Bournemouth University

ssrc
software systems research centre

Success and Failure:

and the importance of Requirements

Dr Keith Phalp

Overview

- Will examine general success and failure characteristics for software projects.
- These include, people, managerial etc.
- Some consideration of technical and phase.
- Of the technical, many studies (Glass, Hall), show that the requirements phase most problematic.
- Hence, will concentrate upon (meat of the lecture) this vital, and overly neglected aspect of projects, as it most crucial, must get right (but even then no guarantee of success).

Factors contributing to success

Kharbanda & Stallworthy, 1986, suggest the following:

- Get/keep good people. Trust them.
- Build a cohesive team.
- Effective communication.
- Divide up a large project. Delegate effectively.
- The Project Manager. [Their **most important** factor.]
- Keep it simple.
- Monitor cost and progress.
- Cut out unnecessary paperwork *(tricky to judge?)*
- Control changes.
- Base your plans on sound estimates (*we might say how later*).

Causes of Failure?

Survey of 150 IT projects in Philips NV during the 1970s, by Geddes, 1990

Causes of Failure

- Poorly defined objectives
- User not involved
- Poor planning and control methods
- User not committed
- Changes in requirements
- Political problems in user organisation

Key Success Factors

Geddes, 1990 (continued)

- Clearly defined objectives (cited by over 90% of respondents)
- High user involvement
- Quality of project manager
- High user commitment.

See also (Chu and Bannister, 1992) for a summary of Geddes' findings.

Success and Failure…

Success Factors: Barnes & Wearne, 1993

- Clear definition and communication of project objectives.
- Assessing risks.
- Involving the project team in decision-making.
- Good management and a committed team.
- Control - but not prevention - of changes.

Causes of Failure: Barnes & Wearne, 1993

- Poor communication
- Inadequate/excessive planning, monitoring and control.
- Failure to properly involve the team in decisions.
- Not learning from the past.

Further opinions on failure

Factors contributing to failure: Ewusi-Mensah, 1997

- Failure to agree or to communicate the project goals.
- Ineffective team.
- Ineffective project manager.
- Little experience with the technology or the applications domain.
- No active support from senior management.

More reasons for failure

Reasons for project failure, Meredith & Mantel, 2005

- Failure to supply the necessary resources for the project.
  - The active support of senior management is required if the project is to succeed.
- Failing to learn from past projects.
- Appointing a project manager who proves incapable of bringing the project to a successful conclusion.
  - A typical problem here is that the project manager is technically competent but lacks some key management attribute.
- Failing to take the time to estimate and plan the project properly at the outset.
- Using the wrong form of organisation for this project.

Experience reports, including:

- Dowson, M., 1997. The ARIANE 5 Software Failure. *ACM SIGSOFT Softw. Eng. Notes*, 22 (2), March 1997, p84.
- Drummond, H., 1996. The politics of risk: trials and tribulations of the Taurus project. *Journal of Information Technology*, 11 (4), Dec. 1996, pp347-357.
- Hougham, M., 1996. London Ambulance Service computer-aided despatch system. *International Journal of Project Management*, Vol.14, No.2, 1996, pp103-110.
- Kirby, E.G., 1996. The importance of recognizing alternative perspectives: an analysis of a failed project. *Int. Journal. of Project Management*, Vol.14, No.4, Aug. 1996, pp209-211.

Technical / Phase

- Chaos reports: 75% of projects failed or failed to deliver key functions.
  - Why? Poor / no requirements engineering.
- Software Runaways (Great Book) – requirements.
- Robert Glass: '*There is little doubt that project requirements are the single biggest cause of trouble on the software project front. Study after study has found that, where there is a failure, requirements problems are usually at the heart of the matter.*' [GLAS98]
- Tracy Hall: People problems and requirements.

What *is* project failure?

"... an information system should only be dubbed a failure when development or operation ceases, leaving supporters dissatisfied with the extent to which the system has served their interests."

(Sauer, 1993)

Success or Failure?

- Can a project that finishes late be considered successful?

- Can a project that finishes on time, within budget and meets specified requirements be considered a failure?

Perceptions of Project Success

- Success criteria for Users and Project Managers differ. (Wateridge, 1998)
- Project Managers: short-term view, judged at system delivery/acceptance.
  - On time? Within budget? Meets spec.?
- Users: longer-term view, based on use of system. E.g. Quality? Usability? Meets needs?
- Implication: Negotiation/Agreement/Visibility of success criteria.

Stakeholders

- Clients.
- Users.
- Business Management.
- Project Manager.
- Development Team.
- Engineers maintaining the system.

Project Post-mortems

- Analysis of the project, identifying what went well, what did nor work well, and what can be learned from this experience.
  - Management, process *and* technical issues.
  - See for example (Meredith & Mantel, 2005).

- Why are these rare?

Observations so far

- Some failures can be attributed to poor project management in conditions where success would otherwise be expected.
  - But: Well-planned projects can and do fail. In particular, senior management can bring about failure – e.g. by starving the project of resources.
- The business environment can contribute to project failure; in extreme cases, the environment can make failure inevitable.
- Good management, coupled with a committed team, can succeed in adverse circumstances.
- The early stages of a project **[requirements]** are particularly important for its success **[most impact];** but they do not *guarantee* success.

More observations

- Management of scope change is vital.
  - It is also difficult to achieve in practice.
- Effective monitoring is required.
  - At least then you know that the project is at risk of failing ...
- Learning from the past - project postmortems - can help to avoid making the same mistake. The definition of "failure" is not straight-forward. Changes of scope and other opportunities to re-plan projects affect perception of success.
- Stakeholders have individual perceptions of success.
- Perceptions of success/failure may change over time.

References so far

- Barnes, M. & Wearne, S., 1993. The future for major project management. *International Journal of Project Management*, 11 (3), Aug. 1993.
- Chu, C. & Bannister, B.J., 1992. Managing Information Technology Projects: CHART(II) Development in a Large Hong Kong Company. *International Journal of Information Management*, 12 (2), June 1992, pp142-157.
- Ewusi-Mensah, K., 1997. Critical Issues in Abandoned Information Systems Development Projects. *Comms. ACM*, 40 (9), Sep. 1997, pp74-80.
- Geddes, M., 1990. Project leadership and the involvement of users in IT projects. *Project Management*, 8 (4), 1990, pp214-216.
- Kharbanda, O.P. & Stallworthy, E.A., 1986. *Successful Projects*. Aldershot: Gower.
- Meredith, J.R. & Mantel, S.J., 2005. *Project Management: A Managerial Approach.* 6th ed. Chichester: Wiley.
- Sauer, C., 1993. *Why Information Systems Fail: A Case Study Approach.* Henley-on-Thames: Alfred Waller.
- Wateridge, J., 1998. How can IS/IT projects be measured for success? *Int. Journal. of Project Management*, 16 (1), Feb. 1998, pp59-63.

Seminar Exercise (1)

- Read either or both these journal articles:
  - Reel, J.S., 1999. Critical Success Factors in Software Projects. *IEEE Software*, 16 (3), May/June 1999, pp18-23.
  - Yeo, K.T., 2002. Critical failure factors in information systems projects. *Intl. Jl. Of Project Management*, 20 (3), 2002, pp241-246.

- What did you learn about:
  - Factors that affect the success of computing projects?
  - The definition of success (or failure) of a computing project?
- What are the limitations of these papers in answering these questions?

Seminar Exercise (2)

- Linberg, K.R., 1999. Software developer perceptions about software project failure: a case study. *Jl. of Systems & Software*, 49 (2-3), 1999, pp177-192.

- Explain why Linberg argues that software developers can perceive a project as a success despite that project being late or cancelled.
- Can Linberg's arguments be extended to other types of computing project?
- [And does anyone have experience relevant to this?]

Seminar Task (3)

- El Emam, K. & Koru, A.G., 2008. A Replicated Survey of IT Software Project Failures. *IEEE Software*, 25 (5), Sep/Oct 2008, pp84-90.

- What did this add to your understanding of the assessment of the success of computing projects, and to your understanding of the factors that contribute to the success of computing projects?

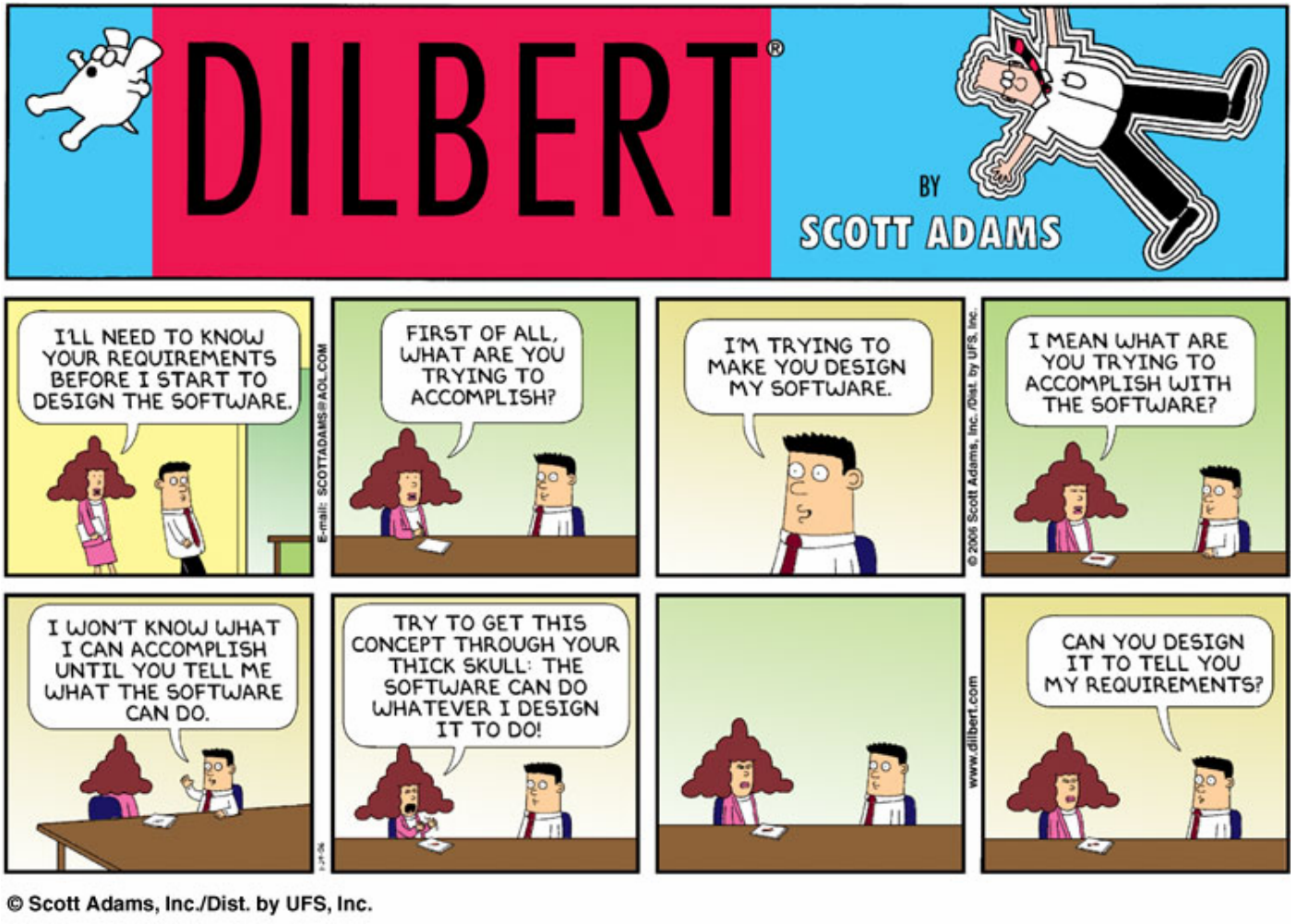
Software Requirements

- Clearly many reason for failure, often attributed to people problems, poor management, poor backing etc.
- However, from an SE perspective the biggest (technical reason) is poor requirements.
- Often cited as the most crucial, most costly, and often poorly understood / executed part of the software process.
- In particular, major issue of business needs and IT requirements. That is, systems can be built but do not satisfy what was really needed (or wanted, or required).
- In examining requirements, will adopt this business needs (customer) perspective.

Overview of the Issue

- Customers want systems to support their business processes. (We can argue about the b word).
- Developers build systems for clients
- “Oh dear. The system doesn’t seem to meet the client’s needs”. (It must be someone’s fault).
- This is a *requirements* problem, and ***very*** common.
- One reason is that the developers didn’t understand the problem: or what they wanted or needed.
- Didn’t do any analysis of the problem (domain).
- Didn’t understand (or model) the client process.

Who is at fault?



Some themes & topics

- We need to unpick this problem. So we will look at:
- What did we mean by requirements.
- Does everyone agree? (Of course agreement is unlikely, even among us).
- How do current approaches fare (and past and proposed and future).
- What might we do to help:
  - What is found to be useful in practice
  - How do we put our ideas together.
- Plus we will look at key research in the area.

Theme 1: What requirements isn't and implications.

- You will have seen some of these arguments before.
- The main thing is to realise that we need to understand that *requirements is about the problem domain*, and is *not the same as specification* (which we will also define).
- Most people mix these up, but:
  - Most people start from specification
  - There is no such thing as a requirements specification.
  - Use cases are really specification.

- Let's start with some views and definitions.

What is Requirements (Analysis)

- Engineering is about **solving problems** (by, **constructing complex, useful artefacts)**
- Requirements (sometimes called analysis) is about **defining those problems** (deciding the **purpose of the artefact).**

- It is a **very good idea** to decide **what** you are going to build before you build it – though often Computing doesn't.
- Typically engineers and programmers want to get going (and their managers too), before figuring out the problem.

Requirements Definitions

- ZAVE (1997)
  - "Requirements engineering… is concerned with the real-world goals for functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour…"
- SWEBOK (2001)
  - "A requirement is defined as a property that must be exhibited in order solve some problem of the real world."
- Robin Goldsmith (2004)
  - "What must be delivered to provide value."

Impact of RE

- Chaos reports: 75% of projects failed or failed to deliver key functions.
  - Why? Poor / no requirements engineering.
- 2004 JobServe report: only 16% of UK software projects are successful.
  - Why? Poor / no analysis skills.
  - *"Projects are often poorly defined, codes of practice are frequently ignored and there is a woeful inability to learn from past experience"*, says Professor McDermid

Requirements Matters

- '*the most crucial activity*' [INCE89]
- '*The most important [phase] by far*' [BRUC89]
- '*the most critical and problem-prone area*' [HOOP82]

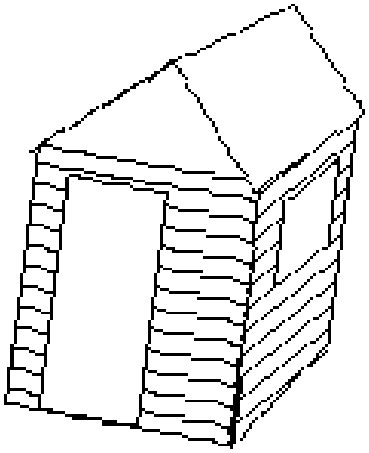
- '*There is little doubt that project requirements are the single biggest cause of trouble on the software project front. Study after study has found that, where there is a failure, requirements problems are usually at the heart of the matter.*' [GLAS98]

RE is at the **start:** it forms the **foundations**

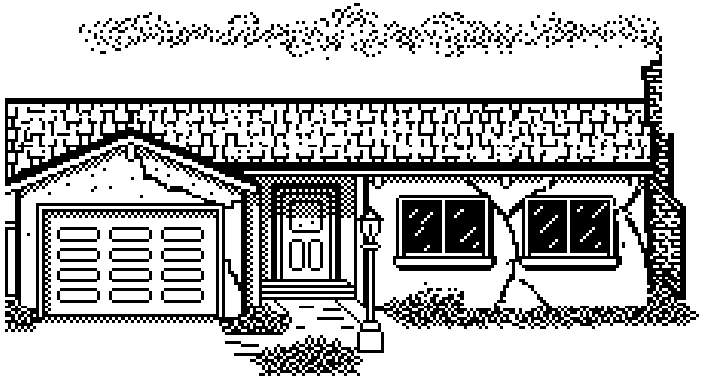
Size Really Does Matter

- Importance of getting the foundations right?

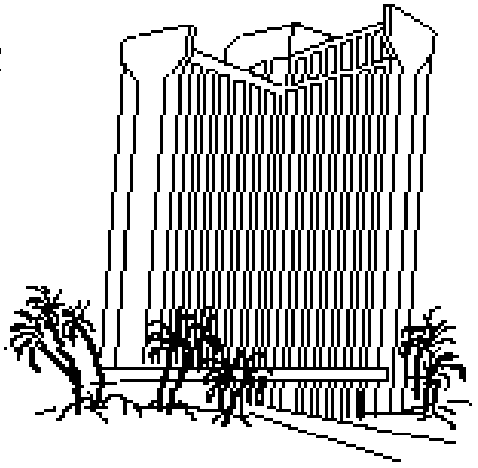
A



B



C



- All taken from:
- Bray, I ‘An introduction to Requirements Engineering’

A stitch in time…

- RE errors are no worse than any others
  - ***provided they are fixed there and then***
- But the cost of fixing escalates
  - Said to be approximately ten times for each major development phase.
- And more software development phases follow RE than the rest.
- The exceptions being business phases: perhaps of the strategic, and business process modelling.

Requirements are Critical

- *'One of the most common reasons systems fail is because the definition of system requirements is bad.'* [SCA81]
- *'The chief villian [sic] in any software fiasco is a non-existent, vague, incomplete or poorly thought out set of requirements.'* [GLAD82]
- *'as much as 90% of subsequent troubles can be traced back to erroneous original specifications.'* [BRUC89]

But… often carried out poorly

- Need domain knowledge & software development expertise (contentious)
- Communication with ‘other types’, with very different worldviews.
- Something from Nothing. It’s been suggested that one ‘invents’ requirements (rather than discovers them).
- Poor technology?
- Lack of expertise, even among software engineers.

Why care about RE

- Estimates vary, but in terms of effort (tome spent) the proportion of the development process spent in each phase is approximately

  - Requirements (including Spec) 30%
  - Design 20%
  - Implementation 20%
  - Testing 30%

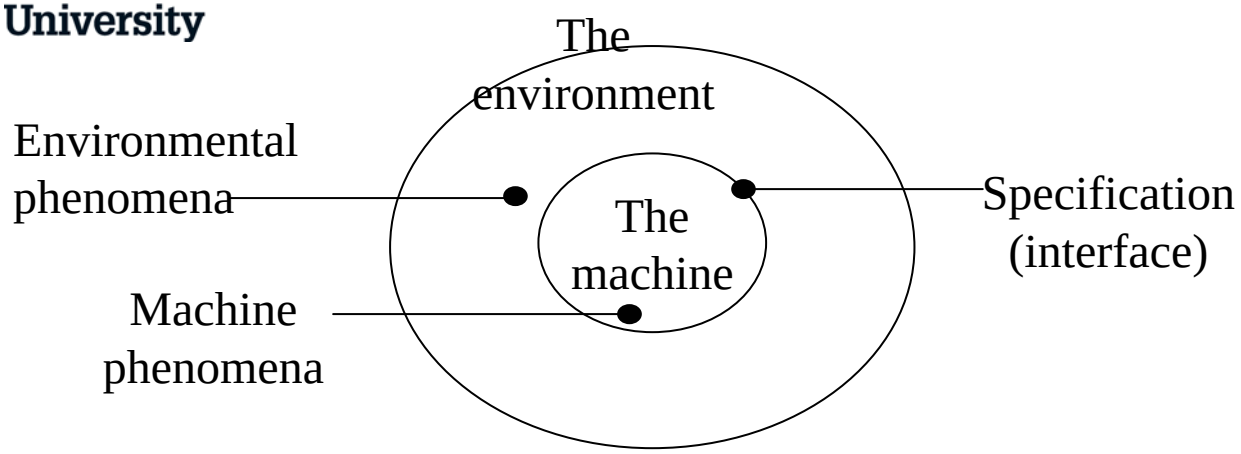
  - Requirements (and possibly testing) are reportedly the least well done phases, with (typically) the least emphasis in education.
  - Similarly, consider the quantity of books devoted even to a single programming language compared to those (relatively few) on requirements (and indeed, test).

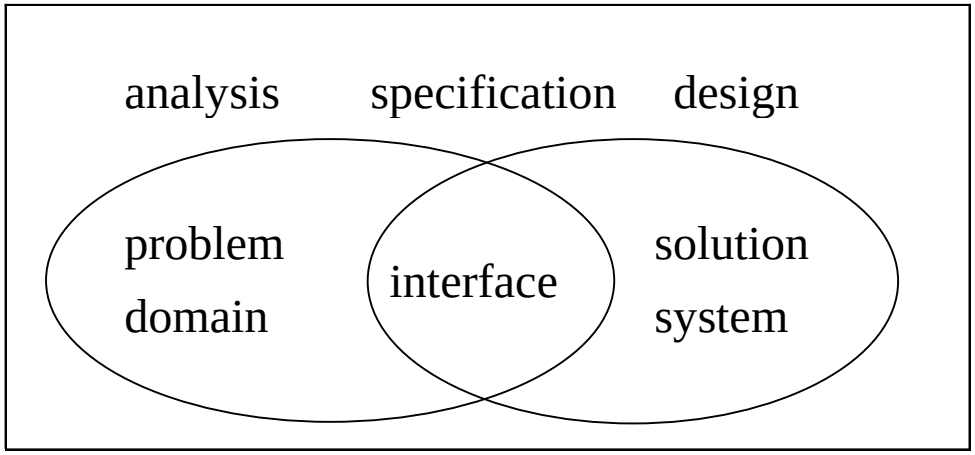
Three descriptions

- Jackson suggests the need to produce three separate, and distinct descriptions.

"In principle you really need three descriptions: the common description; the description that's true only of the machine and the description that's true only of the application domain. If you make all of those descriptions, and separate them carefully, you'll be all right."

Descriptions and phases





Tasks in Requirements

- **Analysis** (Requirements) - concerns study of the problem domain and the problem(s) in it
  - Much misunderstood – as we shall see
- **Specification** - defining the interaction between the solution system and the problem domain
- **Design** - (**not** part of requirements engineering) *largely* concerns inventing the internal workings of the solution system.

Separation of concerns

- Vital to separate distinct tasks, especially, requirements and specification.
- We start by studying the problem domain and producing a **description of the problem domain** and a statement of **the effects that the new system should produce** in the problem domain (i.e., the **requirements**)
- Only then **specify a behaviour** of the **new, solution system,** such that it **will produce the required effects** in the problem domain (the **specification**)

Requirements and Spec

RECAP, or the brief version

- Through a sound understanding of the *problem domain* can we understand and develop a set of REQUIREMENTS for the proposed system.
- Then a SPECIFICATION can be written that should enable system design to occur.

- *Of course this is a rather purist view.*

Requirements

- Requirements: The desired effects that the machine (system) is to have on the problem domain or environment.
  - Purpose: Fulfils the goals.
- Of course there can be different ‘types’ of requirements.

Functional Requirements

- **Functional requirements**
  - The “ordinary” requirements
  - Can be met by appropriate system functionality.
  - **Requirement**: The doors are to be cycled whenever a lift stops at a floor
  - **Corresponding function**:***When the controller detects that a lift has stopped at a floor, it sends the signal to cycle the doors***
- ***But choosing level of abstraction is difficult.***

Performance Requirements

- Parameters of functionality – determine how quickly, how reliably etc. functions must operate
- Often (wrongly) called ‘non-functional’ or ‘non-behavioural’ requirements.
- Usually separated from other (functional) requirements because they are relatively *volatile*.
- Common sub-categories: Speed, Capacity, Reliability, Usability or **SCRU.**

Performance (Speed)

- **Defined in terms of :-**
  - **throughput** (off-line (batch) systems)
  - OR **response times** (interactive or real-time systems)
- E.g.,
  - must process 10,000 transactions per hour
  - must respond to flame-out by closing gas valve within 0.2 seconds

Performance (Capacity)

- The **quantity of data** that can be stored in the system, the **number of simultaneous users, etc, for example:**
  - It will be possible to store at least 10000 transactions
- **Not** to be confused with **constraints** upon the **size of the system, such as:**
  - The new system shall occupy no more than 10 Gbytes of RAM

Performance (Reliability)

- Common to use mean time between failure (really between fault discovery).
  - Note: Software doesn't wear out, faults don't occur, they get found.
- Usually better to specify in terms of **availability**: the proportion of the time (within specified periods) that the system is performing correctly (or, at least, useably).

Performance (Usability)

Best to think in terms of **testability**

Constraints

- The **true** non-functional requirements: (constrain) how the system is built but **not** what it does
- *Under normal usage, would it be apparent to the eventual users of the system whether the requirement has been met?* If **not**, then it is a design constraint.
- Ideally (as developers) we want **no** design constraints…
  - but it doesn't always work out that way

Common Constraints

- target machine(s) upon which the-system must operate
- memory size within which it must operate
- operating system(s) under which it must operate
- programming language(s) that must be used
- other software packages that must be incorporated
- development standards that must be applied
- design methods that must be employed
- algorithms that must be incorporated
- Processes or procedures that must be followed

Summary (so far)

- Problem domain (PD) contains the problem.
- Requirements are the desired effects required in the PD.
- Specify a system to produce the desired effects.
- Activities in RE
  - Analysis - Describes PD and the requirements.
  - Specification - the interface between the PD and the System. OR inventing functions to meet the requirements.

Implications for Requirements

- Need to understand the (whole) *problem* and the client goals, needs and *processes*.
- Need to identify the *system boundary* (and model interactions outside and across).
- Elicitation to directly produce requirements
- Use of process models (existing or new) to inform requirements.
- All before use cases, specification, design.

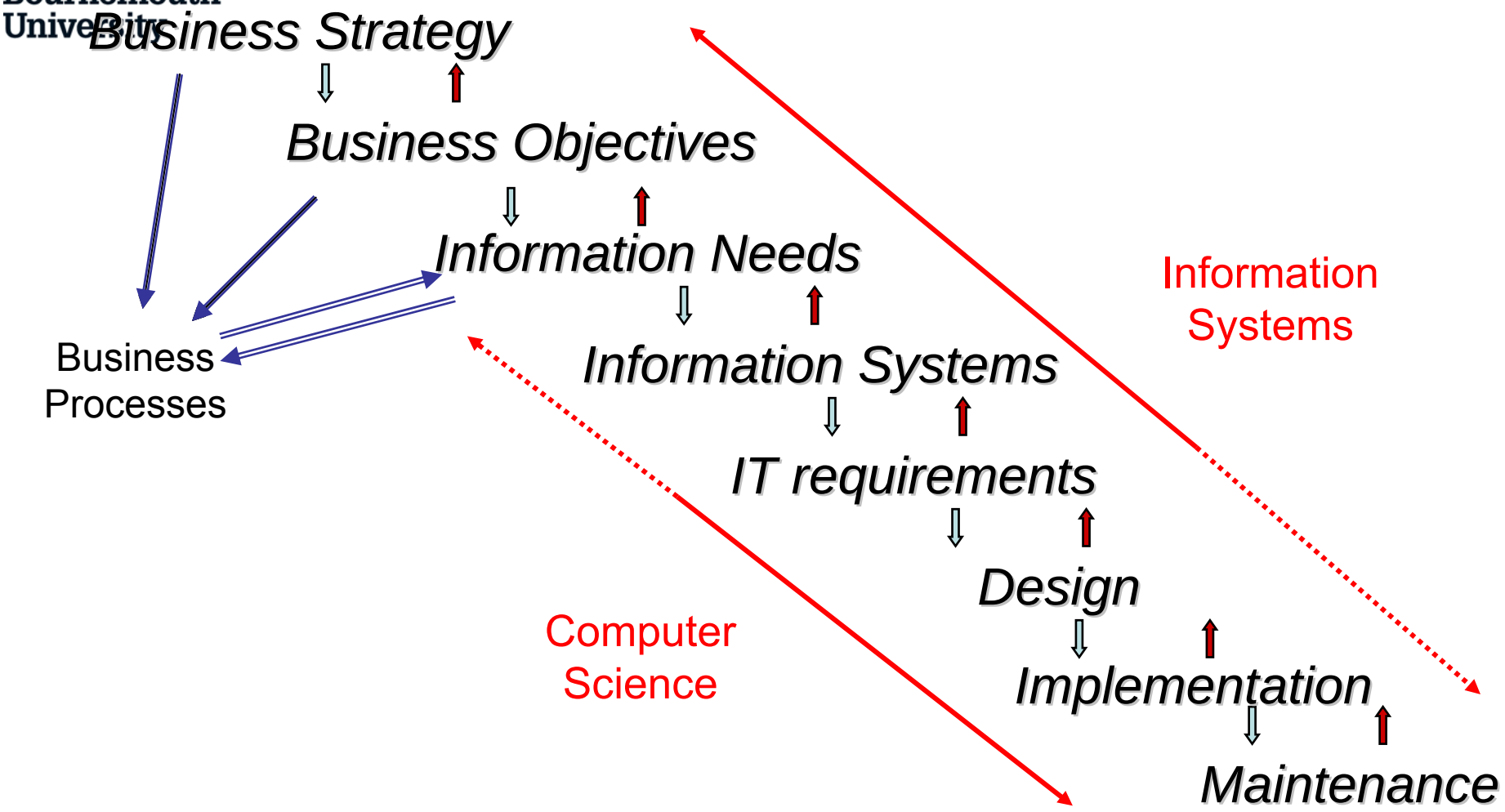
Sources of requirements

- Business need (business processes)
  - i.e. the TASK – *mainly 'functional' requirements, concerned with achieving the business purpose of the system; from business model.* Security/resilience; timeliness; quality.
- User (usability)
  - often 'non-functional' requirements, to do with usability; from user analysis. Related to skills/knowledge, hence "help" and training. Perceptions, prejudices and taste!
- (Physical) operating environment
  - often to do with location, and may relate to physical location; e.g. heat output, power supplies. Panasonic "Tufbook".
- (Regulatory and competitive) operating environment
  - *requirements arising from legislation etc; taken as "givens".*

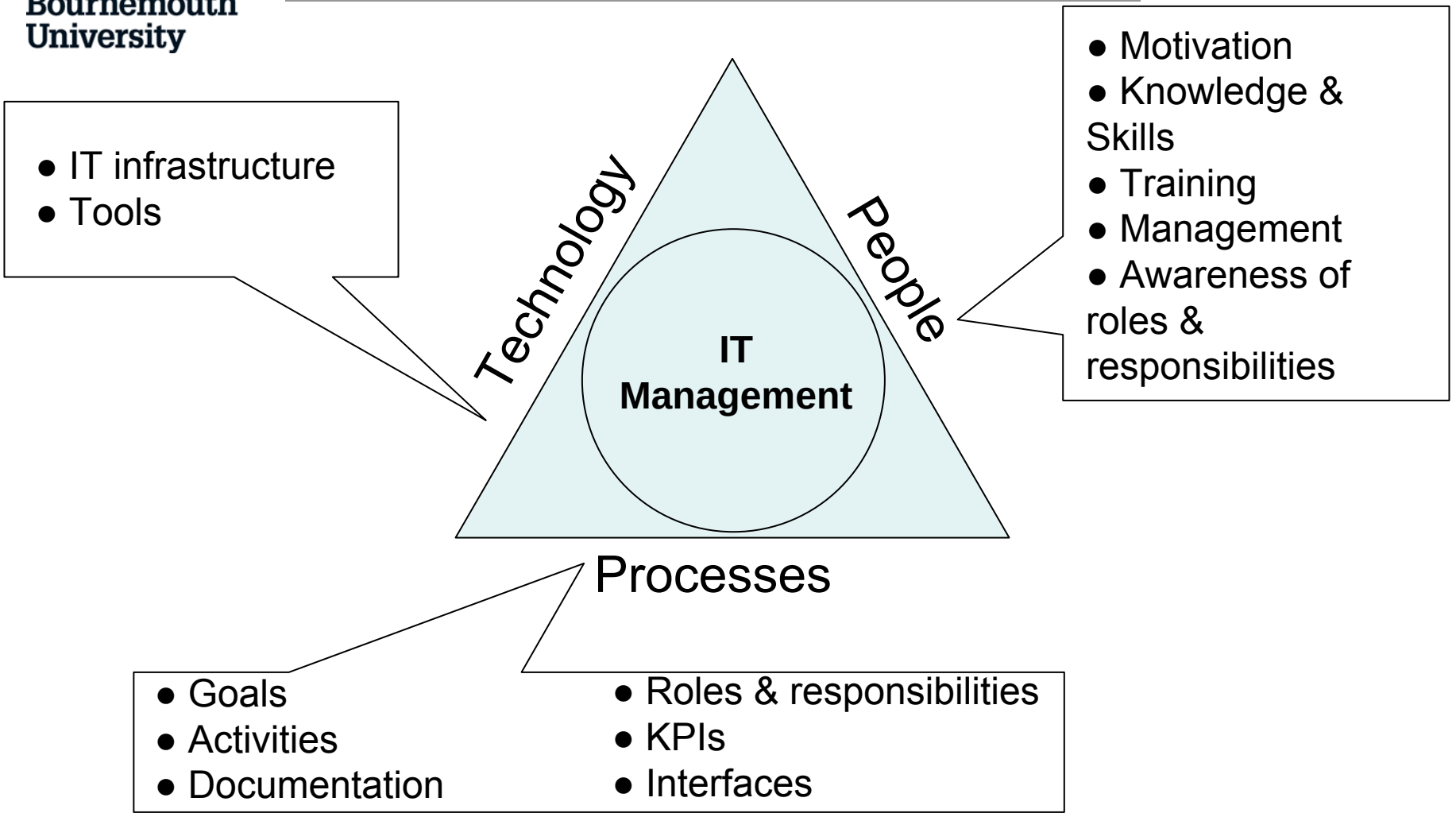
Sources (2)

- Data
  - may dictate performance requirements, storage etc.
- Platform
  - Target Architecture: functional and non-functional requirements; often constraints on functions. What equipment do users of web applications have to have?
- Exceptions
  - How does the system react to errors, e.g. user errors. Type checking of inputs? Error messages?

Relationship Between IT and Business



IT in a Business Context



Summary (sort of)

- Of the technical aspects of development the most crucial is the requirements phase.
- Important to understand what we mean by requirements (understanding the domain).
- Important to separate requirements from specification.
- Need to understand requirements within business context (as you will be the ultimate manager, facilitator for all of these perspectives).
- How we actually inform the requirements of business needs is a topic in itself (but one of my research areas), and involves the use of appropriate process models (ideally mapped to those requirements). See web page for further details.
- Particular issues for multi agency (multiple perspective) projects.

- Before we finish (a word on Model Driven)

A further ‘modern’ view

- Much of this idea, the need to understand the business (perhaps with a process model), as a precursor to specification is reflected in coming (current) approaches to development, such as *model driven development*.

Model Driven Architecture (MDA)

- OMG (2003) *MDA Guide* Version 1.0.1
- Computation Independent Model (CIM)
  - Problem domain and Requirements
- Platform Independent Model (PIM)
  - Class Diagram
  - Activity Diagram
- Platform Specific Model (PSM)
  - Code
- Notation : UML (support for BPMN)

Recap of terms

- The **problem domain** (application domain):
  - that **part of the world** within which the **problem exists** (and within which the solution system will operate)
- **Requirements:**
  - the **effects** that the solution system is **required to produce** in the **problem domain** (PD)
- The **solution system** (SS) (aka. the machine):
  - the **system** (to be produced) that will **bring about** the **required effects** in the **problem domain**
- The **current** system (CS)
  - a *pre-existing* (solution) system that serves a purpose similar to that of the new solution system

Where analysis fits

Tasks :-	**analysis**	**specification**	**design**
Systems :-	**problem domain**	**interface**	**solution system**
Outputs :-	**analysis/ requirements document**	**specification document**	**design document**