# Human Factors (15 pt)

## Effective Executive

**Five Ideas that you agree**

1. Brilliance, intelligence and knowledge need to be tied up with effectiveness in order to get good result in any task.

The author says, *“Brilliant men are often strikingly Ineffectual; they fail to realize that the brilliant insight is not by itself achievement. They never have learned that insights become effectiveness only through hard systematic work.”* He also adds, *“Intelligence, imagination, and knowledge are essential resources, but only effectiveness converts them into results. By themselves, they only set limits to what can be attained.”*

🡪 I completely agree with the idea that brilliance and intelligence doesn’t guarantee effective execution of tasks and being having a brilliant vision or insight is not an achievement unless the insight or the idea is put into execution through systematic work.

The second quote by the writer also emphasizes on the same idea that although knowledge, intelligence and brilliance bring in a great deal of value, they are not enough by themselves.

1. The executive has to himself push and work towards bringing effectiveness into his role.

The writer suggests the above statement by saying, *“Indeed, unless executives work at becoming effective, the realities of their situation will push them into futility*.” He also further adds, “*The fundamental problem is the reality around the executive. Unless he changes it by deliberate action, the flow of events will determine what he is concerned with and what he does.”*

*🡪* I agree to the idea that if the executive lets the events going around him determine his personality and his process of operation, he will not be able to incorporate effectiveness into what he does. To avoid this situation, as the writer suggests, the executive has to make special efforts and gain direct access to the reality outside of the organization.

1. There does not exist a single personality that would define the personalities of all the effective executives. They all differ in attitudes and personality but have a single trait of getting right things done common with each other. The writer mentions, “*I soon learned that there is no "effective personality."\* The effective executives I have seen differ widely in their temperaments and their abilities, in what they do and how they do it, in their personalities, their knowledge, their interests—in fact in almost everything that distinguishes human beings. All they have in common is the ability to get the right things done.”*

🡪I agree that there is no single type of people that can turn into an effective executive.

1. Effectiveness is all about following a practice so strictly that it turns into habit, any interested individual can make the effort of learning effectiveness. The writer highlights the very idea by stating that *“Effectiveness, in other words, is a habit; that is, a complex of practices. And practices can always be learned. Practices are simple, deceptively so; even a seven-year-old has no difficulty in understanding a practice. But practices are always exceedingly hard to do well.”*  I also agree with the fact that the writer highlights that practices can be learned by anybody , but to keep following the practice without fail is the difficult part.
2. I agree with the statement the writer makes regarding how time management is the most important practice to achieve effectiveness. He says, *“Effective executives know where their time goes. They work systematically at managing the little of their time that can be brought under their control.”* Controlling the time and utilizing it to the fullest is the key factor to develop a habit of being effective.

**The Idea you don’t or partially agree:**

The idea that I do not completely agree with is the statement made by the writer regarding priorities. The writer says that, “*Effective executives concentrate on the few major areas where superior performance will produce outstanding results. They force themselves to set priorities and stay with their priority decisions. They know that they have no choice but to do first things first—and second things not at all. The alternative is to get nothing done.”*  While it is very important to prioritize tasks and make sure the highly prioritized task is completed first, it is not always the best idea to concentrate all the resources and energy on one particular task. While I agree that the choice is to get first things first, it is always important to remember that there are other things to complete and the highly prioritized task is not the only task.

## Good to Great

**Five Ideas that you agree**

1. One of the most important factors in order for a company to grow is that the company should recognize what is exactly not going in their favor and they should immediately abort the idea. Jim Collins highlights the idea by saying, *“The good-to-great companies did not focus principally on what to do to become - great; they focused equally on what not to do and what to stop doing.”*
2. Revolutionary transformation that change the company from being good to being great does not happen in one instance. At the time of transformation, the process is not grand or significantly highlighted by a launch event. In most of the cases, the transformation happens because of the quality of the product being developed. The writer conveys the idea by stating, “*the good-to-great companies had no name, tag line, launch event, or program to signify their transformations. Indeed, some reported being unaware of the magnitude of the transformation at the time; only later, in retrospect, did it become clear. Yes, they produced a truly revolutionary leap in results, but not by a revolutionary process.”*
3. Good leaders did not begin by setting up vision, but instead formed a team of right people performing the right tasks. This idea stated by Jim Collins properly highlights the importance of human factors in the software development.
4. I also agree with the notion of “The Flywheel and the Doom Loop” that the writer presents. The breakthrough in any company result always from relentlessly pursuing a goal and working as a team towards achieving it. The companies that make the stage for giant launch and wrenching restructure do not usually succeed in making the leap from good to great company.
5. The companies that wish to transform from good to great need to stand the test of time. The companies need to demonstrate unwavering faith and face great difficulties and brutal realities. Discipline is also another factor that the companies need to have to deal with the brutal facts of current reality.

**The Idea you don’t or partially agree**

The writer states that the technology is never the primary reason of transformation from good to great company. However, I only partially agree with the statement given by the writer.

In my opinion, in the current time of evolution of technology, the technology is often a very crucial if not the primary reason to drive the company to greatness.

## Inside the mid of a master procrastinator

**Five Ideas that you agree**

1. It is true that procrastinators pull out all-nighters and somehow miraculously get the tasks done before the midnight, but the quality of the paper (in case of the speaker) or any other task will be negatively impacted in most cases.
2. There is part of the procrastinator’s brain that lives in the present and always is inclined to be a part of something that is easy and fun and does not allow the person to get started with the thing that is important.
3. The speaker slightly touches the adverse effects like anxiety, guilt, regret, feeling unproductive, feeling self-loath that comes after the procrastinator is over-indulgent in activities that are fun but should not be done at the moment. I completely agree with his idea and can relate to all those feelings.
4. When there is a deadline, the effects of the procrastination are contained.
5. When the situations do not have any tangible deadline, then the fact that panic drives the brain of the procrastinator to get to work, does not apply anymore. This situation can lead to long -term unhappiness and regret.

**The Idea you don’t or partially agree**

1. Non-procrastinators do not exist. It is true that non-procrastinators are rare to find but I have been in a close company of a few people, who get started as soon as they are assigned a task. This is also partially because these people are in the career fields that they are extremely passionate about. But, I have been friends with people who hate to wait till deadline to complete a task. They even are obsessed about completing the task two days before the deadline and are really excited as soon as they get assigned something. So, I do not agree that non-procrastinators do not exist.

# Design Factors (15 pt)

## The World and the Machine

Five questions and answers from this research paper

**Q1: Why is software development considered a kind of engineering?**

A1: The primary reason behind considering software development as engineering is because it is concerned with making physical devices to serve practical purposes in the world. In order to build the machine, we describe it and present it to the general -purpose computer which handles the attributes and the behavior of the machine based on the description.

**Q2: What are the different facets that reflect the relationship between the world and the machine?**

A2: The different facets are:

1. Modelling facet: The facet where the machine simulates the world
2. Interface facet: The facet where the world touches the machine physically
3. The engineering facet: The facet where machine acts as the engine of control over the behavior of the world.
4. The problem facet: The facet where the shape of the machine is influenced by the shape of the world and the shape of the solution is influenced by the shape of the problem.

Q3**:** The problem is not in the machine; and yet the machine can provide the solution to the problem. Which facet is the above statement related with?

A3: The statement is concerned with the interface facet. The machine can provide solution to the problem that is not in the machine because there is interaction at an interface between the machine and the world. The interaction refers to sharing of phenomenon and not passing message or remote procedural call. The participation in the interaction is not symmetrical, one party may have the power to initiate the event and the other may or may not have the power to inhibit it.

Q4: **How are the terms requirements, specifications and programs defined in the paper?**

A4**: Requirements** are solely concerned with phenomenon in the world. They are the effects that the customers want to see being implemented in the world and not only on the machines. **Programs** are concerned solely with the machine phenomena. The purpose of the programs is to describe those properties and behaviors of the machine that will be able to cater to the satisfaction of the customers. Programs are the descriptions of that machines that the general-purpose computers will be able to emulate, and they must be cast in terms that the can be interpreted by the computers

Q5: **How is the gap between requirements and programs bridged? Explain.**

A5: Specifications bridges the gap between requirements and programs. Specifications are concerned solely with the shared phenomenon. As specifications are concerned with both the phenomena of the world and machine, they are both requirements and programs. However, they are satisfiable neither as a program nor as a requirement. It is unsatisfactory as a requirement as the requirements are not always confined to the common area where the world meets the machine. They are unsatisfactory as programs because they may not be executable.

## Foundations for the Study of Software Architecture

Five questions and answers from this research paper

**Q1: List the philosophical foundation laid before presenting the model of software architecture?**

A1: The philosophical foundations are:

1. Understanding the existing disciplines and developing an intuition about software architecture by drawing analogy to them.

2. Proposing a context for software architecture in a multi-level product paradigm

3. Providing some motivation for software architecture as a separate discipline

**Q2: In the context of the architecture, how does the writer define requirements, architecture, design and implementation?**

A2:

**Requirements**: Concerned with the determination of the information, processing, and the characteristics of that information and processing needed by the user of the system

**Architecture:** Related to the selection of architectural elements, the interaction among them, the constraints on those elements. Architecture is also concerned with how the interaction among elements plays a role in satisfying the requirements and serving as a basis for design.

**Design:** Concerned with the interfaces and modularization of design elements, algorithms, procedures, their data types required to satisfy the requirements and form a architecture

**Implementation:** Concerned with how the algorithms and data types that satisfy the design, architecture and requirements are represented

**Q3: What do the writers look forward to be able to do with an architectural specification?**

A3:They want to be able to:

1.Prescribe the architectural constraints to the desired level : They want to be able to separate necessity from luxury and clearly determine the degree of relativeness and absoluteness

2.Separate aesthetics from engineering: They want to have a clear distinction of what is load-bearing and what is decoration

3.Express different aspects of the architecture in an appropriate manner

4. Perform dependency and consistency analysis: They also want to be able to determine how the architecture, requirements and design are interdependent upon each other. They want to see if there is a lack of consistency between several architectural styles.

**Q4: What was the model of the proposed software architecture and define the terms associated with it?**

A4: The model for the software architecture proposed by the writer is:

Software Architecture = {Elements, Form, Rationale}

**Elements**: There are three kinds of elements:  processing elements; data elements; and  connecting elements.

* Processing elements supply the transformation on the data elements
* Data elements contain the information that is used and transformed
* Connecting elements hold the different pieces of the architecture together.

**Form:** The architectural form consists of weighted properties and relationships.

* The weighting indicates importance of the property or the relationship
* Properties are used to constrain the choice of architectural elements
* Relationships are used constrain how the different elements may interact and how they are organized with respect to each other in the architecture.

**Q5: How are process and data interdependent on each other?**

A5:There are some properties that distinguish one state of the data from another and those properties are the result of some transformation produced by some processing element. In this way, the data and process interdependent on each other. On connecting, the interdependence can be seen more vividly as the connecting elements are the mechanisms for moving data around from processor to processor.

## Designing Software for Ease of Extension and Contraction & On the Criteria to be used in decomposing systems into modules

Five questions and answers from these research papers

## Paper 1: On the Criteria to be used in decomposing systems into modules

Q1**: What were the two different criteria for getting the two modularizations in the paper.**

A1: The first decomposition was based on the criterion that each major step in the processing was made a module. A flowchart can be made and followed to get the first decomposition.

The second decomposition was based on the criterion of “information hiding” such that the modules no longer correspond to steps in the processing. Every module in the second decomposition has the knowledge of a design decision that it hides from all others.

Q2**: With respect to changeability, how did the modularization 1 and 2 compare?**

A2: The writer mentions that with changeability in design which is inevitable, in the first modularization we might have to change every single module depending on what the change is. The same change would not require change in the second modularization and the same modules could be used for different design changes. In the second decomposition, many potential changes are confined to a single module

Q3: **What is the potential drawback to the second decomposition and how does David Parnas recommend avoiding it?**

A3: In the paper he mentions that the second decomposition could prove to be less efficient in one circumstance. According to him, if each of the functions is implemented as a procedure with an elaborate calling sequence, there can be a great deal of calling due to repeated switching between modules which would lead to inefficiency.

He recommends using a tool by means of which programs might be written as if functions were subroutines but assembled by appropriate implementation in order to make the efficient use of the second decomposition.

Paper 2: Designing Software for Ease of Extension and Contraction

Q4**: What are the four categories of obstacles encountered in trying to extend or shrink a system?**

A4: The categories are:

* 1. Excessive Information Distribution: If the program is written assuming the feature may or may not exist, this obstacle is encountered.
  2. A Chain of Data Transforming Components: If one component in a program structured with chain of components is not needed, that code is often hard to remove because the output of its predecessor is not compatible with the input requirements of its successor.
  3. Components That Perform More Than One Function: One of the common errors made by the programmers is to combine two simple functions into one because the function might look like they are too simple to separate.
  4. Loops in the “Uses” Relation: Usually Programmers are encouraged to use the work of other programmers as much as possible. But unless some restraint is exercised, one may end up with a system in which nothing works until everything works.

Q5: **What are the parts of the methodology that would help software engineers develop systems that do not include the obstacles discussed in question no 4?**

A5: The parts of the methodology suggested by David Parnas are:

1. Requirements Definition: Identifying the Subsets First
2. Information Hiding: Interface and Module Definition
3. The Virtual Machine (VM) Concept
4. Designing the "Uses" Structure

# Technology Factors (15 pt)

# **ANSI Common Lisp Chapter 1**

## Storyline

Why even after many years of development, Lisp is so popular and what makes Lisp distinctive from other languages? The chapter provides answers to all the questions. The special feature that sets Lisp apart from other programming languages is that it is designed to evolve. Closures are one of the abstractions found in Lisp that is not found in any other programming language. Another unique and valuable feature of Lisp is that Lisp programs are expressed as Lisp data structure which allows users to write programs that write programs. Since Lisp language is made out of the same functions and macros as the user’s programs, extending Lisp is the standard convention instead of writing program. The program is written down toward the language and the language is built up towards the program. It is possible to write a prototype in less time than to write a specification, which is where the term “rapid prototyping” began from.

## Takeaways

1. With macros, closures, and run-time typing, Lisp transcends object oriented programming.
2. Working bottom-up is also the best way to get reusable software
3. Lisp allows you not just to write more sophisticated programs, but to write them faster.

## Arguments

I partially agree with the writer on the quote “Strange as it sounds, less planning can mean better design”. I agree that painstaking and meticulous planning is not always necessary, but I think there should be a necessary amount of planning and defining a vision, before the implementation is carried out. My point of view is in lieu of the writer’s view when he says, “Planning is a necessary evil. It is a response to risk: the more dangerous an undertaking, the more important it is to plan ahead. Powerful tools decrease risk, and so decrease the need for planning. The design of your program can then benefit from what is probably the most useful source of information available: the experience of implementing it.” If anything means better design, it has to be the experience of implementing it multiple times.

# **Chapter -10 Programming Languages Explained (Hackers and Painters)**

## Storyline:

The chapter explain in a very simplistic but effective manner different types of programming languages including the advantages of each type. The writer begins by defining machine language as a list of things a computer can do. In assembly language, the same purpose is fulfilled but by using more programmer- friendly names. The more convenient language which is a high-level language is usually fed to the compiler. But instead of compilers, interpreters can also be used to implement high level languages, but the interpreter executes the corresponding machine language commands one piece at a time. The use of high-level language makes program easy-to-read, less verbose and portable. Open source software provides the user with the source code which makes it easy to understand the code. The user gets more control and also have the capability of fixing a bug if the software is open source. The writer also mentions about the ongoing debate between programmers, where one group argues that the language should prevent users from trying to implement some stuffs and other group that argues that programmers should be allowed to do whatever they want. There is also a similar discussion going on about the adaptation of object-oriented feature and the degree of the object orient in different languages. The writer concludes on a positive note where he believes that the evolution of different kinds of languages could actually end up producing the renaissance equivalent language.

## Takeaways

1. Source code is the high-level language fed to the compiler.
2. The machine language translated by the compiler is called the object code.
3. A language can be very abstract but offers the wrong abstractions. So, the statement that higher-level the language, the better it is not always true.
4. In a statically-typed language, the user needs to know the kind of values each variable can have at the time the program is being written and in dynamic typing any variable can be set to any value.

## Arguments

1. The biggest advantage of having an open source software is the feedback that is available from experts all over the world, which would eventually contribute in making the software better. The writer delivers the same sentiment by mentioning that open source software is like a paper that has been subject to peer review. But, the writer does not highlight that the source code software is open to the community and sometime can rely on the community to take control or fix a bug. It is also open to malicious users who can exploit its vulnerability.
2. I agree with the writer where he states that dynamic typing can give more flexibility to the user and its good when the program does not direct the user.

# **Chapter -14 The Dream Language (Hackers and Painters)**

Storyline

In this chapter the writer tries to find the reason behind popularity of the programming language. He further digs deeper to find out if it is worth trying to define a good programming language.

The writer mentions that there is a difference in opinion between hackers and most programming language designer regarding the idea of a good programming language.

The writer believes that even though expert hackers are among the minority, they can recognize a good programming language and rest of the programmers are usually influenced by their choice.

He also mentions that in order to gain initial users, the best approach is a trojan horse which means building an application that people want but use the newly written language to do so.

The writer then goes ahead to mention and explain in brief some factors like succinctness, hackability, Libraries, Efficiency, Time and Redesign that are liked in a programming language by the hackers. The writer in conclusion comes up with some features that the dream language of the hackers would possess: the dream language would be clean and terse; its syntax would be brief to a fault. The language would have a small core, and powerful, highly orthogonal libraries, and would-be built-in layers. The dream language of the hackers would be open design on top of being open source

Takeaways:

1. Whether or not a language has to be good to be popular, I think a language has to be popular to be good.
2. Succinctness is one place where statically typed languages lose
3. The writer makes interesting observation regarding the factors. For succinctness, he says that the most important kind of succinctness comes from making the language more abstract.
4. Similarly, for hackability, he says that although the hacker would in his lifetime end up subverting the intended model of things once or twice, but the leverage that the program provides to do so is what is admired by them

Arguments:

The writer in the chapter mentions that a good programming language should have features that make the kind of people who use the phrase “software engineering” shake their heads disapprovingly. The tone of the statement has a negative annotation towards the disapproval of the software engineers. But in my opinion, first of all the software engineers would approve of the programming language features that the writer talks about. Secondly, it is very important to take feedback from anyone to make any design of the programming language better, and the reason behind disapproval of the language could help in improving a feature in the new language.

## A rational design process

Five questions and answers from this research paper

**Q1: Why does the writer say that the software design process will always be an idealization?**

A1: The writer gives the following reasons as the answers to the statement:

1. The people who commission the building of the software system do not exactly know and in often cases cannot express it.
2. Many details other than the requirements become known to us as we proceed with the implementation.
3. Despite having known the relevant facts, human beings are bound to make errors as they are not capable of comprehending the plethora of details required to design and build a perfect system that is free of errors.
4. Projects are always subject to change because of external reasons and the change sometimes end up invalidating previous design decisions.
5. Human errors cannot be avoided if humans are behind the system design.
6. Implementing preconceived ideas sometimes does not work in favor of the project.
7. Selecting software that was used for the previous project due to economic constraint might not be the perfect choice for the current project.

**Q2: Despite knowing that a rational process is an idealization, why is the description of such process necessary?**

A2. The description is necessary so that the software developers follow the process as closely as they can. The reasons are listed below:

1. The understanding if the ideal process will be a guidance for designers to begin the process.
2. The actual design will turn out close to a rational design if the description for the process is strictly followed.
3. It will act as a standard for several different kinds of projects in a single organization.
4. It will also be a point of comparison to identify the progress of the project.
5. It will make the external review of the project easy.

Q3: **What should be included in the description of the development process?**

A3: The process description should be in the form of work products and should answer what is the product that should be worked on next and what are the criterion that should be satisfied by the product. The description must also include the details of the type of persons to be working on the product and the information that can be used in the work.

Q4**: List the steps of the rational design process.**

A4: The steps are:

1. Establish and document requirements
2. Design and document the module structure
3. Design and document the module interfaces
4. Design and document the module hierarchy
5. Design and document the module internal structure
6. Write programs
7. Maintain

Q5**: What are the bad and good practices of documenting a project.**

A5: Including the following will make the documentation poor and thus the following practices should be avoided:

1. Poor Organization
2. Boring Prose
3. Confusing and inconsistent terminology
4. Myopia

The writer makes some suggestions in order to improve the quality of the documentation. Some of the suggestions are:

1. Apply systematic checks to increase completeness and consistency
2. Do not duplicate facts and there must be a single place that states every fact in the document
3. The principle guiding the software design, i.e, the principle od separating the corners should be followed for documentation as well.