What is software?

“What...software is a part of a computer system that consists of data and computer instructions...[including] computer programs...”

— Wikipedia

- A program tells a computer how to perform tasks for its user.
- A program takes some input and produces some output.

Example

Microsoft Word tells a computer how to take input, such as typed characters, and produce output, such as a formatted document.
Computer Science

**Definition**

**Computer science** is the scientific and mathematical study of what computers and computer software can do.

- Computer science attempts to answer questions such as:
  - What problems can we solve using computers?
  - How can we develop software to solve them efficiently?

- Computer science is based on (among others):
  - Math
  - Philosophy
  - Physics
  - Electrical engineering
Software Engineering

engineer, n.: “[A] person who uses specialized knowledge or skills to design, build, and maintain . . . equipment, systems, processes, etc.”

— The Oxford English Dictionary

Definition

Software engineering is the application of engineering principles and methods to software development.

- Software engineering attempts to answers questions such as:
  - How do we develop “good” software?
What makes software “good”? 

Example

Compare Microsoft Word to Google Docs — what makes one better than the other?

- Speed?
- Intuitive design?
- Number of features?
- Price?
- Availability?
- Security or privacy?
- Correctness?
Software Engineering Roles

- **Customers**: Those who request that the software be developed
- **Domain Experts**: Those who have specialized knowledge of the field for which the software is intended
- **Developers**: Those who develop the software
  - **Analysts**: Those who decide *how* to build the software
  - **Programmers**: Those who write the actual code
  - **Testers**: Those who ensure the software is correct
- **Users**: Those who use the completed software
The Software Lifecycle

- Requirements
- Design
- Implementation
- Verification
- Validation
- Operation and Maintenance
## The Software Lifecycle

<table>
<thead>
<tr>
<th></th>
<th><strong>Requirements</strong></th>
<th>Customers and domain experts work with analysts to describe the desired behavior.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Design</strong></td>
<td>Analysts decide how the software will meet its requirements.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Implementation</strong></td>
<td>Programmers write code that implements the design.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Verification</strong></td>
<td>Testers check that the written code conforms to the design.</td>
</tr>
<tr>
<td>5</td>
<td><strong>Validation</strong></td>
<td>Testers check that the completed software meets all requirements.</td>
</tr>
</tbody>
</table>
Requirements

Definition

A software requirement describes one desired property of the completed software.

- The process of determining requirements is requirements elicitation.

- A document collecting all the requirements of a software system is its requirements specification.

- Requirements can be further described as:
  - Functional
  - Non-functional
  - Design constraints
  - Process constraints
Requirements

Example

Consider the following problem:
- Given a DNA sequence, count the codons for glycine.

Analysts might ask:
- How is a DNA sequence formatted?
- What is a codon for glycine?

Domain experts might ask:
- Do we have to look for start and stop codons?
- If not, which reading frame should we use? Should both strands be searched?
Requirements

A good requirements specification is:

- **Complete**: If a feature is not documented as a requirement, you cannot assume it will be implemented.

- **Consistent**: Requirements should not contradict one another.

- **Prioritized**: Some requirements are more important than others; this should be noted where appropriate.

- **Modifiable**: If necessary, altering or adding requirements should be straightforward.

- **Traceable**: Every requirement should be easily identified as addressing a portion of the assigned problem.
A functional requirement describes a desired function of the software system.

- A functional requirement should describe one such function.
- A functional requirement should be unambiguous.
  - It should define any new terminology
  - It should provide any necessary formulae
- A functional requirement must be testable.
- Collectively, functional requirements should describe all potential inputs and their desired outputs.
Recall the following problem:

- Given a DNA sequence, count the codons for glycine.

Functional requirements may include:

- **FR1:** The program shall prompt the user to type a DNA sequence (an uninterrupted string of the characters “A”, “C”, “G”, and “T”).

- **FR2:** The program shall count all occurrences of codons for glycine (the triples “GGA”, “GGC”, “GGG”, and “GGT”) within the given sequence.
Functional Requirements

Example

Recall the following problem:

- Given a DNA sequence, count the codons for glycine.

*(functional requirements, continued)*

- **FR3:** The program shall count all occurrences of codons for glycine within the sequence’s reverse complement\(^1\).

- **FR4:** The program shall display the total number of occurrences of codons for glycine on the screen.

---

\(^{1}\)To be complete, we should also define the “reverse complement”.
Non-functional Requirements

Definition

A *non-functional requirement* describes a desired quality characteristic of the software system.

Example

Non-functional requirements may include:

- **NF1:** The program shall count the codons for glycine in DNA sequences of up to 1,000,000 base pairs.

- **NF2:** The program shall take no longer than 10 seconds to complete.
**Definition**

A **design constraint** restricts the possible designs for the software system.

**Example**

Design constraints may include:

- **DC1**: The program shall run in Python 3 on the macOS 10.12 “Sierra” operating system.
- **DC2**: The developers shall supply the customers with a file, “glycine_counter.py”, containing the program.
Process Constraints

Definition

A **process constraint** restricts the resources with which to build the software system.

Example

Process constraints may include:

- **PC1**: The developers shall supply the customers with the completed program by the end of lab on April 5\(^{th}\).
- **PC2**: The developers shall not contact the customers between the hours of 11:00pm and 7:00am.
Validation

Definition

The process of testing that all requirements have been satisfied by the software system is **validation**.

- Every functional requirement — and as many non-functional requirements as possible — should be tested.
- Such tests are called **acceptance tests**.
- Acceptance tests can be written alongside requirements.
Acceptance Tests

1. Start with the most trivial untested input cases.
2. Describe the expected output for each input.
3. Note which requirements have been tested.
   - Sometimes, multiple requirements can be tested at once.
4. Repeat until all requirements are tested.

- Any discrepancies uncovered by acceptance tests are referred back to the analysts and programmers for correction.
Acceptance Tests

- Consider “negative” cases, where there is no meaningful result.
- Consider “edge” cases, where the input is unusual.
- Consider invalid cases, where the correct result is undefined.

Example

- What if the sequence has no glycine codons?
- What if the only glycine codon is at the very end (or at the very beginning) of the sequence?
- What if the sequence is shorter than a single codon?
- What if every single base is ‘G’?
- What if the user accidentally types in an ‘X’?
Acceptance Tests

Example

- **Test 1**: Tests FR1, FR2, and FR4.
  1. Run the program.
  2. When prompted, type “GGG”, then hit “Enter”.
  3. Expect that the program displays “1”.

Example

- **Test 2**: Tests FR1, FR2, and FR4.
  1. Run the program.
  2. When prompted, type “AAA”, then hit “Enter”.
  3. Expect that the program displays “0”.

Acceptance Tests

**Example**

- **Test 3**: Tests FR1, FR3, and FR4.
  1. Run the program.
  2. When prompted, type “ACC”, then hit “Enter”.
  3. Expect that the program displays “1”.

**Example**

- **Test 4**: Tests FR1, FR2, FR3, and FR4.
  1. Run the program.
  2. When prompted, type “GGGACC”, then hit “Enter”.
  3. Expect that the program displays “3”.

Acceptance Tests

Example

- **Test 5**: Tests FR1, FR3, FR4, and NF1.
  1. Run the program.
  2. When prompted, type 1,000,000 ‘A’s \(^2\), then hit “Enter”.
  3. Expect that the program displays “0”.

Example

- **Test 6**: Tests FR1, FR2, and FR4.
  1. Run the program.
  2. When prompted, type “G”, then hit “Enter”.
  3. Expect that the program displays “0”.

\(^2\)There are ways to automate this.
Operation and Maintenance

- Once a software system has been validated, it can be deployed and put to work by users.

- The software lifecycle does not end here!

- Customers may form requirements for newer versions of the software system.

- Users may find errors that were not caught during validation.
During this quarter, you will take on the roles of:

- Customers
- Domain Experts
- Testers
- Users

Thus, you will participate in:

- Writing and analyzing requirements
  - Initially, conveying requirements to developers
  - Later, also creating the appropriate requirements
- Providing data to test and to operate software
- Writing and running acceptance tests to validate software
- Using software to solve assigned problems
On Software Engineering

“How to program if you cannot.”
— Edsger Dijkstra