Unit 1

Introduction to Information Systems (Reflection)

Information Systems

We touched on defining information systems as being about the transformation of data into information and knowledge. Information systems involves more than just data, and at its simplest, consists of **people**, **data** and **processes**. A level deeper, information systems also consist of hardware, and software (which are classified as "technology"). The "people" and "processes" terms make information systems different from purely technical fields.

I consider this definition valid because abstracting every information system always involves at least these three high-level concepts. Since a process accommodates data *inputs* and *outputs* and also operates within a specific context, I see them as being the *engine* in information systems, while data is the *fuel.*

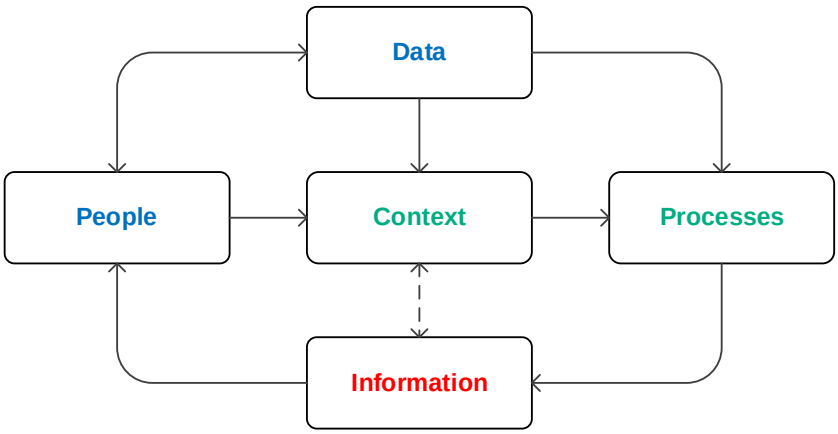

Figure 1 Information Systems with context (created by the author)

When I apply the same abstract concept to organisations, I note a similar pattern of knowledge processing:

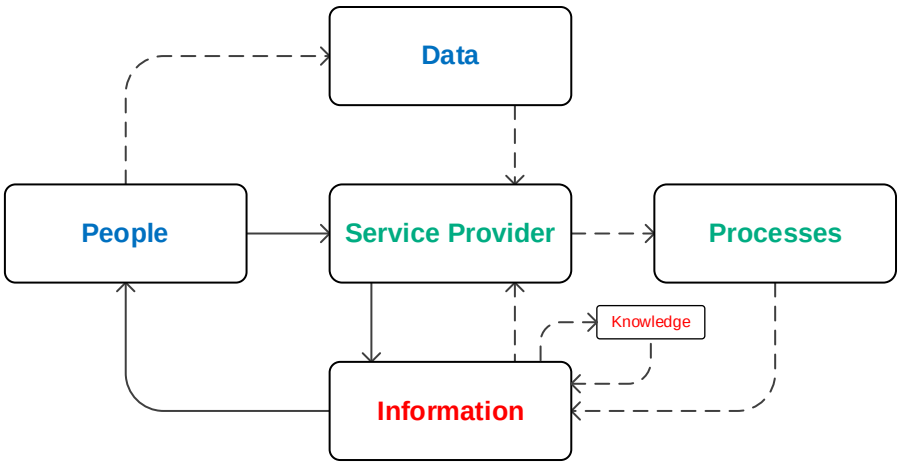

Figure 2 People provide data and meaning to service providers (created by author)

Here, however, Service Providers become the Context of data with People acting as the source of that data. Depending on the providers' processes, customer data *can* potentially turn into useful information they may later leverage for marketing.

Digging deeper into the details of Figure 2 I then considered the role of emerging technology on information systems and find that emerging technologies have a supporting role in every component of an information system.

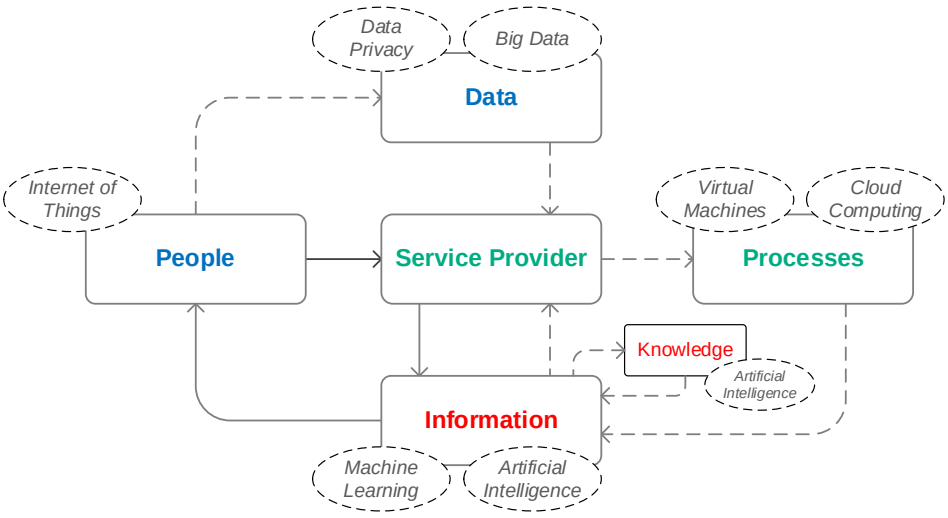

Figure 3 Role of emerging trends in information systems (created by the author)

Therefore, information systems are not merely isolated systems located in a back office performing menial operations; instead, they genuinely impact society and businesses by turning data into knowledge. After all, the *data* is meaningless by itself.

Although information systems facilitate manipulating data into information, and information into knowledge, I contend that *knowledge* is not an intrinsic quality of an information system but is instead a human trait. Knowledge is attributable to biological entities because it builds on experiences, and experiences come through repeated, remembered actions—successful or not. It is knowledge of past experiences that gives guidance on how to handle new challenges. In contrast, while information systems are capable of retraining large quantities of information, they cannot generate new knowledge of their own accord; they have no concept of "knowing" what to do with new/anomalous data inputs. For this reason, I believe one drive behind the rise of artificial intelligence and machine learning is the desire to grant machines the ability to build that knowledge; to work with context.

Building Information Systems

In order to delivery any information system, development teams often follow a *Software Development Lifecycle (SDLC)*. There are three lifecycle approaches

- ***Waterfall.*** This is the traditional approach consisting of six sequential steps: analysis, design, development, testing, implementation and maintenance. Requirements must be defined before moving into a following step and testing is only performed *after* all code has been developed.
- ***V-Model (Validation & Verification Model).*** Built off the Waterfall model and requires validation from previous steps before proceeding to the next. There are a lot more testing-focused steps (unit testing; integration testing; system testing; acceptance testing) performed after the code is developed. In this model, testers and developers can work in parallel.
- ***Rational Unified Process (RUP).*** RUP is an iterative, configurable software engineering process framework used for object-oriented and web system development. It is both a software product and a software engineering process.
- ***Agile.*** Agile is an iterative process. This is best for adaptive teams that can respond quickly to change. The emphasis is on faster, more frequent delivery of sotware. Another aspect is the customer feedback is continuously integrated into the lifecycle. The agile lifecycle consists of short, iterative "cycles" of requirements-develop-test-feedback. Agile avoids up-front requirements gathering (compared to Waterfall),

instead opting for each “cycle” to obtain further requirements for the next cycle of development.

From the above four methodologies, their purpose is to guide the development of information systems, from requirements to final product delivery. Waterfall has fallen out of favour and replaced with Agile in modern projects. Use of Agile is driven in part because of costs associated with software development and that it is cheaper to find and fix bugs early on in the development process, than at some point after the product has been deployed.

Discussion Post

This week I contributed an initial discussion post detailing thoughts on an information system failure experienced by Worldpay, a payments provider in the UK. The system failure was attributed to a software upgrade to one of their payment gateways. The software upgrade (details not specified) resulted in numerous system errors, preventing clients from completing payment transactions via the Worldpay platform. The company had to resort to a manual settlement of transactions while addressing the upgrade issue and downplayed the issue’s severity. Such a system failure is all too common across the industry, and immediately one is inclined to raise the following questions:

1. Was the upgrade tested before deployment?

   All too often, the word “testing”, in my opinion, seems to be left out of the “Software Quality” equation. Forcing customers to perform quality assurance is a poor sign of a proper software development lifecycle.

2. Why was customer communication handled rather poorly?

   To be a trusted service provider relies on a skill called communication. Without this, customers are left high-and-dry with no understanding of why their transactions fail or what they can do to resolve the issue.

3. Does cost correlate to quality?

   Society raises each new generation believing that the more they pay for something, the better the quality of the product. However, there is no correlation data between the cost of software projects and the resulting quality. Thus numerous quality models

exist (Al-Qutaish, 2010) to help quantify the cost of quality. In addition, Hilburn & Townhidnejad (2000) feel that the lack of preparing students to write quality software is another contributing factor to low-quality software.

References

Al-Qutaish, R.E. (2010) Quality models in software engineering literature: an analytical and comparative study. *Journal of American Science* 6(3): 166-175.

Hillburn, TB & Townhidnejad, M (2000) 'Software quality: a curriculum postscript?'. Proceedings *of the thirty-first SIGCSE technical symposium on Computer Science education.* New York, USA, March 2000, SIGCSE. 167-171.

Scott, A. & Menn, J., 2014) Exclusive: Air traffic system failure caused by computer memory shortage. Reuters