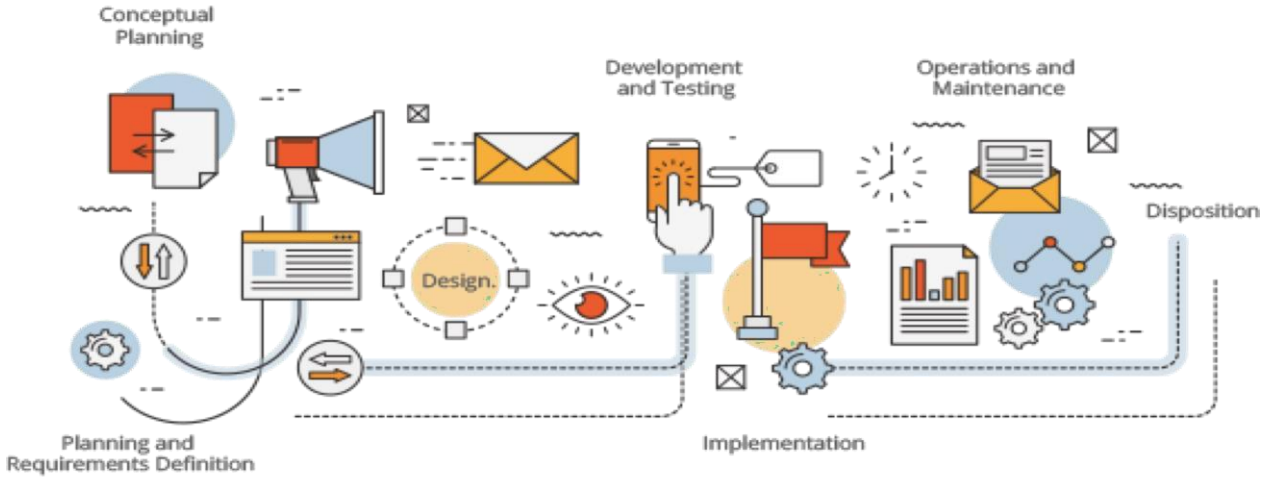
Software Engineering

Introduction



About Me

- Assistant Professor
- PhD(SE) – FAST-NUCES (2021)
  - Full Funded Scholarship
- Research Associate – Quest UAV Dependability Lab (2018-current)
- Professional Trainer of SCM, Web Test Automation, DevOPs
- Over 9+ years of Industry and Academia experience
  - Expertise: Web Test automation, Software Testing , Web application product lines, Java (SE & EE), UML, Model-driven Software Engineering, DevOPs.
- MS(SE) – FAST-NUCES (2015-2017)
  - Full Funded Scholarship
  - Distinction
- Instructor – FAST-NUCES (2014-2018)
- BS(SE) – IIUI (2010-2014)

Course Objectives

- Develop understanding of concepts and methods required to construct large software systems
- To familiarize students with traditional and modern process models.
- To disseminate knowledge about core activities such as requirement engineering, software architecture and design, testing and maintenance.
- Evaluate and implement approaches for a particular context.
- To familiarize students with the tools and techniques to develop better software.
- Develop skills to plan and manage real life software projects

Agenda

- Course Mechanics
- What is software engineering?
- Why is software engineering?
- What do software engineers do?
- What are the different types of software systems?

Course Mechanics

Text and reference books

- Software Engineering, Sommerville, Ian Addison Wesley
- Software Engineering and Testing, b.B Agarwal, P.Tayal, m.Gupta, Jones and Bartlett Publisher.
- Software Engineering: A practitioner's approach, Pressman, R.S & Maxim B., McGraw-Hill
- Software Architecture Design illuminated
- Standards:
  - Software requirement specification (IEEE)
  - Requirement Engineering (IREB)
  - Software Architecture (iSAQB)

What, When and Where

- Lectures (2x a week)
  - Monday and Wednesday (Section A, B)
  - Tuesday and Thursday (Section C)
  - Practice sessions on various case studies
- Assignments, Quizzes and Project
  - 3,4 assignments
  - 4,5 quizzes (both announced and unannounced)
  - 1 project

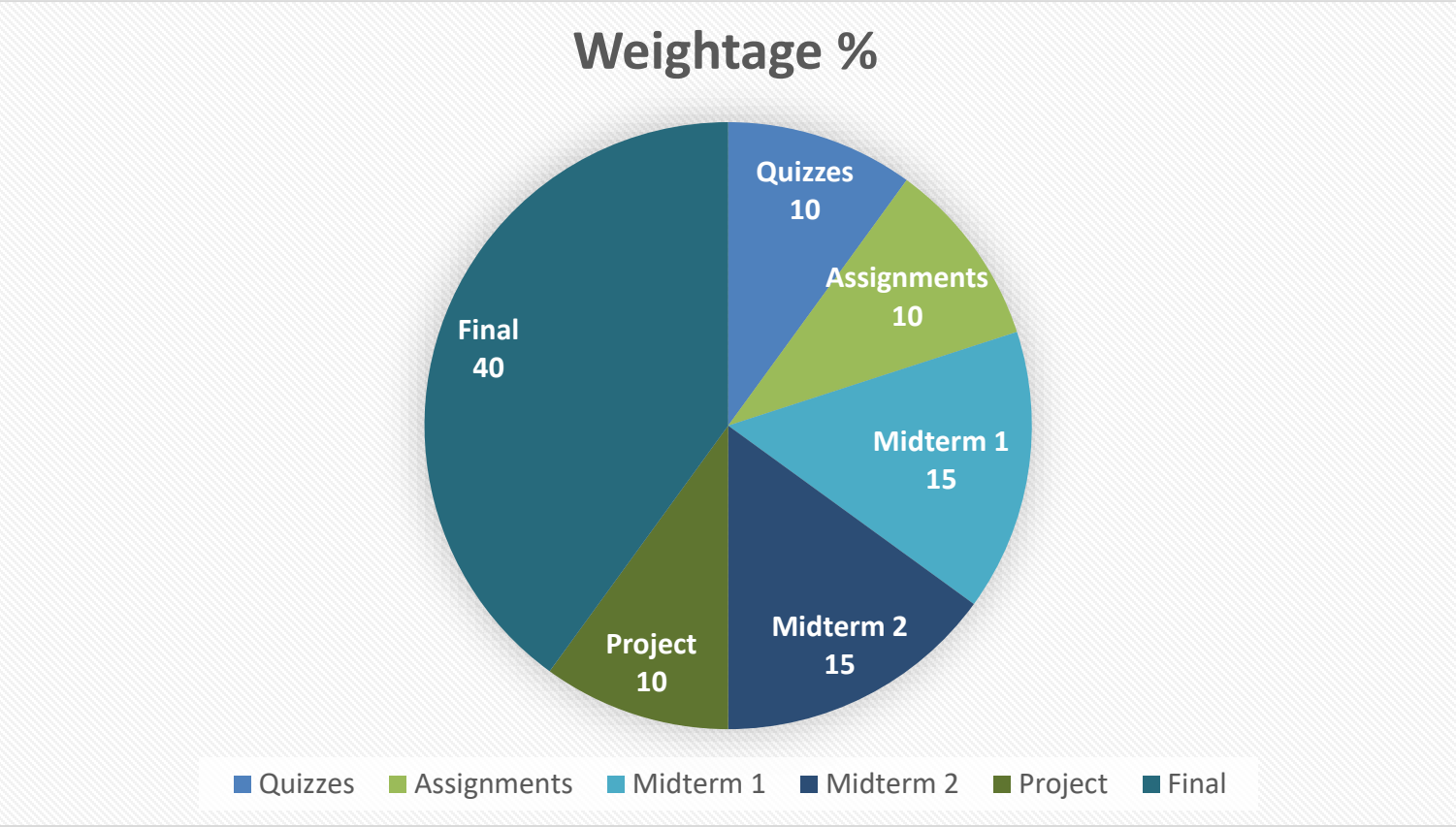
- Exams
  - Two Sessional and Final Exam

Evaluation Breakdown

Grading Policy

Absolute Grading

Theory breakdown



Other Rules

- Submission
  - Submission on or before deadline (GCR/in class)
  - I don’t extend deadlines
- Plagiarism
  - In case of plagiarized content, you will be awarded zero in that particular assignment/quiz
- Punctual
  - Come to class prepared and in time

Communication

- Office Hours
  - Monday and Wednesday
  - 11:30-01:00pm
  - 205B – 2nd floor (C Block)

- GCR
  - GCR code: **iopqite**

Course topics

- You will learn about
  - Introduction to software engineering and its phases
  - Process models
  - Project planning
  - Requirement engineering
  - Architectural styles
  - Project and software quality
  - Testing
  - Project management

What is Software Engineering? (Brief History)

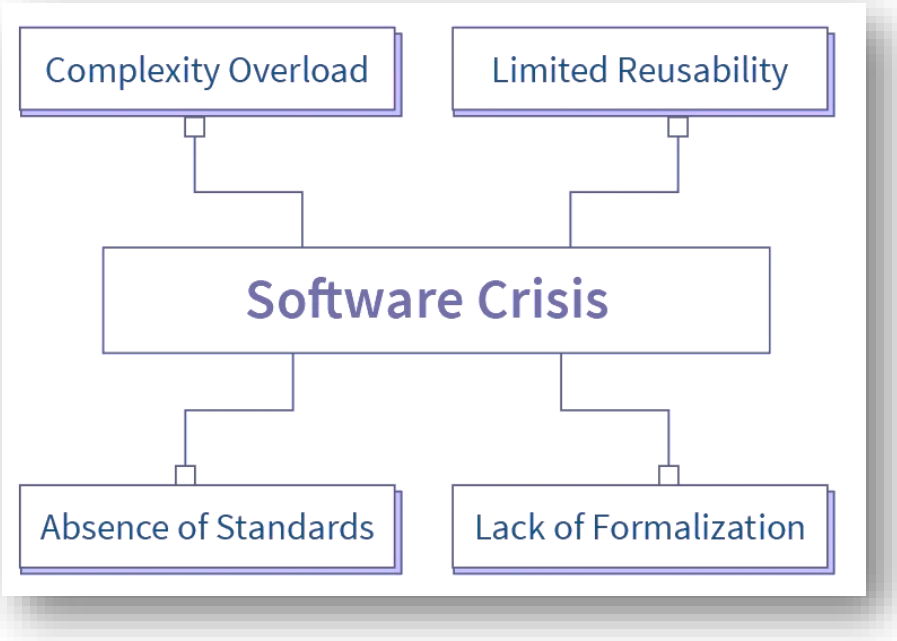
- The **NATO Software Engineering Conferences** held in **1968 (in Garmisch, Germany)** and **1969 (in Rome, Italy)** are widely recognized for addressing what was termed the **"software crisis.“**

- **Goal**: Goal was making software development an engineering discipline based on scientific principles and rigorous practices

- **Purpose:** The conference brought together leading experts to discuss the growing challenges in software development. At the time, software projects often faced:
  - Delays
  - Cost overruns
  - Unreliable performance
  - Difficulty in scaling for complex systems

This crisis led to a number of issues, including:

- Projects running over time and budget.
- Software was becoming more inefficient and error prone.
- Software was frequently failing to meet the requirements for the project.
- Software was increasingly difficult to extend and maintain.
- Some projects failed before they could even deliver workable code at all.



What is Software Engineering? (Brief History)

> The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
>
> ~Edsger Dijkstra

Takeaway:

- Advancements in hardware don't automatically solve software problems; they often create new ones.
- To keep up, we need better methods, tools, and practices in software design and development to handle this growing complexity.

Another quote from a workshop participant sums up the state of the art at the time:

(software is built like Wright brothers build airplanes)

"Today, we tend to go on for years, with tremendous investments, to find that the system, which was not well understood to start with, does not work as anticipated. We build systems like the Wright brothers built airplanes build the whole thing, push it off the cliff, let it crash, and start over again."

- Quote from the report:
  - "The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for … the types of theoretical foundations and practical disciplines that are traditional in established branches of engineering."

What is Software Engineering? (Brief History)

- **Participants:** Attendees included academics, industry experts, and government representatives from NATO countries, collaborating to analyze the challenges and propose solutions.

- **Outcomes:** The conference highlighted key issues such as:
  - The importance of modular design
  - The need for better tools and programming languages
  - Improved management practices for large-scale projects

- The NATO conferences were a pivotal moment in the history of computing, shaping how software development is approached today.

- Since then, software engineering has progressed from an aspiration to an established field.

- There is a growing body of knowledge, incorporating hard won lessons from both successful and unsuccessful projects.

- Professional societies, ACM and IEEE, have published curricula for undergraduate programs and courses. Thousands of books have been written about the subject.

Key Points

1. **Relevance of the Problem:**
   - The issues faced during the "software crisis" (e.g., unreliable systems, delays, cost overruns) still trouble many organizations today.
   - This highlights that even as technology advances, the **core problems** often stem from poor design and architecture, not the technology itself.

2. **The Solution:**
   - The solution to the software crisis was identified early in the history of software engineering.
   - This solution is **co**
     - **Component-**
     - **Separation c**
       reduce comp

3. **Timeless Principles:**
   - The fundamental p
   - These principles w

Takeaway:

The essence of overcoming the software crisis lies in applying timeless principles of good design and architecture. Advanced tools and technologies won't solve the problem alone; engineers must prioritize clarity, modularity, and simplicity in their work.

4. **The Vicious vs. Virtuous Cycle:**
   - Poor design and architecture choices create a **vicious cycle** of complexity, confusion, and inefficiency.
   - Good design practices foster a **virtuous cycle** of clarity, reliability, and scalability.

5. **Modern Challenges:**
   - Despite having more advanced tools and technologies today, engineers often face greater confusion because these tools can add complexity if not used wisely.
   - The passage encourages focusing on **avoiding common mistakes** and sticking to proven principles to reduce confusion and ensure success.

In this course, I'll try to teach you about the state of the art in software engineering including its underlying principles and what experts regard as its best practices and essential concepts.

Definitions of Software Engineering*

- “The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines" [Bauer 1972]

- "Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems." [SEI 1990]

- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [IEEE 1990]

Computer Science vs. Software Engineering

The key difference here is that computer science education program focuses on the science behind making computers work, while software engineering applies those scientific and mathematical principles to the building, designing and implementation of hardware and software programs.

What do software engineers do?

- Analyze users' needs and then design, test, and develop software to meet those needs
- Recommend software upgrades for customers' existing programs and systems
- Design each piece of an application or a system and plan how the pieces will work together
- Create a variety of models and diagrams (such as flowcharts) that instruct programmers how to write software code
- Ensure that a program continues to function normally through software maintenance and testing
- Document every aspect of an application or a system as a reference for future maintenance and upgrades
- Collaborate with other computer specialists to create optimum software

Growing need for software engineers

- Employment expected to grow 17% (2014 – 2024)
  - Much faster than the average for all occupations

- Median annual wage for software systems developers was $100,690 in 2015
  - Top 10% earned more than $153,710
  - Median for application developers was $98,260

- Qualities and Skills
  - Analytical, creativity, problem solving
  - Communication, customer-service, inter-personal
  - Big picture, attention to detail

Definition of Software Engineering

- **Software Engineering** is
  - The art and science of
  - developing reliable software systems that
  - address customer needs,
  - subject to cost and schedule constraints

CUSTOMER-RELATED

TEAM-RELATED

Art & Science

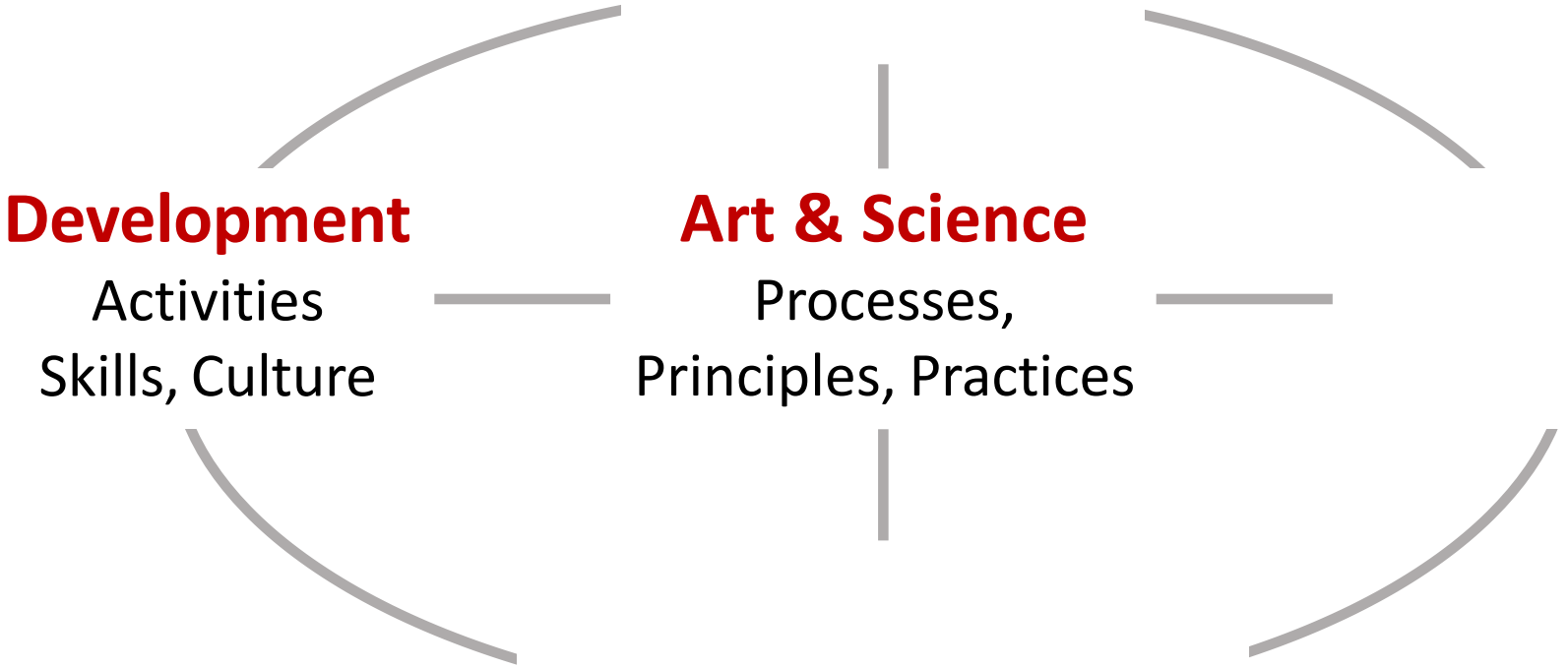
Processes,
Principles, Practices

PRODUCT-RELATED

ORGANIZATION-RELATED

CUSTOMER-RELATED

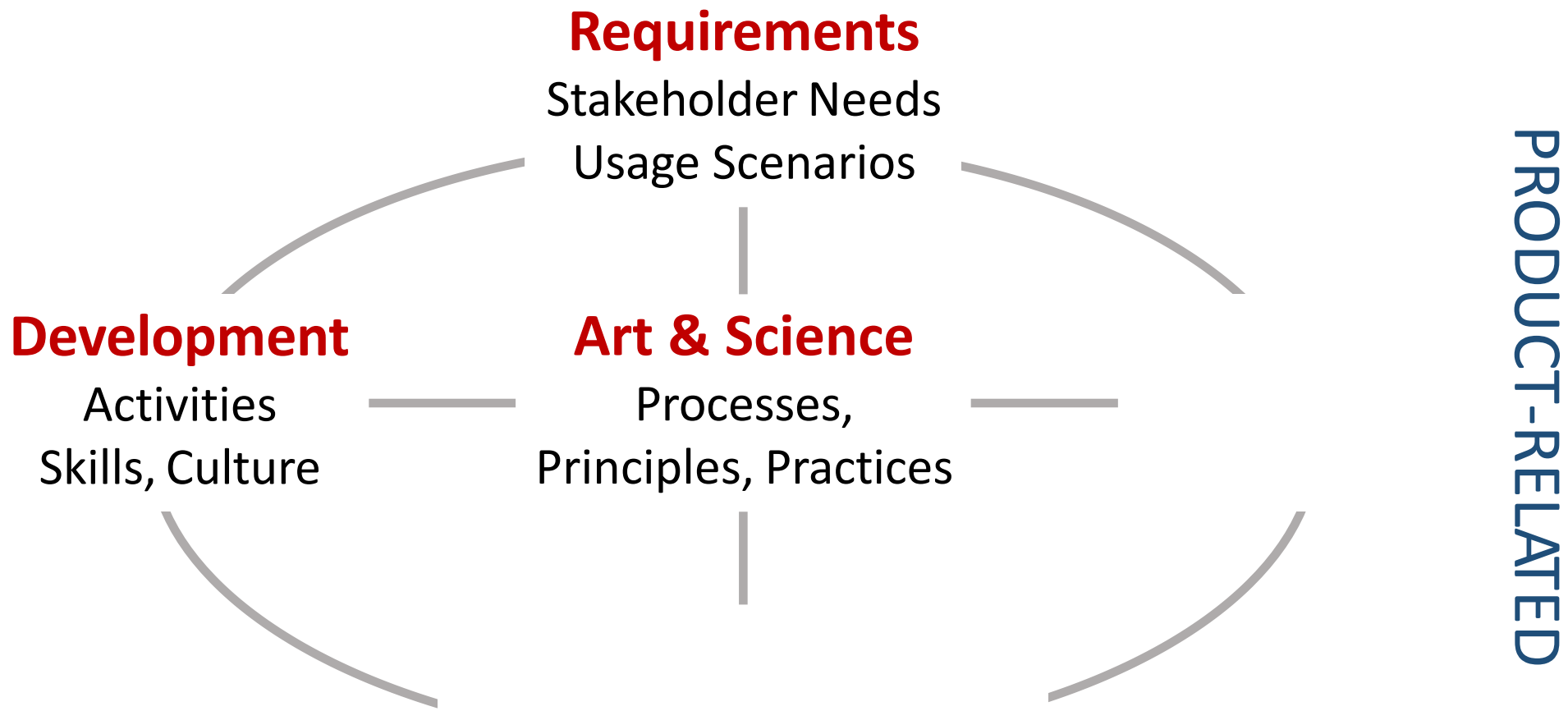
TEAM-RELATED



PRODUCT-RELATED

ORGANIZATION-RELATED

CUSTOMER-RELATED

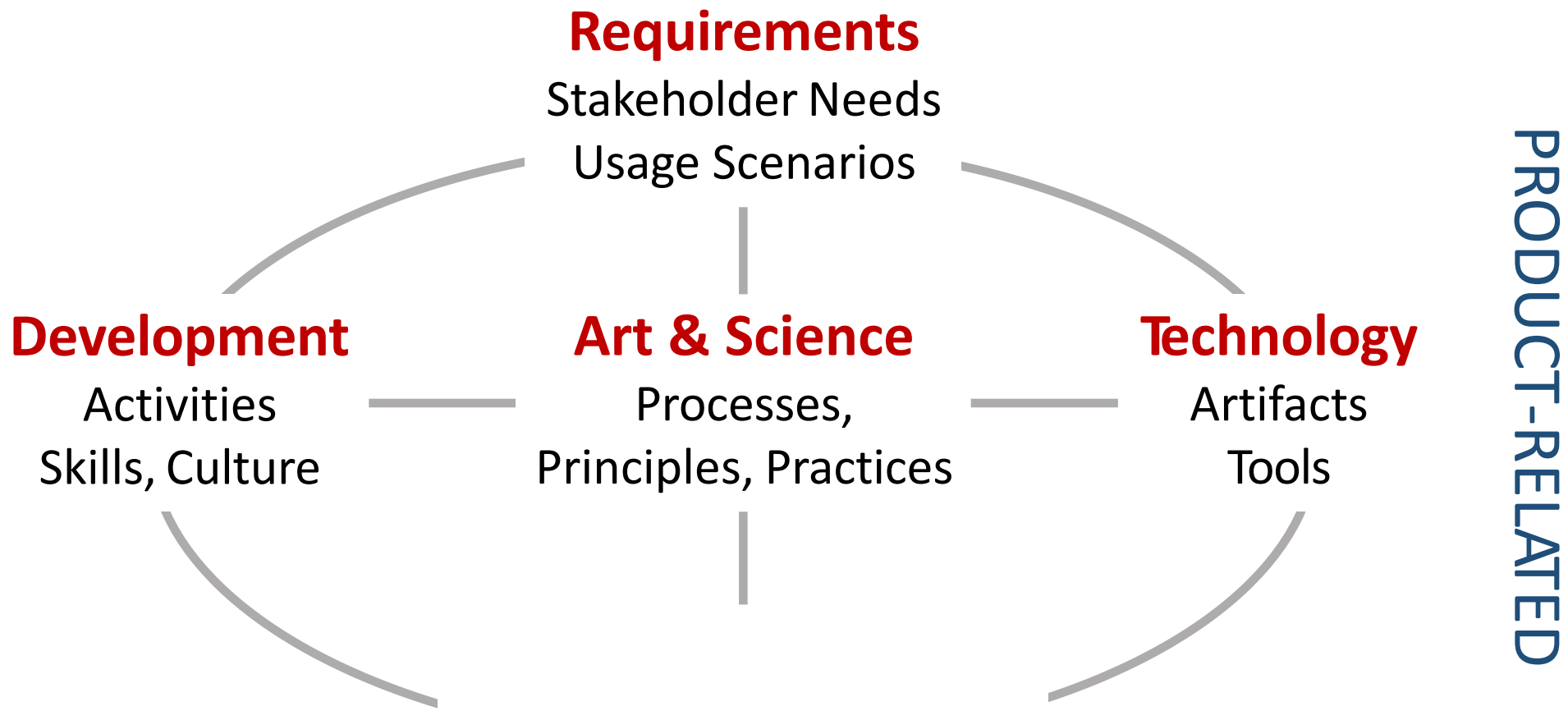


TEAM-RELATED

PRODUCT-RELATED

ORGANIZATION-RELATED

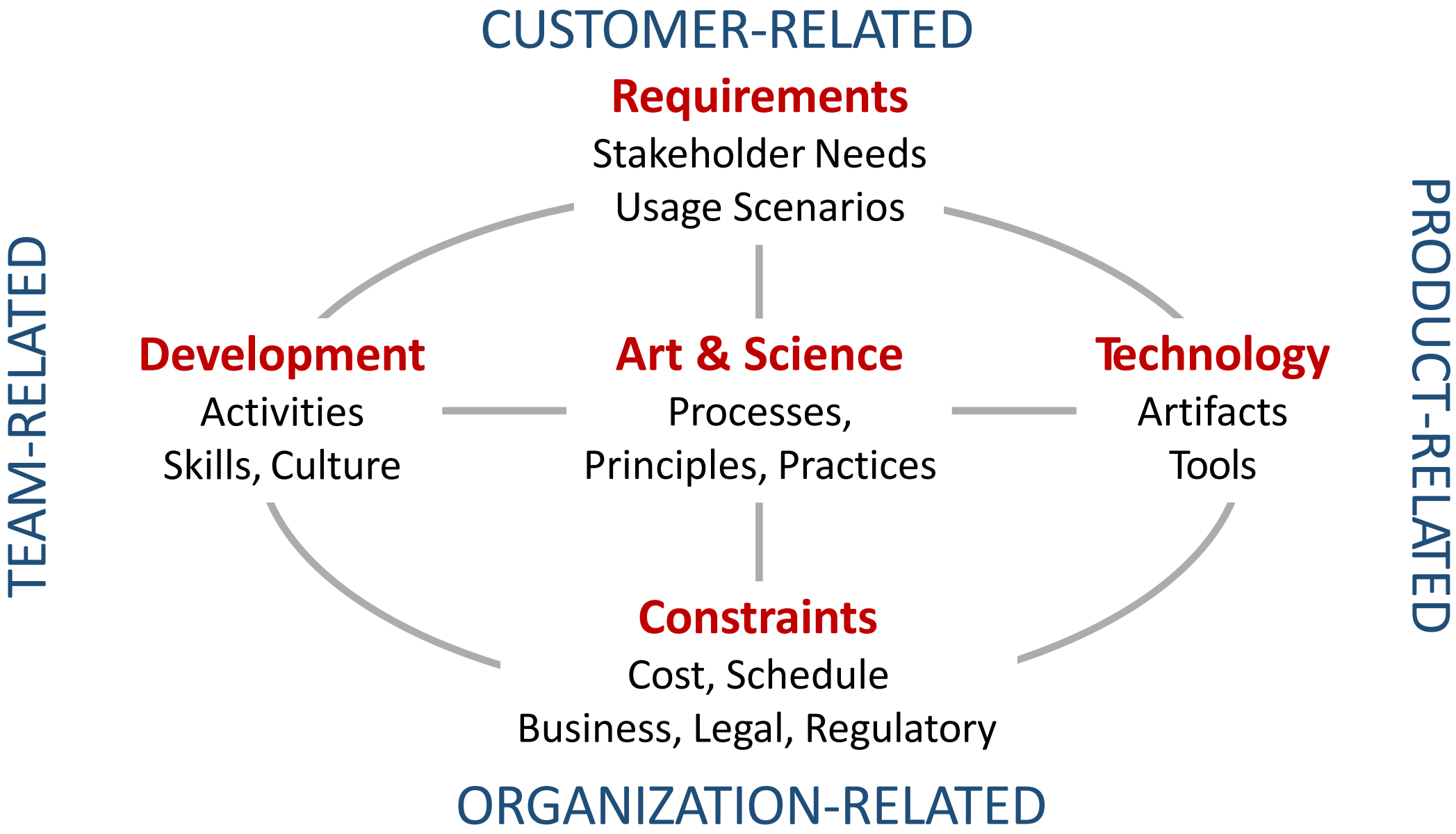
CUSTOMER-RELATED



TEAM-RELATED

PRODUCT-RELATED

ORGANIZATION-RELATED


CUSTOMER-RELATED
Requirements
Stakeholder Needs
Usage Scenarios
TEAM-RELATED
PRODUCT-RELATED
Development
Activities
Skills, Culture
Art & Science
Processes,
Principles, Practices
Technology
Artifacts
Tools
Constraints
Cost, Schedule
Business, Legal, Regulatory
ORGANIZATION-RELATED

CUSTOMER-RELATED

Requirements

Stakeholder Needs

Usage Scenarios

TEAM-RELATED

Development

Activities

Skills, Culture

Art & Science

Processes,

Principles, Practices

Technology

Artifacts

Tools

PRODUCT-RELATED

Constraints

Cost, Schedule

Business, Legal, Regulatory

ORGANIZATION-RELATED

Software Engineering: Alternative Definition

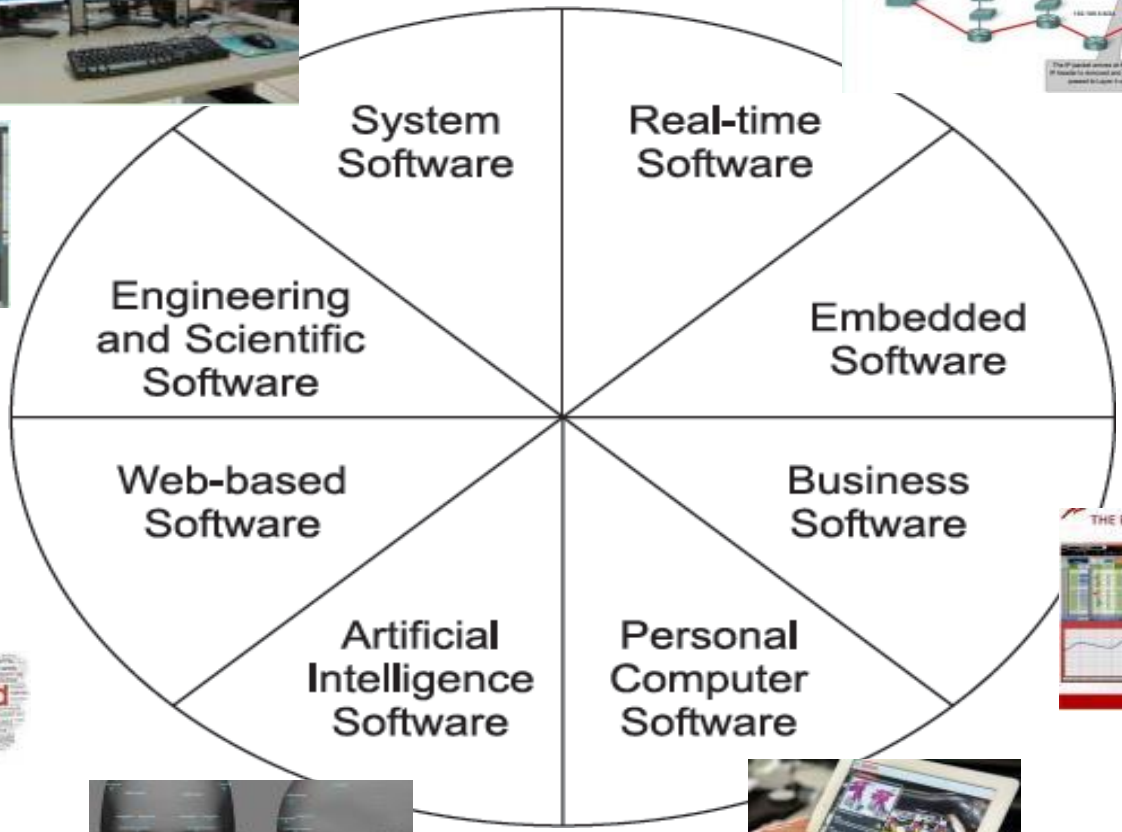
- "Multi-person development of multi-version programs"

Multi Person Coordinate teams Design for modularity	**Multi Person** **Multi Version** X
Single Person	**Multi Version** Develop program families Evolve & maintain releases

Why Software Engineering

- It is necessary to produce a high-quality software to fulfill the below given points.
  - ▶ Consistency
  - ▶ Improved quality
  - ▶ Minimum cost
  - ▶ Within time
  - ▶ Reliability &
  - ▶ Fulfill the need of user

Software Products



Why Software Engineering?

Cause: increase in number, size and application domains of computer programs

Effect: cost, quality and timeliness delivery of system

1. Software products are among **most complex** of man-made systems, and software by its very nature has essential properties (e.g., complexity, invisibility, and changeability) that are not easily addressed [Brooks 95].
2. **Programming techniques** and processes that worked effectively for an individual or a small team to develop modest-sized programs **do not scale-up well** to the development of large, complex systems (i.e., systems with millions of lines of code, requiring years of work, by hundreds of software developers).
3. The **pace of change** in computer and software technology drives the **demand for new and evolved software products**. This situation has created customer expectations and competitive forces that strain our ability to produce quality of software within acceptable development schedules."