Chapter 2.3: Robust Programming

Introduction

Robust programming ensures that software works reliably, even in unexpected situations. It involves techniques like input validation, error handling, and secure authentication. This chapter explains defensive design, validation, verification, authentication, maintainability, and testing to build strong, reliable programs.

Video Link: 2.3 Defensive design considerations 1

2.3.1 Defensive Design

What Is Defensive Design?

Defensive design anticipates errors or problems before they happen. It ensures the program remains functional and secure under unexpected conditions.

Video Link: 2.3 Defensive design considerations 2

Defensive Design in Action

Input Validation: Checks if user inputs are correct before using them.
Example:
python
Copy code

```
age = input("Enter your age: ")

if age.isdigit() and 0 <= int(age) <= 120:

    print("Valid age.")

else:

    print("Invalid age.")
```

-

Error Handling: Prevents crashes when errors occur.
Example:
python
Copy code

```
try:

    number = int(input("Enter a number: "))
```

```
    print(f"You entered {number}.")
except ValueError:
    print("Please enter a valid number.")
```

-

Security Features: Filters dangerous inputs to prevent attacks like SQL injection.
Example:
python
Copy code

```
user_input = input("Enter your username: ")
sanitized_input = user_input.replace("'", "")
print(f"Welcome, {sanitized_input}")
```

-

2.3.2 Validation

What Is Validation?

Validation checks if input meets specific criteria before the program uses it.

Video Link: 2.3 Suitable test data

Validation Techniques

- **Range Check:** Ensures values fall within a valid range.
- **Type Check:** Confirms data types (e.g., numbers, not letters).
- **Length Check:** Verifies input length is appropriate.
- **Presence Check:** Ensures required fields are not empty.

Example:

python

Copy code

```
password = input("Enter your password: ")
if password.isspace() or password == "":
    print("Password cannot be empty or all spaces.")
```

2.3.3 Verification

What Is Verification?

Verification ensures data entered matches the original source.

Video Link: 2.3 Making algorithms more robust

Verification Techniques

Double Entry: The user enters data twice to confirm accuracy.
Example:
python
Copy code

```
email = input("Enter your email: ")

confirm_email = input("Re-enter your email: ")

if email == confirm_email:

    print("Emails match.")

else:

    print("Emails do not match.")
```

-
- **Checksums:** A calculated "digital fingerprint" confirms data integrity.

2.3.4 Authentication

What Is Authentication?

Authentication verifies that users are authorized to access the system.

Video Link: 2.3 Defensive design considerations 1

Techniques

Username and Password: Ensures secure access.
Example:
python
Copy code

```
stored_password = "secure123"
```

```
entered_password = input("Enter your password: ")

if entered_password == stored_password:

    print("Access granted.")

else:

    print("Access denied.")
```

-
- **Two-Factor Authentication (2FA):** Combines passwords with phone codes.
- **Biometric Authentication:** Uses fingerprints or facial recognition.

Security Practices

Password Hashing: Encodes passwords securely.
Example:
python
Copy code

```
import hashlib

password = "secure123"

hashed_password = hashlib.sha256(password.encode()).hexdigest()

print(hashed_password)
```

-

Limit Login Attempts: Prevents repeated login failures.
Example:
python
Copy code

```
attempts = 0

while attempts < 3:

    password = input("Enter your password: ")

    if password == "secure123":

        print("Access granted.")

        break

    else:

        attempts += 1
```

```
        print("Incorrect password.")
if attempts == 3:
    print("Account locked.")
```

-

2.3.5 Maintainability

What Is Maintainability?

Maintainable code is easy to understand, update, and debug.

Video Link: 2.3 Maintainability

Techniques

Comments: Explain what the code does.
Example:
python
Copy code

```
# Calculate the area of a rectangle
area = width * height
```

-
- **Naming Conventions:** Use clear and meaningful variable names.
- **Indentation:** Makes code readable and organized.

Modular Design: Break the program into smaller, reusable functions.
Example:
python
Copy code

```
def calculate_area(width, height):
    return width * height
```

-

2.3.6 Testing

What Is Testing?

Testing ensures the program works as intended under different conditions.

Video Link: 2.3 The purpose & types of testing

Types of Test Data

- **Normal Data:** Expected inputs.
- **Boundary Data:** Values on the edge of valid ranges.
- **Invalid Data:** Incorrect inputs the program should reject.
- **Extreme Data:** Inputs that push the system to its limits.

Example:

python

Copy code

```
age = input("Enter your age: ")

if age.isdigit() and 0 <= int(age) <= 120:

    print("Valid age.")

else:

    print("Invalid age.")
```

Trace Tables

Trace tables track variable changes during program execution.

Example:

python

Copy code

```
total = 0

for i in range(1, 4):

    total += i

print(total)
```

Trace Table:

Step	i	Total
1	1	1
2	2	3
3	3	6

Summary

Robust programming ensures software reliability through defensive design, validation, verification, and authentication. Writing maintainable code and thorough testing improve program quality and user trust.