1

Information System Development

1.1 Information systems and IT

In the 21st century, the idea of an information system that does not rely on sophisticated information technologies is almost unimaginable. Information systems have always been essential to an organization's survival and effective computer-based information systems have come to be recognized as essential business assets. Laudon & Laudon (2002) suggest four factors that have altered the business landscape and made information systems strategically important:

- *The global economy*: Foreign trade in the United States represents more than 25% of economic activity, and successful organizations must be able to operate globally, compete in world marketplaces, and manage distributors, suppliers, and customers 24 hours a day in many countries. To be successful in the global market, organizations need powerful information systems to support this activity
- *Industrial transformation*: Industrial economies are becoming knowledge-based service-oriented economies. Services are information and knowledge intensive, and many jobs rely on creating and distributing information. Obvious examples include education and healthcare. Even the most tangible of products, such as motorcars, are becoming more and more knowledge intensive both with regard to the processes for designing and building them and in terms of the products themselves. For example, the *General Motors'* OnStar programme allows remote operators to detect when an accident occurs (the air bag is triggered) and to call the emergency

services (the car can even set in motion processes that arrange for its own servicing, such as brake pad replacement). Consequently, manufacturers need less 'metal-bashers' and more designers, engineers, and IT specialists

- *Business transformation*: Traditional businesses are organized hierarchically and rely on standard operating procedures to deliver mass-produced product and services. New business organizations have flattened organizational structures and flexible combinations of resources that are organized around customer requirements to deliver mass-customized products and services. Organizations need considerable investments in information technology to meet these coordination and communication needs
- *Digital firms*: E-businesses conduct relationships with customers, suppliers, and employees and are able to sense and respond to changes in their environment rapidly. An example of a digital firm is *Egg,* a provider of financial services online, including mortgages, savings, insurance, and investments. We consider new forms of digital business in chapter 4.

An information system is a set of interacting components – people, procedures, and technologies – that together collect, process, store, and distribute information to support control, decision-making and management in organizations. The information system contains information about the organization itself, for example, the state of its internal operations, and about its environment, for example, information about customers, suppliers, and competitors. Without an information system an organization would not survive. This does not mean that the information system must use information technologies in the form of computers and communication networks. There are other forms of information system. Organizations have always needed information systems, although the formal aspects of these information systems would have been implemented using paper-based filing systems in the pre-IT era.

Our concern in this book is with *computer-based* information systems. By and large we will address the formal aspects of information systems, but must remember that many information systems in organizations are informal – the office grapevine and conversations at the water-cooler are typical examples of these. Although the informal aspects of information systems are difficult to manage and are not amenable to an engineering approach, their influence should not be under-estimated.

The definition of 'technology' is not without difficulty. Technology can and does often apply to devices such as computers and communications networks, but can also be applied to practices (for example, the software development process), and techniques (for example, database design). Information systems can be expected to make use of information technologies, but are not synonymous with technology. An information system is in essence a human

activity system situated in an organizational context – technology is important to information systems but must be considered jointly with human and organizational dimensions. The WISDM framework for information systems development is in large part intended to help the development team keep organizational, human, and technological perspectives in balance.

1.2 Information systems development methodologies

Fitzgerald (1998) has summarized the evidence for a software crisis and reports that: average completion time for information systems development (ISD) projects ranges from 18 months to 5 years; 68% of projects overrun schedules; 65% exceed budget; 75% face major redesign following initial implementation; and 35% of companies have at least one runaway project. ISD is clearly a difficult task and some would argue that the answer lies in better and more professional approaches to development. One area that organizations have invested in is ISD methodologies. A methodology has been defined by Avison & Fitzgerald (2002) as:

> A collection of procedures, techniques, tools, and documentation aids which will help the system developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub-phases, which will guide the system developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate information systems projects.

There are many justifications for adopting a methodology to guide IS development. According to Fitzgerald (1996), these reasons include:

- The subdivision of a complex process into manageable tasks
- Facilitation of project management and control. One role of management is to manage risk and uncertainty
- Purposeful framework for applying techniques
- Economics – skill specialization and division of labour
- Epistemological – a framework for the acquisition and systemization of knowledge. A methodology should promote organizational learning
- Standardization – interchangeability of developers, increased productivity and quality.

However, Fitzgerald also identifies problems with methodologies. There is a basic issue about what is a methodology (definitional anomalies) and a tendency for methodology authors to engage in 'method wars'. Many methodologies have no or weak conceptual and empirical foundations, and where there are conceptual underpinnings these tend to be rooted firmly in a

scientific and engineering paradigm. Engineering was possibly a suitable reference discipline when development was concerned with hardware and software support for repetitive processes, but it is a crude approach when dealing with social systems.

It is also possible that the methodology becomes an end in its own right. Wastell (1996) noted that:

> Methodology becomes a fetish, a procedure used with pathological rigidity for its own sake, not as a means to an end. Used in this way, methodology provides a relief against anxiety; it insulates the practitioner from the risks and uncertainties of real engagement with people and problems.

It is tempting to assume that a single methodology can be applied universally – to all types of project in all types of organization. Coupled with an inadequate recognition of developer-specific factors, IS development methodologies can become the 'one size fits all' solution. We do not suggest that the response to this should be to declare a free-for-all where IS development is tackled in an ad hoc manner, but it is important to be aware of the dangers of methodologically-induced blindness; a methodology helps organize and frame problems more clearly, but it can also be a way of not seeing.

1.3 Information systems development life-cycles

1.3.1 Waterfall life-cycle

The waterfall life-cycle of systems development subdivides the process into formal stages where the output of one stage forms the input to the next. The deliverables cascade down the waterfall with a completed software system at the bottom. The first stage, the feasibility study, is concerned with defining the scope and purpose of the new system, considering a range of alternative solutions and investigating the impact of the new system on the organization. The decision point following the feasibility study is for management to sign off on a preferred solution.

In the requirements stage, the requirements are specified in logical form, that is, in terms of 'what' is required rather than 'how' it will be achieved technically. The requirements specification should be signed off by the business user – an agreement that the system, if built as specified, will meet the needs of the business.

Design is concerned with translating the requirements into a software system specification that can then be turned into a working system by systems developers (coding). Thorough testing is vital and is often done at three levels. The first level of testing is for the programmers to unit test the program modules. Once the programmers are satisfied with the individual modules,

then the system can be put together for a systems test. The systems development staff develop a comprehensive test plan and devise test data and subject the system to exhaustive testing. This might include peak capacity testing to see how many transactions the system can manage simultaneously before performance degrades to an unacceptable level or the system fails entirely. The third level of testing is the user acceptance test. Once the development team is happy with the systems testing, then the system is handed over to the users who then conduct their own (and independent) test of the system. Once the system successfully passes user acceptance it can be transferred to live operations, where it will then be monitored and maintained as software errors ('bugs') and enhancements are identified.

1.3.2 Problems with the waterfall life-cycle

Taken at face value, the waterfall life-cycle shown in figure 1.1 is appealing for its orderly and systematic stepwise refinement of a complex problem into smaller and smaller problems.

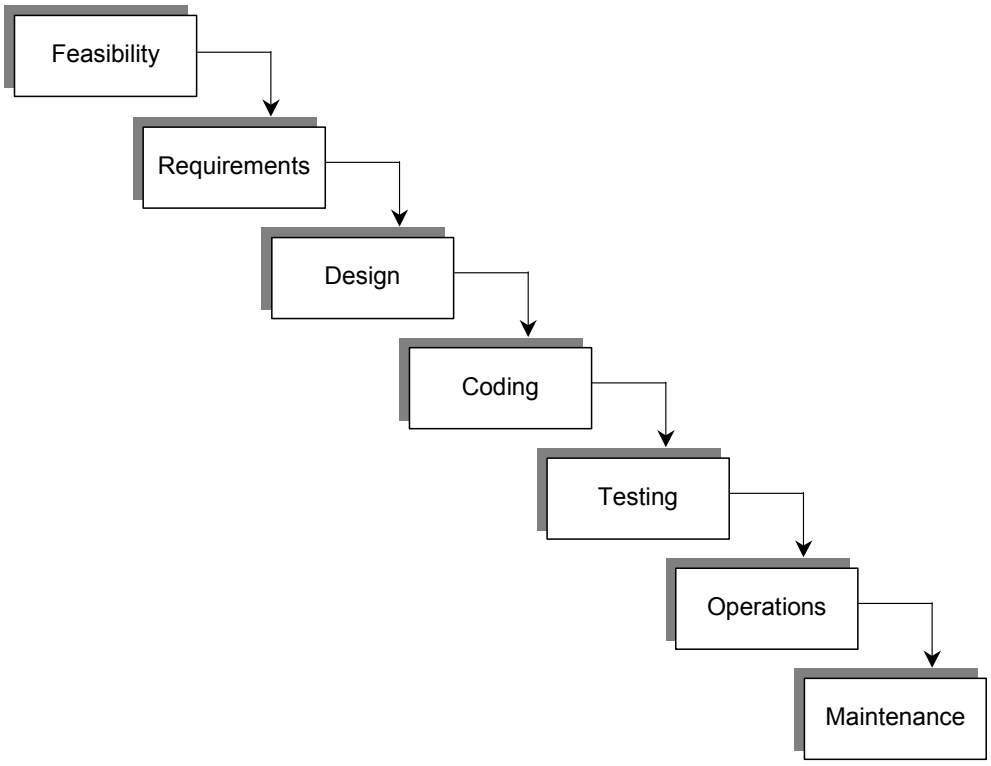

Figure 1.1: The waterfall life-cycle for systems development

The waterfall life-cycle has roots in hardware and software engineering and although suitable for designed artefacts such as computer chips and bridges, it

is less appropriate to human activity systems where the human and organizational factors are less easy to identify. (We would also argue that bridges and chips are human activity systems that can also suffer from a strict engineering-only approach.) For information systems with well-structured requirements, such as an airline reservations system, or applications with a safety critical aspect (for example, a nuclear reactor control system), then the formality of the life-cycle model, possibly combined with formal methods, may well be appropriate.

However, for many information systems development projects, the life-cycle approach has limitations: it is expensive, time-consuming, and inflexible. Perhaps the most significant limitation is that the life-cycle approach assumes that the requirements can be specified with reasonable completeness before design begins. Often, users do not know (or cannot articulate explicitly) the requirements until the finished system is in use. Further, requirements are not fixed – they change over time as the project unfolds and the environment changes. This makes the formulation of a complete and permanently correct specification of requirements an unrealistic goal. It is also difficult to separate out the specification of an information system from its implementation, that is, the 'what' and 'how' of IS development are closely related and difficult to separate. Knowledge of how to implement and what is technically feasible affects perceptions of what can be done in the user world.

The waterfall life-cycle prescribes clear communication points between users and developers (for example, sign off of specification, sign off of user acceptance test) but it does not promote a mutual process of learning and coordination. Furthermore, the formal products of IS development, such as the specification of requirements, are often meaningless to users (even if they had the time and the inclination to read them). Any change to requirements after the sign-off point will be expensive – it is estimated that requirements errors cost 200 times more to fix if not discovered prior to implementation. All of these problems result in an inflation of the maintenance activity as developers attempt to fit the information system to actual working practices based on real experience. As maintenance often comes from a different budget than the original development, many organizations significantly underestimate the true cost of IS development.

So, why is the waterfall life-cycle still used? A cynical view would be that it is used to provide milestones and frozen deliverables to keep managers happy and to give them the illusion that they are in control of the IS development process. A more balanced view suggests that the life-cycle approach may work well in certain circumstances. Where the requirements can be articulated clearly, as might be the case where an existing IS is being replaced and the developers have in-depth experience of the application domain, such as a

payroll or inventory application, the waterfall life-cycle approach may well be appropriate.

1.3.3 Alternative life-cycles

The shortcomings of the waterfall life-cycle have led to more flexible approaches being explored. Alternatives to the waterfall approach include evolutionary development (figure 1.2b) and incremental development (figure 1.2c).

Evolutionary development

In evolutionary development, shown as figure 1.2b, the system is developed using a prototype and refined through user feedback of the system in use and changes in the application itself. The operational system is improved all the time through responses to this user feedback. Change is therefore seen as the norm and catered for – a sharp comparison to the waterfall life-cycle discussed earlier, where change could be very expensive to implement.

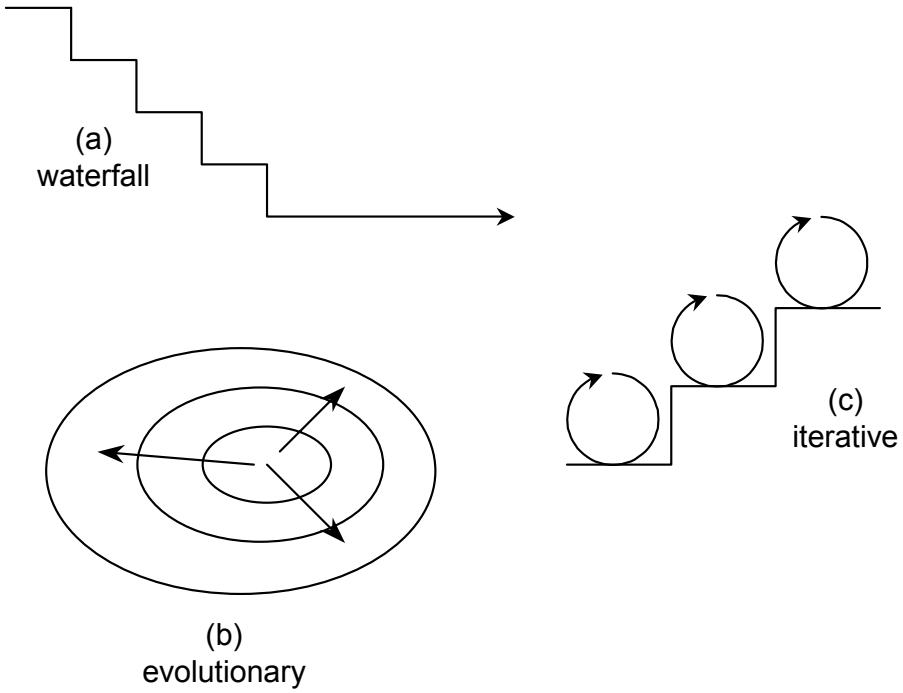

Figure 1.2: Life-cycles for systems development

Iterative (rapid) application development

The key objective of rapid application development (RAD) (figure 1.2c) is the delivery of working business applications in shorter timescales for less investment. With RAD the system is developed in chunks of functionality in

accelerated and time-boxed cycles of development. RAD normally involves users and developers to jointly agree requirements in workshops. In particular, critical requirements (as against those 'nice to have') are identified. Different aspects of the system may be developed at different times, the most important usually being in the first time-box, and the overall system is therefore implemented incrementally in these chunks by small teams of users and developers. The assumptions underlying RAD include (adapted from Beynon-Davies et al., 2000):

- *Iterative incremental* development: an information system can be complete but it is never finished and must be allowed to evolve and adapt over time. Users cannot specify with absolute certainty what they want from an information system in advance and therefore an incremental and iterative approach is needed to explore the requirements as they unfold. RAD is a process of organizational learning through iteration of design, construction, testing, and evaluation of development artefacts
- *Top down refinement*: it is difficult for users to specify requirements accurately and in detail in advance; it is better to specify the requirements at a level that is appropriate to the stage of the development and to develop the detail as the project progresses through incremental development
- *Active user involvement*: the schism of the user/developer divide, which is structurally imposed in the waterfall life-cycle, leads to difficulties in requirements elicitation, ownership, and politics. In RAD the divide is broken down through co-located teams and joint application development workshops
- *Small empowered teams*: teams of four to eight are typical in RAD, promoting effective communications and empowered users. Small teams help reduce bureaucracy and enable quick and effective decision-making. The team is often supported by a facilitator and can benefit from extra-curricular team-building activities
- *Frequent delivery of products*: in RAD the delivery of working applications is constrained to shorter periods, such as 90 days – the aim is to deliver working product, not process
- *Software development toolset*: toolsets are available which allow developers to generate program code with graphical user interfaces from requirements specifications automatically, allowing them to provide the user with prototype applications quickly for evaluation. In RAD the application becomes its own self-documenting model, avoiding the tendency in the waterfall life-cycle to overwhelm the user with large amounts of paper-based documentation.

End-user development

A rather more radical approach to IS development is to hand over the development work to the users. The advantages are obvious, in particular, the users will know the requirements and once involved to the extent of developing the application, will have control over this aspect of their work, thus increasing job satisfaction. The approach has the potential of leading to more successful applications. Many of the limiting factors to end-user development are disappearing. Users use computer systems normally in their job, frequently on personal computers and use applications software, such as the Microsoft Office suite of word processor, spreadsheet and database, and other applications. Such software, particularly MS Excel and MS Access, can be used to develop quite sophisticated applications. Present-day users do not usually have the attitude that computers are nothing to do with them, which used to be commonplace. Further, there are software tools available which are designed to help less-experienced users carry out rudimentary system development for themselves.

1.3.4 Prototyping

Prototyping can be used in all three life-cycle models of figure 1.2. With the traditional waterfall life-cycle, prototypes can be generated in the requirements phase to test out the understanding of the user requirements. This gives the users an executable specification and gets round the problem of poor communications using formal paper-based specifications. It would seem far better to use a form of requirements specification that is similar to the final system. The prototype is typically to be thrown away – it is dispensed with prior to design. As a working system it might be inefficient, incomplete or inadequate in some other way.

In evolutionary development, the prototype evolves through a series of improvements until the users find a version acceptable. Even so, it might be operational fairly early on, but the prototype operational system develops further through incremental improvements and it may never reach a stage where it can be seen as the final version.

In RAD, the prototype typically becomes the final system or at least part of the final system as the time-box modules are implemented in turn. Nevertheless, it might also be thrown away in favour of a more robust design that follows the applicability of the prototype. Thus, to speak of prototyping as a life-cycle for IS development can be misleading.

1.3.5 Agile Development

Agile software development (ASD) encompasses approaches such as Extreme Programming (XP), Lean Development, and Scrum (Highsmith & Cockburn, 2001). As with rapid application development, the premise of ASD is that

change is inevitable. Rather than treat the changes to requirements that arise as a project unfolds as an exception it is better to assume that requirements will change and to focus instead on reducing the cost of the associated rework. Extreme Programming encourages development teams to:

- Produce the first delivery in weeks – get feedback quickly and create credibility through an early win
- Go for simple solutions that can be changed easily and quickly
- Improve design quality so that subsequent versions cost less to build and implement
- Make testing an on-going activity so that errors are caught early.

ASD focuses on the 'keeping it real' through the creation of program code – code is less forgiving (it generally works or it doesn't) than requirement specifications and abstract designs. The Agile Software Manifesto (Highsmith & Cockburn, 2001) values:

- Individuals and interactions over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan.

The manifesto recognizes there is value in the terms on the right, but values the left (italicized terms) more highly. Typical cycles are of two to six weeks duration with Scrum working on 30-day cycles. The agile approach is feature led, rather than project plan driven, with features being prioritized dynamically as circumstances change. The focus on program code and speed has opened ASD up to the criticism of being a hackers' charter. A more balanced view is to see agile methods in a spectrum that ranges from hacking to development based around micro-milestone ironbound contracts (figure 1.3). The choice of where to position a project on the spectrum must take account of a broad range of factors such as the organization culture, the external environment, type of application, and the level of risk.

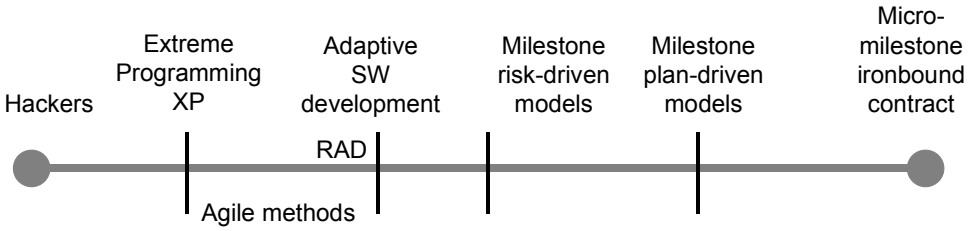

Figure 1.3: IS Development planning spectrum (adapted from Boehm, 2002)

1.4 Alternative strategies for information systems acquisition

In this book we will focus on custom systems development, that is, where the construction of the IS is part of the life-cycle. It is also possible to outsource IS development to a third party, to buy a package solution, or to rent an IS solution. We will look at these in turn.

1.4.1 Packaged solutions

Some organizations have decided not to embark automatically on major in-house system development activities but to first ascertain whether their requirements can be purchased in the form of application packages or 'turn-key' systems. This is regarded as a quick and relatively cheap way of implementing systems for organizations that have fairly standard requirements. A degree of package modification and integration may be required which may still be undertaken in-house. If there is a deep divide between requirements and provision, then the modifications may be very expensive to implement and a packaged solution a poor investment. On the other hand, many applications are similar between organizations, and these are likely to include all the accounting applications. In such cases, an application package is likely to be cheaper and quicker to implement.

1.4.2 Outsourcing

Outsourcing is the provision of services by an external company. In the IS context, it may be as small as the provision of a few contract programmers, to the external development of an application, hardware provision, and finally to the development and running of all applications. It is argued that an outsourcing company is more likely to have the human resources available for a new system and be quicker to develop the application. The vendor may agree to manage and be responsible for the provision and development of the information system for the organization who will be less concerned about how the system is developed and more interested in the end results. There is a trend toward outsourcing as it rids the organization of IT and IS responsibilities, which may be seen as being not of strategic importance – it is not their core business. Nevertheless, it is usually regarded as important that the organization maintains some residual IS specialists to ensure that the vendor is fulfilling both the contract and the organization's needs.

1.4.3 Application service providers

A combination of Internet and mobile technologies, IS skills shortage and growing application size and complexity has encouraged the development of a new kind of outsourcing, provided by application service providers (ASP). ASPs are sometimes formed from an alliance between a hardware vendor and a

software provider such as Hewlett-Packard and SAP or IBM and Oracle, or may be third party providers such as the large consultancy groups. An ASP will build the data pool, develop applications, ensure the appropriate level of security, run the applications, allow access, monitor usage, tune, upgrade and enhance the application. Typically, these applications might relate to customer relationship management, enterprise resource planning or e-business. The relationship between an ASP and its client organization is critical, since an ASP places a further layer between the organization and its information resource.

Summary

- Information systems are central to organizational survival.
- A methodology for developing information systems is needed to bring order and structure to a complex technical and organizational process.
- Many traditional methodologies are inflexible, and end up being ignored or followed for their own sake rather than for organizational benefit.
- There are alternative approaches to the stepwise refinement approach of the waterfall life-cycle model that may be more appropriate to organizations.
- Although it is commonplace for organizations to develop their own applications, there are alternatives, such as packaged solutions, outsourcing and application service providers.

Exercises

1. Suggest different types of information system used at your university or another organization: formal, informal, clerical and computerized.
2. For any information system, argue for and against using an information systems development methodology.
3. What are the crucial differences between the waterfall life-cycle model and alternative models for information systems development?
4. What risks might an organization be exposed to in adopting extreme programming and agile development?
5. Argue for and against developing in-house applications and alternatives, such as packaged solutions, outsourcing and application service providers.

Further reading

Avison, D. and Fitzgerald, G., (2002). *Information Systems Development: Methodologies, Techniques and Tools*, 3rd edition, McGraw-Hill, Maidenhead.