COMP1100 — Week 4 Notes

Table of contents

Minimum Viable Products and Prototyping **1**
- Outline . . . 1
- Minimum viable products . . . 2
- Prototyping . . . 2
- Have code generation tools killed wireframing? . . . 2
- Key takeaways . . . 2
- Next week . . . 3

Reference material **3**
- Minimum Viable Products (MVPs) . . . 3
- Prototyping . . . 5

Minimum Viable Products and Prototyping

Week 4 seminar. See comp1100[1] for course logistics.

Outline

1. Submissions and feedback
2. Minimum viable products — what they're for, the six principles, the types
3. Prototyping — the four axes, paper prototypes, wireframes, Wizard of Oz
4. The economics of design

[1] courses/comp1100/comp1100.pdf

Minimum viable products

See minimum-viable-products[2] — what an MVP is for (it exists **to test the ability of the product to meet minimal customer needs**, not to earn revenue and not to find bugs), the six principles, the five types, and the prototype-vs-MVP table.

The distinction to hold onto: an MVP is **not a full product, but it is a product, not a prototype**. A prototype is shown to people; an MVP is released to them.

> "By far the dominant reason for not releasing sooner was a reluctance to trade the dream of success for the reality of feedback."
>
> — Kent Beck, *Approaching a Minimum Viable Product* (2009)

Chapter 6 of the course notes covers this.

Prototyping

See [prototyping] — prototyping as a form of experimentation inside the design cycle, the four axes (representation, precision, interactivity, evolution), paper prototyping, wireframes, Wizard of Oz, and the economics-of-design cost ladder.

The ladder is the practical takeaway: low fidelity prototypes are *super cheap* and focus on flow and interactivity; high fidelity ones are *quite cheap* and focus on look, feel and usability; code prototypes are *more expensive* and focus on code feasibility; the MVP is *most expensive* and focuses on getting to market. Spend the cheapest amount that answers your question.

Have code generation tools killed wireframing?

An open question, with arguments on both sides — see [prototyping]. The strongest point for "no": wireframes are *intentionally* low fidelity, to get feedback on the user **experience** rather than the user **interface**.

Key takeaways

- MVP = the most important features.
- MVP purpose: **experiment!**
- Simple and easy to build.
- Different MVPs allow us to run different experiments.

[2] courses/comp1100/minimum-viable-products.pdf

Next week

Seminar: 2025-08-25-the-history-of-computing[3] and 2025-08-25-prompt-engineering[4] — the last seminars of the semester.

Reference material

Minimum Viable Products (MVPs)

What an MVP is for

The seminar opens with a quiz: what is the *primary* purpose of a minimum viable product?

- **(a) To test the ability of the product to meet minimal customer needs.**
- (b) Get as much revenue as we can as early as we can.
- (c) Identify bugs in a product with a fixed feature set that meets a set of requirements.
- (d) All of the above.

The answer is (a). An MVP is an **experiment**, not a revenue play and not a QA exercise.

It is not a full product — but it *is* a product

The distinction the course insists on: an MVP is not a full product, but it is a **product, not a prototype**. It has to actually work.

> "By far the dominant reason for not releasing sooner was a reluctance to trade the dream of success for the reality of feedback."
>
> — Kent Beck, *Approaching a Minimum Viable Product* (August 2009)

That quote is the same failure mode Paul Graham describes in lean-startup[5], from the other direction: shipping is avoided because shipping produces information, and information can be disappointing.

Prototype vs MVP

[3] courses/comp1100/2025-08-25-the-history-of-computing.pdf
[4] courses/comp1100/2025-08-25-prompt-engineering.pdf
[5] courses/comp1100/lean-startup.pdf

Prototype	MVP
Use for visualising and testing product ideas	Functional product with core features
Often not actually functional	Must be functional (even if minimal)
Super fast to build and test	Requires working code, deployment, etc.
Aims to test and refine ideas	Aims to test **market fit**
Ideas tested in a "controlled" environment	**Released to the market**

The dividing line is the last row: a prototype is shown to people, an MVP is released to them.

Six principles for an MVP

1. **KISS** — keep it simple, stupid.
2. **Fast to build.**
3. **Positive user experience.**
4. **Early adopters** — build for them, not the majority (see customer-discovery[6]).
5. **Iterate and refine.**
6. **Set success metrics.**

Principle 6 is the one that connects back to test-cards-and-learning-cards[7]: without a success metric agreed in advance, the MVP produces impressions rather than evidence.

Types of MVP

Not all MVPs are software:

Type	What it is
Pre-order	Take orders before the thing exists — tests willingness to pay directly
Show and tell	Demonstrate the concept and measure the reaction
Audience building	Build the audience first, product second
"Wizard of Oz" product	The customer-facing experience is real; the machinery behind it is a human
Digital product	An actual, minimal, working piece of software

Different MVP types let you run **different experiments** — choose the one that tests the hypothesis you actually have. If the open question is "will anyone pay?", a pre-order tests it far faster than building the product would.

[6] courses/comp1100/customer-discovery.pdf
[7] courses/comp1100/test-cards-and-learning-cards.pdf

Competitors

The MVP has to be minimum *relative to what already exists*. The value proposition diagram from value-proposition-canvas[8] returns here — what your customer wants, what you do well, and **what your competitors do well**. Features your competitors already deliver well may be table stakes rather than optional.

Key takeaways

- MVP = the **most important features**.
- MVP purpose: **experiment!**
- Simple and easy to build.
- Different MVPs allow us to run different experiments.

See [prototyping] for the cheaper things to build first.

Prototyping

Prototyping as experimentation

Prototyping forms a key part of the design cycle, can be **exploratory or evaluatory**, and — the point most often missed — **does not have to start with writing code**.

It sits inside the design loop:

```
Establish customer problems → Design value propositions
        → Prototype value propositions → Evaluate value propositions → MVP
```

So a prototype is a way of running an experiment (experimentation[9]) about a value proposition, before committing to the MVP[10].

[8] courses/comp1100/value-proposition-canvas.pdf
[9] courses/comp1100/experimentation.pdf
[10] courses/comp1100/minimum-viable-products.pdf

Axis	Range

Four axes

Prototyping can be thought of along four axes:

Axis	Range
Representation	Online vs offline; high vs low fidelity
Precision	Level of detail
Interactivity	Fixed or open
Evolution	Rapid to iterative

Placing a prototype on these four axes is a more useful question than "is it good enough yet?" — it says what the prototype is *for*.

Paper prototyping

> An offline, interactive, rapid method of exploring aspects such as layout, information architecture, interactive design and more.

Low cost, easy to do, interactive, and fun. It's the default first move, and it deliberately scores low on representation and precision while scoring high on interactivity and speed.

Wireframes and Wizard of Oz

- **Digital prototyping using wireframes** — online, still low fidelity, focused on structure rather than looks.
- **"Wizard of Oz" prototyping** — the user experiences a working system; a human is doing the work behind the curtain. Lets you test a value proposition that you have no idea how to build yet.

The economics of design

The reason to start cheap:

	Low fidelity prototypes	High fidelity prototypes	Code prototypes	MVP
Cost	Super cheap	Quite cheap	More expensive	Most expensive

	Low fidelity prototypes	High fidelity prototypes	Code prototypes	MVP
Focus	Flow & interactivity	Look & feel, usability	Code feasibility and usability	Getting to market

Each step right costs more and answers a narrower question. Spending code-prototype money to learn something a paper prototype would have told you is the mistake this table exists to prevent.

Have code generation tools killed wireframing?

A genuinely open question the seminar puts both sides of:

Yes?	No?
Quicker to generate interfaces than to draw boxes	Wireframes help focus on user experience: "what are they trying to build?"
The gap to code prototypes is smaller	LLM designs are more mediocre
Real interactions instead of simulated ones	Wireframes are *intentionally* low fidelity — to get feedback on the user **experience**, not the user **interface**

The strongest argument in the right column is the last one: low fidelity isn't a limitation to be overcome, it's what stops your interviewee commenting on the colour scheme instead of the flow. See prompt-engineering[11] for the tooling this question is about.

[11] courses/comp1100/prompt-engineering.pdf