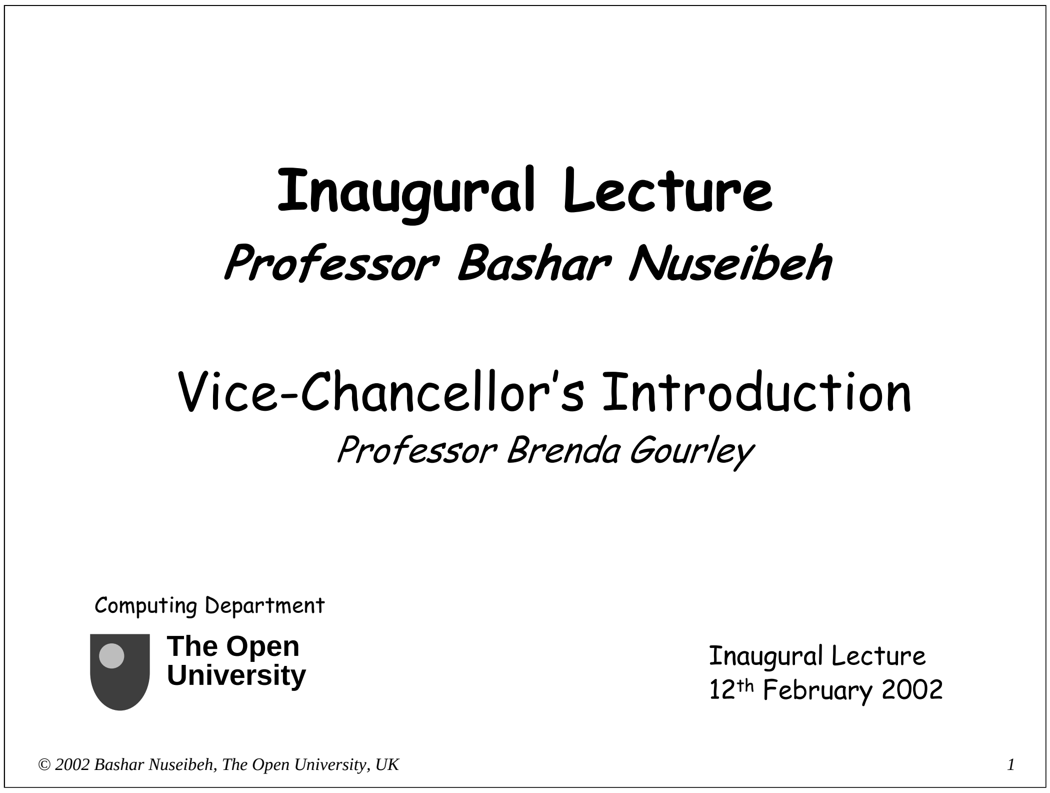


Inaugural Lecture

Professor Bashar Nuseibeh

Vice-Chancellor's Introduction

Professor Brenda Gourley

Computing Department

The Open University

Inaugural Lecture
12th February 2002

1

Ladies and Gentlemen, it is my pleasant task this evening to welcome you to this inaugural lecture by Bashar Nuseibeh, and a very big pleasure it is for me too. I'd like to, in particular, welcome the family of Bashar - very nice to have you here this evening - and friends and colleagues who have travelled from elsewhere to be with us.

I think an inaugural lecture is a special event in the life of the university and we enjoy it in a particular kind of way: celebratory at one point and testing at another. I assured Bashar that nobody had every died from giving an inaugural lecture, and he came straight back and said that nobody had every died from introducing an inauguree! So that was one each, or one-love!

If you look at the roots of the word "inaugural", of course it's "auger", which is sometimes a religious event, in which omens and signs are discerned, and sometimes this entails looking at the entrails of birds and things like that! You can be a prophet or a soothsayer, you can be predicting some future event. We don't have to get into some such messy stuff, we are looking to our new professor to tell us what the future holds for his particular discipline. And we look forward to hearing from him, but in the meantime, let me introduce him in a slightly more formal way.

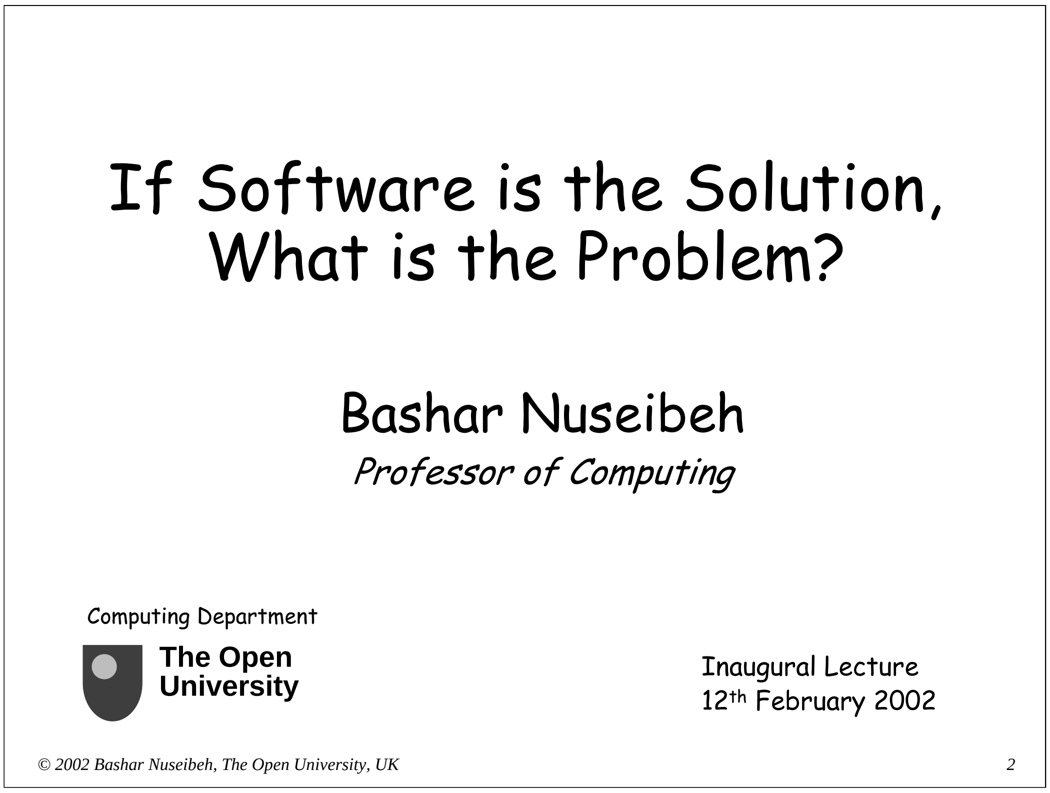
Bashar is a Palestinian from a very old Jerusalem family. He spent his school years in Jordan, Scotland, and the United Arab Emirates.

In 1985, he joined the University of Sussex as an undergraduate in Computer Science, and emerged, three years later, with a First Class Honours degree in Computer Systems Engineering. The following year he obtained a Masters in the Foundations of Advanced Information Technology from Imperial College, London. And, after an eye-opening year in industry – his words , not mine – working as a "Sales Engineer" with an electronic instrumentation company called Tektronix, he eased himself back into academia in 1990 as a Research Assistant at Imperial College, where he completed his PhD, and was then appointed as a Lecturer and then Reader in Computing. When he left Imperial College to join the Open University last year, he was Director of Imperial's Centre for Systems Requirements Engineering, which he co-founded with colleagues at Imperial College, University College London (UCL), and the Institution of Electrical Engineers (IEE).

Bashar's research has often been motivated by the desire to address real industrial problems, and many of his research projects have involved industrial collaborations with organisations like Philips, Texas Instruments, the UK National Air Traffic Services (NATS), and NASA, where he also spent a short sabbatical in 1998.

Bashar engages in the usual collection academic and professional activities – pause for you to guess – serving as Editor-in-Chief of an international journal and Member of various conference programme committees. He actually enjoys organising research conferences in attractive locations, I am told, most recently in Toronto and Honolulu.

Bashar joined the OU as Professor of Computing on 1st January 2001, and promptly took that day off! He has worked a little harder since then, and last month he became Director of Research in the Computing Department. Today he will talk about some of his own research, in his inaugural lecture, which is entitled: *"If Software is the Solution, What is the Problem?"*



If Software is the Solution, What is the Problem?

Bashar Nuseibeh

Professor of Computing

Computing Department

The Open University

Inaugural Lecture
12th February 2002

2

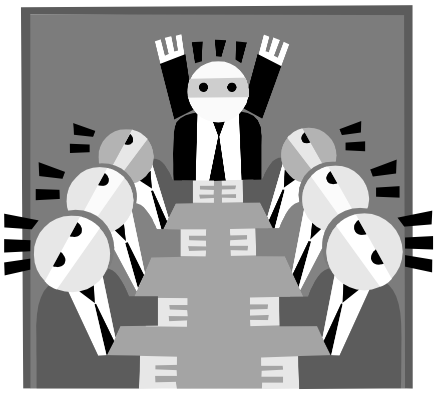
Thank you Vice-Chancellor for that introduction. Good afternoon ladies and gentlemen and thank you to all of you for coming. It really is very nice to see so many friends and colleagues here today. It is also very nerve-racking – I find it much easier to make presentations to strangers than to friends, because I will still have to face my friends the next day!

It is also very dangerous to pose a question in the title of a lecture. It sets up an expectation in the audience that the speaker is going to provide an answer!

Of course, it doesn't help when the title of the lecture itself is also a little ambiguous. In case you were wondering, software, of course, is not the solution, or at least not always the solution to many problems that we face. Please don't misunderstand me, I think software is great – I'm a software engineer myself, looking for better ways to build better software. But I am also a *software requirements engineer,* looking for the right *problems* that software can solve.

This lecture is about discovering problems, and, if necessary, matching those problems to software solutions.

The “voice of the customer”



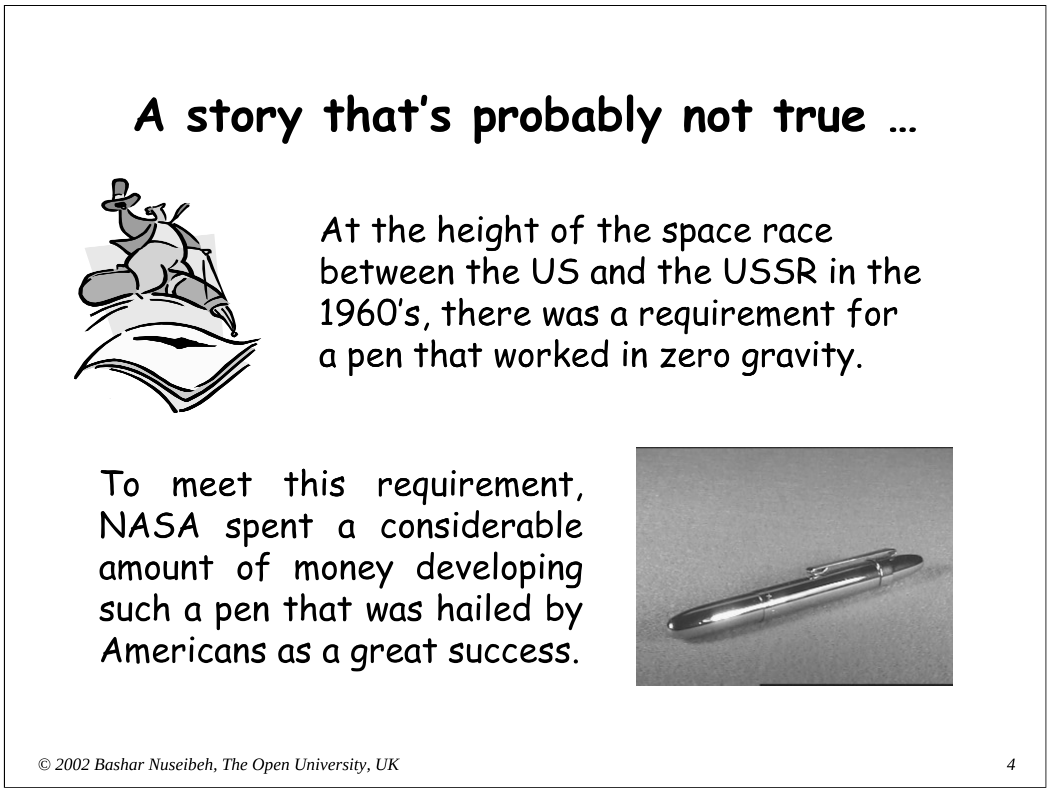
3

Very often, these problems are presented by customers, but as any software engineer will tell you, the “voice of the customer” does not always explain to the listener what the problem really is. Here’s an illustration – a sound clip from a pretend customer – to illustrate what I mean. I have had to edit it because my parents are in the audience!

[BILLY’S SOUND CLIP] – click on graphic to hear the following

“We want this and that.
We demand a share in that and most of that.
*Some of this and ****ing all of that.*
*Less of that and more of this and ****ing plenty of this.*
And another thing, we want it now.
*I want it yesterday, and I want ****ing more tomorrow.*
*And the demands will all be changed then, so ****ing stay awake!”*

What our customer Billy’s problem is, nobody knows …



A story that's probably not true …

At the height of the space race between the US and the USSR in the 1960's, there was a requirement for a pen that worked in zero gravity.

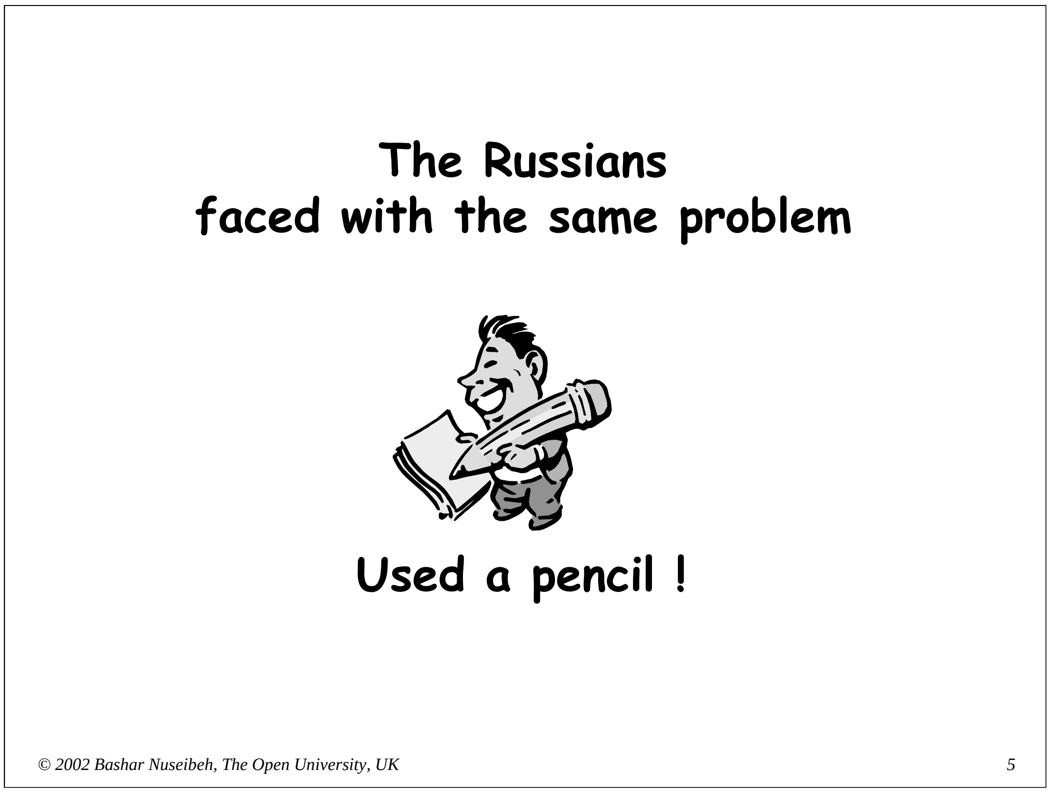
To meet this requirement, NASA spent a considerable amount of money developing such a pen that was hailed by Americans as a great success.

4

Of course, our difficulties are not due to unreasonable or demanding customers only. Even less excitable customers can also lead to difficulties. Take this next story that I think goes some way towards illustrating my point.

Apparently, at the height of the space race between the US and the USSR in the 1960's, there was a requirement for a pen that worked in zero gravity.

To meet this requirement, NASA spent a considerable amount of money developing such a pen that was hailed by Americans as a great success.

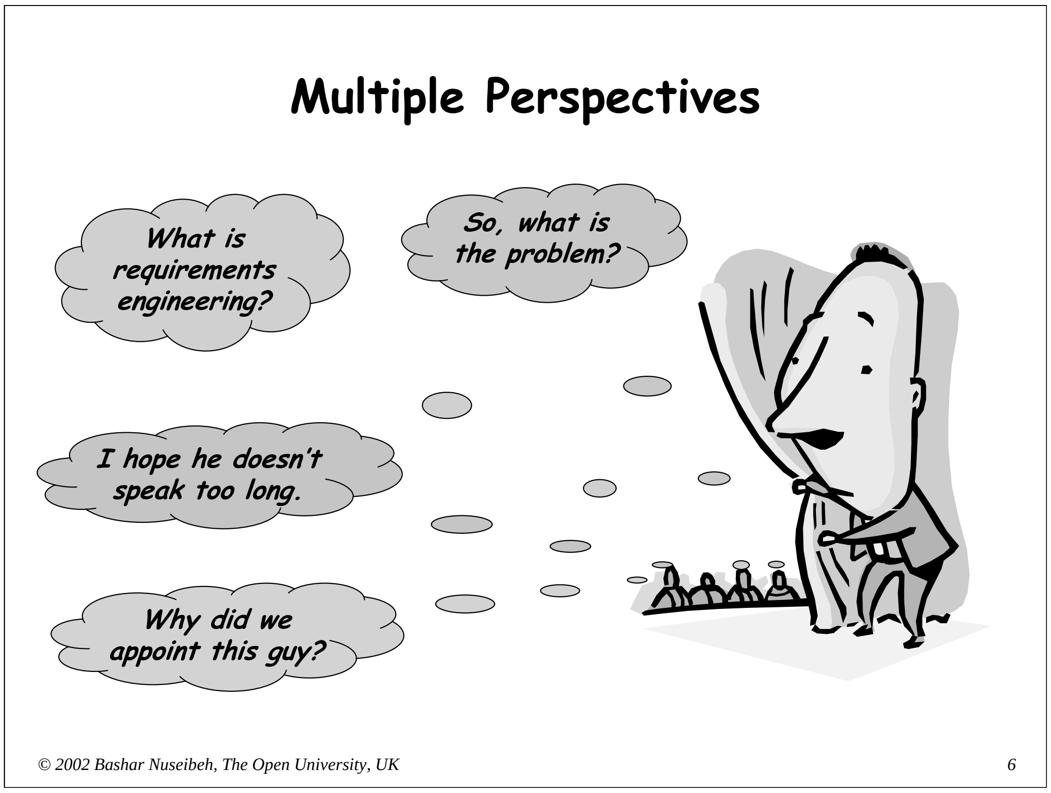


The Russians faced with the same problem

Used a pencil !

The Russians faced with the same problem, used a pencil.

A simply stated requirement is not enough. An understanding of the *real problem*, the *real reason* that the requirement came about is much more important.



Multiple Perspectives

What is requirements engineering?
So, what is the problem?
I hope he doesn't speak too long.
Why did we appoint this guy?

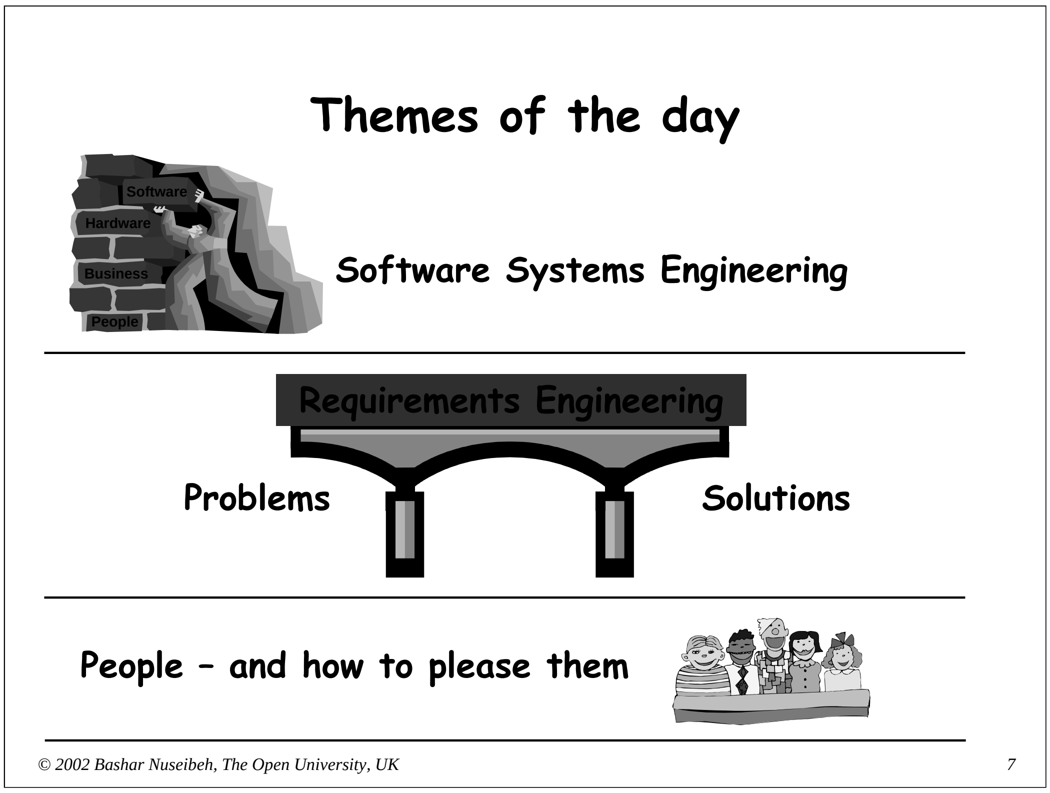
6

Of course – you guessed it – our difficulties don't end here either. Most realistic scenarios involve many people with different requirements, and with different views as to what the problem really is. Take this inaugural lecture for example, which, I have to say, was really very difficult to prepare.

It presents the same kinds of difficulties that I am talking about. Difficulties like: "how do I produce something that meets the different expectations of so many people, each with their own view of what makes a good inaugural lecture?"

I know that some of my colleagues in Computing who have heard me speak before and who know my work well, are probably looking for some new ideas or insights. Other colleagues from outside Computing, perhaps from other parts of the University, may be looking for an overview of my research area and perhaps a vision of what the future holds (an impossible task at the best of times!). Family and friends may be anxious that I don't fumble my words or forget what I planned to say, and hope that perhaps I will throw in a joke or two to make the whole experience bearable!

There are probably others in the audience whose expectations I haven't been able to anticipate – for all I know, they may even be here expecting an answer to the question posed by the title of my lecture.



Themes of the day

Software
Hardware
Business
People

Software Systems Engineering

Requirements Engineering

Problems Solutions

People – and how to please them

© 2002 Bashar Nuseibeh, The Open University, UK 7

This "multiple perspectives problem" has been at the heart of my research activities over the years and I will return to it later. But, before I do that, I would like to outline the three main themes of my lecture today. They are:

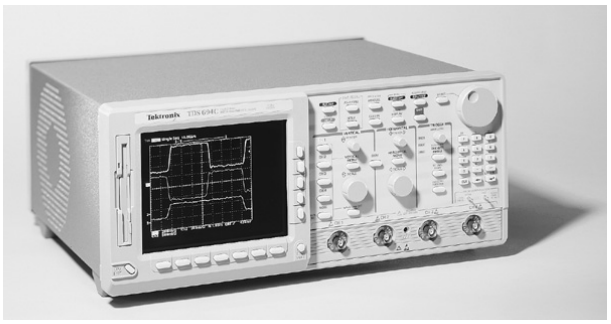
1. Software Systems and their Engineering - the key words being *systems* and *engineering* even more than software… The nature and role of software in systems and the role of engineering in software systems development.

2. Problems, solutions, and the role of *requirements engineering* in linking the two – this will be my main focus.

3. *People* and how to please them - and I don't mean telling good jokes, but rather how discover the different views that people hold about problems, solutions, and software, and how to identify and handle the conflicts between people's views.

Of course, like all inaugurals, this lecture is also about acknowledging the many people whose counsel, support, and friendship have brought me here today.

The Oscilloscope



But let me begin talking about something that for some strange reason keeps cropping up in my life, even though it didn't particularly interest me, I don't really understand how it works, and my knowledge of its uses is fairly limited. It is the electronic instrument, known as the *oscilloscope*.

When I joined the University of Sussex as a Computer Science undergraduate in the School of Engineering in 1985, I soon discovered that what little Computer Science I would learn was not going to be taught till the final year of the degree, and that in fact I would be spending much of my lab time hunched over circuits boards fiddling about with a soldering iron and *an oscilloscope*. The oscilloscope allowed me to poke electronic circuits and find out properties of the various bits of circuit that I had assembled.

Although, up until that point, I hadn't really thought much about the difference between science and engineering, it was still very clear to me that what I was doing was definitely engineering. So, armed with a petition signed by all my fellow students, I managed to persuade the university authorities to change the name of the degree we would get when we graduated from "Computer Science" to "Computer Systems Engineering", as I felt that if I was going to learn to be an engineer then I might as well get a degree that recognised that I had studied to be one!

Of course, by the time I had graduated, I had also decided that I was actually more interested in software not hardware, so I joined the Masters programme at Imperial College, to study for an MSc in the "Foundations of Advanced Information Technology", appropriately abbreviated as F.A.I.T.! I quickly learnt though that the foundations of software and IT were not programming and technology, but mathematics and logic. So another year ended without me experiencing much software development, which is what I had been after.

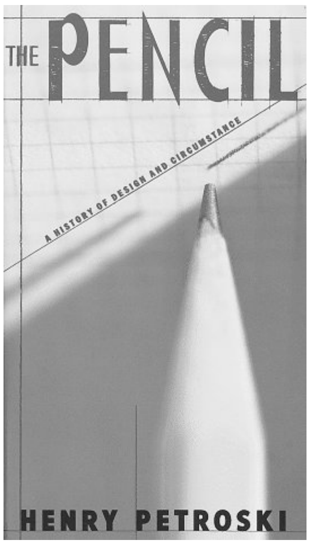
OK, I thought. I'll try my luck in industry. So, I joined a reputable high tech company called Tektronix, who gave me the job title "Sales Engineer". But, after a brief induction, I found myself sitting in front of oscilloscope specifications, and was instructed by my manager to memorise these specifications and to sell as many units of the products as I could – to whomever would buy them! So there I was armed with this hammer – everything looked like a nail! If my customers were teachers, then the Tektronix oscilloscope was their ideal teaching tool. If my customers were engineers, then the Tektronix oscilloscope packed with features – that I could list backwards – was for them! And so on.

Needless to say, I left that job within a year, when Anthony Finkelstein, who was to become my PhD supervisor offered me a Software Engineering Research Assistant job at Imperial College – on an interdisciplinary project, he told me, with City University.

To cut a long story short, it turned out that this project was a joint enterprise between the Department of Computing at Imperial College and the Electrical Engineering Department at City University. And, it wasn't too long before I found myself studying oscilloscope specifications again; this time to investigate the relationships between software engineering and electronic instrument/engineering design! There was no escape from the oscilloscope!

Of course, modern oscilloscopes are fascinating and highly complex systems. They are composed of hardware and software, and they have a range of users working in potentially very different environments, from warm physics labs to wet and windy oilrigs in the middle of the ocean. Yes I was interested in software engineering, but it was becoming blatantly obvious to me that, very often, I couldn't just work on the software without understanding the wider context, or *system*, in which it would be introduced and used.

The Pencil



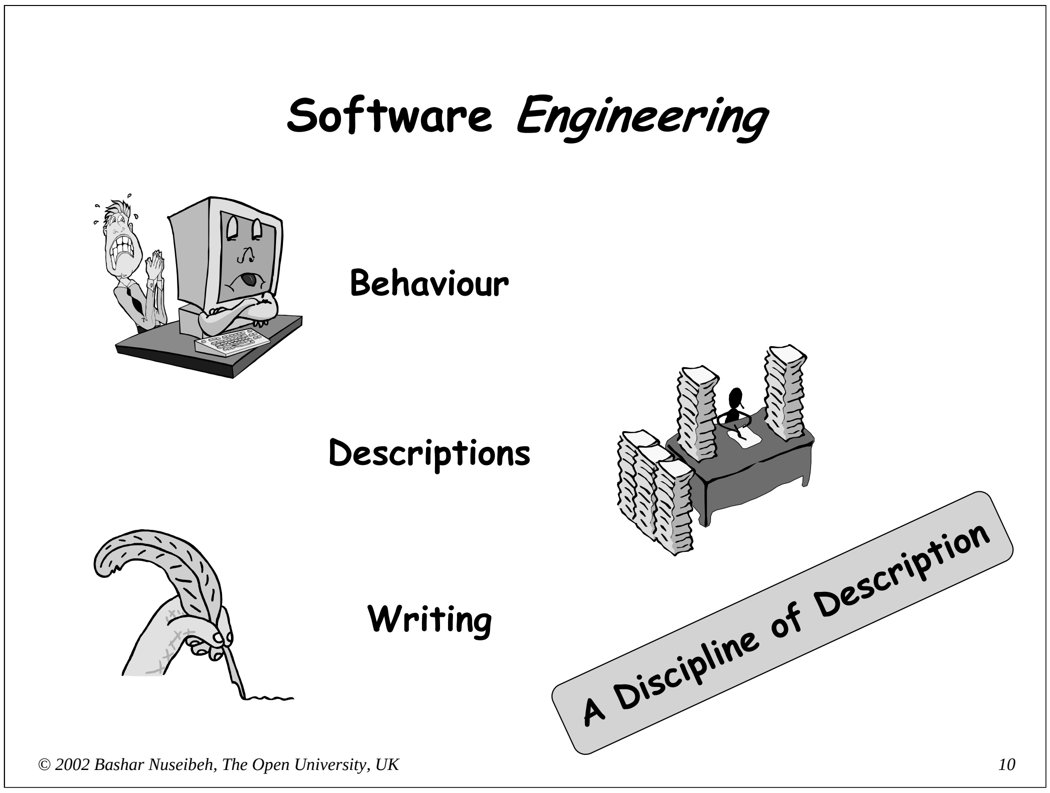
© 2002 Bashar Nuseibeh, The Open University, UK 9

Of course, the oscilloscope is far too complex a system upon which to base this talk, so I'd like to return to the simple pencil.

But is a pencil really all that simple?

In 1989, the famous author and civil engineer, Henry Petroksi, wrote a 430-page book entitled "The Pencil: A History of Design and Circumstance", in which he comments on the rich history of pencil design, and the various trade-offs engineers engaged in, in order to produce a pencil that leaves a mark on the paper that it touches, that glides comfortably on paper without tearing it, and does not poison the writer who decides to chew its top!

So, even the simplest-looking systems can actually be very complex to develop, and their design can be a complex trade-off between many requirements and design possibilities.



Software *Engineering*

Behaviour

Descriptions

Writing

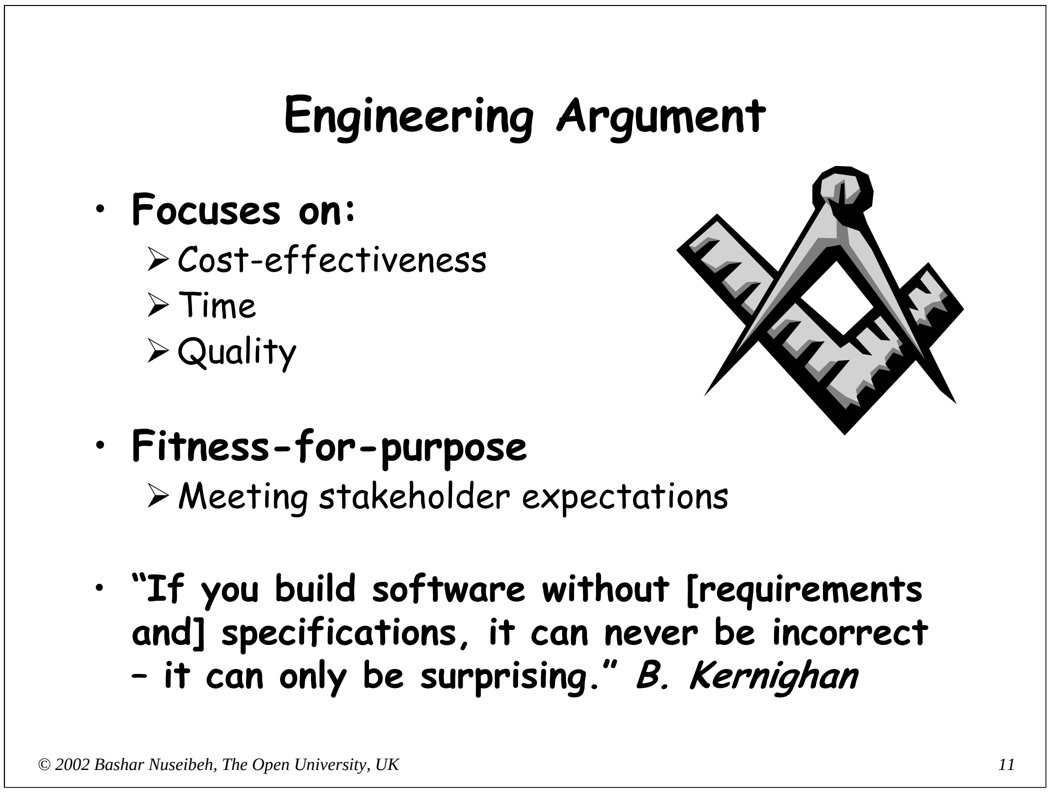
A Discipline of Description

10

But what has this got to do with software and its engineering design? After all, software is not tangible, it needs no engineering materials, and - hardware aside - it requires no manufacturing, a process so prominent in other engineering disciplines. Well, I would argue that this has everything to do with software. The tangible part of software is its *behaviour*, the engineering materials it uses are *descriptions*, and the manufacturing process is, I would also argue, one of the most demanding processes of all: creative and technical *writing*.

Software engineers develop descriptions of different kinds. These include descriptions of customer requirements often written in natural language – such as English – sentences, descriptions of software designs – such as structured diagrams with boxes and arrows, programs that execute on computers often written in the programmer's favourite programming language, and large volumes of documentation for all kinds of people to read.

So, as Jackson has often pointed out, Software Engineering is a "discipline of description".



Engineering Argument

- **Focuses on:**
  - Cost-effectiveness
  - Time
  - Quality
- **Fitness-for-purpose**
  - Meeting stakeholder expectations
- **"If you build software without [requirements and] specifications, it can never be incorrect – it can only be surprising."** ***B. Kernighan***

© 2002 Bashar Nuseibeh, The Open University, UK *11*

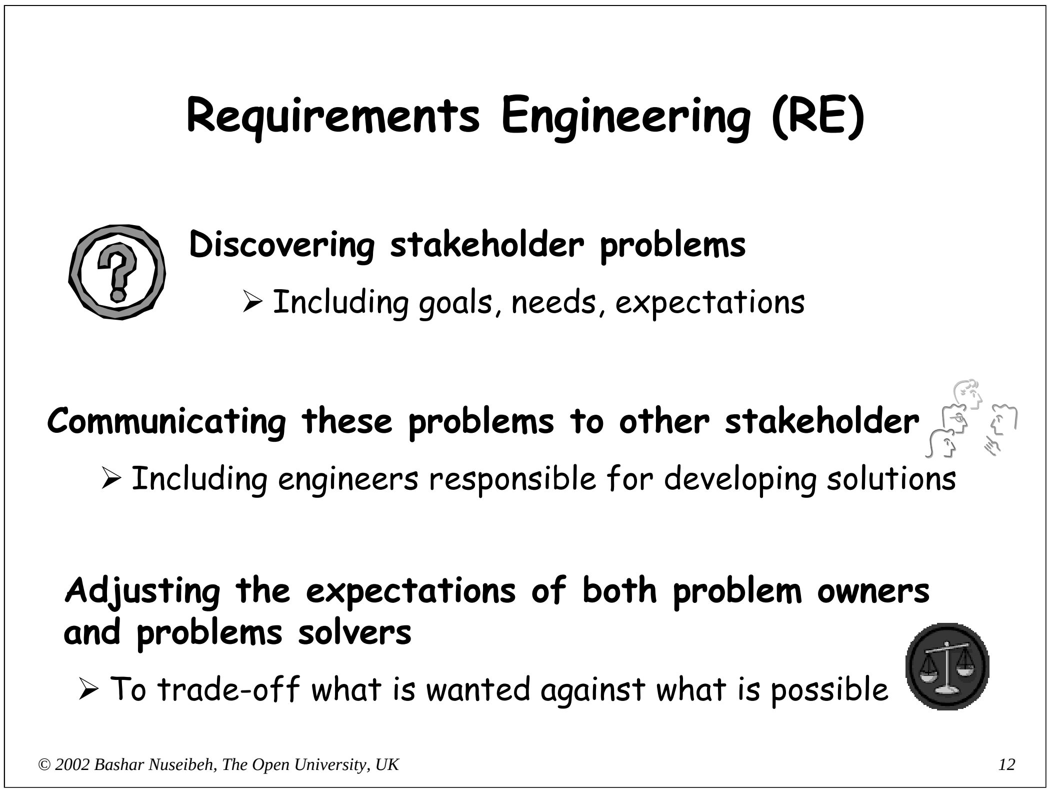
But why does all this description-generation activity make software development an engineering discipline? Well, I'd like to argue that what makes it engineering is its considerations of cost-effectiveness, time, and quality.

An engineer is often described as someone who can build something for £1 that anyone can build for £10. Software engineering is about bringing these kinds of considerations to the development of software.

This *engineering argument* for the need to treat software development as an engineering discipline (rather than a science, an art, or a branch of mathematics), I find very convincing, and has shaped my view of Computing over the years. It is an argument that focuses on the *quality* of the software product and its fitness for purpose. I also believe that the ultimate arbiter of quality and fitness for purpose is the degree to which the product meets stakeholder expectations – or requirements. Indeed the celebrated computer scientist Brian Kernighan is quoted as saying:

"If you build software without [requirements and] specifications, it can never be incorrect – it can only be surprising."

Without knowing what a customer expects, one cannot hope to meet their expectations.



Requirements Engineering (RE)

Discovering stakeholder problems

- ➢ Including goals, needs, expectations

Communicating these problems to other stakeholder

- ➢ Including engineers responsible for developing solutions

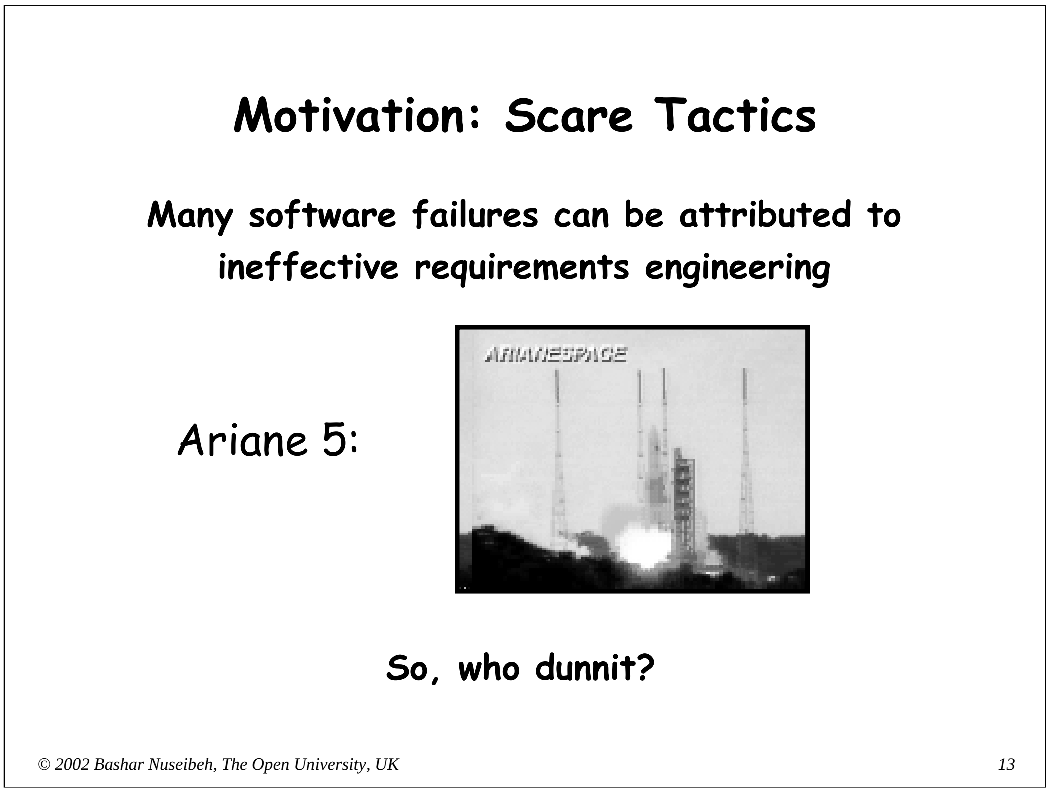
Adjusting the expectations of both problem owners and problems solvers

- ➢ To trade-off what is wanted against what is possible

12

I think it is worth giving an informal definition of Requirements Engineering (RE) at this point. RE is about *discovering the problems* that stakeholders have, *communicating these problems* to other stakeholders, including the engineers responsible for developing solutions to these problems, and *adjusting the expectations* of both problem owners and problem solvers about what is wanted and what is possible.

While all this may sound sensible and uncontroversial, the state of the software industry suggests that many people remain unconvinced. Large volumes of software are written without any explicit RE being performed, despite many studies that show that early and effective RE can save money and effort later on during the software development life cycle.



Motivation: Scare Tactics

Many software failures can be attributed to ineffective requirements engineering

Ariane 5:

So, who dunnit?

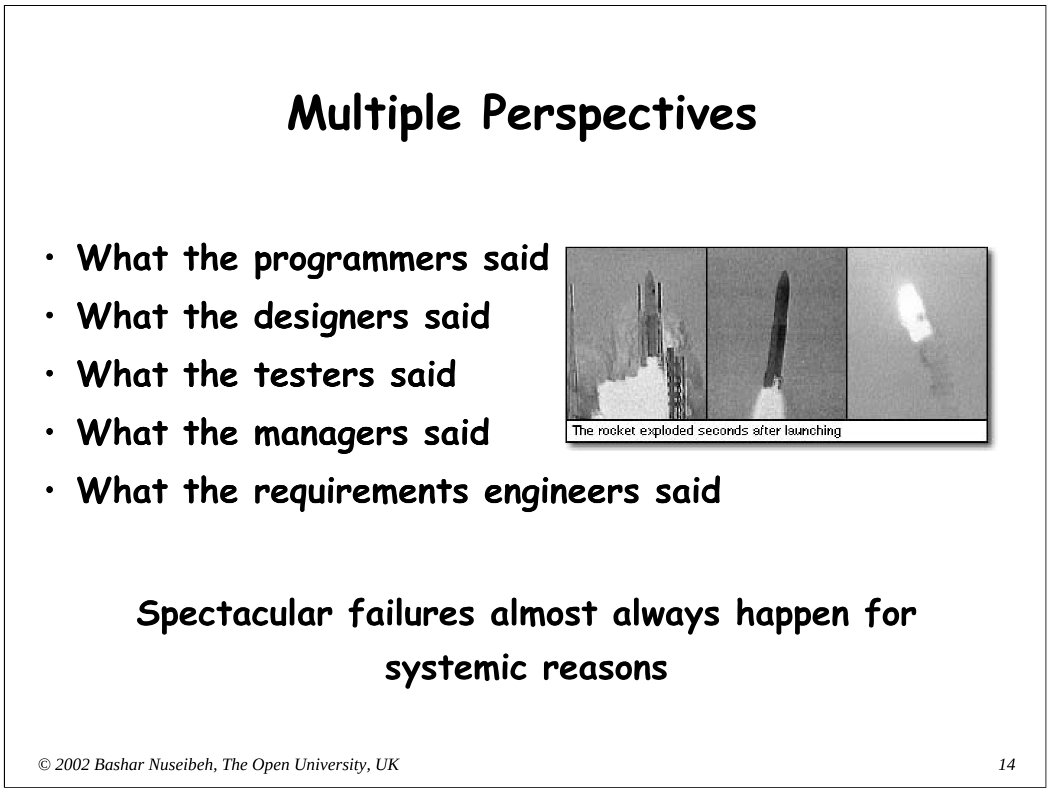
© 2002 Bashar Nuseibeh, The Open University, UK *13*

Many studies also identify inadequate RE as a major contributor to software systems failure, which in turn can cost money and sometimes lives. Indeed there has been no shortage of software systems failures that have been used to criticise software engineering practice and to motivate further research. A particularly 'popular' failure story, that some of you will have heard of, is the catastrophic failure of the European Space Agency's Ariane 5 rocket in 1996, which exploded about 37 seconds after take-off, destroying a full payload of scientific experiments (many of which were, unfortunately uninsured).

This example is particularly interesting from a research perspective, because although a "software error" was quickly blamed for the failure, actually a number of other factors were in play. Here's what happened.

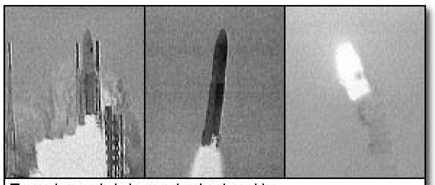
[DESCRIPTION OF WHAT HAPPENED TO ARIANE 5, WITH VIDEO CLIP] – click on picture to view video clip. Reference: http://www.esa.int/export/esaCP/Pr_33_1996_p_EN.html

So, who Dunnit?



Multiple Perspectives

- What the programmers said
- What the designers said
- What the testers said
- What the managers said
- What the requirements engineers said



The rocket exploded seconds after launching

Spectacular failures almost always happen for systemic reasons

© 2002 Bashar Nuseibeh, The Open University, UK *14*

Abbreviated description:

What the programmers said: the problem was clearly due to a programming error (unhandled exception) that could have been avoided by having better programming practice.

What the designers said: the problem was clearly a design error (treating software failure/redundancy the same as hardware failure/redundancy) that could have been mitigated with a better design.

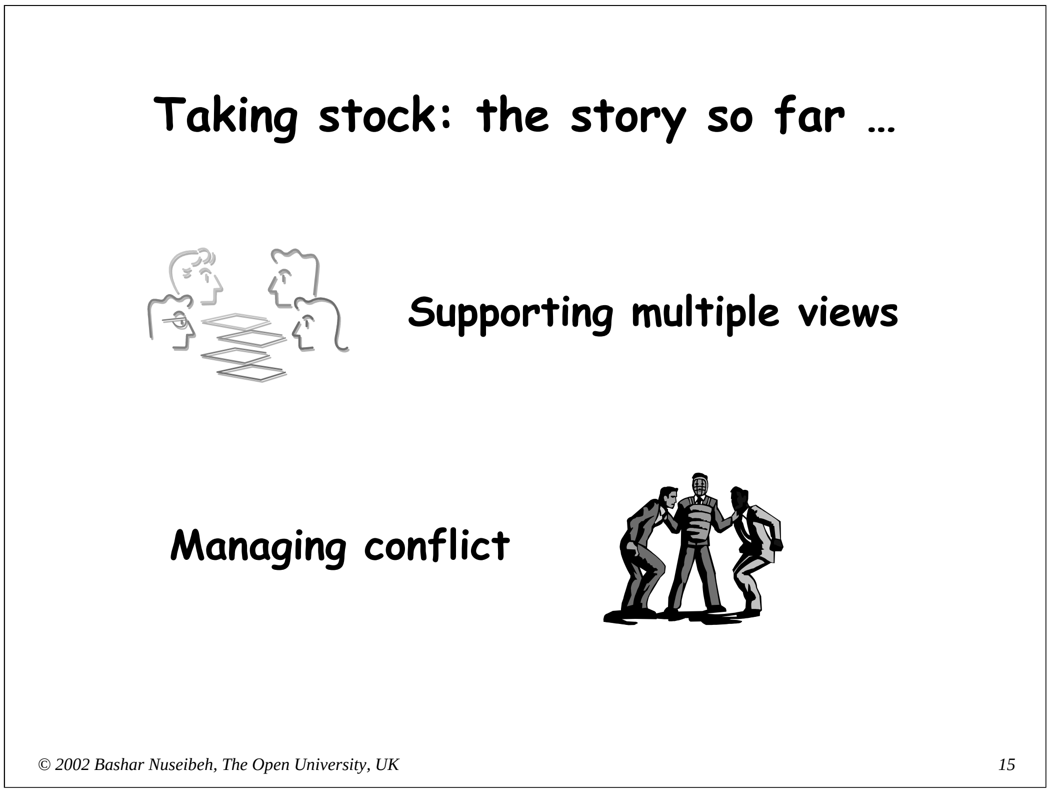
What the testers said: the problem was clearly due to inadequate testing that could have been addressed by choosing cases that tested the system under realistic launch conditions.

What the managers said: the problem was a management one that could have been addressed through better project reviews and schedules.

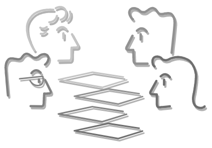
What the requirements engineers said – and I count myself as one – the problem was clearly due to the reuse of the same software when the requirements of Ariane 5 had changed from those of Ariane 4.

In reality, of course, the Ariane 5 disaster was a systemic failure, with the lack of re-evaluation of risk playing a significant part in the failure.

Reference: B. Nuseibeh, “Ariane 5: Who Dunnit?”, *IEEE Software,* 14(3): 15-16, May 1997, IEEE CS Press .



Taking stock: the story so far …



Supporting multiple views

Managing conflict

© 2002 Bashar Nuseibeh, The Open University, UK 15

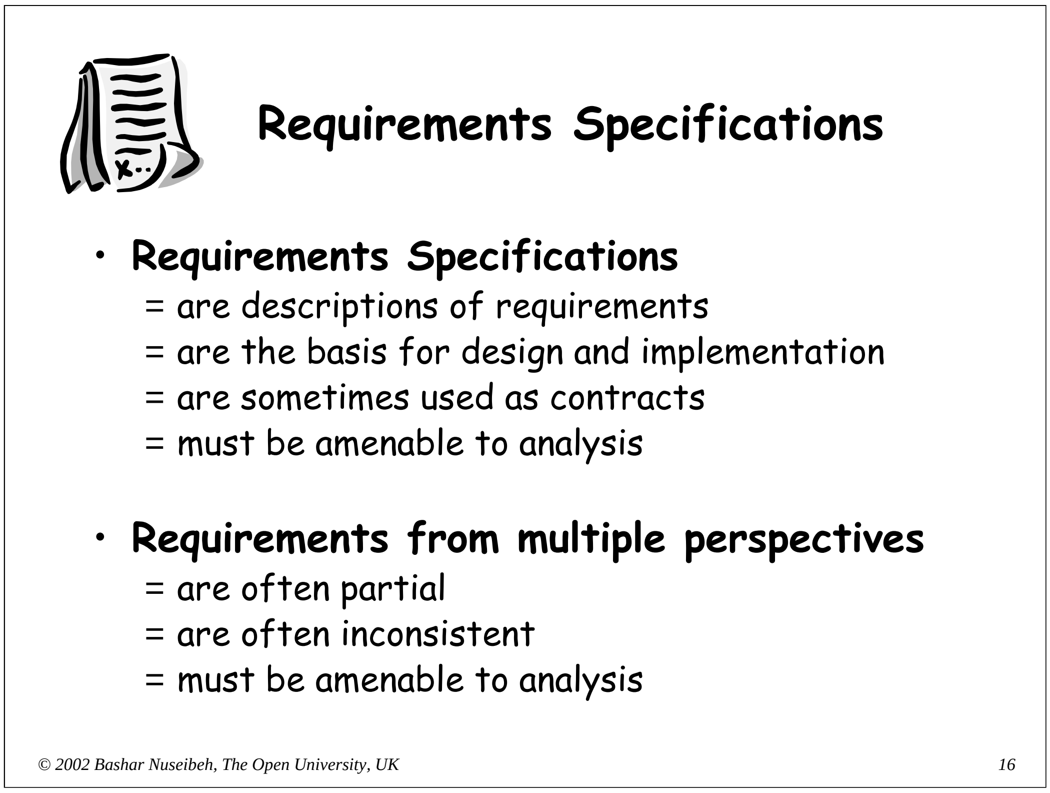
What I have tried to motivate so far is the crucial role of requirements engineering in effective software development. The last ten years have seen the growth and establishment of the discipline of software requirements engineering. It is a discipline that draws on other engineering disciplines in its motivation to build cost-effective software products. It comprises many activities that time constraints today don't allow me to elaborate. Activities like requirements elicitation, modelling, analysis and negotiation. It is also multi-disciplinary in its foundations, borrowing techniques from sociology, cognitive psychology, systems engineering and linguistics, as well as its more traditional base in computer science and mathematical logic.

Like most researchers, my own work in the field has focused on some very specific sub-areas that I have already alluded to in passing:

1. Support for multiple views of requirements, held by multiple stakeholders; and

2. Analysis of requirements specifications to discover and handle inconsistencies and conflicts between them.

While there are many social, organisational, and other non-technical issues surrounding these two research areas, my own research focus has been on the technical manifestation of these issues. In requirements engineering, this technical manifestation normally comes in the shape of so-called "requirements specifications".



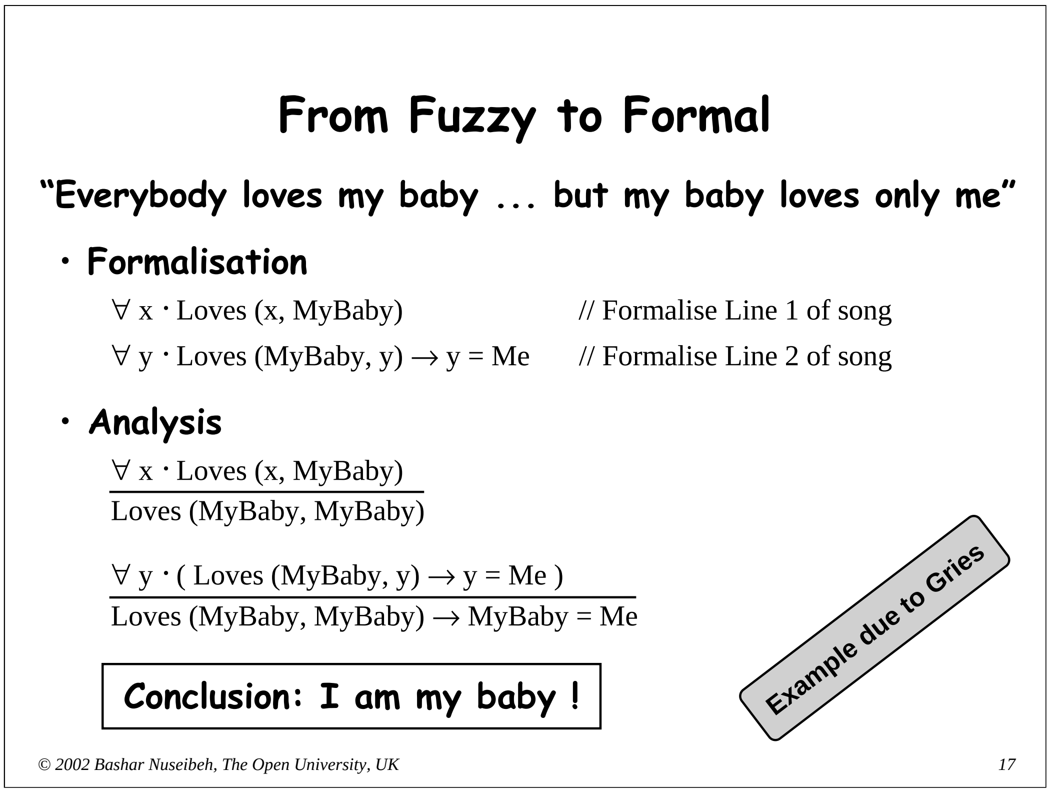
Requirements Specifications

- **Requirements Specifications**
  - = are descriptions of requirements
  - = are the basis for design and implementation
  - = are sometimes used as contracts
  - = must be amenable to analysis
- **Requirements from multiple perspectives**
  - = are often partial
  - = are often inconsistent
  - = must be amenable to analysis

16

Requirements Specifications are descriptions of requirements that form the basis for design and implementation, and are sometimes used as contracts. Ideally, they should be amenable to analysis

Additionally, requirements/specifications from multiple perspectives are often partial and inconsistent, but must also, ideally, be amenable to analysis.



From Fuzzy to Formal

"Everybody loves my baby ... but my baby loves only me"

- **Formalisation**

  $\forall x \cdot \text{Loves}(x, \text{MyBaby})$ // Formalise Line 1 of song

  $\forall y \cdot \text{Loves}(\text{MyBaby}, y) \rightarrow y = \text{Me}$ // Formalise Line 2 of song

- **Analysis**

$$\frac{\forall x \cdot \text{Loves}(x, \text{MyBaby})}{\text{Loves}(\text{MyBaby}, \text{MyBaby})}$$

$$\frac{\forall y \cdot (\text{Loves}(\text{MyBaby}, y) \rightarrow y = \text{Me})}{\text{Loves}(\text{MyBaby}, \text{MyBaby}) \rightarrow \text{MyBaby} = \text{Me}}$$

Conclusion: I am my baby !

Example due to Gries

© 2002 Bashar Nuseibeh, The Open University, UK *17*

But in order to analyse requirements specifications, it is important to take informal statements of requirements, often expressed as wishes, wants, and desires of stakeholders, and to write these down more formally, often using the language of mathematics and logic, to make them amenable to automated analysis.

I promise that this will be the only slide that will contain any mathematical symbols, but I have included it for those in the audience who find this sort of thing readable and interesting! Those who don't enjoy the maths may be amused by the results of the analysis of the example anyway.

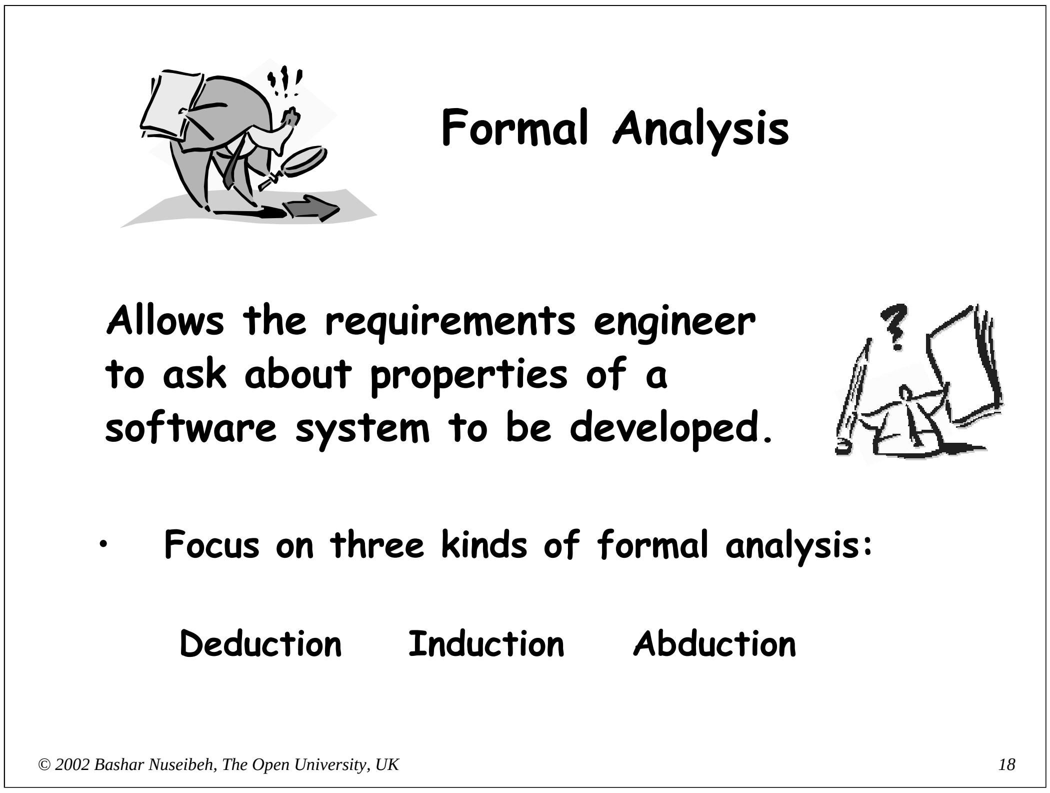
The example is taken (and adapted) from the lyrics of a well-known song: "Everybody loves my baby", and goes something like:

"Everybody loves my baby … but my baby loves only me."

Those who have seen my wife, Juliet, today, will know that I have babies on my mind!

GRIES EXAMPLE: formalisation, followed by analysis (quantifier elimination).

The example illustrates that, through a process of formalising informal statements, one can discover interesting properties, ambiguities, and possible problems with the statements. This kind of formal analysis can be very useful if the engineer is presented with more serious requirements from stakeholders.



Formal Analysis

Allows the requirements engineer to ask about properties of a software system to be developed.

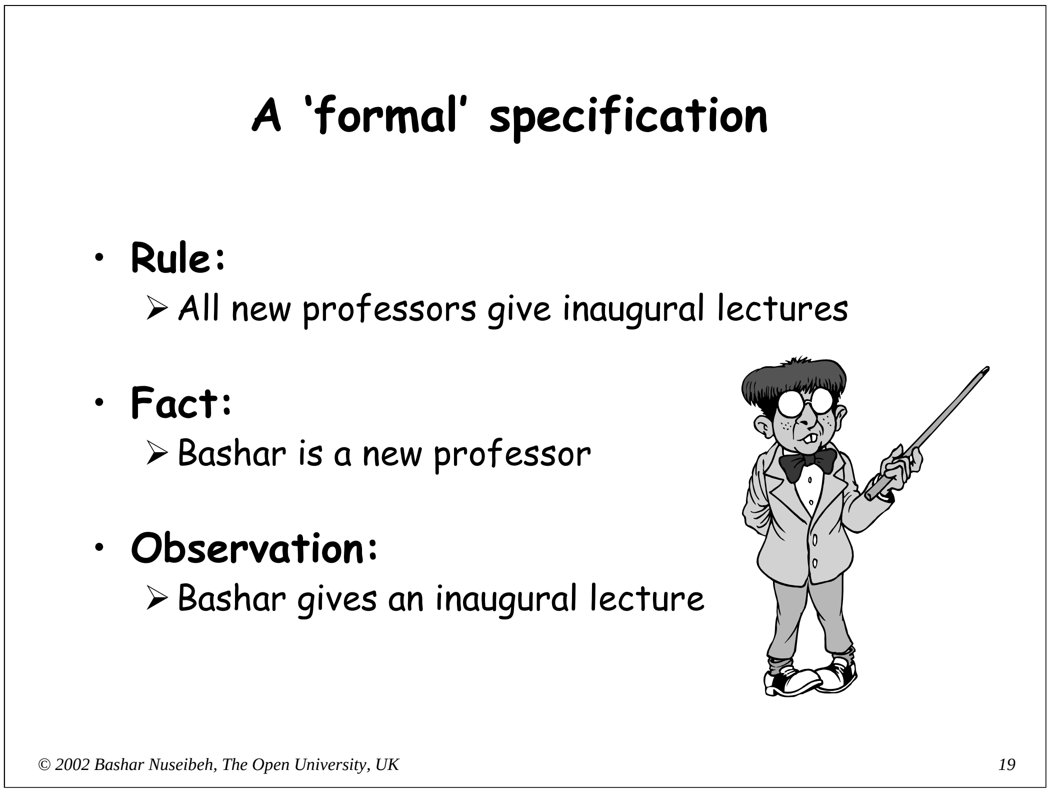
- **Focus on three kinds of formal analysis:**

Deduction Induction Abduction

18

So, what is formal analysis and what does it mean in the context of requirements engineering? Well, formal analysis of requirements specifications allows the analyst (whom I call here a "requirements engineer") to ask questions about the properties of a software system that some stakeholders want to be developed. This is analogous to a *civil engineer* being able to ask questions about the structural properties of a building, or an *electrical engineer* being able to ask questions about the properties of an electronic circuit, or an *aeronautical engineer* is able to ask questions about he properties of an aircraft wing.

Some of my research work has examined three kinds of formal analysis techniques – *Deduction, Induction, and Abduction* – and I would like to illustrate each of these in turn with an example.



A ‘formal’ specification

- **Rule:**
  - All new professors give inaugural lectures
- **Fact:**
  - Bashar is a new professor
- **Observation:**
  - Bashar gives an inaugural lecture

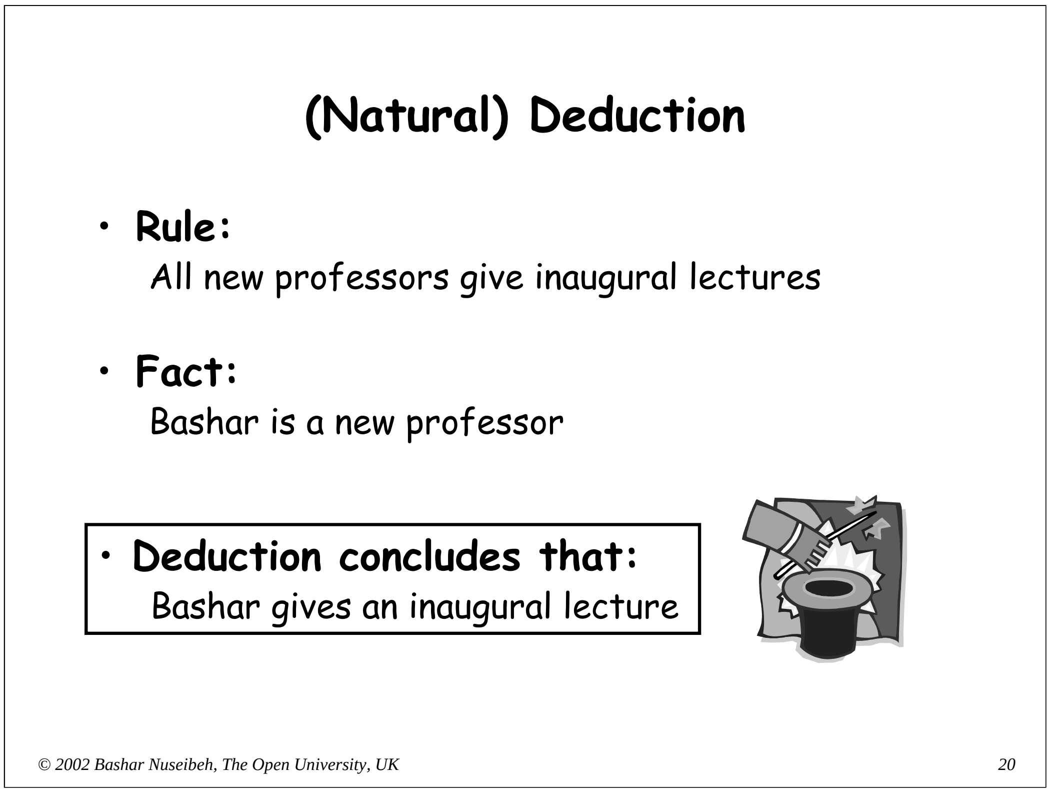
19

Let me begin with a (formal) specification containing the following three kinds of information (statements):

1. **Rule**: All new professors give inaugural lectures

2. **Fact:** Bashar is a new professor

3. **Observation:** Bashar gives an inaugural lecture

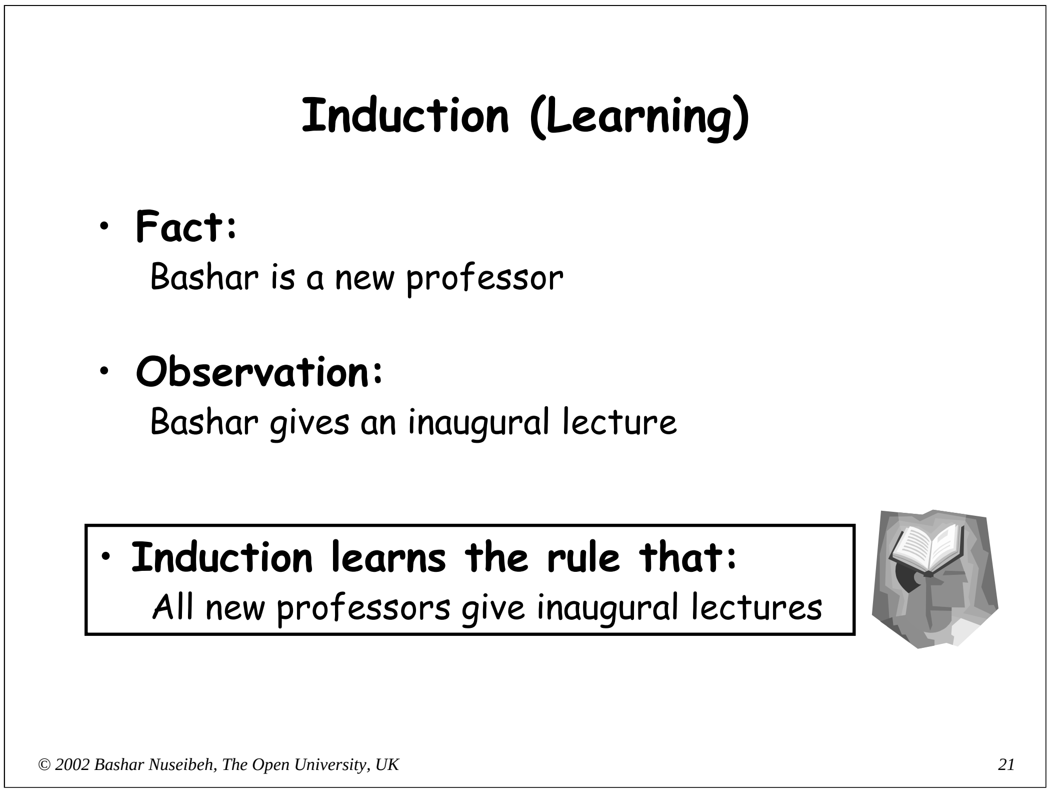


(Natural) Deduction

- **Rule:**
  All new professors give inaugural lectures
- **Fact:**
  Bashar is a new professor
- **Deduction concludes that:**
  Bashar gives an inaugural lecture

20

Given the above rule and fact, natural *deduction* allows us to conclude that: “Bashar gives an inaugural lecture”.



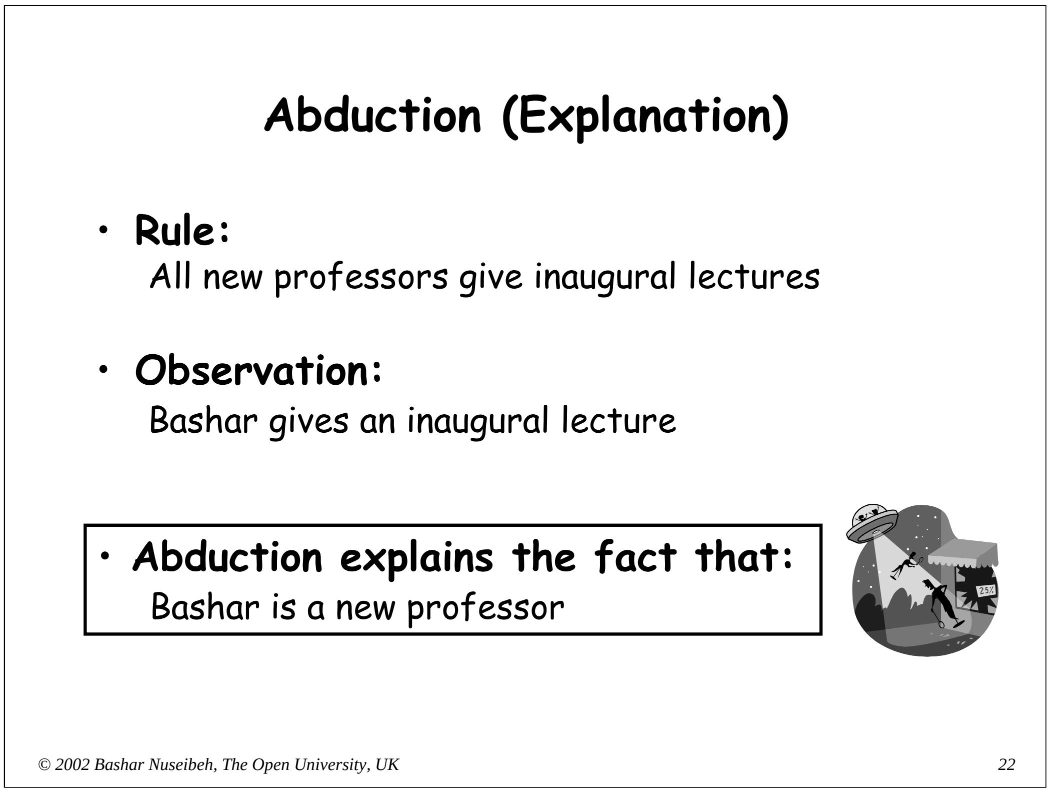
Induction (Learning)

- **Fact:**
  Bashar is a new professor
- **Observation:**
  Bashar gives an inaugural lecture
- **Induction learns the rule that:**
  All new professors give inaugural lectures

© 2002 Bashar Nuseibeh, The Open University, UK 21

Given the above fact and observation, *induction* infers (learns) the rule that “All new professors give inaugural lecture”.

Unlike natural deduction, the result of induction can be wrong, if new, more up-to-date information comes along (such as new facts or observations).



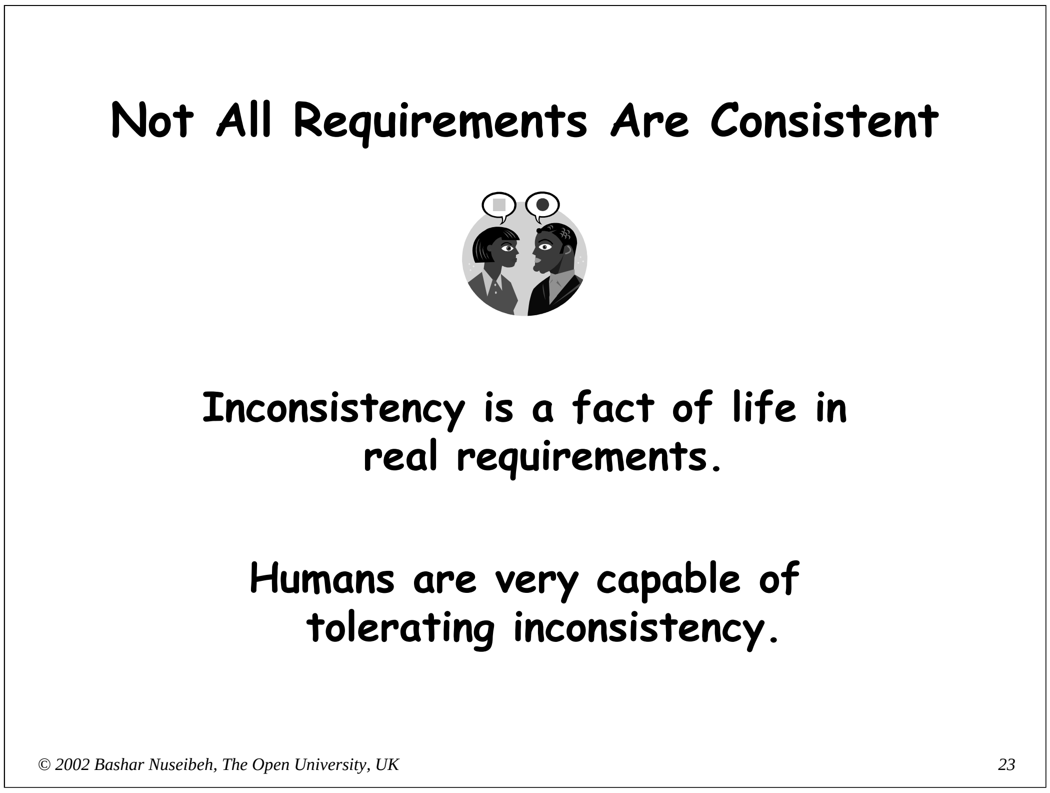
Abduction (Explanation)

- **Rule:**
  All new professors give inaugural lectures
- **Observation:**
  Bashar gives an inaugural lecture
- **Abduction explains the fact that:**
  Bashar is a new professor

22

Given the above rule and observation, *abduction* provides an explanation that: “Bashar is a new professor”.

This explanation offered by abduction may be only one of a number of possible explanations for the observation, and may not be the best explanation.



Not All Requirements Are Consistent

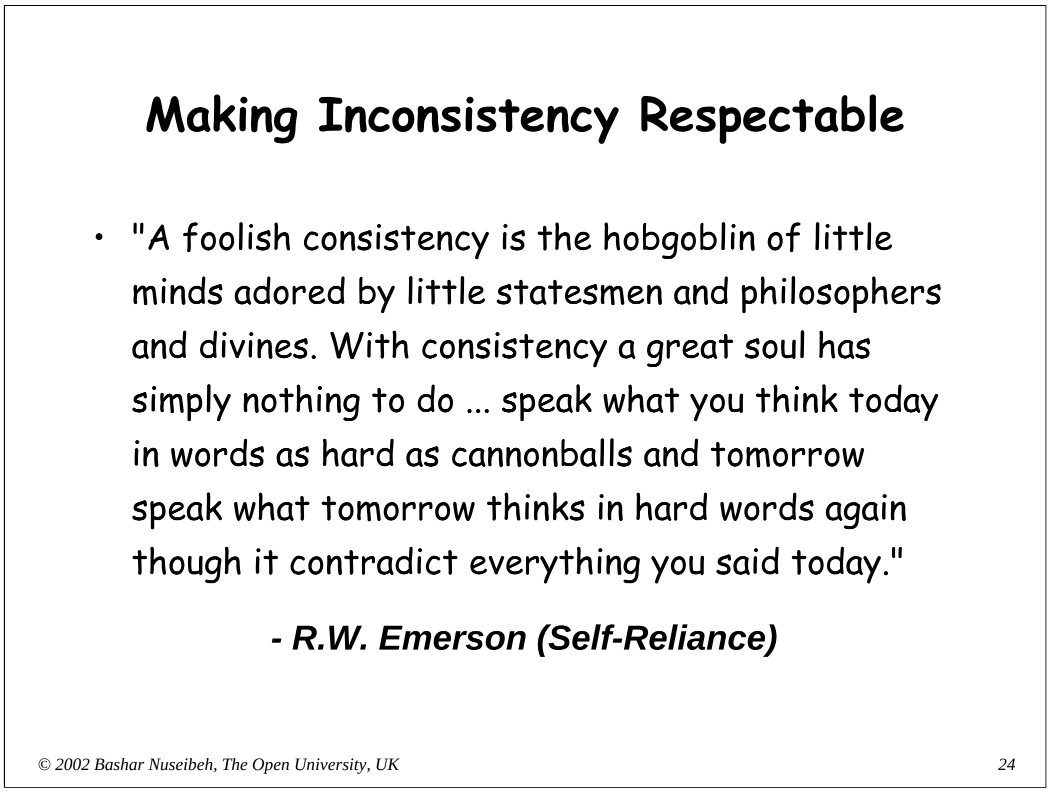
Inconsistency is a fact of life in real requirements.

Humans are very capable of tolerating inconsistency.

23

Unfortunately, all these formal analysis techniques assume that, and only work properly if, the information that is presented to them is *consistent*, which is often not the case in real life. Inconsistency is a fact of life in many real requirements, and humans are actually very capable of tolerating such inconsistency, even if they prefer consistency.

Despite this reality, the research literature is full of examples of formal analysis techniques for detecting inconsistencies in requirements and specifications, and for eradicating them as soon as they are discovered.



Making Inconsistency Respectable

- "A foolish consistency is the hobgoblin of little minds adored by little statesmen and philosophers and divines. With consistency a great soul has simply nothing to do ... speak what you think today in words as hard as cannonballs and tomorrow speak what tomorrow thinks in hard words again though it contradict everything you said today."

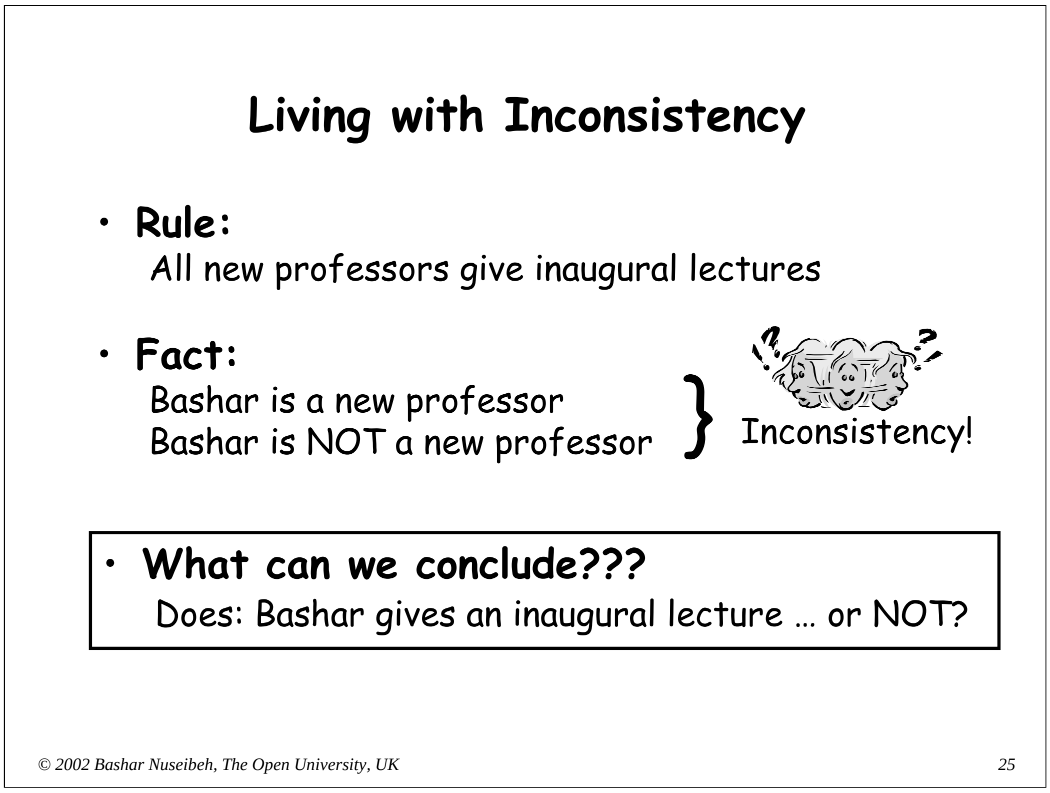
- R.W. Emerson (Self-Reliance)

© 2002 Bashar Nuseibeh, The Open University, UK 24

The focus of research of my colleagues and I in recent years has been on “making inconsistency respectable” – a phrase coined by Gabbay & Hunter – and trying to find ways in which software engineers can work around it, or even make use of it.

Philosophers, of course, have long accepted the need to live with inconsistency, and the impossibility of eradicating inconsistency. Emerson, for example, says:

"A foolish consistency is the hobgoblin of little minds adored by little statesmen and philosophers and divines. With consistency a great soul has simply nothing to do ... speak what you think today in words as hard as cannonballs and tomorrow speak what tomorrow thinks in hard words again though it contradict everything you said today."



Living with Inconsistency

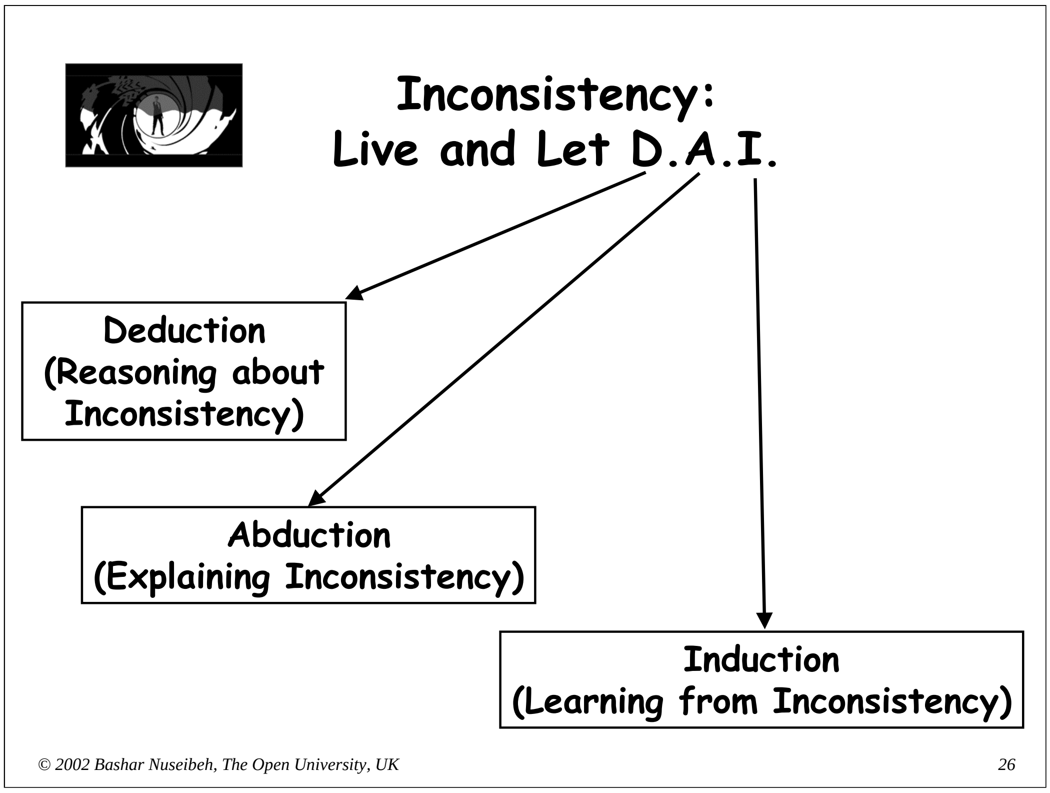
- **Rule:**
  All new professors give inaugural lectures
- **Fact:**
  Bashar is a new professor
  Bashar is NOT a new professor } Inconsistency!
- **What can we conclude???**
  Does: Bashar gives an inaugural lecture … or NOT?

25

Let me return to my simple inaugural lecture example to give you a flavour of what I mean when I talk about “making inconsistency respectable” or “living with inconsistency”. In that example, what can we conclude if we have two contradictory facts in our specification, like:

“Bashar is new professor” and “Bashar is *not* a new professor”?

Should Bashar give an inaugural lecture, or not?



Inconsistency: Live and Let D.A.I.

Deduction (Reasoning about Inconsistency)

Abduction (Explaining Inconsistency)

Induction (Learning from Inconsistency)

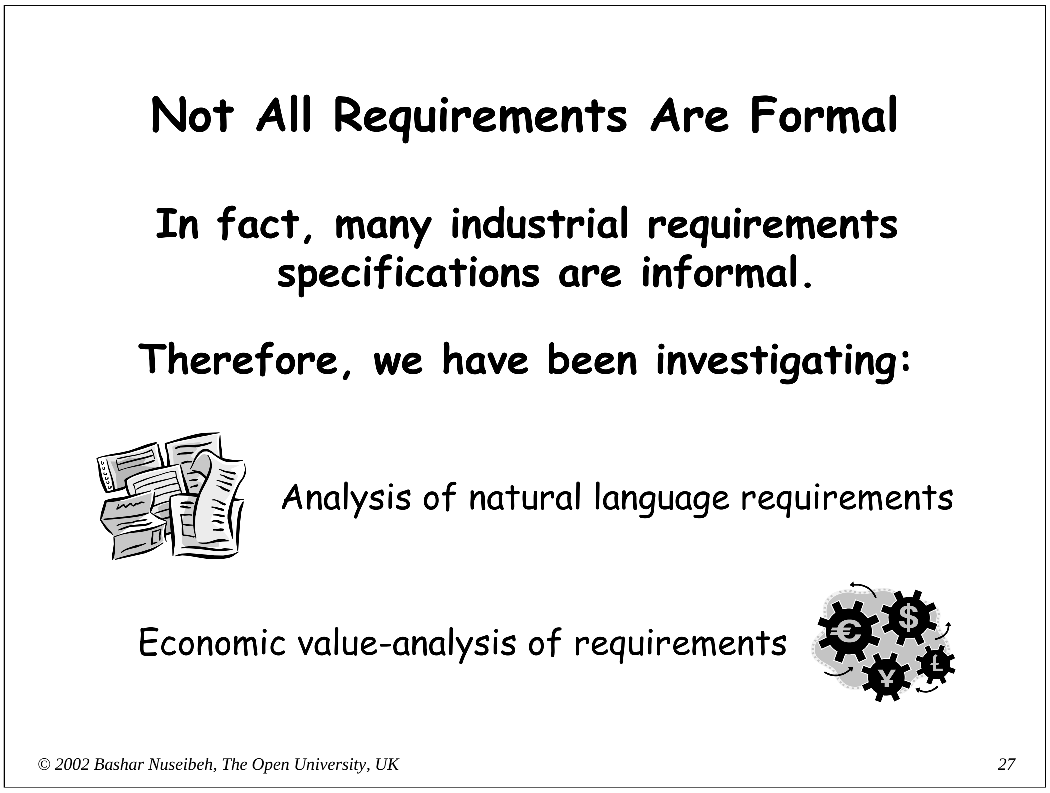
26

We have been looking at ways to perform the formal analysis techniques of deduction, abduction and induction, in the presence of inconsistencies in requirements specifications.

Inconsistency: Live and Let D.A.I.

D = Deduction: Reasoning with inconsistency
A = Abduction: Explaining inconsistency
I = Induction: Learning from inconsistency

Our goal has been to develop a collection of techniques that assist and guide requirements engineers to analyse and understand the requirements, consistent or otherwise, given to them by different stakeholders. The kind of information that our analysis produces can be very useful for prioritising requirements, for conflict resolution, and for informing design.



Not All Requirements Are Formal

In fact, many industrial requirements specifications are informal.

Therefore, we have been investigating:

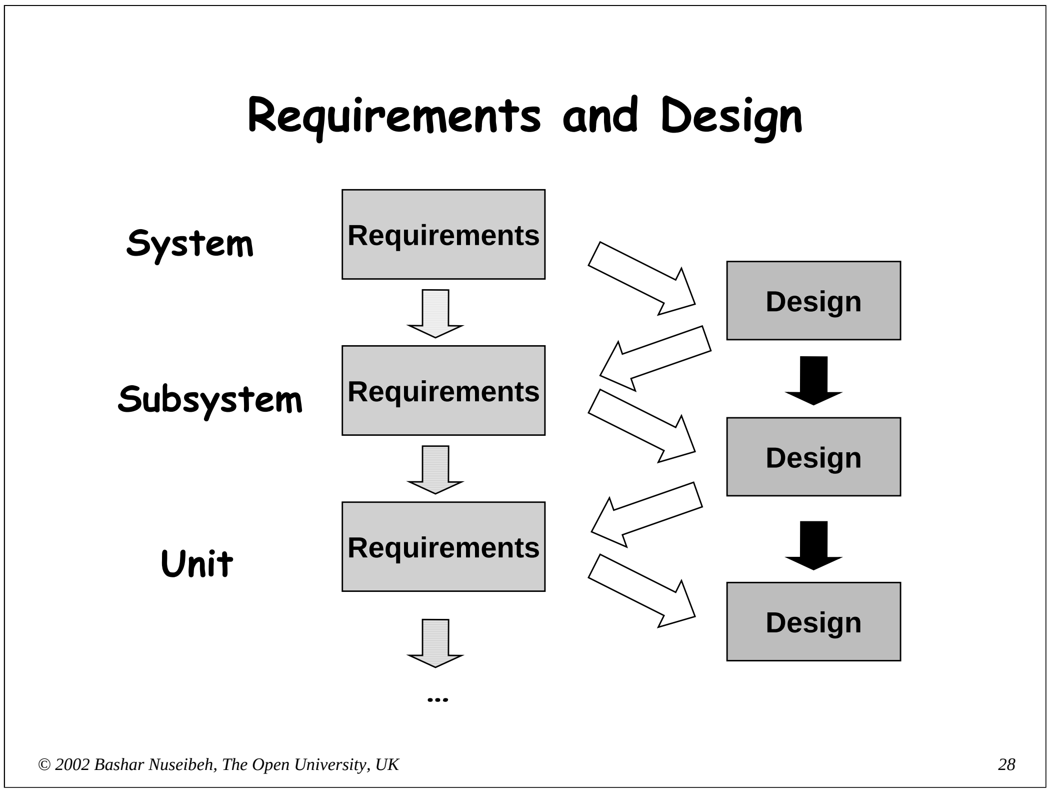
Analysis of natural language requirements

Economic value-analysis of requirements

Of course, not all requirements are formal, as many of our industrial research partners have shown us. So, my colleagues and I have also been looking at a variety of other ways in which to analyse informal requirements. These include:

1. Analysis of natural language (NL) requirements – which are the most common kinds of requirements. We have been trying to find ways of automatically searching for inconsistencies in NL requirements, despite their informality, by exploiting their structure and domain concepts..

2. Economic value-analysis of requirements in order to prioritise them based on 'market' value.

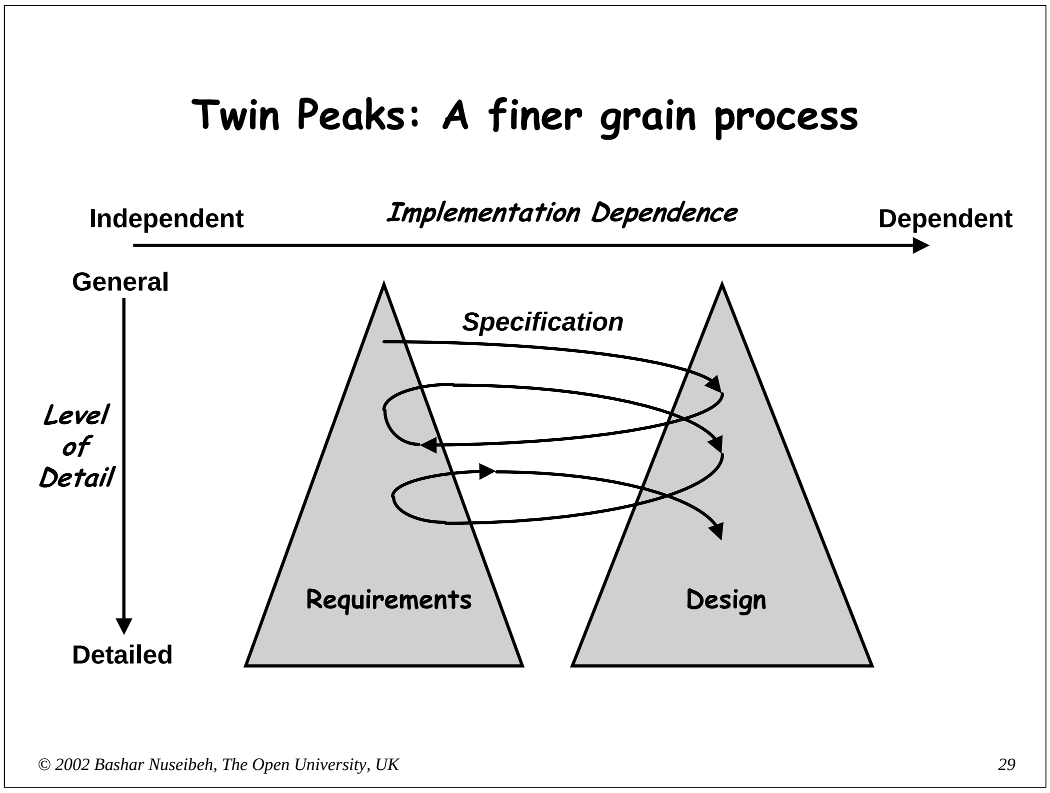


Requirements and Design

© 2002 Bashar Nuseibeh, The Open University, UK 28

Since joining the OU, my colleagues and I in the Computing Department have also been looking different ways in which to perform requirements engineering. Traditionally, the development process has been largely portrayed as sequential, with, at best, one level of development informing the next.

This does not reflect the much more complex and real process practiced in industry in which the activities of requirements and design specification are much more closely intertwined.



Twin Peaks: A finer grain process

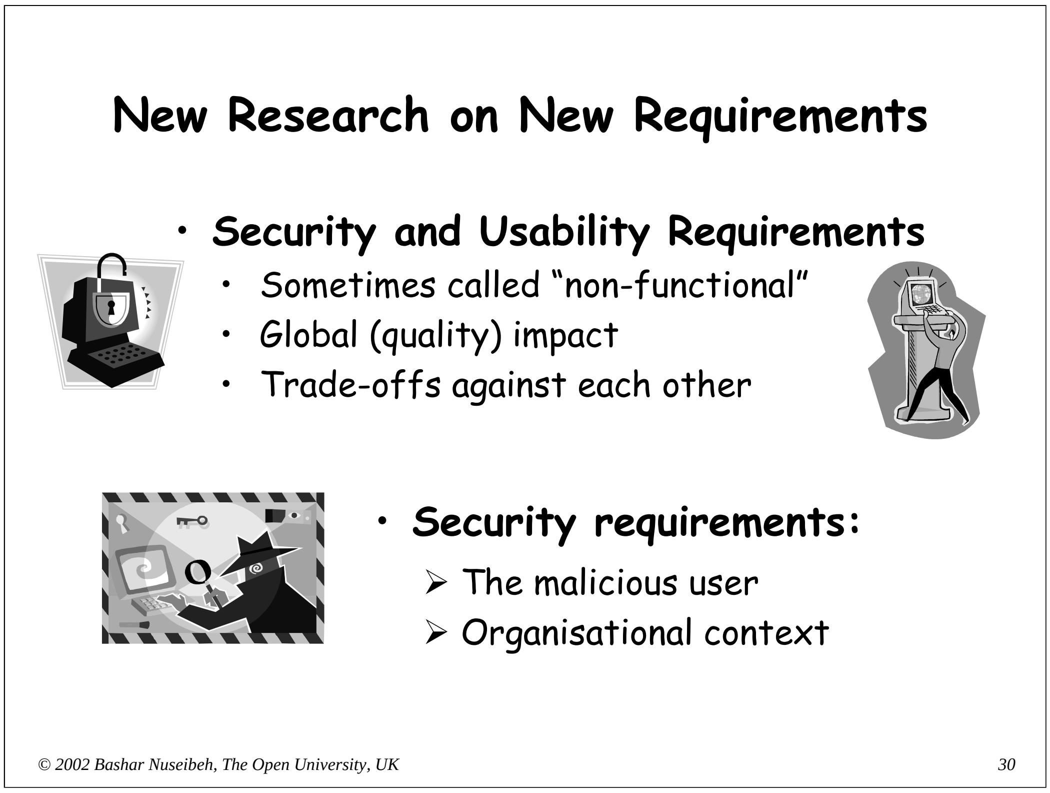
29

So, we have been looking at new, more effective, ways of relating requirements to design.

[TWIN PEAKS FIGURE – see reference below for details: basically requirements and designs need to be developed concurrently, since they inform/constrain each other]

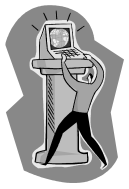
Some of our research in this area has been informed by collaboration with industrial partners, who present us with the reality of development processes in their organisations.

Reference: B. Nuseibeh, "Weaving Together Requirements and Architecture", *IEEE Computer*, 34(3):115-117, March 2001.



New Research on New Requirements

- **Security and Usability Requirements**
  - Sometimes called "non-functional"
  - Global (quality) impact
  - Trade-offs against each other



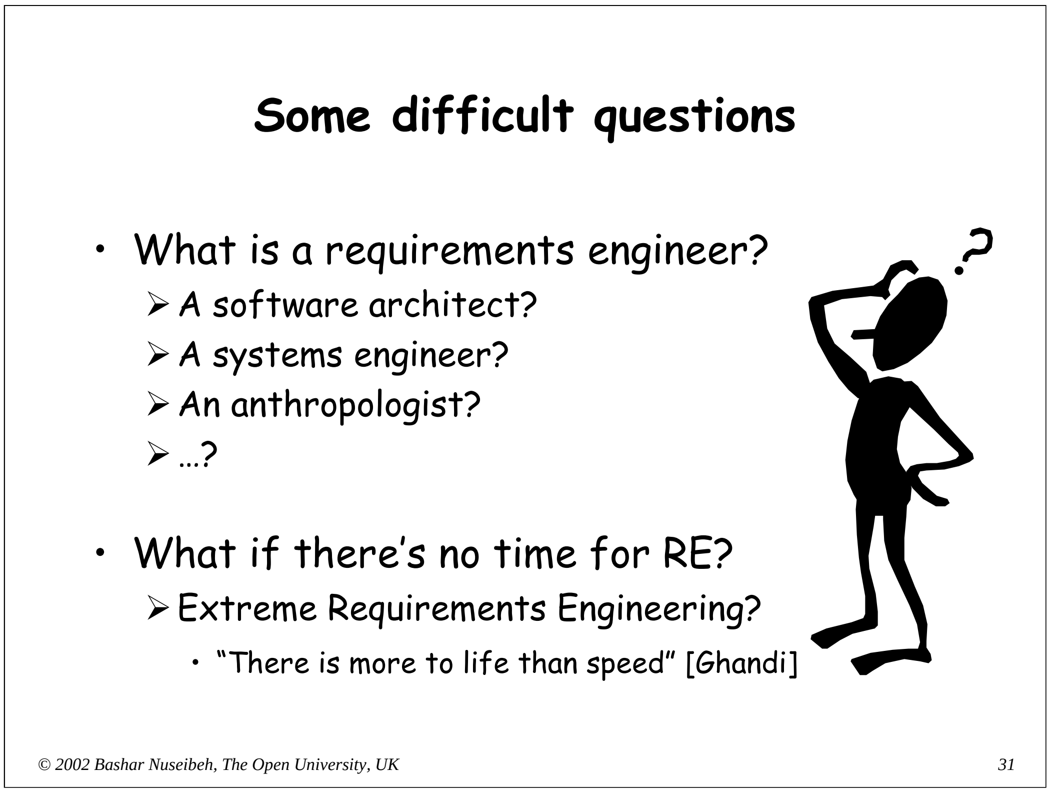
- **Security requirements:**
  - ➢ The malicious user
  - ➢ Organisational context

30

We have also been looking at a group of requirements, sometimes called "non-functional requirements", such as security and usability. These requirements relate to attributes of systems as whole, and can be more difficult to specify and analyse. They often need to be traded-off against each other – for example, a more secure system can be less usable if the user is prompted for a password for every action they need to perform.

Security requirements are particularly interesting. Unlike trying to understand 'normal' user requirements, security requirements relate to *malicious* users who are trying to break into the software systems we are building. So, the research strategy we are developing is one that attempts to incorporate and build in security into software systems from every early stages of their development, combined with our analysis approaches that attempt to discover obstacles to achieving our security requirements.

There are some very interesting and open research questions in this area, and the area itself is conveniently very topical too!



Some difficult questions

- What is a requirements engineer?
  - A software architect?
  - A systems engineer?
  - An anthropologist?
  - …?
- What if there's no time for RE?
  - Extreme Requirements Engineering?
    - "There is more to life than speed" [Ghandi]

31

There are also some broader and difficult questions that are always at the back of my mind, and for which I continue to seek an answer.

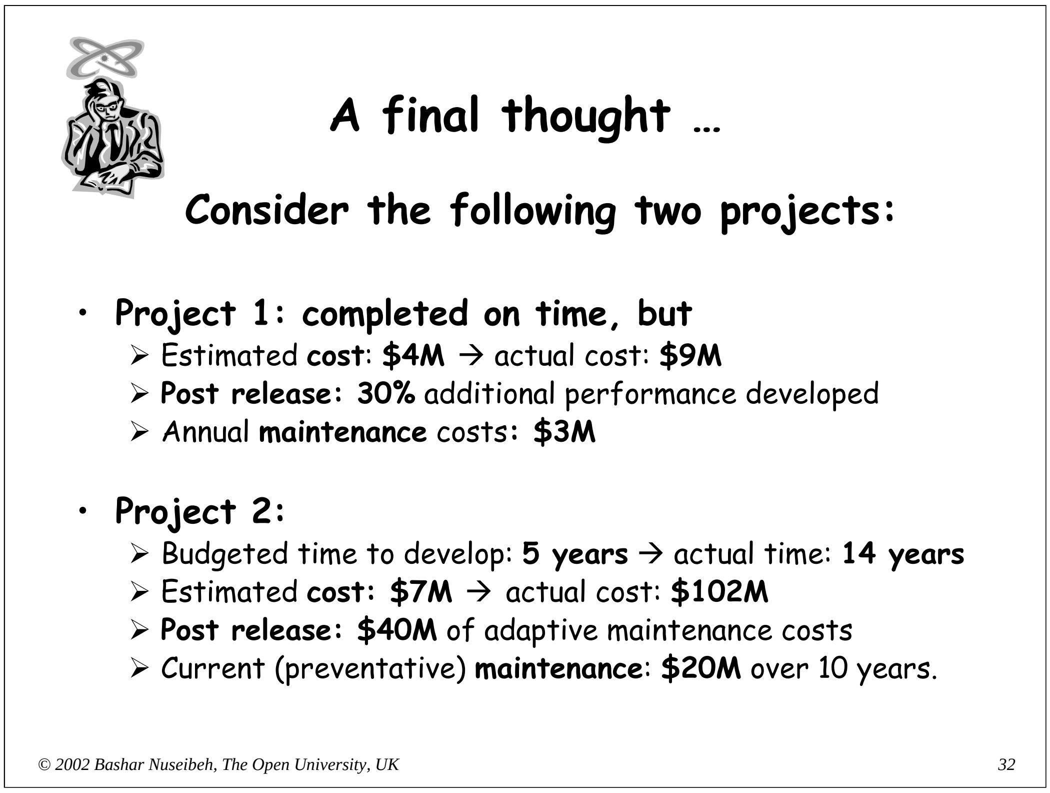
What is a requirements engineer?

1. A software architect?
2. A systems engineer?
3. … ?

For me, the requirements engineer is a Jack-of-all-trades and master of all! Indeed, I prefer to talk about the *discipline* of requirements engineering rather than the engineer.

What if there is no time for RE?

I am not very sympathetic to those who prefer to build systems before getting some idea of requirements from stakeholder, however, there is sometimes a genuine need to find out these requirements more quickly than usual. The emerging notion of *Extreme Requirements Engineering* has been suggested by some as a way to reduce some of the time associated with some RE activities. In any case, it is worth bearing the saying by Ghandi: "There is more to life than speed".



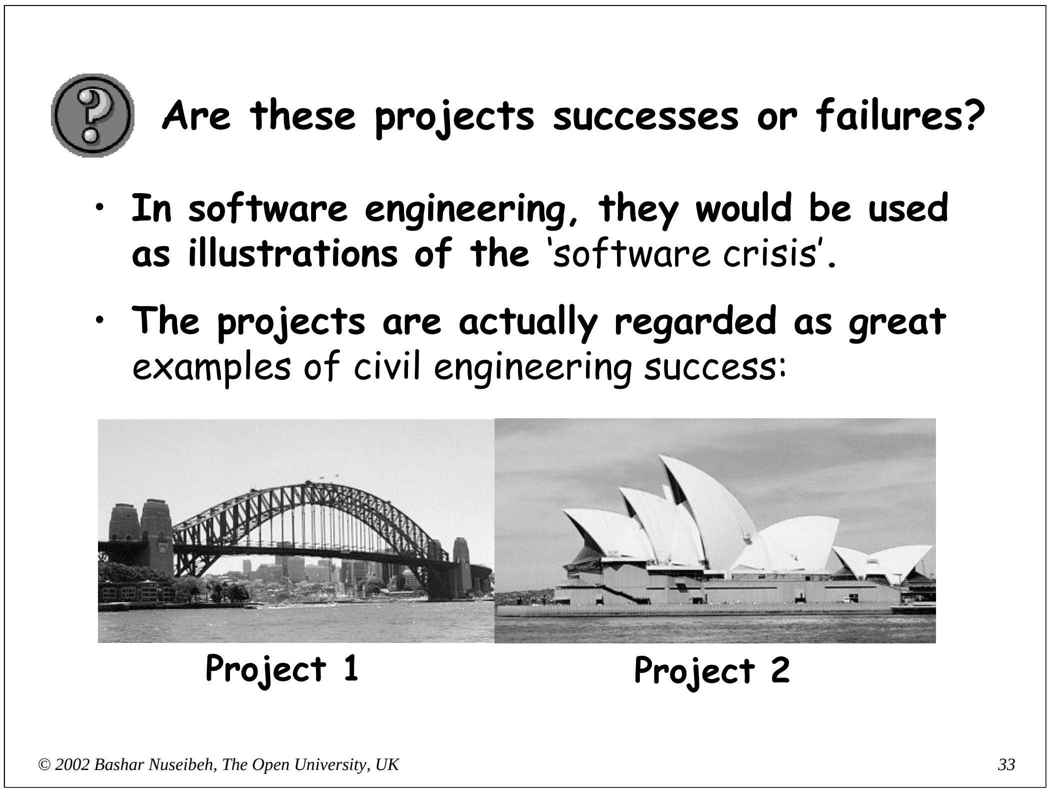
A final thought …

Consider the following two projects:

- **Project 1: completed on time, but**
  - Estimated **cost**: **$4M** → actual cost: **$9M**
  - **Post release: 30%** additional performance developed
  - Annual **maintenance** costs: **$3M**
- **Project 2:**
  - Budgeted time to develop: **5 years** → actual time: **14 years**
  - Estimated **cost: $7M** → actual cost: **$102M**
  - **Post release: $40M** of adaptive maintenance costs
  - Current (preventative) **maintenance**: **$20M** over 10 years.

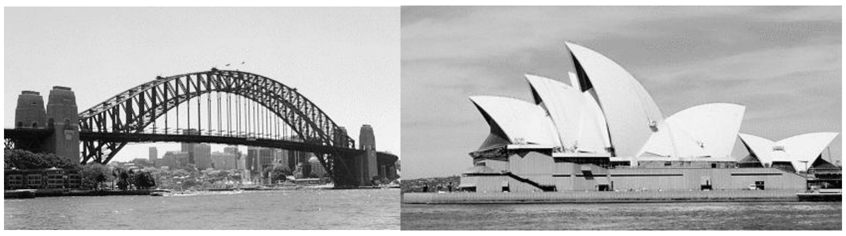
32

I removed this and the next slide out of my inaugural on the morning of the lecture, because I was concerned about having too many slides!



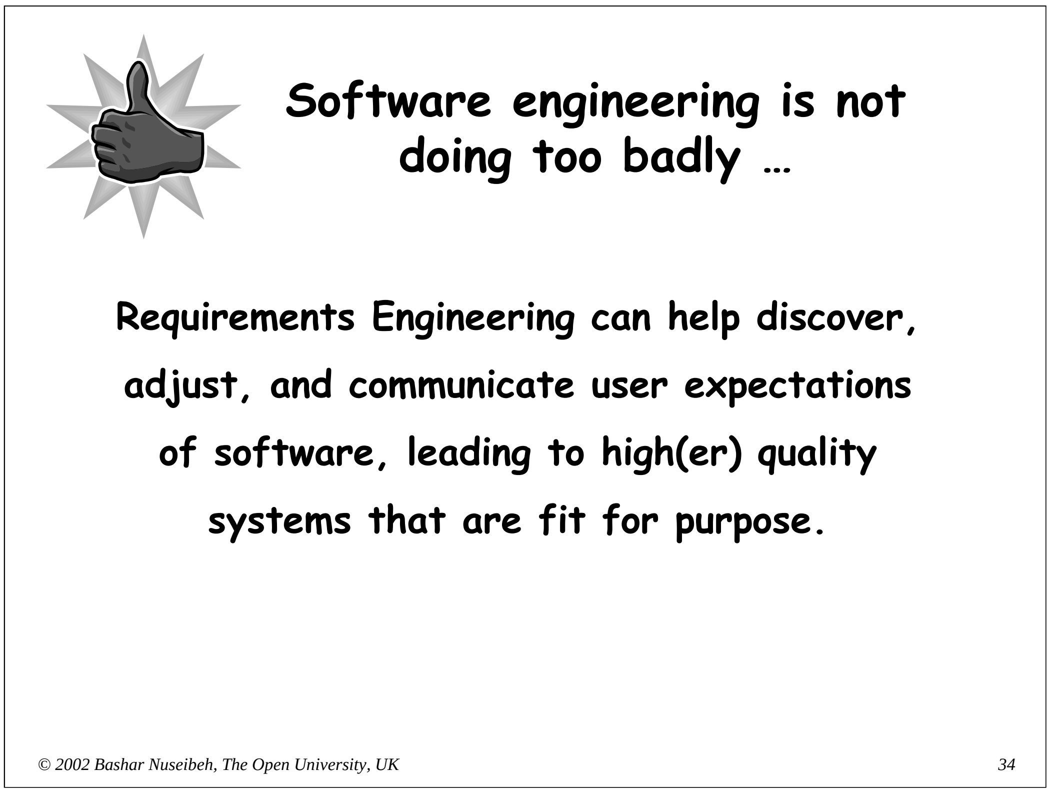
Are these projects successes or failures?

- In software engineering, they would be used as illustrations of the 'software crisis'.
- The projects are actually regarded as great examples of civil engineering success:



Project 1 Project 2

I removed this and the previous slide out of my inaugural on the morning of the lecture, because I was concerned about having too many slides!

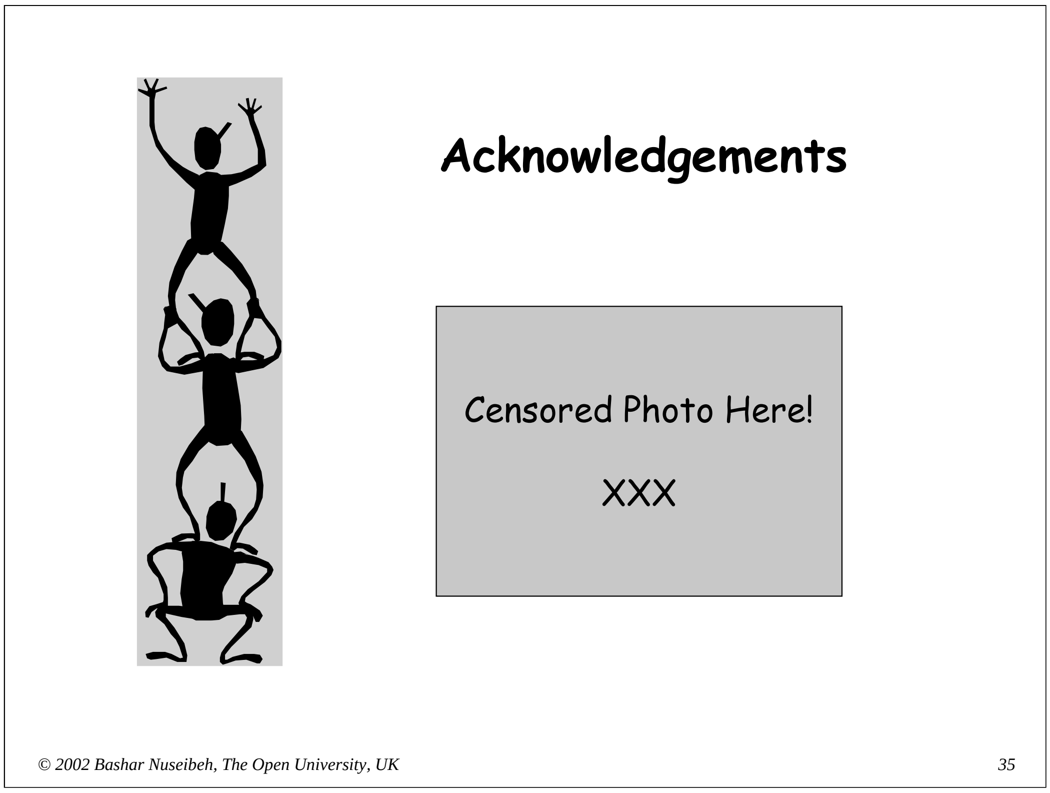


Software engineering is not doing too badly …

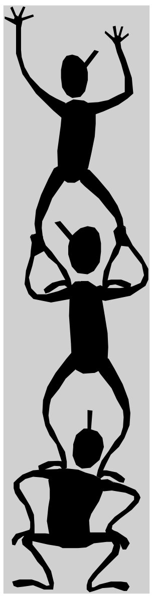
Requirements Engineering can help discover, adjust, and communicate user expectations of software, leading to high(er) quality systems that are fit for purpose.

Requirements Engineering can help discover, adjust, and communicate user expectations of software, leading to high(er) quality systems that are fit for purpose. It is as much about *discovering problems* and *managing expectations*, as it is about developing software.

In this lecture, I have tried to give you lots of reasons why RE is important, some reasons why it is difficult, and a few ways in which we have been addressing a few of these difficulties. I would like to conclude by giving you lots of reasons why it has been such fun to work in this field.



Acknowledgements



Censored Photo Here!

XXX

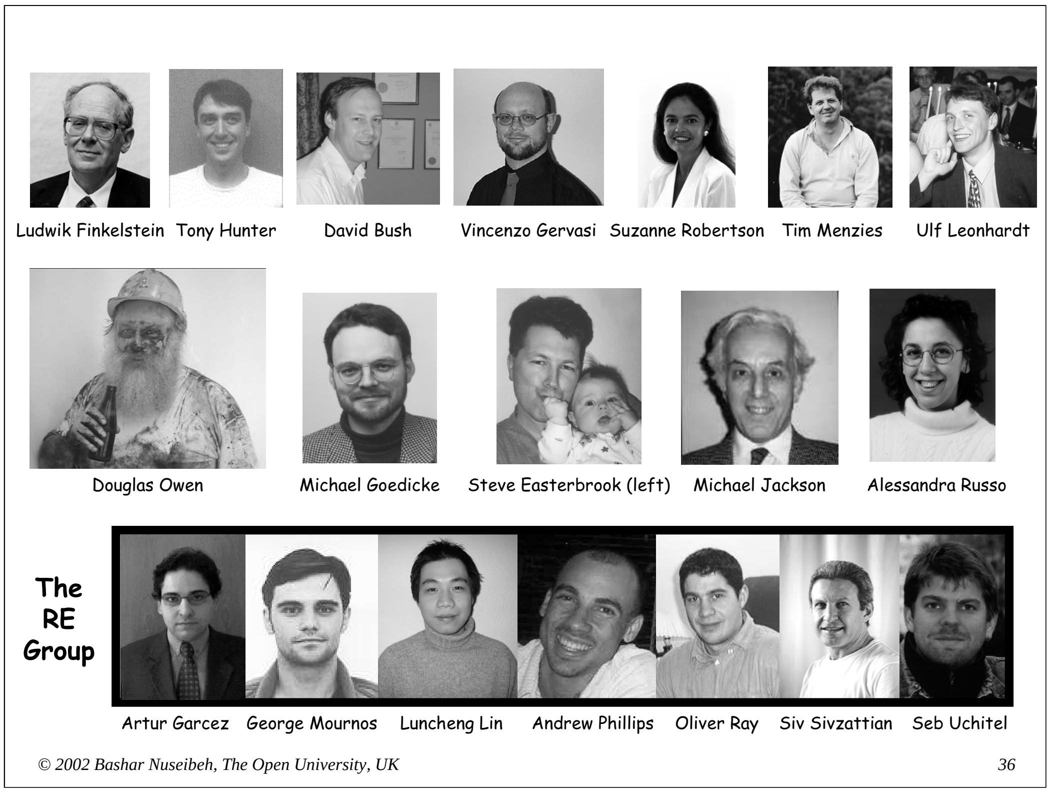
© 2002 Bashar Nuseibeh, The Open University, UK 35

It has been such fun because of the people with whom I have worked.

We have worked together in very comfortable work environments, but also under vary basic conditions!

[COLLEAGUES AND I ENJOYONG A “RESEARCH” MEETING IN JAPANESE BATHS IN KANAZAWA, JAPAN!!]

Not all my slides are suitable for universal viewing!



Ludwik Finkelstein shared with me his considerable knowledge and experience of electronic instruments when I started working with him in the early 1990's.

Tony Hunter patiently introduced me to logic and solved many of the hard problems that we faced.

David Bush continues to teach me about the realities of software systems development in my work with him at NATS.

Vincenzo Gervasi shared with me his technical expertise in analysing specifications as we struggled to make sense of NASA Space Station requirements specifications.

Suzanne Robertson taught me how to teach practitioners requirements engineering.

Tim Menzies, whom I met while visiting NASA, continues to amuse and amaze me with his creative ideas.

Ulf Leonhardt in six months managed to develop a software system that might have taken others much longer to develop.

Doug Owen, a mining engineer, with whom I did some early work, convinced me that you're not an engineer unless you get more than just your hands dirty!

Michael Goedicke, during his sabbatical at Imperial College, laid the foundations for our work on multiple perspectives for many years to come.

Steve Easterbrook and I shared the same PhD supervisor at Imperial College, but not at the same time, which may explain the "blood brothers" type relationship we have had over the years. We spent sabbaticals visiting each other, but wrote some of our best papers while at opposite ends of the earth…

Michael Jackson has been an inspiration to the entire RE community, and his seminal contributions to the field have had a tremendous impact on my own research work. I thank him for agreeing to join us here at the OU as a Visiting Research Professor in the Computing Department.

Alessandra Russo is as brilliant a logician and as passionate a researcher as they come. She has been an inspirational influence on the analysis research I described today, and my patient logic teacher, without whom I would be lost. She is an absolute delight to work with.

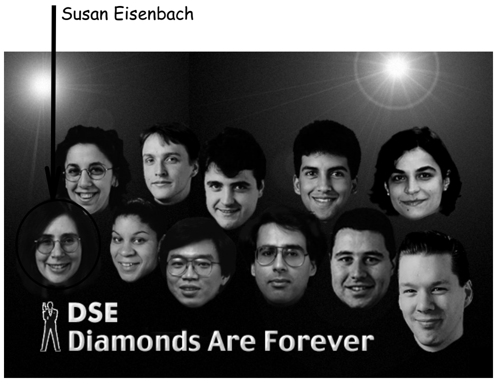
The RE-Group of RA's and students at Imperial College, and increasingly at the OU, has been a weekly source of fun and debate. Each of its members Artur, George, Luncheng, Andrew, Oliver, Siv, Sebastian, and Will – for whom I have no photo – is destined to greatness I'm sure!

Jeff Magee Morris Sloman Jeff Kramer

Distributed Software Engineering (DSE) Group At Imperial College

Susan Eisenbach

Anthony Finkelstein



37

I spent over ten years of my career, before joining the OU, as part of the Distributed Software Engineering *(DSE)* at Imperial College. The group's unofficial trademark is the "DSE in Black" poster, featuring three of the group's founders! The two *Jeff's and Morris* have been "formidable role models", if I can use that phrase.

It has been a particular privilege to work with *Jeff Kramer*, as a student, as an RA, as a colleague, and as a friend. He has been an unfailing source of wisdom and advice to me over the years, in the face my unending torrent of questions and dilemmas – not all of them technical! He's not bad at delivering best man speeches at wedding receptions either!

The rest of the *DSE group*, past and present, has been, and continues to be, a source of friendship and collaboration. I am happy to see *Emil*, *Naranker*, and *Manny* in the audience here today.

Susan Eisenbach deserves a special mention. For over five years, Susan and I shared the teaching of an advanced fourth year software engineering course. Our friendly rivalry kept us both on our toes and ensured that the students received a very polished lecture course. Today, we share the supervision of a PhD student, Andrew Phillips, who also still keeps us on our toes!

Last but not least, a special thanks to *Anthony Finkelstein*. I met Anthony for the first time in 1988, when I was an MSc student at Imperial College, and his creativity, generosity, and perseverance, have been inspiration to me since then. You can hardly make out his face in this picture, having just tobogganed down a steep mountain in the middle of the night in pitch-black darkness. You may just about be able to make out a figure hurtling down the mountain behind him. You can guess who that might be! Working with Anthony has always been exhilarating, but surprisingly down to earth experience! [Only a portrait photo shown in the above slide!]

New Colleagues, Students, & Friends

An exciting time in the Computing Department and at the OU !

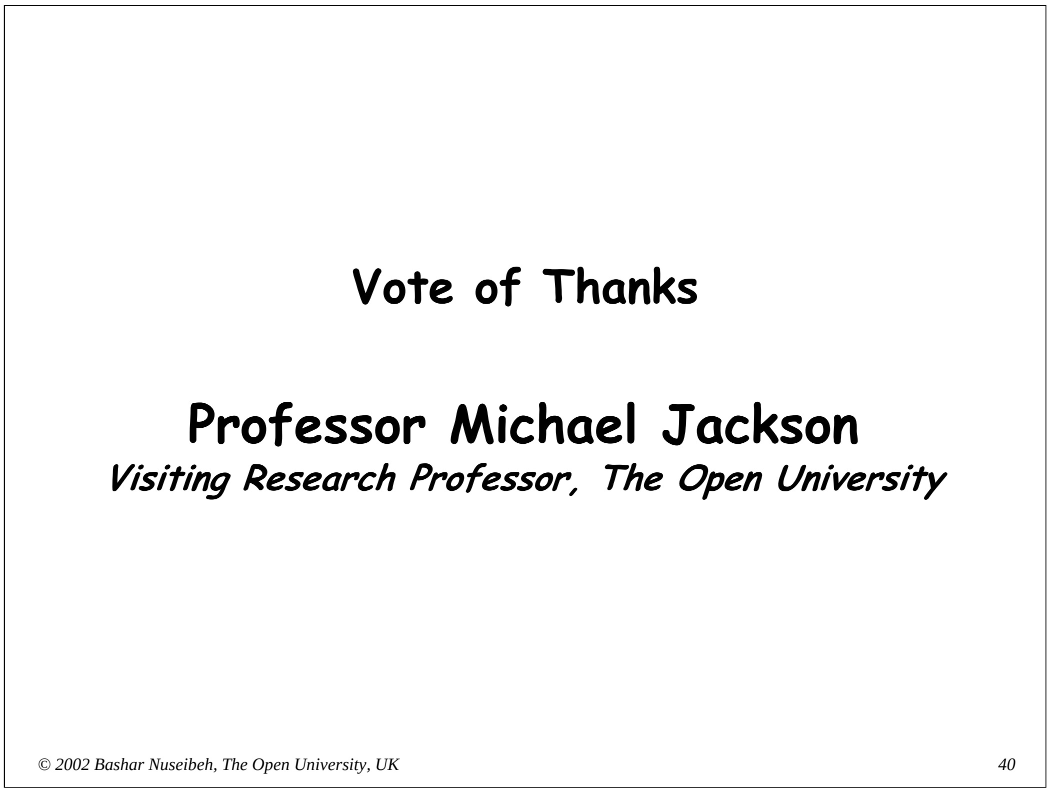
And finally. It was a very difficult decision for me to leave Imperial College, after 10 happy years there, but it is a decision that I have not regretted for one moment. The OU and all its members have been tremendously warm and welcoming, and colleagues in the Computing Department and the Faculty of Maths & Computing have been a joy to work with.

The Computing Department at the OU is a wonderful (and cosy!) place to work. It is buzzing with enthusiasm and activity, thanks to the passion and commitment of its members. I'm looking forward to many happy and productive years there.

Thank You

39

My time is up. Thank you for listening.



Vote of Thanks

Professor Michael Jackson

Visiting Research Professor, The Open University

© 2002 Bashar Nuseibeh, The Open University, UK 40

Vice Chancellor, Professor Nuseibeh, ladies and gentlemen: It is a great honour to have been asked to thank Professor Nuseibeh for his Inaugural Lecture. As the Vice-Chancellor said, an inaugural lecture is a very special occasion. There is a well-established custom that an inaugural lecture takes place not at the beginning of the tenure of a Chair but only later, after the eminent lecturer has occupied the Chair for some time. At first this may seem puzzling. But in fact it is entirely appropriate. It allows the professor’s colleagues time to have worked with him, and to have learned something of his quality. I am lucky to have been able to work with Bashar during the past year, here at The Open University, to my great enlightenment and profit, and I am very grateful for that. And as the Vice-Chancellor said in her introduction, to inaugurate the tenure of a Chair is more than to make a beginning. As its etymology indicates, it is also to make that tenure auspicious, to make it significant of the future, and to ensure its good fortune.

Professor Nuseibeh has come to the Open University with well-earned renown from his previous work at Imperial College. Among the subjects of that work that he has described today are two very important concepts in Software Requirements Engineering: Viewpoints, and the Toleration and Management of Inconsistency. Each of these ideas is more than a technical contribution: it is a vital influence in shaping a more human and more humane approach to the development of software, recognising that many different people have a stake in a system, and that their points of view may be not merely different but inconsistent, and in active conflict.

Professor Nuseibeh quoted Emerson in praise of inconsistency. He might also have quoted another American, Walt Whitman, who famously said: "Do I contradict myself? Very well then, I contradict myself. I am large, I contain multitudes." Software developers and requirements engineers must be equipped by their education and their technology to be large, to contain multitudes, to deal honestly with the variety and conflict and contradiction great of human experience, and to resolve or reconcile these differences where resolution or reconciliation is possible.

Professor Nuseibeh: your inaugural lecture has given us all great amusement, great pleasure and great illumination today. On behalf of us all, I thank you for that lecture, and tell you that we are now looking forward to your continued tenure of the chair to the great benefit and advantage of the University, of your postgraduates students and of all of your colleagues, now and in the years to come. May your tenure be auspicious, may it be significant of the future, and may it be attended by every good fortune. Thank you very much.