting for test | Test... |
| Waiting for test | Test... |

*Image №24 – Columns for fill test results*

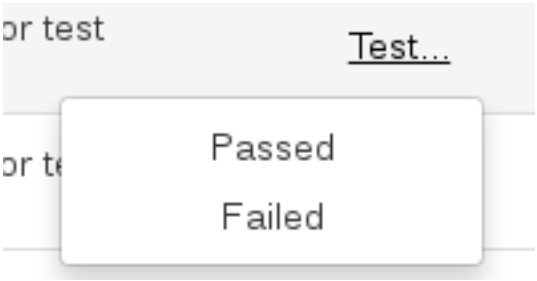



*Image №25 – Test result select box*

# 5.4 Configuration

For using this new tool few configuration steps needed, such as configuring smtp mail server address for possibility to send notification mails, configuring connector to Jira API, for possibility to get or edit issues, fill private files with initial data about users, roles, versions, components and mail groups for corresponding database collections.

To configure smtp server address need to set environment variable `MAIL_URL` in the following form `smtp://USERNAME:PASSWORD@HOST:PORT`. If it configured right, Meteor Email package will use it for sending mails, if not, mails will be printed to standard output.

To configure connector to Jira API needed to edit file called configJira.json, which can find in private directory of tool. Example of configured connector shown on image №26: “Example of configJira.json file”.

```
{
"protocol": "https",
"host": "yourjira.com",
"port": "443",
"user": "user",
"password": "password",
"apiVer": "2",
"strictSSL": false
}
```

*Image №26 – Example of configJira.json file*

Private files with initial data for collections also placed in private directory and can be edited before initial run of Hotfix portal.

On image №27: “Example of file with user accounts users_init.json” shown an example of file with defined four initial users with attached roles, which are corresponds to graphical interface views.

```
[{"name":"stefan","email":"stefan@unify.com","roles":["production"]},
  {"name":"alina","email":"alina@unify.com","roles":["tester"]},
  {"name":"sergii","email":"sergii@ixperta.com","roles":["developer"]},
  {"name":"franta","email":"franta@ixperta.com","roles":["developer"]}]
```

*Image №27 – Example of file with user accounts users_init.json*

On image №28: “Example of file with versions list versions.json” shown the possible initial supported products and versions.

```
[{ "value" : "assistant_V8_0.1", "label" : "Assistant V8 R0.1",
"product" : "assistant", "version" : "v8_r0.1" },
    { "value" : "manager_V8_0.1", "label" : "Manager V8 R0.1",
"product" : "manager", "version" : "v8_r0.1" },
    { "value" : "assistant_V8_1.5", "label" : "Assistant V8 R1.5",
"product" : "assistant", "version" : "v8_r1.5" },
    { "value" : "manager_V8_1.5", "label" : "Manager V8 R1.5",
"product" : "manager", "version" : "v8_r1.5" },
    { "value" : "manager_V7_2.14", "label" : "Manager V7 R2.14",
"product" : "manager", "version" : "v7_r2.14" },
    { "value" : "assistant_V7_2.14", "label" : "Assistant V7 R2.14",
"product" : "assistant", "version" : "v7_r2.14" }]
```

*Image №28 – Example of file with versions list versions.json*

On image №29: “File with statuses statuses.json” shown the possible Hotfix statuses with allowed next statuses list for each.

```
[{"value": "defined", "label": "Defined", "nextStatuses":
["requested"]},
  {"value": "requested", "label": "Requested", "nextStatuses":
["produced"]},
  {"value": "produced", "label": "Produced", "nextStatuses":
["start_test","repro_req"]},
  {"value": "repro_req", "label": "Reproduction requested",
"nextStatuses": ["produced"]},
  {"value": "start_test", "label": "Started up/In test",
"nextStatuses": ["tested","repro_req"]},
  {"value": "tested", "label": "Tested", "nextStatuses":
["released","repro_req"]},
  {"value": "released", "label": "Released", "nextStatuses": []}]
```

*Image №29 – File with statuses statuses.json*

On image №30: “Example of file with mails mail_list.json” shown the file with needed format to define initial mail groups.

```
[
    { "value": ["sergii.khunovych@ixperta.com",
"khunovich.s@gmail.com"], "label": "Development" },
    { "value": ["sergii.khunovych@ixperta.com",
"khunovich.s@gmail.com"], "label": "Testers" },
    { "value": ["sergii.khunovych@ixperta.com",
"khunovich.s@gmail.com"], "label": "Management" },
    { "value": ["sergii.khunovych@ixperta.com",
"khunovich.s@gmail.com"], "label": "Production" }
]
```

*Image №30 – Example of file with mails mail_list.json*

On image №31: “Example of file with components components.json” shown the possible initial list of components.

```
[
    { "value": "ASswt", "label": "ASswt" },
    { "value": "ASswa", "label": "ASswa" },
    { "value": "ASfm", "label": "ASfm" },
    { "value": "ASlogm", "label": "ASlogm" },
    { "value": "AShg3550m", "label": "AShg3550m" },
    { "value": "ASipsm", "label": "ASipsm" },
    { "value": "AScm", "label": "AScm" },
    { "value": "AScommon", "label": "AScommon" },
    { "value": "AScomwin", "label": "AScomwin" },
    { "value": "ASxie", "label": "ASxie" }
]
```

*Image №31 – Example of file with components components.json*

# 6 Conclusion

As a result of this work, the web based application called Hotfix Portal was created. It allows users to do their daily job in Hotfix releasing process in more easy way. Each user under specified role, as such as developer, tester and product coordinator, see, modify and cooperate only with allowed view of application.

For instance, Hotfix portal allows developer to submit correction information for fixed issue, via special form with minimized manual steps, or view previously submitted correction information with description under which Hotfix it attached and view current statuses of all existing Hotfixes. For tester it allows to place test results, via special table, where seen only Hotfixes for test, and, the same as for previous user, view list of all Hotfixes. Product coordinator has the ability to define new supported version and new Hotfixes under them, change status of each Hotfix with corresponding automatic mail notification for all participants defined in mail list. Also, he has possibility to attach correction information tickets (CITs) to needed Hotfixes and get a list of component to be produced for specified Hotfix.

Before creating application, used technologies and methodologies were studied. Also, core of problems and process were analyzed, described and the way to eliminate them are proposed. Alliance of Meteor, MongoDB and React was chosen as a common and open source way to develop modern web application. Next step was to define all features that such application must have and choose what will be implemented. On this basis, database structure and graphical user interface, with improving usability, accessibility and easiness of use, were designed. After that, the Hotfix portal application was implemented in JavaScript programming language.

The application was implemented to be flexible, as it is possible, for different products and users. For this purposes were added configuration files with ability to edit list of components, create initial list of supported version and products, create users with defined roles. Also, were implemented separately mail notification interface, where with some improvements could be changed view of sending mails.

The staff training and presentation using the new tool was conducted.

## 6.1 Next steps

In order to achieve more automation in implementation of Hotfix releasing process can be provided next improvements. Bigger integration with product management tools to create correction information by changing issue status in Jira to solved. In workflow of product coordinator, big improvement could be automatic build of Hotfix set of components with following start up on test system. Also, defining the list of Jira issues which must be included into specified Hotfix and following their automatic

attaching to it via submitting correction information. Mail notifications could be improved by using templating mechanism and sending HTML-based emails.

Another improvement might be adding scheduled time and date for Hotfix producing with automation build and start upping.

# References


[1] *Meteor introduction*, Meteor [online]. [cit. 2017-05-23]. Available at the URL: <https://guide.meteor.com>

[2] *Meteor React tutorial*, Meteor [online]. [cit. 2017-05-23]. Available at the URL: <https://www.meteor.com/tutorials/react>

[3] *Meteor-roles*, Atmosphere [online]. [cit. 2017-05-23]. Available at the URL: <https://atmospherejs.com/alanning/roles>

[4] *Meteor methods*, Meteor [online]. [cit. 2017-05-23]. Available at the URL: <https://guide.meteor.com/methods.html>

[5] *Using the Email Package*, The Meteor Chef [online]. [cit. 2017-05-23]. Available at the URL: <https://themeteorchef.com/tutorials/using-the-email-package>

[6] *MongoDB Overview*, TutorialsPoint [online]. [cit. 2017-05-23]. Available at the URL: <https://www.tutorialspoint.com/mongodb/mongodb_overview.htm>

[7] *Thinking in Documents: Part 1*, MongoDB blog [online]. [cit. 2017-05-23]. Available at the URL: <https://www.mongodb.com/blog/post/thinking-documents-part-1>

[8] *Databases: Part 1*, MeteorTips [online]. [cit. 2017-05-23]. Available at the URL: <http://meteortips.com/first-meteor-tutorial/databases-part-1>

[9] *Databases: Part 2*, MeteorTips [online]. [cit. 2017-05-23]. Available at the URL: <http://meteortips.com/first-meteor-tutorial/databases-part-2>

[10] *ReactJS Tutorial*, TutorialsPoint [online]. [cit. 2017-05-23]. Available at the URL: <https://www.tutorialspoint.com/reactjs>

[11] *ReactJS Hello World*, Facebook React [online]. [cit. 2017-05-23]. Available at the URL: <https://facebook.github.io/react/docs>

[12] *Jira (software)*, Wikipedia [online]. [cit. 2017-05-23]. Available at the URL: <https://en.wikipedia.org/wiki/Jira_(software)>

[13] *Jira REST APIs*, Developer Atlassian [online]. [cit. 2017-05-23]. Available at the URL: <https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis>

[14] *JavaScript JIRA API for node.js*, Jira-node [online]. [cit. 2017-05-23]. Available at the URL: <https://jira-node.github.io>

# List of Appendices

# Appendix A

CD/DVD contents:

- Technical documentation
- Source code of application
- Private configuration files