MONASH University

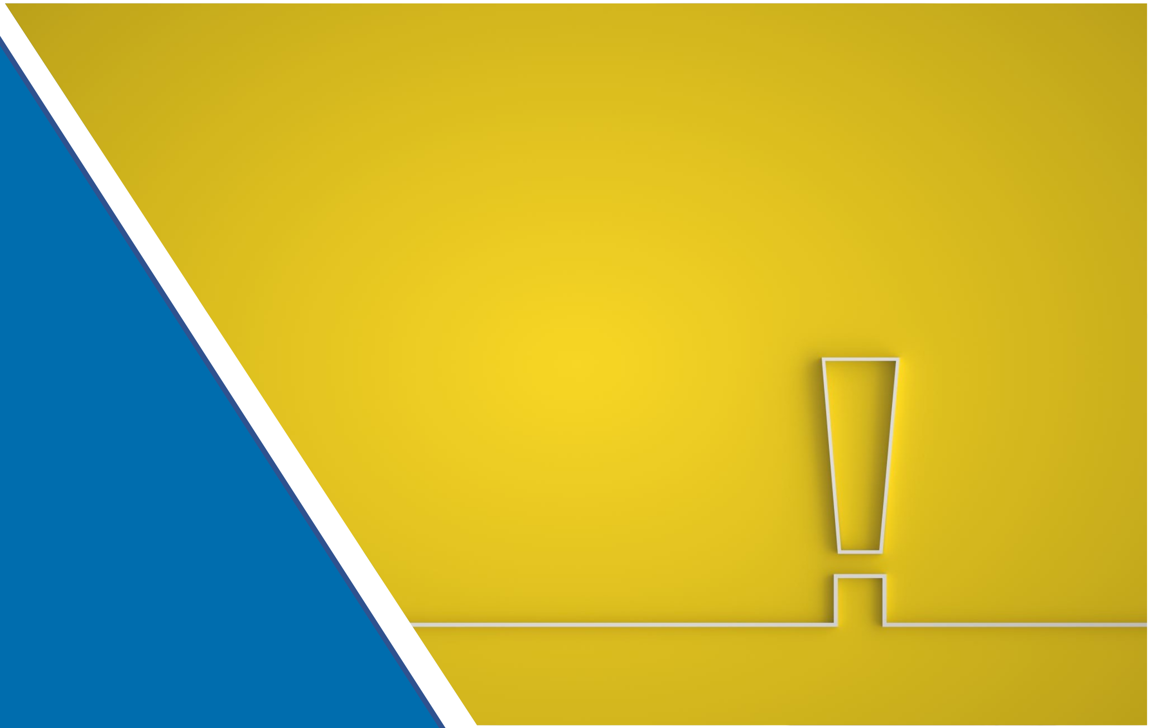
FIT1056: Collaborative engineering for web applications

Introduction to Software Engineering: Implementation Issues / Software Prototyping

Dr Chetan Arora

Faculty of Information Technology

Errors & Exception Handling



MONASH University

Errors in Python

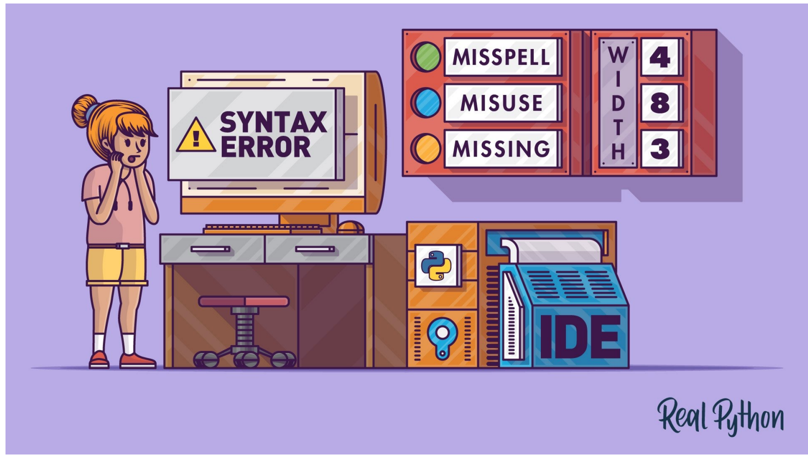
Syntax Errors: When Python attempts to parse your code and encounters these types of errors, it will terminate the program and display an error message without executing any part of it.

Runtime Errors: If a program has no syntax errors, the Python interpreter will execute it. However, the program could still abruptly terminate while running if it comes across a runtime error. These are issues not identified during the parsing stage but become apparent only when a specific line of code is executed.

Logical Errors: Logical errors are often the most challenging to troubleshoot. They don't cause the program to crash; instead, they lead to incorrect outcomes. These issues stem from issues in the program's logic rather than syntax or runtime errors, making them hard to detect no error messages are generated.

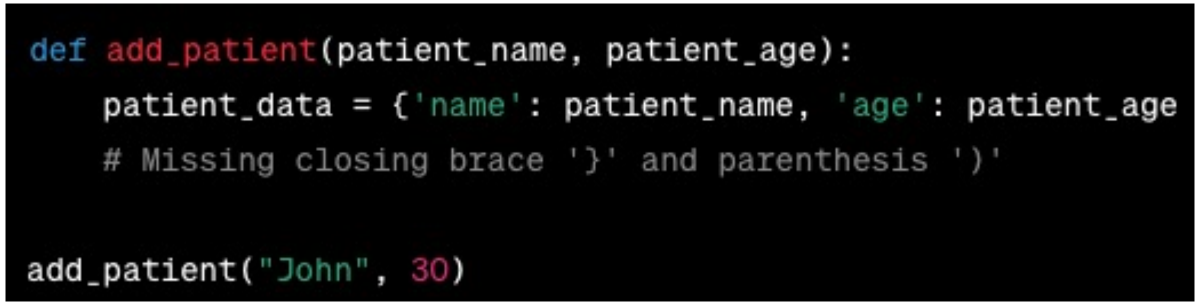
https://www.cs.uct.ac.za/mit_notes/python/Errors_and_Exceptions.html

Syntax Errors



Common Python syntax errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

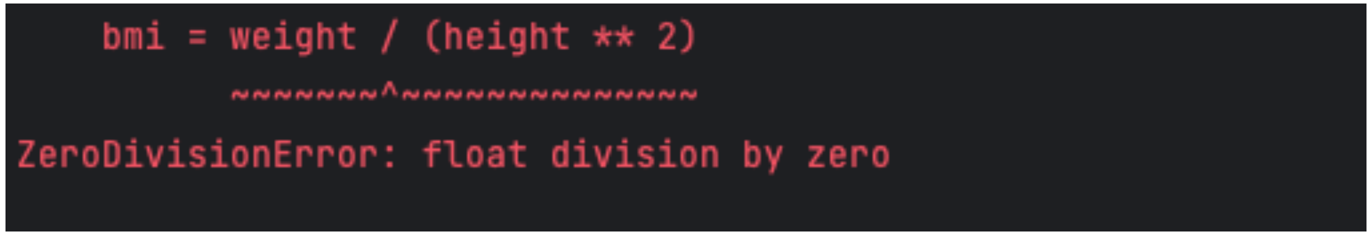


```
def add_patient(patient_name, patient_age):
    patient_data = {'name': patient_name, 'age': patient_age
    # Missing closing brace '}' and parenthesis ')'

add_patient("John", 30)
```

https://realpython.com/invalid-syntax-python/

Runtime Errors



```
    bmi = weight / (height ** 2)
          ~~~~~~~~^~~~~~~~~~~~~~~
ZeroDivisionError: float division by zero
```

Common Python runtime errors include:

- division by zero
- performing an operation on incompatible types
- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn’t exist
- trying to access a file which doesn’t exist

Exception Handling and Assertions in Python

Exception Handling:

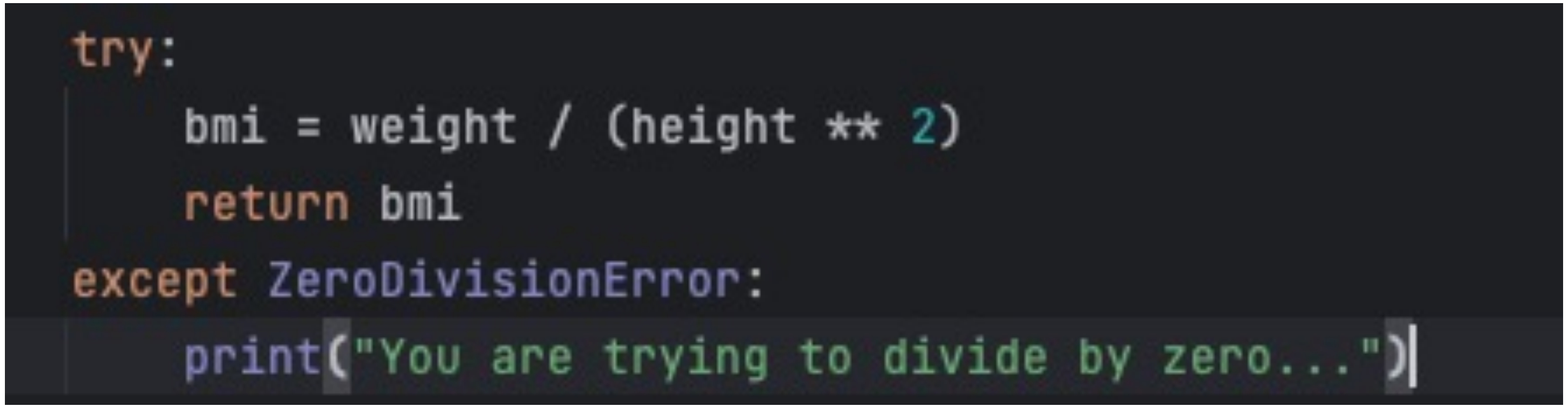
- Handling unexpected errors that occur during program execution.
- Prevents program crashes and provides graceful error recovery.

Assertions:

- Used for debugging and testing assumptions about code.
- Raises an error if the assumption is false.

Exception Handling Basics

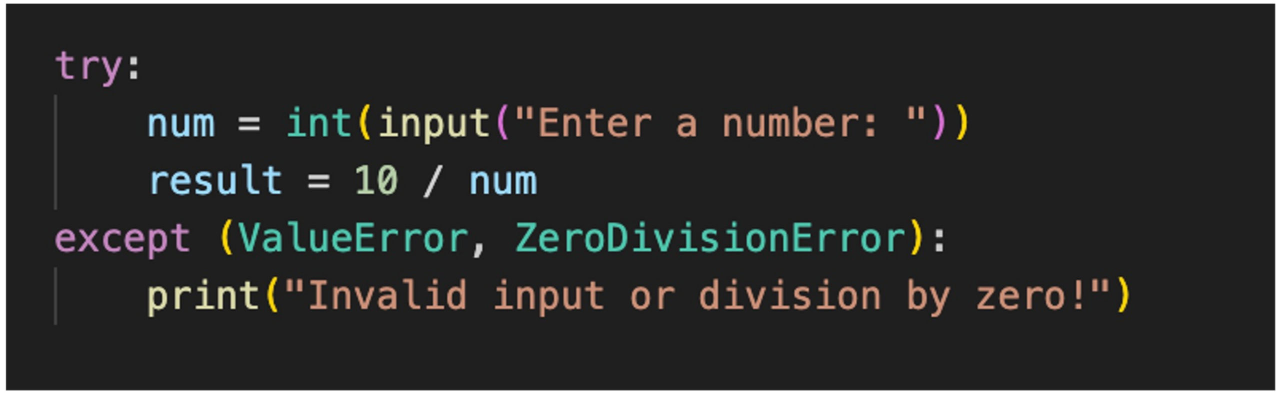
- Events that disrupt the normal flow of program execution.
- Examples: division by zero, file not found, invalid input.



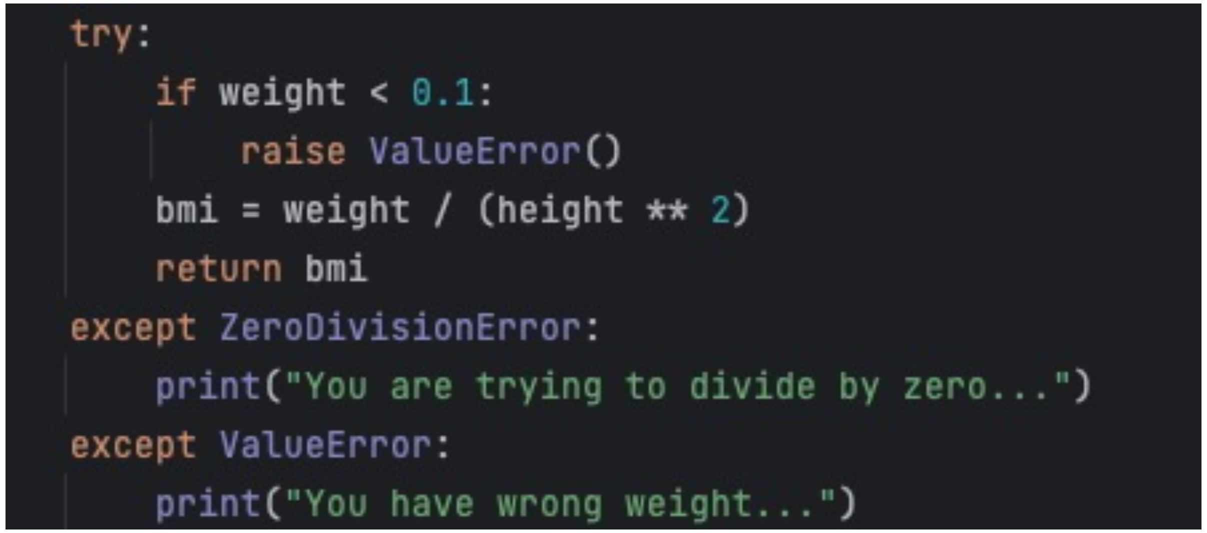
```python
try:
    bmi = weight / (height ** 2)
    return bmi
except ZeroDivisionError:
    print("You are trying to divide by zero...")
```

Handling Multiple Exceptions

- **Single except Block:**
  - Handling multiple exceptions in a single block.
- **The Base Exception Class:**
  - Catching all exceptions using the base Exception class.
- **Order of except Blocks:**
  - The first matching except block is executed.



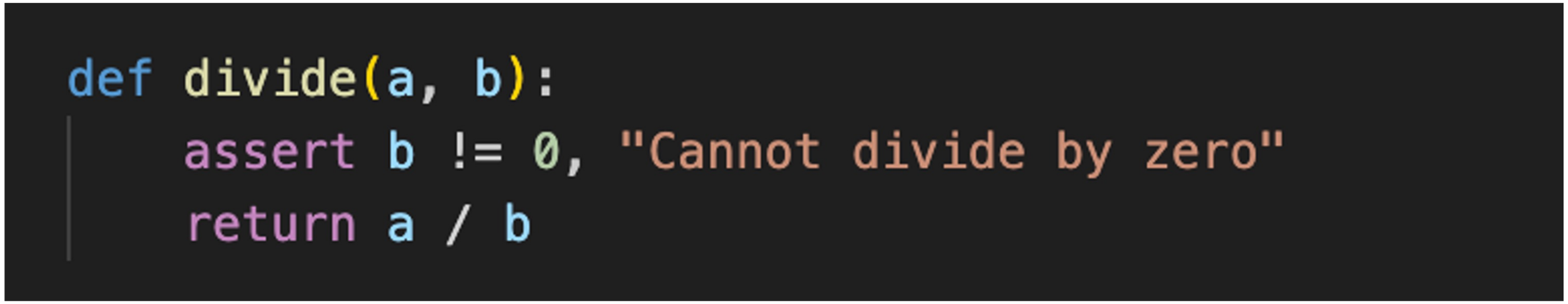
```python
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except (ValueError, ZeroDivisionError):
    print("Invalid input or division by zero!")
```



```python
try:
    if weight < 0.1:
        raise ValueError()
    bmi = weight / (height ** 2)
    return bmi
except ZeroDivisionError:
    print("You are trying to divide by zero...")
except ValueError:
    print("You have wrong weight...")
```

Assertions in Python

- Verify assumptions during development.
- Not intended for handling runtime errors, but for debugging.



```python
def divide(a, b):
    assert b != 0, "Cannot divide by zero"
    return a / b
```

Best Practices for Exceptions and Assertions

- Provide meaningful error messages to aid debugging.
- Use exceptions for exceptional cases, not normal flow control.
- Avoid catching all exceptions using the base class.
- Keep exception handling code separate from main logic.
- Write readable and maintainable code.

Importance:

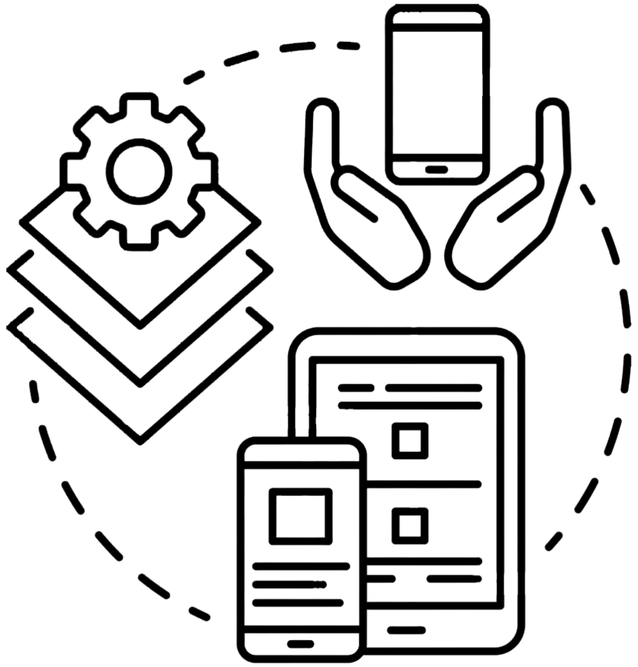
- Writing robust and reliable code.
- Enhancing code quality and maintainability.

Prototyping

Software Prototyping: An Overview

- **Software prototyping** is a development approach that involves creating a **preliminary version** of the software to provide stakeholders and users with a tangible representation of **the final product early in the process**.

- By **visualising the prototype**, stakeholders can better understand the project's direction and offer valuable feedback **before extensive development** takes place.



Prototyping

MONASH University

Prototyping Process

1. **Requirements Gathering**: Collaborate with stakeholders to outline project goals and collect initial requirements.
2. **Designing the Prototype**: Translate requirements into a prototype design, focusing on user interface and functionality.
3. **Building the Prototype**: Develop the working prototype using rapid development tools and techniques.
4. **User Feedback and Iteration**: Share the prototype with users for feedback. Iterate to improve features and design.
5. **Prototype Evaluation**: Assess the prototype's effectiveness in meeting project objectives and user needs.

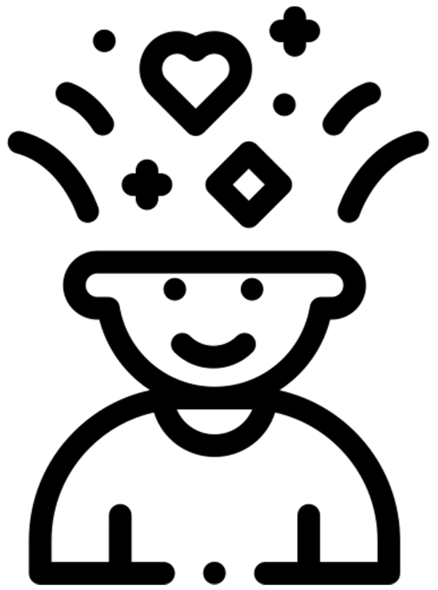
Dealing with Changes and Scope

- Establish a change management process to assess the impact of proposed changes on the project timeline and budget.

- Prioritize changes based on their significance and feasibility.

Potential Challenges in Prototyping

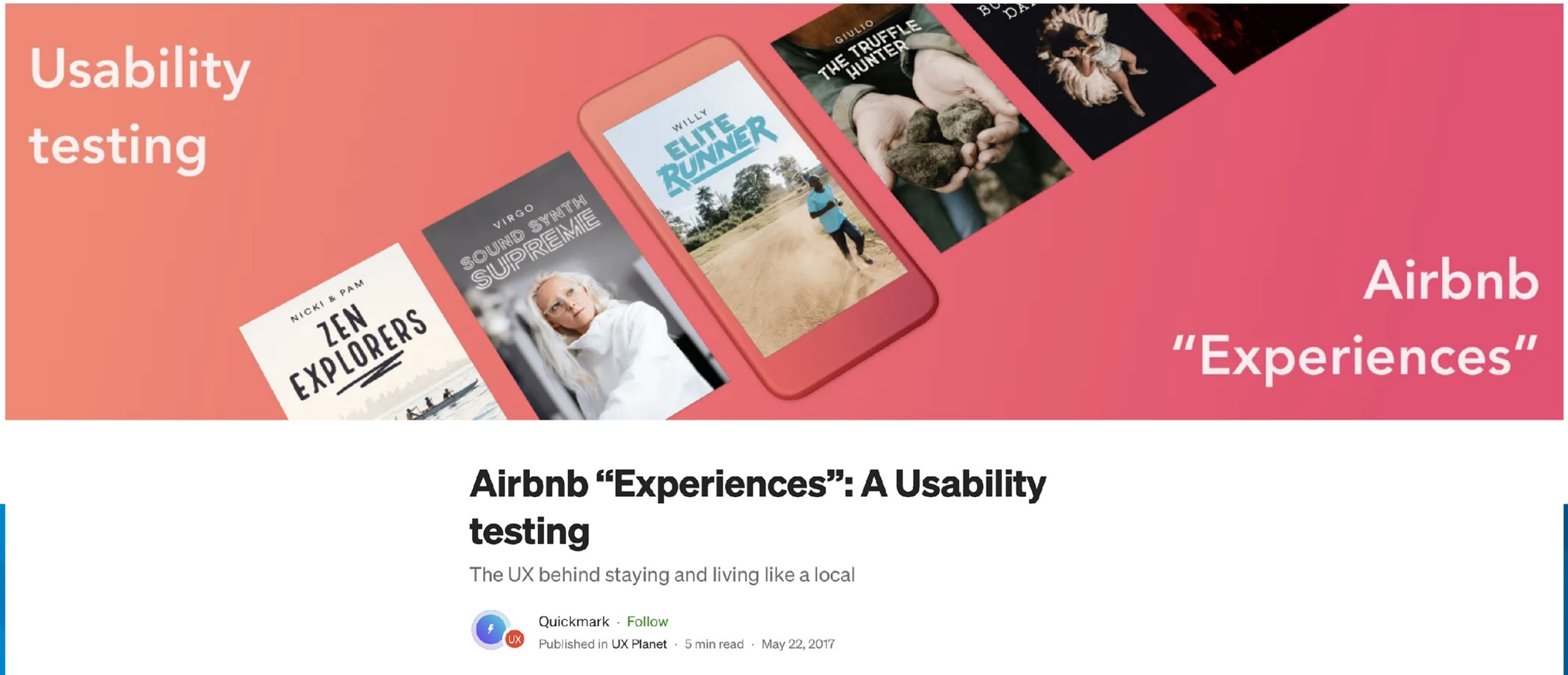
Scope Misunderstanding: Stakeholders might focus solely or the prototype's features, missing broader project objectives.



Managing User Expectations: Users might mistake the prototype for the final product, expecting all features to be implemented.

Success Story: Airbnb's Prototype Testing – Case Study 1

- Airbnb aimed to enhance the user experience and increase bookings on its platform.
- Through prototyping, they redesigned the property listing page to improve navigation and booking options.
- Iterative feedback led to a new design that significantly increased user engagement and bookings, driving revenue growth.



Airbnb "Experiences": A Usability testing

The UX behind staying and living like a local

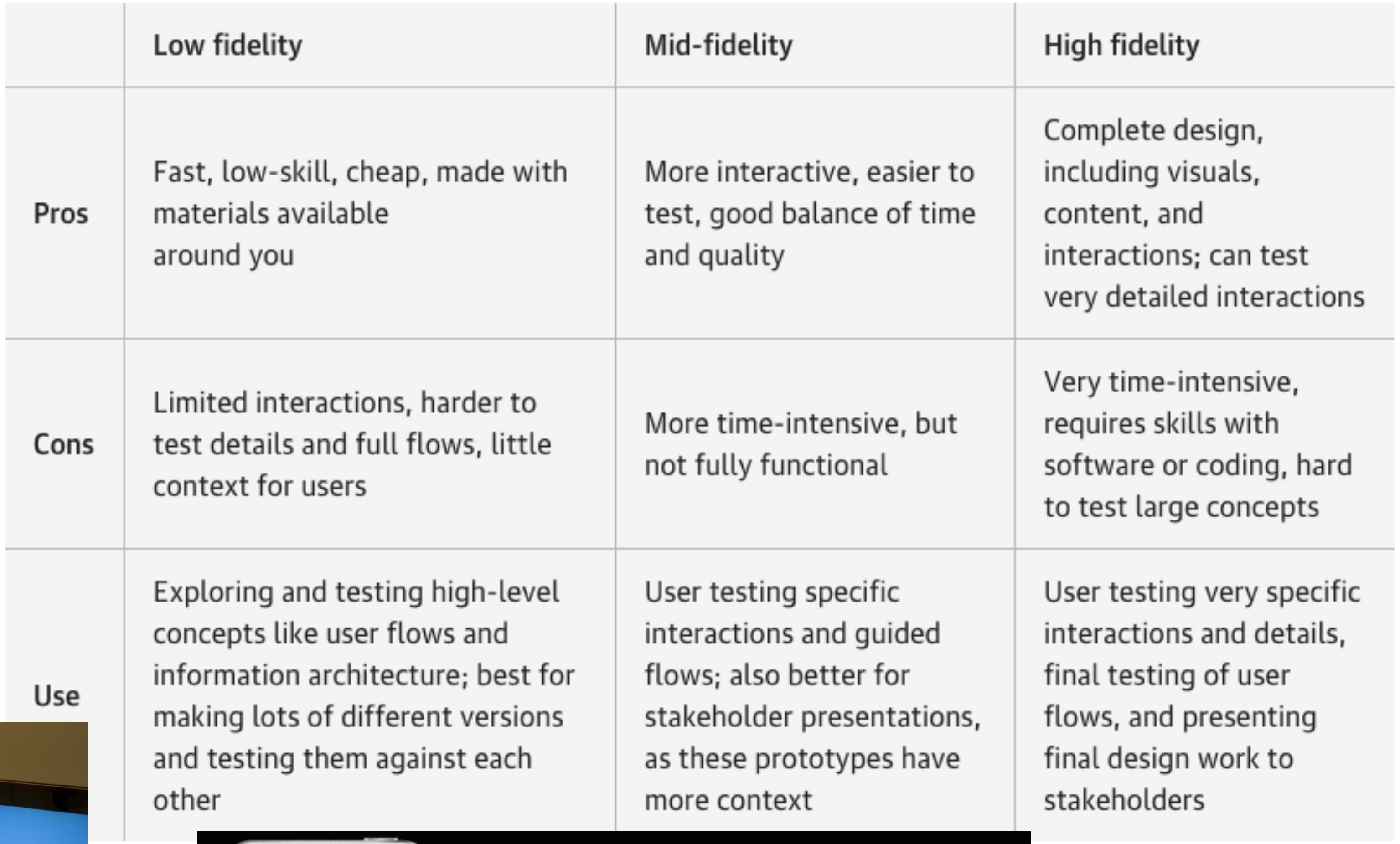
Quickmark · Follow

Published in UX Planet · 5 min read · May 22, 2017

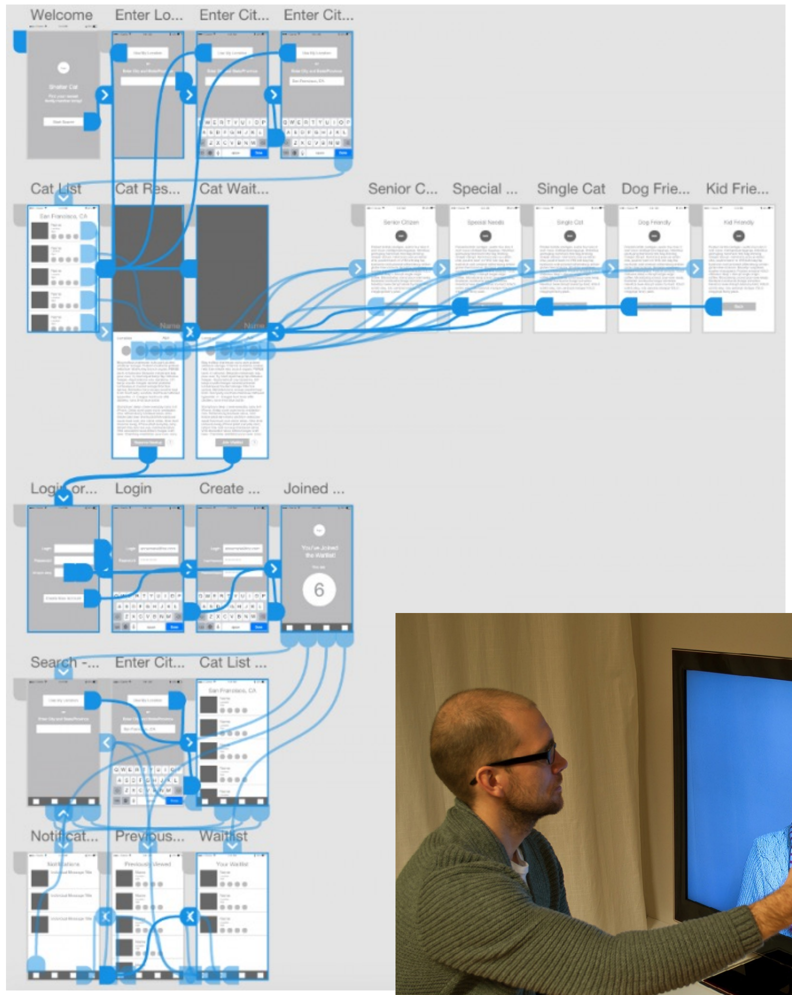
Implementation at Spotify: Evolving the Mobile App - Case Study 2

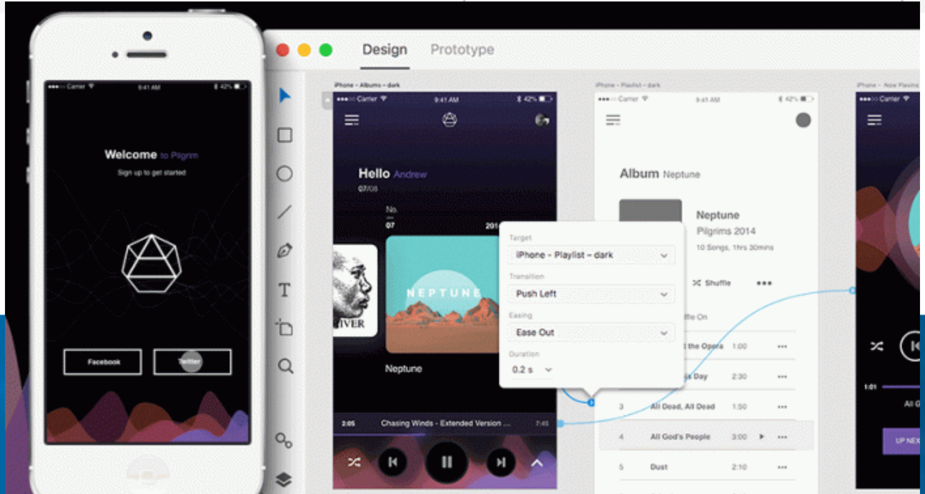
- Spotify used prototyping to refine its mobile app interface for improved user satisfaction.
- By testing prototypes with users, they identified the most intuitive design elements.
- Incorporating user preferences, Spotify launched an updated app, providing a seamless and personalized music streaming experience.

Prototype Types



	Low fidelity	Mid-fidelity	High fidelity
Pros	Fast, low-skill, cheap, made with materials available around you	More interactive, easier to test, good balance of time and quality	Complete design, including visuals, content, and interactions; can test very detailed interactions
Cons	Limited interactions, harder to test details and full flows, little context for users	More time-intensive, but not fully functional	Very time-intensive, requires skills with software or coding, hard to test large concepts
Use	Exploring and testing high-level concepts like user flows and information architecture; best for making lots of different versions and testing them against each other	User testing specific interactions and guided flows; also better for stakeholder presentations, as these prototypes have more context	User testing very specific interactions and details, final testing of user flows, and presenting final design work to stakeholders





When to Use Prototyping?

- **Unclear Requirements**: When project requirements are still evolving and need further clarification.

- **User-Centric Projects**: Ideal for projects where user feedback plays a critical role in defining the final product.

- **Innovative Concepts**: Useful for exploring innovative ideas that require validation through user testing.

- **Rapid Iteration**: When the development team needs to rapidly iterate on design and functionality.