Content(2)

- **Object-oriented Software Development Methodology**
  - Outline of Unified Process and Use-case Driven Approach
  - Elevator Control System:
    Problem Description and Use-case Model
  - Elevator Control System:
    Finding of Problem Domain Objects
  - Elevator Control System:
    Sub-System Design and Task Design
  - Elevator Control System:
    Performance Evaluation
- **Product Line Technology**
  - Feature modeling
- **Aspect Oriented Software Design**
- **Contribution of OOT in Software Engineering**
  - History of SE Technologies and Contribution of OOT in SE field

Contribution of OO technologies in Software Engineering

Japan Advanced Institute of Science and Technology
School of Information Science

Koichiro Ochimizu

Software Development is Challenging but Difficult to Achieve!

- Software entities are more complex than most things people build like buildings, automobiles or VLSI.

- *Within only 30 years the amount of software in cars went from 0 to more than 10,000,000 lines of code. More than 2000 individual functions are realized or controlled by software in premium cars, today. 50-70% of the development costs of the software/hardware systems are software costs. (Manfred Broy, "Challenges in Automotive Software Engineering", ICSE2006, pp33-42,2006)*

Why is Software Development so difficult ? (F.Brooks,Jr)

1. Complexity

Computer programs are complex by their nature: a huge amount of part and their relationships.

2. Conformity

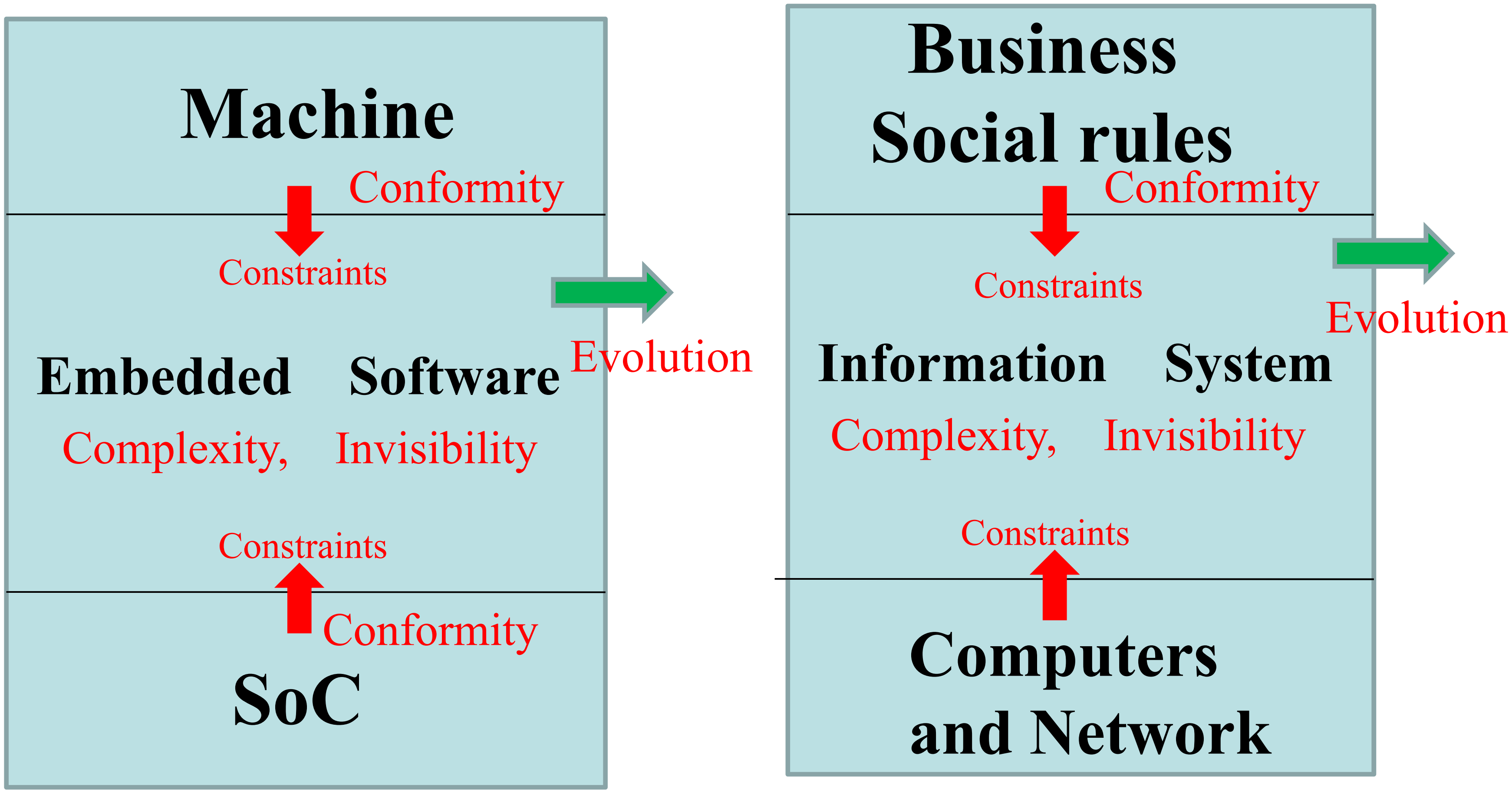
Software can not be created in isolation, but must conform to real-world constraints – pre-existing hardware , third party components, government regulations, legacy data formats, and so on.

3. Changeability

Software is always evolving, as the outer environments of software change.

4. Invisibility

Software doesn't exist in a way that can be represented using geometric models, especially for representing the behavior of software.


Machine
Conformity
Constraints
Embedded Software
Complexity, Invisibility
Constraints
Conformity
SoC
Evolution
Business
Social rules
Conformity
Constraints
Evolution
Information System
Complexity, Invisibility
Constraints
Computers
and Network

Who makes such a complex software?

- Human beings
- A group of human being should collaborate to complete the work within specified time and cost with producing high quality product.
- Difficult to deal with the following problems caused by human beings
  - **instability**
  - **Suddenness**
  - **Uncertainty**

Software Engineering can support their activities

- Software Engineering Technologies
  - Provide us to control the problems specific to software developments
  - Support the team to proceed the work smoothly

Major Topics in Software Engineering

- **Software Process Model** (SPM)
  - SPM provides for the strategy for software development
- **Project Management Technologies** (PM)
  - The application of knowledge, skills, tools and techniques to project activities to meet project requirement. Managing a project includes: identifying requirements; establishing clear and achievable objectives; balancing the competing demands for quality, scope, time and cost (PMBOK).
- **Software Development Methodologies** (SDM)
  - SDM provides for the desirable structure of software and define the procedure how to form them
  - Several examples of structures :easy to verify correctness, easy to encapsulate the change impact, easy to divide the whole work into independent parts, easy to reuse, easy to evolve
- **Languages and Environments**
  - Languages and Environments(Collection of tools) facilitates software engineering activities

Role of Software Process Model(SPM)

- Need to adopt the proper SPM for the project or the organization to integrate individual effort of team members to achieve the goal.
- Because individual member of a project team has different levels of skills
- Sometimes, a project consists of people who belong to different organizations

Is it enough to adopt the proper SPM?

- Can not achieve the high degree of software quality only by adopting the proper software process model.
- A project need to follow some standardized procedure, Software Development Methodology(SDM) , to achieve the high degree of software quality.
- Need a SDM(procedure) based on some SPM(strategy) to achieve the successful software development.

Role of Software Development Methodologies (SDM)

- In the field of SDM study, we have been studying the desirable structure of software and have been defining the procedure how to form them
- Several examples of structures :easy to verify correctness, easy to encapsulate the change impact, easy to divide the whole work into independent parts, easy to reuse, easy to evolve

Is it enough to choose proper SPM and SDM?

- There still remains problems on QCD after adopting the proper SPM(integration of efforts to the goal) and the SDM(standardization of procedure).
- Software development project sometime end up with: cost overruns; schedule delay; poor quality.

Role of Technical Project Management

- The role of PM is:
  - Initiating and planning a project to meet project requirements within limited resources such as human resources , facilities, budget and information
  - to achieve the high quality products on time within budget
  - Monitoring and Controlling the project status, detecting project –specific risks that could not be estimated or predicted at the beginning of the project and being revealed as the project progress

History of SPM, SDM, PM

- Waterfall model (early in the 1970s)
- Development of Programming Methodologies (early in the 1970s)
- Development of Design Methodologies (late in the 1970s)
- Development of Requirement Engineering Technologies (late in the 1970s)
- Beginning of Technical Project Management (late in the 1970s to early in the 1980s)
- Improvement of Waterfall model (V model) (middle to late in the 1980s)
- Iterative Waterfall Model (mini waterfall, spiral) (early in the 1980s)
- Prototyping (early in the 1980s)
- Executable specifications and Formal Methods (middle in the 1980s)
- Process Programming (late in the 1980s)
- SPI (early in the 1990s)
- CASE tools (early in the 1990s)
- Architecture centric Development (middle in the 1990s)
- Object oriented software development technologies (after 1980s)
- Maturity of Software Assessment technologies (late in the 1990s)
- UML (late in the 1990s)
- Iterative Software Process Model(2000s)
- Agile (2000s)
- GORE, IR,COTS (middle of 2000s)

Change of SPM

- Waterfall Model
  - Custom development, Large-scaled software development
- V Model (System Engineering)
  - Outsourcing
- Iteration by Mini Waterfall Model or Spiral
  - Risk Management
- Prototyping
  - User involvement
- Iterative & Incremental SPM
  - Reduction of uncertainty by studying the project specific features

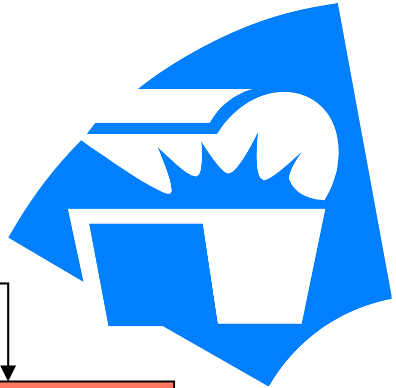
How was Waterfall model constructed?

- Design of phases
  - Starting from “Analysis” and “Coding”
  - Add necessary phases to control a large program development
    - System Requirements, Software Requirements, Program Design, Testing, Operation
  - Add the Preliminary Design Phase to define the constraints
  - Add information about ordering of phases

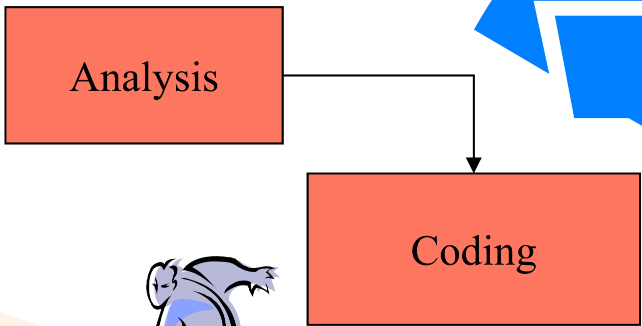
    The design proceeds the change process is scoped down to manageable limits. At any point in the design process after requirements analysis is completed there exists a firm and close-up, moving baseline to which to return in the event of unforeseen design
- Dealing with backtrack problems
  - Implementation described the above item is risky and invites failure. The testing phase which occurs at the end of the development cycle is the first event for which timing, input/output transfer , etc., are experienced. If the wrong phenomena occurs, it may cause backtrack to program design or even to software requirements definition.
  - R.Winston proposed the way how to deal with this problem.

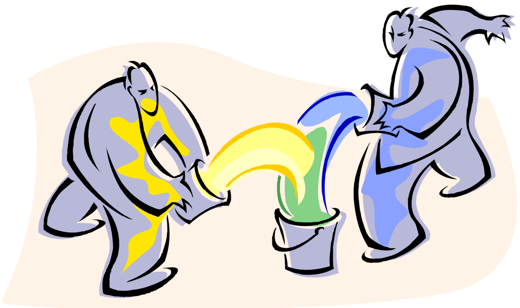
Winston.W. Royce,”Managing the Development of Large Software Systems: Concepts and Techniques”, Proc. of IEEE WESCON, pp.1-9, 1970 (Proc. of 9th ICSE, pp328-338, 1987。

Implementation steps to deliver a small computer program for internal operation



Customers are happy to pay

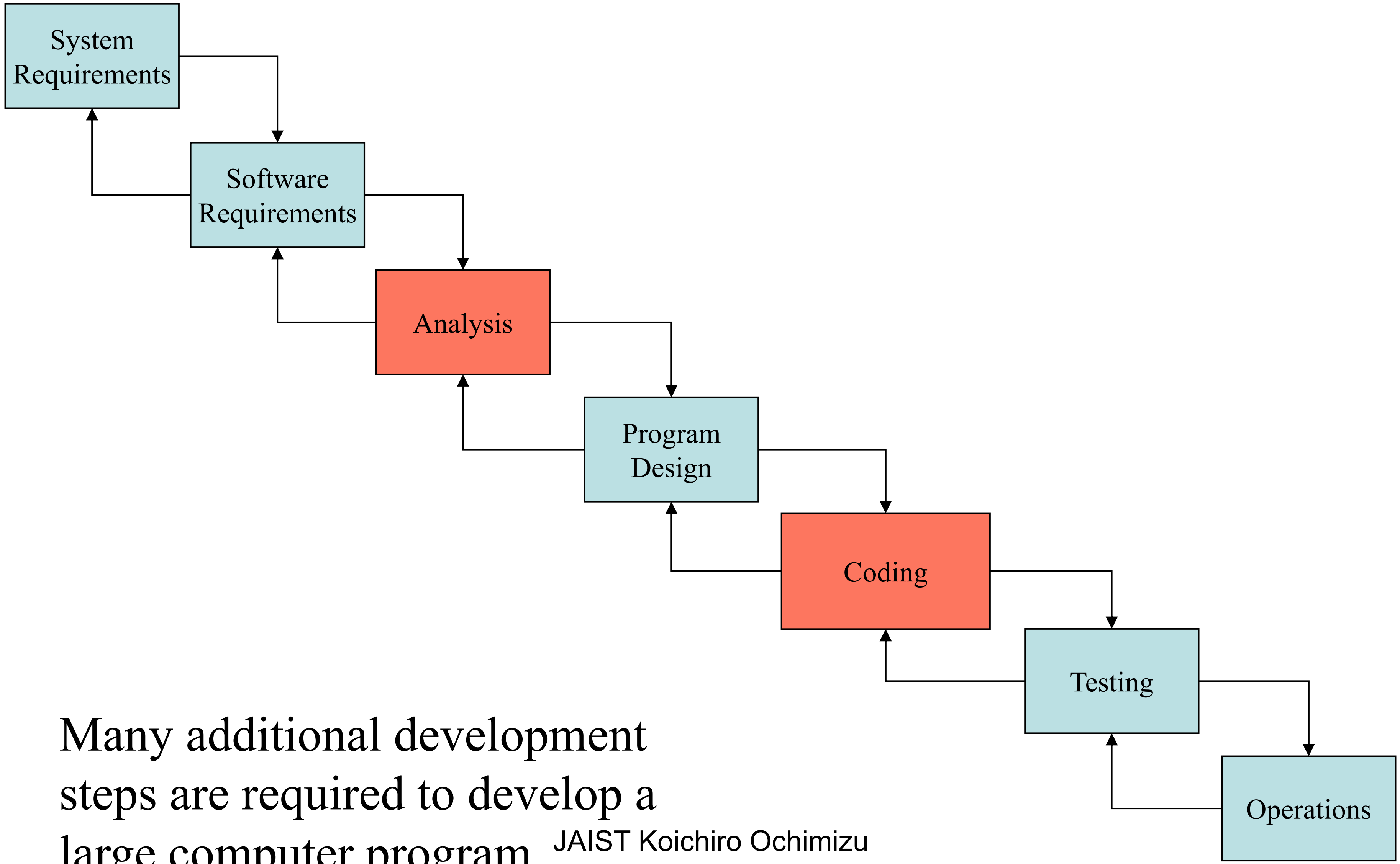




Programmer: both steps involve genuinely creative work

Effective for internal use

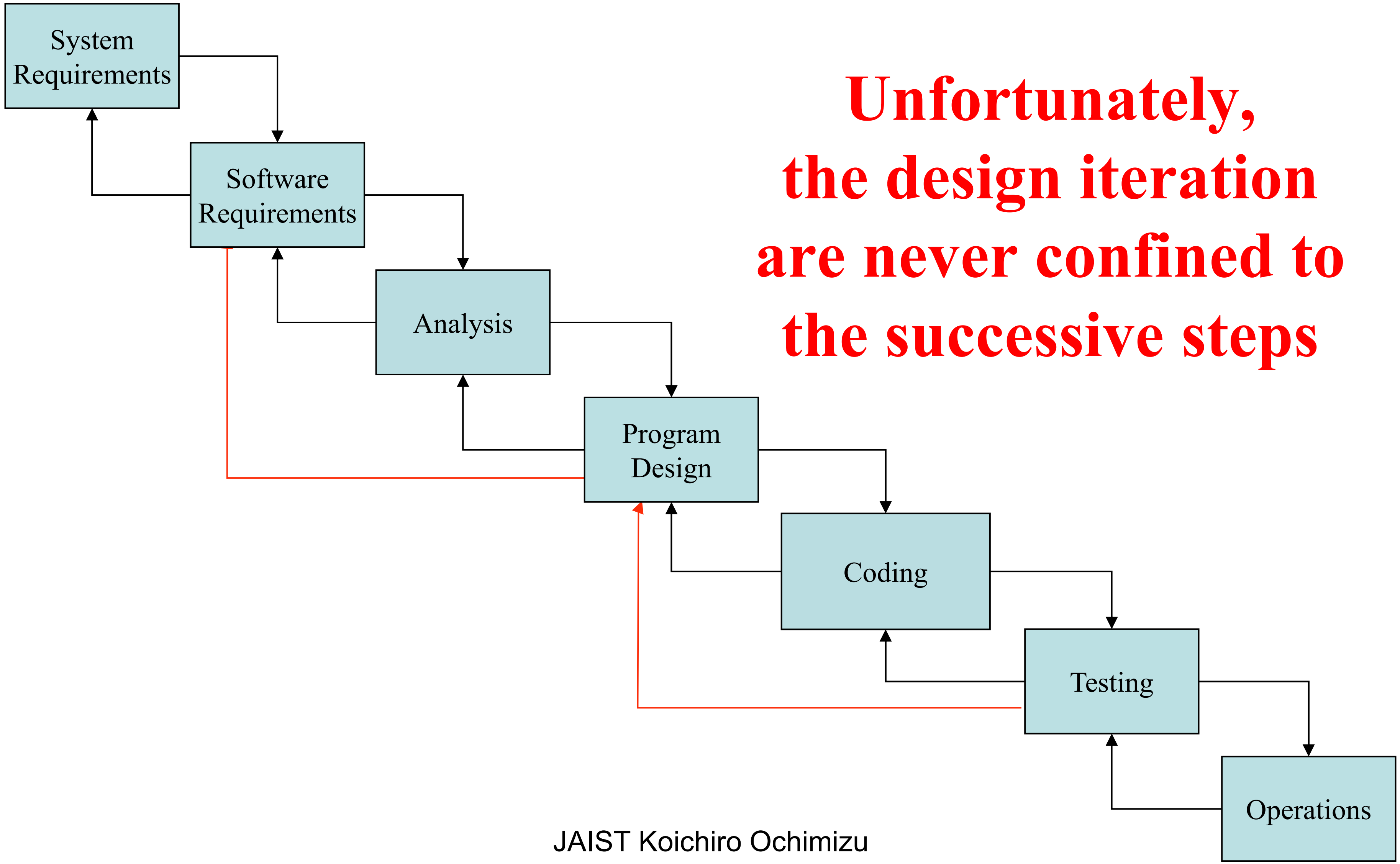
Implementation steps to develop a large computer program for delivery to a customer



Many additional development steps are required to develop a large computer program

JAIST Koichiro Ochimizu

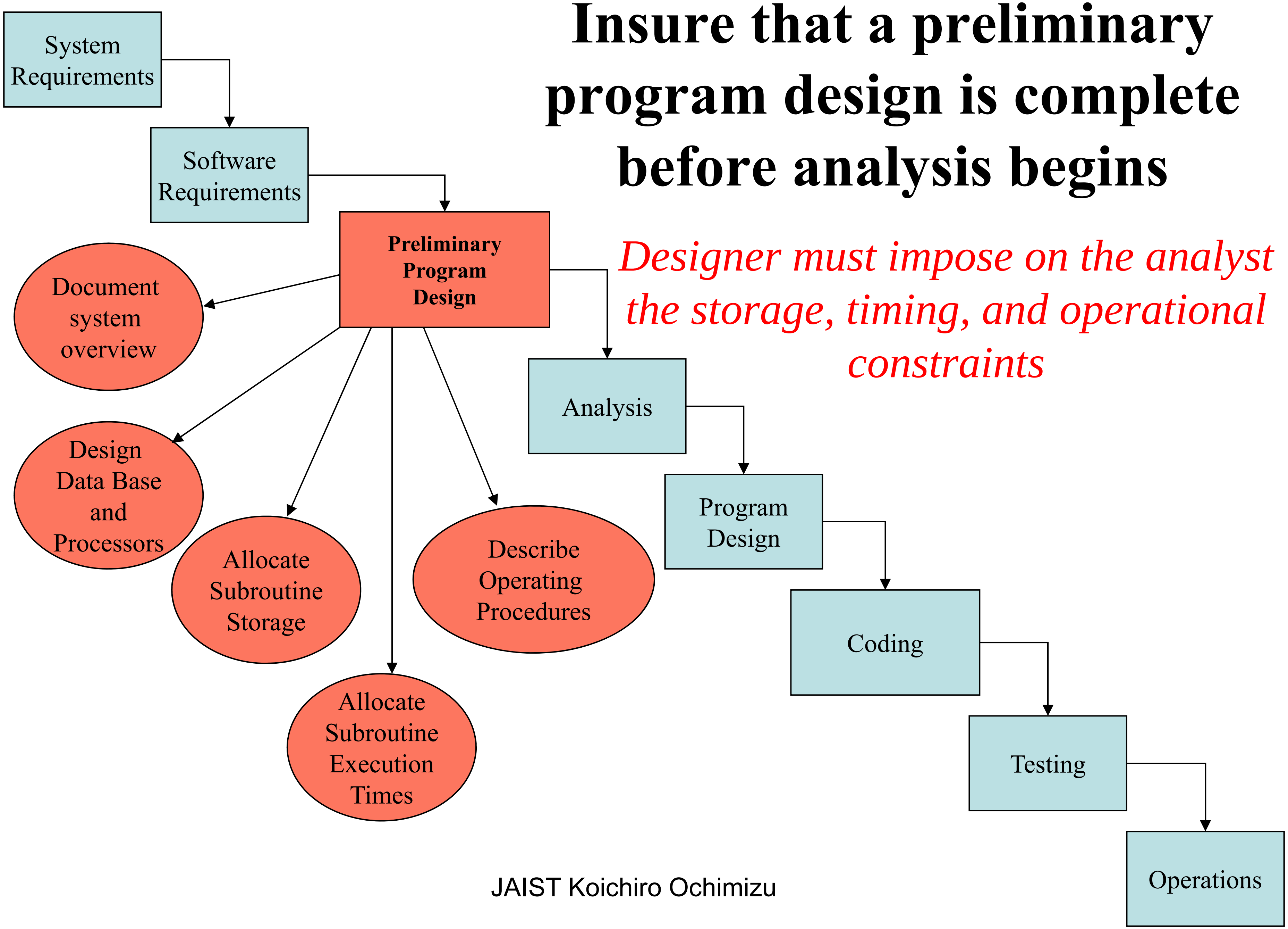
Hopefully, the iterative interaction between the various phases is confined to successive steps



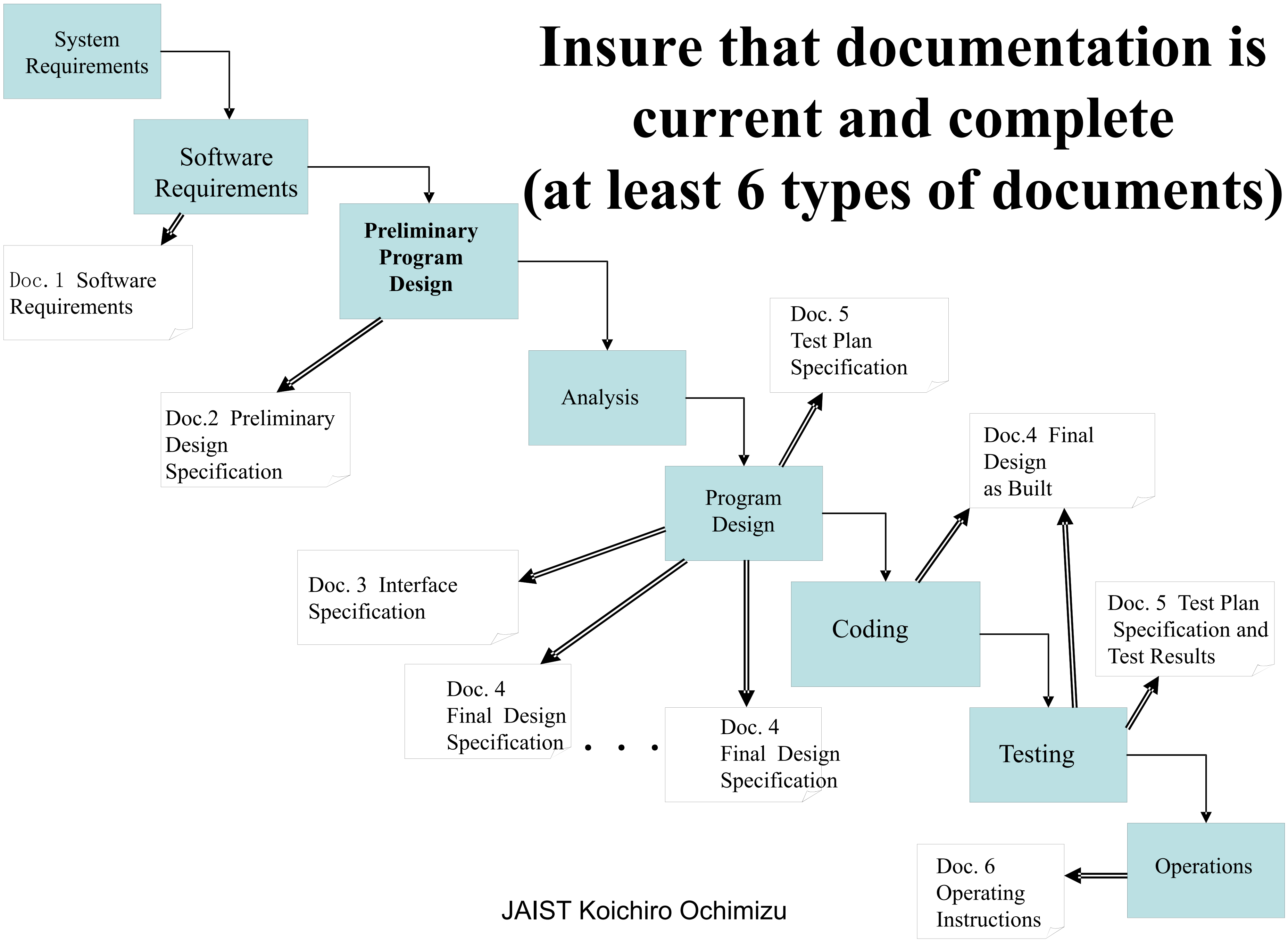
Unfortunately, the design iteration are never confined to the successive steps

Insure that a preliminary program design is complete before analysis begins

Designer must impose on the analyst the storage, timing, and operational constraints

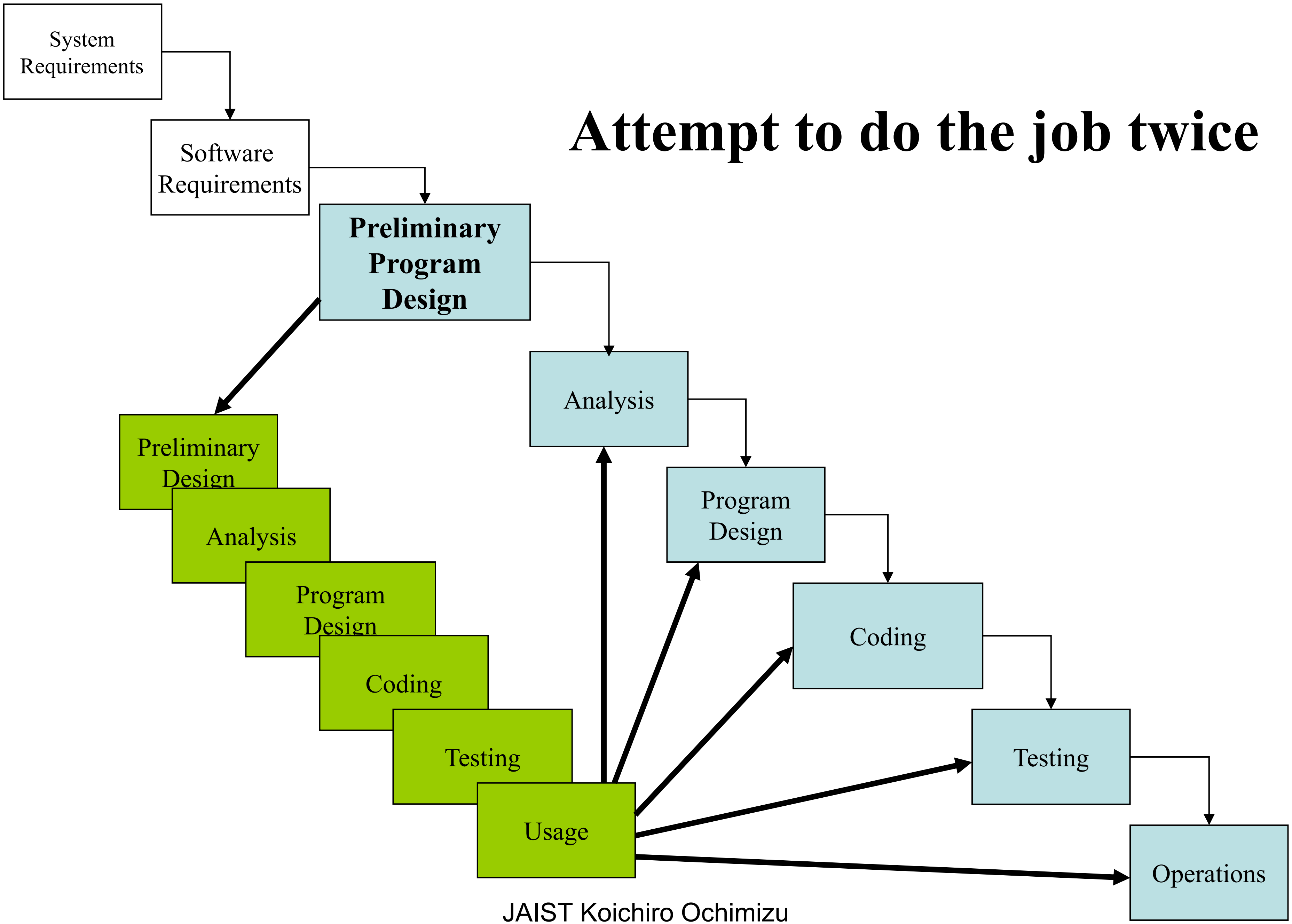


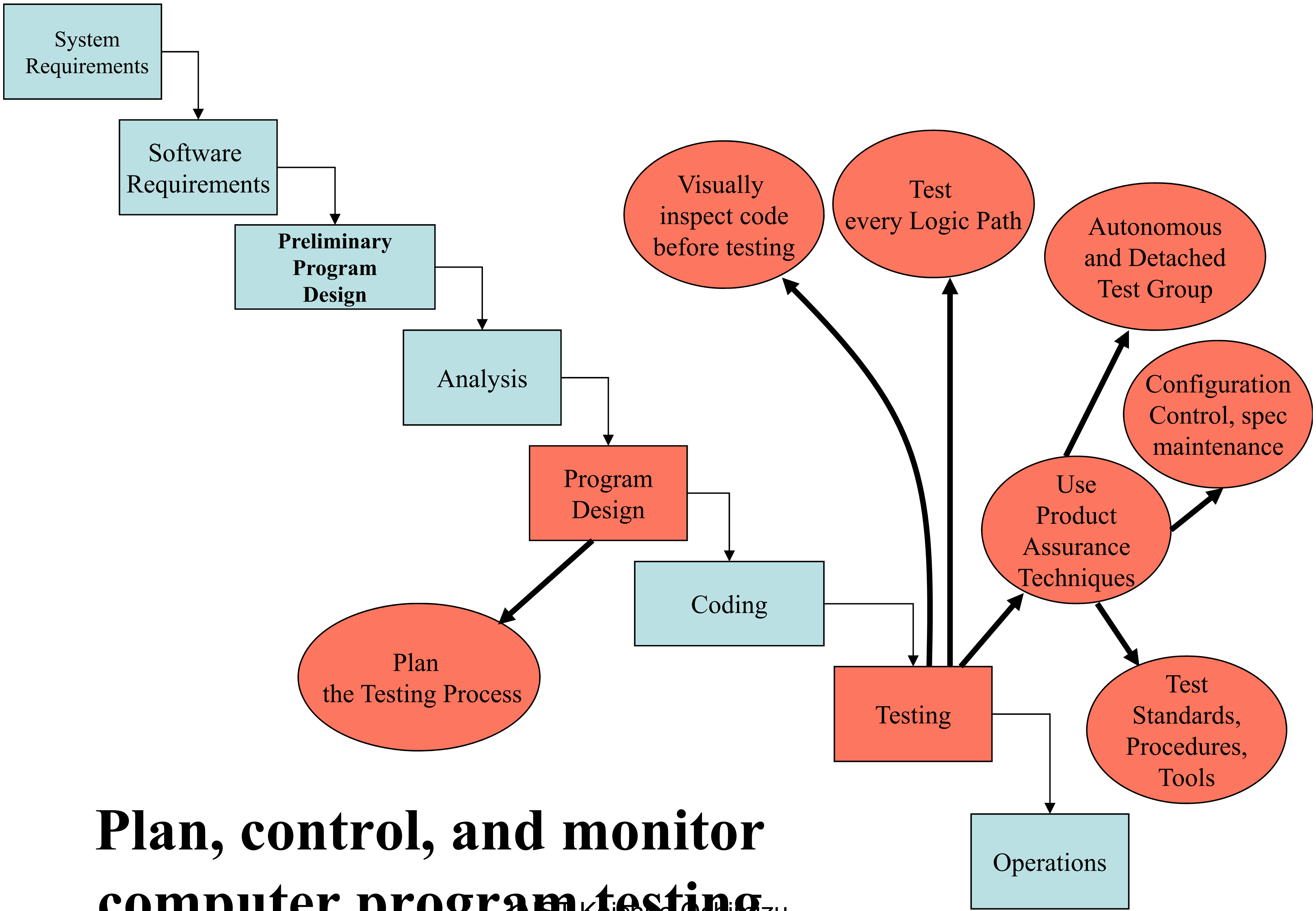
Insure that documentation is current and complete (at least 6 types of documents)



JAIST Koichiro Ochimizu

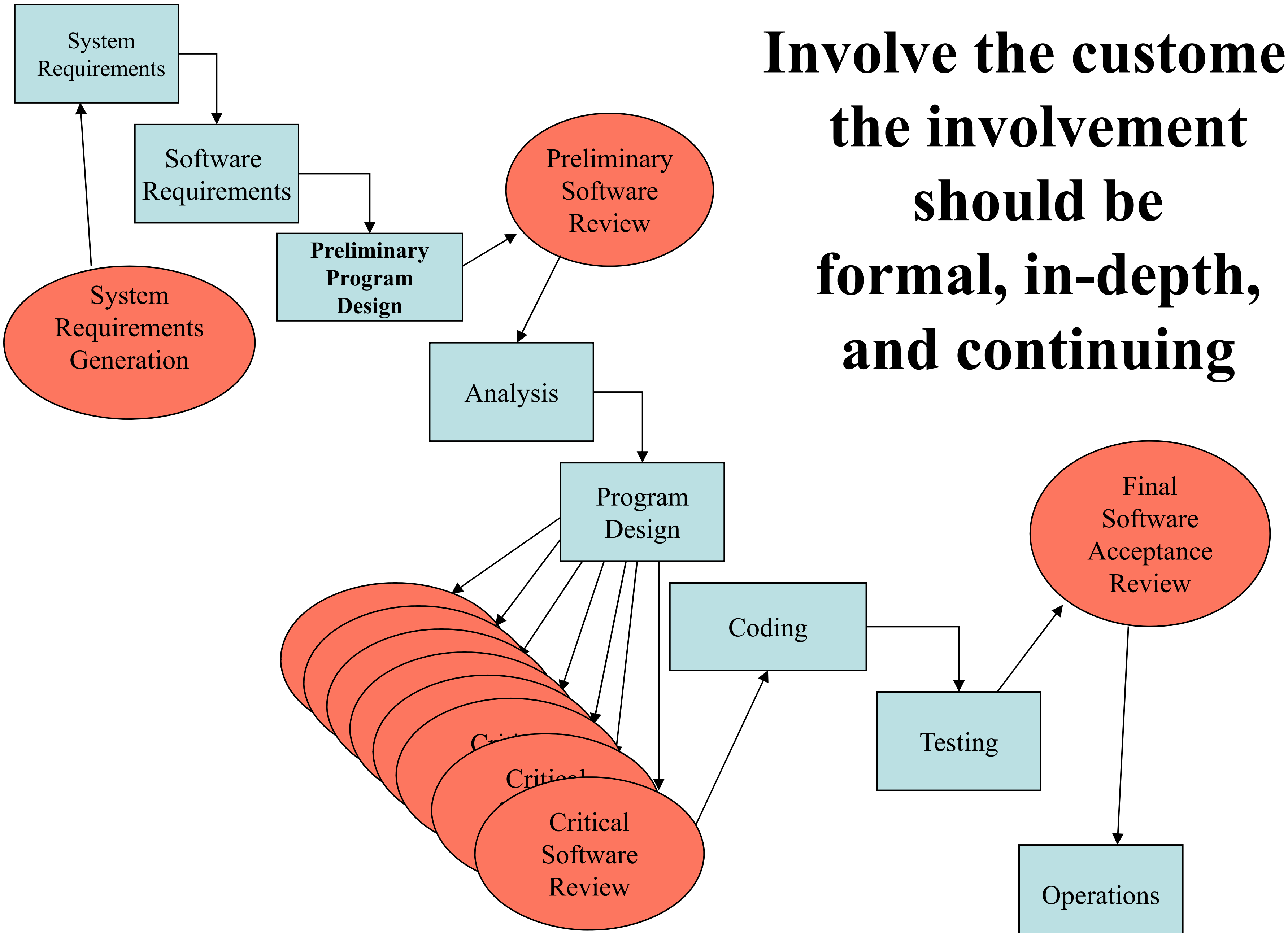
Attempt to do the job twice

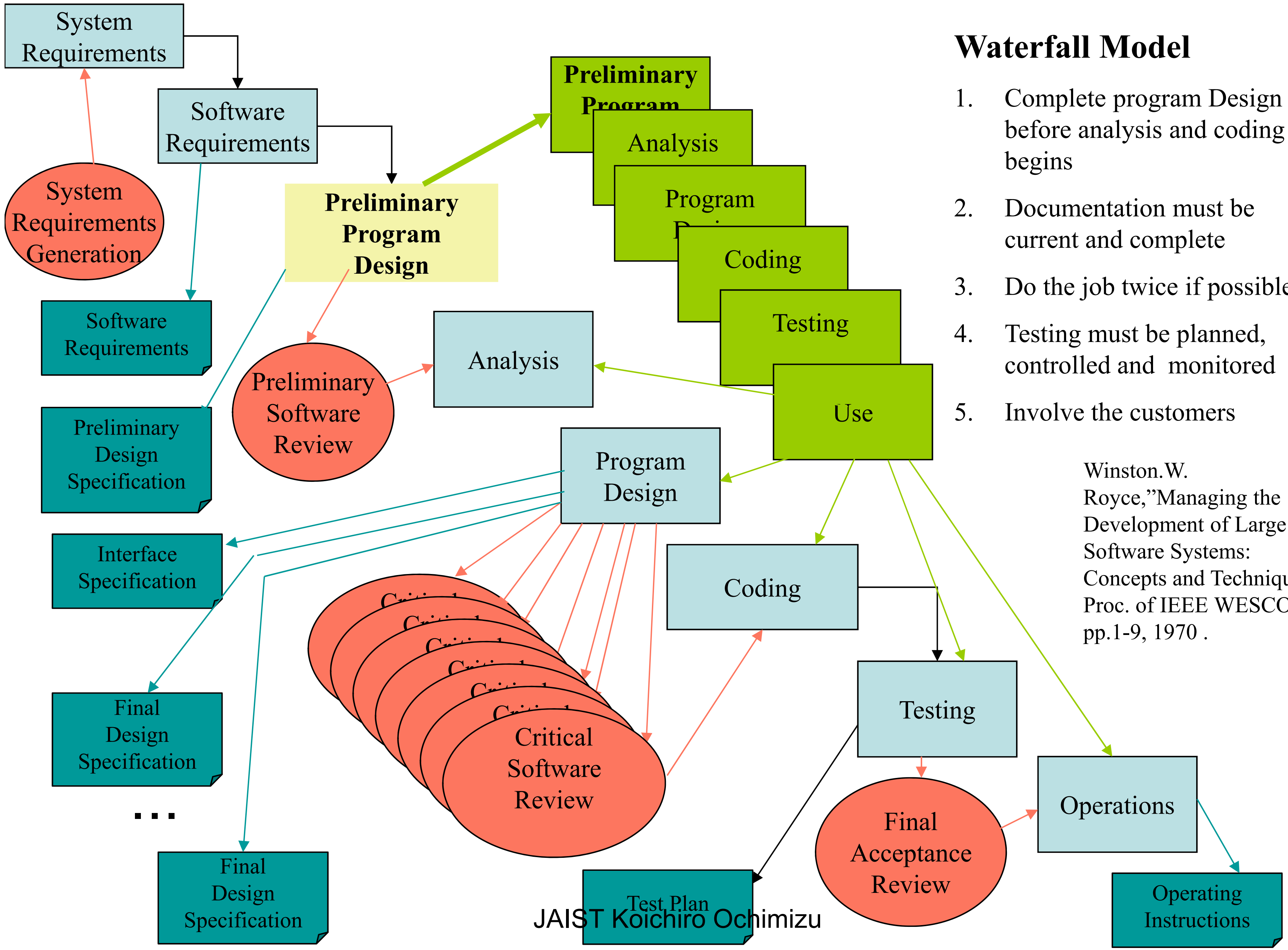




Plan, control, and monitor computer program testing

Involve the customer the involvement should be formal, in-depth, and continuing





Waterfall Model

1. Complete program Design before analysis and coding begins
2. Documentation must be current and complete
3. Do the job twice if possible
4. Testing must be planned, controlled and monitored
5. Involve the customers

Winston.W. Royce,"Managing the Development of Large Software Systems: Concepts and Techniques", Proc. of IEEE WESCON, pp.1-9, 1970 .

Introduction of System Engineering

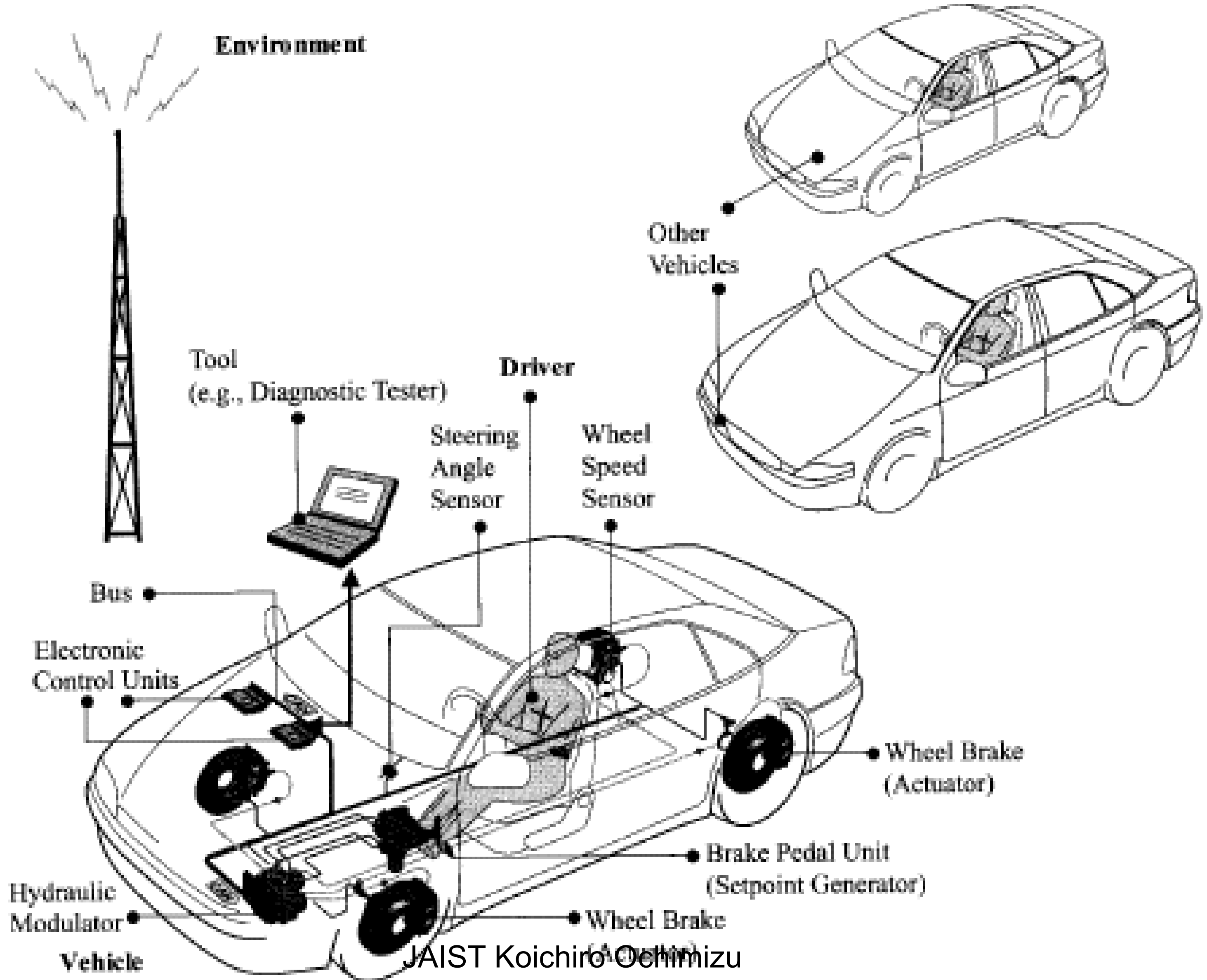
- Systems engineering techniques are used in complex projects: spacecraft design, computer chip design, robotics, software integration, and bridge building. Systems engineering uses a host of tools that include modeling and simulation, requirements analysis and scheduling to manage complexity.
- The **V-model** is a software development process which can be presumed to be the extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

Wikipedia "System engineering" and "V-Model"

V Model

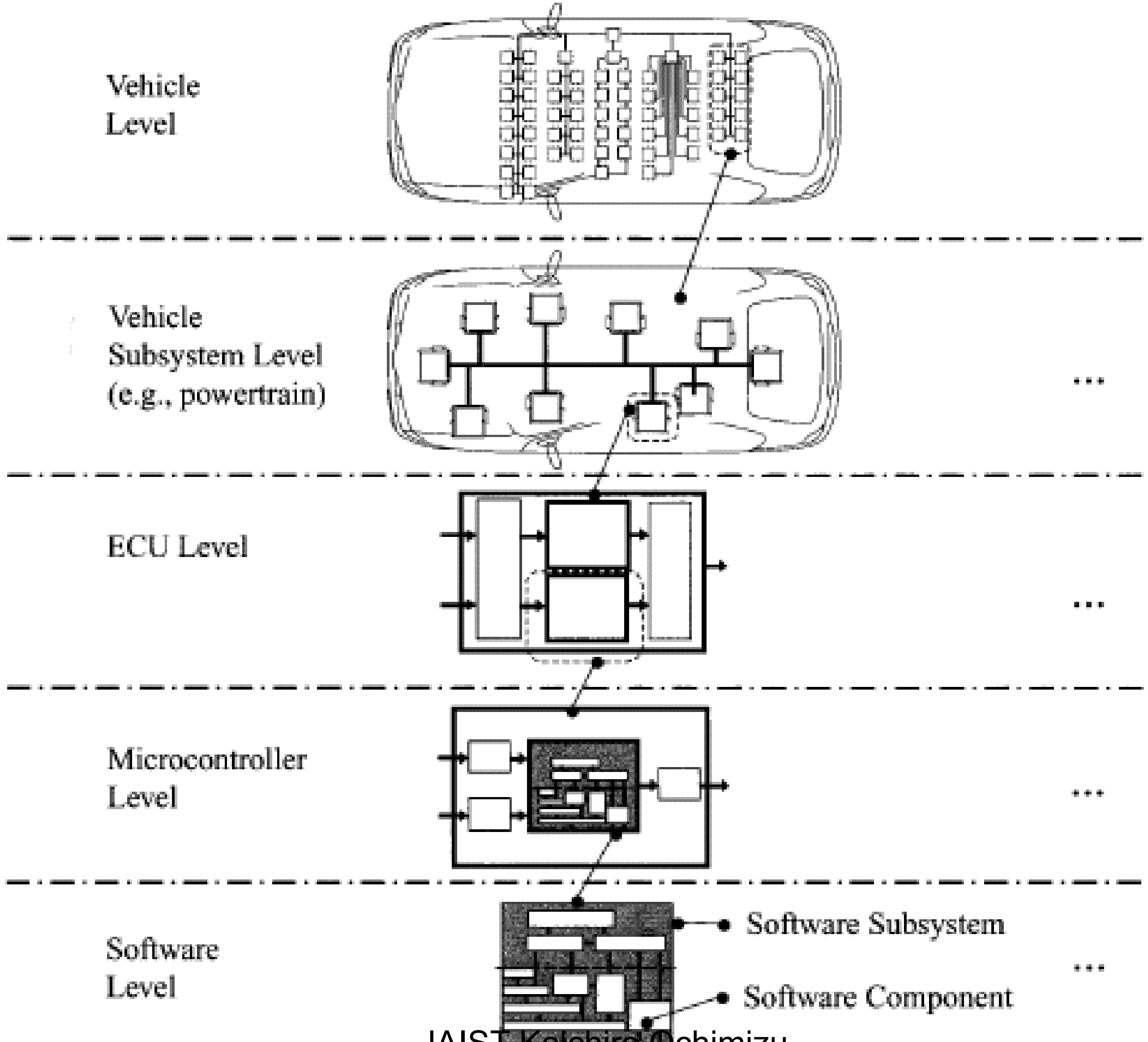
- Introduction of System Engineering Approach
  - Define System Requirements
  - Allocate system requirements to subsystems
  - Define the detailed components
  - Test components, Test subsystems
- Write specification to outsource parts of the whole system with producing the specification of acceptance test in the same level of abstraction.

Inside Structure of a Car



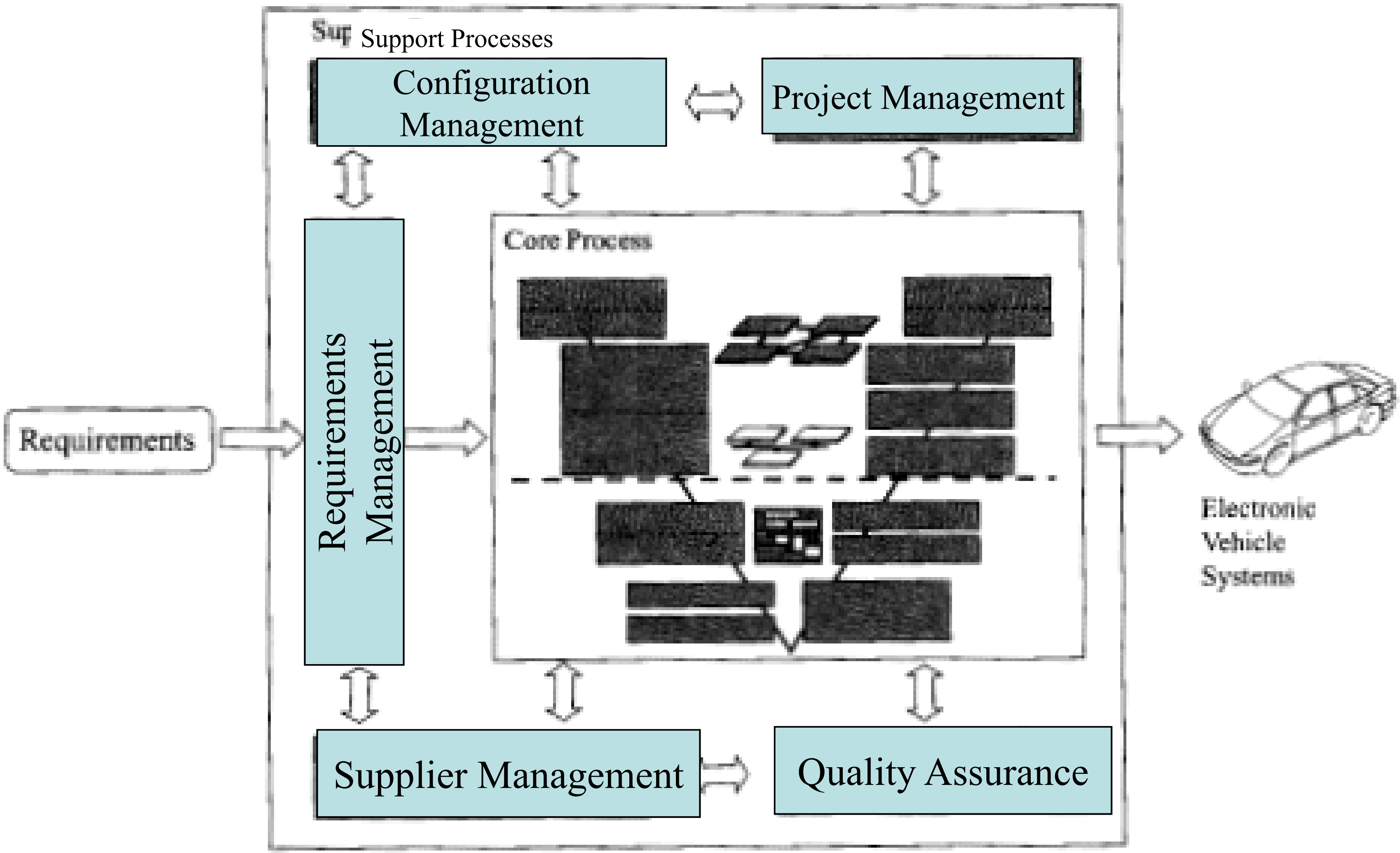
Jorg Schauffele, Thomas Zurawka, “Automotive Software Engineering”, SAE Intnl. 2005.

System levels in automotive electronics



Jorg Schauffele, Thomas Zurawka, “Automotive Software Engineering”, SAE Intnl. 2005.

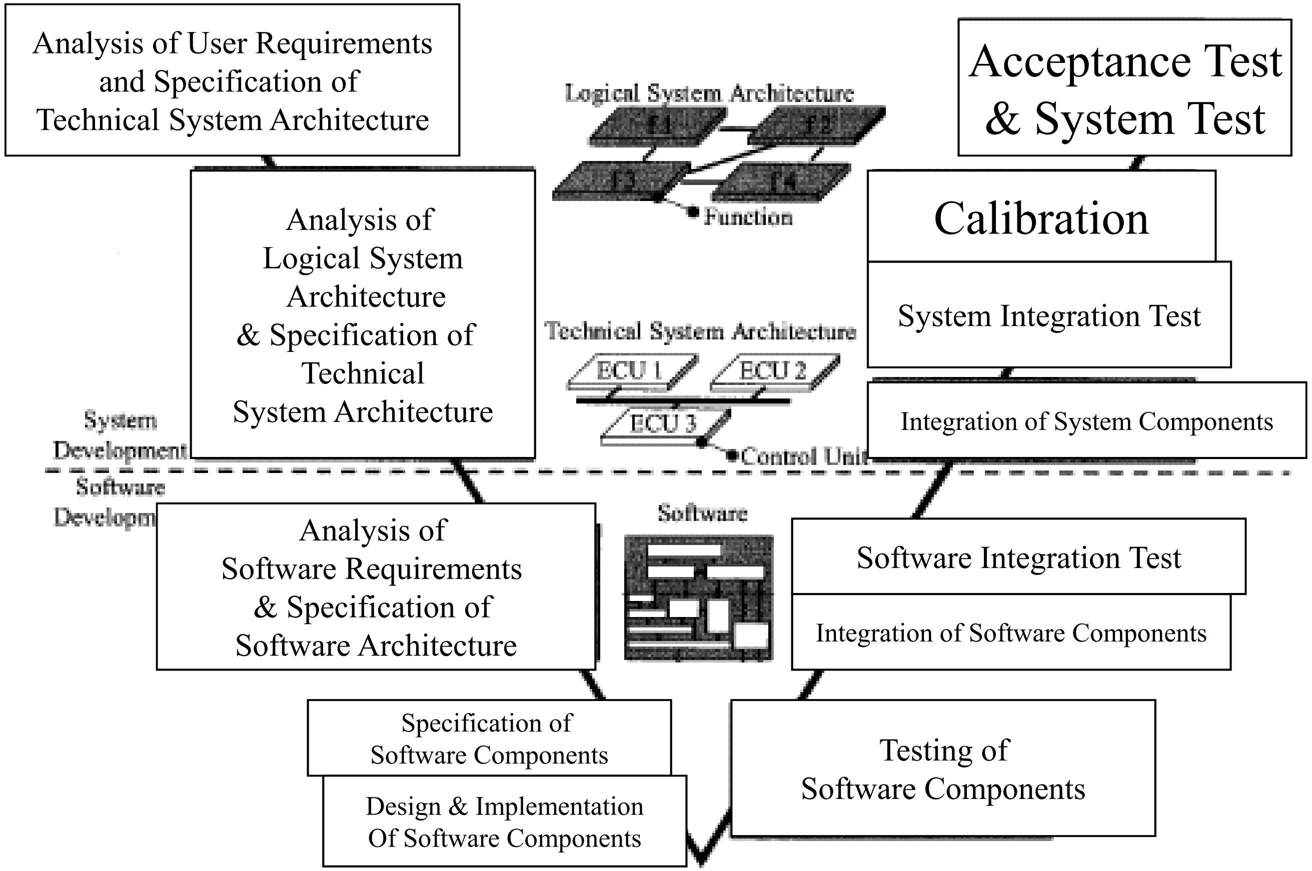
Overview of support processes for the development of electronic systems and software



JAIST Koichiro Ochimizu

Jorg Schauffele, Thomas Zurawka, “Automotive Software Engineering”, SAE Intnl. 2005.

Overview of the core process for the development of electronic systems and software



JAIST Koichiro Ochimizu

Jorg Schauffele, Thomas Zurawka, “Automotive Software Engineering”, SAE Intnl. 2005.

Backtracking to Iteration

- Iteration
  - Mini-Waterfall model, Spiral Model
  - Risk Management
  - Detect and Deal with project-specific risks on QCD early
- Prototyping
  - Involve user into iteration to fix requirements smoothly

Iterative & Incremental Approach

- The basic idea behind iterative enhancement is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added.(wikipedia)
- The earlier we can detect the project-specific problems, the greater the chance to correct them become
- At the beginning of the project, we can not understand the project goal clearly. After getting the development experience once, we can use the knowledge got from the first increment and can have a chance to change the process

What do Software Engineering Projects consider important? by Pete McBreen

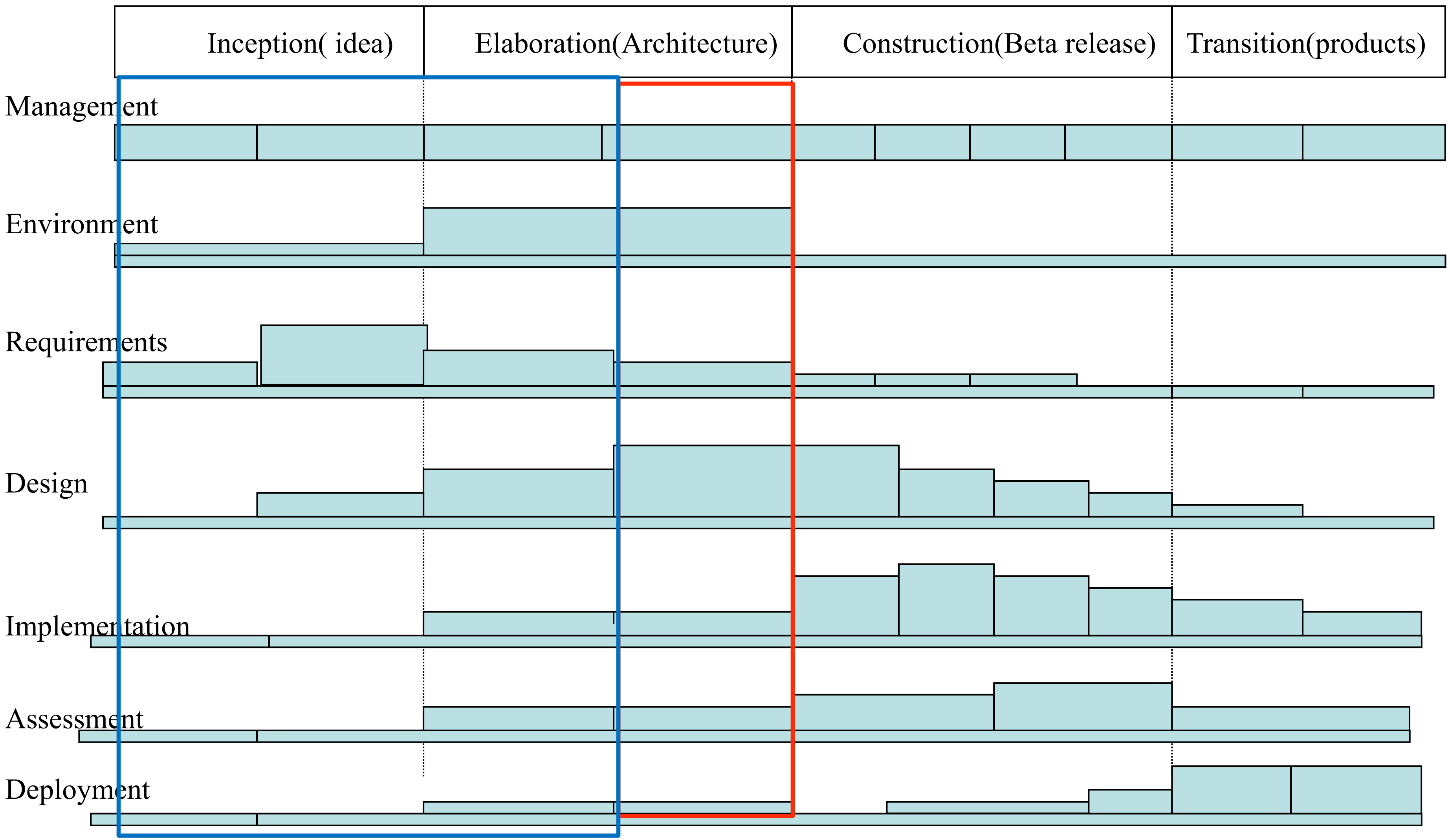
- **Traditional Waterfall Projects**
  - Specialization of staff into different roles to support the different phases is claimed to promote efficiency by reducing the number of skills a person needs.
  - With clear milestones between phases and known dependencies between deliverables, it is easy to display a waterfall project on a PERT chart.
  - Comprehensive documentation is important, so that at the end of the project it is possible to justify the overall costs. This supports the tracking of the project because it makes everything available for external review. A side benefit of all of this documentation is traceability.
- **Unified Process (supports Incremental development in the context of a phased approach)**
  - Inception(evaluating the economic feasibility of the project, forcing the team to define the overall project scope, plan the remaining phases, and produce estimates)
  - Elaboration (evaluating the technical feasibility of the project by creating and validating the overall software architecture)
  - Construction (at the end of each increment, new and changed requirements can be incorporated into the plans, and the estimates can be refined based on experiences in the previous increments)

Pete McBreen, “Questining eXtreme Programming”, Addison-Wesley, 2003.

Features of Iterative model

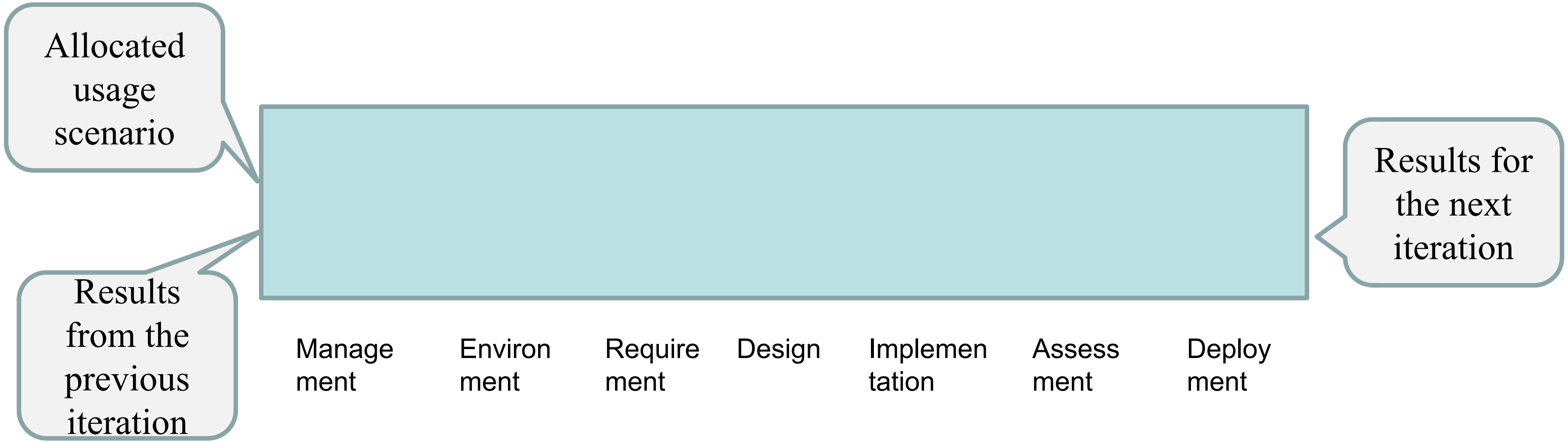
- Rather than being built sequentially, the artifacts are evolved together, and the constraints , the different levels of abstractions, and the degree of freedom are balanced among competing alternatives.
- The primary difference from the conventional approach is that within each life-cycle phase, the workflow activities do not progress in a simple linear way, nor does artifact building proceed monotonically from one artifact to another
- Instead, the focus of activities sweeps across artifacts repeatedly, incrementally enriching the entire system description and the process with the lessons learned in preserving balance across the breadth and depth of information
- An iteration represents the state of the overall architecture and the complete deliverable system. An increment represents the current work in progress that will be combined with the preceding iteration to form the next iteration

Relative levels of effort expected across the phases



Walker Royce," Software Project Management: A Unified Framework", ADDISON-WESLEY,

Unit of Iteration



Activities

- Management: iteration planning to determine the content of the release and develop the detailed plan for the iteration; assignment of work package, or tasks, to the development team
- Environment: evolving the software change order database to reflect all new baselines and changes to existing baselines for all product, test, and environment components
- Requirements: analyzing the baseline plan, the baseline architecture, and the baseline requirements set artifacts to fully elaborate the use cases to be demonstrated at the end of this iteration and their evaluation criteria; updating any requirements set artifacts to reflect changes necessitated by results of this iteration's engineering activities

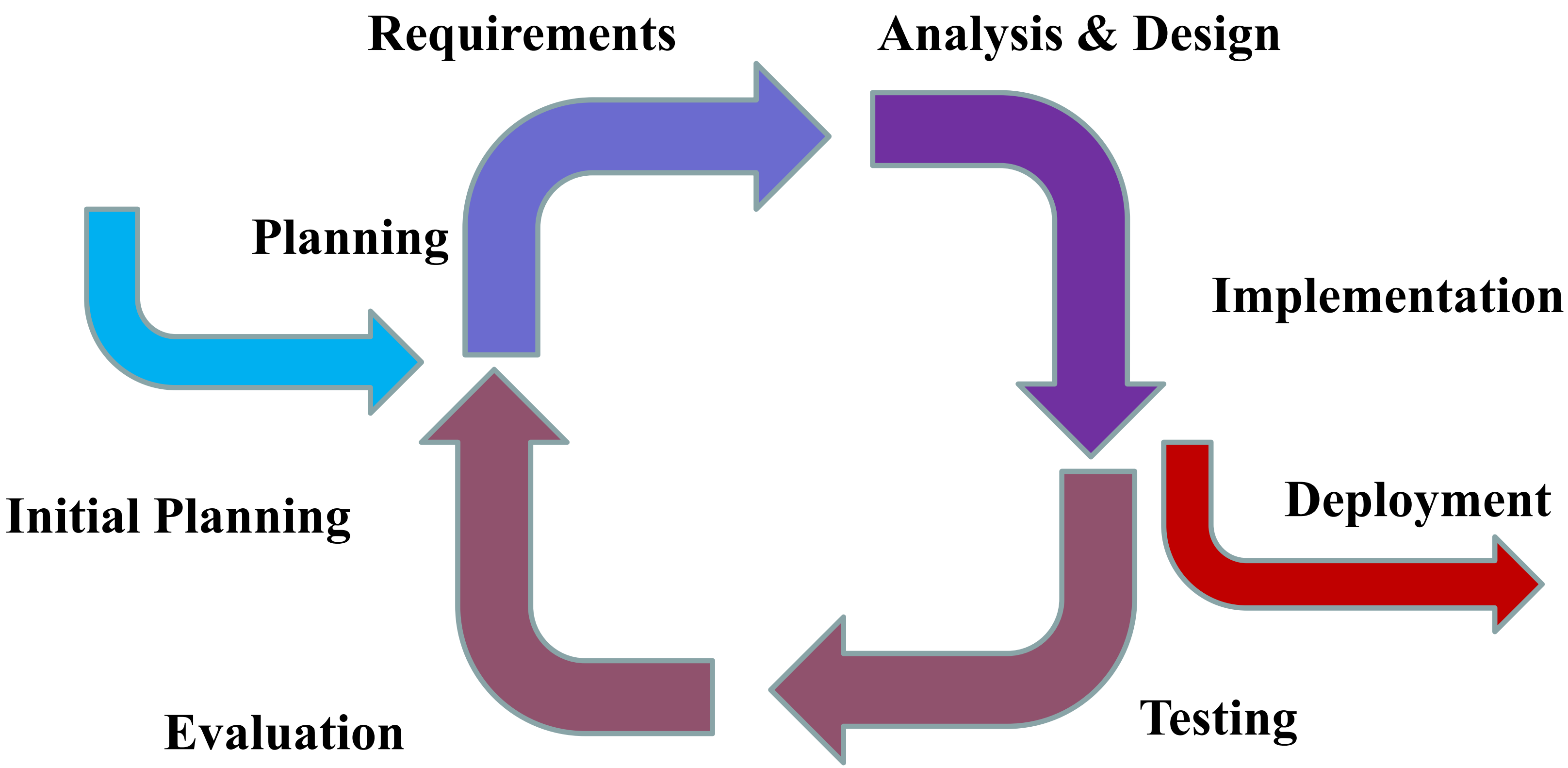
Activities

- Design: evolving the baseline architecture and the baseline design set artifacts to elaborate fully the design model and test model components necessary to demonstrate against the evaluation criteria allocated to this iteration; updating design set artifacts to reflect changes necessitated by the results of this iteration's engineering activities
- Implementation: developing or acquiring any new components, and enhancing or modifying any existing components, to demonstrate the evaluation criteria allocated to this iteration; integrating and testing all new and modified components with existing baselines(previous versions)

Activities

- Assessment: evaluating the results of the iteration, including compliance with the allocated evaluation criteria and the quality of the current baselines; identifying any rework required and determining whether it should be performed before deployment of this release or allocated to the next release; assessing results to improve the basis of the subsequent iteration's plan. Testing is only one aspect of the assessment workflow
- Deployment: transitioning the release either to an external organization(such as a user, independent verification and validation contractor, or regulatory agency) or to internal closure by conducting a post-modern so that lessons learned can be captured and reflected in the next iteration

An iterative development model



Iterations

Inception

management	Environment	Requirement	Design	Implementation	Assessment	Deployment
Prepare business case and vision	Define development environment and change management infrastructure	Define operational concept	Formulate architecture concept	Support architecture prototypes	Assess plans, visions, prototypes	Analyze user community

Elaboration

management	Environment	Requirement	Design	Implementation	Assessment	Deployment
Plan development	Install development environment and establish change management database	Define architecture objectives	Achieve architecture baseline	Produce architecture baseline	Assess architecture	Define user manual

Iterations

Construction

management	Environment	Requirement	Design	Implementation	Assessment	Deployment
Monitor and control development	Maintain development environment and software change order database	Define iteration objectives	Design components	Produce complete componentary	Assess interim release	Prepare transition materials

Transition

management	Environment	Requirement	Design	Implementation	Assessment	Deployment
Monitor and control deployment	Transition maintenance environment and software change order database	Refine release objectives	Refine architecture and components	Maintain components	Assess product release	Transition product To user

The Artifact Sets

Requirements set	Design set	Implementation set	Deployment set
1. Vision Document Contract Provided Services Constraints 2. Requirement Model Use case model Domain model	1. Design models components of the solution space 2. Test model 3. Software Architecture description	1. Source code baselines 2. Associated compile-time files 3. Component executables	1. Integrated product executable baselines 2. Associated run-time files 3. User manual

Management Set

Planning Artifacts

1. Work breakdown structure
   (activity breakdown and financial tracking mechanism)
2. Business case
   (cost, schedule, profit expectation)
3. Release specification
   (scope, plan, objectives for release baselines)
4. Software development plan
   (project process instance)

Operational Artifacts

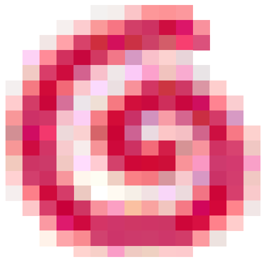
5. Release descriptions
   (results of release baseline)
6. Status assessments
   (periodic snapshots of project progress)
7. Software change order database
   (descriptions of discrete baseline changes)
8. Deployment documents
   (cutover plan, training course sales rollout kit)
9. Environment (hardware and software tools, process automation, document, additional training)

Software Process Workflow

- **Management workflow:** controlling the process and ensuring win conditions for all stakeholders
- **Environment workflow:** automating the process and evolving the maintenance environment
- **Requirements workflow:** analyzing the problem space and evolving the requirements artifacts
- **Design workflow:** modeling the solution and evolving the architecture and design artifacts
- **Implementation workflow:** programming the components and evolving the implementation and deployment artifacts
- **Assessment workflow:** assessing the trends in process and product quality
- **Deployment workflow:** transitioning the end products to the user

Summary on SPM

- From “Controlling the Scale”
- To “Controlling Risks
  caused by Instability, Suddenness, Uncertainty”



Risk management

Outsourcing and Off-shore Development

History of SDM
What structures and How

- Structured Programming
  - easy to verify correctness a program, easy to divide the whole work into independent parts
- Information Hiding Module
  - Encapsulation of change impact
- Structured Analysis and Design
  - Encapsulation of change impact
- Requirement Engineering
  - Requirements definition
- Executable Specifications and Formal Methods
  - Verifying and proving some properties of a program, Generation of a program,
- Object-Orientation
  - easy to encapsulate the change impact, easy to reuse and easy to evolve a program
- Goal Oriented Requirement Engineering, Integrated Requirement Engineering, COTS
  - Shortening the development time

Principles on Software Engineering

1. **Rigor and Formality**

   Rigorous approach enables us to produce more reliable products, control their cost, and increase our confidence in their reliability. Formality is a stronger requirement than rigor; it requires the software process to be driven and evaluated by mathematical laws.

2. **Separation of Concerns**

   To deal with different individual aspects of a problem and we can concentrate on each separately.

3. **Modularity**

   Kind of Separation of Concerns. A complex system may be divided into simpler pieces called modules, allowing details of each module being handled in isolation.

4. **Abstraction**

   Kind of Separation of Concerns; Separation of what from how. The we can identify the important aspects of a phenomenon and ignore its details.

5. **Anticipation of Change**

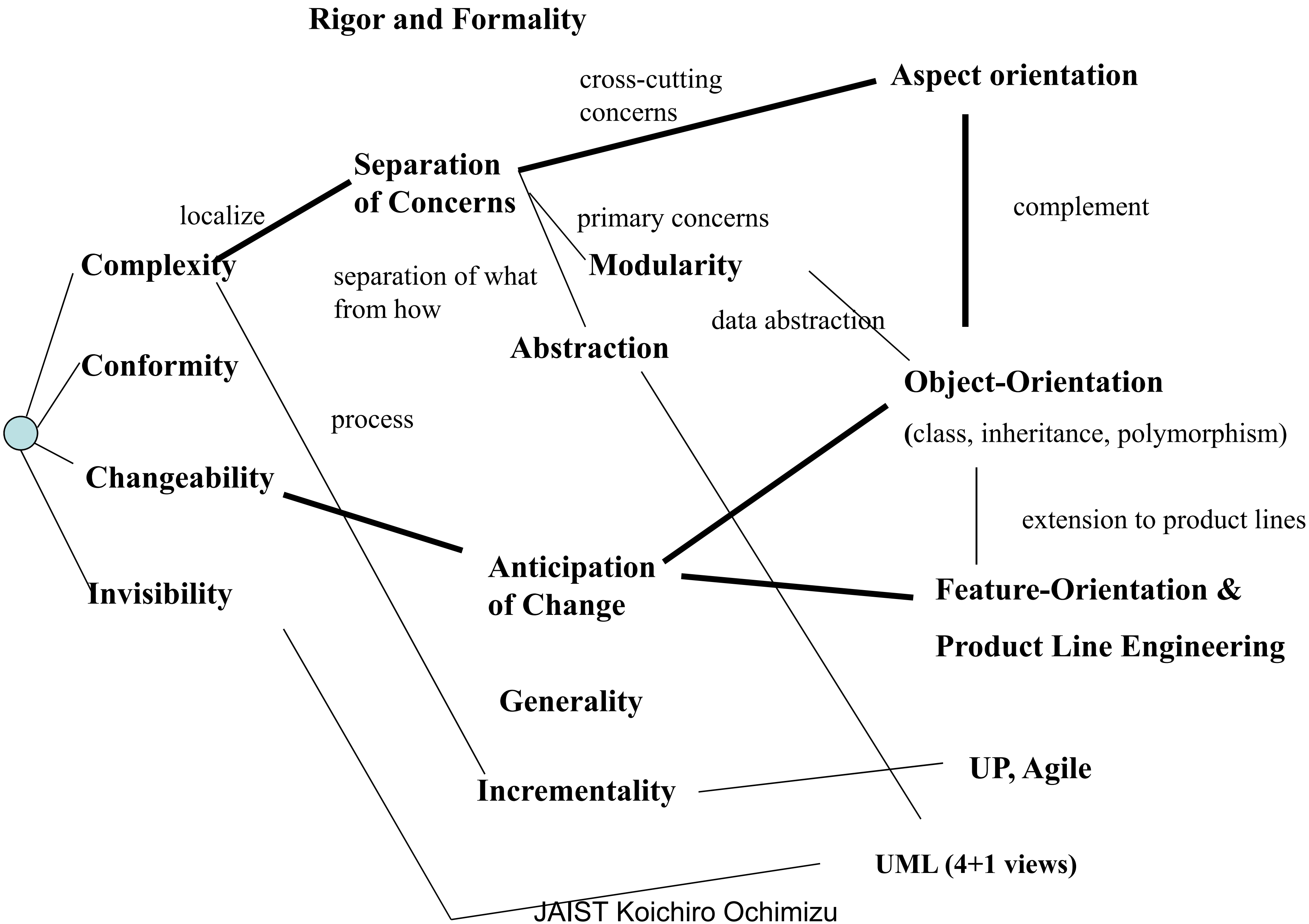
   When likely changes are identified, special care must be taken to proceed in a way that will make future changes easy to apply.

6. **Generality**

   Generalizing the problem to make the solution more potential one for being reused.

7. **Incrementality**

   A process that proceeds in stepwise fashion, in increments, for risk reduction.



JAIST Koichiro Ochimizu

SawSanda Aye and K. Ochimizu," Defining Ontology for Complexity Issues in Software Engineering", Natnl Conf. of JSSST, 2004.

Summary on SDM

- Principles pursued
  - Objects to be made, verified and modified should appear in the same part of source code
  - Reduce the volume of codes to be written

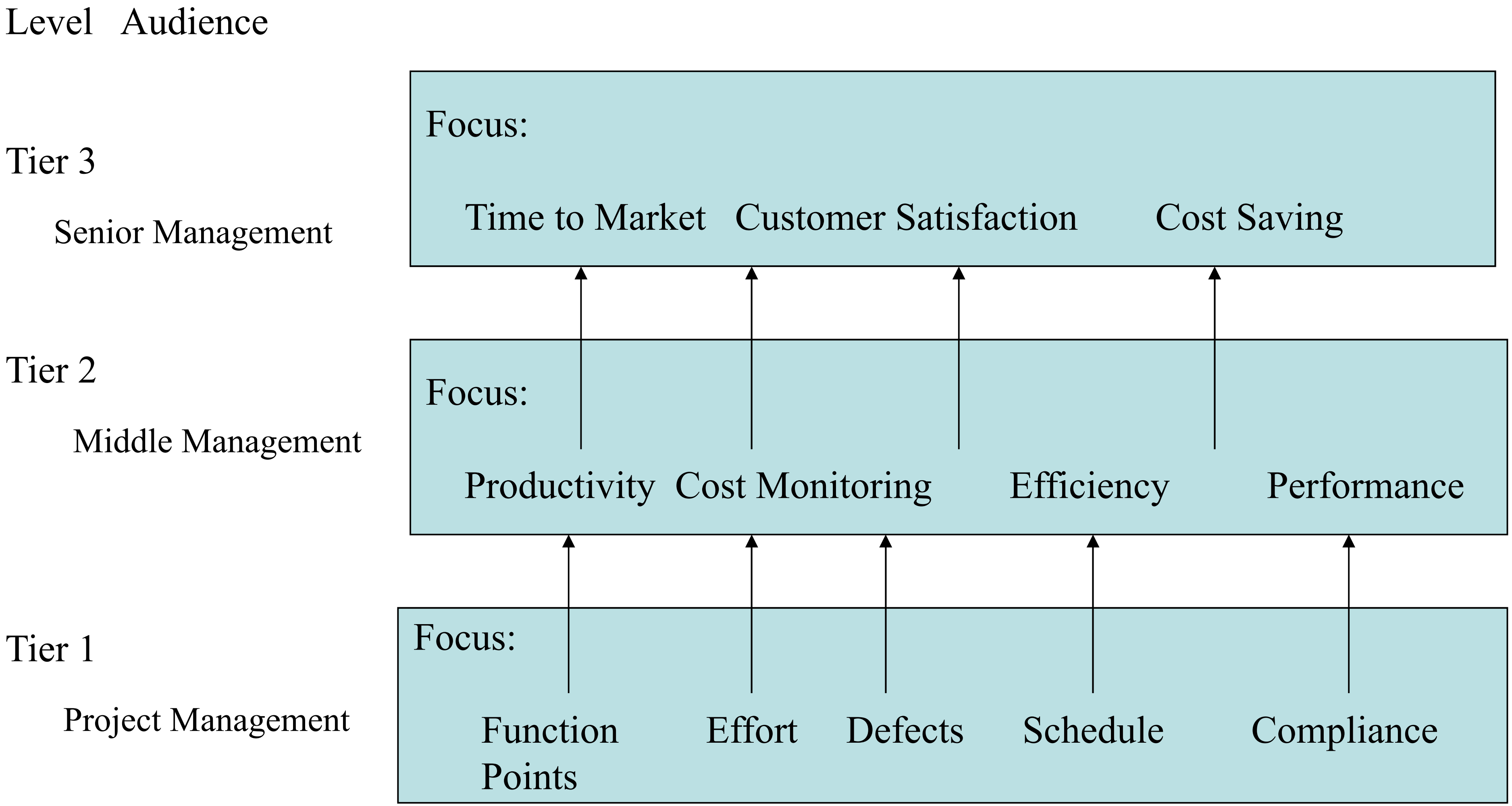
Development of SPM

- Various Measures
  - Cost-estimation
  - Detection of risky factors (Software complexity measures, V measure, E measure)
  - Decision support for terminating test activities (software reliability growth model)
- Measurement
  - Function Points
- CMM
  - Maturity Levels and Best Practices
- Software Assessment
  - Benchmark and Baseline
- PMBOK
  - Knowledge

History of Project Management

- 1910s: the Gantt chart by Henry Gantt
- prior to the 1950s, projects were managed on an *ad hoc* basis using mostly Gantt Charts, and informal techniques and tools. At that time, two mathematical project scheduling models were developed. The "Critical Path Method" (CPM) and the "Program Evaluation and Review Technique" or PERT for Polaris missile submarine program; These mathematical techniques quickly spread into many private enterprises.
- In 1969, the Project Management Institute (PMI) was formed to serve the interests of the project management industry.
- 1970s: Theory of Constraint (TOC): Drum Buffer, Rope by E.M. Goldratt

The Need for Software Measurement



David Garmus, David Herron, “Function Point Analysis” ADDISON-WESLEY, 2001.

Software Assessment

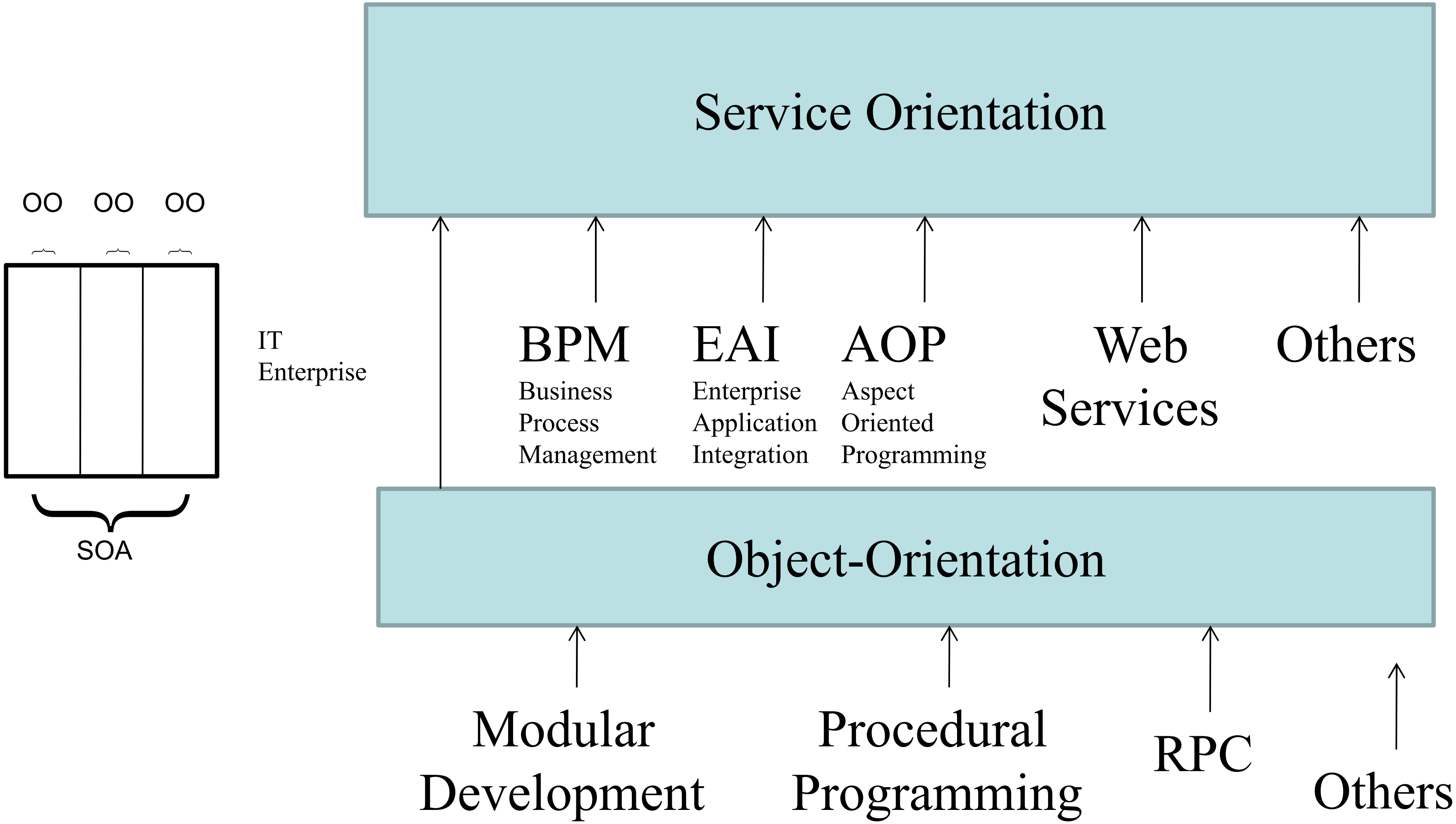
- Understand the source of troubles by qualitative data and justify them by quantitative data
- There are several useful measures(FP measures) calculated by Function Point
- **Productivity:** Hours per FP, Information technology productivity, Rate of delivery, Delivered functionality and developed functionality
- **Quality:** Functional requirement size, completeness, Rate of change, Defect removal efficiency, Defect density, Test case coverage, Volume of Documentation
- **Financial:** Cost per FP, Repair cost ratio, portfolio asset value
- **Maintenance:** Maintainability, Reliability, Assignment scope, Rate of growth, portfolio size, Backfire value, Stability ratio

David Garmus, David Herron, " Function Point Analysis" ADDISON-WESLEY, 2001.

Summary on PM

- Not good to follow the successive phases in a linear way(waterfall). Better to overlap activities of phases(iterative)
- Still something wrong!
- Traditional PM techniques pursue the efficiency, use up human resources to the maximum
- Should take account of capacity and load of a project team.

OOAD, SOA and Cloud



JAIST Koichiro Ochimizu

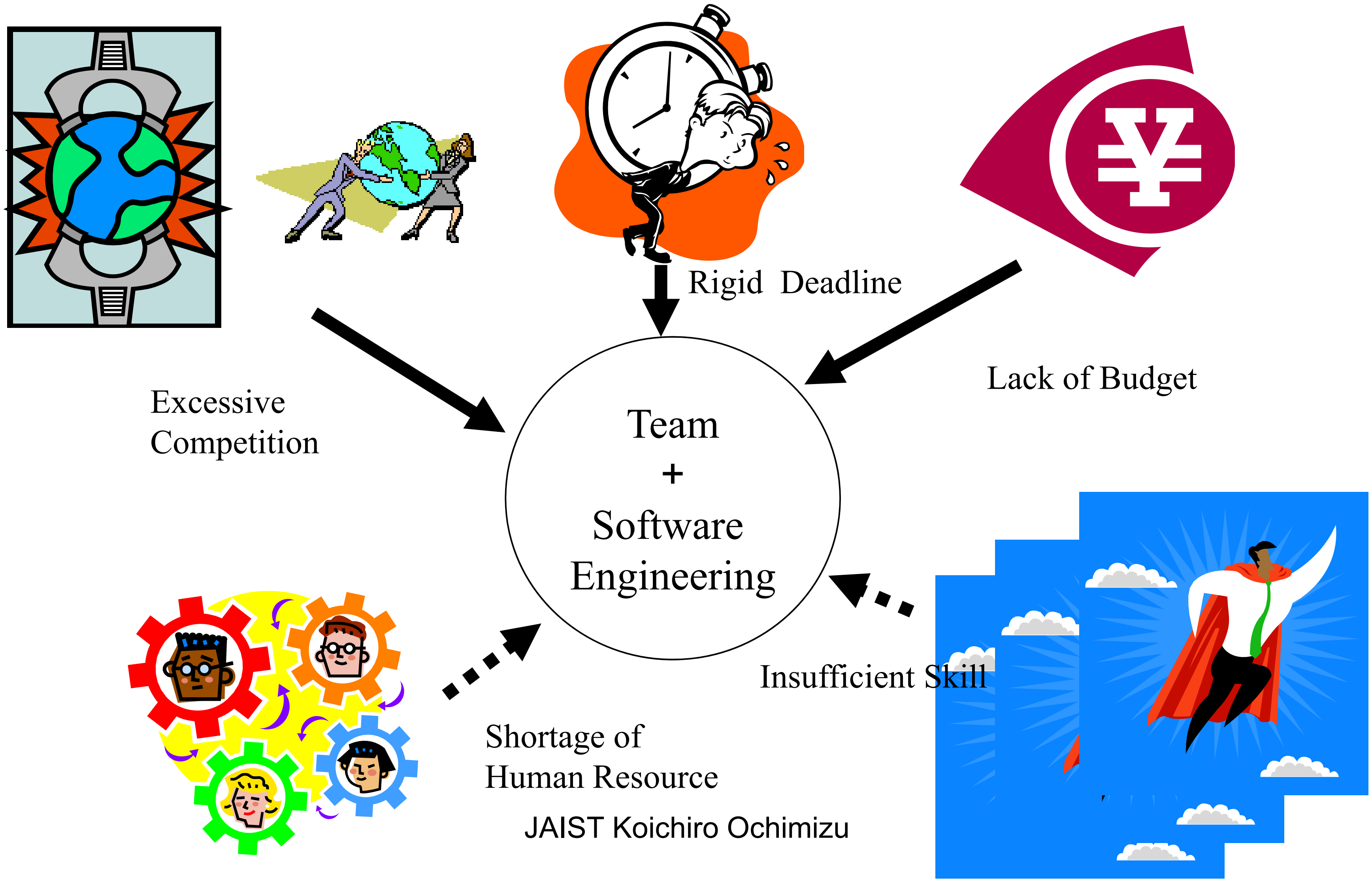
Thomas Erl, “SOA Principles of Service Design”, PRETICE HALL, 2008

OOAD, SOA, and Cloud

- The major benefit of the concept behind cloud computing is that the average user does not require a compute that is extremely powerful to handle complex database indexing tasks that server farms can
- Instead, with the use of broadband, users can easily connect to the cloud, which would commonly be referred to as the point of contact with the larger network.
- With this point of contact, cloud computing users from all across the world can reap the benefits of enormous processing power without major capital or technical know-how.
- Benefits: Flexibility, Scalability, Capital Investment, Portability
- Drawback: Dependability, Security, Little or No Reference

"Cloud Computing – The Complete Cornerstone Guide to Cloud Computing Best Practices", The Art of Services, 2008

Wrong Usage of Software Engineering



Software Engineering is a tool to increase the Capacity of Software Team