TU/e

Project Fingerpaint

URD0.2

User Requirements Document

Authors:
Tessa Belder
Lasse Blaauwbroek
Thom Castermans
Roel van Happen
Benjamin van der Hoeven
Femke Jansen
Hugo Snel

Junior Management:
Simon Burg
Areti Paziourou
Luc de Smet

Senior Management:
Mark van den Brand
Lou Somers

Advisor:
Ion Barosan

Customer:
Patrick Anderson

April 26, 2013

Abstract

This document describes the User Requirements of Fingerpaint . This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The User Requirements Document (URD) is based on the ESA standard for software development, as set by the European Space Agency (ESA) [1]. This documents lists what the Fingerpaint project should be capable of, how it should function and in what environment it should function.

Contents

Document Status Sheet

General

Document title:	User Requirements Document
Identification:	URD0.2
Author:	Tessa Belder, Roel van Happen, Benjamin van der Hoeven, Femke Jansen, Hugo Snel
Document status:	Draft

Document history

Version	*Date*	*Author*	*Reason of change*
0.1	24-Apr-2013	Tessa Belder Roel van Happen Benjamin van der Hoeven Femke Jansen Hugo Snel	Initial version.
0.2	26-Apr-2013	Tessa Belder Roel van Happen Benjamin van der Hoeven Femke Jansen Hugo Snel	Revised version as prompted by the client meeting on 25-Apr

Document Change Records since previous issue

General

Datum:	26-Apr-2013
Document title:	User Requirements Document
Identification:	URD0.2

Changes

For each of the document changes listed here, we will refer to the necessary versions or IDs from the documents, to clarify which paragraphs from which versions have been changed.

Version	*Page*	*Paragraph*	*Reason to change*
0.1	6	2.1	Clarified that we do not compute any matrices.
0.1	6	2.2	Further divided this section into two sections: *Mixing constraints* and *Additional capabilities*.
0.1	6	2.2	Added two new constraints and reworded the original two.
0.1	6	2.2	Clarified the possible geometries.
0.1	6	2.2	Clarified user input.
0.1	6	2.2	Added that the user should be able to decide to reset, or re-use a received concentration distribution after each iteration of the protocol.
0.1	8	3.1	Split requirement CPR01 from URD0.1 in CPR01, CPR02 and CPR03, as prompted by the client meeting on 25-04.
0.1	8	3.1	Split requirement CPR02 from URD0.1 in CPR04, CPR05 and CPR06, as prompted by the client meeting on 25-04.
0.1	8	3.1	Changed the priority from requirements CPR03 and CPR04 in URD0.1 to *won't have*, as prompted by the client meeting on 25-04.
0.1	8	3.1	Renamed requirement CPR05 to CPR11, because of the change in order of the previous requirements in URD0.2.
0.1	8	3.1	Split requirement CPR06 from URD0.1 in CPR12, CPR13 and CPR14, as prompted by the client meeting on 25-04.

Version	*Page*	*Paragraph*	*Reason to change*
0.1	8	3.1	Added new requirements CPR09, CPR10, CPR15 in URD0.2, as prompted by the client meeting on 25-04.
0.1	9	3.1	Added new requirements CPR15, CPR16, CPR17, CPR18 and CPR20 in URD0.2, as prompted by the client meeting on 25-04.
0.1	9	3.1	Merged requirements CPR07 and CPR08 into requirement CPR19, as prompted by the client meeting on 25-04 and performed the rename because of change of the previous requirements in URD0.2.
0.1	9	3.1	Renamed requirements CPR09 to CPR19 from URD0.1 to CPR21 to CPR31, because of the change in order of the previous requirements in URD0.2.
0.1	9	3.1	Changed the format in CPR22 from URD0.2 to SVG instead of PNG/GIF.
0.1	9	3.1	Changed the format in CPR25 from URD0.2 to EVA instead of APNG/AGIF.
0.1	10	3.2	Removed constraint requirements CNR01 to CNR11, to lessen the restrictions on the (amount of) interfaces that will be used in the application.
0.1	10	3.2	Renamed requirements CNR12 to CNR17 from URD0.1 to CNR4 to CNR9, because of the change in order of the previous requirements in URD0.2.
0.1	10	3.2	Changed the iOS version from CNR4 in URD0.2 from 5 to 6, as prompted by the client meeting on 25-04.
0.2	10	3.1	Explanation about capability requirements added.
0.2	13	3.1	Explanation about constraint requirements added.
0.2	-	Abstract	Added that the project is a SEP project from the TUE, and described document content
0.2	5	1.1	Changed 'the URD' to 'this document' in the first sentence
0.2	5	1.3	Added URD to the definitions
0.2	5	1.5	replaced 'remainder chapters' with 'remaining chapters'
0.2	6	1.5	moved the paragraph references to the end of each line.

Chapter 1

Introduction

1.1 Purpose

This document contains the requirements for Fingerpaint . These requirements are a negotiated agreement between prof.dr.ir. P.D. Anderson and Group Fingerpaint . All of the listed requirements, and only these, will be implemented in Fingerpaint, according to their priorities. Any changes to these requirements require the full consent of both parties.

1.2 Scope

Fingerpaint is an application which visualises fluid mixing on a mobile device. Users can define the initial concentration, as well as manipulate the mixing protocol. The resulting fluid distribution can be stored and analyzed by the user for comparison purposes.

1.3 List of definitions

2IP35	The Software Engineering Course
Client	prof.dr.ir. P.D. Anderson
CM	Configuration Manager
CPR	Capability Requirement
CNR	Constraint Requirement
TU/e	Eindhoven University of Technology
SEP	Software Engineering Project
SR	Software Requirements
SRD	Software Requirements Document
TBC	To Be Confirmed
TBD	To Be Defined
URD	This document, the User Requirements Document

1.4 List of references

[1] ESA, *ESA Software Engineering Standards.* March 1995.

[2] COLEY consulting, "Moscow prioritisation." http://www.coleyconsulting.co.uk/moscow.htm. [Online; accessed 24-April-2013].

1.5 Overview

The remaining chapters describe the user requirements in more detail. Chapter 2 gives a general description of

- The relation to other systems (2.1)
- The main capabilities (2.2)
- Constraint information and justification (2.3)
- User characteristics (2.4)
- The operational environment (2.5)
- Assumptions and dependencies (2.6)

Chapter 3 gives a detailed list of the system's capability requirements in section 3.1, and a list of the constraint requirements is given in section 3.2.

Chapter 2

General description

This chapter describes general aspects of the application to be created as requested by the client.

2.1 Product perspective

The aim of this project is to deliver an application that allows the user to easily visualize the mixing of fluids. A user interface should be created which can be used to specify initial parameters, after which output should be shown on the screen. All of this should be possible using an easy to use, attractive interface on a mobile device.

A similar project was initiated around eleven years ago. The result of this project was a MATLAB implementation that achieved a similar goal as our project. However, its user interface is outdated by now, and it is impossible to comfortably use this solution on a mobile device. Part of this original solution was a FORTRAN implementation for the flow of the fluids. This implementation is still available, and we are to use it as a black box to which we can send constraints and a vector, after which the server will compute the flow. The resulting concentration distribution and its performance results are then sent back to the mobile device.

2.2 General capabilities

2.2.1 Mixing constraints

The system should be able to simulate the flow and mixing of a number of fluids, given some constraints. There are a number of constraints to be specified. The first constraint is the geometry of the mixer. There are four kinds of geometries in total: rectangle, square, circle, and *journal bearing*. We will start by implementing support for rectangular geometries, and will implement more geometries if time permits.

The second constraint concerns the characteristics of the mixer, which will be specified by different matrices. These matrices are located on the server and are pre-computed. Each geometry has its own set of possible characteristics and hence its own set of matrices. These characteristics influence the flow of the fluids.

The third constraint concerns parameters applicable to the mixing protocol. Available parameters are determined by the type of mixer defined in the second constraint. For example, a rectangular mixing geometry has two walls that can be moved. The step parameter D indicates the amount of time the wall should move. Possible values for D are 4, 2, 1, 0.5, 0.25 and 0.1. It is possible to both specify an entire protocol consisting of multiple wall movements, and to only execute one step at a time.

The fourth parameter is the initial concentration of the fluids, which can be specified by tapping on and dragging over the screen. If desired, it is also possible to load an existing initial distribution.

2.2.2 Additional capabilities

The main user interface should be run on a mobile device, such as an iPhone. On this device, the constraints mentioned above should be specified. When the initial parameters have been set, the computation is offloaded to a server, which computes the flow of the fluids. When the final result has been computed, this result is of course sent back to the mobile device, where it can be saved.

A history of past simulations is stored on the device, to compare previous runs with the current. The result of runs should also be exportable to easily sharable formats, such as .png or .pdf, with support for an alpha channel (to realize transparency). For more interactive results, entire runs should be exportable to animated .png or .gif files.

After each intermediate result is received by the mobile device, the user has the choice of continuing with the received concentration distribution, or to start over with the original initial concentration distribution.

2.3 General constraints

The user interface should be suitable for mobile devices, so it is easy to visualize the results and show them to other people without much hassle. To make it even easier to quickly demonstrate mixing results to others, the actual computation on the server should not take too long (a couple of seconds at most). We do not want to be locked to one specific type of device, so we have chosen to design a cross-platform solution. To easily share results, it should also be possible to export the result of a mixing run to image files, and entire runs to animated files.

2.4 User characteristics

As mentioned before, the user of the application should be able to specify initial parameters, and, after the application has sent these off to the server, should be able to view the results. The user can then store these results for later reference, and to export these results to image files with transparency.

2.5 Environment description

The main device for the user interface is the mobile device. We are planning to create a cross-platform solution, which means it will be possible to use the application on various kinds of devices. Examples of supported devices are Apple iPhones, Android phones or tablets. The initial concentration of the fluids, the mixing protocol and the shape of the mixer will be specified on such a device.

As mobile devices typically do not have the power (both processing power and battery capacity) to perform intensive computations, the hard work of computing the mixing will be offloaded to a server. The starting parameters described above will be sent to the server, which has an efficient FORTRAN implementation to solve the problem. While solving, intermediate results are sent back to the mobile device for displaying.

2.6 Assumptions and dependencies

This section contains some assumptions for the application to function properly.
As a mentioned in the previous section, the application uses the FORTRAN implementation on the server to perform all the calculations. Therefore, we assume this server always answers requests within a few seconds.

Chapter 3

Specific requirements

This chapter explicitely states all requirements and constraints of the application to be developed. The final product will be delivered confirm these requirements. Any requirements following from further request will be added here.
The requirements are prioritized using the MoSCoW model [2]. This model assigns one out of four priorities to each requirement:

Must have; requirements with this priority are essential for the product, and must be implemented.
Should have; requirements with this priority are not essential for the product to work. However, they are nearly as important as the *must have*'s and are therefore expected to be implemented.
Could have; requirements with this priority are a nice addition to the product, and may be implemented, if time and budget allow this.
Won't have; requirements with this priority will not be implemented in this version of the product, but may be nice to implement in future versions.

3.1 Capability requirements

This section lists all capability requirements for the product. These requirements explicitely state what the system should do.

CPR01	*must have*
The user can select a rectangular mixer geometry	
CPR02	*should have*
The user can select a square mixer geometry	
CPR03	*could have*
The user can select a circle or 'Journal Bearing' mixer geometry	
CPR04	*must have*
The user can define an initial concentration distribution with black and white by drawing on the touchscreen with his/her finger.	

CPR05 *should have*
The user can select an initial concentration distribution from a list of previously saved distributions.

CPR06 *could have*
The user can select a predifined initial concentration distribution from a list.

CPR07 *won't have*
The user can define a initial concentration distribution with more than two different colours.

CPR08 *won't have*
The user can choose which colours are used for the initial concentration distribution.

CPR09 *must have*
The user can reset the current concentration distribution.

CPR10 *should have*
The user can save an initial concentration distribution he/she has specified by drawing on the touchscreen.

CPR11 *must have*
The user can define a mixing protocol for a rectangular geometry as a sequence of movements of the upper and lower walls.

CPR12 *should have*
The user can define a mixing protocol for a square geometry as a sequence of movements of the upper and lower walls.

CPR13 *could have*
The user can define a mixing protocol for a circle geometry as a sequence of rotation of circle.

CPR14 *could have*
The user can define a mixing protocol for a 'Journal Bearing' geometry as a sequence of rotations of inner and outer circles.

CPR15 *must have*
The user can define a single movement and step (D) to be executed directly on the current concentration distribution.

CPR16 *should have*
The user can select a mixing protocol for the specified geometry from a list of previously saved mixing protocols.

CPR17 *could have*
The user can select a predifined mixing protocol for the specified geometry from a list.

CPR18 *must have*
The user can reset the current settings for the mixing protocol.

CPR19 *must have*
The user can define a step (D) for each movement from the mixing protocol, to indicate the time that this movement is applied.

CPR20 *must have*
The user can define how many times the mixing protocol is applied ($\#steps$).

CPR21 *must have*
Users can view an image of the endresult of applying the mixing protocol on the initial concentration distribution.

CPR22 *should have*
The user can save the image from CPR21 locally to their device in a vector format (e.g. SVG format).

CPR23	*should have*
Users can remove previously stored images from their device.	
CPR24	*could have*
Users can view an animation of applying the mixing protocol on the initial concentration distribution.	
CPR25	*could have*
Users can save the animation from CPR24 locally to their device, using a animated vector format (e.g. EVA format).	
CPR26	*could have*
Users can remove previously stored animations from their device.	
CPR27	*should have*
Users can view the mixing performance of the mixing protocol in a graph.	
CPR28	*should have*
Users can save the mixing performance results locally on their device.	
CPR29	*should have*
Users can retrieve the mixng performance results that are stored locally on their device.	
CPR30	*should have*
Users can retrieve mixing performance results from multiple mixing protocols simultaneously, after which they are depicted in one graph.	
CPR31	*should have*
Users can remove mixing performance results that are stored on their device.	

3.2 Constraint requirements

This section contains all constraint requirements for the application. These requirements state all demands with regard to interfaces, portability, adaptability availability, security, safety, standards, resources and time scales.

CNR01	*must have*
The application runs on iOS Safari versions 6.0 and higher.	
CNR02	*should have*
The application runs on Firefox versions 20 and higher, and Google Chrome versions 26 and higher.	
CNR03	*could have*
The application runs on Internet Explorer version 10 and higher, Opera versions 12.1 and higher and Safari versions 6.0 and higher.	
CNR04	*must have*
The application runs on devices runnning on iOS versions 6 and higher.	
CNR05	*should have*
The application runs on devices runnning on Android version 4.0 and higher.	
CNR06	*could have*
The application runs on devices runnning on Windows 8.	
CNR07	*must have*
Waiting time between submitting input and receiving output is not longer than 5 seconds.	
CNR08	*should have*
Waiting time between submitting input and receiving output is not longer than 3 seconds.	
CNR09	*could have*
Waiting time between submitting input and receiving output is not longer than 1 second.	
CNR10	*must have*
The application should be easily extendable with new mixers.	

Appendices

Appendix A

Use cases

A.1 View mixing history

Goals: To view the result of a mixing run.
Preconditions: At least one mixing run must have been executed and saved.
Summary: The performance of the selected mixing run is shown, accompanied by a picture of the final result.
Priority: Should have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *View history* button.	2. A list with previously saved runs is displayed.
3. The user taps one of the runs shown in the list.	4. The performance result of the selected run and the final mixing result are shown.

A.2 Remove mixing run from history

Goals: To remove the result of a mixing run.
Preconditions: At least one mixing run must have been executed and saved.
Summary: The details of the selected run (image and performance result) are removed from the history.
Priority: Should have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *View history* button.	2. A list with previously saved runs is displayed.
3. The user taps one of the runs shown in the list.	4. The performance result of the selected run and the final mixing result are shown.
5. The user taps the *Delete* button.	6. The details of the selected run are deleted from storage.

A.3 Save mixing run image

Goals: To save the resulting image of a mixing run.
Preconditions: The user has defined an initial concentration distribution and a mixing protocol, and has pressed the submit button.
Summary: The image of the executed mixing run is stored locally on the user's device.
Priority: Should have.
Steps:

Actor actions:	*FingerPaint response:*
	1. The results of the mixing run are visualized on the device.
2. The user selects the *Save Image* option.	3. The save interface is displayed.
4. The user selects a location on his/her device to save the image.	
5. The user chooses a name for the image.	
6. The user taps the *Save* button.	7. A confirmation message is shown.
8. The user taps the *OK* button.	9. The ouput interface is displayed again.

A.4 Save mixing run performance graph

Goals: To save the performance graph of a mixing run.
Preconditions: The user has defined an initial concentration distribution and a mixing protocol, and has pressed the submit button.
Summary: The performance graph of the executed mixing run is stored locally on the user's device.
Priority: Should have.
Steps:

Actor actions:	*FingerPaint response:*
	1. The results of the mixing run are visualized on the device.
2. The user selects the *Save Performance* option.	3. The save interface is displayed.
4. The user selects a location on his/her device to save the performance graph.	
5. The user chooses a name for the performance graph.	
6. The user taps the *Save* button.	7. A confirmation message is shown.
8. The user taps the *OK* button.	9. The ouput interface is displayed again.

A.5 Save mixing run animation

Goals: To save the resulting animation of a mixing run.
Preconditions: The user has defined an initial concentration distribution and a mixing protocol, and has pressed the submit button.
Summary: The animation of the executed mixing run is stored locally on the user's device.
Priority: Could have.
Steps:

Actor actions:	*FingerPaint response:*
	1. The results of the mixing run are visualized on the device.
2. The user selects the *Save Animation* option.	3. The save interface is displayed.
4. The user selects a location on his/her device to save the animation.	
5. The user chooses a name for the animation.	
6. The user taps the *Save* button.	7. A confirmation message is shown.
8. The user taps the *OK* button.	9. The ouput interface is displayed again.

A.6 View multiple mixing performance results from previous runs

Goals: To view the mixing performance of multiple runs in the same graph.
Preconditions: At least two mixing runs must have been executed and saved.
Summary: The performance of the selected runs are shown in the same graph.
Priority: Should have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *View history* button.	2. The history interface is displayed.
3. The user selects the *View performance* option.	4. A list of previously saved runs is displayed.
5. The user selects two or more runs from the list.	
6. The user taps the *Submit* button.	7. The mixing performances of the selected runs are displayed in one graph.

A.7 Define a mixing geometry and mixer

Goals: To define a mixing geometry and mixer.
Preconditions: none.
Summary: The user selects the geometry used for the mixing process.
Priority: Could have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *start mixing* button.	2. Go to the the mixing interface.
3. The user selects mixing geometry of choice from pop-up menu (rectangle, square, circle or journal bearing).	4. Closes menu, opens new menu with possible mixers associated with the mixing geometry
5. The user selects mixer of choice from select/pop-up menu.	6. Display blank initial distribution menu, conform chosen mixing geometry and mixer

A.8 Load a predefined distribution

Goals: To load a predefined distribution.
Preconditions: none.
Summary: The user loads a predefined distribution.
Priority: Could have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *Load* C_0 button.	2. Display select/pop-up -menu with predefined distributions for selected geometry.
3. The user on the predefined distribution of choice	4. Display canvas with selected distribution.

A.9 Define an initial distribution

Goals: To define an initial distribution.
Preconditions: Mixing geometry + mixer have been chosen.
Summary: The user defines the initial concentration distribution
Priority: Must have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *Color button* (white or black).	2. Gives visual feedback on selected colour.
3. The user moves finger on screen to define initial distribution.	4. Gives real-time visual feedback of selected area in the selected colour.
5. Repeat 1 & 3 until satisfied	6. Repeat 2 & 4 accordingly.
7. Optional: The user taps the *save as predefined distribution* button	8. Saves the current distribution as a predefined distribution for the selected geometry & mixer.
9. The user does or does not tick the *Intermediate steps* tickbox	10. Saves preference

A.10 Define mixing protocol (1)

Goals: To define the mixing protocol
Preconditions: Initial concentration distribution has been defined.
Summary: The user defines the mixing protocol
Priority: Must have.
Steps:

Actor actions:	*FingerPaint response:*
	0. Disables Intermediate steps checkbox. It cannot be edited anymore.
1. The user taps on *stepsize display* (D)	2. Gives visual feedback stepsize has been selected
3. The user taps the adjacent increment/decrement buttons (with 0.1 accuracy)	4. Increments/decrements value in the display (with 0.1 accuracy).
5. The user moves his/her finger (left(L) to right(R) or right to left) adjacent to geometry to indicate movement	6. * Case Rectangle/square : Interprets it as L to R or R to L movement of top or bottom wall based on proximity. Gives feedback about which movement has been selected. * Case Circle / Journal bearing: Interprets as clockwise/anti-clickwise movement of (1st or 2nd) circle based on proximity. Gives feedback about which movement has been selected.
7. Repeat 1-3-5 until satisfactory parameters have been selected	8. Repeat 2-4-6 accordingly. The most recent parameter value is applied.

Remark: 1-2-3-4 and 5-6 can also be executed in vice-versa order.

A.11 Define mixing protocol (2)

Goals: To define the mixing protocol
Preconditions: Define mixing protocol(1) and *Intermediate steps* has been ticked.
Summary: The user defines the mixing protocol
Priority: Must have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps the *Mix now!* button	2. Adds movement to protocol-log . Computes result of applying given movement to the distribution and displays it on the canvas.
3. Repeat **Define mixing protocol (0)** + (1) until satisfied	4. Give according responses
5. (optional): Click on Save protocol	6. Saves the protocol-log and (intermediate) visualisation on local storage.

A.12 Define mixing protocol (3)

Goals: To define the mixing protocol
Preconditions: Define mixing protocol(1) and *Intermediate steps* has NOT been ticked.
Summary: The user defines the mixing protocol
Priority: Must have.
Steps:

Actor actions:	*FingerPaint response:*
1. The user taps on the *Add to protocol!* button	2. Adds selected movement to protocol-log .
3. Repeat **Define mixing protocol (1)** + (1) until satisfied	4. Give according responses
5. The user taps on the emphShow mixture! button	6. Computes result of applying all movement in the protocol-log to the initial concentration distribution and displays it on the canvas.
7. (optional): The user taps on the *save protocol* button	8. Saves the protocol-log and (intermediate) visualisation on local storage.