THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2025

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

Au Reikura
School of Computing and Mathematical Sciences

THE SOFTWARE DEVELOPMENT LIFECYCLE: EVALUATION

Last week you began building the back-end of your application and as you have already created the front-end (UI) in an earlier session, this means the prototype for Parkway Commute is nearly complete. This week we look at evaluating software in a usability sense to make sure that it meets the needs and expectations of the intended users.

Usability

Jakob Nielsen states, "Usability is a quality attribute that assesses how easy user interfaces are to use. The word 'usability' also refers to methods for improving ease of use during the design process." Usability is made up of five different principles.

- **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design?
- **Efficiency:** Once users have learned the design, how quickly can they perform tasks?
- **Memorability:** When users return to the design after a period of not using it, how easily can they reestablish proficiency?
- **Errors:** How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- **Satisfaction:** How pleasant is it to use the design?

The reason that we cover these five different points is when people hear the word usability, they assume that it only means is the system easy to use. Here we define the five different points of what usability actually is. We need our system to be learnable. It needs to be efficient, memorable, allow users to recover from errors and have a high satisfaction rate for the target audience. Bearing this in mind, what makes a useful system?

What Makes a Useful System?

There are two things that make a useful system. That is utility and usability. We just defined what usability is and what a system needs for it to be usable. Utility, on the other hand, is what you have been focusing on throughout the course of this term. Utility is looking at the requirements and working out whether a system provides the features that you need. For a

system to be useful, it must have usability and utility. One without the other does not satisfy a useful system.

Usefulness, Utility, Usability: 3 Goals of UX Design

Watch this video from Jakob Nielsen about the three goals of UX design:
https://www.youtube.com/watch?v=VwgZtqTQzg8

Why is Engaging End Users Important?

Keep in mind that if you are learning how to program, if you're interacting with technology, if you're the person that a family member comes to to ask for help with technology, you are not the typical user of software. You have more expertise, knowledge and understanding than the average user. Therefore, we need to engage end users to make sure that the software that we design works for those end users. As we're not typical users, something that we design as a system for ourselves, we can probably use no problem, however other people may not be able to use it. When you're designing a product, such as a piece of software, it's crucial that it's usable by whoever your target audience is.

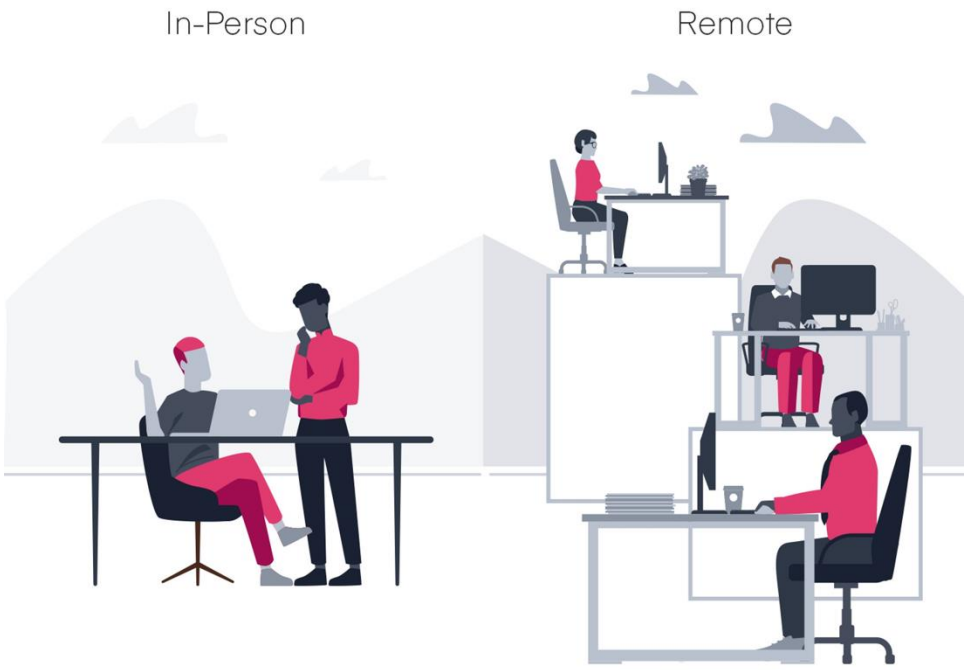

Figure 47: In-Person vs Remote User Testing

One of the big problems we hear frequently regarding user testing is that it's expensive, however it actually isn't, or at least doesn't have to be. This is because participants are asked to do some work with a system that's already partly implemented, and they're giving you their time. Usually, participants require a small reward for their participation but what is expensive is the time cost.

However, if you have a well-designed testing schedule which puts testing throughout the software development process, you're going to identify errors faster, which means you can solve them sooner. As the later an issue is discovered, the more costly it is to rectify, employing user testing actually saves money in the long run. Furthermore, by doing user testing and engaging with those end users, you're creating convincing evidence that you can use, if you're looking for an investor, that your product works and that the target audience can use it (and likes it). This is provided, of course, that you do it correctly. If you do it incorrectly, you're going to get the wrong results, which is the incorrect evidence that you need.

Heuristic Evaluation

Heuristic evaluation is a way of evaluating a user interface if you don't have users on board yet. This technique is used throughout the entire development process but is often preferred early in the design process due to its applicability to both complete and incomplete systems. If the functionality of the system hasn't been finished or even decided, heuristic evaluation can be used on paper prototypes. Then once you start developing that functionality, it can be performed again to make sure the digital prototype you're starting to implement still meets the same criteria as your paper prototype. There are other techniques of course, but today we're going to focus on heuristic evaluation.

Nielsen's Ten Usability Heuristics

Nielsen's set of heuristics allow us to evaluate usability (see figure 48). We do this by going through each heuristic and comparing the software against that heuristic to determine how well it adheres (or violates) that criterion. This allows us to identify potential errors within the system or issues with the usability quickly and cheaply.

Watch this short video on heuristic evaluation:
http://www.youtube.com/watch?v=6Bw0n6Jvwxk

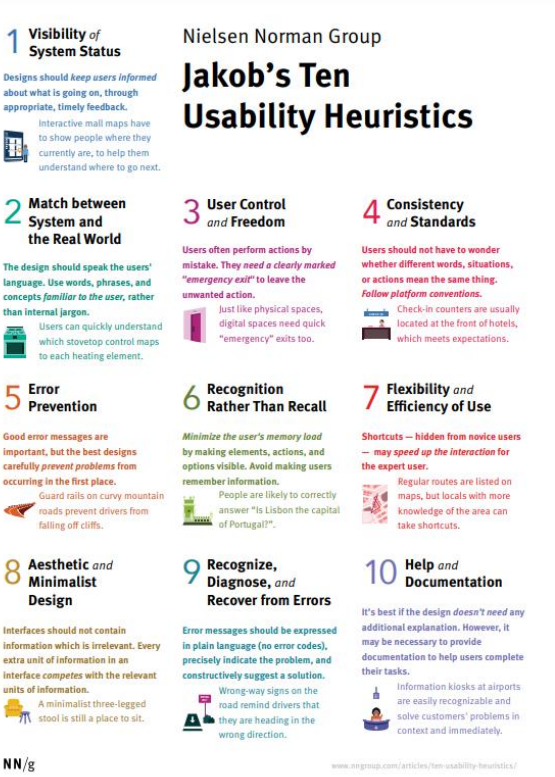

Figure 48: Nielsen's Ten Usability Heuristics for User Interface Design

Visibility of System Status is heuristic number one (see figure 49). What this means is the design of software should ensure that users know what step of the process they're at. For example, if you think about conducting an online purchase, there might be a progress bar across the top of the screen informing you of the current stage. The bar should update as you check out the cart, enter your details and pay for the item, before then confirming the order.

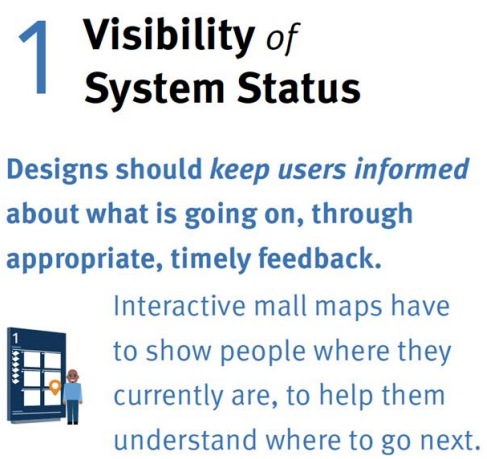

Figure 49: Nielsen's 1st Heuristic

Heuristic two is making sure that there's a match between the system and the real world (see figure 50). The system should include things that your target audience are familiar with. We should avoid using jargon (technical terms) as we don't want to make it more difficult for the user. The example given by the Nielsen Norman Group is the stovetop. When you use a stovetop element, there's usually a small dial either across the top of the stove or beside it, purposely positioned so the user knows what dial control corresponds to what heating element. This speaks the user's language because they can visibly see the mapping of heating elements to controls.

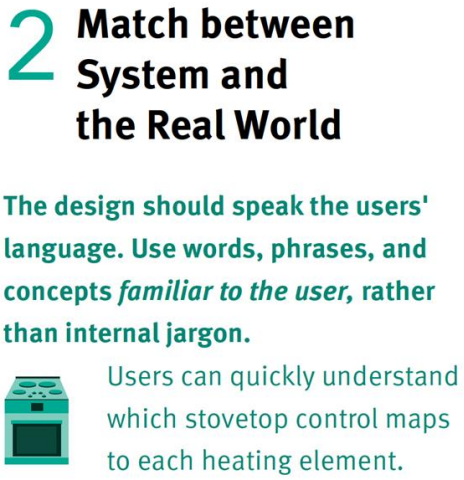

Figure 50: Nielsen's 2nd Heuristic

User control and freedom, the third heuristic, simply means that if a user makes a mistake, there's an emergency exit (i.e. the system includes functionality like undo and redo). This is important because if a user makes a mistake, they are likely to want to be able to undo that so they can go back to the previous state (see figure 51).

Figure 51: Nielsen's 3rd Heuristic

Heuristic four is Consistency and Standards (see figure 52). If you're building an application for a web development project, it's unlikely that you would try and design it like a smartphone application. We create the application and the user interface in a way that is consistent with other user interfaces or other types of software for that particular platform. This means following platform conventions. If you're designing for Android or iOS, they have their own conventions that they specify for development. Quite often if you don't meet those conventions, you cannot upload your application to the Google Play store or to the App Store. So it's really important that you meet those standards for the particular type of software you are creating.

Figure 52: Nielsen's 4th Heuristic

Heuristic five is Error Prevention (see figure 53). Rather than allowing an error to occur in the first place, we should try to prevent problems. This can be done by designing the user interface in such a way that if an issue occurs, there are guardrails in place. For example, if you are using a medical device that infuses medication, a catastrophic problem could be the system administers an incorrect dose of medication. To prevent this, we could put a guardrail in place that would check the amount of medication entered is within safe, pre-defined limits.

Figure 53: Nielsen's 5th Heuristic

The sixth heuristic is Recognition Rather than Recall (see figure 54). Instead of forcing the user to remember particular details of a system, we let them recognize certain elements and actions, and make them visible. For example, desktop software usually has information across the top with organisational elements like file edit menus, print menus, so on and so forth, which is common in almost all software. This allows the user to recognize something familiar and be able to find how to print something regardless of the application that you're using. It's important that we do this as it reduces the load for the user, thus reducing the amount of work they have to do to use our system (which is likely to have an impact on the system's difficulty, and for the better).

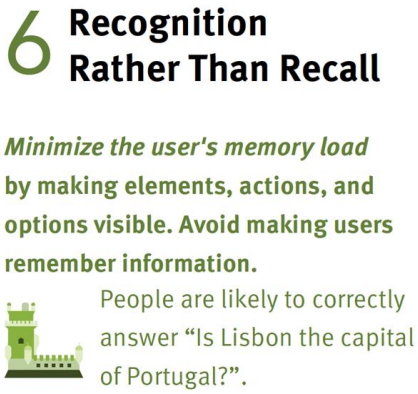

Figure 54: Nielsen's 6th Heuristic

Heuristic seven regards a system's flexibility and efficiency of use (see figure 55). When someone is learning how to use the system, instructions or implied actions need to be clear to guide them. We might want to put in tool tips or hints of how to use the system. However useful initially, once a user has grasped the basics, those tooltips and hints are likely to become annoying and frustrating to them. What we want to do is make sure that we include shortcuts (or "accelerators" as Nielsen calls them) in the system so that interactions can be tailored to users' needs. For expert users, an example of this are the shortcut keys that you could use for different applications. For example, almost every application uses `control + s` for save. By using that shortcut, it means an expert user doesn't have to find the save button element and they can simply save their document quickly and easily.

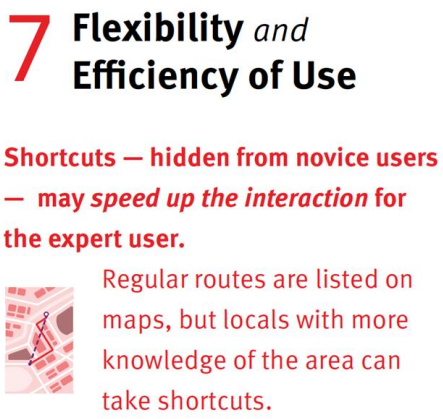

Figure 55: Nielsen's 7th Heuristic

Heuristic eight is Aesthetic and Minimalist Design (see figure 56). When you create a user interface, everything that you put into that interface competes for attention. This means that you need to make sure that only relevant information is directly in front of the user. You don't want any extra information that would confuse the design, and in turn, the user.

Figure 56: Nielsen's 8th Heuristic

Recognize, Diagnose and Recover from Errors is the ninth heuristic (see figure 57). This relates to what should happen if an error occurs. Firstly, you need to provide the user with a way of recovering from that error. Some unhelpful things that you might have noticed yourself when using a computer is an odd error message that pops up on your screen. It may state something like "error 109" and include an okay button that doesn't help you recover from the error. These occurrences are what we want to avoid when designing software. We don't want to give unhelpful error codes, instead we want to give easy to understand, helpful error messages. And if we can, we want the system to recover from the error rather than forcing the user to. If we can't get the system to recover from the error, we need to make it clear for the user how they can do so.

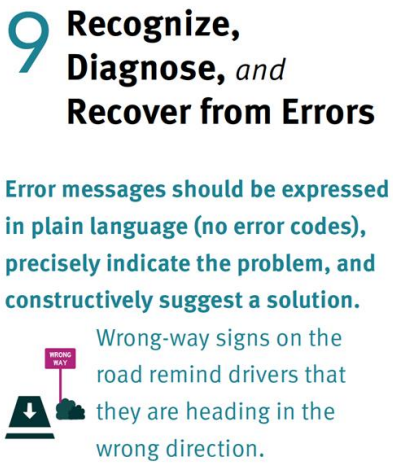

Figure 57: Nielsen's 9th Heuristic

Lastly, heuristic ten is Help and Documentation (see figure 58). In an ideal world, a system shouldn't need any explanation, but it is often difficult to achieve this due to the complexity of modern-day computing systems. Therefore, it is important that the documentation we provide to the users helps them to complete their tasks. An example being instruction manuals. Essentially, we want to give a new user some instructions for how to use the system that once followed, can be ignored because the user has become competent.

Figure 58: Nielsen's 10th Heuristic

Today's Exercise

Over the course of the term you have been building software with a primary focus on functionality and utility. Today's exercise asks you to use the 10 heuristics to perform an evaluation of your software. Consider each heuristic and list how well it does or does not meet

the criteria. Links to posters of the heuristics are included in the 'Useful Resources' section below.

Summary

In this session, we discussed usability, what it is and why it is important, as well as a common technique for evaluation of interfaces, heuristic evaluation. We covered Nielsen's Ten Usability Heuristics for User Interface Design and used them to perform a heuristic evaluation of the software you built for the Parkway Commute project. The next session is a group session where you and your peers will be tested on what you have learnt this term via a small competition called Battle of the Bytes.

Useful Resources

- Usability 101: https://www.nngroup.com/articles/usability-101-introduction-to-usability
- 10 Usability Heuristics for User Interfaces: https://www.nngroup.com/articles/ten-usability-heuristics
- Nielsen Norman Group: https://www.nngroup.com/