SSW-540: Fundamentals of Software Engineering

What is Software Engineering?

Dr. Richard Ens
School of Systems and Enterprises

Course Overview

Each week…

- Lecture and class activities, intermingled
- Text and other assigned reading
  - Sommerville, Ian, Software Engineering, **10th Edition**, Addison Wesley, 2016, ISBN-10: 0-13-394303-8; ISBN-13: 978-0-13-394303-0.
  - Severance, Charles, Python for Everybody: Exploring Data in Python 3. Download from http://www.pythonlearn.com/book.php
- Small python assignment
- Topical assignment building to final project
- About every other week: a mini-exam/online quiz

Use your course schedule!

Weekly Course Topics

1. Introduction to Software Engineering
2. Software development processes and activities
3. Software requirements engineering
4. Software construction and modeling
5. Software architecture and design
6. Software implementation and UML/SysML
7. Software testing and evolution
8. Software defect tracking and estimation
9. Software project management
10. Software dependability and reliability engineering
11. Software dependability: safety, security and resilience engineering
12. Software reuse and component-based software engineering (CBSE)
13. Final project readouts

Python

Software Engineers do write code, so we will be sure you know how to code.

Why Python?

"Hello World" in C++

```
#include <iostream.h>

void main()

{

    cout << "Hello, world." << endl;

}
```

"Hello World" in Python

```
print "Hello, world."
                    (Python 2.7)

print ("Hello, world.")
                    (Python 3.0)
```

It's easy for beginning students to comprehend and increasingly being used in many engineering domains (chemical, mechanical, biomedical, electronic, etc.)

For a complete comparison of 2.7 and 3.0, see https://docs.python.org/3.3/whatsnew/3.0.html

What is Python

- Interpreted, (or compiled into byte code to run in VM). Operates either
  - in command-line mode (you enter your program statements and the interpreter prints the result)
  - in a "script" mode (you put all your program statements into a file and the interpreter executes the file's contents). Python scripts use `.py` as an extension.
- Like any programming language, Python is formal language designed to express computations.
  - Formal languages have strict rules about syntax, both tokens (basic elements) and structure (how tokens are arranged)
  - Give me a well-structured English sentence with unrecognizable tokens in it.
  - Give me a another sentence with valid tokens but with invalid structure.
- As we learn to use Python, we will learn its tokens and its structures.
  - What Python token have you already learned?

So Why Python?

- It is well documented and free – one of the largest and best-organized open source projects
- It runs everywhere – cell phones to supercomputers – on every OS!
- It has a clean syntax
- It is relevant—used by thousands of companies; it's one of Google's 3 "official" languages
- It is well supported by tools
  - Legacy editors like *Vi* and *Emacs* have Python editing modes
  - Professional quality IDE's are available, some even free to students like Enthought's *Canopy* (www.enthought.com/products/canopy/) for Python 2 and Spyder (https://github.com/spyder-ide) for Python 3

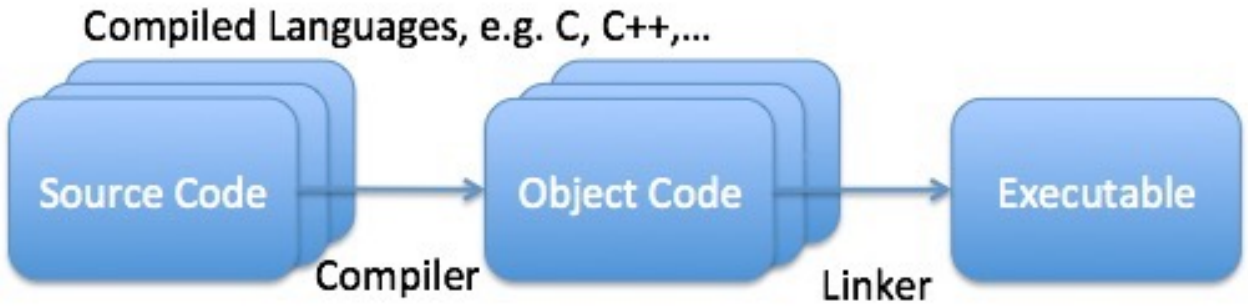
Python work each week

Write and submit a script (a .py program) that…

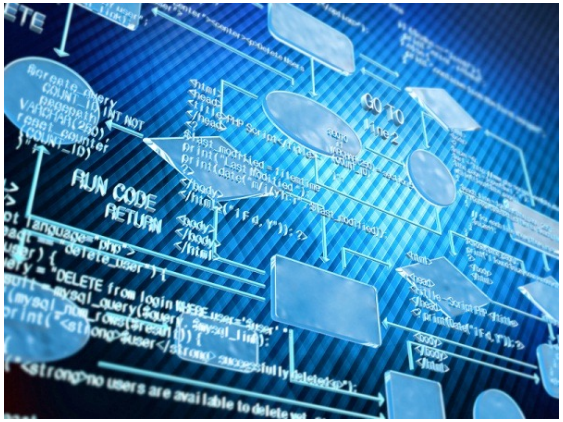
1. Set up Environment – …prints out your name in response to a query.
2. Uses conditionals (if-then-else) and exception handling.
3. Uses user defined functions
4. Guesses a number using loops and a function
5. Manipulates strings
6. Slices and dices input
7. Finds unique strings
8. Counts unique items
9. Examines the contents of an ASCII file
10. Uses regular expressions
11. Accesses the web to find additional web links
12. Uses web API's, in this case, a geocoding API

What is software?
More than just a program…

- Software consists of instructions for manipulating and processing data
  - Varying formats of software:



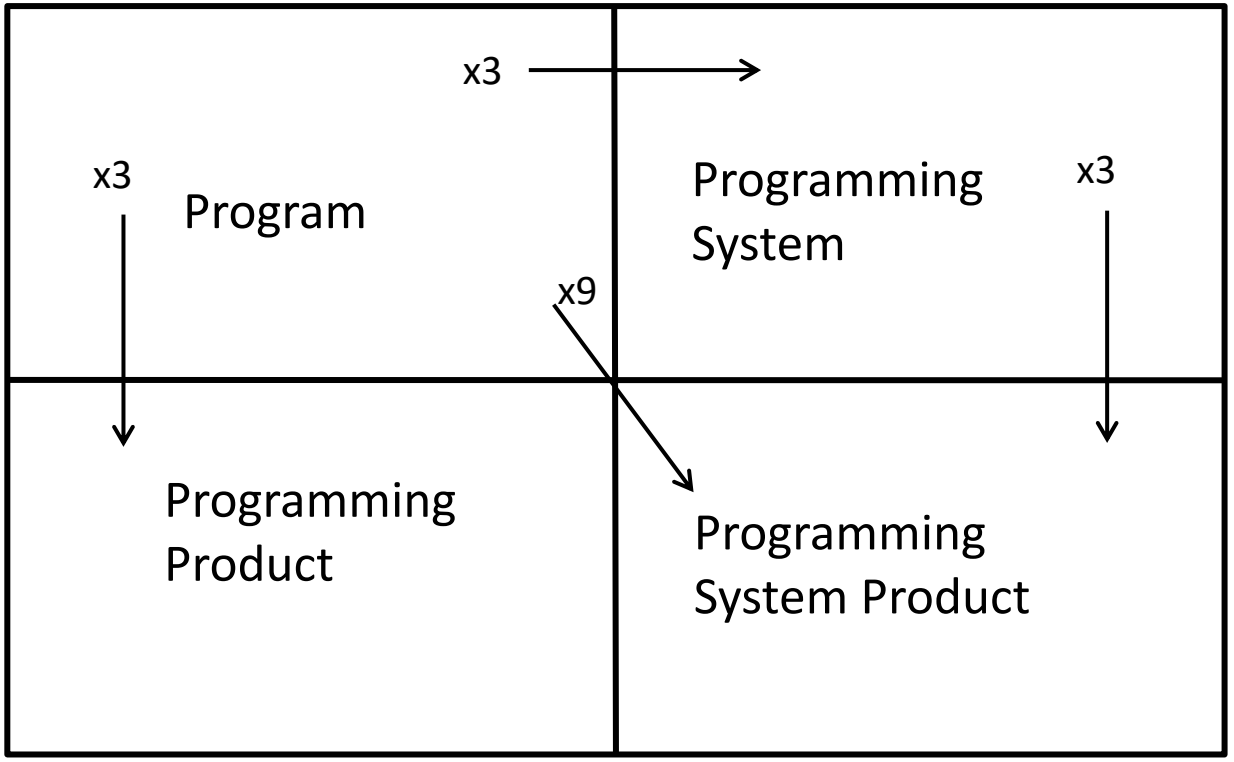
  - Source code to byte code as well
- Software may also be thought to include code documentation
  - Documentation within the source code strongly encouraged



Brooks' *Mythical Man Month*: Software vs. Software Engineering

3 times the effort needed to produce a ***Program*** is needed to produce a ***Programming Product*** or a ***Programming System***.

A ***Programming System Product*** requires 9 times the effort.



Software is not just code!

- Software includes all electronic documentation that is needed by
  - System users
  - Quality assurance staff
  - Developers
- Essential attributes of software products include
  - Acceptability
  - Dependability and security
  - Efficiency
  - Maintainability
- Most software products consist of multiple modules of code and other files linked together to form a complete product.

Organizing software - example

/ (a root)

- src – Source code files (may include modules, header files, os
- ext – (Extensions) Third-party libraries
  - libname – [w/version #]
    - *include* – Headers
    - *lib* – Compiled lib files
    - Download.txt – Contains link to download the version used
- *ide* – storage for project files
- *bin* - (Binaries) Compiled exe files
- *build* – The compiler's build files
- doc – Documentation of any kind
- README
- INSTALL
- COPYING

Modules directory for the Apache Web Server – Part 1

Subdirectory	
aaa	~22 modules dealing with authorization and authentication
arch	3 directories for ~3 OS-specific header files
cache	~18 modules that implement file and data caching capability
database	DBD framework that manages connections to SQL backends efficiently
cluster	2 modules for working with multiple servers
dav	~15 modules that implement WebDAV functionality for fs, lock & main
echo	module for echoing input back to the user
examples	4 sample modules to help you develop your own Apache modules
experimental	interesting functionality still in early stages of development
filters	~20 modules that perform general inline data filtering
generators	~10 modules that perform data generation functions

Modules directory for the Apache Web Server – Part 2

Subdirectory	
http	8 modules that basic HTTP protocol implementation
http2	>100 modules that provide HTTP/2 protocol implementation
loggers	5 modules that handle logging functions
mappers	~10 modules that handle URL mapping and rewriting
metadata	~11 modules that deal with Header metadata
proxy	~20 c and header files for the proxy module for Apache
ssl	~20 modules with code for OpenSSL functionality
test	Should be modules that test various components of Apache, but test modules have moved to http-test/perl-framework/c-modules [You do not compile these into a production server.]

Also directories for code headers, debugging routines, ldap linkages and others.

Compiling and installing Apache HTTP Server on Unix - example

Download	$ lynx http://httpd.apache.org/download.cgi

Extract	$ gzip -d httpd-*NN*.tar.gz
	$ tar xvf httpd-*NN*.tar
	$ cd httpd-*NN*

Configure	$ ./configure --prefix=*PREFIX*
Compile	$ make
Install	$ make install
Customize	$ vi *PREFIX*/conf/httpd.conf
Test	$ *PREFIX*/bin/apachectl -k start

NN must be replaced with the current version number, and *PREFIX* must be replaced with the filesystem path under which the server should be installed. If *PREFIX* is not specified, it defaults to /usr/local/apache2.

Git and the GitHub platform

- Git is a distributed version control system—its keeps track of file changes made over time
- GitHub is a website where you can upload your source and other files, centralizing it for sharing, viewing and making changes to it.
  - Provides the ability to document bugs using *Issues*
  - Using branches and *pull requests*, you can collaborate on file contents
  - You an review work in progress and see team progress over time
- Originally a tool for storing source code
- Now includes tools for specifying, discussing and reviewing software
- Widely use in the Open Source communities
- An excellent collaboration tool

Key Concepts within GitHub

Commit – saving changes in Git

Commit message – explanation of the change required with each commit

Branch – an independent series of commits

Master branch (master) - default

Feature (or topic) branch – new functionality

Release branch – a feature branch that has been released

Merge – combining branches

Tag – a specific historic commit reference

Check out – see the files of a particular point in the project' history

Pull request – a request for review

Issue – documentation of the need for a discussion

Wiki – used for linking to documentation

Clone – copy of the repository

Fork – copying the repository onto your GitHub account to change it

What is Software Engineering?

You tell me!

1. What is software?

2. What is engineering?

3. Is all software engineered?

The software development process

What's included?

- Specification
- Development
- Validation
- Evolution

How much effort goes into each?

History of *Software Engineering*

- 1968 international conference bemoaning
  - Insufficient reliability and insufficient scalability of software
  - Unmet schedules and cost overruns
  - Lack of
    - stand-alone software
    - interchangeable software components
    - mass production techniques in software
    - standardized software education
- In 1968, typically, software was part of a system delivery and **<15%** of all projects were on-time and on-budget
- The (academic) discipline of *Software Engineering* was created to solve these problems!

Software wasn't just "*apps*"

LEO – The first automated office video

- Pre-1970's one bought systems, not software
  - Hardware was king!
- Post-1970's
  - Hardware becomes a commodity; almost a consumable
  - Hardware becomes a Trojan Horse for software
  - Software evolves into the more critical element.

We've come a long way (baby) – sort of…

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database.

Source: Standish Group Chaos Report, 2015

Size and Methods matter:

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011-2015 within the new CHAOS database segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

Software engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
  - Dependability, security and performance are important for all types of systems.
  - Understanding and managing the software requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

Software Systems

- Software engineering is not an isolated activity but is part of a broader systems engineering process.
  - Sometimes overlooked: Software must run on hardware, general purpose or custom-built.
- Software systems are not isolated systems but are essential components of broader systems
  - Technical computer-based systems, e.g., control systems
  - Socio-technical systems that have a human, social or organizational purpose, e.g., business and command systems
- The properties and behavior of system components are inextricably inter-mingled. This leads to complexity.

"Software is eating the world, in all sectors"

- Mark Andreessen, Netscape founder
- "*In the future, every company will become a software company*."

- Elon Musk: The Model S is not a car but a "*sophisticated computer on wheels*".
- Tesla Autopilot 100% software feature delivered wirelessly

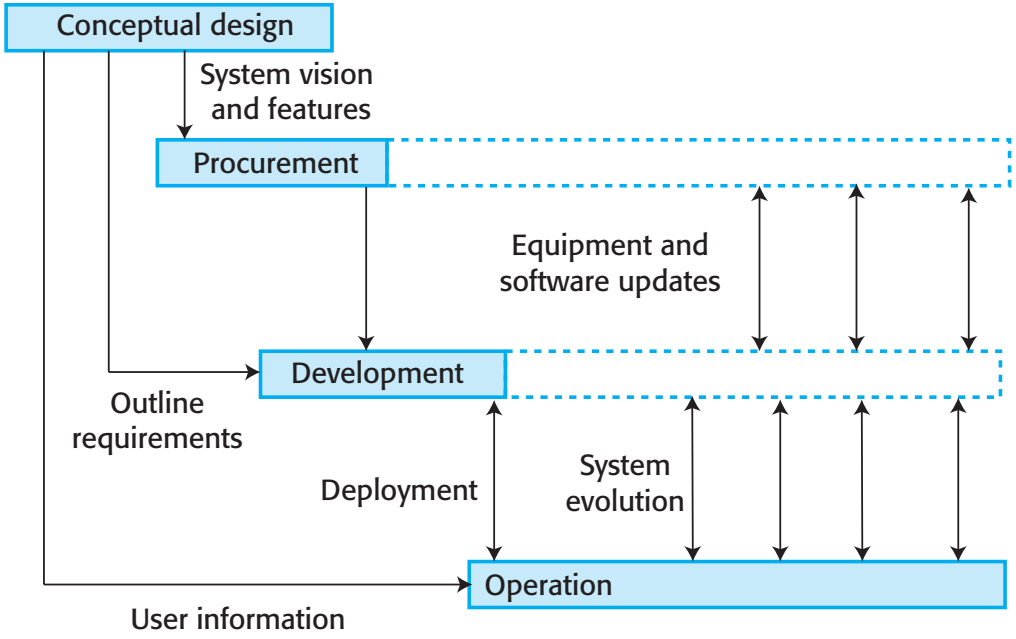
Systems and software engineering

- Software is now the dominant element in all enterprise systems.
  - Software engineers have to play a more active part in high-level systems decision making if the system software is to be dependable and developed on time and to budget.
  - Systems engineers have to include software considerations in their decision making
- Needed is a broader awareness of how software interacts with other hardware and software systems, and the human, social and organizational factors that affect the ways in which software is used.

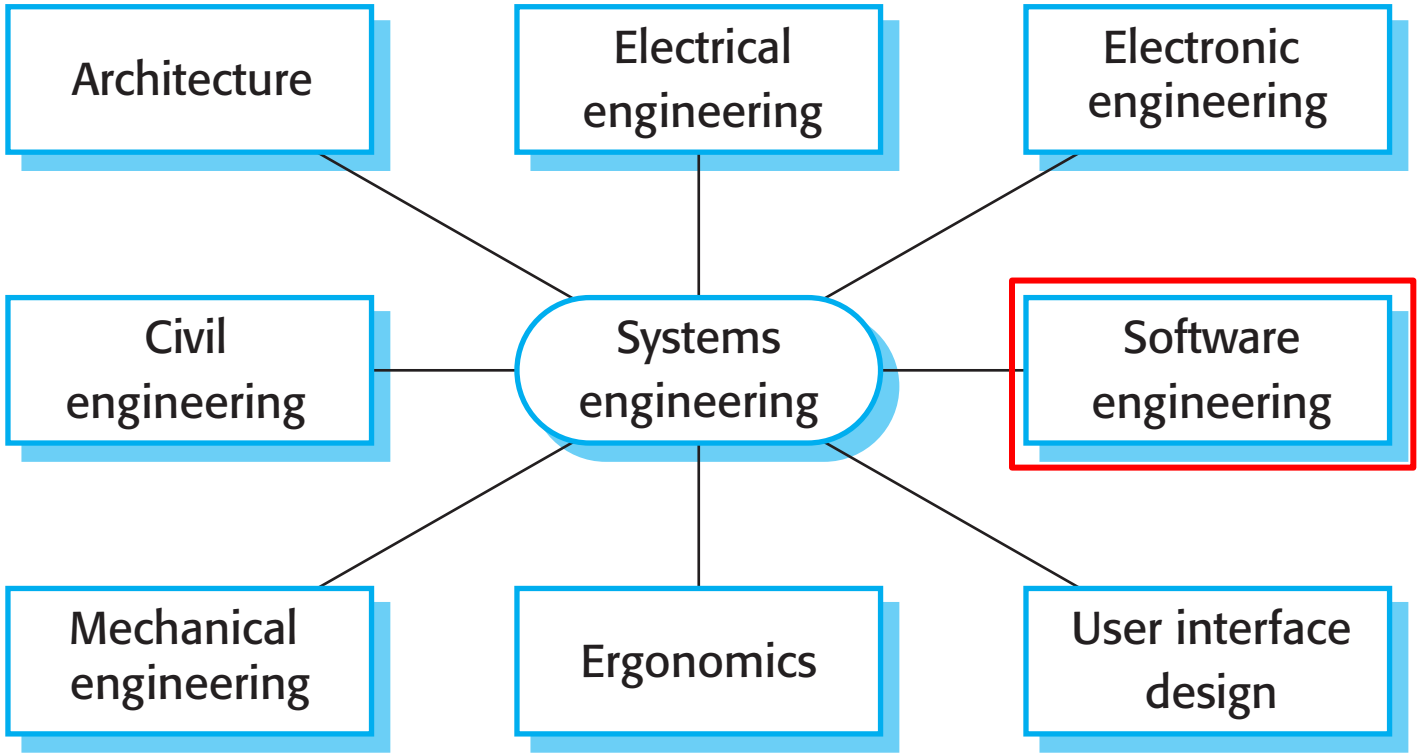
Systems engineering

- Procuring, specifying, designing, implementing, validating, deploying and maintaining systems.
- Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used to fulfill its purpose or purposes.

Systems engineering stages



Professional disciplines involved



Inter-disciplinary challenges

- Communication difficulties
  - Different disciplines use the same terminology to mean different things. This can lead to misunderstandings about what will be implemented.
- Differing assumptions
  - Each discipline makes assumptions about what can and can't be done by other disciplines.
- Professional boundaries
  - Each discipline tries to protect their professional boundaries and expertise and this affects their judgments on the system.

System characteristics

- Emergent properties
  - Properties of the system of a whole that depend on the system components and their relationships.
  - E.g., performance, reliability, usability, safety and security
- Non-deterministic
  - They do not always produce the same output when presented with the same input because the systems' behavior is partially dependent on human operators.
- Complex relationships with organizational objectives
  - The extent to which the system supports organizational objectives does not just depend on the system itself.

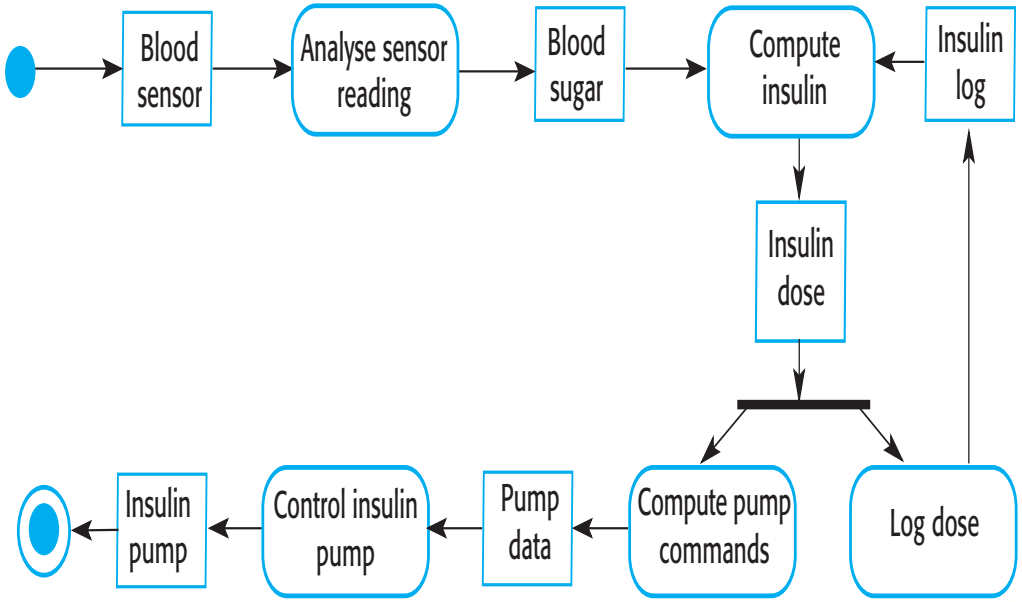
Emergent properties

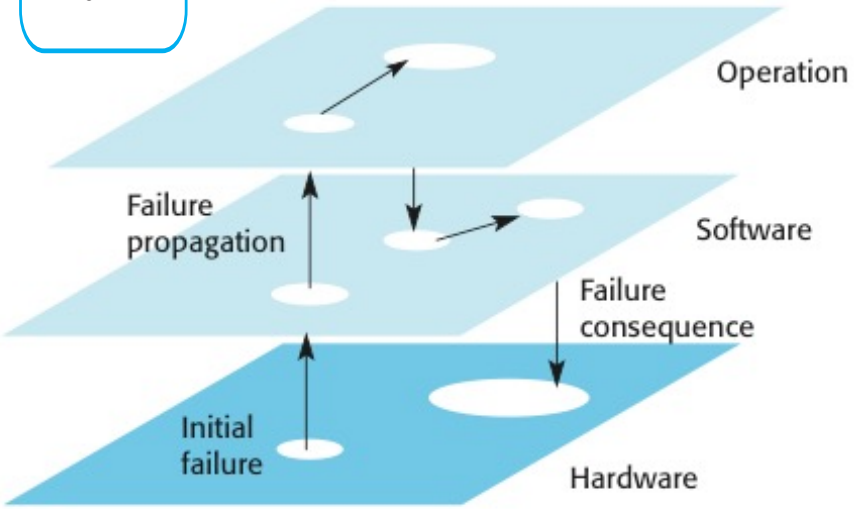
- Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
- Emergent properties are a consequence of the relationships between system components
- They can therefore only be fully assessed and measured once the components have been integrated into a system

Reliability as an emergent property

- Because of component inter-dependencies, faults can be propagated through the system.
- System failures often occur because of unforeseen inter-relationships between components.
  - Broadly consider “components” to include the environment of operations as well as internal parts
- It is practically impossible to anticipate all possible component relationships.
- Software reliability measures may give a false picture of the overall system reliability.

Failure propagation

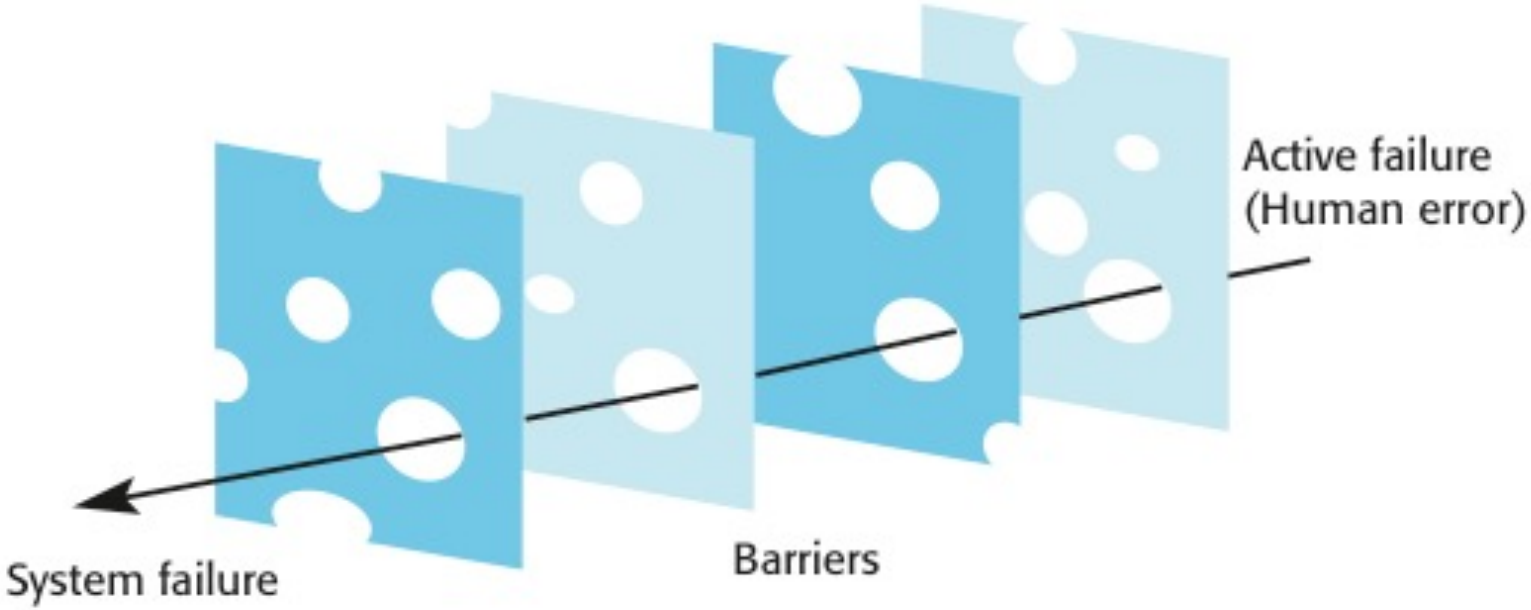




Non-determinism

- A deterministic system is one where a given sequence of inputs will always produce the same sequence of outputs.
- Software (by itself) is deterministic; systems that include humans (socio-technical systems) are non-deterministic
  - A socio-technical system will not always produce the same sequence of outputs from the same input sequence
  - Human elements
    - People do not always behave in the same way
  - System changes
    - System behavior is unpredictable because of frequent changes to environment, hardware, software and data.

Reason's Swiss cheese model of system failure



- Defensive layers within a system can stop the propagation of an error
- These layers can be software and hardware – the more diverse, the better

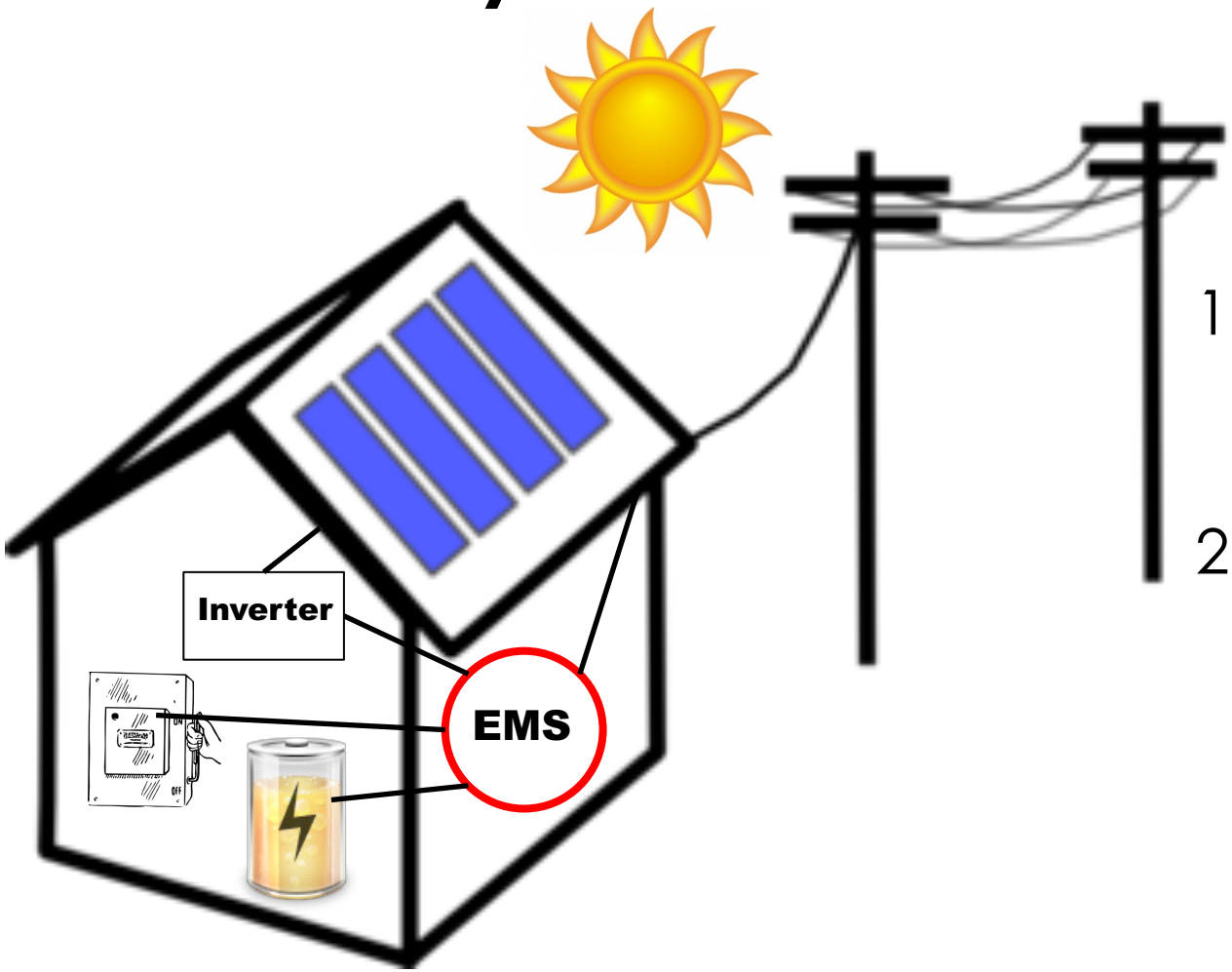
Success criteria

- Complex systems are developed to address ‘wicked problems’ – problems where there cannot be a complete specification.
- Different stakeholders see the problem in different ways and each has a partial understanding of the issues affecting the system.
- Consequently, different stakeholders have their own views about whether or not a system is ‘successful’
  - Success is a judgment and cannot be objectively measured.
  - Success is judged using the effectiveness of the system when deployed rather than judged against the original reasons for procurement.

System evolution for complex systems is costly

- Proposed changes have to be analyzed very carefully from a business and a technical perspective.
- Subsystems are never completely independent so changes to a subsystem may have side-effects that adversely affect other subsystems.
- Reasons for original design decisions are often unrecorded. Those responsible for the system evolution have to work out why these decisions were made.
- As systems age, their structure becomes corrupted by change so the costs of making further changes increases.

The Energy Management Software System - EMSS



1. Who has a stake in the success of this project?
2. What is their "interest" and/or responsibility?

Your turn…

In many parts of the world today, the availability of electricity and the stability of the power grid are major issues. Renewable energy installations can provide solutions to these problems, but only with appropriate energy management systems.

Your project this term will be the design of software for an energy management system that operates with residential/small business solar installations to provide real-time power generation for the home/business, overflow power directed to the central utility grid and power (battery) storage for off-grid operation.

In the coming weeks, you will develop software requirements and use cases for this Energy Management System, working in teams of three to four students. Your requirements will support the sale of surplus generated power back into the power grid, but also the sale of power from the grid to the home/office to support the excess needs of the unit. Each week's assignment will add content to your software plan.