Module 1

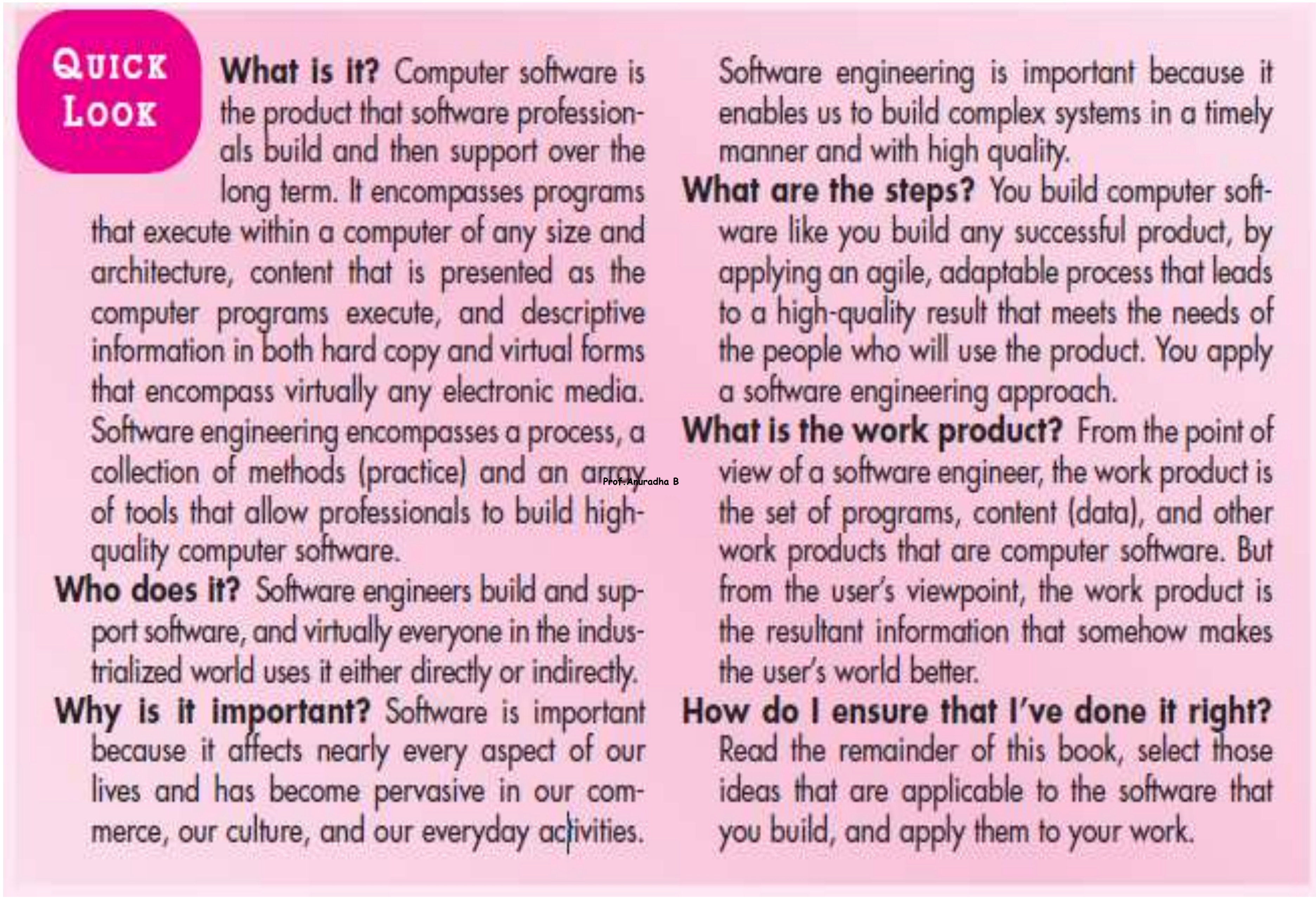
Prof.Anuradha B

Chapter 1
Software and software Engineering

Prof. Anuradha B

Quick Look

- **Computer software** is the product that software professionals build and then support over the long term.
- It encompasses programs that execute within a computer of any size and architecture, content that is presented as the computer programs execute, and descriptive information in both hard copy and virtual forms that encompass virtually any electronic media.
- **Software engineering** encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high quality computer software.



QUICK LOOK

What is it? Computer software is the product that software professionals build and then support over the long term. It encompasses programs that execute within a computer of any size and architecture, content that is presented as the computer programs execute, and descriptive information in both hard copy and virtual forms that encompass virtually any electronic media. Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high-quality computer software.

Who does it? Software engineers build and support software, and virtually everyone in the industrialized world uses it either directly or indirectly.

Why is it important? Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture, and our everyday activities. Software engineering is important because it enables us to build complex systems in a timely manner and with high quality.

What are the steps? You build computer software like you build any successful product, by applying an agile, adaptable process that leads to a high-quality result that meets the needs of the people who will use the product. You apply a software engineering approach.

What is the work product? From the point of view of a software engineer, the work product is the set of programs, content (data), and other work products that are computer software. But from the user's viewpoint, the work product is the resultant information that somehow makes the user's world better.

How do I ensure that I've done it right? Read the remainder of this book, select those ideas that are applicable to the software that you build, and apply them to your work.

1.1 THE NATURE OF SOFTWARE

- Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation

THE NATURE OF SOFTWARE

Quote:

"For I dipped into the future, far as the human eye could see, Saw the vision of the world, and all the wonder that would be."

Tennyson

KEY POINT

Software is both a product and a vehicle for delivering a product.

The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application. And yet, the same questions asked of the lone programmer are being asked when modern computer-based systems are built:

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all the errors before we give the software to customers?
- Why do we continue to have difficulty in measuring progress as software is being developed?

Quote:

"Computers make it easy to do a lot of things, but most of the things that they make it easier to do don't need to be done."

Andy Rooney

1.1.1 Defining Software

- *Instructions (computer programs) that when executed provide desired function and performance*
- *Data structures that enable the programs to adequately manipulate information*
- *Documents that describe the operation and use of the programs.*

Characteristics of Software

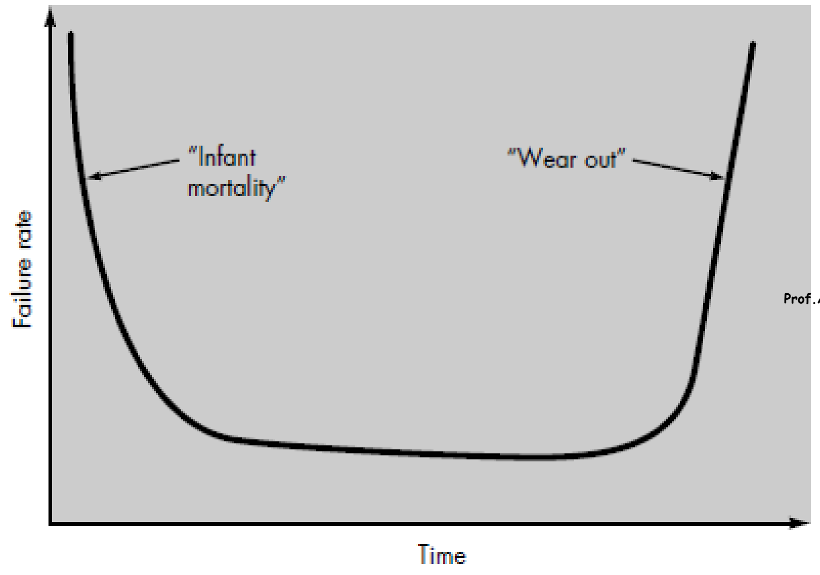
1.Software is developed or engineered, it is not manufactured in the classical sense

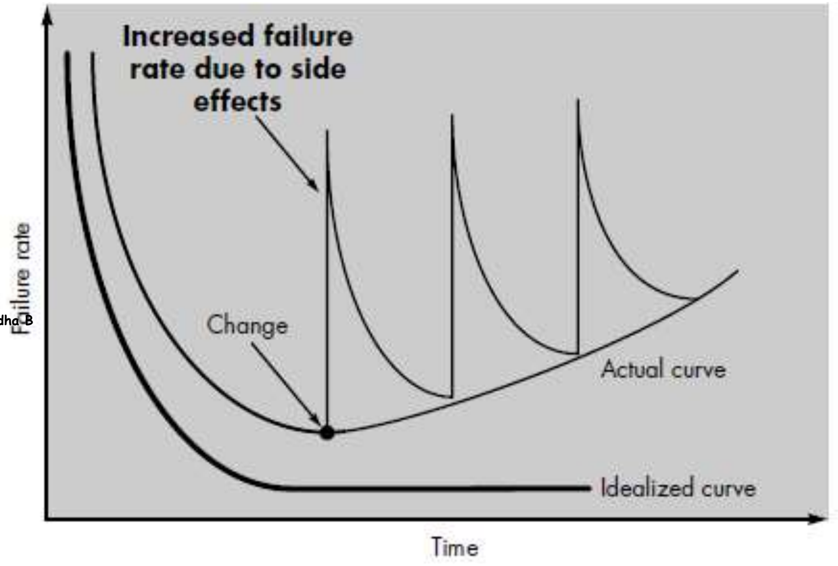
Hardware	Software
Manufacture	Development
High Quality with good Design	High Quality with good Design
Quality problem	Can be corrected easily
Require construction of product	Require construction of product but with different approach
Manageable	Cost are concentrated in Engineering

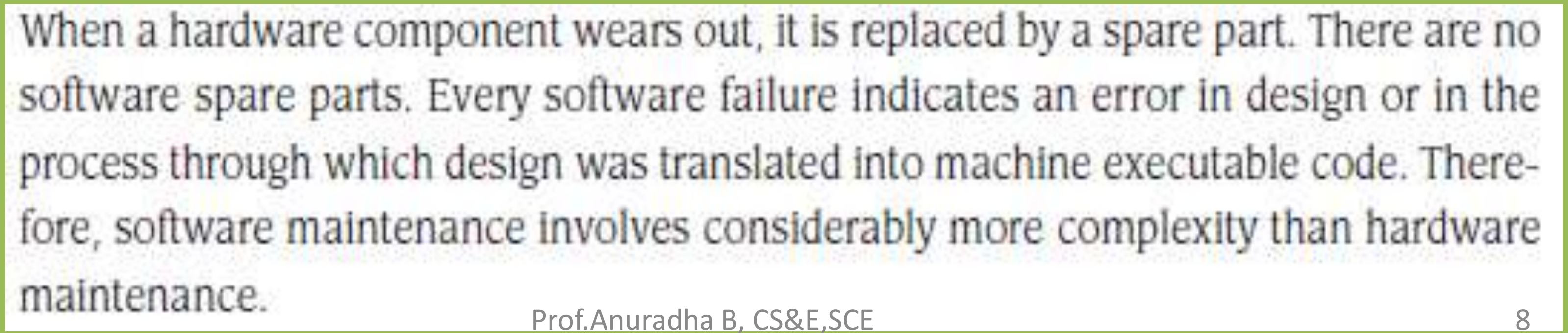
2. Software doesn't "Wear out"

Failure curve for hardware

Idealized and actual failure curves for software





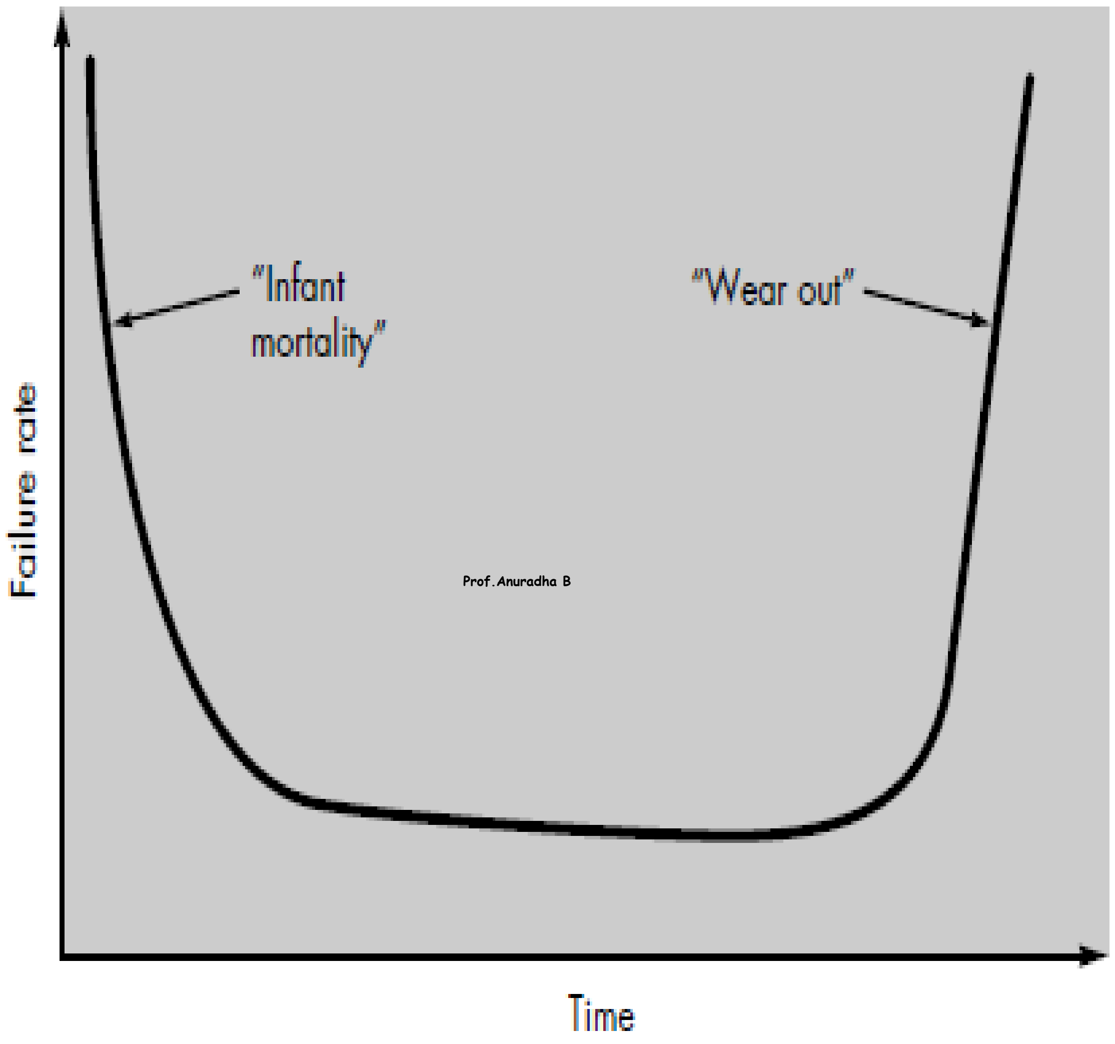


When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity than hardware maintenance.

Bathtub Curve

High failure rate early in the life

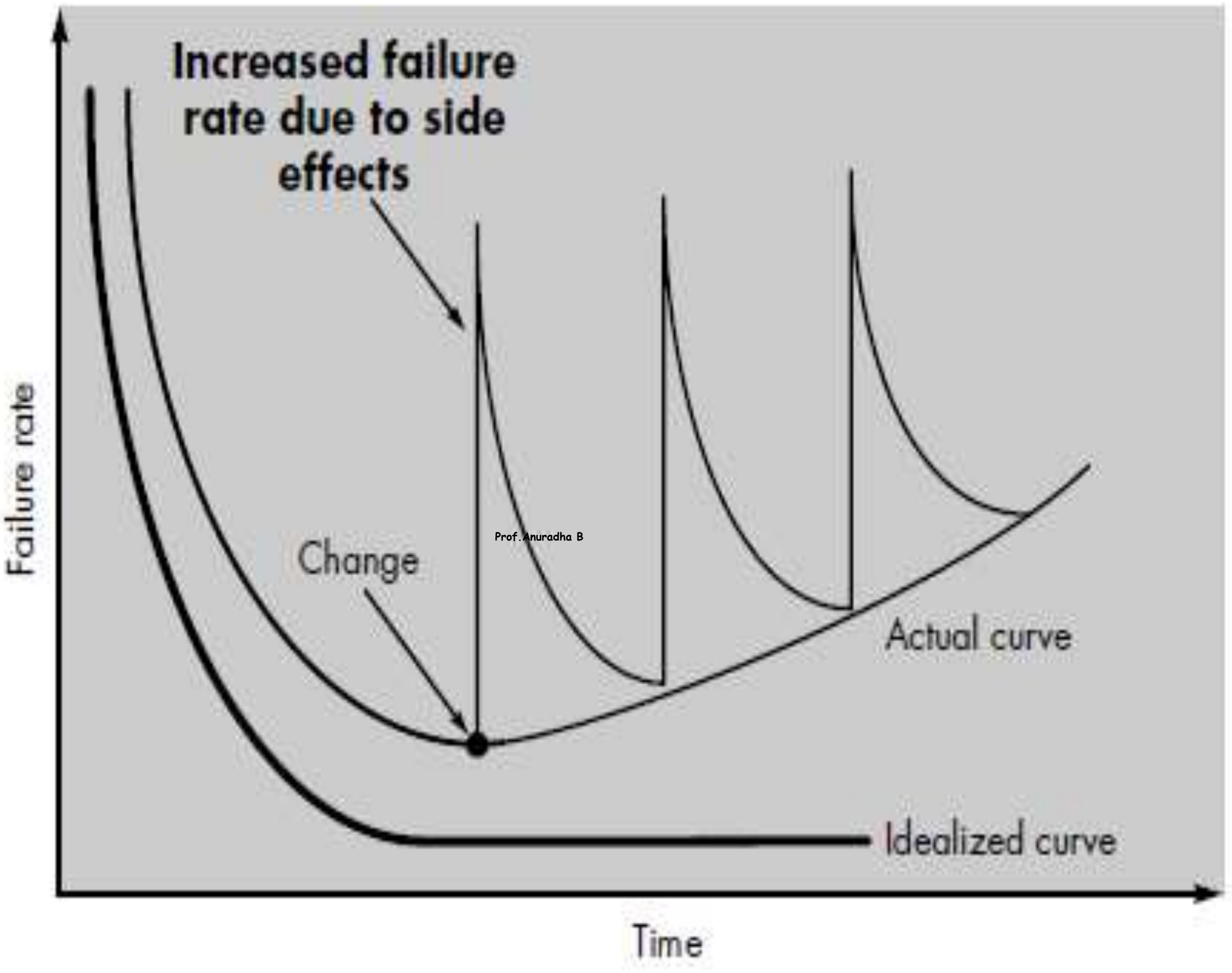
Design and Manufacturing Defects



Failure rate increases

Dust
Vibration
Abuse
Temperature extremes

Steady state level


Increased failure rate due to side effects
Failure rate
Change
Prof.Anuradha B
Actual curve
Idealized curve
Time

3.CBSE

Although the industry is moving toward component-based assembly, most software continues to be custom built.

- A software component should be designed and implemented so that it can be reused in many different programs

- ➢ Subroutine
- ➢ Data structure
- ➢ GUI
- ➢ The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

1.1.2 Software Application Domains

- **System software:** collection of programs written to service other programs. Some system software processes complex, but determinate , information structures. e.g., compilers, editors, and file management utilities, OS , Drivers, n/w softwares, telecommunication Processors

- **Application software:** stand-alone programs that solve a specific business need. In addition to conventional data processing applications, application software is used to control business functions in real time

- **Engineering/scientific software:** Applications range from astronomy to volcanology, from **automotive stress analysis to space shuttle orbital dynamics**, and from molecular biology to automated manufacturing

- **Embedded software:** resides within a product or system and is **used to implement and control features and functions** for the end user and for the system itself.

- **Product-line software:**designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

- **Web applications :** WebApps are evolving into sophisticated computing environments that not only provide stand-alone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications

- **Artificial intelligence software:** makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Challenges for Software Engineers

- **Open-world computing**—the rapid growth of wireless networking may soon lead to true pervasive, distributed computing.

❖ **Challenge** - The Challenge for software engineers will be to develop systems and application software that will allow mobile devices, personal computers, and enterprise systems to communicate across vast networks.

- **Netsourcing**—the World Wide Web is rapidly becoming a computing engine as well as a content provider.

❖ **Challenge** - The Challenge for software engineers is to architect simple (e.g., personal financial planning) and sophisticated applications that provide a benefit to targeted end-user markets worldwide.

- **Open source**—a growing trend that results in distribution of source code for systems applications (e.g., operating systems, database, and development environments) so that many people can contribute to its development

❖ **Challenge** - The challenge for software engineers is to build source code that is self-descriptive, but more importantly, to develop techniques that will enable both customers and developers to know what changes have been made and how those changes manifest themselves within the software.

1.1.3 Legacy Software

- *Older programs—often referred to as legacy software*

❖ "Legacy software systems . . . were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve"

❖ "Many legacy systems remain supportive to core business functions and are 'indispensable' to the business."

- Hence, legacy software is characterized by longevity and business criticality.

Legacy Software

- Characteristic that is present in legacy software—*poor quality*
- Legacy systems sometimes have inextensible designs, convoluted code, poor or nonexistent documentation, test cases and results that were never archived, a poorly managed change history—the list can be quite long. And yet, these systems support "core business functions and are indispensable to the business." What to do?
- *Do nothing*

Legacy Software

❖ As time passes, legacy systems often evolve for one or more of the following reasons:

- **The software must be adapted to meet the needs of new computing environments or technology.**

- **The software must be enhanced to implement new business requirements.**
- **The software must be extended to make it interoperable with other more modern systems or databases.**
- **The software must be re-architected to make it viable within a network environment.**

1.2 THE UNIQUE NATURE OF WEBAPPS

- In the early days of the World Wide Web (circa 1990 to 1995), *websites* consisted of little more than a set of linked hypertext files that presented information using text and limited graphics
- Today, WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have been integrated with corporate databases and business applications.
- Powell [Pow98] suggests that Web-based systems and applications **"involve a mixture between print publishing and software development, between marketing and computing, between internal communications and external relations, and between art and technology."**

Attributes of WebApps

- **Network intensiveness:** must serve the needs of a **diverse community of clients**
- **Concurrency:** A large number of users may **access the WebApp at one time**
- **Unpredictable load:** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance:** If a WebApp user must wait too long , he or **she may decide to go elsewhere**.
- **Availability:** users of popular WebApps often demand **access on a 24/7/365 basis**
- **Data driven :** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user **what about Database Integrity ?**

- **Content sensitive:** The **quality and aesthetic nature** of content remains an important determinant of the quality of a WebApp.

- **Continuous evolution:** It is not unusual for some WebApps to be updated on a **minute-by-minute schedule**

- **Immediacy:** get software to market quickly—is a characteristic of many application domains, **WebApps often exhibit a time-to-market that can be a matter of a few days or weeks**

- **Security:** In order to protect sensitive content and provide secure modes of data transmission, **strong security measures must be implemented throughout the infrastructure that supports a WebApp and within the application itself.**

- **Aesthetics:** An undeniable part of the appeal of a WebApp is its **look and feel**. When an application has been designed to market or sell products or ideas, **aesthetics may have as much to do with success as technical design**

1.3 SOFTWARE ENGINEERING

- We must recognize a few simple realities:
  - ***that a concerted effort should be made to understand the problem before a software solution is developed.***
  - ***that design becomes a pivotal activity.***
  - ***software should exhibit high quality.***
  - ***software should be maintainable.***

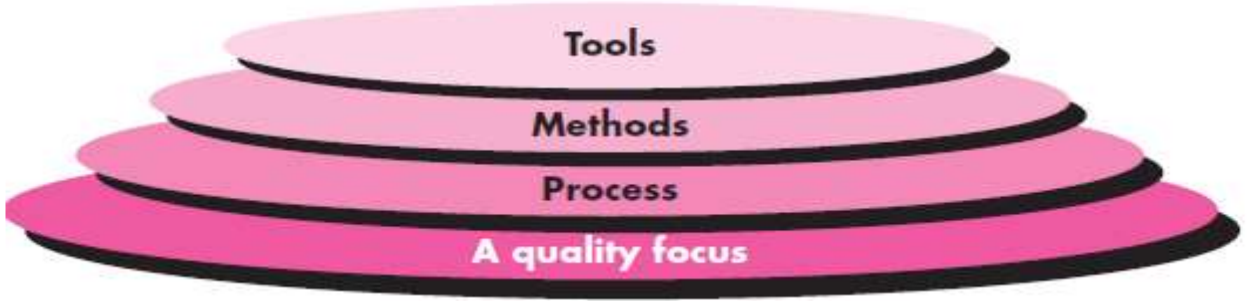
Prof.Anuradha B

Software Engineering is

(1) The application of a **systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software**; that is, the application of engineering to software.

(2) The **study of approaches** as in (1).

- **Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.**



Process:

•The foundation for software engineering is the ***process*** layer.

•The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.

.

Methods:

Prof.Anuradha B

- Software engineering *methods* provide the technical how-to's for building software
- Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques

Tools:

- Software engineering *tools* provide automated or semi automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called *computer-aided software engineering*, is established.

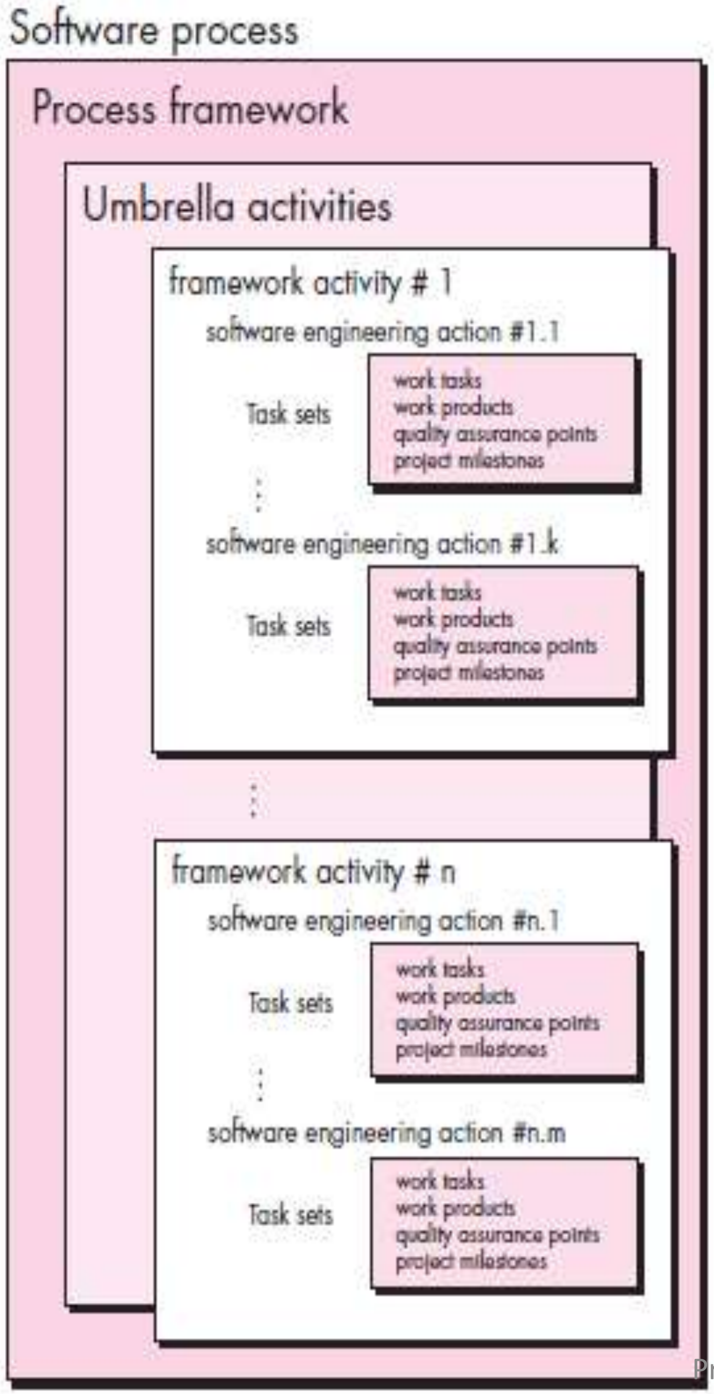
1.4 THE SOFTWARE PROCESS

- A *process* is a collection of **activities, actions, and tasks** that are performed when some work product is to be created.
- An *activity* strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An *action* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

THE SOFTWARE PROCESS

- A ***process framework*** establishes the foundation for a complete software engineering process by **identifying a small number of *framework activities*** that are applicable to all software projects, regardless of their size or complexity

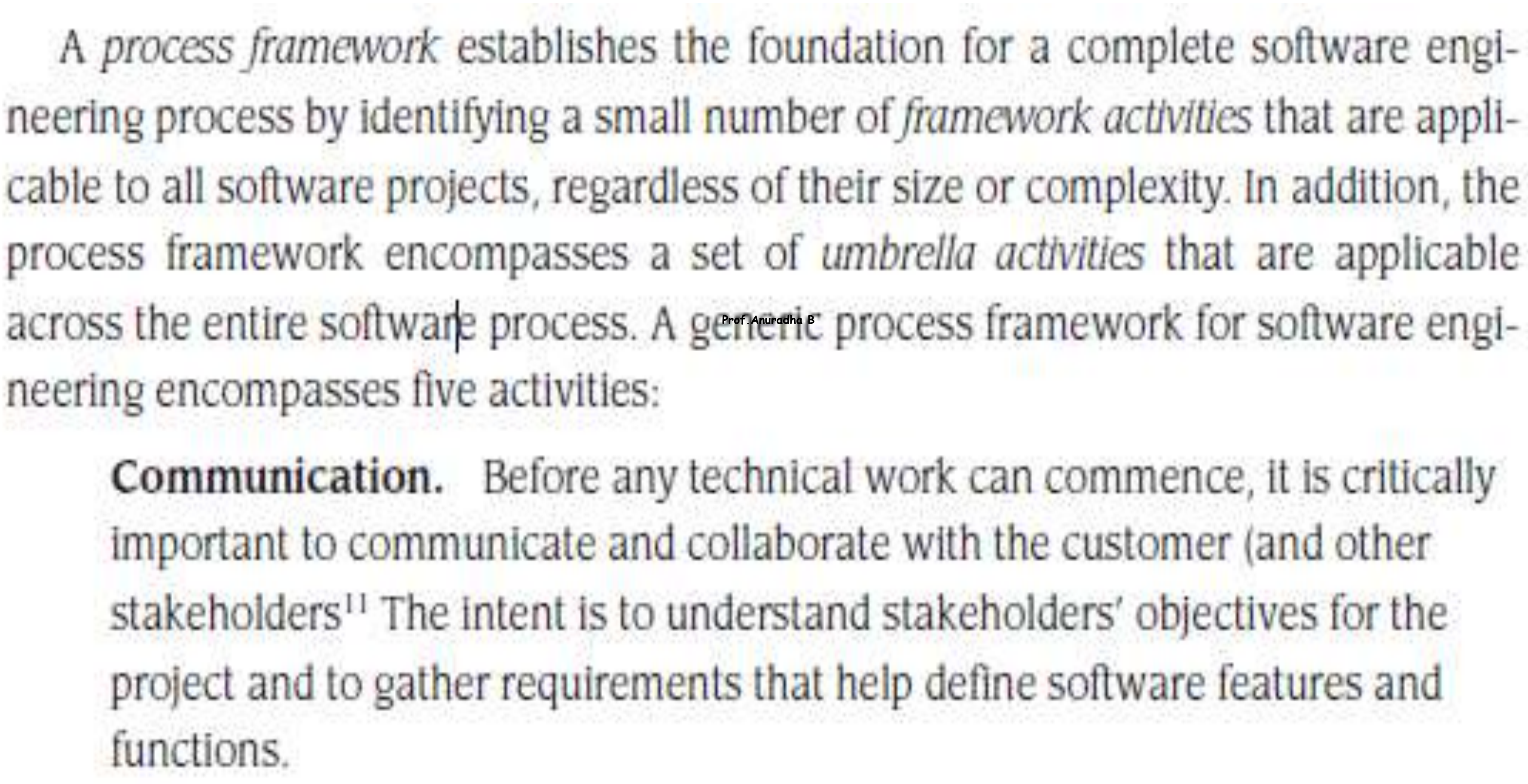
- In the context of software engineering, a process is *not* a rigid prescription for how to build computer software.
- Rather, it is an adaptable approach that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks



Process Frame work

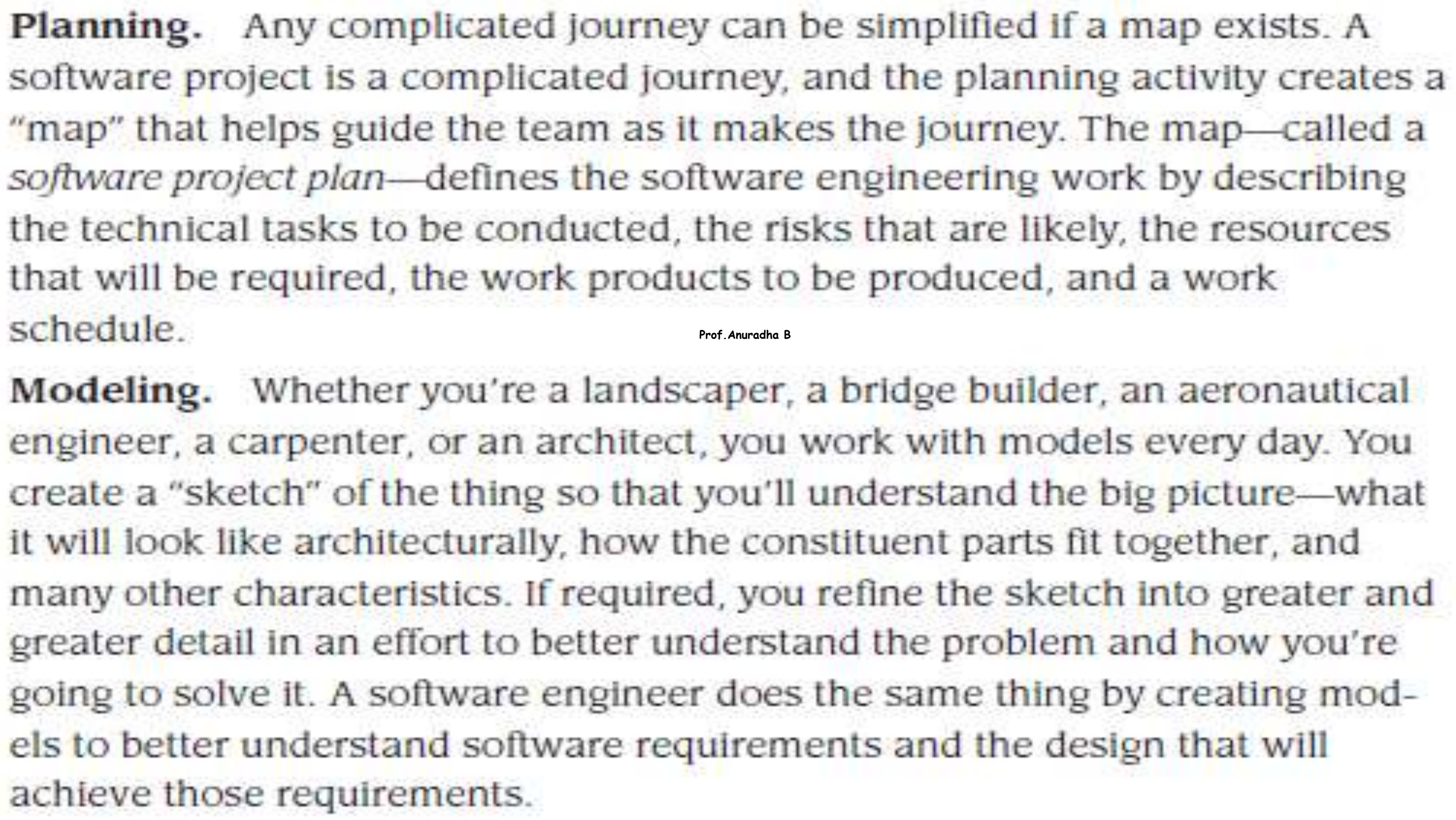
Process framework for software engineering defines five framework activities—

- **Communication**
- **Planning**
- **Modeling**
- **Construction**
- **Deployment.**



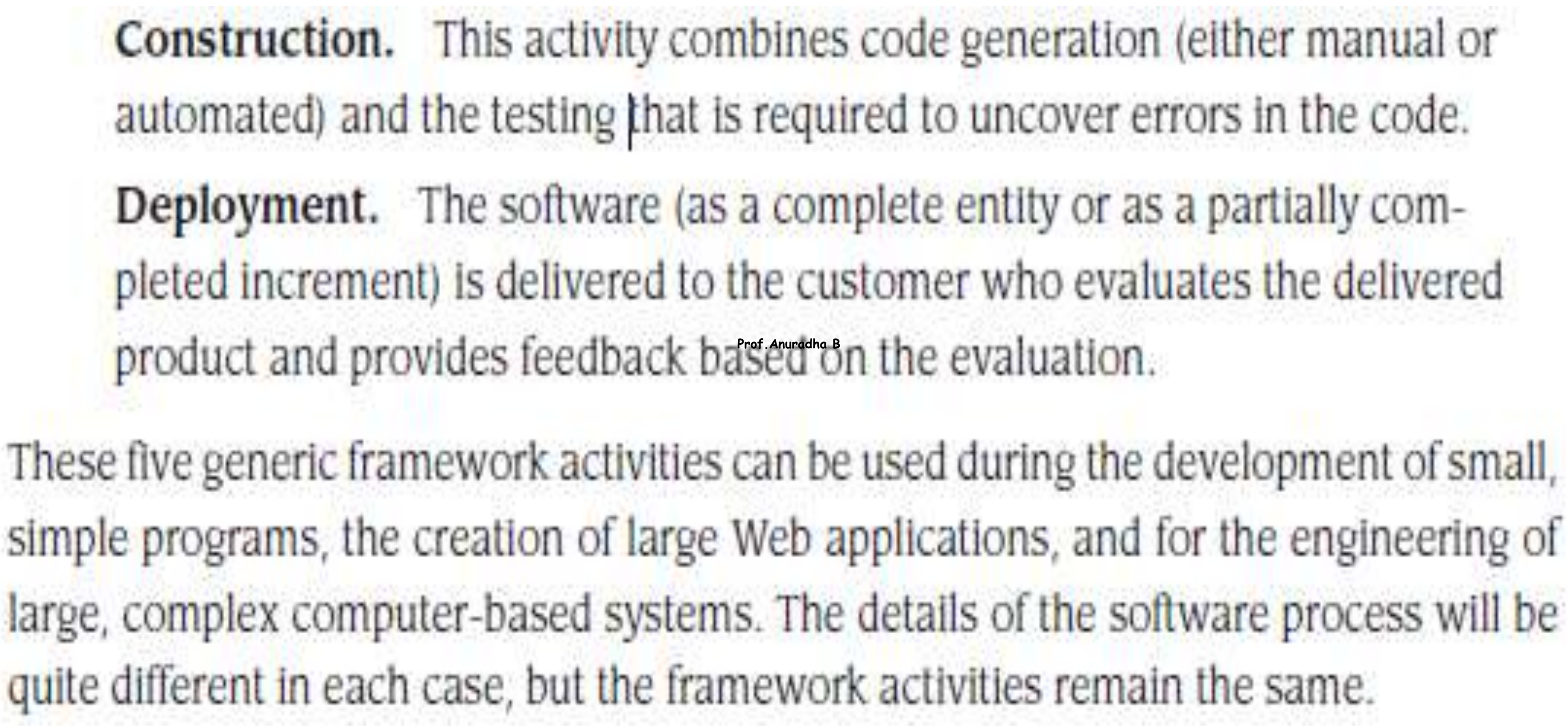
A *process framework* establishes the foundation for a complete software engineering process by identifying a small number of *framework activities* that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of *umbrella activities* that are applicable across the entire software process. A generic process framework for software engineering encompasses five activities:

Communication. Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders[11] The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.



Planning. Any complicated journey can be simplified if a map exists. A software project is a complicated journey, and the planning activity creates a "map" that helps guide the team as it makes the journey. The map—called a *software project plan*—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

Modeling. Whether you're a landscaper, a bridge builder, an aeronautical engineer, a carpenter, or an architect, you work with models every day. You create a "sketch" of the thing so that you'll understand the big picture—what it will look like architecturally, how the constituent parts fit together, and many other characteristics. If required, you refine the sketch into greater and greater detail in an effort to better understand the problem and how you're going to solve it. A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.



Construction. This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

Deployment. The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small, simple programs, the creation of large Web applications, and for the engineering of large, complex computer-based systems. The details of the software process will be quite different in each case, but the framework activities remain the same.

Umbrella Activities

- Software engineering process framework activities are complemented by a number of *umbrella activities.*
- In general, umbrella activities are applied throughout a software project and **help a software team manage and control progress, quality, change, and risk.**
- Typical umbrella activities include:

Communication + Planning + Modeling + Construction + Deployment + project tracking and control+Risk management+ Software quality assurance+ Technical reviews + Measurements + Configuration mgmt + reusability mgmt + Work product preparation and production

- **Software project tracking and control:** allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management:**assesses risks that may affect the outcome of the project or the quality of the product
- **Software quality assurance :** defines and conducts the activities required to ensure software quality.
- **Technical reviews :** assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Umbrella Activities

- **Measurement :** defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders’ needs; can be used in conjunction with all other framework and umbrella activities
- **Software configuration management :**manages the effects of change throughout the software process.

- **Reusability management :** defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production :** encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Process Frame work

- Defining a Framework Activity
  - *What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project?*

Example: Small Software Project
Activity: Communication
Action:
1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval

Example: Complex Software Project
Activity: Communication
Action:
1. *Inception : initiation*
2. *Elicitation : bringing facts*
3. *Elaboration : detailed*
4. *Negotiation : agree*
5. *Specification : detailed info,Fix*
6. *Validation: approval*

1.5 SOFTWARE ENGINEERING PRACTICE

- Generic framework activities—**communication, planning, modeling, construction,** and **deployment**—and umbrella activities **establish a skeleton architecture** for software engineering work
- But how does the practice of software engineering fit in?

1.5.1 The Essence of Practice

1. *Understand the problem* (communication and analysis).

2. *Plan a solution* (modeling and software design).

3. *Carry out the plan* (code generation).

4. *Examine the result for accuracy* (testing and quality assurance).

1.Understand the problem

- It's sometimes difficult to admit, but most of us suffer from hubris when we're presented with a problem.
- **We listen for a few seconds and then think, *Oh yeah, I understand, let's get on with solving this thing.***
- Unfortunately, understanding isn't always that easy.
- It's worth spending a little time answering a few simple questions:

- ❑ ***Who has a stake in the solution to the problem?*** That is, who are the stakeholders?
- ❑ ***What are the unknowns?*** What data, functions, and features are required to properly solve the problem?
- ❑ ***Can the problem be compartmentalized?*** Is it possible to represent smaller problems that may be easier to understand?
- ❑ ***Can the problem be represented graphically?*** Can an analysis model be created?

2.Plan the solution

Now you understand the problem (or so you think) and you can't wait to begin coding. Before you do, slow down just a bit and do a little design:

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?

- *Can sub problems be defined?* If so, are solutions readily apparent for the sub problems?
- *Can you represent a solution in a manner that leads to effective implementation?*
- Can a design model be created?

3.Carry out the plan

The **design** you've created serves as a **road map** for the system you want to build. There may be unexpected detours, and it's possible that you'll discover an even better route as you go, but the "plan" will allow you to proceed without getting lost.

• *Does the solution conform to the plan?* Is source code traceable to the design model?

• *Is each component part of the solution provably correct?* Have the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

4.Examine the result

You can’t be sure that your solution is perfect, but you can be sure that you’ve designed a sufficient number of tests to uncover as many errors as possible.

• ***Is it possible to test each component part of the solution?*** Has a reasonable testing strategy been implemented?

• ***Does the solution produce results that conform to the data, functions, and features that are required?*** Has the software been validated against all stakeholder requirements?

1.5.2 General Principles -

focus on software engineering practice

The First Principle: ***The Reason It All Exists***

A software system exists for one reason**:** ***to provide value to its users***. All decisions should be made with this in mind. Before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development processes, ask yourself questions such as**: "Does this add real value to the system?"** If the answer is "no," don't do it. All other principles support this one.

The Second Principle: ***KISS (Keep It Simple, Stupid!)***

- Software design is not a haphazard process.
- There are many factors to consider in any design effort. ***All design should be as simple as possible, but no simpler*****.**
- This facilitates having a more easily understood and easily maintained system. This is not to say that features, even internal features, should be discarded in the name of simplicity.
- Indeed, **the more elegant designs are usually the more simple ones.**
- **Simple also does not mean "quick and dirty."**
- In fact, it often takes a lot of thought and work over multiple iterations to simplify.
- **The payoff is software that is more maintainable and less error-prone**

The Third Principle: ***Maintain the Vision***

- *A clear vision is essential to the success of a software project*.
- Without one, a project almost unfailingly ends up being **"of two [or more] minds"** about itself.
- Without conceptual integrity, a system threatens to become a **patchwork** of incompatible designs, held together by the wrong kind of screws. . . . Compromising the architectural vision of a software system weakens and will eventually break even the well-designed systems.
- Having an empowered architect who can hold the vision and enforce compliance helps ensure a very successful software project.

Prof.Anuradha B

The Fourth Principle: ***What You Produce, Others Will Consume***

- *Always specify, design, and implement knowing someone else will have to understand what you are doing*.
- Someone may have to debug the code you write, and that makes them a user of your code.
- Making their job easier adds value to the system.

The Fifth Principle: ***Be Open to the Future***

- ***Never design yourself into a corner*.**
- Always ask "what if," and prepare for all possible answers by creating systems that solve the general problem, not just the specific one.
- This could very possibly lead to the reuse of an entire system.

- **The Sixth Principle:** ***Plan Ahead for Reuse***
- Reuse saves time and effort
- The reuse of code and designs has been proclaimed as a major benefit of using object-oriented technologies

- *Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated*.

The Seventh principle: ***Think!***

- *Placing clear, complete thought before action almost always produces better results*.
- When clear thought has gone into a system, value comes out

1.6 SOFTWARE MYTHS

- Software **myths—erroneous beliefs** about software and the process that is used to build it—can be traced to the earliest days of computing.

- Myths have a number of attributes that make them insidious

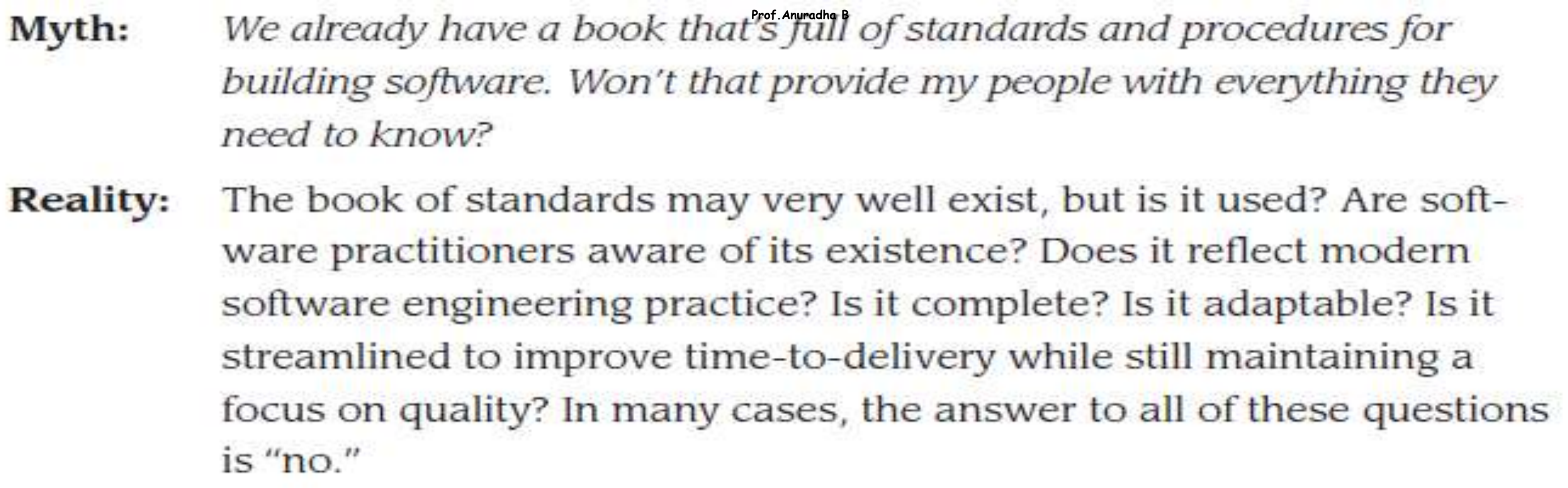
❑ **Management myths.**

❑ **Customer myths.**

❑ **Practitioner's myths.**

Management myths

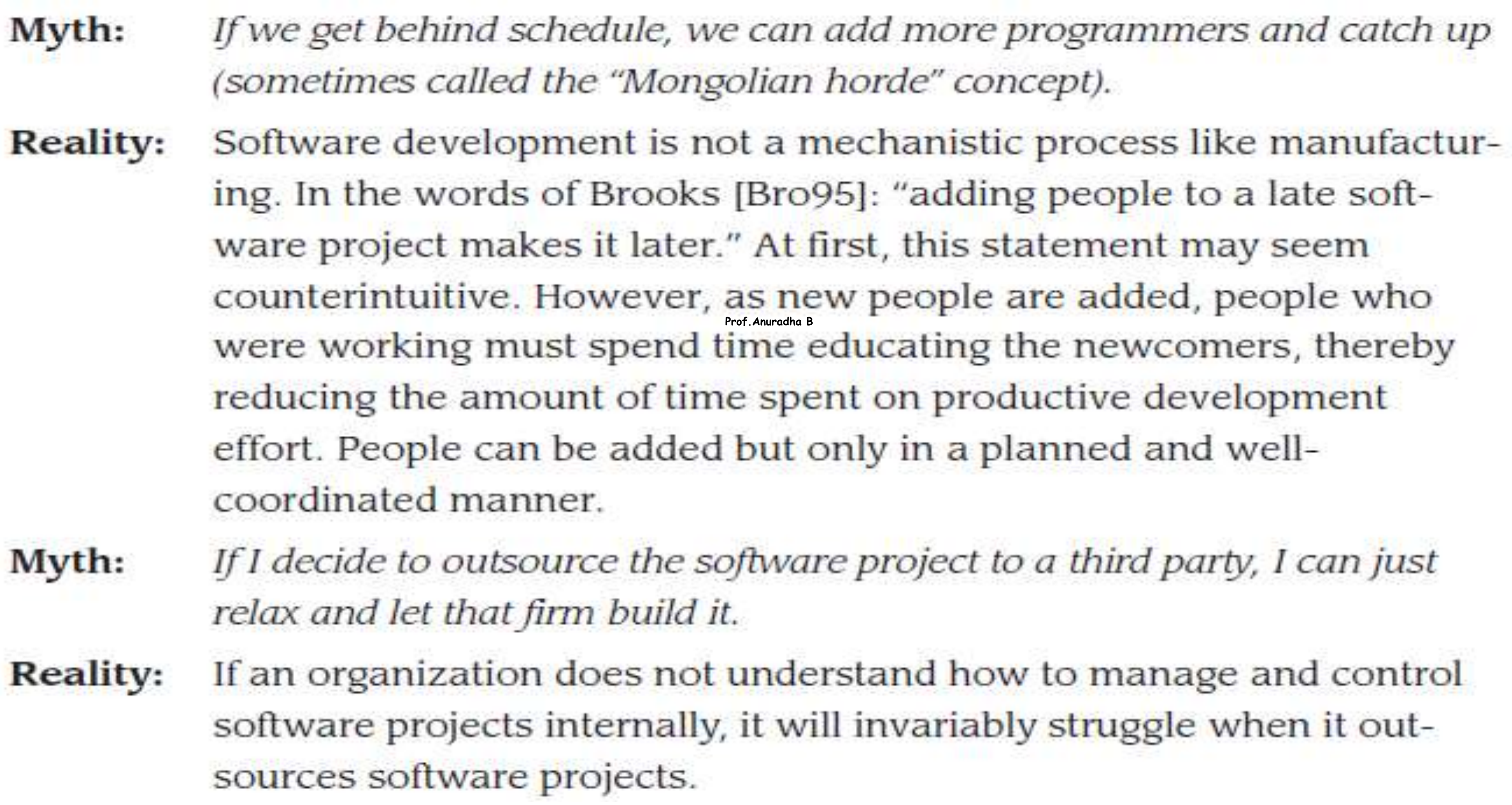
- Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).



Myth: *We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?*

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Management myths



Myth: *If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).*

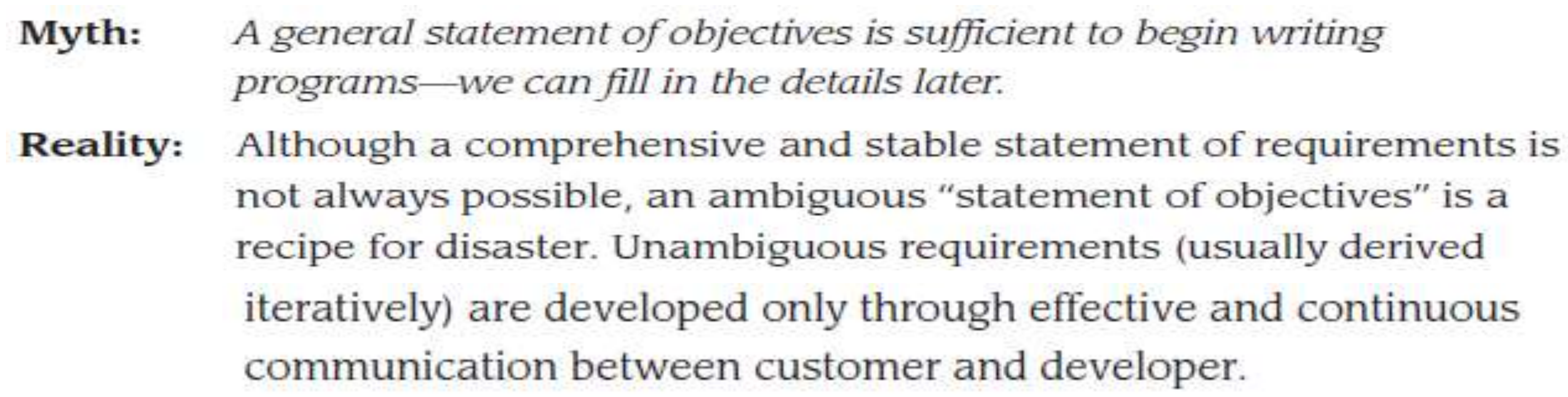
Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks [Bro95]: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth: *If I decide to outsource the software project to a third party, I can just relax and let that firm build it.*

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths

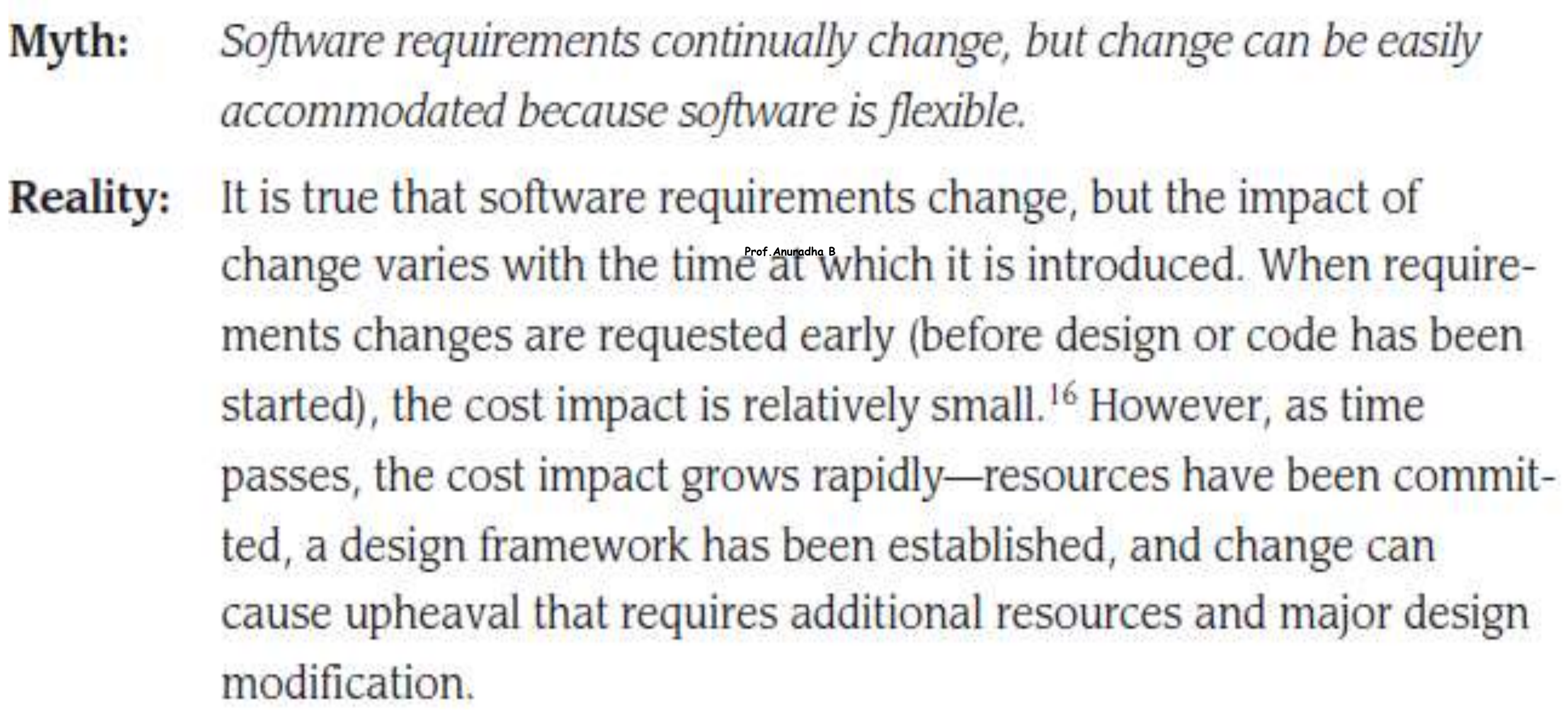
- A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.



Myth: *A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.*

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

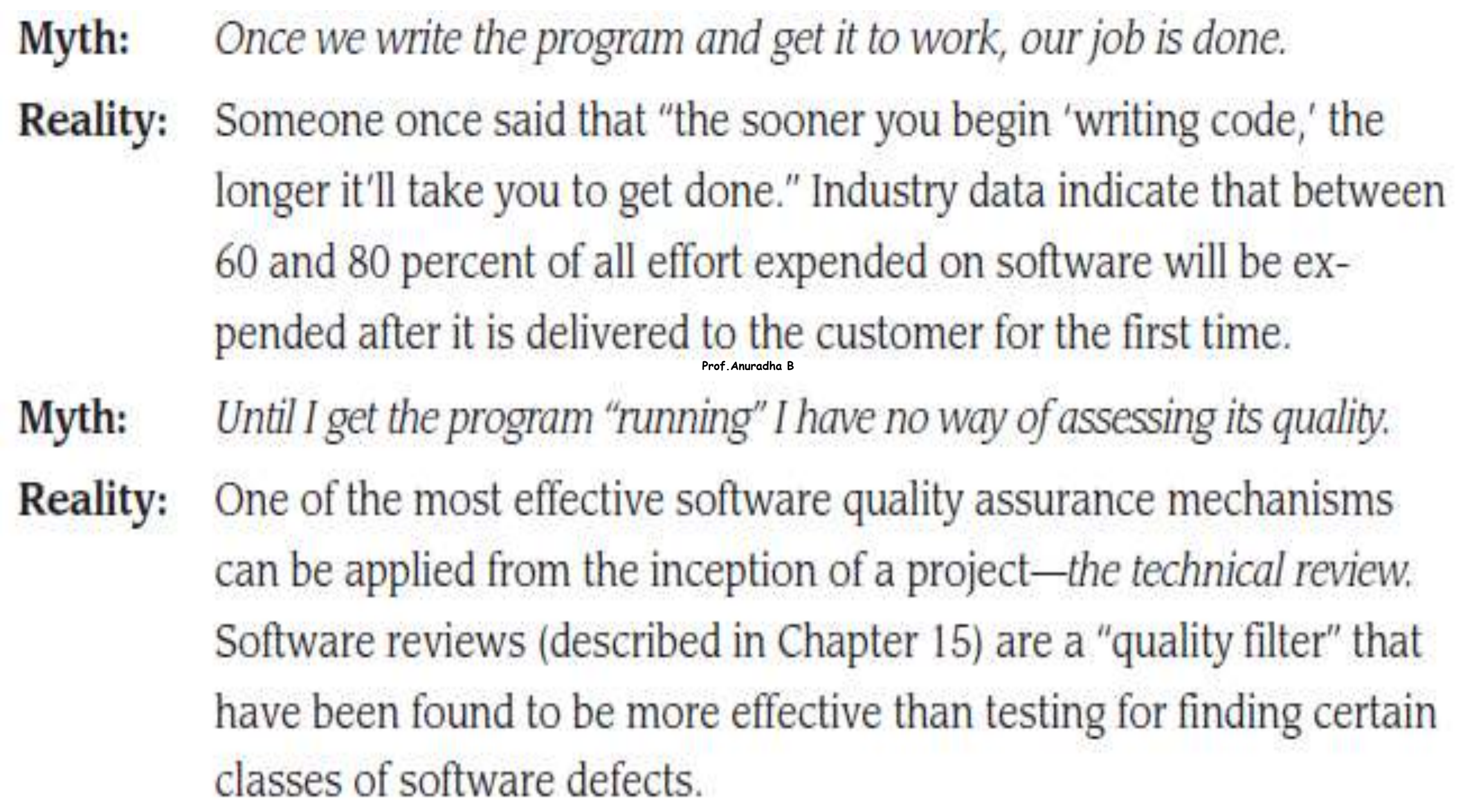
Customer myths



Myth: *Software requirements continually change, but change can be easily accommodated because software is flexible.*

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small.[16] However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

Practitioner’s myths



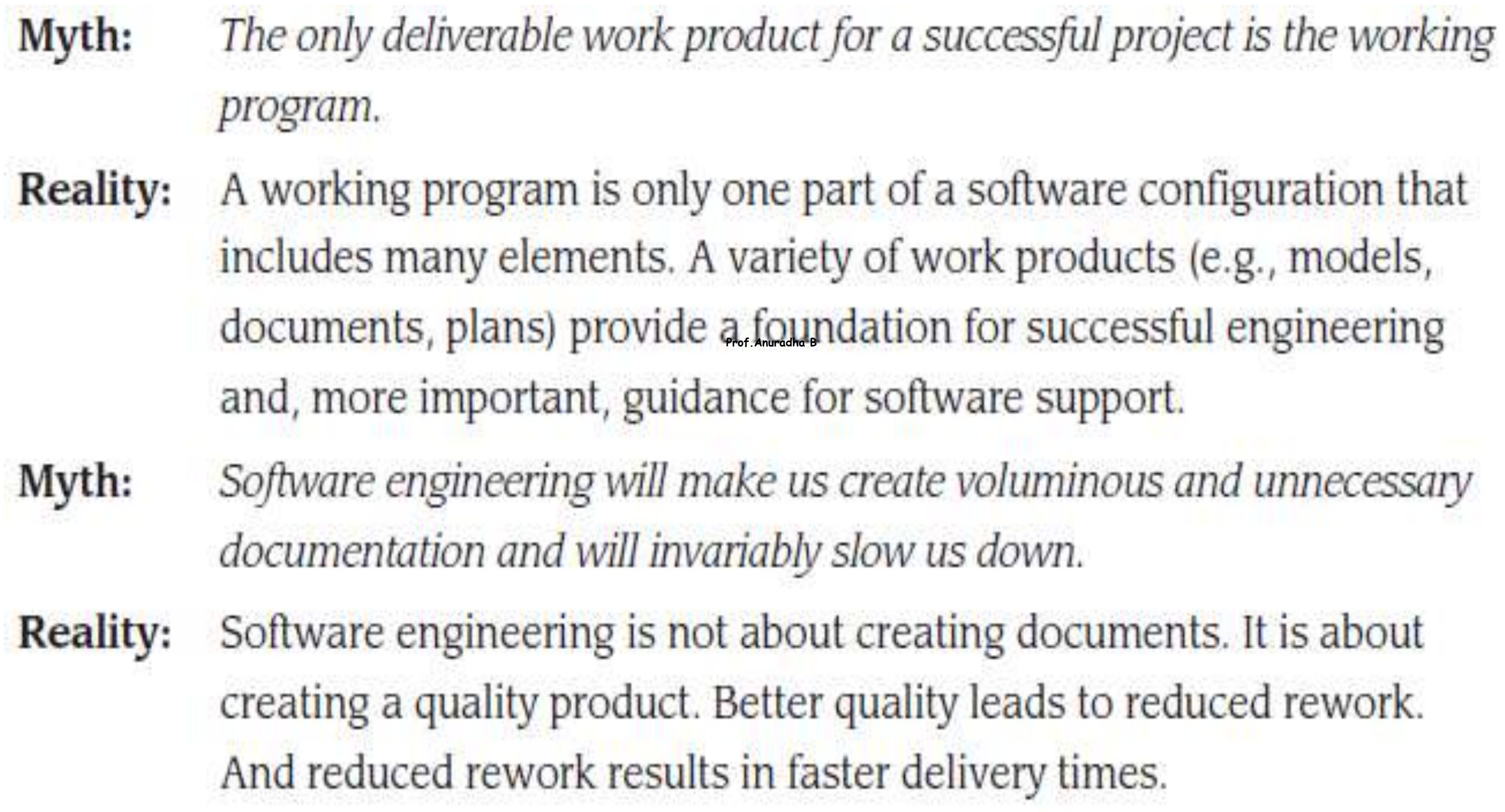
Myth: *Once we write the program and get it to work, our job is done.*

Reality: Someone once said that “the sooner you begin ‘writing code,’ the longer it’ll take you to get done.” Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: *Until I get the program “running” I have no way of assessing its quality.*

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—*the technical review.* Software reviews (described in Chapter 15) are a “quality filter” that have been found to be more effective than testing for finding certain classes of software defects.

Practitioner's myths



Myth: *The only deliverable work product for a successful project is the working program.*

Reality: A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

Myth: *Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.*

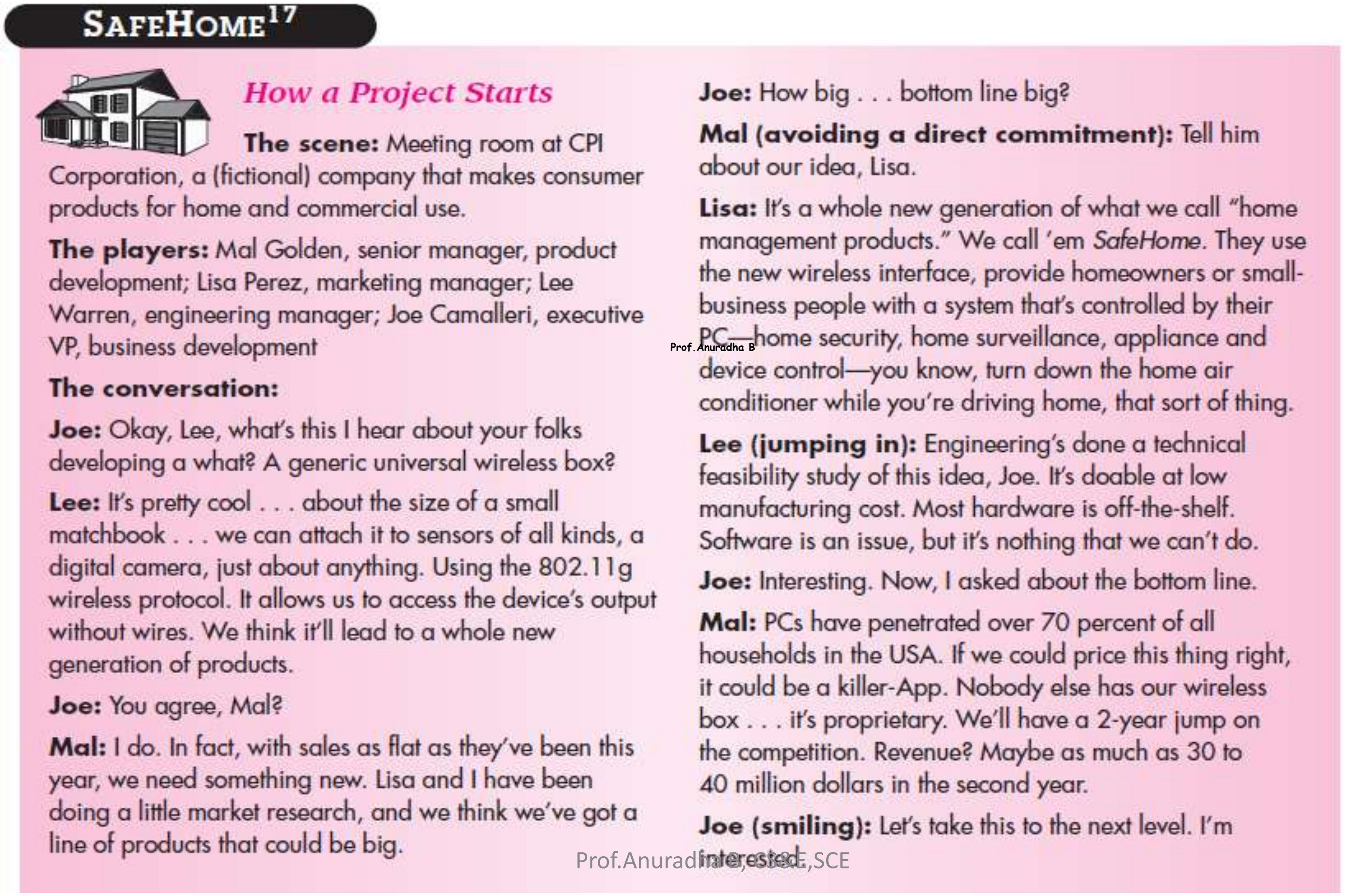
Reality: Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

1.7 HOW IT ALL STARTS

- Every software project is precipitated by some business need—the need to correct a defect in an existing application; the need to adapt a "legacy system" to a changing business environment; the need to extend the functions and features of an existing application; or the need to create a new product, service, or system.
- At the beginning of a software project, the business need is often expressed informally as part of a simple conversation. The conversation presented in the sidebar is typical.
- With the exception of a passing reference, software was hardly mentioned as part of the conversation. And yet, software will make or break the *SafeHome* product line. The engineering effort will succeed only if *SafeHome* software succeeds. The market will accept the product only if the software embedded within it properly meets the

  customer's (as yet unstated) needs.
- We'll follow the progression of *SafeHome* software engineering in many of the chapters that follow.

1.7 HOW IT ALL STARTS



SAFEHOME[17]

How a Project Starts

The scene: Meeting room at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

The players: Mal Golden, senior manager, product development; Lisa Perez, marketing manager; Lee Warren, engineering manager; Joe Camalleri, executive VP, business development

The conversation:

Joe: Okay, Lee, what's this I hear about your folks developing a what? A generic universal wireless box?

Lee: It's pretty cool . . . about the size of a small matchbook . . . we can attach it to sensors of all kinds, a digital camera, just about anything. Using the 802.11g wireless protocol. It allows us to access the device's output without wires. We think it'll lead to a whole new generation of products.

Joe: You agree, Mal?

Mal: I do. In fact, with sales as flat as they've been this year, we need something new. Lisa and I have been doing a little market research, and we think we've got a line of products that could be big.

Joe: How big . . . bottom line big?

Mal (avoiding a direct commitment): Tell him about our idea, Lisa.

Lisa: It's a whole new generation of what we call "home management products." We call 'em *SafeHome*. They use the new wireless interface, provide homeowners or small-business people with a system that's controlled by their PC—home security, home surveillance, appliance and device control—you know, turn down the home air conditioner while you're driving home, that sort of thing.

Lee (jumping in): Engineering's done a technical feasibility study of this idea, Joe. It's doable at low manufacturing cost. Most hardware is off-the-shelf. Software is an issue, but it's nothing that we can't do.

Joe: Interesting. Now, I asked about the bottom line.

Mal: PCs have penetrated over 70 percent of all households in the USA. If we could price this thing right, it could be a killer-App. Nobody else has our wireless box . . . it's proprietary. We'll have a 2-year jump on the competition. Revenue? Maybe as much as 30 to 40 million dollars in the second year.

Joe (smiling): Let's take this to the next level. I'm interested.