Computing and Informatics, Vol. 29, 2010, 3–25

ENHANCING USE CASES WITH SCREEN DESIGNS. A COMPARISON OF TWO APPROACHES

Łukasz Olek, Mirosław Ochodek, Jerzy Nawrocki

Institute of Computing Science
Poznan University of Technology
ul. Piotrowo 2
60-965 Poznan, Poland
e-mail: {Lukasz.Olek,Miroslaw.Ochodek,Jerzy.Nawrocki}@cs.put.poznan.pl

Revised manuscript received 16 October 2009

Abstract. This paper presents a language called ScreenSpec that can be used to quickly specify screens during the requirements elicitation phase. Experiments and case studies presented in this paper show that it is easy to learn and effective to use. ScreenSpec was successfully applied in 9 real projects. Visual representation generated from ScreenSpec can be attached to requirements specification (e.g. as adornments to use cases).

Keywords: Use cases, GUI design, prototyping, screenspec

Mathematics Subject Classification 2000: 68N01, 68N30, 68U35

1 INTRODUCTION

Use cases are the most popular way of specifying functional requirements. A survey published in IEEE Software in 2003 [13] shows that over 50 % of software projects elicit requirements as use cases or scenarios. Use case is a good way of describing interaction between user and system at the high level of abstraction, so maybe now the number can be even higher. At the same time many practitioners (in about 40 % of projects [13]) draw user interfaces to visualise better how the future system will behave. This is wise, since showing user interface designs (e.g. prototypes [7, 16, 21, 22], storyboards [9]) together with use cases helps detect problems with

requirements [14][1]. Unfortunately, details of the user interface can clutter use-case description and should be kept apart from the steps [5, 6]; however, they can be attached to use cases as adornments [5].

Much has been said about writing use cases [5, 6, 8, 10, 19] (e.g. how to divide them into main scenario and extensions, what type of language to use); however, it is not clear how to specify UI details as adornments. Practitioners seem to either draw screens in graphical editors and attach graphical files to use cases, or just describe them using natural language. Both approaches have advantages and disadvantages. The graphical approach is easier to analyse by humans; however, more difficult to prepare and maintain. On the other hand, the textual approach is much easier to prepare, but not so easy to analyse.

The goal of this paper is to propose a simple formalism called ScreenSpec to specify user interface details. It has both advantages of the approaches mentioned earlier: as a textual approach it is easy to prepare and maintain, and can be automatically converted to the graphical form (attached to use cases as adornments can stimulate readers visually). Currently the language is limited to describe user interface of web applications.

It is easy to propose a new formalism, but it is much more difficult to prove that it is useful. This paper presents a case study, where ScreenSpec has been successfully used in 9 real projects. An investigation was carried out to find out how much effort is needed to use ScreenSpec, and how much time does it take to learn how to use it. Finally, an experiment was conducted in order to compare the efficiency of using ScreenSpec versus a graphical tool – Microsoft Visio.

The plan of this paper is as follows. Section 2 describes chosen approaches to UI specification that are widely used. Section 3 describes the ScreenSpec language. Section 4 describes a way to generate graphic files representing particular screens from ScreenSpec. Section 5 presents how the visual representation of screens can be embedded in requirements documents: as adornments, or as a mockup. Section 6 describes case study and experiment that were conducted to verify whether ScreenSpec is complete enough and flexible to be used for specifying screens of real applications and how much effort does it take to specify screens at requirements elicitation phase.The whole paper is concluded in Section 7.

2 RELATED WORK

There are many approaches to screen specification. They can be roughly divided into two groups: user interface specification languages and screen sketching tools. Both have their advantages and disadvantages.

The modern UI specification languages, such as XForms [17] or XUL[2] are development-focused. They are used as a flexible way for coding user interface. They allow to formally specify high-fidelity screens. Unfortunately, they require

[1] See experiment conclusions in the section “Mockup helps to unveil usability problems”.

a substantial effort to describe a screen, and thus are not suitable to be used for quick screen sketching.

There are other technologies that are getting more and more popular nowadays, like for instance MDSD (Model Driven Software Development) approaches, that provide an ability to generate a whole application (with the user interface) from a set of models (e.g. WebML [3], UWE/ArgoUWE [4]). Unfortunately, this approach still seems to require too much effort, to be successfully used at the requirements elicitation phase. There are companies using such approach, that Poznan University of Technology cooperates with. According to their experience it takes at least several hours to describe a single use case with MDSD models. It is definitely too long to be used at requirements elicitation phase, so they use generic text editors to specify use cases and screens.

On the other hand, there are tools for sketching the user interface (such as Microsoft Visio). They allow to draw screens quickly in the visual form. This approach seems to be most often used in practice to sketch user interface.

3 SCREENSPEC – LANGUAGE FOR SCREEN SPECIFICATION

ScreenSpec is used to specify the structure of the user interface. Since this approach is supposed to be used at early stages of requirements elicitation phase, it would be wise to focus on the structure of screens and information exchanged between user and system, rather then on such attributes like colours, fonts, or layout of components. This is called a low-fidelity approach ([18, 24]) and is used in ScreenSpec. There were research experiments conducted to compare low-fidelity and high-fidelity approaches (e.g. [23, 24]). Researchers concluded that there is no significant difference in the number, type, and severity of usability issues found by reviewers of low- and high-fidelity prototypes.

It is best to explain how ScreenSpec specification looks like on a simple example. Let us imagine a screen for sending e-mail messages (see Figure 1). It contains a field for entering the title, the content of a message, radio buttons for selecting format and some buttons.

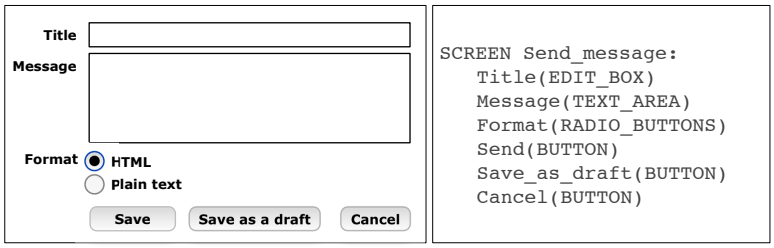

Fig. 1. A simple screen and its corresponding specification in ScreenSpec

3.1 Introduction to the Language

3.1.1 Screens

ScreenSpec specification consists of a set of screens. The definition of each screen starts with the `SCREEN` keyword followed by screen identifier (each screen must have an unique ID). The following lines are indented and describe components that belong to the screen (see Figure 2). In the simplest form of ScreenSpec, which can be used at early stages of requriements elicitation, the components have only their names, without the specification of the types. The formal grammar of the ScreenSpec language is presented in Section 3.3.

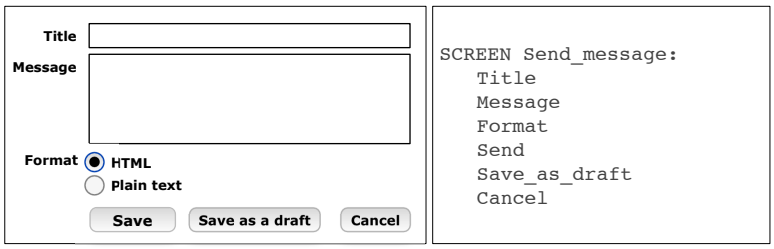

Fig. 2. A simple screen and the description of its structure in ScreenSpec

3.1.2 Basic Components

Basic components are mostly simple widgets known from the HTML language. They are specified by putting a component type in parentheses after the component name. Component type can be one of:

- `BUTTON` – represents an HTML button (`<input type="submit"/>`, `<input type="button"/>` or `<input type="reset"/>`),
- `LINK` – represents an HTML link (`<a href="URL">Link title</a>`),
- `IMAGE` – represents an HTML image (`<img src="URL"/>`),
- `STATIC_TEXT`, `DYNAMIC_TEXT` – represents a text fragment on a page, that is meaningful from the testing perspective; the `STATIC_TEXT` is a text that is each time the same (e.g. a comment or an instruction), and the `DYNAMIC_TEXT` is a text calculated dynamically by the system (e.g. invoice total),
- `EDIT_BOX` – represents an HTML edit input (`<input type="edit"/>`),
- `PASSWORD` – represents an HTML password input (`<input type="password"/>`),
- `COMBO_BOX` – represents a drop-down HTML select component (`<select><option>...</option>...</select>`),

- `LIST_BOX` – represents an HTML select component with an ability to select more than one option (`<select multiple="multiple"><option>...</option>...</select>`),
- `RADIO_BUTTONS` – represents a group of HTML radio buttons (`<input type="radio"/>`),
- `CHECK_BOXES` – represents a group of HTML check boxes (`<input type="checkbox"/>`),
- `CUSTOM` – used to denote a component that is not included in the standard set of HTML components (e.g. date pickers, maps, etc.); mockup has no underlying functionality, so it cannot render such components; they will be visualised as empty rectangles.

The screen from Figure 2 supplemented with component types is presented in Figure 1. Component types are defined in parentheses after their names.

3.1.3 Groups – Structured Components

Groups are component structures, used to specify lists, tables, or just to group the components visually on the screen in one section. A group is specified with a name followed by a colon, and a set of indented lines – with specification of components that belong to the group (see Figure 3). Groups can be declared as simple, list or table, with a type specified in parentheses between its name and a colon. The meaning of the group type is as follows:

- `SIMPLE` (default) – such group is just used to put a couple of components in one section on the screen (e.g. section personal details can contain: name, email, etc.), but it does not provide any additional semantics to the components,
- `LIST` – its components are repeated in a list, all child components specify a single list item,
- `TABLE` – its components are repeated in rows, as a table (similar to `LIST`, but different layout).

There are two more types available for components declared inside a group: `CHECK_BOX` and `RADIO_BUTTON`. These types define a group of these components across all elements of the group, and can be used for declaring e.g. a radio button that would allow to select one row from a table (see Figure 4).

3.2 ScreenSpec Advanced Features

3.2.1 Static Values

For some components we already know their initial values at specification time. For example, a group of radio buttons that allows to choose user sex will always have

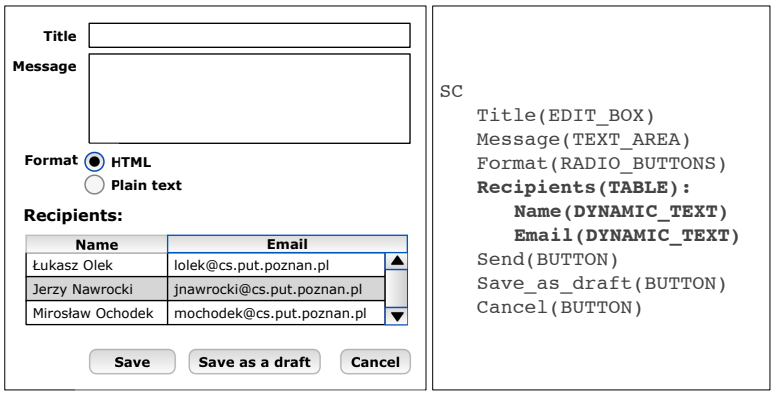

Fig. 3. The screen with a group component (table)

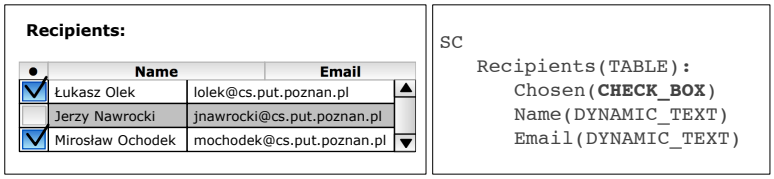

Fig. 4. Additional component types are available inside a group: `CHECK_BOX` and `RADIO_BUTTON`

two values: male and female. Static values are used to specify the values in such situations. The meaning of the static value is different for different components (see Table 1). In order to specify a static value for a component, its declaration is followed by a colon, and a list of values separated by a vertical bar "|" (see Figure 5).

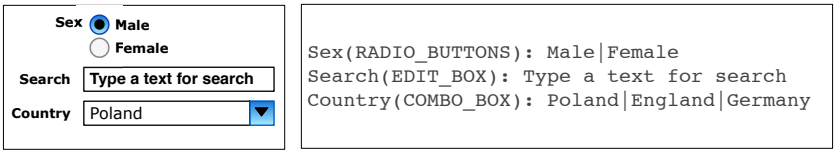

Fig. 5. Static values can define initial values for components

3.3 ScreenSpec Grammar Specification

This section presents a grammar of ScreenSpec language in the EBNF notation [25]. Since the original EBNF notation does not allow to specify indentation-based languages very comfortably, it was extended to pass arguments to non-terminal sym-

Control Type	**No. of applicable static values**	**Semantics of static values**
`BUTTON`, `LINK`	1	Static value specifies the caption of the component
`STATIC_TEXT`	1	Static value specifies the text visible for the user
`RADIO_BUTTONS`, `CHECK_BOXES`	many	Static values specify the descriptions of particular buttons. Some values can be preceded with '`=`' – these values will be initially selected (there can be one such value for `RADIO_BUTTONS` and many values for `CHECK_BOXES`)
`COMBO_BOX`, `LIST_BOX`	many	Static values specify the options for the components. Some values can be preceded with '`=`' – these values will be initially selected (there can be one such value for `COMBO_BOX` and many values for `LIST_BOX`)
`EDIT_BOX`, `TEXT_AREA`	1	The static value is an initial value for the component
`PASSWORD`, `DYAMIC_TEXT`	–	No static values can be applied to these components

Table 1. The semantics of static values depending on control type

bols. These arguments are used to count a proper number of indents, and are passed in brackets. The grammar assumes the following terminal symbols will be recognized by lexer:

- `value` – any string that does not contain the "—" character,
- `identrifier` – a string that can contain letters, digits, "`-`", "`_`",
- `indentation` – the tab character,
- `line break` – the new line character(s).

```
specification = screen *;
screen = "SCREEN", identifier, line break,
    (component (1)) *;
component (i) = component indication (i) | simple component (i)
    | group component (i);
component indication (i) = indentation (i), identifier, line break;
simple component (i) = indentation (i), identifier,
    "(", component type, ")", [":", static values], line break;
static values = value ("|", value)*;
component type = "BUTTON" | "LINK" | "IMAGE" | "STATIC_TEXT"
    | "DYNAMIC_TEXT" | "EDIT_BOX" | "COMBO_BOX" | "RADIO_BUTTONS"
    | "RADIO_BUTTON" | "LIST_BOX" | "CHECK_BOXES" | "CHECK_BOX"
    | "CUSTOM";
```

```
group component (i) = indentation (i), identifier,
    ["(", group type, ")"], ":", line break,
        (component (i+1), line break) +;
group type = "SIMPLE" | "LIST" | "TABLE";
```

3.4 Evolutionary Approach to Screen Specification

ScreenSpec is designed to be used by analyst at requirements elicitation phase. This phase is exploratory, which means that change-involving decisions are made frequently. Thus we propose to use ScreenSpec in an incremental way. At the beginning an analyst can just roughly describe the structure of information at particular screen, and add more details later (after getting a confirmation from a customer that it is correct). ScreenSpec has 3 levels of details:

L1 Component names – need to be specified at the beginning.

L2 Types of controls and groups – specifies types of information connected with each screen.

L3 Static values.

These levels can be mixed throughout the specification process: some screens can be specified at one level of details, and other screens can be specified at other levels.

4 VISUAL REPRESENTATION OF SCREENS

ScreenSpec can be authored using a dedicated tool. This is a simple editor that detects each change, and automatically regenerates graphics files (PNG) that can be attached to requirements documents. The generator uses simple rules to transform ScreenSpec to visual representation:

1. For each component:
    - `EDIT_BOX`, `COMBO_BOX`, `LIST_BOX`, `CUSTOM` – a label (equal Component ID) is displayed on the left side of the control, the control's value is taken from the defined static value, or it is left empty. CUSTOM component is displayed as the `EDIT_BOX`.
    - `BUTTON`, `LINK` – displays a control with a caption equal to the defined static value, or component ID.
    - `STATIC_TEXT`, `DYNAMIC_TEXT` – displays a piece of text equal the static value or component ID.
    - `RADIO_BUTTON`, `CHECK_BOX` – displays a control followed by a label (label's value equals the static value or component ID)
    - `IMAGE` – displays a label on the left (equal to component ID) and an empty image frame on the right.

2. For each group:

   - `SIMPLE` – a header and a frame is created, all children components are placed inside this frame.
   - `LIST` – a header and a frame is created. In the frame 3 rows are displayed (this visualises that a list can have more elements): two rows having the child components, and the third one containing "..."
   - `TABLE` – is similar to a `LIST`, however a new table column is created for each child component. Its label is displayed in the table header rather than on the left (near its control).
   - `TREE` – is similar to a `LIST`, but for each row a nested and smaller list is displayed.

The following example (Figure 6) shows a visual representation of a simple screen specified in ScreenSpec.

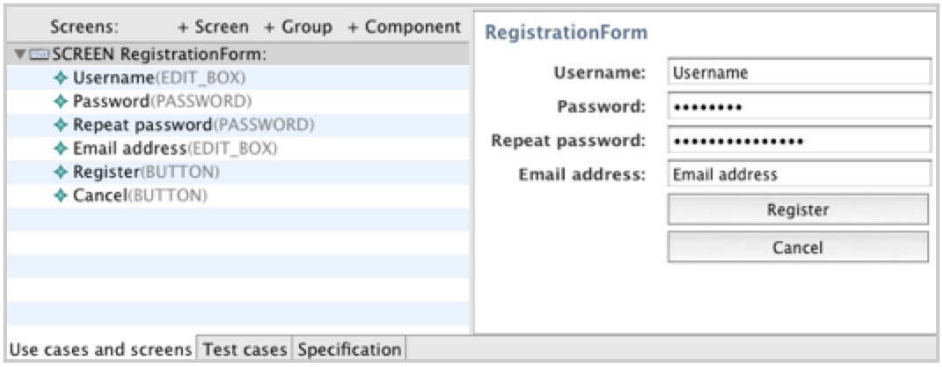

Fig. 6. A screenshot showing the ScreenSpec editor with a generated visual representation for a registration screen

5 SCREENSPEC MEETS USE CASES

Visual screens generated from ScreenSpec can be directly inserted into requirements specification in adornments section of particular use cases. Having up-to-date graphic files allows to update the specification easily, because many modern text editors allow to link with external files, and update them each time the document is opened (e.g. Microsoft Word, OpenOffice).

5.1 Mockup

Mockup is an interesting artefact created by connecting screens to particular steps of use cases. It is rendered as a simple web application that can display both use

cases and screens at the same time. Use case (displayed on the left side) shows the interaction between an actor and a system (see Figure 7). After selecting particular step, an according screen is displayed (on the right side). This artefact seems to be useful in practice, initial feedback from commercial projects using mockups is very positive.

It is difficult to connect screens to use case steps in generic text editor, so a dedicated tool called UC Workbench [12] was developed at Poznan University of Technology.

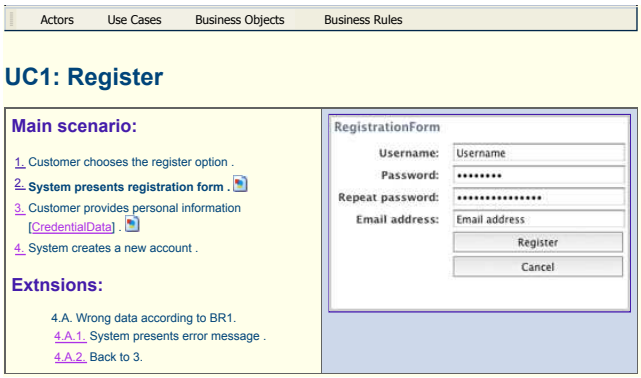

Fig. 7. A screenshot of Mockup – showing use case with corresponding screens at the same time

6 EXPERIENCE WITH SCREENSPEC

6.1 Specifying Screens for the Real Projects – Case Studies

Analysts usually use word processors and sheets of papers to author requirements. Keeping it in mind, it seems that introducing formalised requirements models can be risky. It may happen that some of the features might be too difficult to describe with the formalised model.

To make sure that the ScreenSpec formalism is complete and flexible enough to be used for describing real systems, nine case studies were conducted. They included a large variety of projects. Some of them were internally-complex (large number of sub-function[2] requirements), with a small amount of interaction with a user (e.g. Project A, Project C). Others were interaction-oriented, with a great number of use cases and screens (e.g. Project D, Project G). First 6 projects were selected from the Software Development Studio course at Poznan University of Technology. These projects were developed for external customers by students of the Master of Science

[2] After Cockburn [6]: a sub-function requirement is a requirement that is below the main level of interest to the user, i.e. “logging in”, “locate a device in a DB”.

in Software Engineering. Students were successfully using ScreenSpec approach to specify screens. They also raised some minor suggestions for ScreenSpec language, and small simplifications were introduced afterwards. Then screens for 3 commercial projects were also written using ScreenSpec language. In both cases all screens were successfully specified.

It seems that the number of lines of code (LOC) per screen may differ depending on the screen complexity. In analysed projects average LOC per Screen varies from 3.0 to 14.5 (see Table 2).

Project	Business Level UCs	User Level UCs	Sub-function UCs	Screens	Average LOC /Screen	Total LOC /Screen
Project A		4	2	4	14.5	58
Project B	3	13	2	5	9.4	47
Project C		5		4	3.0	12
Project D		16		27	4.7	128
Project E		4		7	3.9	27
Project F	1	3	2	3	13.0	39
Project G		44	39	92	9.5	917
Project H	2	12		7	5.1	36
Project I	8	57		72	14.3	1 026

Table 2. Nine projects selected for the case study

6.2 ScreenSpec Efficiency Analysis

Although an average amount of code required to specify a screen with ScreenSpec seems to be rather small, two important questions arise:

- Q1: how much effort is required to specify[3] a screen?
- Q2: how much time is required to learn how to use ScreenSpec?

The second question is also important because practitioners tend to choose solutions, which provide business value and are inexpensive to introduce. If an extensive training is required in order to use ScreenSpec efficiently, there might be a serious threat that the language will not be attractive to the potential users.

In order to answer these questions, a controlled case study was conducted[4]. Eight participants were asked to specify sequence of 12 screens coming from the real

[3] The term "specifying" is understood here as the process of transcribing the vision of the screen into the ScreenSpec code.

[4] The case study is labeled here as controlled, because the methodology was similar to that used in case of controlled experiments; however, the nature of questions being investigated refers rather to the "common sense" than to some obtainable values (e.g. compare average learning time, to the one which is acceptable for the industry).

application (provided as the series of application screenshots). The time required for coding each of the screens was precisely measured (up to the seconds). The code was written manually on sheets of paper. The participants were also asked to copy a sample screen specification, in order to examine their writing speed. Before they started to specify screens, they had been also introduced to the ScreenSpec during the 15-minutes lecture, and each of them was also provided with a page containing the ScreenSpec specification in a nutshell. All materials provided to participants are published at [1].

6.2.1 Descriptive Analysis and Data Clearing

During the completion of each task (single screen specification) two values were measured:

- time required to finish the task
- lines of code developed to specify the screen.

Screens specifications developed by participants differed in respect to their size, because they were specified only on the basis of the screen-shots, which were perceived slightly differently by different people. Moreover, some of the ScreenSpec structures might be used optionally. The detailed results of the case study are presented in Table 3.

Before proceeding to the further analysis, results for all tasks were carefully analysed in order to find potential outliers. The task was marked as suspicious if the variability in lines of code provided by participants was high (or there were outlying observations). According to the box plots presented in Figure 8 tasks 1, 4, 5, 7, 8, 9, 10, 11, 12 were chosen for further investigation in order to find out the reasons for the LOC variability. It turned out that tasks 4 and 8 were ambiguous, because in both cases there were two possible interpretations of the screens semantic. Moreover, the amount of code required to specify each of two versions differed visibly. Therefore those tasks were excluded from the further analysis.

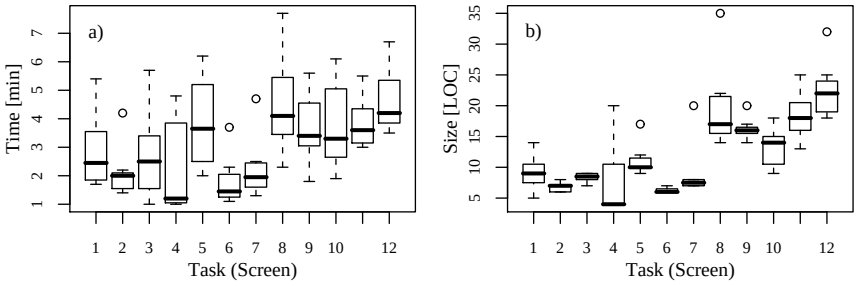

Fig. 8. Variability of effort and size of code (LOC) for each task, *box-plots presenting variability of a) effort for each task, b) lines of code for each task*

Time [min]											
Participant	Sample Screen	1	2	3	5	6	7	8	9	11	12
P1	0.9	4.3	2.0	2.0	6.2	2.3	4.7	7.7	3.3	5.5	6.3
P2	1.6	2.8	2.0	3.6	3.6	1.2	2.0	4.1	3.1	3.1	4.2
P3	1.2	1.8	2.0	3.0	4.6	1.5	2.4	6.2	4.8	3.9	4.4
P4	1.3	1.7	1.4	5.7	2.5	1.8	1.9	3.3	3.5	4.7	3.9
P5	1.0	2.4	2.2	1.0	2.5	1.4	1.8	3.6	4.3	3.0	4.2
P6	1.0	2.5	1.6	1.5	3.7	1.3	1.4	4.1	3.0	3.3	3.8
P7	1.0	1.9	1.5	1.6	2.0	1.1	1.3	2.3	1.8	3.2	3.5
P8	1.6	5.4	4.2	3.2	5.8	3.7	2.5	4.7	5.6	4.0	6.7
Mean	1.2	2.9	2.1	2.7	3.9	1.8	2.3	4.5	3.7	3.8	4.6
SD	0.3	1.3	0.9	1.5	1.6	0.9	1.1	1.7	1.2	0.9	1.2
Lines of code - LOC											
Participant	Sample Screen	1	2	3	5	6	7	8	9	11	12
P1	8	14	6	8	17	7	20	35	20	25	32
P2	8	9	7	9	12	6	8	21	16	18	19
P3	8	8	6	9	10	6	7	22	16	18	25
P4	8	7	6	9	10	6	7	15	17	20	19
P5	8	11	8	8	11	6	8	16	16	15	23
P6	8	9	7	8	9	7	8	17	14	17	23
P7	8	10	7	9	10	6	7	17	15	21	21
P8	8	5	7	7	10	6	7	14	16	13	18
Mean	8.0	9.1	6.8	8.4	11.1	6.3	9.0	19.6	16.3	18.4	22.5
SD	0.0	2.7	0.7	0.7	2.5	0.5	4.5	6.8	1.8	3.7	4.5

Table 3. Effort and lines of code for each participant and task (*sample screen refers to the task measuring participants writing speed*)

6.2.2 Productivity Analysis

Based on the effort and code size measured for each task and each participant, the productivity factor can be calculated. It will be defined here as the time required to write a single line of code. It can be calculated according to Equation (1).

$$PROD = \frac{Effort}{Size}, \tag{1}$$

where:

- $PROD$ is a productivity factor understood as the number of minutes required to develop a single line of code
- $Effort$ is the effort required to complete the task (measured in minutes)
- $Size$ is the size of code developed to specify the screen (measured in LOC).

The effort measured during the case study consists of two components: 1) the time required for thinking and 2) writing down the screen. It would be difficult to precisely measure both of them, however knowing the writing speed of each participant (see Equation (2)) it is possible to calculate the approximate effort spent only on thinking. It can be further used to estimate cognitive productivity factor (see Equation (3)). It can be understood as a productivity of thinking while coding the screen. It is independent from the tool (the effort needed mentally to produce the screen-specification code).

$$V_{writing} = \frac{Size_{sample}}{Effort_{sample}}, \tag{2}$$

where:

- $V_{writing}$ is the writing speed (measured in LOC per minutes)
- $Effort_{sample}$ is the effort required to copy the code for the sample screen (measured in minutes)
- $Size_{sample}$ is the size of the code for the sample screen – 8 LOC.

$$PROD_{cognitive} = \frac{Effort - (Size/V_{writing})}{Size}, \tag{3}$$

where:

- $PROD_{cognitive}$ is an estimation of cognitive productivity factor understood as a number of minutes spend on thinking in order to produce a single line of code
- $Effort$ is the effort required to complete the task (measured in minutes)
- $Size$ is the size of code to specify the screen (measured in LOC)
- $V_{writing}$ is writing speed (measured in LOC per minute).

Cognitive and standard productivity factors were calculated for each task and participant. The chart presenting mean values for each task is presented in Figure 9.

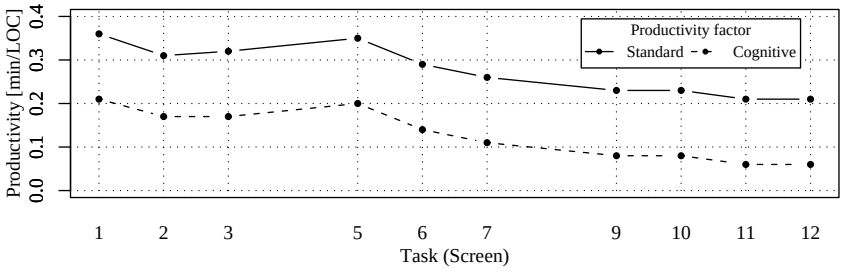

Fig. 9. Mean (cognitive and standard) productivity factors for each task (screen)

6.2.3 Q1: How Much Effort Is Required to Specify a Screen?

If the mean productivities from the first and the last task are compared, it would mean that the average beginner produces around 2.81 LOC/minute while person with some experience 4.7 LOC/minute (this of course may vary depending on the screen complexity). This means that the total effort for specifying all of the screens for the largest project included in the case studies – Project I (72 screens with total of 1026 LOC of screen specifications) would vary from 3.6 to 6.1 hours depending on analyst skill. Furthermore, an average screen size is between 8 and 9 LOC (average from Table 2), which could be specified in less than 2 minutes (for experienced analyst, and around 3 minutes for the beginner). Therefore, it seems that the ScreenSpec notation might be used directly during the meetings with customer. It is also worth mentioning that if there was an efficient editor available (with high usability), the productivity factor for potential user would be closer to the cognitive one. This means that a 8 LOC screen would be specified in about 30 seconds.

6.2.4 Q2: How Much Time Is Required to Learn How to Use Screenspec?

By looking at the productivity chart presented in Figure 9, the learning process can be investigated. The ratio between productivity factors calculated for the ending and beginning task is 1.69. In addition it seems that after completing 8–10 tasks the learning process saturates. Therefore, it seems that participating in a single training session which includes a short lecture and ten practical tasks (about an hour) should be enough to start using ScreenSpec effectively.

An interesting observation is regarding the task number 5, because the productivity factor suddenly increased at this point (more time required to produce one line of code). This issue was further investigated, and the finding was that the screen for that task contained interactive controls, which appeared for the first time in the training cycle (edit boxes, check boxes etc.). Thus an important suggestion for a preparation of the training course would be to cover all of the components available in the ScreenSpec language.

6.3 Comparison Between ScreenSpec and Visual Graphical Editors

As mentioned already, approaches to specify screens can be divided into two main groups. The first of them is to present structure of the screen by enumerating elements being displayed. Alternative approach is to use graphical editor to prepare visual representation of the application screen.

ScreenSpec belongs mainly to the first group (however, it can be also transformed to the simplified visual representation). A benefit of using structured text to specify screens rather than drawing them in graphical editor is that the text can be easily modified. This is important especially if we consider how unstable are the requirements at the initial software project stages.

Therefore we would like to investigate which approach to specify screens (ScreenSpec or graphical editor) is more suitable to be used at the early stages of projects. In order to complete this task we would like to find answers to two research questions:

- Q3: is ScreenSpec more efficient for specifying new screens than graphical editor?
- Q4: is ScreenSpec better for modifying existing screens than graphical editor?

In order to be able to answer those questions we decided to conduct the experiment, in which participants were asked to specify screens using ScreenSpec and high-quality graphical editor.

6.3.1 Experiment Design

The *independent variable* of the experiment was a choice of tools for screens specification. One of the tools was a prototype ScreenSpec editor. Then we had to choose a representative graphical editor. We decided to use Microsoft Visio with a set of stencils especially designed to draw low-fidelity sketches of screens.

The *dependent variable* analysed in the experiment was the effort required to prepare a sketch of a screen based on the screen-shot from the real application. We prepared a set of 22 tasks. This included 2 warm-up tasks, 7 tasks which goal was to prepare a new screen and 13 tasks which aim was to modify existing screens (add, remove, update screen components or divide a screen into a set of sub-screens).

Participants of the experiment were 3^{rd}-year CS students (127 people), who were completing the 2^{nd} semester of the Software Engineering course. They were randomly assigned to one of two groups:

- ScreenSpec (SS) – 66 participants
- Microsoft Visio (Visio) – 61 participants

6.3.2 Experiment Operation

The experiment was executed on March 2009 at Poznan University of Technology. Each participant had access to a web-application developed for the purpose of the experiment. It served for description of the tasks and stored screen sketches developed by participants. The system was also measuring tasks completion times. Each participant had also access to a presentation with a short tutorial.

The experiment time was fixed to 1.5 hours. Participants started by familiarising themselves with the tutorials. After they finished, they were asked to complete 2 simple warm-up tasks which were not considered during the analysis. Their goal was to make participants familiar with the editors and web-application used to control the experiment. As soon as they completed this stage they started solving the rest of 22 tasks until they finished all of them or the time has finished. During the experiment participants were supervised by a teacher, who was present in the classroom all the time (teachers were not allowed to provide any hints).

6.3.3 Analysis

Analysis started from assessing correctness of screens specified by participants. After reviewing all solutions, 27 Visio and 186 SS screens were rejected. The reason for relatively large number of rejections for the SS group was that participants were specifying screens based on the application screen-shots. Thus the semantics of the screens was missing. Such lack of knowledge was especially important in case of structured components (e.g. list of product properties). Participants could treat them as dynamic lists or explicitly present each property. In most cases those two approaches yield different number of ScreenSpec LOC. As a result solutions different from the reference one were rejected.

As the next part of analysis we applied descriptive statistics (short summary is presented in Table 4) and visualized collected data using box-plots (see Figure 10). Each outlying observation was investigated once again.

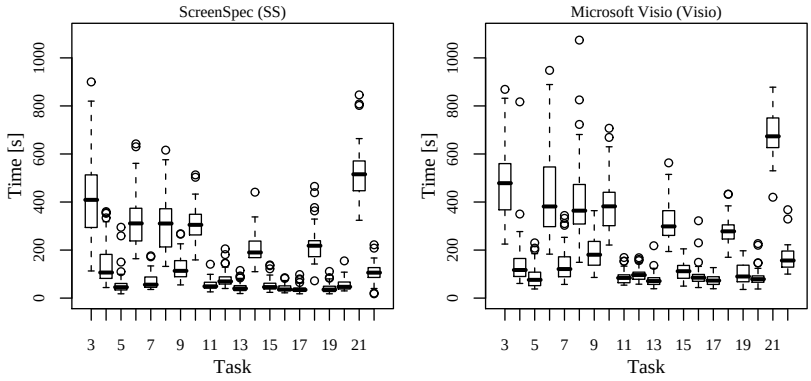

Fig. 10. Box-plot presenting completion times for both groups after data clearing

After visualizing the results of the experiment we suspected that most of the samples are not derived from the normally distributed population. This was confirmed by the Shapiro-Wilk test [20] (significance level α was set to 0.01). Only in case of two tasks (21 and 22) both samples seemed to be derived from the normal distribution. Therefore, we decided to use non-parametric testing procedures.

In order to be able to answer questions Q3 and Q4 the central tendencies of effort required to complete each task have to be compared. Because the assumption of normally distributed populations was violated we decided to use medians to formulate the following hypotheses (for each task):

Null hypothesis – the median effort required to complete i^{th} task is equal for both groups ($H_0^i : \Theta_{SS} = \Theta_{Visio}$)

Alternative hypothesis – the median effort required to complete i^{th} task is lesser for the group using ScreenSpec ($H_1^i : \Theta_{SS} < \Theta_{Visio}$).

		ScreenSpec (SS)		Microsoft Visio (Visio)	
Task	Type	Screens	Median Effort [s]	Screens	Median Effort [s]
3	new	56	409	60	478.5
4	modify	66	106.5	61	117
5	modify	66	45.5	61	76
6	new	31	311	60	381.5
7	modify	30	56	61	121
8	new	42	310.5	60	364
9	modify	42	113.5	58	180.5
10	new	49	305	59	382
11	modify	64	49	58	84
12	modify	64	69.5	55	96
13	modify	63	40	57	71
14	new	55	190	54	298.5
15	modify	53	46	51	112
16	modify	53	36	51	85
17	modify	54	34.5	50	73
18	new	39	218	45	278
19	modify	40	35.5	41	90
20	modify	49	47	43	79
21	new	38	515.5	20	673.5
22	modify	39	106	12	156.5

Table 4. Summary of the experiment results (*Type: new – participants were supposed to specify a new screen, modify – participants had to introduce modifications to the last "new" screen; Screens: is a number of solutions accepted for the task and group*)

We applied the Mann-Whitney test [11] to investigate hypotheses. The significance level α was set to 0.01. As a result the null hypothesis was not rejected only for the task number 4 (which was the first modification task). In case of other tasks median effort required to complete tasks was significantly lesser for the group using ScreenSpec.

6.3.4 Threats to Validity

The most important threats to *internal validity* of this study are:

Level of commitment. Because participants were students there is a threat regarding their motivation and commitment. We were trying to mitigate this problem by introducing marks for performing the tasks (based on completion time and correctness). We also decided to use fixed time (1.5 h) to avoid the risk of decreasing productivity due tiredness.

Familiarity with tools. Another issue is difference in experience with using the tools. Although the participants had never used ScreenSpec before, they were familiar with various graphical editors.

Objectivness of tasks descriptions. There is a problem with providing description of the screen in such form that it will not favour any of the tools. We decided to use screen-shots from the real application. It seems that this form of presentation is more favourable for graphical editor, because one can make a copy of the screen without understanding its meaning. However, in the real environment the analyst has to understand the semantic of the screen before he/she is able to specify it.

The most important threats to *external validity* of this study are:

Students instead of practitioners. In this experiment participants were students, although the method is supposed to be used by the members of software development teams. However, activities in the experiment did not involve analytical skills and were limited only to the preparation of the screen designs.

Quality of sketches. In case of graphical representation participants were supposed to use low-fidelity approach. Although low-fidelity sketch presents a simplified version of the screen, it still should be done tidily if one would like to share such screen design with the customer. This refers mainly to components such as alignment, size etc. In case of the experiment screens sketches were not rejected due to such issues as long as they were correct.

Usability of the tool. Usability of the tools chosen for the exepriment could influence the productivity of the group. In case of graphical editors, we chose Microsoft Visio, which is a top class editor, however in case of ScreenSpec we had only a simple prototype editor. Therefore results could differ if MS Visio was compared to equivalently good editor for ScreenSpec.

6.3.5 Q3: Is ScreenSpec More Efficient Than Graphical Editor?

The role of screen sketches in the early stages of requirements elicitation phase is to present the structure and semantics of screens. This can be done using both structured text (e.g. ScreenSpec) or screen images (e.g. MS Visio). However, it would be beneficial if analyst could specify a screen "online" during the meeting with the customer in order to receive immediate feedback.

Therefore, the time required to prepare a screen should be as short as possible. In case of the experiment for all tasks involving specifying a new screen, the group using ScreenSpec was faster. The ratio between median time required to specify a new screen using ScreenSpec and MS Visio was 0.79 (for all tasks difference was statistically significant).

6.3.6 Q4: Is ScreenSpec Better for Altering Screens Than Graphical Editor?

From the practical point of view it is more important to investigate how much effort is required to alter the previously specified screen.

Although the initial sketch of the screen is prepared once only, it can be further modified frequently as a result of changes in requirements. From our experience this is the main drawback of using graphical editors for specifying screens. In most cases simple modifications like reordering, adding, or removing controls can be time consuming. In case of "modification" tasks the ratio between median time required to alter a screen using ScreenSpec and MS Visio was 0.57 (for 12/13 tasks difference was statistically significant).

7 CONCLUSIONS

User interface designs are often attached to use cases as adornments, because it helps understand the requirements by non-IT people. However, it is not clear how to specify UI details. In this paper we proposed a language called ScreenSpec that can be used for this purpose. ScreenSpec is a formalism that was thoroughly validated. It was used to describe UI in nine real software projects. ScreenSpec allows to work incrementally on screen designs, starting with the general structure of information at particular screen, and then adding more details about widgets. It is very efficient, it takes on average about 2 minutes to specify a single screen. ScreenSpec is also easy to learn, it takes about an hour for a person that has never seen ScreenSpec to become proficient in using it.

ScreenSpec seems to be especially well suited to be used during the requriements elicitation phase. This stage involves constant changes of requirements and screen designs. According to performed experiment, on the average analysts can reduce the effort required to prepare new screens by 21 % when using ScreenSpec instead of graphical editors like e.g. Microsoft Visio. What is more important when screen modifications are considered, this on-average reduction is about 43 %.

Although it is interesting to use ScreenSpec at requirements elicitation stage, it could be even more interesting to use it at later stages. One can think about generating skeleton user interface code (in XUL, SWT, Swing or other technologies), that could be refined during implementation. Appropriate research will be conducted as a future work.

Acknowledgments

Authors would like to thank the companies which cooperate with Poznan University of Technology: Polsoft and Komputronik. They find time and courage to try our ideas in practice and provide us with a substantial feedback.

The research presented at the CEE-SET '08 [15] and being part of this paper has been financially supported by the Polish Ministry of Science and Higher Education under grant N516 001 31/0269.

Additional case studies and comparison experiment have been financially supported by Foundation for Polish Science Ventures Programme co-financed by the EU European Regional Development Fund.

REFERENCES

[1] A Web Page Containing All Materials for a ScreenSpec Evaluation Case Study: `http://www.cs.put.poznan.pl/lolek/homepage/ScreenSpec.html`.

[2] Home page for Mozilla XUL. Availaible on: `http://www.mozilla.org/projects/xul`.

[3] The Web Modeling Language Home Page. Availaible on: `http://www.webml.org`.

[4] UWE – UML-based Web Engineering Home Page. Availaible on: `http://www.pst.informatik.uni-muenchen.de/projekte/uwe/index.html`.

[5] Adolph, S.—Bramble, P.—Cockburn, A.—Pols, A.: Patterns for Effective Use Cases. Addison-Wesley, 2002.

[6] Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Boston 2001.

[7] Constantine, L. L.—Lockwood, L. A. D.: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA 1999.

[8] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[9] Landay, J. A.,—Myers, B. A.: Sketching Storyboards to Illustrate Interface Behaviors. In: CHI '96: Conference companion on human factors in computing systems, New York, NY, USA, ACM Press 1996, pp. 193–194.

[10] Leffingwell, D.—Widrig, D.: Managing Software Requirements: A Use Case Approach, Second Edition. Addison-Wesley Professional, May 2003.

[11] Mann, H. B.—Whitney, D. R.: On a Test of Whether One of Two Random Variables Is Stochastically Larger Than the Other. The Annals of Mathematical Statistics, Vol. 18, 1947, No. 1, pp. 50–60.

[12] Nawrocki, J.—Olek, Ł.: UC Workbench – A Tool for Writing Use Cases. In: 6th International Conference on Extreme Programming and Agile Processes, Lecture Notes in Computer Science, Vol. 3556, June 2005, pp. 230–234.

[13] Neill, C. J.—Laplante, P. A.: Requirements Engineering: The State of the Practice. Software, IEEE, Vol. 20, 2003, No. 6, pp. 40–45.

[14] Olek, Ł.—Nawrocki, J.—Michalik, B.—Ochodek, M.: Quick Prototyping of Web Applications. In L. Madeyski, M. Ochodek, D. Weiss, and J. Zendulka (Eds.): Software Engineering in Progress, NAKOM, 2007, pp. 124–137.

[15] Olek, Ł.—Nawrocki, J.—Ochodek, M.: Enhancing Use Cases With Screen Designs. In: 3rd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2008, 2008.

[16] Pressman, R.: Software Engineering – A Practitioners Approach. McGraw-Hill 2001.

[17] Raman, T. V.: XForms: XML Powered Web Forms. Addison-Wesley Professional 2003.

[18] Rudd, J.—Stern, K.—Isensee, S.: Low vs. High-Fidelity Prototyping Debate. Interactions, Vol. 3, 1996, No. 1, pp. 76–85.

[19] Schneider, G.—Winters, J. P.: Applying Use Cases: A Practical Guide. Addison-Wesley 1998.

[20] Shapiro, S. S.—Wilk, M. B.: An Analysis of Variance Test for Normality (Complete Samples). Biometrika, Vol. 52, 1965, No. 3-4, pp. 591–611.

[21] Snyder, C.: Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces. Morgan Kaufmann Publishers 2003.

[22] Sommerville, Y.—Sawyer, P.: Requirements Engineering. A Good Practice Guide. Wiley and Sons 1997.

[23] Virzi, R. A.—Sokolov, J. L.—Karis, D.: Usability Problem Identification Using Both Low- and High-Fidelity Prototypes. In: Proceedings of the CHI Conference, ACM Press 1996.

[24] Walker, M.—Takayama, L.—Landay, J. A.: High-Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes When Testing Web Applications. In Proceedings of the Human Factors and Ergonomics Society 46th Anuual Meeting, 2002, pp. 661–665.

[25] Wirth, N.: Extended Backus-Naur Form (EBNF). ISO/IEC, 14977, 1996.

Łukasz Olek is a Ph. D. student working in the Institute of Computing Science at the Poznan University of Technology. He is doing research in the area of requirements engineering and software testing.

Mirosław Ochodek is a Ph. D. student working in the Institute of Computing Science at the Poznan University of Technology. He is mainly working in the domain of requirements engineering, software metrics, functional size measurement, and software effort estimation.

Jerzy Nawrocki received the M. Sc. degree (1980), the Ph. D. degree (1984), and the Dr. hab. degree (1994) all in informatics and all from the Poznan University of Technology (PUT), Poznan, Poland. Currently he is the Dean of the Faculty of Computing and Management at PUT, and the Secretary of IFIP Technical Committee 2: Software Theory and Practice.