

A Succinct Study on Requirements Engineering for User Interfaces

Diya Burman

Abstract—Throughout the history of engineering practices, least amount of attention has been given to the designing and development of user interfaces. Developers and customers are so focused on maximizing functionality and performance and minimizing costs for development that the usability of the application or appliance has been overlooked too often. This paper attempts to analyze some of the past and current research being done at the intersection of various fields such as Software Engineering and Human Computer Interaction with Requirements Engineering and shed some light on existing issues in user interface designing which can be dealt with via improved requirements engineering.

I. INTRODUCTION

User interface (UI) design or User Interface engineering is the process of designing user interfaces for hardware and software, with an emphasis on maximizing usability and enhancing the user experience. The goal of UI design is to make the human-computer interaction as simple, intuitive and efficient as possible, in terms of accomplishing user goals.

On the other hand, Requirements engineering (RE) refers to the process of eliciting, documenting and maintaining requirements during development of software as well as hardware systems.

So how does Requirements Engineering aid User Interface Design?

- RE sets the stage for effective user interface design by applying usability requirements analysis techniques.
- We can extract usability goals from requirements analysis and use them to drive design.
- RE allows us to apply a structured approach to user interface design.
- RE allows us to apply iterative evaluation techniques to validate designs before they are implemented.
- RE allows us to plan for and manage the use of usability engineering techniques within the overall project plan.
- RE allows us to design organizational structures and processes to foster good interface design[16][8].

In this paper, we will explore the world of UI design, namely: common mistakes we encounter in modern day applications, some of the consequences of historic UI design disasters followed by past and ongoing research in the field of software engineering (the parent field of requirements engineering) and human-computer interaction (HCI) focused at tackling usability concerns in software development.

II. COMMON APPLICATION DESIGN MISTAKES

Usability of an application is maximized when users can figure out how to operate the UI easily. The UI is self-explanatory and guides the user through the workflow. Not

following common guidelines regarding UI design sabotages the usability of an application. More often than not, applications fail because they (a) create more problems than they solve (b) implement the wrong features for a problem (c) complicate the features for the user to understand[18].

Any variation of the above three mistakes can jeopardize an app yet it is difficult to come up with strict rules for the design process. What's the problem? Have we pinpointed the right problem or simply focusing on secondary issues? What are the appropriate features? What complicating curlicues can safely be cut from those features? For every problem domain and target audience, these questions have very distinctive and very different answers.

A good approach therefore can be to base our decisions on sufficient user research than “experienced” guesses. User needs are constantly evolving and it is wiser to conduct field studies and task analysis before proceeding with system design. The design needs to be prototyped before any concrete plan is made and before resources are implemented into something which will have to be changed shortly after development initiation. Feature refinement through iterative design and swift user testing helps prevent any surprises or disappointments after product launch. Of course customers might complain about repeatedly meeting with the developing team, but having a tried and tested user interface speeds up development processes and a user doesn't have to deal with an “engineer's UI” because we all know that what's “natural” to a developer isn't so to the common man[9].

The general notion is that programmers need not interact with the customers or users. Quite to the contrary. A programmer needs to observe a sufficient number of end users to get an idea of “how” to implement what is being asked for. Simply implementing features as described by customer representative and business analysts (they neither seem to represent any customer's interests nor seem to have analyzed any of their requirements). The most common mistake we make is take for granted what analysts say users want rather than actually watching what they do. Initial requirements are almost always wrong. We need to iteratively refine them and show them to users to reach satisfactory designs.

Here's a list of some of the most egregious and frequent design mistakes in application UIs:

A. Non Standard GUI Controls

Basic GUI widgets such as radio buttons, scrollbars and hyperlinks form the dictionary for modern day applications. Users are so used to the general functionality of such lexical units that if we change the behavior of such units we might risk confusing the user — like inserting foreign words into

one's natural language. For instance: having a list of things with a transparent scrollbar can often confuse the user into thinking that what appears is the entirety of the list and he/she perhaps won't bother to scroll down to see other options[18].

Over the decades, interaction designers have refined and defined the standard look and feel of some of our basic GUI units. It is unlikely that over a single weekend someone would come up with something new and assuredly better. Even if one is to assume that a newer button is sleeker and better looking, humans by nature are resistant to change. It is highly likely that a user using GUIs with certain inherent assumptions will get confused and/or feel annoyed at having to learn a new behavior for a standard control. Jakob's Law of the Internet user experience says that "users spend most of their time on other websites". This means that users will always prefer to use existing websites as opposed to your new website because they assume they would have to expend energy to learn new things rather than using their time and energy in actually utilizing features to help them achieve their goal. Consequently, Jakob's Second Law is even more crucial: "Users have several thousand times more experience with standard GUI controls than with any individual new design".

Looking Like a GUI Control Without Being One:

One common problem is encountering websites that have something that looks like a GUI control but isn't one. This affects the usability of the website. E.g. We often see text that are colored or underlined and assume that they might be hyper-links, only to click and find out that they do nothing. It can lead to users assuming the site to be broken[18].

Another example would be the case when a something looks like a button but renders no action on clicking. Or a radio button that isn't actually a choice. Referencing an example from a case study conducted by the Nielsen Norman Group on a clothing store's website:

The page as in Fig. 1 shows options for new and existing customers to place orders. A fraction of the user testing the website kept frustratingly clicking on "New Customer" and "Existing Customer" assuming they were buttons. However, the screen element was not a button but an unclickable heading. It was only a while later that users overcame this confusion and proceeded with their orders. While one may argue that this would be a small fraction of inexperienced users who might make such mistakes, but the counter here is: isn't it our job to make interfaces intuitive even to the most inexperienced user? Any business owner would know that disgruntled customers tend to add up very quickly.

Another mistake within the same example is the use of radio buttons. The options shown are not mutually inclusive. In most of the cases observed the user had two simple workflow models in mind: there were two separate issues. One, to select a new or existing user. Two, to then select the appropriate action to give measurements. A standard way to do this would have been to provide a set of radio buttons to first select new or existing customer and then another set of buttons should have been revealed depending on the first

choice to proceed with measurements. A good design choice would never display a single set of radio buttons for options which do not come under a single issue.

B. Inconsistency

Non-standard GUI controls are a special case of the more general problem of inconsistency in design. Users tend to be confused when different words are used to describe the same function within an application. Or when the same word is used to describe disparate concepts within an application. Worse is when things move around within an application, violating display consistency for the user. For instance: in Fig 2. the button to "Cancel" switches sides in the iOS7 EMail application within the same application on different "pages" i.e. "New Message" and "Contact". Why?

Jakob Nielsen terms this as the double-D rule: differences are difficult[18].

Another example from Nielsen case study: Expedia pops up a two month view of the dates when a user tries to select departure and arrival dates for their trip. The combined snapshot shows what happens when a user attempts to make a booking with the departure date as March 10 and return date as March 15.

After a user selects a date in March, in the second pop up the March column moves to the left making way for April to appear. Now while it makes sense to take February off the display because obviously someone traveling in March will not be returning in February but, there is an intuitiveness issue here. As an user my intuition is to click on March 15 at the same spot as the previous pop up. So at any point (and I certainly have too many such experiences to back up this study) a user could end up selecting the correct day but the wrong month for return.

Booking a wrong date for flights could have disastrous consequences for the traveler. The passenger might arrive at the airport with a wrong ticket for his expected flight. Or, assuming the travel website has good email notification system in place and the customer discovers this issue beforehand, it will still lead to annoying customer calls and expensive rescheduling/cancellation fees.

Again one can argue that the user should be careful enough to double check his bookings but as developers our concern isn't what a user should do, rather what the user usually does. The time apparently saved in not having to click the next button for April is wasted in correcting a wrong selection of dates due to intuitive clicking.

C. No Perceived Affordance


By "affordance" we're referring what an object can afford to do. For instance, a checkbox affords selection and de-selection. A scrollbar afford moving up and down. Therefore "Perceived Affordance" refers to a user's ability to figure out a lexical unit's functionality before actually using it or feeling it in case of a tangible device.

Perceived affordance is particularly significant to UI design. Each screen pixel allow clicking on it but obviously there will only be a few elements on the screen which are

NEW CUSTOMER


Give us your measurements

Take or ask someone to help take your measurements, by following our easy instructions . It takes just 5 minutes!



Send us your best fitting shirt* (go directly to cart)


If you prefer not to take measurements, you can mail us your best fitting shirt. Our Master Tailor will take the necessary measurements and will return your shirt along with your order.



* : Your shirt will be used for measurements only. We will not copy it.

Visit our NYC showroom (go directly to cart)

Contact us at contact@listerouge-paris.com to plan a private appointment at our New York showroom (Madison Ave & 40th St.).



EXISTING CUSTOMER

Your measurements are on file (go directly to cart)

If your last order fits perfectly, we will make the new shirts with exactly the same measurements.

If your measurements have changed

Simply note your measurements changes compared to your previous shirts.

Fig. 1. Selection page for a clothing store

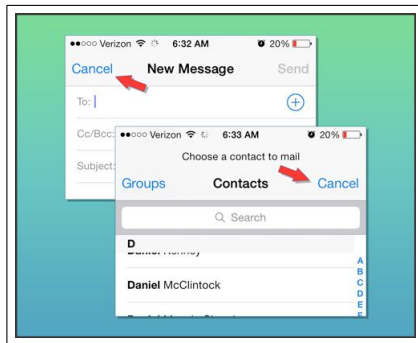


Fig. 2. "Cancel" on different sides on different occasions in the same application[17]

actually actionable. The user does not have the time to play a mine sweeping game to figure that out. Therefore, something like drag-and-drop options are unintuitive unless something makes their drag-and-drop nature obvious such as a piece of attached text or floating icons and a basket beside it etc. Using checkboxes and radio buttons on the other hand saves time and is painfully obvious to users new and old.

Usual signs of the lack of perceived affordances are:

Common question — "What do I do here?"

Users tend to be annoyed with verbose multi-stage instructions to operate an application especially if those instructions disappear (as they often and generally do) after the first step is taken. Therefore, any application needs to equip the user to perceive what it can afford easily. For instance, if I open a document editing application and it just creates a blank white full screen for me I would think for a moment that the program has crashed. Or an immediate question would be — "What do I do here?". Instead, if it opens a blank white document but with a few menu options along one side and a blinking pointer somewhere in the document I can "perceive" that I have to begin writing in that space. This is similar to

the problem that users of the early Macintosh machines must have faced.

"Tiny click targets" are a similar perceived affordance issue especially with aged users and users with motor disabilities. Some clickable elements on websites can be so small that users can miss them or have no actionable response even after clicking because they think they clicked on the correct spot but they actually did not — leading them to believe that the functionality is not working[18].

D. No Feedback

One of the primary guidelines for improving usability of an UI dialog or element is to provide feedback:

- Show users the current state of the system
- Tell users about how their command will be interpreted
- Tell users something at least about acceptable and unacceptable inputs
- Tell user what happens when they issues a certain command[10]

However, a lot of applications keep users playing the guess-game.

A very common example would be — when you enter a wrong username or password on a login page, websites seldom tell you exactly which of the two was incorrectly entered. You then try resetting password only to figure out it was actually your username that was wrong. This frustrated me personally on a daily basis. Simple for the sake of programming speed, this tiny little feature is overlooked by most developers.

Another common example will be when applications do not furnish any information if an action is taking long to complete. Users can be led to thinking that the application is broken and might start clicking around the screen. If you can't meet the recommended response time limits, say so, and keep users informed about what's going on.

The screenshot displays the Expedia Travel App interface for selecting travel dates. It features two side-by-side calendar views for February and March 2008. Above the calendars are input fields for 'Departing' and 'Returning' dates and times, with dropdown menus for 'Any' time. Below the calendars are dropdown menus for 'Adults (19-64)' (set to 1) and 'Seniors (65+)' (set to 0). The March calendar shows the date 10/03/2008 selected. Both calendar grids have a 'Close' button at the bottom.

Fig. 3. Two Month View on Travel App Expedia

E. Bad Error Messages

Confusing and unclear error messages can be particularly frustrating for the average user in their experience of an application. Following standard guidelines for error messages can help to prevent this frustration, but the primary objective in writing error messages is to communicate the error using detailed but accessible language. Merely stating that there has been an error without providing the user any context or solutions is not ideal. The user should be provided with enough information to address the problem independently or at the very least be given sufficient search criteria to research and address the problem via an external source.

Providing users with intelligently phrased error messages can be empowering. If users can easily comprehend and obtain information from an error message, they are more likely to take the time needed to manage the error.

F. Asking for the Same Info Twice

Requiring users to enter the same data in multiple fields is unnecessary and redundant. Developers should always take extra care to ensure that forms will be auto-populated as needed. The only reason users have to repeat themselves is because developers feel too lazy to transfer the data from one part of the app to another.

G. No Default Values

Applications should include default values. Default values can streamline the experience of using an application by eliminating the need for users to input values where the default values are suitable. They are also useful in the sense that they can provide a baseline example for the kind of response acceptable for the question asked, making the application that much more user-friendly. For users who are unfamiliar with an application or unsure of the appropriate values for each field, the existence of default values can guide users toward a safe outcome, by letting them accept the default if they don't know what else to do. If users do not have any particular preference, default values can help speed up process across selection modules on apps.

H. Dumping Users into the App

Most Web-based applications encountered are a result of users surfing on the Internet. Even if users deliberately seek out a new app they go into it with a preconceived model of how it works. They do not know (and neither do they care much) for the workflow, features they will be manipulating and the expected outcome. For traditional applications, this is less of a problem. Even if someone has never used Microsoft Word, they've probably seen a word editing software. Thus, a new Word user will have at least a bare-bones understanding of the application before using it for the first time.

Usability suffers when users are just dumped into an app without any kind of initial training or setup information to give them an idea of what is about to happen[12]. Sadly, most users are put off by verbose instructions so screenshots and video tutorials are the new way to go to help users grasp the main point. For example, an application can have a snapshot of some of the common actions to go through on app-startup. This way the user doesn't have to explicitly go elsewhere to view a tutorial and can seamlessly be introduced into the app.

I. Not Indicating How Information Will Be Used

Users are often required to enter information without any knowledge of what the information is being used for. In this scenario, the developer should rather clarify the purpose of said information, making the process more efficient. Users are less likely to regret irreversible entry if they are fully aware of its purpose. One example of a mistake that can occur from lack of information is when a user enters something silly or inappropriate into a "nickname" field on a website not knowing that the name will be used to refer to him/her for all eternity and probably is irreversible according to app rules[11].

Another example would be: an e-commerce site prompting users with a demand for their ZIP code before they can view product pages. This can be a big turn-off since people are wary of privacy violations and hate snooping sites. An alternative design could work much better: explaining that the site needs to know the user's location so it can show shipping charges for the products being viewed.

III. REAL LIFE INSTANCES OF USER INTERFACE DESIGN DISASTERS

Until the day man and machine start communicating telepathically, we will be needing some kind of menu, control panel or intermediate medium to interact with one another. And since neither understands each other's language or motives, it is crucial to make the connecting medium as simple and intuitive to operate as possible. Yet more often than not we find user interfaces in the real world which are as unintuitive as they are complicated to work on. While some of the consequences might be as harmless as ordering a Big Mac meal instead of a single burger, others can be quite disastrous...

A. *The USS Vincennes Shot Down a Civilian Plane Because of Bad Cursors*

Towards the end of the Iran-Iraq war of 1988, the USA and Iran had a bit of their own war on the sidelines. A most unfortunate incident took place when in the middle of a confrontation between the USS Vincennes and the Iranian gunboats, Vincennes mistook a civilian aircraft for a combat aircraft in attack mode and shot it down. If the question is — why didn't the ship have a radar system to identify friendly objects from enemy vessels — well they did have such a system, except, it had the most unintuitive user interface[17][13].

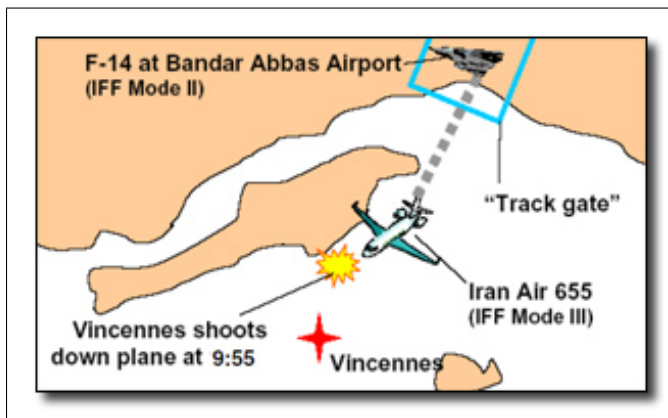


Fig. 4. The Vincennes shoots down Iran Air 655[17]

The Dumb Problem:

The interface was so designed that if you clicked on an object the interface started tracking it. If you wanted extra information about the object such as altitude, speed, direction etc. then you had to move to a separate cursor and click on the object again. Such a clumsy, complicated design wouldn't even make it out of the brainstorming session for a modern app.

As it would happen the operator thought that he was viewing information about the aircraft he had clicked on (which was the Iranian passenger Airbus) whereas his other cursor was displaying information about a F14 Fighter Jet 14 miles away because this other cursor was on the combat aircraft. Of course, you could say that there were other

factors governing the decision such as altitude and direction of flight. Sadly the system made it easy to make an error in judging those too. The system displayed altitude on a smaller separate monitor and an operator had to take separate readings of an object's altitude, note it down and calculate the difference to determine whether an object was flying up/away or down/toward. In this case, the operator made a miscalculation leading to them thinking that the Airbus was flying down towards them when it most probably trying to get away from that zone.

Result: All 290 passengers onboard died.

B. *Air Inter Flight 148 Crashed Because a Display Screen Was Too Small*

Know of Microsoft's Paint? If you want to change the size of the picture in addition to entering a value you need to select an appropriate mode for it as in pixels or inches or percentage. Else 300 percentage instead of 300 pixels will likely make your monitor suddenly explode. Remember our digital alarm clocks before the mobile explosion of the 21st century? It had modes to select AM and PM so you don't set your alarm to 6 in the evening when you have to wake up 6 in the morning. Well our dear designers of Air Inter Flight 148 made a similar mistake. Except that in this case, people died[17].

There were many factors that were claimed to have been involved in the crash but after extensive investigation the bottom line was: the plane crashed into a mountain because the display screen was too small. Huh?

The Dumb Problem:

Apparently, the tiny display screen in the pilot's control panel provided space to configure parameters for the flight's landing. The pilots set it to "-3.3" for the plane to descend at an angle of -3.3 degrees. If all went well, the plane would have landed at 800 feet per minute and everyone would have reached home in time to catch Seinfeld (or whatever the French counterpart was). But sadly, what happened was that the pilots entered the value "3.3" into a mode that basically determined the speed of the plane in feet per minute. There was no mode indicator on that small screen so basically it would have shown the same result even as if the mode was correct.

Result: 87 out of 96 passengers died.

C. *The Kegworth Air Disaster Happened Because of a Digital Dial*

In 1989, a Boeing 737 crashed into the side of the M1 motorway near Kegworth, England. The issue that arose was that the aircraft began vibrating tremendously and the pilots thought it was the good engine that had malfunctioned and turned it off. With one engine gone bad and the good one turned off, what happened? — same thing that could happen to a metal vessel up in the air weighing hundreds of tons. It crashed[17].

The Dumb Problem:

A couple of months prior to the crash the cockpit panel in the aircraft was updated to give it what was called a

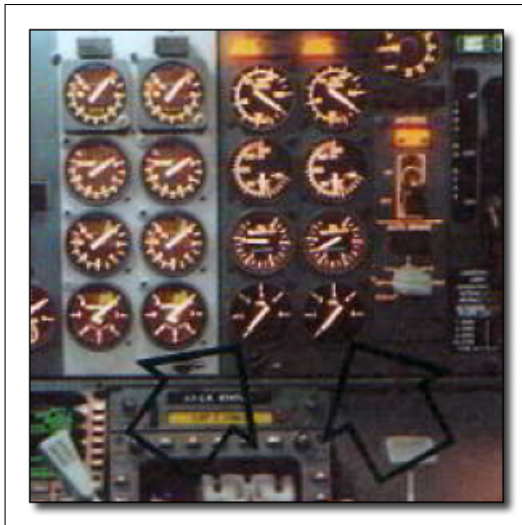


Fig. 5. Older larger dials[17]



Fig. 6. New dials from same distance as older dials[17]

“slicker looking design”. At one point during the flight the dial indicating vibration in the faulty engine rose to the maximum and stayed there for a full three minutes but the pilots missed it because of a badly designed, tiny display. The older model had amply visible mechanical pointers which a pilot could see even a few steps away from the cockpit and which helps in such emergency situations where one does not have the time to squint and see what’s what on every dial. Here’s a picture:

With large clear interfaces it’s easy to spot indicators and any concerning changes or information floating up. Attempt to find the needle on the new dials: Fig. 7 and Fig. 8

Do you think two people in the middle of trying to land a collapsing aircraft have the time to microscope their way to clarity? The three tiny lines you see outside the dial in the scaled up image? That’s the entire pointer. The dial was already difficult to read and now add to that a shaking plane and no one can blame the pilots. Even if they tried reading

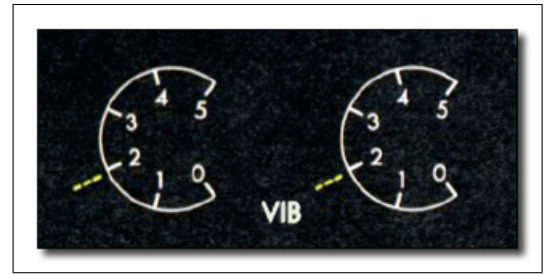


Fig. 7. New dials zoomed in[17]

the dials, the need for style overshadowing readability must have made it extremely difficult if not impossible to read.

D. Technology and its apparently User Friendly Interfaces

When Apple released the Apple Watch in 2016, there were as many people frustrated with as excited about the product. The primary reason for dislike was that the watch was unnecessarily complicated to use. There were eight different ways to interact with the watch: tap screen, hard-press screen, clicking the knob on its side, swipe across the screen, and pinch the screen.

The following images shall speak for themselves as far as UI design mistakes in some of our modern day applications go: Fig. 9., Fig. 10 and Fig. 11.

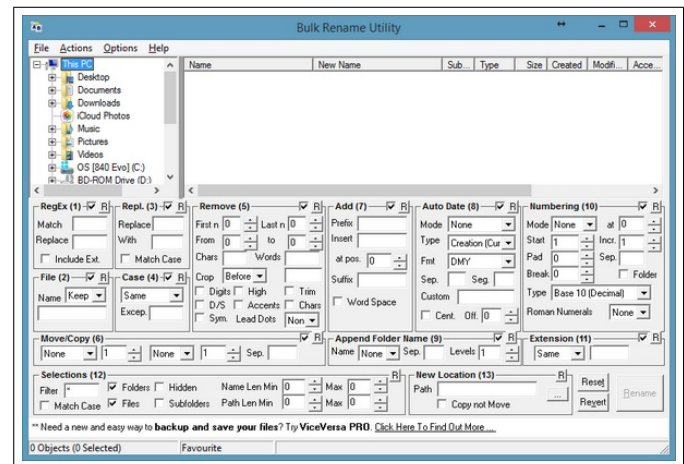


Fig. 8. Someone said it right — Engineers don’t let engineers design user interfaces.

IV. RESEARCH IN INTEGRATING REQUIREMENTS ENGINEERING INTO USER INTERFACE ENGINEERING

In this section I attempt to present a review of past and ongoing research in the field of designing user interfaces by evaluating requirements engineering practices. While doing so, I discovered a significant amount of overlap between the fields of Human Computer Interaction (HCI) and Requirements in trying to address UI design concerns.

A. From early requirements to user interface prototyping

Alicia Martinez et al.[1][2], have tried to define a software production process which embodies the relationship between

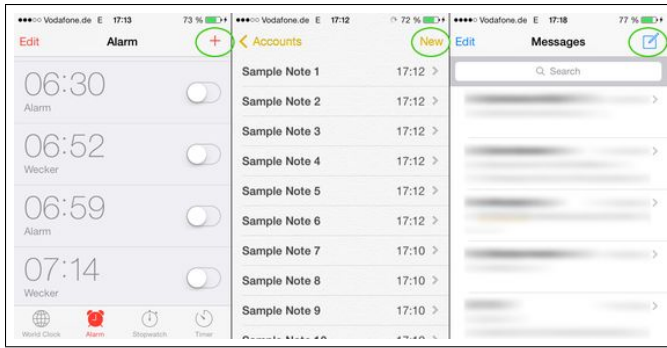


Fig. 9. 3 different visual paradigms for the -Create New- feature in different apps on the iOS interface. Why?

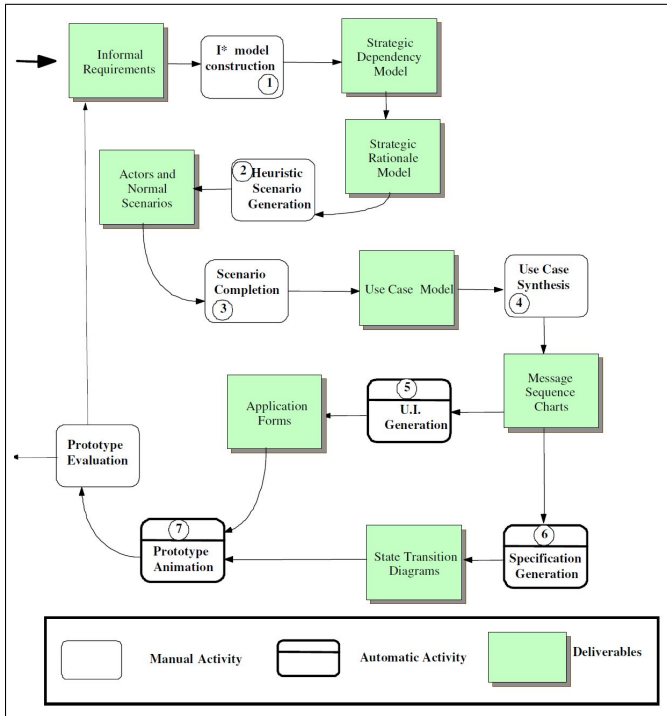


Fig. 10. Schematic representation of Martinez et al.'s method[1]

the primitive components of a business model (as represented by the I* framework) and the interface of the system. The resultant user interface complies with the Unified Model Language. They utilize Use Case Models to represent the transition between business requirements and the application software. In this way, they attempt to achieve proper embedding of early requirements engineering into the traditional software engineering process so that the users can validate their requirements as early as possible. User verification is done through validation of the user interfaces thus generated as a result of the software production process. These user interfaces are not perfect but can serve as a starting point and aid in further refinement in the software development process.

Fig. 12 represents an overall general view of how user interface prototypes are derived from organizational models. This process can be divided into three phases: In the first

phase, a business process model is created. In the second phase, the business model is translated into use cases. In the third phase, the prototype of the user interface is constructed from the use case model. Figure 12 shows the seven steps used in the method:

- The first step of the process is the collection of the informal requirements of the organizational environment. This data serves as input for the business process modeling phase using the I* framework. As a result, we obtain a strategic dependency model (SDM) and a strategic rationale model (SRM) both of which together depict the business process of the organization.
- The results from the first phase serves as input to the second phase which is basically the use case generation phase. The generation process houses a set of heuristics which help detect an initial use case model, the actors of the use case model as well as the general interaction scenarios with the system.
- The third phase of process begins with scenarios completion. This phase develops the general interaction scenarios by adding alternate and exception scenarios. The use case model is similar to that of a Message Sequence Chart (MSC). These MSC diagrams are rich with information regarding how the user requirements scales into the user interface and aids in generating visual components of the interface. This information is analyzed to generate a target Application Form. Finally, the Forms and the transition diagrams are used as input to symbolically carry out the user interface.

The authors contend that they have devised a methodological approach to incorporate a layer of initial requirements, through the I* framework within the software production process. This initial layer is used to describe the organizational context in which the system is immersed. A use case model of the system is then generated using the set of heuristics. The model then serves as input to a semiautomatic process which generates the user interface specification. This specification is used in the last phase of the method to animate the interfaces generated. The process is supported by tool which is programmed in Delphi and with a relational repository in Interbase. The tool apparently generates prototypes in the languages Delphi, Visual C++ and HTML.

B. Scenario-based requirements engineering facilitating interaction design

When the requirements and design concerns of a system's interface are separated, the resulting system will be less than optimal as most likely the application's usability will be hampered. It may happen that even if all the features in the requirements have been implemented, a bad user interface induces errors during the human-computer interaction. It may also happen that the fabulous user interface of a system with features that are not required will not be very useful either. Therefore the primary motivation is to improve software engineering practices by integrating requirements analysis and user interface design, especially facilitating the latter. In this paper, Dr. Hermann Kaindl argues for combining

the two based on usage scenarios which can be assumed to be a sequences of actions aimed at accomplishing a task or goal[16].

A general overview of the method includes:

- Functions/tasks, goals, scenarios/use cases:
It encompasses particular cases of how the system is to be used via a Use-Case Report such as:
Brief description, flow of events, special requirements, pre-conditions, post-conditions, extension points, relationships, use-case diagrams, other diagrams.
It also elaborates on the interaction design through depiction of:

Design of interactions between human and computer, relation to requirements engineering and relation to task analysis.

There is no commitment to specific user interface during this step.

This phase may also outline functional requirements i.e. high-level statements of what the system should be able to do, describe the functions of the system to be built in detail (but not yet its design or implementation) etc.

- Requirements and object-oriented models:
In this phase we layout two types of requirements such as — constraints on system and constraints on process.
Constraints on system:
Performance, Reliability, Security, Safety, Portability, Maintainability, Reusability, Interface, Usability
Constraints on process:
Specific development process to follow, specific programming language for implementation, specific tools to be used, specific hardware to be used, political issues, time to market, terms of delivery, cost.

- A systematic design process
This step is for figuring out goals and linking them to each other to link various end features. E.g. If some goal is known from the old system, then figure out whether this is still a goal in the new system that will include the system to be built. Or, if some goal is known for the new system, then try to link it to one or more scenarios for the new system that are already known. So on and so forth.

- Scenarios/use cases for interaction design
Interaction scenarios with attached task descriptions e.g. A mockup of a Menu of Options for a User.

Kaindl emphasizes that goals, scenarios and functions/tasks should be combined as this combination serves as a basis for a systematic approach.

C. Usability Engineering Methods for Software Developers

Andreas Holzinger[4] argues on behalf of the HCI community that awareness of basic usability methods is a must to enhance ease of use and consequently users' performance and their satisfaction. Thus it is important that a software developer not only be aware of the various usability methods but also be able to decide which method is ideal for which software project.

Menu of Options

Screen 1:

Select one option:

☐ Register to vote

☐ Mark ballot

☐ ???????

☐ ???????

Fig. 11. A Menu of Options Mockup[19]

In this regard he presents a few Usability Inspection as well as Usability Testing approaches:

	Inspection Methods			Test Methods		
	Heuristic Evaluation	Cognitive Walkthrough	Action Analysis	Thinking Aloud	Field Observation	Questionnaires
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no
Comparison of Usability Evaluation Techniques						

Fig. 12. Comparison of Usability Evaluation Techniques[4]

• Usability Inspection Methods

This is a set of methods for identifying usability issues and improving the usability of an interface design by pitting it against established standards.

1. **Heuristic evaluation (HE)** is the most common informal method[5]. It involves having usability specialists evaluate whether each interactive element follows established usability principles (e.g. Nielsen's Usability Heuristics). First each specialist evaluates the interface individually and then share their views via common dialogue. This ensures unbiased evaluations followed by informed common decisions.

The heuristics must be carefully opted for so they suit the specific system being inspected, especially for Web-based services where additional heuristics often become increasingly important. Usually 3-4 specialists take part in this process.

Advantages include the application of recognized and accepted standards, intuitiveness, usability evaluation early on in the development process and effective identification of major and minor issues.

Disadvantages include separation from end users, inability to identify or allow for unknown users' needs

and unreliable domain-specific problem identification. Further, HE does not necessarily provide a holistic and complete design evaluation, since there is no mechanism to ensure the entire design was explored, and specialists may focus too much on one section and too little on another. (Also, the validity of Nielsens guidelines has been questioned.)

2. Cognitive Walkthrough (CW)

Cognitive Walkthrough is a task oriented method where the specialist evaluates the system's functionalities through simulation of step-by-step user behavior for a certain task. CW evaluates cognitive aspects, such as learnability, by analyzing the mental processes required of the users. This is achieved by making the repository of available actions more noticeable — providing an obvious way to undo actions, offering limited alternatives etc. The background needed for this is derived from exploratory learning principles.

Advantages include a more fully functioning prototype compared the one produced by HE, helping designers work from a user's perspective, effectively identifying issues arising from human-computer interaction and the ability to better understand the users' goals and assumptions.

Disadvantages include possible tediousness and the risk of an inherent bias due to improper task selection, emphasis on minor details, and non-involvement of the end user once again as in HE.

3. The action analysis method is divided into formal and back-of-the-envelope action analysis — in both, the emphasis is more on what the practitioners do than on what they say they do.

The formal method involves close inspection of the action sequences a user executes in order to complete a task. This is also called keystroke level analysis. It involves breaking the task down into smaller tasks and calculating the times needed to perform the tasks. Back-of-the-envelope analysis is less detailed and gives results that are less precise but it can be performed faster than keystroke analysis.

The main difficulty with task analysis is in accommodating complex tasks completed by more than one individual. Furthermore, the representation of a task analysis is complex, even when a simple task is being studied, and tends to become very unwieldy very quickly.

The advantage is that we get precise accounts of time taken to complete certain tasks and a deeper insight into user behavior.

• Usability Test Methods

1. Thinking Aloud (THA)

This is probably one of the most effective methods for testing the UI design. Users are asked to evaluate the system by verbalizing as they work their way through an application. This enables the developers to understand how the user views the system and what are the major confusions and misconceptions. A variant of THA called constructive interaction involves having two

SE Development Stage	Generic Technique Name	Name Used by ASPD Authors	Primary Studies	Application Type
Requirement Engineering	Contextual Inquiry	Contextual analysis	[1]	As Is
		Context of use	[17][23]	As Is
		Field studies	[26]	As Is
		Context inquiry	[8]	As Is
		Contextual Inquiry	[42]	As Is
		Ethnographical Observation	[8]	As Is
	User Analysis	Ethnographic research	[26]	As Is
		Observational studies of users	[30][46]	With modifications
		Card Sorting	[8][23][38][42][26][40]	As Is
		Personas	[10][21][25]	With modifications
		Lightweight personas	[31]	With modifications
		Extreme personas	[48]	With modifications
	Task Analysis	Essential use cases	[17]	As Is
		Task Scenarios	[23][25][40]	As Is
		Scenario based descriptions	[17]	As Is
		Scenarios	[30][46]	With modifications
		Scenario based approaches	[31]	As Is
		Prototypes	[11]	With modifications
	Prototyping	Paper prototypes	[23]	As Is
		Hand-drawn wireframe on paper	[27]	With modifications
		Mock-ups and prototypes	[17]	As Is
		Informal cognitive walkthrough	[20]	With modifications

Fig. 13. HCI Techniques Adopted By The ASPD In Requirements Engineering Activities[3]

test users use a system together (co-discovery learning).

While time consumption is a concern in THA, if done right it allows us to closely approximate how individuals actually use the system, rich store of information regarding preferences, performance, intuitiveness etc., received from storing user comments.

A disadvantage to the method is its failure to lend itself to most performance measurements because one user might be a slower learner, another user might look at comfort over features, another user might look at it analytically etc. Also forcing users to concentrate especially for the purpose of testing can create less than natural environments.

2. Field Observation (FO)

This is the simplest of all usability testing methods: visit a user at his workplace to observe him. Notes are taken as inconspicuously as possible without making the user conscious or interfering in between. Sometimes video is used to make the observation process less obtrusive. Some other means also include electronic data logging, querying the users via detailed questionnaires or a combination of any of the above methods.

Advantage is that possible user frustrations, confusions, anxieties etc. can be easily understood. Disadvantages include that results have low validity — discrepancies between subjective and objective user reactions must be taken into account. Also, this method needs sufficient responses to be significant.

Usability inspection and testing need to be combined and incorporated into software development because they aid in understanding the users' misconceptions, culture and capabilities.

D. Intersection of Human-Computer Interaction (HCI) and Requirements Engineering (RE) to aid UI Design

Daniel A. Mages et al.[3] contend that interest in the integration of the agile software development process (ASDP) and user-centred design (UCD) has grown over the last decade. However, not a lot of research addresses this issue holistically or attempts uncover the current state of this integration. Their objective was to conduct a study to answer the

HCI TECHNIQUES ADOPTED BY THE ASDP IN DESIGN AND EVALUATION ACTIVITIES				
SE Development Stage	Generic Technique Name	Name Used by ASDP Authors	Primary Studies	Application Type
Design	Interaction Design	Screen Snapshots	Interaction design and information architecture [22]	As Is
		Upfront design	[34][47]	With modifications
		User workflows	[17]	As Is
		Conceptual model	[23]	As Is
		Conceptual designs	[26]	As Is
Evaluation	Evaluation by Experts	Navigation Maps	Interface and navigation design [22]	As Is
		Prototyping	Lo-fi prototyping [25][26][28][40][47]	With modifications
		Heuristic Evaluation	Heuristic evaluation [30][46]	With modifications
		Cognitive Walkthrough	Informal cognitive walkthrough [20]	With modifications
		Evaluation by Experts	Usability expert evaluations [25][26][40][48]	As Is
	Usability Testing	Thinking Aloud	Simplified thinking aloud [30]	With modifications
		Laboratory Usability Testing	Thinking aloud [37][46]	With modifications
		Usability Testing	Laboratory usability testing [26]	As Is
			Usability testing [2][23][40]	As Is
			Usability test [17][25]	As Is
			Rapid iterative test and evaluation (RITE) [15][26]	With modifications
			Usability sessions [27]	With modifications
			Rapid usability testing [32]	With modifications
			Micro testing [39]	With modifications
			Extended unit tests for automated usability evaluation [25][29][48]	With modifications
Follow-Up Studies of Installed Systems	Questionnaires	Usability tests	[25][33][48]	With modifications
		Informal usability testing	[15]	With modifications
		User testing	[16]	With modifications
		Usability evaluation	[35]	With modifications
		System usability scale (SUS) questionnaire	[1]	As Is
		Product reaction cards (PRC)	[1]	As Is
		Focus Groups	User workshops [39]	With modifications
	User Feedback	Feedback days	[39]	With modifications
		Usability research	[27]	With modifications
		User forums	[11]	With modifications

Fig. 14. HCI Techniques Adopted By The ASDP In Design And Evaluation Activities[3]

following research question: “What is the current state of integration between agile processes and usability techniques?”. The method they adopted was to conduct a systematic mapping study (SMS). An SMS is a form of systematic literature review with the aim of identifying and classifying research papers addressing a specific issue. Results: they retrieved a total of 31 primary studies suggesting that the ASDP community is beginning to adopt usability techniques into its software development processes. What’s interesting about the paper is that they notice most of the human-computer interaction (HCI) techniques that the ASDP is adopting are related to requirements engineering — especially techniques for requirements elicitation and analysis. They conclude that there is a rising trend of consideration of usability in the ASDP. However, the literature surveyed does not contain any general, formal or systematic proposal to help agile development teams adopt usability techniques in their development projects. They organize the HCI techniques being adopted by the ASDP in requirements engineering and UI design, into two tables showcased in Fig. 15 and Fig. 16.

Judy Brown[7], further speaks from the community of HCI, of 3 major design processes within the HCI community for building superior user interfaces: user centered design, cognitive modeling and participatory design.

1. User centered methodology focuses on rapid iterative prototyping until design meets functional requirements to the optimal possible level. Often for this purpose, Nielsen’s steps to a practical and economical process for user interface design are adopted which include: knowing the user, competitive analysis, setting usability goals, collect feedback from field use, parallel design, empirical testing etc.[6]

2. The emphasis of cognitive modeling is to understand and model an activity as it is understood by the user. Cognitive modeling asserts that this model can create a design that will be intuitive to the user. Cognitive modeling

advocates are interested in why users behave as they do or why one design is better than another. This approach is not methodology focused although there are some methods such as GOMS (Goals, Operators, Methods and Selection Rules, as described in Section IV.B.) which are utilized from time to time.

3. For participatory design advocates, the important issue is communication between users and designers. This is especially important because the user is a participant in the design process. PD advocates focus on developing methods to enhance communication between users and designers. sayeveral PD researchers noted that it was not the particular methods and techniques that were decisive [in system development], but a strong political focus on participation, communication and learning.

V. CONCLUSIONS

Research in both the HCI and Software Engineering communities is beginning to focus on evaluating and identifying usability inspection and testing methods to better aid the developer understand the needs, challenges and culture of the target user base. No longer is it feasible or wise to think of requirements engineering and user interface design as disparate phases of the software development process nor is it advisable to ignore either phase with the assumption that the “engineer knows best”. They don’t. While the volume of research in current years in this direction is quite significant, a lot of such good practices are yet to be adapted in the industry. Hopefully, sooner than later, usability inspection and testing will become a concrete aspect of the software development process.

REFERENCES

- [1] Martnez, Alicia, et al. sayFrom early requirements to user interface prototyping: A methodological approach. Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on. IEEE, 2002.
- [2] Snchez J; Pastor O; Fons J.J. “From User Requirements to User Interfaces: a methodological approach”. CAISE 2001. 4-8 Junio 2001. Interlaken Suiza.
- [3] Mage, Daniel A., John W. Castro, and Silvia T. Acuna. “HCI usability techniques in agile development.” Automatica (ICA-ACCA), IEEE International Conference on. IEEE, 2016.
- [4] Holzinger, Andreas. “Usability engineering methods for software developers.” Communications of the ACM 48.1 (2005): 71-74.
- [5] Curtis, Bill, and Bill Hefley. “A WIMP no more: the maturing of user interface engineering.” interactions 1.1 (1994): 22-34.
- [6] Paech, Barbara. “Formal user-centered requirements engineering.” SIGCHI BULLETIN 29 (1997): 54-55.
- [7] Kaindl, Hermann, et al. How to combine requirements engineering and interaction design?. IEEE, 2008.
- [8] Brown, Judy. “HCI and Requirements Engineering-Exploring Human-Computer Interaction and Software Engineering Methodologies for the Creation of Interactive Software.” SIGCHI Bulletin 29.1 (1997): 32-35.
- [9] Sutcliffe, Alistair G. “Requirements engineering.” The Encyclopedia of Human-Computer Interaction, 2nd Ed. (2013).
- [10] S. Adikari, C. McDonald, and J. Campbell, “Little design up-front: A design science approach to integrating usability into agile requirements engineering.” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 5610 LNCS, vol. 1, pp. 549-558, 2009.
- [11] Cohn, Mike. Succeeding with agile: software development using Scrum. Pearson Education, 2010.

- [12] Marcus, A. Dare we define user-interface design? *interactions* 9, 5 (2002), 1924.
- [13] Shneiderman, B. *Designing the User Interface*, 3rd ed.. Addison-Wesley, Reading, MA, 1997.
- [14] Constantine, L. L., Biddle, R. and Noble, J. Usage- Centered Design and Software Engineering: Models for Integration. In IFIP Working Group 2.7/13.4, editor, *ICSE 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction*, Portland, Oregon, 2003.
- [15] Constantine L.L; Lockwood L.A.D. "Software for Use: A practical Guide to the Models and Methods of Usage- Centered Design". Addison Wesley 1999.
- [16] Kaindl, Hermann. "Scenariobased requirements engineering facilitating interaction design." *INCOSE International Symposium*. Vol. 21. No. 1. 2011.
- [17] www.cracked.com - "6 Disasters Caused by Poorly Designed User Interfaces"
- [18] www.nngroup.com - Nielsen Norman Group' Tutorial on "Top 10 Application-Design Mistakes"
- [19] Professor Daniel Berry's slides for "User Interface Specification" for CS645, Fall 2016, University of Waterloo, Canada.