COMP1100 — Week 5 Notes

Table of contents

Prompt Engineering **1**
- The AI policy . . . 2
- Outline . . . 2
- Machine learning and LLMs . . . 2
- Prompt engineering . . . 2
- LLMs as tools for thought . . . 3
- Homework . . . 3

The History of Computing **4**
- Outline . . . 4
- The history . . . 4
- Why this is in an innovation course . . . 4
- Representation . . . 5
- Next . . . 5

Reference material **5**
- Customer Discovery . . . 5
- Experimentation . . . 9
- The History of Computing and AI . . . 12
- Large Language Models . . . 17
- Prompt Engineering . . . 18
- Test Cards and Learning Cards . . . 21

Prompt Engineering

Week 5b seminar: the second half of the week 5 session and the **last seminar of the semester**. Everything from week 6 on is studio and project work. See comp1100[1] for course logistics.

[1] courses/comp1100/comp1100.pdf

The AI policy

> Under what circumstances can you use generative AI or other related tools to complete your work in COMP1100?
>
> a) None. It is prohibited.
> b) For coding only — no use for writing hypotheses, test cards, etc. **c) For anything except generated interview or other data.**
> c) For any task.

The exception is the important part, and large-language-models[2] explains why: a model will fluently generate plausible interview data, and fabricated data defeats the entire purpose of customer-discovery[3].

Outline

1. What is machine learning?
2. Large language models — how they work, and where they break
3. Prompt engineering: seven tactics
4. LLMs as tools for thought

Machine learning and LLMs

See large-language-models[4] — curve fitting as the intuition for machine learning, next-token prediction with the "The woman ran up the _____" probability tables, and the apples example where the model subtracts 300 from both the red and the green count and reports −99 green apples without noticing.

Prompt engineering

See prompt-engineering[5] for all seven tactics with worked prompts:

1. Use delimiters
2. Ask for structured output
3. Ask it to check whether conditions are satisfied
4. Specify the steps required
5. Iterate

[2] courses/comp1100/large-language-models.pdf
[3] courses/comp1100/customer-discovery.pdf
[4] courses/comp1100/large-language-models.pdf
[5] courses/comp1100/prompt-engineering.pdf

6. Be as specific as possible
7. Build code incrementally

LLMs as tools for thought

The most transferable idea is to ask the model to *interrogate you* rather than answer you:

> I am doing a project using the Lean Startup methodology. I have a hypothesis about my solution that I want to test. I would like to construct a test card for it. Can you please interact with me to help me construct this? **Do not give me answers, just ask a series of questions** until I have the test card. Provide feedback on my answers.

The resulting test card[6] contains your team's hypotheses, not generated ones — which is what keeps it on the right side of the AI policy above.

Homework

Teams: by the submission next Thursday 4:00pm you should have a clear idea of your customer segments and their problems, have those hypotheses formalised as test cards, have tested them, and recorded results as learning cards. You should be starting to formulate what solutions you could provide — some teams may have tested solutions already.

So in week 5 you should be:

- Running final tests for hypotheses related to customer segments and their problems.
- Analysing interview data for common problems and pain points.
- Starting to brainstorm potential solutions leading to a unique value proposition.
- Preparing your repository to submit iteration 1 — see the submission requirements under 'Assessment' and in the week 5 studio handout.

Individuals: complete your interviews about your problem hypotheses, record them (with consent), have them transcribed, then commit and push the transcripts to your GitHub repository so the team can use them. **Also participate in the data analysis** — that's expected of everyone in the team.

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

The History of Computing

Week 5a seminar. The second half of the same session is 2025-08-25-prompt-engineering[7]. See comp1100[8] for course logistics.

Outline

1. In what year was the first programmable computer built?
2. The eras of computing and AI — the golden age, two winters, the revival
3. Another AI winter?
4. History and representation

The history

See history-of-computing-and-ai[9] for the full reference: the six defensible answers to when the first programmable computer was built (87 BCE through 1951), the era-by-era timeline, the Turing Test, Dartmouth 1956, both AI winters and what went wrong in each, the revival, and six people the field's history remembers.

Why this is in an innovation course

This is more than history. The argument is the pattern:

- **1956:** "Machines will be capable, within twenty years, of doing any work a man can do." — Herbert Simon
- **1958:** the Perceptron is "the embryo of an electronic computer that will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." — *New York Times*
- **1970:** "In from three to eight years we will have a machine with the general intelligence of an average human being." — Marvin Minsky
- **1999:** "Artificial intelligence will reach human levels by around 2029." — Ray Kurzweil
- **2014:** "We will have fully self-driving cars on the road by 2017." — Elon Musk
- **2024:** "We are now confident we know how to build AGI as we have traditionally understood it." — Sam Altman

Twice, that pattern of overpromising ended in a funding collapse. **Amara's Law** names the shape of it:

[7] courses/comp1100/2025-08-25-prompt-engineering.pdf
[8] courses/comp1100/comp1100.pdf
[9] courses/comp1100/history-of-computing-and-ai.pdf

> We tend to overestimate the effect of a technology in the short run and underestimate the effect in the long run.

For a course about testing your hypotheses[10] rather than believing them, this is the field's own cautionary tale — the hype cycle is what happens when nobody makes a falsifiable claim.

> "By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it." — Eliezer Yudkowsky

Representation

Six people the field's history remembers: Ada Lovelace, Alan Turing, Margaret Hamilton (who coined the term *software engineering*), Grace Hopper, Fei-Fei Li and Timnit Gebru. See history-of-computing-and-ai[11].

Next

2025-08-25-prompt-engineering[12] — the second half of the same session.

Reference material

Customer Discovery

> **Customer discovery** is the process of turning an initial idea or vision about a market or customers into a set of *facts*. The process tests whether the business model for a company solves real customer problems, and whether the product features fit those customer problems.

It is the first stage of the customer development process in lean-startup[13], and the stage COMP1100 spends nearly all its time in — all three assessment submissions are iterations of it.

[10] courses/comp1100/experimentation.pdf
[11] courses/comp1100/history-of-computing-and-ai.pdf
[12] courses/comp1100/2025-08-25-prompt-engineering.pdf
[13] courses/comp1100/lean-startup.pdf

The four-step process

```
        1. Define hypotheses
                ↓
        2. Test the problem
                ↓
        3. Test the solution
                ↓
   4. Verify, pivot, or refine  →  (customer validation)
```

The loop repeats. Each pass turns more of the business-model-canvas[14] from hypothesis into fact.

Step 1 — define our hypotheses

The initial hypotheses *are* our vision. Write answers to:

1. What are our customers' top problems?
2. How does our idea help to solve these problems?
3. What would a day in the life of a customer look like before and after our product?

These are **guesses or assumptions at best** — which is exactly why they get tested rather than built on.

The step also includes sketching a vision. This means a rough sketch, not a polished rendering.

Step 2 — test the problem

Gather data to **refute or confirm** our hypotheses about the problem. This is where the distinction that runs through the whole course appears:

	Who they are
User	People who *use* a product
Customer	People who *pay for or approve* a product

They are frequently not the same people, and a hypothesis needs to say which it's about.

[14] courses/comp1100/business-model-canvas.pdf

Step 3 — test the solution

Gather data to refute or confirm hypotheses about our *solution*. The thing being sought is **product/market fit** — the match between the customer's problems and needs on one side, and the product's features and value proposition on the other.

> "Ideas are like babies because everyone thinks theirs is cute, therefore be objective when judging your own ideas."
>
> — Tina Seelig

Step 4 — verify, pivot, or refine

Refine the product, pivot the strategy, or (rarely) rethink the vision. See lean-startup[15].

A worked example: the medical alert system

The running example is an emergency alarm for older people living alone, with a speaker, microphone, emergency button, well-being check button, status light and a wireless transmitter.

Initial hypotheses:

1. Older people want to be safe.
2. Families of older people want to know, on a daily basis, that their older family member is safe.
3. Older people want to be able to initiate a call for help and cancel any call for help.
4. Older people want to maintain their independence/autonomy.
5. Older people want to be cared for.
6. Older people do not want to have to press the green button each day to let the care company know that they are alive.
7. Older people do not want to wear a visible (and ugly!) emergency alarm around their neck or wrist.

Hypothesis 1 is about both users and customers, while hypothesis 2 is mainly about *customers* — the family, not the user.

After testing the problem, hypothesis 5 was refuted outright and 7 was partly refuted, producing **new hypotheses**:

- 5. Older people want to be cared **about**. *(the original said "cared for" — a small wording change carrying a completely different product)*
- 8. Some older people wouldn't mind wearing a wireless transmitter if it was not visible to others.

[15] courses/comp1100/lean-startup.pdf

- 9. Older people wanted to be in touch with **their family members**, rather than a service provider.
- 10. Older people would prefer a medical alert that allowed them to be mobile — the existing alarms were attached to a phone line and the transmitters had short range.

This is the whole method in miniature: the product that survives contact with real users is not the one that was sketched.

Early adopters

Not all of the market matters at the start. The adoption curve:

Segment	Share
Innovators	2.5%
Early adopters	13.5%
Early majority	34%
Late majority	34%
Laggards	16%

The first release targets early adopters, not the majority.

Customer development heuristics

1. **There are no facts inside your building, so get outside.**
2. Develop for the few, not the many.
3. Early adopters make your product.
4. And they know more than you.
5. Focus groups are for big companies, not startups.
6. Surveys don't give insight compared to interviews — follow-up questions are difficult.
7. The goal for release 1 is the **minimum feature set for early adopters**.

Heuristic 6 is the reason the course's assessment requires recorded, transcribed interviews rather than a survey: you cannot ask "why?" to a form.

See value-proposition-canvas[16] for the ground rules on actually conducting those interviews, and test-cards-and-learning-cards[17] for the format the hypotheses and results are written in.

[16] courses/comp1100/value-proposition-canvas.pdf
[17] courses/comp1100/test-cards-and-learning-cards.pdf

Experimentation

Why experiment?

To gain insights about customers, their problems, and our solutions — before spending the effort to build.

Two types, two reasons

Two types of experiment:

1. **Generative / exploratory** — explore a space.
2. **Evaluative** — find data to support or refute a hypothesis.

Two reasons to experiment:

1. Evaluate our understanding of our customers and market fit.
2. Evaluate products.

The scientific method

```
1. State hypothesis → 2. Design experiment → 3. Test with data → 4. Gather insights
```

The lean startup version of this loop (Steve Blank) expands into seven steps:

1. Extract hypotheses
2. Prioritise hypotheses
3. Design tests
4. Prioritise tests
5. Run tests
6. Capture learnings
7. Make progress — and repeat

Both prioritisation steps are easy to skip and shouldn't be: a team has more hypotheses than it has time, and testing the wrong one first wastes a week.

What makes a good hypothesis?

The full list: **testable, reliable** (not fake), **concise, replicable, specific, ethical**. Condensed to four:

	Property
#1	**Measurable/testable falsifiable**
#2	**Specific**
#3	**Ethical**
#4	**Simple and clear**

Falsifiability is the load-bearing one. An example of a *non*-falsifiable hypothesis:

> "This solution will help this segment of customers improve their health over the long term."

Nothing about it can come back false — "improve", "health" and "long term" are all unmeasurable as stated, so no result could ever refute it. The exercise is to rewrite it until a specific observation could prove it wrong.

Structuring a hypothesis

The test-cards-and-learning-cards[18] format gives hypotheses a fixed shape:

> We believe that [**target**] are [**showing behaviour / display interest in**] [**for reason Y**]. To verify that, we will [**undertake behaviour**] to [**identify behaviour**]. And measure [**particular variable**]. We are right if [**some measurable statement**]. Therefore, we will ask [**specific questions**].

Types of data collection

Six ways to gather evidence, roughly in increasing order of cost:

Method	What it is
Interviews	Talk with potential customers
Observations	Observe people while doing their "jobs"
Data collected by others	Find existing data that tests your hypothesis
Immersion	Work in your customer's environment
Low fidelity prototyping	Paper prototypes or low-fidelity digital prototypes
High-fidelity solutions	Implement new functionality in an existing product

[18] courses/comp1100/test-cards-and-learning-cards.pdf

See prototyping[19] for the last two.

Experiments should

- Be as **small and comprehensible** as possible.
- Have **clear expected outcomes**.
- Have a clear definition of **success/failure that everyone agrees to** — agreed *before* the results come in, or the team will argue about interpretation afterwards.
- Be as brief as possible when interviewing — **preferably under 30 minutes**.

More tips

Do: listen; observe people's behaviour, tone and reactions, not just what they say; ask "why" and "why not"; ask who else you should talk to; ask if you can follow up later.

Don't: "correct" people (except to clarify a misunderstood question); convince or sell to them.

Learning from experiments

When results come back, interrogate them rather than just recording them:

- If people would not pay for this, what *would* they pay for?
- If these are not our customers, then who is?
- Did we see any **outliers** that could open up new ideas?
- **Pay attention to why our hypotheses are rejected.**

A refuted hypothesis is more informative than a confirmed one — it tells you something you didn't already believe.

Key takeaways

- Experiments = gain insights about customers and products.
- Hypotheses must be **falsifiable**.
- **Test cards** structure our hypotheses; **learning cards** structure our learnings.
- Refuted hypothesis ask *why*.

[19] courses/comp1100/prototyping.pdf

The History of Computing and AI

In what year was the first programmable computer built?

The question is a trick. There are six defensible answers:

Year	Machine
87 BCE	**Antikythera machine**
1642	**Pascaline**
1805	**Jacquard machine**
1837	**Analytical Engine**
1943	**Colossus**
1951	**CSIRAC**

Which one counts depends entirely on what you mean by "programmable" and "computer" — which is the point.

Timeline

Period	Era	What happened
1837		Charles Babbage invents the Analytical Engine. **Ada Lovelace** recognises it is a general purpose calculating machine.
1941–1943		British mathematicians and engineers build the **Bombe** and **Colossus** for cracking codes during World War II.
1950		**Alan Turing** proposes the Turing Test as a goal for artificial intelligence, rather than defining "intelligence".
1956		The **Dartmouth Conference** is held. The proposal coins the term *artificial intelligence*.
1956–1974	**The Golden Era**	Foundational search algorithms and artificial neural networks are invented.

Period	Era	What happened
1974–1980	**The first AI winter**	The hype of the golden era did not eventuate. Funding dried up and excitement dwindled.
1980–1987	**The Knowledge Era**	Knowledge-based systems, such as Prolog and expert systems, invented.
1987–1993	**The second AI winter**	The hype of the knowledge era did not eventuate. Funding dried up and excitement dwindled.
1994–now	**The Revival Era**	Intelligent agents, computational power, the Internet, and machine learning.

The Turing Test (1950)

> "The new form of the problem can be described in terms of a game which we call the 'imitation game' "
>
> — Alan Turing, *Computing Machinery and Intelligence*, Mind, LIX(236):433–460, 1950

Turing sidesteps defining intelligence and substitutes a *test* for it. That move — replacing an unanswerable definition with an operational check — is the same instinct as making a hypothesis falsifiable in [experimentation].

Dartmouth (1956)

The founders present: John McCarthy, Marvin Minsky, Claude Shannon, Ray Solomonoff, Alan Newell, Herbert Simon, Arthur Samuel, Oliver Selfridge, Nathaniel Rochester, Trenchard Moore.

> "We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College… The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."

Two months, ten men. The proposal's confidence is the first data point in the pattern that follows.

Its lasting output was the **divide and conquer model of AI**, splitting the problem into perception, knowledge representation, learning, reasoning and planning.

The Golden Age (1956–1974)

- **Reasoning as search** — Shakey the robot and A*, Hart, Nilsson and Raphael (1968); STRIPS (Stanford Research Institute Problem Solver), Fikes and Nilsson (1971), with its preconditions/postconditions action model.
- **Perceptrons and neural networks** — the single-layer perceptron, Rosenblatt (1958).

The first AI winter (1974–1980)

What went wrong — outcomes failed to live up to the hype:

- **Scalability**
- **Commonsense knowledge**
- **Perceptron limitations**
- **Moravec's paradox**

The result: funding dried up, global interest died down, and criticism came from philosophers and cognitive scientists.

The Knowledge Era (1980–1987)

- **Knowledge-based systems** — Prolog (Colmerauer and Roussel, 1972), and formal ontologies.
- **Expert systems** — MYCIN, for diagnosis of blood diseases (Shortliffe, Buchanan and Cohen, 1970s).

The second AI winter (1987–1993)

Same headline, different specifics — outcomes again failed to live up to the hype:

- **Scalability**
- **Maintenance**
- **The qualification problem**
- **Moravec's paradox** *(again)*

The result: funding dried up (DARPA declared AI was "not the next wave"), global interest died down, AI companies went bankrupt.

The revival (1994–present)

- **Intelligent agents and decision theory** — Bayesian networks, Pearl (1988)
- **Computational power** — Moore's law
- **The Internet and big data**
- **Machine learning** — backpropagation in deep neural networks, Rumelhart, Hinton and Williams (1986)

Another AI winter?

This is the real argument of the history. We have been here before:

> "In from three to eight years we will have a machine with the general intelligence of an average human being." — **Marvin Minsky (1970)**

> "Machines will be capable, within twenty years, of doing any work a man can do." — **Herbert Simon (1956)**

> "…the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." — **New York Times on the Perceptron (1958)**

> "Artificial intelligence will reach human levels by around 2029." — **Ray Kurzweil (1999)**

> "We will have fully self-driving cars on the road by 2017" — **Elon Musk (2014)**

> "We are now confident we know how to build AGI as we have traditionally understood it" — **Sam Altman (Jan 2024)**

Amara's Law

> We tend to **overestimate** the effect of a technology in the **short run** and **underestimate** the effect in the **long run**.

Which is why the pattern above isn't simply "the predictions were wrong" — the winters came from the short-run overestimate, and the revival from the long-run underestimate.

What are some of the risks?

> "By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it." — Eliezer Yudkowsky

> "People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world." — Pedro Domingos

> "The beauty of AI and what we can do with it is currently being overshadowed by reckless hype, monotheistic techniques, and absurd deification. Yet, it's not the first time in 65+ years nor it'll be the last one. We haven't learnt a thing." — Dagmar Monett

Representation in computer science

Six people the field's history remembers:

Person	For
Ada Lovelace	The first "computer programmer"; the first person to consider general purpose computers doing more than numerical calculation
Alan Turing	Turing machines, code cracking in World War II, the Turing test, so many other things
Margaret Hamilton	Software engineer at the MIT Instrumentation Lab; developed on-board flight software for NASA's Apollo program; **coined the term "software engineering"**
Grace Hopper	Identified that computer programs can be written in a high-level language and compiled into machine code
Fei-Fei Li	Machine learning researcher; founder of AI4ALL
Timnit Gebru	Research on bias in machine learning models; founder of Black in AI

> CSSE1001 covers overlapping ground from a different angle — see CSSE1001's history of computing[20] for the hardware timeline and the machine-code-to-Python language progression.

[20] courses/csse1001/history-of-computing.pdf

Large Language Models

Machine learning, briefly

The seminar builds up from curve fitting. Given data on sleep hours against health, machine learning is the business of finding the function that fits:

- $y = ax + b$ — a straight line
- $y = ax^2 + b$ — a curve
- $y = ax + by + c$ — a plane, once there are two inputs (sleep *and* exercise)

The model is the function; learning is finding its parameters.

How LLMs actually work

An LLM predicts the **next token**, one at a time, from a probability distribution over the whole vocabulary.

Given the prefix *"The woman ran up the ____"*:

Candidate	Probability
"hill"	1.2%
"stairs"	1.9%
"the"	1.9%
"to"	1.2%
"baking"	0.005%
"package"	0.0001%
"myriad"	0.000004%
"gazpacho"	0.00001%

Add one more word to the context — *"The woman ran up the"* — and the distribution shifts sharply: "hill" jumps to 10.8% and "stairs" to 6.9%, while "the" collapses to 0.00001%.

That's the whole mechanism. And it explains why *"The woman ran up the ____"* and *"What did the woman run up? ____"* are, to the model, the same kind of problem: continue the text plausibly.

Where it breaks: the apples

The seminar's worked failure, from ChatGPT:

> **Q:** I have 419 red apples and 201 green apples. I eat 300 of the apples. How many apples do I have left?
>
> **A:** 620 (initial count) − 300 (eaten) = **320 apples left.**

Correct. Then:

> **Q:** How many red apples do I have left?
>
> **A:** 419 − 300 = **119 red apples left.**
>
> **Q:** How many green apples do I have left?
>
> **A:** 201 − 300 = **−99 green apples.** It appears that you've eaten more green apples than you initially had…

The model subtracts the *whole* 300 from each colour independently, producing a negative count — and then narrates the absurdity rather than noticing it. The underlying problem was never specified (which apples were eaten?), and the model doesn't flag the ambiguity; it generates a plausible continuation for each question in isolation.

The lesson for the course: an LLM produces text that *reads* like reasoning. Verify anything that has to be true.

Why this matters for COMP1100

The course's generative AI policy permits AI for anything **except generating interview or other data** (see comp1100[21]). The apples example is the argument for that rule — a model will happily produce fluent, confident, wrong content, and interview data that was generated rather than gathered defeats the entire point of customer-discovery[22].

See prompt-engineering[23] for getting useful work out of them anyway.

Prompt Engineering

Seven tactics, aimed at programming tasks. See large-language-models[24] for why they work.

[21] courses/comp1100/comp1100.pdf
[22] courses/comp1100/customer-discovery.pdf
[23] courses/comp1100/prompt-engineering.pdf
[24] courses/comp1100/large-language-models.pdf

Tactic 1 — use delimiters

Mark where the input starts and stops, and mark the shape of the output:

```
Here is some code in Python: $$$ … $$$

… format the output as follows: """On [DATE], [NAME] … """
```

Tactic 2 — ask for structured output

```
… Put these into an HTML table. The table should have three columns.

… format it as a JSON object: """On [DATE], [NAME] … """
```

Example:

> Write a python program that can generate 5 sample emails, including email header. Each email should contain a body that has a greeting, then the main part, and then a sign-off. The main part should contain an order for a part, of the format:
>
> `""" I would like to order [NUMBER] [PARTS]. """`

Tactic 3 — ask it to check whether conditions are satisfied

Tell it what to do when the input doesn't match the expected shape, rather than leaving it to improvise:

> Below is an email from someone. Extract the following into a structured format of: `"""On [DATE], [PERSON] ([SENDER EMAIL ADDRESS]) from [COUNTRY] sent an email with a message about [SUMMARY]"""`. If the email contains a phone number, determine the country number, and insert the country name. **Otherwise, write "unknown country" in place of [COUNTRY].**

Tactic 4 — specify the steps required

Tactic 5 — iterate

> The following is an email from a customer outlining their opinion on a project. Create an email that responds to them, thanks them for their review, and addresses any items in the email.

Tactic 6 — be as specific as possible

A before-and-after:

This:

> Create a Python program that downloads emails from the email address "uqtmiller@gmail.com" and gets an image from them.

Versus this:

> Create a Python program that: 1. Downloads any email in the account "uqtmiller@gmail.com" 2. Saves any image attachments (JPG or PNG) in the directory called "images" 3. Deletes the email from the email inbox.

A longer example, showing how far the specificity goes — including the error case and the runtime environment:

> Create a simple html page that: 1. Use javascript to select a random image from the directory 'images/' 2. When the image is selected, display a message that says "a new image has been received" 3. When the user clicks the message, the randomly selected image is displayed. 4. If no image is available, print "There is no image in the directory <DIRECTORY>", where DIRECTORY is the complete path. 5. A message "An email has been sent to your_family@fake.com" should be displayed. 6. Make sure this runs locally.

Tactic 7 — build code incrementally

Don't ask for the whole system at once. Ask for the page, then ask for the specific piece:

> **Prompt 1:** Create a simple html page: 1. Use javascript to select a random image from the directory 'images/' …
>
> **Prompt 2:** Create javascript code that selects the file with the most recent timestamp from a list called 'images'.

LLMs as tools for thought

The most interesting use is not asking for an answer, but asking to be interrogated:

> I am doing a project using the Lean Startup methodology. I have a hypothesis about my solution that I want to test. I would like to construct a test card for it.
>
> Can you please interact with me to help me construct this?
>
> **Do not give me answers, just ask a series of questions until I have the test card.**
>
> Provide feedback on my answers.

This inverts the usual pattern: the model runs the process and you supply the content, so the test card[25] ends up containing your team's actual hypotheses rather than a plausible-sounding generated one. It stays on the right side of the course's AI policy for exactly that reason — see comp1100[26].

Test Cards and Learning Cards

Two Strategyzer card formats that give experiments a fixed shape. A **test card** is written *before* an experiment; a **learning card** is written *after* it. Together they're the unit of work in [experimentation], and the artefacts the COMP1100 submissions are built from.

Test card

Five lines, filled in before running anything:

```
We believe that          _____________________
To verify that, we will  _____________________
And measure              _____________________
We are right if          _____________________
Therefore, we will ask   _____________________
```

The "we are right if" line is what forces falsifiability — you commit to a threshold before you see the data, so you can't move the goalposts afterwards.

[25] courses/comp1100/test-cards-and-learning-cards.pdf
[26] courses/comp1100/comp1100.pdf

Worked example — users

> **We believe that** our end users are people over 65 who live on their own who would use technology to keep in touch with their family more often because they do not see them much.
>
> **To verify that, we will** interview 50 people over 65 who live on their own.
>
> **And measure** how many would like more regular digital interaction with their family.
>
> **We are right if** 70% of older people say that they would like more regular digital interaction with their family.
>
> **Therefore, we will ask** "how often would you like to interact with your family members?"

Worked example — paying customer

Same product, different party — note that the *user* and the *customer* need separate cards:

> **We believe that** our paying customers are close family members of our end users who would use technology to keep in touch with older family members using a photo-sharing app rather than delegate to a service, because they would like assurance that their family members are doing well in a more personal manner.
>
> **To verify that, we will** interview 50 people who have family members over 65 that live on their own and have an existing emergency alarm system.
>
> **And measure** how many would prefer regular (every day) digital interaction with their family member rather than delegating to a service.
>
> **We are right if** 70% of family members say that they would like more regular digital interaction with their family member rather than delegating to a service.
>
> **Therefore, we will ask** "how often would you like to interact with your older family member?"

Learning card

Four lines, filled in after:

```
We believed that         ______________________
We observed              ______________________
From that we learnt that ______________________
Therefore, we will       ______________________
```

Worked example — hypothesis accepted

> **We believed that** our end users are people over 65 who live on their own who would like to interact digitally with their family more often because they do not see them much.
>
> **We observed** that 44 out of 50 people we interviewed confirmed that they would like daily digital interaction with their family.
>
> **From this we learnt** that our end users would consider using a product that supported this.
>
> **Therefore, we will** accept this hypothesis.

44/50 is 88%, over the 70% threshold the test card committed to — so the decision is already made by the card, not by argument.

Worked example — hypothesis rejected

This is the more instructive one:

> **We believed that** close family members would like to interact digitally with older family members using a photo-sharing app.
>
> **We observed** that 12 out of 50 people we interviewed confirmed that they would use such an app.
>
> **From this we learnt** that family members would not use a new product. **However**, we learnt that some family members would send photos from their smartphone to such an app.
>
> **Therefore, we will** reject this hypothesis, and test a new hypothesis about how family members would take and send photos for their older family member.

Two things to copy from it. First, the rejection produces a *replacement hypothesis*, not a dead end. Second, the "however" — the most valuable finding was a side observation that the headline number would have buried. This is what [experimentation] means by paying attention to outliers.

The templates are in the course's business model and testing templates.