Introduction to Software Engineering

The “computing entity”, software/hardware divide, algorithmic concepts, modern solution development, modelling philosophy and techniques, system concepts and aspects, model conversion, development phases, abstraction, solution complexity control, modularity and solution efficiency.

This document © 2003-2016 Dr Ernest Cachia
The intention of this part of the course is to introduce the basic concepts of solution development as a rigorously structured approach. Structure and its application will be introduced and discussed. This part of the course will introduce one to the benefits that can be gained in terms of solution relevance and stability through the control of solution complexity. This part will introduce the basic concepts of functional association through data flow, modularity, module structure and relationships, through the use of basic development paradigms such as stepwise refinement and levels of abstraction. This part will also introduce entity-event modelling concepts and its relevance to solution quality. Throughout this part, the student will be exposed to various standard analysis and design paradigms and notations.

This part of this course will deal with both the human and the technical dimension of software development.
General Introduction

- Don’t we have to learn all this structured stuff because we need to write “good” programs?
- Do modern programs really need to be all that good from the start? ... and BTW, why do we bother with programming in the first place?
- Aren’t machines powerful enough to simply be “told” what to do?
- If I am to follow structure in my development to be like everyone else, you’re curbing my creativity – isn’t that totalitarian?
- If one is a “good” developer, one doesn't need a “guidebook” to how to develop. So all this is probably for people who don’t know how to think well – right?
- **So, let’s forget about this stuff and go have a drink...**

Yeah, you wish!
Session Contents

- Automated solution development
- The computing entity and the notion of data
- Historical precedent
- Modern ICT solution development
- The software crisis
- Quality issues emergent from the crisis
- Programming and its constructs – where do they hail from?
Automated Solution Development

In the modern sense, this generally implies:

– A computerised solution when relating to the field of ICT;
– The presence of a Human-Computer Interface (HCI) or a Computer-Computer Interface (CCI), or both;
– A solution to improve a well-defined process happening in the real world;
– A solution that can be clearly explained and at times even of formally proven correctness;
– A solution that is cost effective and “lasts” for as long as possible.
Consider the nature of a computer as a tool

- Non conventional in that it’s universal
- Reasons for it being so (separation of entities)

- The current computer organisation model (Von Neumann)
The parts of any model of a computerised solution are:

- The processing part
- The data storage part
- The input/output part

Therefore, the abstract control structure should basically conform to this model.
The components of any algorithm are tightly linked to the hardware organisation that they control. Therefore, in the case of a modern “Von Neumann” computer, these would be:

- Processing (executive) components
- Data storage (memory) components
- Inputting (data receiving) components
- Outputting (data transmitting) components
Data

In singular form, “datum”.

Datum \(\text{Da"tum}\), n.; pl. Data. Something given or admitted; a fact or principle granted; that upon which an inference or an argument is based; -- used chiefly in the plural.

[Taken from Webster’s Revised Unabridged Dictionary, through dict.org]

Some interesting facts about data:

• Data in its pure form is actually meaningless;

• For data to convey meaning it must be given structure.

Therefore, one can say that (Information = Data + Structure)
At a conceptual level, these are basically two:

- **Processing data**
  - That is, the data that implements the algorithm of the solution itself.

- **Processed data**
  - That is, the data that is used by the algorithm of the solution.
An abstract framework in control of the hardware

- Not a modern idea - has been around since Victorian era (Charles Babbage & Ada Lovelace, daughter of Lord Byron)
- Renders a computing system different from other tools (the notion of “tool universality”)
- Forces a more formal definition of process models
- Must match human thought patterns
- Must be able to be represented
- Must be understandable to both human and machine
The concept of data is not a modern idea - has been around since Victorian era (Charles Babbage & Ada Lovelace, daughter of Lord Byron)

Incidentally, interesting to note: Lord George Gordon Byron (cka Lord Byron) was a renowned 18/19th century English poet and fought against the Ottoman Empire. Augusta Ada King, nee Byron, Countess of Lovelace (cka Ada Lovelace) was his only child in wedlock. Ada Lovelace is considered to be the world’s first programmer. She was a keen mathematician. The Ada Programming Language, created by the US Department of Defence in 1980, was named after her. It was one of the very first Object-Oriented languages and extends the Pascal language with many concurrency features. The latest version is “Ada 2012”. Since 2008, there is an annual Ada Lovelace medal given to women computer science students issued as part of the British Computer Society (BSC) “Lovelace Colloquium” competition. In the 1830s/40s Ada wrote the world’s first programs for Charles Babbage’s Difference Engine and Analytical Engine (the latter never completely built). Babbage was an 18/19th century English polymath – a mathematician, philosopher and mechanical engineer. [adapted from various sources, printed and on-line]
Some interesting facts about data

- Data in its pure form is actually meaningless;
- For data to convey meaning it must be given structure;
- There are two fundamentally different types of data.

*Therefore, in a manner, one can say that (Information = Data + Structure)*

“You can have data without information, but you cannot have information without data.” - Daniel Keys Moran
At a conceptual level, the two types of data are:

• **Processing data (aka “control”)**
  The data that *is* the algorithm of the solution.

• **Processed data (aka “data”)**
  The data that *is used by* the algorithm of the solution.

Consider a university and all its administration without students. All the processing is in place, but there is nothing to process!
What came first, the chicken or the egg?

(What I call) The “Michael Jackson Effect”

*Paraphrased and re-interpreted, of course...*

The structure of an information system must mirror that of its data

*An implication of this?*

Such a system’s functionality must be built *around* the data it processes

So, what comes first in modern information systems?

...Data, of course!
Data accessibility

- Human conception issues
- Cultural diversity issues
- User preference issues
- Background and technical expertise issues
- Business/Organisational issues

- Metaphors
- Mental models
- Model navigation rules
- Look-and-feel

- Background
- Expectations
- Level of education
- Character
- Peer pressure

- Language size
- Character representation
- Scripting rules
- Semantic issues
- Beliefs
- Social sensitivities
- Gender sensitivity

- Making the connection
- Extending the known
- User Technical capabilities
- Preconceptions
- Bias and expectations

- Standardisation and profiles
- Logos and “corporate look”
- Perceived efficiency of use
- Customisation
- Generalisation
- Nature of business
Emphasis Over Time – usability
The march of “Technology”

- Hardware
  - Broke down often
  - Slow performance
  - Difficult to maintain
  - Too large
  - Too fragile
  - Too power hungry
  - Too user unfriendly
  - Very restricted use
  - Etc.

- Computation
  - Not efficient
  - Not optimised
  - Booming industry demands
  - Not secure
  - Not reliable
  - Not collaborative
  - Not measurable
  - Etc.

- Data
  - Boom in generation
  - Size and penetration of market
  - Social awareness
  - Mobility
  - Access
  - Structuring
  - Consistency
  - Etc.

- Usefulness
  - Relevance
  - Information overload
  - Semantic Web
  - Veracity
  - Privacy & rights
  - Commodity vs Security
  - Pervasive vs invasive
  - Etc.
Emphasis Over Time – networking
The march of “Communication”

- Sharing messages
  - Asynchronous
  - Short
  - Limited formats
  - Low verification
  - Low bandwidth
  - Etc.

- Sharing data
  - Asynchronous
  - Constrained formats
  - Size limitations
  - Limited reach
  - Etc.

- Sharing information
  - Linked data
  - Compression
  - Interoperability
  - Wide range of formats
  - Etc.

- Sharing value
  - Collaborative
  - Shared environments
  - e-Commerce
  - Entertainment
  - Etc.

- Sharing things
  - MM-Communication
  - Autonomous behaviour
  - Real-Time requirements
  - Etc.
Emphasis Over Time – Data
The march of “Information”

- Raw Data
  - Direct representation
  - Not intuitive
  - Fundamental
  - Internal
  - Etc.

- Coded Data
  - Not intuitive
  - Cumbersome
  - Internal
  - Specific
  - Minimally informative
  - Etc.

- Structured Data
  - Meaningful
  - Expressive
  - Direct meaning
  - Abstract-able
  - Functionally map-able
  - Etc.

- Information
  - Intuitive
  - Shareable
  - Wider usability
  - Process related
  - Non-technical
  - Etc.

- Value
  - Commercial commodity
  - Trading potential
  - Morphological properties
  - Efficient
  - Etc.
Today’s ICT Realities

**KNOWLEDGE & ON-DEMAND CULTURE**

- People (us!)
- Mobile technology
- Cloud technology
- Sensor technology
- Social networks
- Communication
- Business processes
- Data overload
Data is now of crucial business importance... don’t take it from me, see what these people are saying

“Data really powers everything that we do.” – Jeff Weiner, CEO of LinkedIn;
“Data is the new science. Big Data holds the answers.” – Pat Gelsinger, the CEO of VMware, Inc;
“Data are becoming the new raw material of business.” – Craig Mundie, Senior Advisor to the CEO at Microsoft;
“Data matures like wine, applications like fish.” – James Governor, founder or RedMonk and Shoreditch Works;
“The world is one big data problem.” – Andrew McAfee, the associate director of the Center for Digital Business at the MIT Sloan School of Management;
“Without big data, you are blind and deaf and in the middle of a freeway.” – Geoffrey Moore, Marketing Executive at Rand Information Systems, founder of the Chasm Group, marketing consultant;
“For every two degrees the temperature goes up, check-ins at ice cream shops go up by 2%.” – Andrew Hogue, Foursquare;
“Getting information off the Internet is like taking a drink from a firehose.” – Mitchell Kapor, founder of Lotus Development Corporation and designer of Lotus 1-2-3; investor founder of Mozilla Firefox, creator of Second Life;
“The world is one big data problem.” – Ronald Coase, Economics, Nobel Prize Laureate;
“Data beats emotions.” – Sean Rad, founder of Adly, a leading digital marketing software and services provider;
“The goal is to turn data into information, and information into insight.” – Carly Fiorina, former Chief Executive of Hewlett-Packard (HP);
The fundamental nature of data

“A friendship founded on business is a good deal better than a business founded on friendship.”

www.forbes.com

“A solution founded on information is a good deal better than information founded on a solution.”

Author’s personal re-interpretation
Generating Data (1/6)

(Next 6 slides are taken from various on-line sources 2011 to 2013)

The CERN Large Hadron Collider (LHC) generates 1 Petabyte of data per second.
Generating Data (2/6)

In 2012, Google was processing over 26 petabytes of data per day.
Generating Data (3/6)

500 Terabytes of data uploaded daily into Facebook databases.
Generating Data  (4/6)

The whole Internet sees the creation of 1 Exabyte of data per day – the proposed Square Kilometre Array (SKA) telescope on its own is predicted to generate about 1 Exabyte of data per day.
Astronomical Data Deluge

Megadata

Square Kilometre Array

- €1.5 billion global science project
- Astronomers and engineers from more than 70 institutes in 20 countries
- 3000 dishes, each 15m wide
- Using enough optical fibre to wrap twice around the Earth
- A combined collecting area of about one square kilometre

In excess of 1 Exabyte of raw data in a single day - more than the entire daily internet traffic

IBM Information Intensive Framework

- Automated data classification = faster with fewer errors
- Guided search = easier access for scientists and non-scientists alike
- Frees researchers to be more productive and creative

Enough raw data to fill over 15 million 64Gb iPads every day

A prototype software architecture to manage the megadata generated by SKA

Top image: SPOC/Seinbume Astronomy Productions
Generating Data (5/6)

As of May 2013, Netflix had 3.14 petabytes of video “master copies” (out of which they produce 100 different formats for steaming);

One petabyte of average MP3-encoded songs (for mobile, roughly one megabyte per minute), would require 2000 years to play;

The film “Avatar” is reported to have taken over 1 petabyte of local storage at “Weta Digital” for the rendering of the 3D CGI effects;

It is estimated that the human brain’s ability to store memories is equivalent to about 2.5 petabytes of binary data.
Generating Data (6/6)

1 Yottabyte of data constitutes our digital universe today (2013) – it was “just” 3 Zettabytes at the end of 2011 – an over 330-fold increase in under 2 years!...Various predictions seem to agree that data production will be over **40 times greater** in 2020 than it was in 2009.

**Happening every minute...**

98,000 tweets; 695,000 status updates; 11 million instant messages; 700,000 Google searches; 170 million emails sent; 1.82PB of data created. *(Data from Colin Mahony’s presentation “Harnessing the Power of Big Data”, 2013, Hewlett-Packard Development Company)*

---

<table>
<thead>
<tr>
<th>Megabyte</th>
<th>Gigabyte</th>
<th>Terabyte</th>
<th>Petabyte</th>
<th>Exabyte</th>
<th>Zettabyte</th>
<th>Yottabyte</th>
<th>Y-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^{12}$</td>
<td>$10^{15}$</td>
<td>$10^{18}$</td>
<td>$10^{21}$</td>
<td>$10^{24}$</td>
<td>$10^{27}$</td>
</tr>
</tbody>
</table>
The Issue is Clear

“Data is a precious thing and will last longer than the systems themselves.” – Tim Berners-Lee, inventor of the World Wide Web.

Data collection is not something brought about by technology – it has always been there as an inevitable “by-product” of intelligent existence. However, it is accelerated by technology. So tackling the problem as a static one is bound to simply postpone the inevitable – data overload.

What can be done?...
Research Potential and Directions

Real-Time Analytics

Storage & Retrieval Technology

Faculty of ICT
Ernest Cachia Department of Computer Information Systems
(What I call) The “Data Vicious Cycle”

Technologies

Bbreeds & Supports

Innovation

Generates

Data & Information

Necessitates & Rationalises

Produces & Validates

Value

Ernest Cachia Department of Computer Information Systems
Data will be (is?) Tomorrow’s (Today’s?) Money

• **PROFILING**
  Data is being generated in such a dramatic way that it is what defines who we are and what we do;

• **VALIDITY**
  Having the “right” data is ever increasingly becoming a critical commercial commodity;

• **ANALYTICS**
  Information is based on how data is understood, interpreted and represented;

• **VALUE**
  Information can be, and generally is, “different” from the data upon which it is based.
One asks: What lessons have we learned from the past?

• From programming
  – Languages must be versatile;
  – Constructs must mimic natural reasoning;
  – Reasoning must be scalable;
  – Structures must be interoperable.

• From software development
  – Abstraction is crucial;
  – Modularity is a central concept;
  – Collaborative activities are vital for competitiveness;
  – Data modelling must precede functional builds.

• From artificial intelligence
  – Meaning needs to be correctly extracted to have any usefulness;
  – Cognitive analysis can lead to derivation of functionality;
  – Systems can learn and adapt.

...together with other things.
The notion of an algorithm

• A possible *informal* definition
  – A stepwise description of time separated actions with respect to time which are required to perform a definite task

• Steps *preceding* the effective application of an algorithm
  – Conceive the task
  – Define the task
  – Specify the requirements

• Entities *resulting from* the effective application of an algorithm
  – The implemented system or a model of it
  – The system’s complete and matching documentation
  – The software’s full and matching description
Desirable properties of an algorithm

An algorithm should be:

- Clear
- Unambiguous
- Alterable
- Adaptable
- Produce the expected (useful) outcome
- Be demonstrative of its outcome
- Correct (with respect to predefined conditions)
- Exhibit comprehensible functionality
By its nature, an algorithm is an abstract notion. Therefore, to manifest itself in a real sense it must be represented. This representation can take several forms.

- **Forms are generally categorised as:**
  - Diagrammatic
  - Textual
  - Formal

- **And are based on the following system aspects:**
  - Behavioural
  - Data
  - Dynamic
The “status quo” in computing

- Historically hardware has always developed at a faster rate than software;
- This fact was not given real importance till the late 1980s;
- Hardware must be engineered to work at all while software need not;
- Software is *expected* to contain bugs;
- Software is fragile;
- Nowadays, our life could very well depend on software.

*Therefore, one can conclude that software is at least as important as hardware.*

(Nowadays, incredibly more important)
The “moral of the story”

Therefore:

SOFTWARE MUST BE DEVELOPED RATHER THAN SIMPLY WRITTEN.

(Hence the term “Software Development”)

Naturally, this eventually leads to the conclusion that developed solutions must therefore be engineered to guarantee their quality attributes – but more of this in the second year.
Some disturbing facts about software development in the 1980s up to the 1990s:

- Most software was never well planned from the start and was never designed using rigorous design methods;
- Software was not quantifiable or qualify-able until its actual application;
- Software was extremely fragile (i.e. non-robust);
- Software rarely fully matched its requirements (i.e. not effective).

More points on the next slide...
...Continued from previous slide.

- Software development involved considerable duplication of effort;
- The idea of software prototyping was to run the system and do a post-mortem check when it failed;
- When software did work well and reliably there was a rush to find out why!

Therefore, one can say that founding a discipline based on such ad hoc software development methods is not feasible!
The demands on computerised solutions

Nowadays, the demands and expectations from software is dramatically increasing literally by the day. People are now used to the idea of fully computerised solutions in every sphere of life and social sector.

However, areas of additional criticality do exist. These include:

- Real-time embedded systems
- Safety-critical systems
  - High finance
  - Machine automation
- Life-critical systems
  - Civil
  - Military
- Any others I might have left out.
Strange sort of situation

So, basically, this is what we have:

• Machine hardware is generally well engineered (otherwise it probably wouldn’t work at all)
• Software is recognised to be crucial
• Software controls the functioning of machine hardware in an absolute sense
• Software is often not engineered well (or at all)
• Faulty software inevitably means machine (computer) “malfunction”
• People are happy to entrust their dearest possessions, and indeed, their life to computers or computer controlled systems.
A popular depiction

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed
A Software System

• In general, a modern ICT system is one involving co-operating software and hardware providing a solution for a specific (business or otherwise) problem.

• Therefore, a software system can be viewed as a software solution to a specific, or part of a specific, real-world problem.
  – The focus of attention and effort.
  – Generally viewed as one of the main product components of ICT development.

• So, what are the characteristics of a typical software system?
  ...
Attributes of Sound Systems

• They are sophisticated
• They are structured hierarchically
• The depth of the hierarchy is subjective (abstraction)
• The hierarchy is generally made up of recurring components in different arrangements
• They exhibit stronger cohesion than they do coupling
• A valid sophisticated system always evolves from a valid simpler version
Visual Representation of Sys Characteristics

System environment

System boundary

Inputs

The System
(What it does)

Outputs

System Control
(How it is controlled)

Control

Feed-forward

Feed-back
An Unsustainable Situation

• **On one hand...**
  – People build software systems using no particular method
  – People build fragile software systems
  – People build inaccurate software systems
  – Software system development is expensive
  – Software system development requires considerable planning and effort

• **On the other hand...**
  – Modern software systems are ever-increasing in sophistication
  – Demands on software systems is always rising
  – Software systems are what make a computing entity
  – People consciously or unconsciously rely on software for most of their social activities
The panic...

- In the mid 80s people began to worry about the sorry state of software development.
- By the late 80s this worry became a panic, the main reasons being:
  - The evident poor quality of software
  - The huge jump in reliability demand as software started to control even more critical aspects of society
  - The very poor choice (if any) of software design, verification and implementation standards
- People simply didn’t pay serious enough attention to software in the past - now that a crises was created, attention was “force-focused” on it.
The Software Crisis

- The tension created by the conflicting interests outlined in the previous slide gave rise to a software crisis peaking in the late 70s / early 80s.

- The software crisis threatened to cripple the progress of computing as a whole.

- Effects of the crisis still linger in the form of software development always playing “catch-up” to hardware development.
An “unhealthy” state of affairs

Traditional methods of solution construction is not effective for modern system development, because they do not specifically cater for:

- Process control
- Product or process guarantees
- True quality of management
- Client confidence
- Process visibility / traceability
- Metrication
- Communication *(a cornerstone of modern software development)*

*In other words... no quality control.*
The quality crisis

Tools

+ Methods

PROCESS

Q

Q

Q

Ernest Cachia Department of Computer Information Systems

Faculty of ICT
Why do programs look the way they do?

Ever stop to think about this?

• First of all… that’s how we humans think! In sequences, and repetitions of actions subject to decisions.
  – From this fact come the basic algorithmic constructs of sequence, iteration and decision.

• If one had to analyse all the possible combinations of these elements (i.e. sequence, iteration, decision), then an interesting fact surfaces – see the next slide…
Combinations of basic constructs

Taking all combinations of sequence, iteration, and decision, and joining nodes (up to four nodes), produces the following picture:
Summary

• The notions behind modern computerised solution development;
• Historical facts leading to what we use today;
• Algorithmic representation forms;
• The notion of a system;
• Factors leading to and influencing the software crisis;
• The nature of programming constructs.