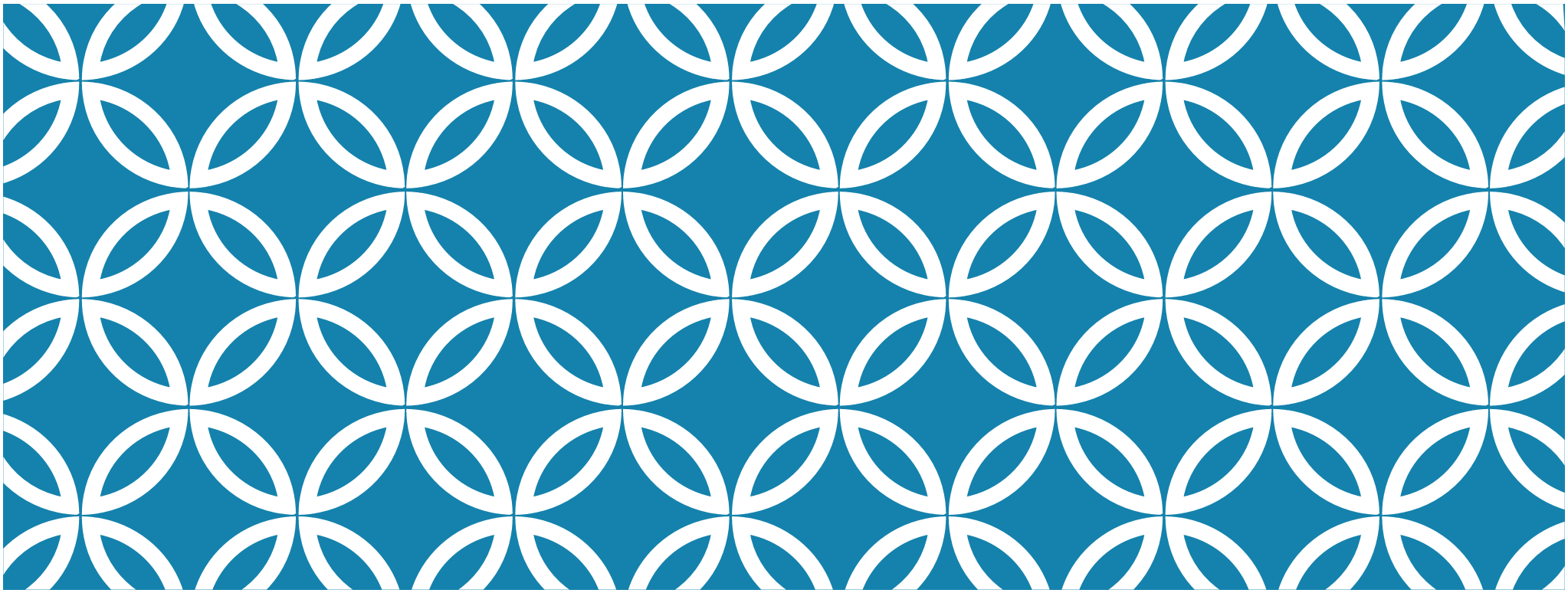


SOFTWARE DEVELOPMENT PROCESS

CS 346: Application Development

WHAT IS A PROJECT (AND WHY DOES IT MATTER?)

A project a set of tasks that collectively produce a desired outcome. e.g., redesigning my backyard, which includes planting flowers, cutting down trees.

Projects are often:

- More complex than individual tasks, with many tasks that need to be coordinated.
- Repeatable using the same steps (important if you care about reliability, reproducibility).

Regardless of their goal, projects usually include these high-level activities:

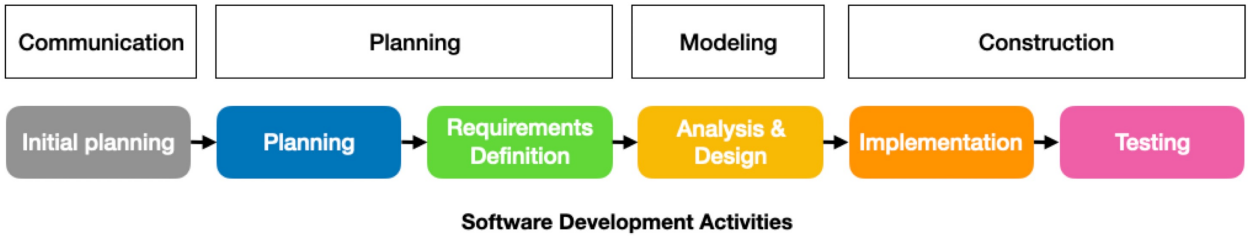
- **Communication** – discussion among people involved.
- **Planning** – agreement on the people, resources and objectives.
- **Modelling** – an abstract representation e.g., a sketch or blueprints.
- **Construction** – performing the required tasks!
- **Deployment** – delivering to a customer, getting paid etc.

PROCESS MODEL

We use the term **process model** to describe this structure of activities.

> "A process model defines the complete set of activities that are required to specify, design, develop, test and deploy a product, and describes how they fit together."

A **software process model** is a process model adapted to describe how we might build software systems. Here's an example of a simple software process model that adapts these common project activities.



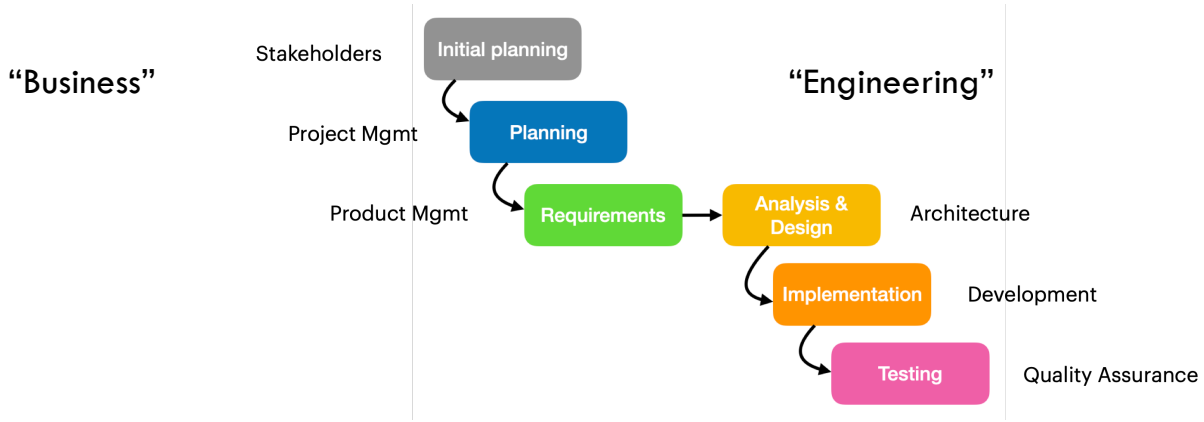
Software Development Activities

^ How do these fit? Can we just do them in order?

ADDING ORDER: WATERFALL

In a 1970 paper, Winston Royce laid out a mechanism for formalizing the large-scale management of software projects [Royce 1970], dubbed the **Waterfall model**. This process model envisions software production as a series of cascading steps.

- Stages were assigned to organizational units (e.g. Requirements "belonged" to Product Mgmt).
- Steps needed to be followed in order. There were often gating criteria when progressing through steps.



CHALLENGES

Key challenges when project planning using the Waterfall model:

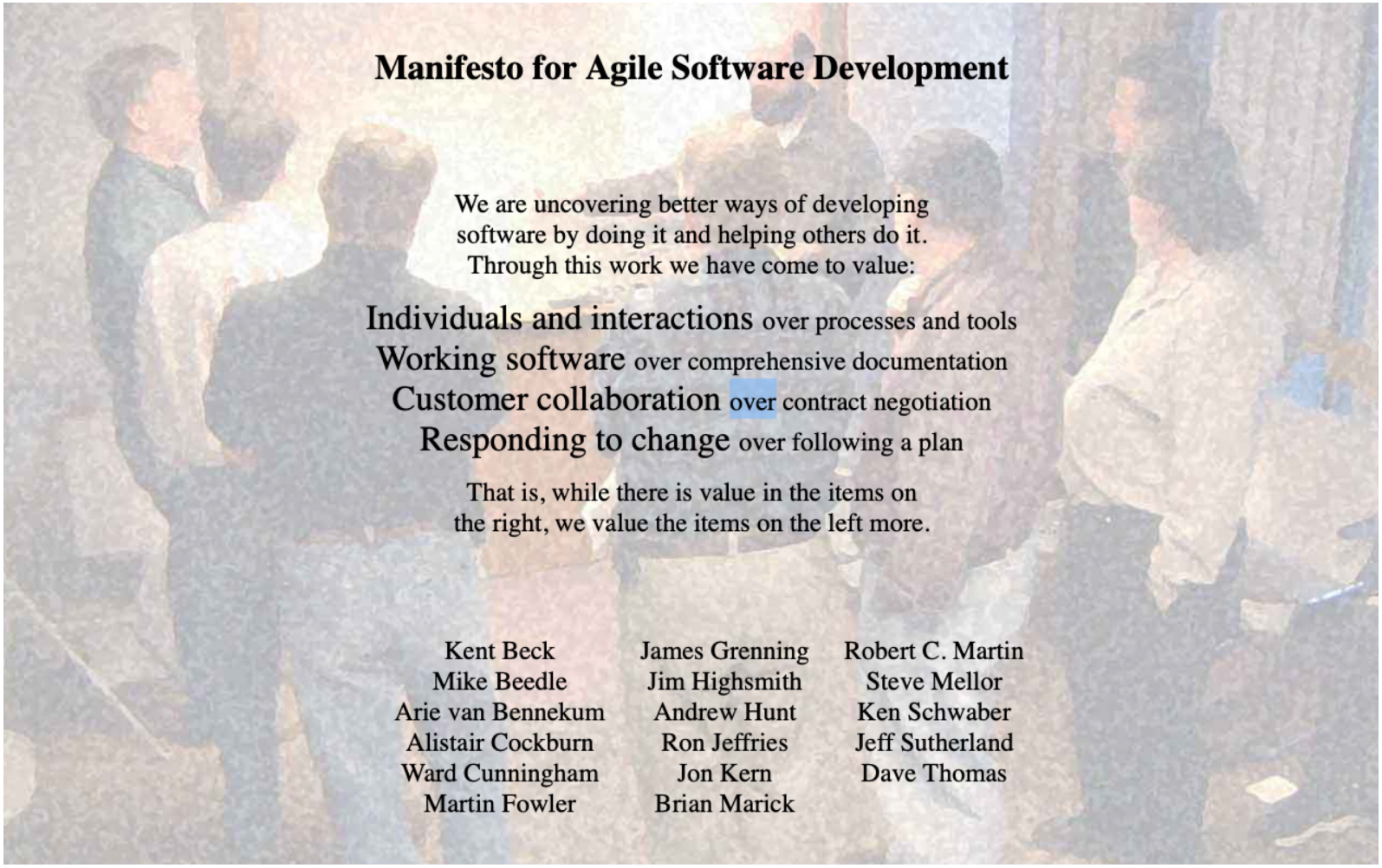
1. **Customer priorities will likely change** over the course of a project. It is difficult to predict in advance which software requirements will persist and which will change.

2. **Waterfall suggests that you can define requirements without technical analysis.** This is difficult, since technical analysis is necessary to know how long a task might take, risks etc. Your understanding of a problem *will* increase as you move through the project.

3. For many types of software, **design and construction are interwoven.** That is, both activities should be performed in tandem so that the design is proven as it is created.

4. **Analysis, design, construction, and testing are not as predictable as we might wish.** We're often designing something new or novel, which by its nature will be difficult to predict!

These challenges make it extremely difficult for *any* model to accurately describe software production. There's often "too many unknowns" for precise up-front planning.



https://agilemanifesto.org/ (2001)

WHAT IS AGILE?

> [Agile Software Development] **encourages team structures and attitudes that make communication easier** (among team members, business-people, and between software engineers and their managers). It emphasizes **rapid delivery of operational software,** but also recognizes that planning has its limits and that a **project plan must be flexible.**
>
> — Pressman & Maxim 2020.

There are MANY Agile process models e.g., Crystal, Lean, Scrum, XP.

They share some principles:

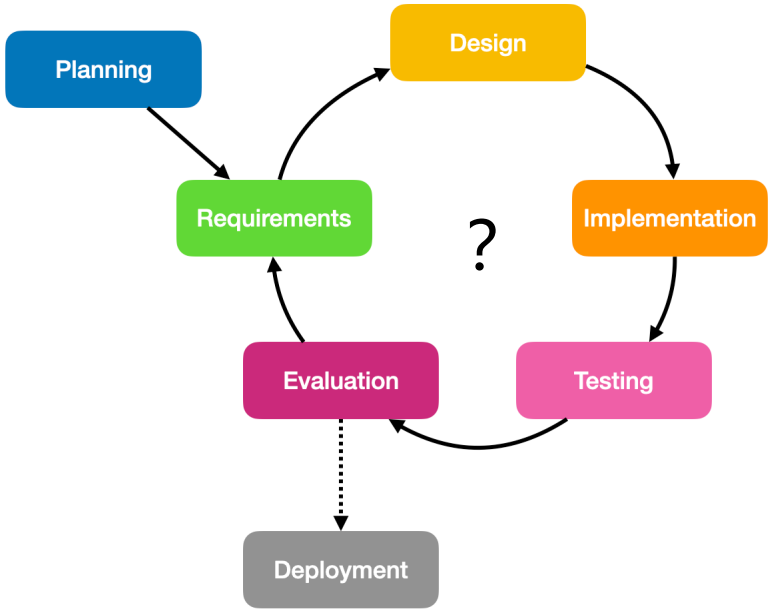
- Focus on teams! Communication, honesty and collaboration are critical.
- Be responsive and ready to adapt as requirements change.
- Focus on quality at every step. Never "just get it working". // why not?

ITERATIVE MODELS

In modern software development, we now view development as an **iterative** process.

Instead of building a "complete" system and then asking for feedback, we instead attempt to deliver features in small increments.

This allows us to respond to changing requirements and provides a regular mechanism to check our progress and get feedback.



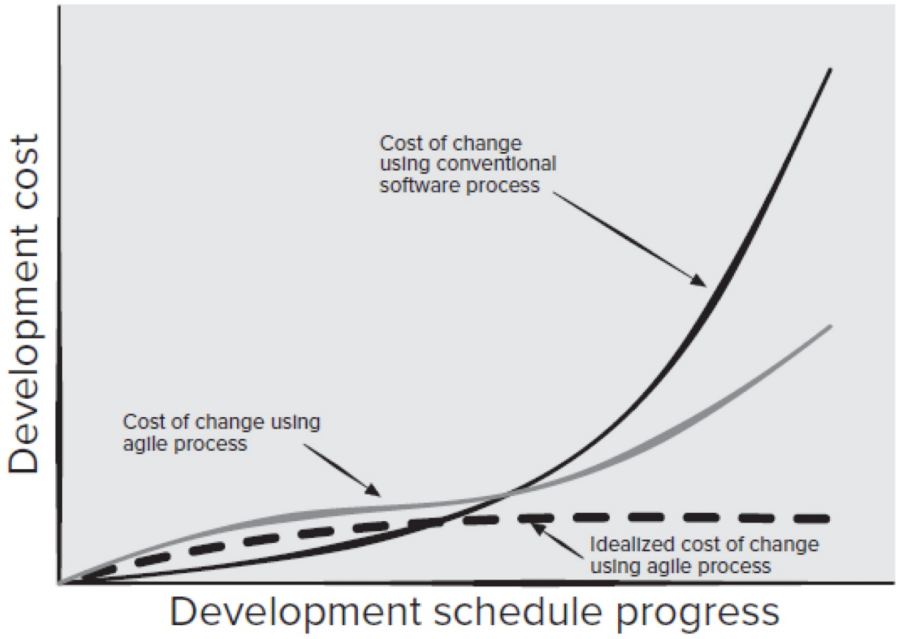
This is a first-draft of a process model.
It's probably *not* sufficient to just iterate like this…

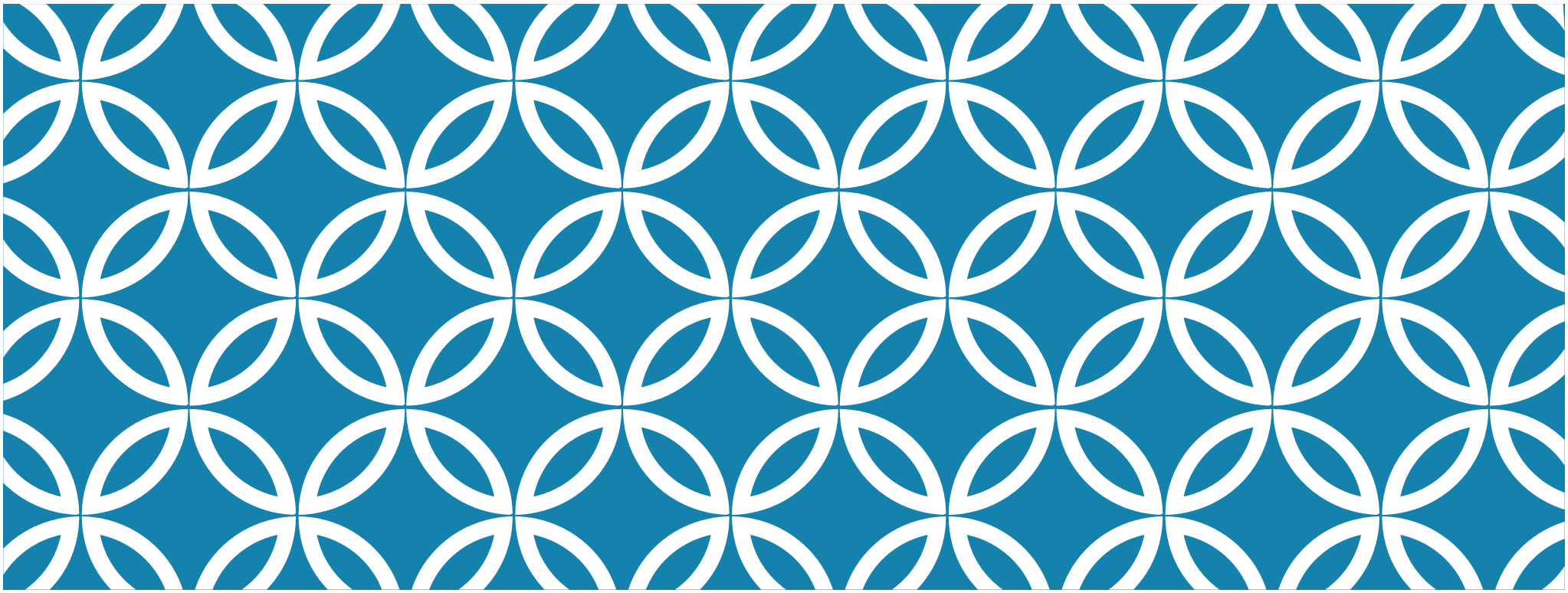
WHY ITERATE? TO IDENTIFY CHANGES EARLY!

The cost of change increases nonlinearly as a project progresses.

Agility allows for project and requirements changes as the project progresses and reduces the likelihood of breaking late-project changes.

In other words, iteration encourages you to make required changes earlier in the process, when the cost of doing so (in money, time, effort) is lower.





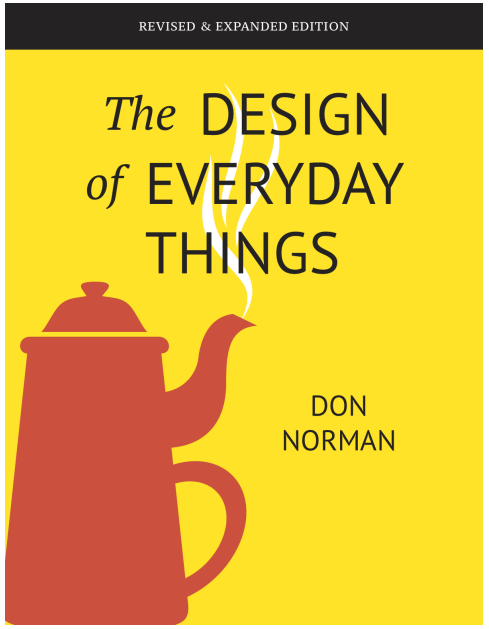
DESIGN MODEL > UCD

CS 346: Application Development

DESIGN & THE BIRTH OF UX

In the mid 1980s, while on sabbatical at the Applied Psychology Unit at Cambridge University, Don Norman found himself puzzled by the light switches he encountered. Why light switches? The switches moved in the opposite direction to what he expected ("up" denotes "on" by North American convention, but not in the UK).

These ponderings eventually led to the publication of The Design of Everyday Things in 1988 [Norman 1988, 2013]. This book launched the UX movement.



"I walk around the world and encounter new objects all the time. How do I know how to use them?" — Don Norman

PRINCIPLES OF INTERACTION

Norman illustrates what he considers the fundamental principles of interaction:

Affordances: cues that allow a person to figure out what an object does without any instructions. e.g., a door with a "pull" handle (formally called "perceived affordance", which positions then as properties of the perceiver, as much as the object itself).

Signifiers: visible signs or sounds that communicate meaning. They indicate what is happening, or what can be done. e.g., a door that says "push" on it.

Mapping: the relationship between two sets of things that conveys associated meaning. e.g. grouping controls and putting them next to the thing that you want to manipulate.

Feedback: the result of an action, communicated to a user. e.g., a button changing color when depressed; a walk indicator that lights up when the walk button is activated.

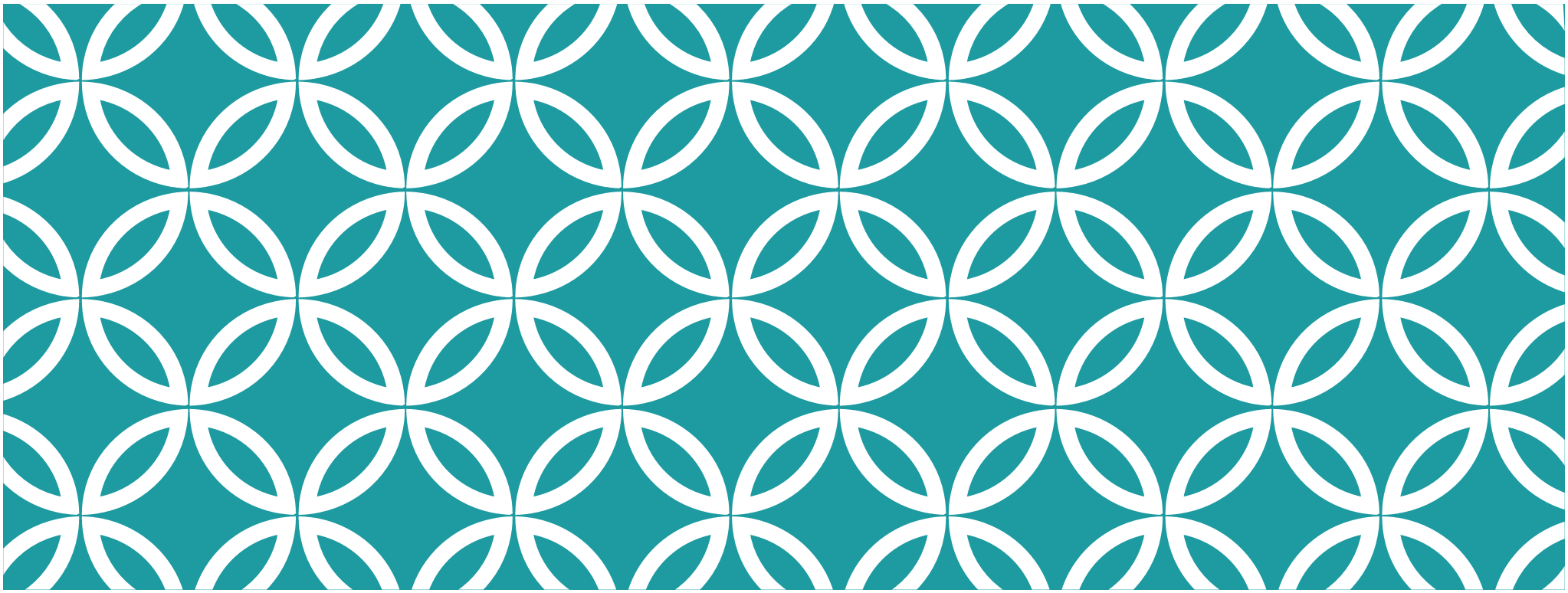
Conceptual models: simplified explanations of how something works; related to mental models that people automatically form when working with an object to help them explain and predict its behavior.

USER-CENTERED DESIGN (UCD)

"The first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use."

– Don Norman & Jacob Nielsen

User-Centered Design is a design process that focuses on **addressing the needs of users first,** to ensure that we're solving the correct problem for them. UCD uses a mixture of investigative tools (e.g., interviewing) and generative methods (e.g., brainstorming) to design systems and collect user feedback through the entire process.



UCD > IDENTIFYING USERS

CS 346: Application Development

IDENTIFYING A PROBLEM

Application software is intended to solve a real-world problem for a user.

Building something "because you can", or "because you find it interesting" doesn't guarantee that it is useful for anyone else.

We should attempt to:

- (a) identify a body of users that you know well or have access to.
- (b) find some problems those users have that haven't been adequately addressed yet (the key word is *adequate* - there's often room to offer an improved solution).

How do you figure this out? Talk to people!

Find out what challenges them at home/work?

INTERVIEWING USERS

You need to understand the details about the problem your set of users has. To do this, you need to speak with them.

- Determine your target user group.
- Find people that fall into this group.
- Get their opinion! (Yes it's that simple).

Ideally, interview 5 users [Nielsen 2000]*

- Ask open-ended questions.
- Use follow-up questions to to make sure that you understand what they are saying.
- Take lots of notes (you can even record the interview and review afterwards).

* 5 users is enough to identify 75% or more of your usability problems.

INTERVIEW QUESTIONS

Good interview questions

- "We are building X. Would it be useful to you in your role as <something>?"
- "If not, what could we change to make it more useful for you. "
- "What features would you like to see added? What is "missing" from your perspective?"
- "How would you expect X to work?"

Poor interview questions:

- "I built this. Isn't it cool?"
- "You want *what*? That's a dumb idea."

DESCRIBING YOUR USERS: PERSONA

A **persona** is an archetypal description of a user of your product. Creating personas can help you step out of yourself… It can help you to identify with the user you're designing for.

There are three approaches to take when defining a persona:

1. **Goal-based persona**: represents a user's attitudes, situation, and their goal in that moment.

2. **Role-based persona**: focuses on the role that the user plays in the organization, and designs from the perspective of expectations of that role (somewhat goal-driven as well).

3. **Engaging persona**: incorporate both goal and role-directed personas. Engaging personas are designed so that the designers who use them can become more engaged with them.

Personas are created after your interviews. Use your interviews to identify characteristics that you can build into your persona.

EXAMPLE PERSONAS

Bio: Fictional name, job title, company, job description, family status, and more. Include any details that can help your team understand the persona better.

Demographic info: Fictional age, gender, income, education, location, and other background information.

Image: A vivid image can help your team picture your users clearly and help establish a consistent understanding of the target.

Goals & motivations: You should also make it clear what your audience wants to get or to do with your product.

Max Banker

BIO

Max is a business studen
has had multiple internsh
institutions. He's a capabi
computer for browsing th
paper.

GOALS

Max has a lot of trouble k
has to work on. Anything
notes would be useful.

FRUSTRATIONS

Incentive: Low
Fear: Low
Achievement: Low
Growth: Low
Power: Low
Social: Low

QUOTES

"I'm too busy to learn son
"I hate my computer (Wir

DOMAIN AWA
Not much. He scribble
paper notebook at

Jessica Coder

BIO

Jessica is a professional software developer. She graduated from UW with a BCS and a minor in CO. She is the tech lead for her current project at Shopify.

GOALS

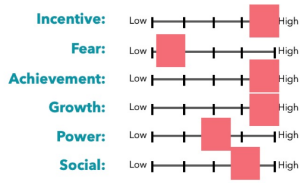
She is a packrat for saving technical information - code snippets, articles online.

FRUSTRATIONS

She finds it difficult to keep track of everything that she needs to work with in a day.

MOTIVATIONS

Anything to make her more efficient.



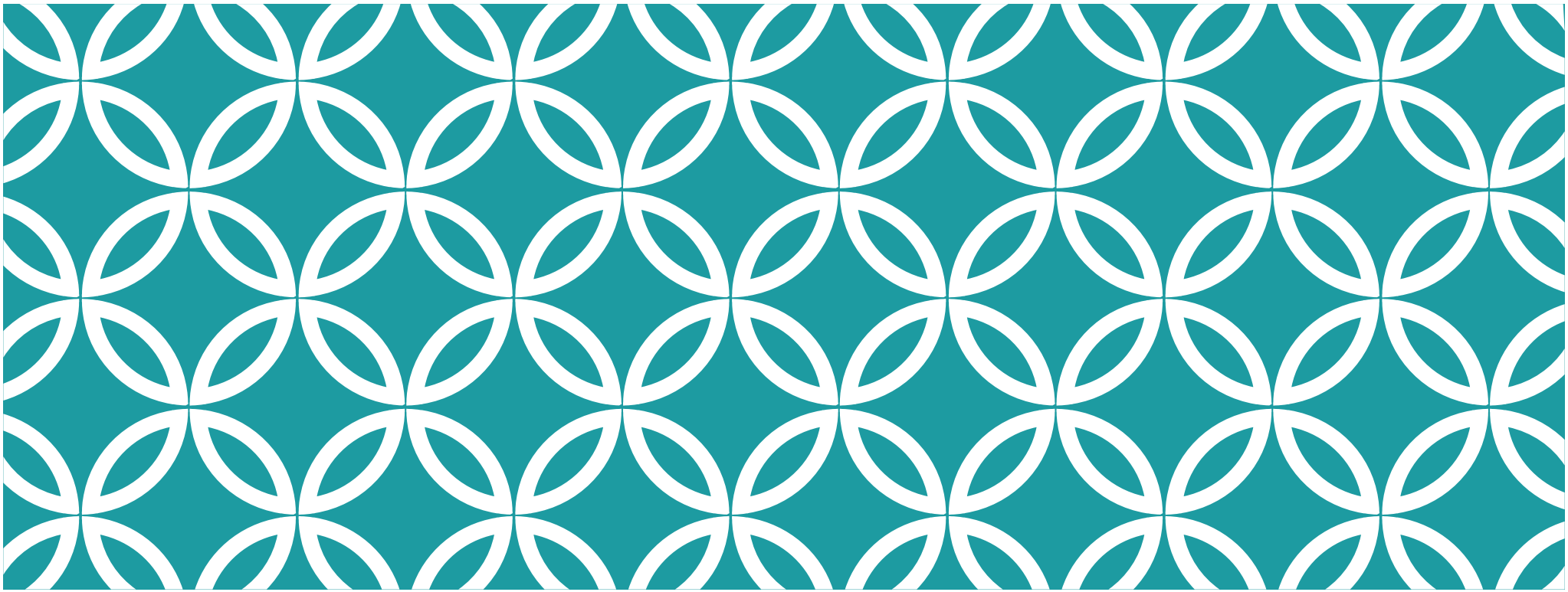
Age: 26
Occupation: Software Developer
Status: Single
Location: Toronto
Archetype: Developer
Personality: "Gadget person"; loves new tech; Android over iOS because she can tweak it.

QUOTES

"If I have to use a mouse, that just annoys me. I'm faster with a keyboard. VIM all the way".
"Information should be easy to access, and always accessible. Having everything searchable without breaking the flow would be amazing."

DOMAIN AWARENESS
Currently uses markdown notes and manually "greps" based on keyword.

TECH KNOWLEDGE
Expert in software, technology. Expects an advanced, polished solution.



UCD > USER STORIES

CS 346: Application Development

COLLECTING USER STORIES

From your user interviews, you should have identified some basic activities that are important to your users! These will often come in the form of stories or narratives…

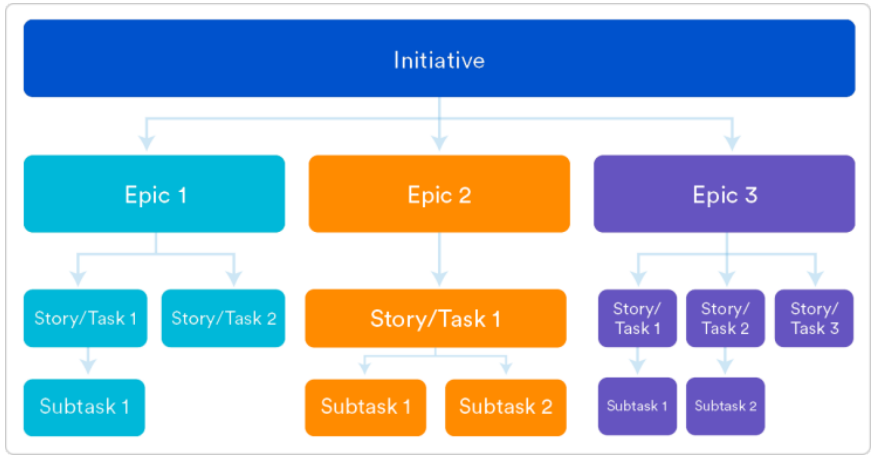
User-Stories are a description of activities that are performed with a system, from the perspective of a user.

- "As the HR manager, I want to create a screening quiz so that I can understand whether I want to send possible recruits to the hiring manager."
- "As a user, I can indicate folders to skip when backing up, so that my backup drive isn't filled up with things I don't need".

Use these stories to identify problems and desirable solutions. e.g., Jessica and Max both have problems keeping track of information and would be amenable to a mobile solution.

TERMINOLOGY: INITIATIVES, EPICS, USER STORIES

Related user stories can be grouped under a heading called an **epic.** An epic is simply a larger body of related user stories. Similarly, an **initiative** is a collection of related **epics.** These terms are useful in larger products, but not used in this course.



https://www.atlassian.com/agile/project-management/epics-stories-themes

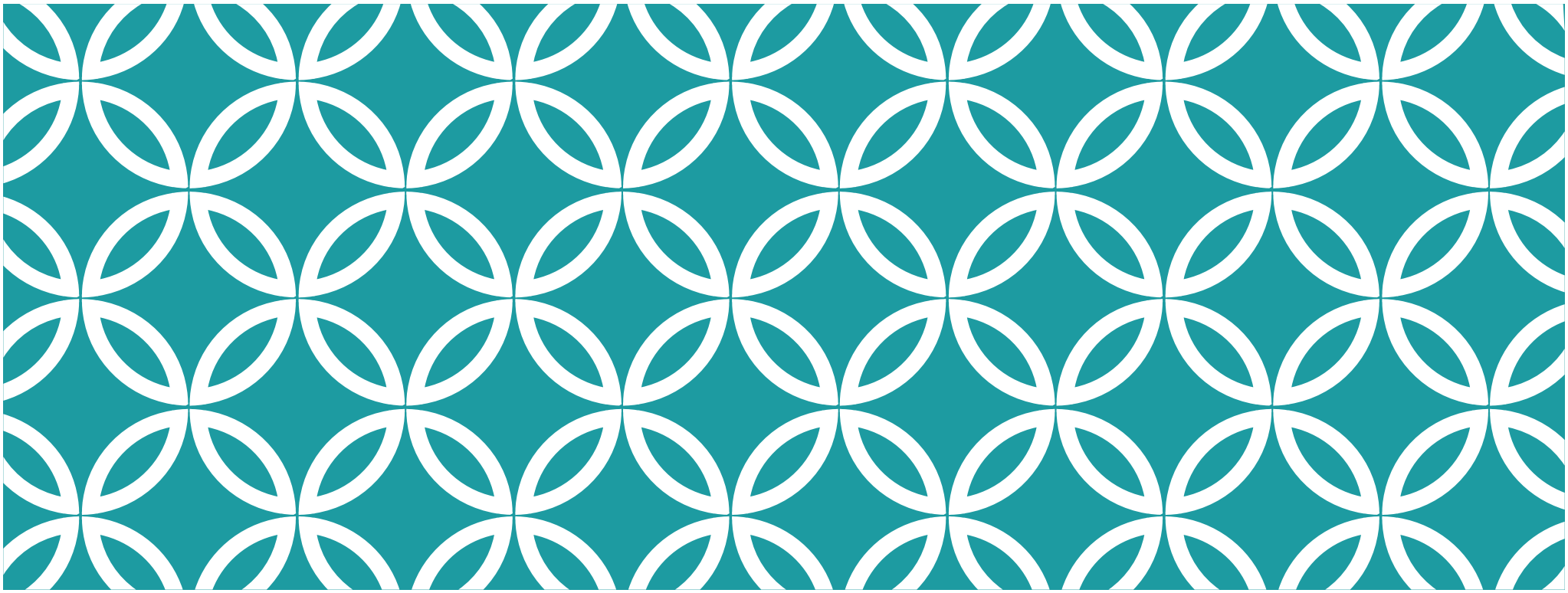
WHAT TO DO WITH USER STORIES?

User stories identify desired features from the point-of-view of your user. Analyze them to determine what features (or sets of related features) you would require to meet that user's needs.

From there, enter issues in your GitLab project corresponding to each feature that you wish to implement. Add details provided by your user as background and motivation for the design of that feature.

When deciding which features to include:

- 1. Focus on the problem you are trying to solve. Make sure your feature addresses it.
- 2. Consider features that support each other. e.g., copy-paste and keyboard shortcuts, or cloud data storage and authentication.



UCD > PROTOTYPES

CS 346: Application Development

ITERATIONS: PROTOTYPING

It's helpful to iterate on your product early in the project to ensure that you are planning out the features appropriately.

One good strategy is to develop early mockups that demonstrate the layout and structure of your application. These should NOT be 100% functional, but instead serve as placeholders for the later functionality.

We call these early iterations **prototypes**. We recommend that before you start coding, you:

1. Create a prototype of your projects interface (since the UI is what the user will see, so it's the best way to express how functionality will work with respect to the user).

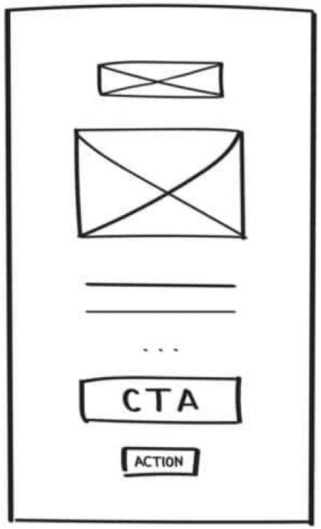
2. Show to your user, collect feedback and revise.

PROTOTYPES

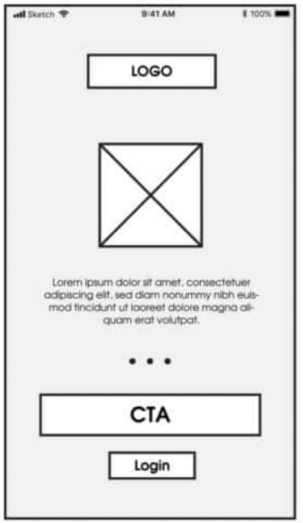
When iterating over an interface, focus on low-fidelity prototypes.

Low-fidelity prototypes are deliberately simple, low-tech, and represent a minimal investment.

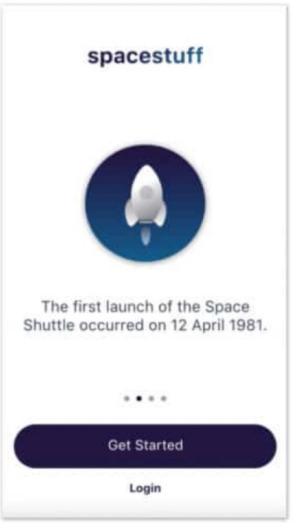
- You can sketch something on paper.
- Many online tools help you build wireframe diagrams that you can demo.
- You can even make them semi-interactive to test progression through the interface.



Low



Medium



High

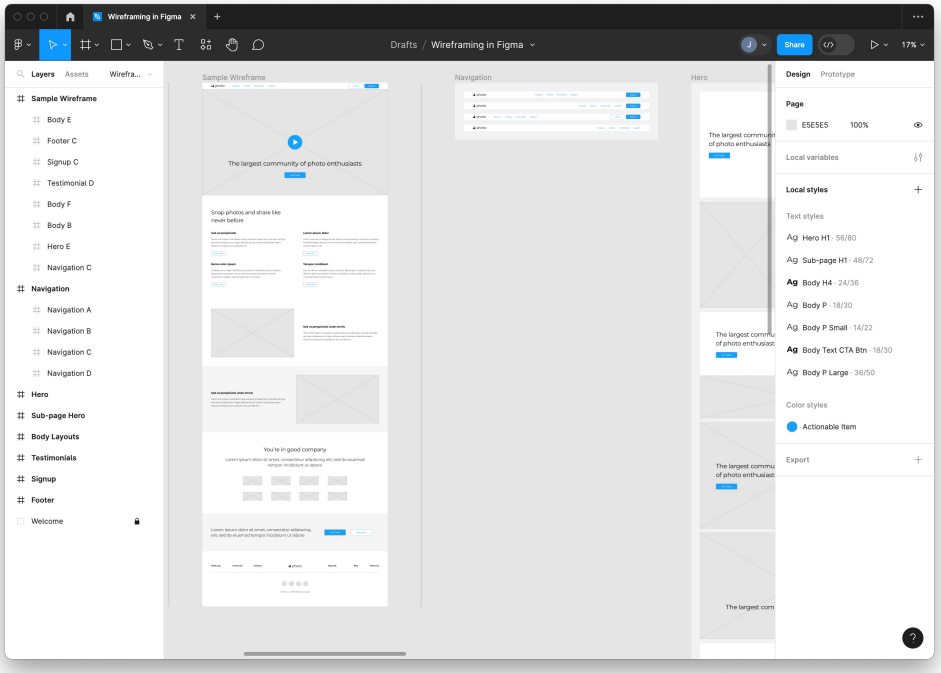
ONLINE TOOLS

Figma is a very popular prototyping tool (desktop, mobile, web).

- Use it to mock screens and interactions.
- Can build design-specific UI designs (e.g., iPhone, iPad, desktop).
- Makes iteration much easier compared to paper prototyping.
- Can export code(!)

Other options

- Omnigraffle (mac)
- Balsamiq (win, mac, web)
- Hand-drawn diagrams



https://www.figma.com/

FEEDBACK!

Remember those users?

You really want to get their feedback before/during your project.

- When determining requirements ("did we understand this correctly?").
- When building prototypes ("what do you think this does? Is it clear?")

You will submit a Design Proposal, where your user (TA) will provide written feedback.

- Also solicit their feedback during demos to help fine-tune your designs!

Max Banker

BIO

Max is a business studen
has had multiple internsh
institutions. He's a capabi
computer for browsing th
paper.

GOALS

Max has a lot of trouble k
has to work on. Anything
notes would be useful.

FRUSTRATIONS

Incentive: Low
Fear: Low
Achievement: Low
Growth: Low
Power: Low
Social: Low

QUOTES

"I'm too busy to learn son
"I hate my computer (Wir

DOMAIN AWA
Not much. He scribble
paper notebook at

Jessica Coder

BIO

Jessica is a professional software developer. She graduated from UW with a BCS and a minor in CO. She is the tech lead for her current project at Shopify.

GOALS

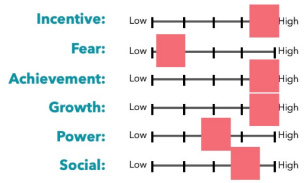
She is a packrat for saving technical information - code snippets, articles online.

FRUSTRATIONS

She finds it difficult to keep track of everything that she needs to work with in a day.

MOTIVATIONS

Anything to make her more efficient.



Age: 26
Occupation: Software Developer
Status: Single
Location: Toronto
Archetype: Developer
Personality: "Gadget person"; loves new tech; Android over iOS because she can tweak it.

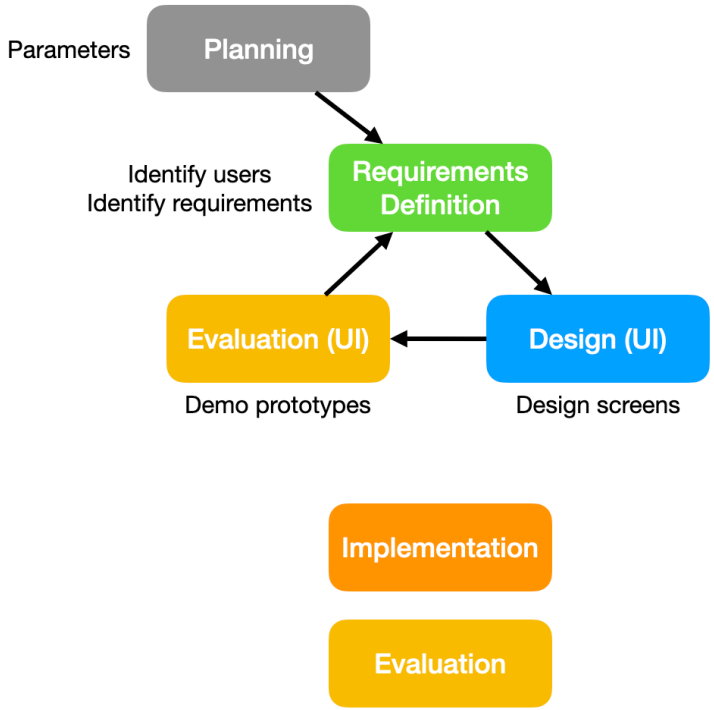
QUOTES

"If I have to use a mouse, that just annoys me. I'm faster with a keyboard. VIM all the way".
"Information should be easy to access, and always accessible. Having everything searchable without breaking the flow would be amazing."

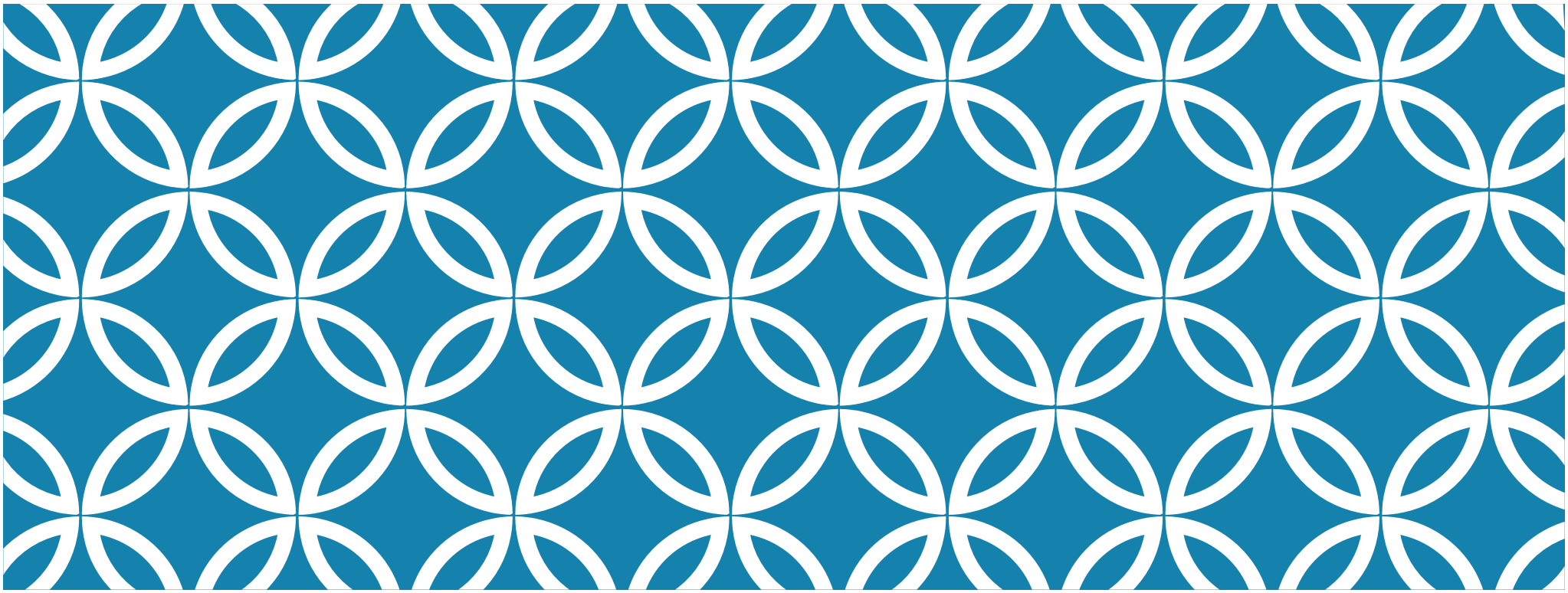
DOMAIN AWARENESS
Currently uses markdown notes and manually "greps" based on keyword.

TECH KNOWLEDGE
Expert in software, technology. Expects an advanced, polished solution.

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)



Partial SDLC including requirements and design only.



PROCESS MODEL > SCRUM

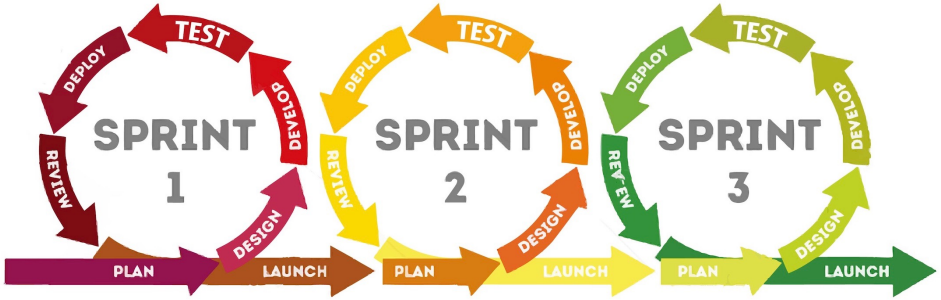
CS 346: Application Development

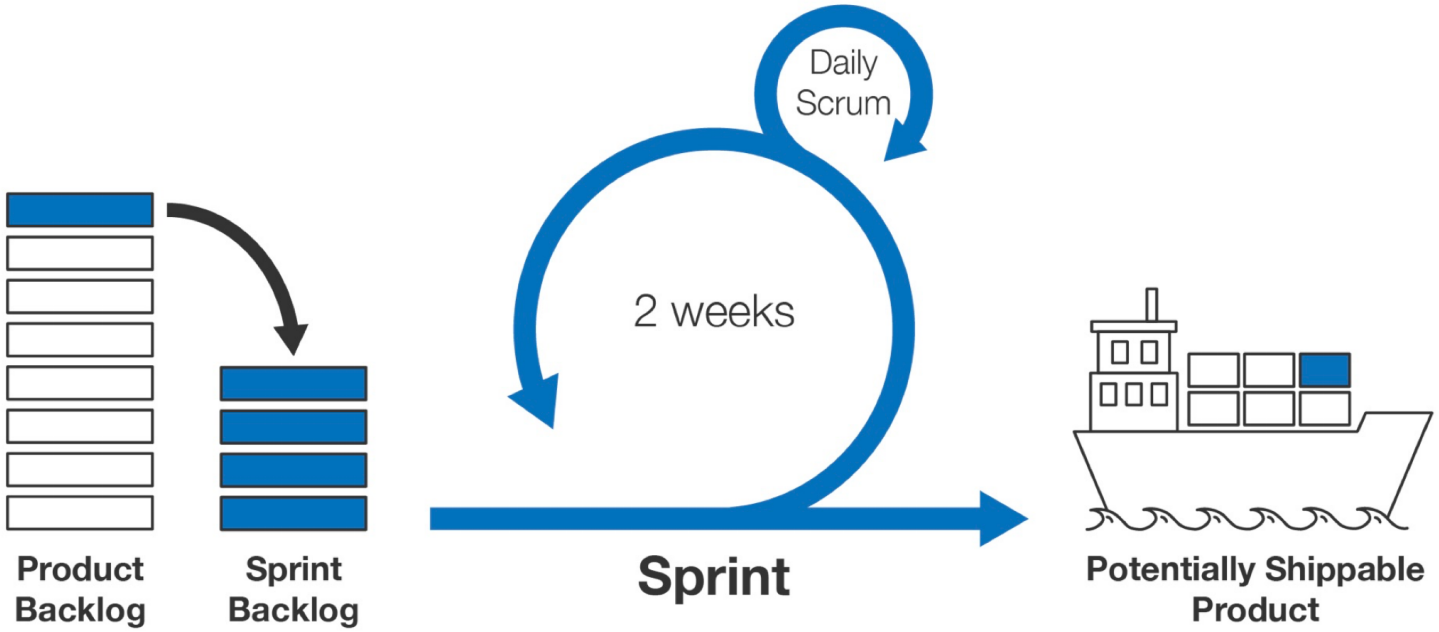
AGILE PROCESS MODELS: SCRUM

Scrum is a popular Agile process model that we'll use.

Scrum breaks down a project into fixed-length iterations called **sprints** (typically 2-4 weeks in length for each sprint). Sprints are defined so that you iterate on prioritized features in that time and produce a fully-tested and shippable product at the end of the sprint.

Typically, you iterate until you and the customer together decide that you are "done".





https://kunzleigh.com/about/our-approach/

Product Backlog is a list of all possible features and changes that could be considered.

- Product Owner is responsible for gathering and placing requirements in the product backlog.

Sprint Backlog is the set of features that are assigned to a sprint.

- Scrum Master is the person that helps facilitate work during the sprint (e.g., Technical Lead).

OUR ITERATIONS

In our course, we will have four sprints, each 2-weeks long. Each includes:

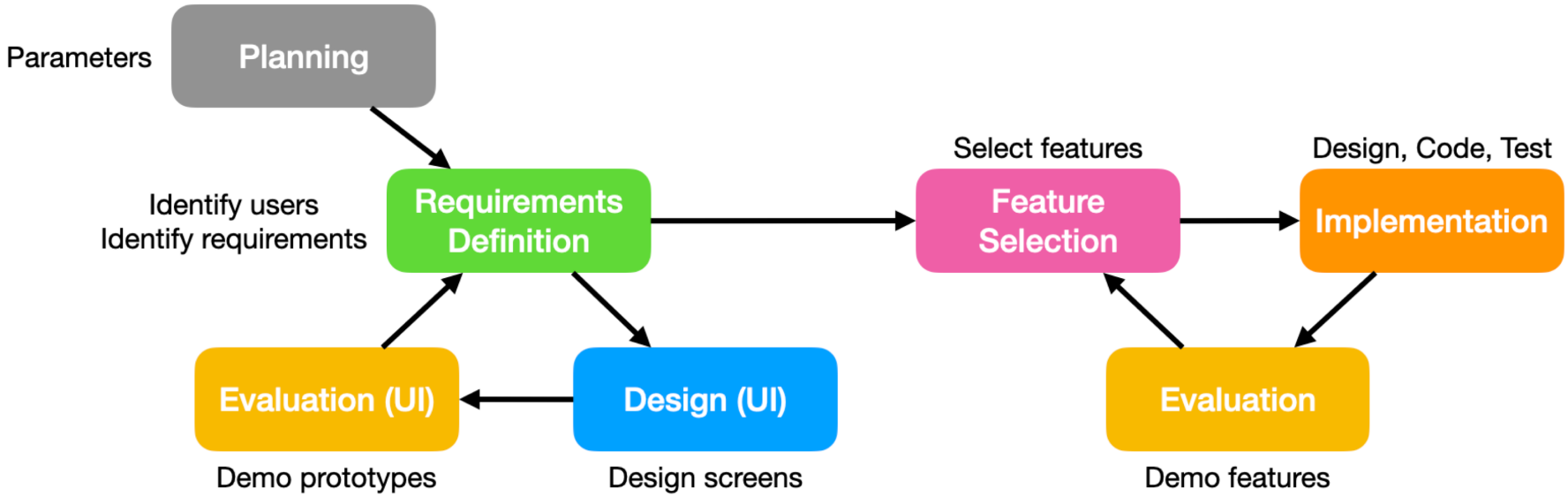
Feature Selection. On the first day of the Sprint, the team meets and selects features. Issues are moved from the Product Backlog into the Sprint Issues list and assigned. You are committing to work that you can complete in the upcoming sprint.

Implementation. During the sprint, the team iterates on their features. As each feature is completed, unit tests are written/passed. When complete, the issue is closed.

Evaluation. At the end of the Sprint, the team meets with the Product Owner (TA) to demo what they have completed and get feedback. Only completed work is shown. Issues that are not completed are moved back into the Product Backlog to be reconsidered.

Retrospective. The team should also reflect on what worked well, and what could improve. You should always be looking for ways to improve how you manage your project.

SOFTWARE DEVELOPMENT LIFECYCLE



Adding sprints to SDLC for managing iterations.