What is Software Engineering all about?

Many people can build an AM radio. Are they Electrical Engineers?
Many people can build a log cabin. Are they Civil Engineers?
Many can program. Are they Software Engineers?
Software Engineers can:

- Deal with large and complex systems.
- Precisely specify the behaviour of software.
- Create software with required behaviour.
- Design software that can be adapted.
- Adapt previous designs.
- Do all this when the software is huge and requires many software engineers to cooperate.

Why is SE important to Computer Engineering

Digital systems contain hardware and software components.
Hardware is usually stock.
Functionality is shifting to software.

- Software has lower manufacturing cost.
- Software promises greater flexibility and functionality.
- Examples:
  - Phone switches. Imagine modern features without software.
  - Oscilloscopes.. Analogue —> Digital —> Computer based.
  - Data bases. Filing cabinets —> software
  - Wordprocessing: Typewriters —> software
  - Phones. Analog devices —> Digital—> software

The nature of software

Software must interact with hardware and other software

- Thus it must be precise.

Software errors can be hard to detect and trace.

- Thus it must be correct.

Software is hard or impossible to test completely.

- Thus it should be demonstrably correct.

Errors can be fatal or costly

- Thus it should be demonstrably correct.

Requirements change and errors occur

- Thus it must be flexible

Programs come in multiple versions

- Thus it must be changeable

Software is expensive to design

- Thus designs (and code) should be reusable

The nature of software production

Driven by multiple goals: Cost, time, quality, functionality.
Complex. Software must be designed, as well as written.
Complexity exceeds one mind. —Thus needs planning and documentation.
Software is low level embodiment of mental abstractions. —Thus needs planning and documentation.
Multi-person. — Thus needs planning and documentation.
Expensive — Reuse of code and designs is important.

Some examples

Most commercial jets are now “fly-by-wire”

- Pilot controls computer, computer controls airplane
- Software is suspected in several crashes
- 737 MAX crashes caused by failure to design software to behave correctly in the presence of sensor error.

U.S. Airline lost 65M$

- Reservation system reported flights were as sold-out.

Patients were burned or killed by the Therac-25 radiation machine.

- Software error was to blame. Poor (software) engineering?

Arianne 5 lost with 650M$ satellites

- Software error was to blame. Poor (software) engineering?

Finite element analysis program used to design oil-rig

- Rig sank in sea trials.

Denver Airport opens late. Cost 1G$?

- Software to blame in part

Qualities of good software

External (of concern to end user)

Usefulness

- Is it what the end user wants?

Performance

- Is it fast enough?
- Is it small enough?

Correctness

- Does the product meet its specification?
- E.g. Compiler. Does it report at least on error for any erroneous program?
- Software is either correct or it isn't.
- Definition presupposes an unambiguous specification.

Reliability

- Does the product do what the user wants most of the time?
- Definition is subjective.

Robustness

- Product is resilient to error (internal or external).

Usability

- User interface provides easy access of the functionality.
- Depends on nature of user: Mousy interfaces may be better for novices.

Interoperability

- Use of standards. Standard file formats, clip-board, OLE, etc.

User documentation.

Qualities of good software

Internal

Attributes of products of software engineer: Source code and documentation.

Correctness:

- Is internal documentation self-consistent.
- Is source code consistent with documentation.

Changeability

- Can the software be easily changed?
- Can most likely changes be made most easily?

Understandability

- Is source and documentation understandable

Reusability

- Can components be reused in other projects?

Qualities of good software

Process

Cost

- How much will it cost to produce?
- How much will it cost to maintain?

Timeliness

- When will it be ready?

Controllability

- Can the progress of the production be monitored?

Poor Design and Good Design

Robert Martin's Symptoms of poor design

- Rigidity — The design is hard to change
- Fragility — The design is easy to break
- Immobility — The design is hard to reuse
- Viscosity — It is hard to do the right thing
- Needless Complexity — Overdesign
- Needless Repetition — Mouse abuse
- Needless Repetition – Just kidding
- Opacity — The design is hard to understand.

Conversely: the signs of good design

- Adaptability — The design is easy to change
- Robustness — The design is hard to break
- Reusability — The design can be reused
- Fluidity — It is easy to do the right thing
- Simplicity — The design is the "simplest thing that will work"
- Terseness — No unneeded duplication of code
- Perspicuity — Organized and clear

S.E. Products

Direct Products of S.E. (Deliverables)

- Requirements Document
- Specification
- User documentation
- High-level design documentation
- Source code
- Executable code

Indirect product

- Behaviour — ultimately the above all describe behaviour as much as structure

Software Engineering is **Behaviour** Engineering.
Time is what makes software (and CE/EE) interesting and difficult.
Aside: Civil engineers design buildings and bridges etc. Actual physical objects. They don't design blueprints. Similarly software engineers don't design software, they design behaviour. Behaviour —while not a physical object— is a part of reality. When the behaviour is fully designed and described in source code, we are done, just as a civil engineer is done when a building is described by blue-prints. However there is a difference. The correspondence between a blue-print and a building is straight-forward. The correspondence between source code and behaviour is not. This is why software engineering is very challenging. (Same

comment applies with suitable ammendments to other parts of computer engineering and to electical engineering.) **End of aside.**

What is “designing” and what is “the design”?

In English the word “design” is a verb and a noun.
“Design” refers both to three things

- the process of designing
- the decisions made during that process (“it’s all designed, but still in my head”), and
- the written expression of those decisions.

As engineers, an important part of your job is to produce designs and express them accurately.

So what is the design?

We must design and describe the desired behaviour of the system.

- Thus the “**requirements and specification documents**” and the “**user documentation**” might be considered part of “the design”

We must achieve that desired behaviour

- **High-level design documentation** — describes how that behaviour is to be achieved at a high-level, without going into all the details of the source code.
- **Source code** — describes how the behaviour is to be achieved in full detail.
- **Source comments** — supplements the source code with natural (or sometimes mathematical) descriptions.

Are the source code and the source comments really part of the (expression of) the design? See Reeves' article[1] 'What is software design?'
Reeeves' key arguments:

- By considering source code as (part of) the design, we can properly verify our design. Otherwise the process lacks the feed-back loop characteristic of other engineering disciplines.
- Translating high-level design to source is not a matter of routine. Software is too complex and the devil is often in the 'details'.
- Note that Reeves does not claim that the source code is the *only* expression of the design.

Tools just as Javadoc allow parts of the source code together with parts of the source comments to be abstracted to form a sort of high-level design documentation.

[1] http://www.developerdotstar.com/mag/articles/reeves_design_main.html and also appendix D in Martin's *Agile Software Development*.

Change

All successful software is changed.
Sources of change:

- User needs change.
- Hardware/Software platform changes.
- Additional platforms are supported.
- Errors are corrected.
- Efficiency is improved.
- New features / capabilities are added.
- A similar but different product is required.

Fact: *Most of the cost of software is spent changing it not developing new systems.*

Planning for change

Identify likely varieties of change.
Design for change.
Document that design.

Software decay

When software is changed in ways that undermine its design (see symptoms of poor design), it decays.
Changes lead to bugs – bugs lead to more changes — a downward spiral.

Refactoring

When an unanticipated change is needed, first redesign so that this change and ones like it are natural. Then make the change.
This way the software becomes more adaptable in response to change, rather than more rigid and more viscous.

The Software Development Process

Software Engineering consists of a number of activities.

Requirements analysis

- Figure out and document what is required of the product
- Output: Requirements document (often uses UML use-cases)

Specification

- Decide on the exact external behaviour and properties of the product
- Output: Specification document (often uses UML model of the domain and use-cases)

Test planning

- Output: Test plan

High-level design

- Decide on the internal structure of the product
- Output: Design document and (often uses UML model of the software)

Implementation (aka low-level design)

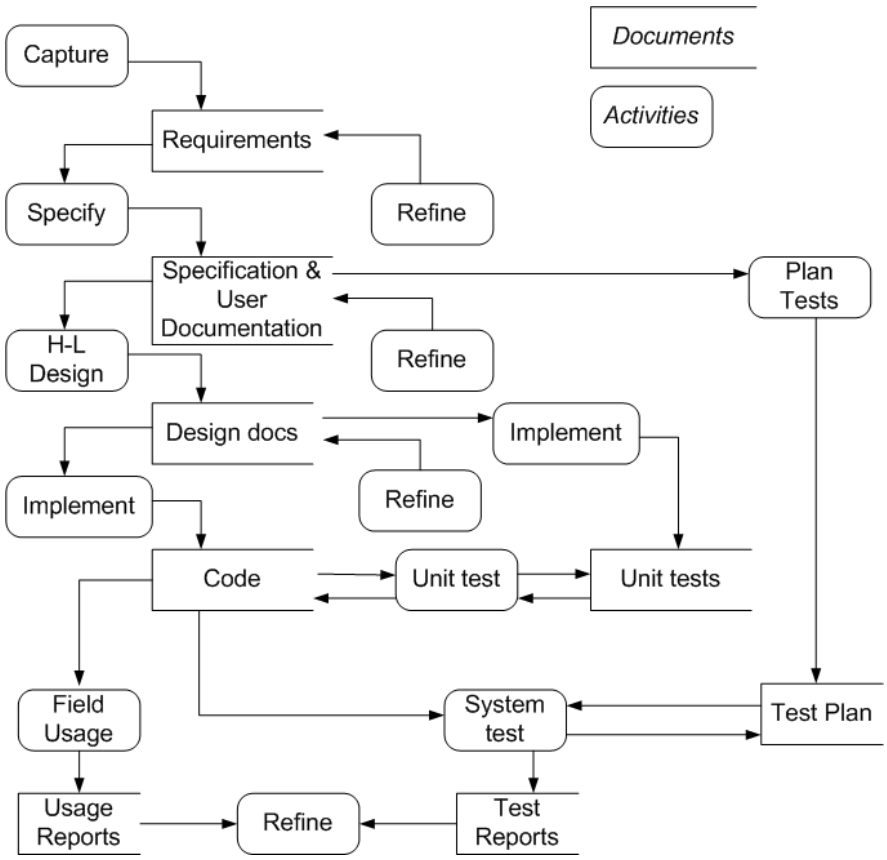
- Coding. Unit testing. Peer review.
- Output: Code

System testing

- Output: Test report

User documentation

- Output: User manual, help pages, etc.



Development Process (cont.)

The activities are typically not done in sequence.
Rather they are interleaved and iterated.
E.g.: specify, design, code, revise design, revise code, review and revise specification, revise design, revise code, test system, revise code, ...
A lot has been said about organizing software processes, but this is not the course for that.

Which part is design?

Narrow definition of design.

- Design is that bit between specification and implementation

Broad definition of design.

- Design is the whole process.

So what is this course about?

We will take the broad view for this course, but the emphasis is on high-level design, including some specification.

- Requirements analysis deserves more attention than we can give it in this course.
- Instead we will work well known problems, so requiremenst analysis shouldn't be crucial.
- You already know how to implement and test — I hope.

So why implement? Is the campus strewn with bridges designed by civil students?

- Because the test of a H-L software design is its implementability.
- In particular the cost of a software design is in its implementation.
- Because the high-level design always changes as we implement.
- You can always learn more about implementation and testing.
- As Reeves says, the source code is (part of) the design.