Software Engineering

CS 1025 Computer Science Fundamentals I

Stephen M. Watt

University of Western Ontario

Software Engineering

- Writing small programs is easy.
- Writing big programs is hard.
- This sounds trivial, but therein lies a deeper truth.

Software Engineering

- Why are bigger programs harder to work with than smaller programs?

- There are at least two reasons:

  1. The problems they are trying to solve are inherently more difficult and more complex.

  2. They have more interacting parts.

Interactions Among Parts

• 1 part	=>	no interactions
• 2 parts	=>	1 pair-wise interaction
• 3 parts	=>	3 2-way interactions,
		1 3-way interaction
• 4 parts	=>	6 2-way interactions,
		4 3-way interactions
		1 4-way interaction.
• 100 parts	=>	4950 2-way interactions
		161,700 3 -way interactions
		3,921,225 4-way interactions
	...	
• 1000 parts	=>	499,500 2-way interactions
		166,167,000 3-way interactions
		41,416,124,750 4-way interactions
		...
• *n* parts	=>	*n* × *(n-1)/2* 2-way interactions
		2^*n* –(*n*+1) interactions all together.

Interaction Among Parts

- This growth in possible number of interactions among parts leads to:

1. New emergent phenomena at each level of program size.

   Different concerns arise in programs with 100 *n* lines that do not exist in programs of with *n* lines. This is true for all *n.*

2. One of the most important considerations in software design becomes limiting the interaction among parts.

Smart Choice of Parts

- One of the most important aspects of software design is in deciding what the parts of a program will be.
- This is *problem decomposition.*
- A good choice of parts will provide
  - *re-usable components*, with
  - *well-defined interfaces* that
  - allow parts to be *composed flexibly* and
  - *stylize the interaction* among parts
- Think Lego bricks, not Swiss army knives.

What are the Parts?

- Depends on the programming environment and the granularity.

- Lines of code.
- Methods or functions.
- Classes or modules.
- Packages.
- Application suites.
- Computer systems.
- Networks.
- Planetary infrastructures.

Software Engineering

- *Software Engineering* is the body of knowledge and practice of managing the life-cycle of software.

- It focuses on how teams of people can design, build, test, deploy and maintain software modules.

Top-Down *vs* Bottom-Up *vs* Middle-Out

- There are various ways to design software, each with its own adherents.

- *Top-down* software design is based on successive refinement of ideas.

- *Bottom-up* software design is based on crafting basic modules and successively abstracting interfaces.

- *Middle-out* software design is based on starting in the middle and doing *both*.

(Until you have a **lot** more experience top-down is the best.)

What Does Wikipedia Say?

- **Top-down** and **bottom-up** are strategies of information processing and knowledge ordering, mostly involving software, and by extension other humanistic and scientific system theories (see systemics).

- In a **top-down** approach an overview of the system is first formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes" that make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model.

- In a **bottom-up** approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, whereby the beginnings are small, but eventually grow in complexity and completeness. However, "organic strategies", may result in a tangle of elements and subsystems, developed in isolation, and subject to local optimization as opposed to meeting a global purpose.