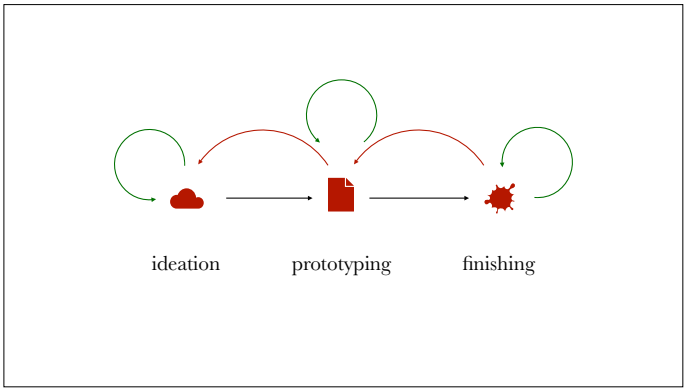
Design thinking is a common name for a set of practices and philosophies that are common in industries dedicated to design: web design, graphic design, product design and so on. It's difficult to build such products in such a way that everybody can be involved, especially non-experts, and so a set of best practices has emerged for structuring such processes.

Why should we care about design thinking? CS is a *creative* enterprise. Not in the sense that we get in touch with our feelings and try to express them in our work, but in the simple, literal sense that we create things. When we start something isn't there and when we finish, something is. Even if the core business is science, which is not creative in the sense that you uncover a truth rather than make it, the results of that science still have to be packaged in some way. We spend a lot of our time writing papers, planning conferences and workshops, design course and so on. All of which require creative work.



Here is the simplest outline of a creative process. You can make it more detailed, but three stages will suffice for us. In ideation you come up with ideas, in prototyping you start making crude test version of the final product, like a painter sketching their composition, and in finishing, you build the final design, in all its detail.

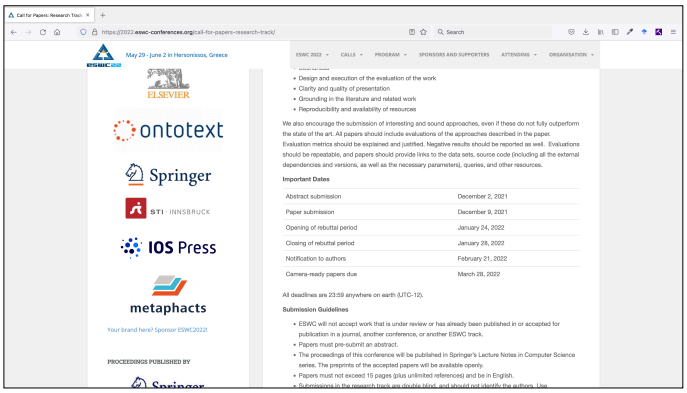
The main thing to say about this process, is that it can't be linear, left-to-right. You can not hope to get your ideas right in one go, and then make a perfect prototype. In fact the whole point of the prototyping stage is to uncover mistakes in your ideas before you move on to make the far more costly final product.

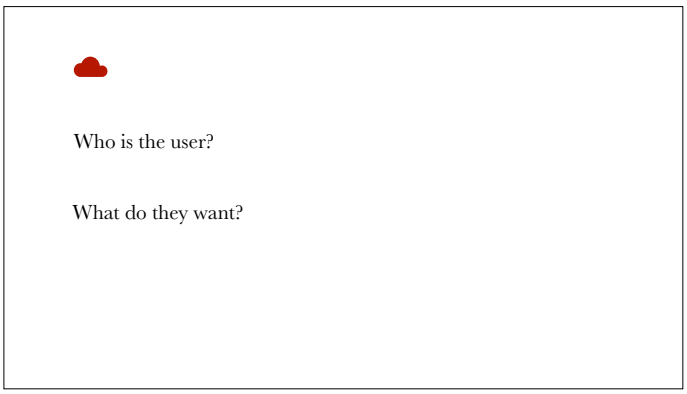
To do this effectively, you need to do two things. First, you need to iterate each stage. You need to

force yourself to evaluate what you've come up with so far and allow yourself to fix anything that doesn't seem right. This is difficult, since we are lazy creatures by default, so it helps to have tools that force you to look at your ideas and design in a new light.

Second, you have to allow yourself to go back. This makes people nervous, because it feels like you're moving in the wrong direction. But in order to develop a good product you have to find its problems and allow yourself to go back to the drawing board.

Where to to start? What is the first question you ask?





User stories

As a website visitor, I want to know the submission date, so that I can plan my paper.

This is an example of a user story. The default format is "As a <blank> I want to <blank> so that <blank>.

You can deviate from this format, but ...

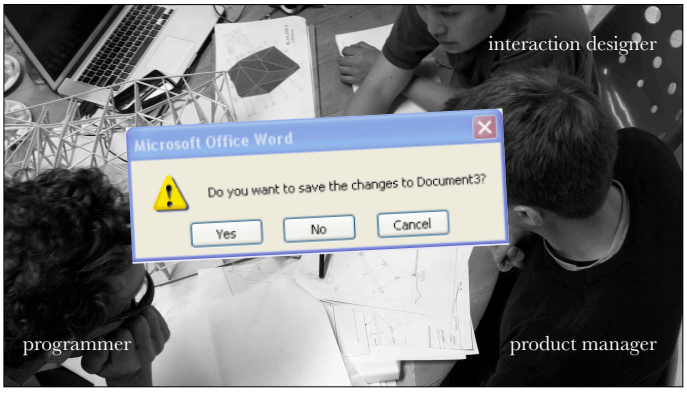
User stories

The rules:

- User stories are **short**: 1-3 sentences.
- They identify **a type of user** and **a goal**. Optionally, a context.
- They **do not** imply anything about how the problem is solved.

These are the main rules.

The last one is particularly important. The whole point of user stories is to discuss the problem without committing yourself to any particular solution.



Consider the following example: a design team is creating a document editor. The client demands that before closing the editor, a dialog appears, asking the user to save her documents. The designer disagrees: confirmation dialogs are bad practice. The designer suggests saving automatically. The client isn't sure: she may not want to save her changes, she may want to discard them. The programmer notes that the confirmation dialog is the easiest option, and the manager chooses the confirmation dialog. It is only a small problem after all.

As a user, I don't want to lose unsaved work.

As a user, I don't want to lose unsaved work when I close the application.

As a user, I don't want to lose unsaved work when my computer crashes.

When we ask people to write user stories, we can investigate the problem without committing ourselves to a solution that affects the rest of the application.

Here, the more user stories we write, the more it turns out that versioning system doesn't just satisfy this one user story, it satisfies many other. The more user stories you collect, the more you make the case for the versioning system.

As a user, I want to go back to previous versions of my document.

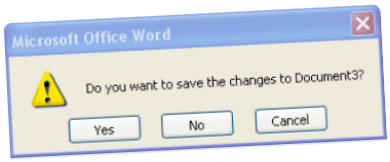
As a user in a team, I want to collaborate on a document with teammates.

As a user in a team I want to track what changes others make to a document.

As a user, I want to access a document from different devices.

In fact, you may find that you've made some incorrect implicit assumptions about your product. Here, we see that a desktop application may not be the way to go at all. A web application (with a versioning system) hits a lot more of these user stories.

By arguing over solutions, you end up committing yourself more and more to arbitrary choices. By developing user stories, you can investigate the design space without committing yourself.



Exercise. Write some user stories.

To make user stories even more applicable, we can move from users to general **stakeholders**. A stakeholder is anybody who wants something from from the product. For a webshop, both the buyer and the seller are stakeholders. So is the supplier, the government regulator and so on.

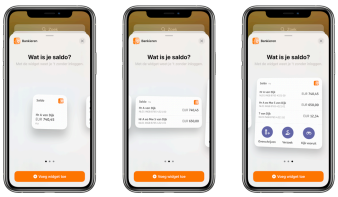
As an example, imagine a team designing a banking app. The security manager and the interaction designer get into a fight over the login procedure. The security manager wants an airtight, two-factor login procedure. The interaction designer worries that this will make the app unusable.

As a user, I want to log in without too much hassle.

As a security manager, I don't want logins to be unsecure.

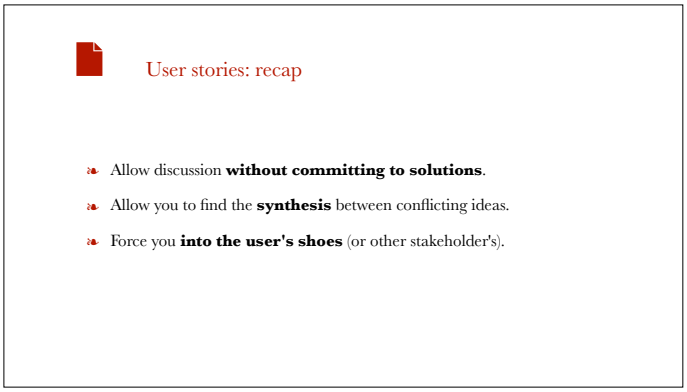
As a security manager, I don't want sensitive operations like transferring money to be insecure.

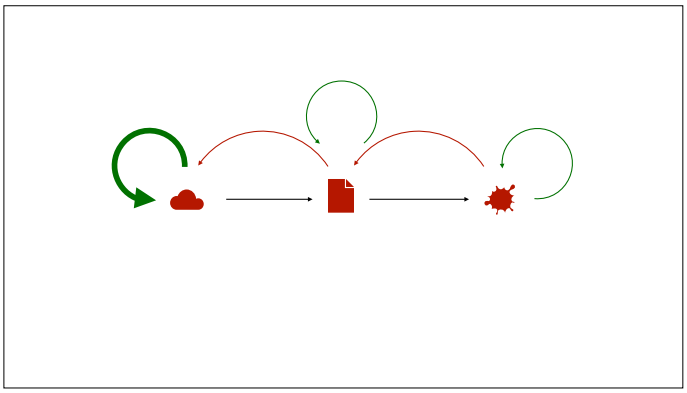
User stories show us that in design meetings two people can be right at the same time, even though their position conflict. The solution is not to compromise or pick one winner, but to find a solution that satisfies both user stories.



For instance, an app where you can check your balance without logging in (or on your home screen, without even opening the app), but where more sensitive actions, like transferring money require authentication.

This is a little like dialectics: there is a thesis and an antithesis, to apparently opposing ideas. The solution is to find a *synthesis*: a new idea that unifies the two.



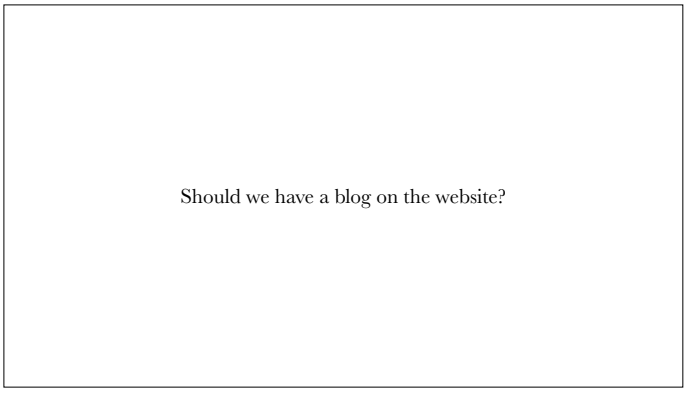


So, once we have our user stories, how do we develop them? There are a few tactics.

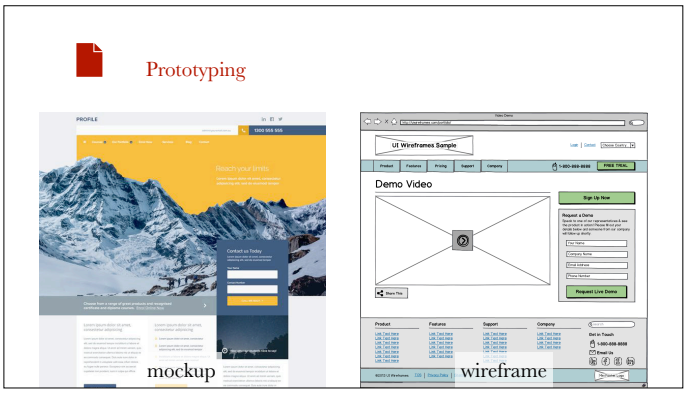
The first is simply structured discussion. Sit together with a bunch of people, and discuss the product. In such cases, it helps to have a **facilitator**. Somebody who doesn't have a position in the discussion, and whose only job is to provide structure. They'll do things like keep you on topic, stop the discussion from becoming too heated, and ask people to frame their position as a user story.

Another technique is **competitor research**. See what other people in similar fields are doing. Find out what's good about it and what isn't.

This can become especially powerful if you combine it with user testing. This does not need to be scientific. All you need to do is to find two or three people, give them a task and have them talk out loud while they use the competitors product. This will often generate a large number of ideas for the next iteration.



Exercise question.

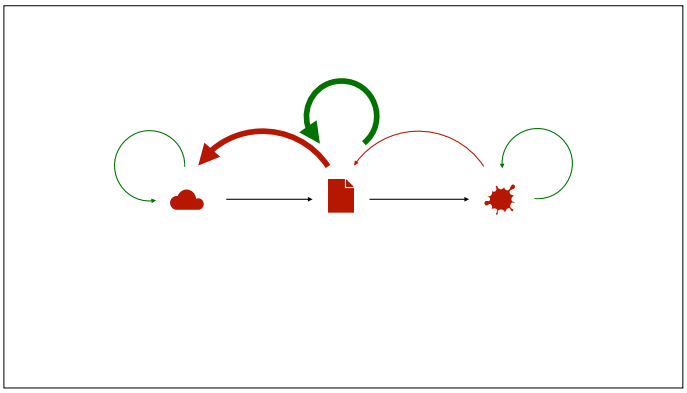


Here are two ways to design a website without actually building it. The first is a mockup, usually made in a program like Photoshop, Sketch or Figma. Its aim is to be a pixel perfect representation of what the website should look like. They look nice but they're expensive to produce. What's more, if you show somebody a mockup and ask for feedback, you will get comments on everything from the colors to the fonts to the splash photo. What you will never get feedback on, are the basic concepts of the site: the menu structure, which fields the form has, whether the copy text is long enough.

Put simply, a mockup is a terrible way to find out if your design is right. People don't focus on the right things, and even if you do find a basic mistake, you spent a lot of time on the mockup, so you'll be disinclined to redo all that work to fix your mistake. People may not even be confident enough to comment on your design. After all, you're the designer, you know about this stuff. And even if they spot a mistake, they may not want to cause you to have to redo all that work.

This is why we start with wireframes instead. These are simple, hand-drawn diagrams that communicate the basic layout and structure of the website, *without* any styling. This forces people to focus on the larger ideas, not the details. There's also no fear to tear the idea down, because wireframes are easy to draw. Finally, they level the playing field. Everybody can grab a pencil and draw a wireframe of their own. This makes it a far more equal communication tool.

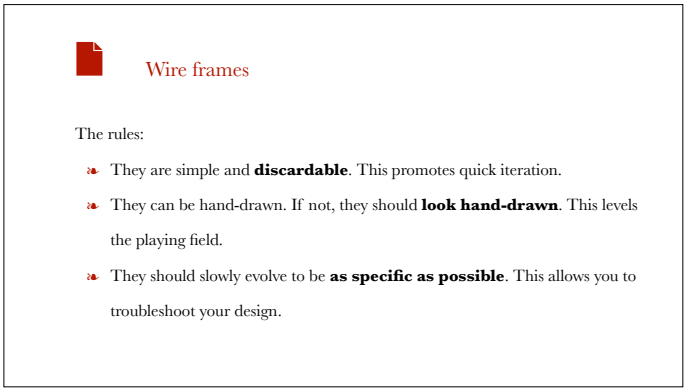
Pencil is a decent free tool for developing wire frames.



Prototyping helps you with these two arrows. While developing your prototypes you will immediately start to see the problems with your design. This allow you to go back to the user stories, and to work them out in greater detail.

You can then iterate your prototype, making it more detailed and uncovering more potential problems.

Here, again, **user testing** can be an immensely powerful tool. You take your paper prototype, and let a user "use it" give them some task, and ask them what they would do (which button they would click, for instance). The more detailed your prototype, the more extensively you can test.



Finishing

"Remember: when people tell you something's wrong or doesn't work for them, they are almost always right.

When they tell you exactly what they think is wrong and how to fix it, they are almost always wrong."

—Neil Gaiman

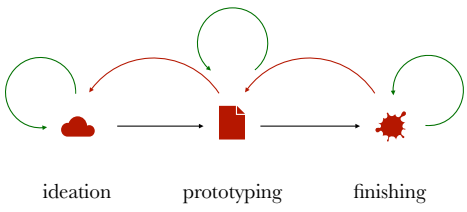
Intelligent discussions about design are hard. This quote by Neil Gaiman gives you most of the problem, and the solution in one package. If you are talking with a designer, you don't know what they've tired, their process or their philosophies. You can't tell them what they should do.

You can, however, tell them if something (a story, a website, a piece of music) doesn't work for you. That's helpful. But you have to leave it up to them to figure out how to fix it. Stick to those rules, and you have a healthy basis for talking to a designer.

Other media

Logos	Conferences	Book writing
Software	Course design	Hiring people
Articles	Grant writing	Running meetings
Videos	Talks and lectures	Other?

We don't have time to go into the details, but these principles can be transferred to many other things we make as academics. Ask me about the specifics if you're curious.



The most important ideas, on a high level:

Ideation. Explore ideas without committing to solutions. Who is the user and what do they want to do?

Prototyping. Find the problems with your design without investing too much effort. Make quick, cheap-looking mockups (like wireframes), and use them to **test** your design. Force yourself to go back.

Finishing. Put this off until you've given yourself a chance to debug your design. In discussions, respect the designer: focus what what doesn't seem to work, not how to fix it.