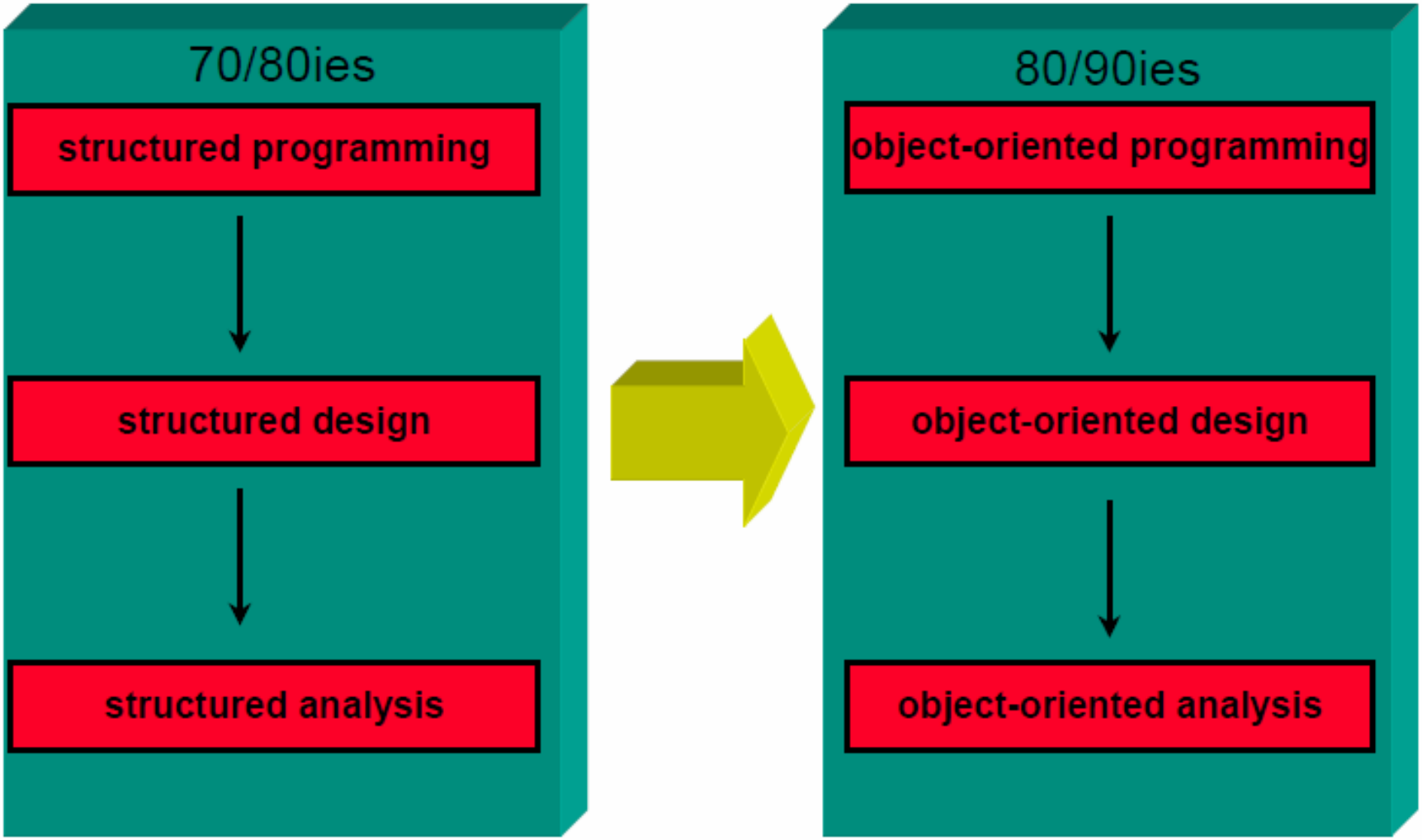
Unified Modeling Language (UML) dan Objectory Method

Pengembangan Sistem Berorientasi Obyek (KOM334)
Departemen Ilmu Komputer IPB
2010

Evolution of OO Development Methods



History of OOAD leading to UML

1970 First object-oriented languages (Simula-67, Smalltalk).

1980 More than 50 different OOAD languages cause the users trouble to find complete and appropriate tools.

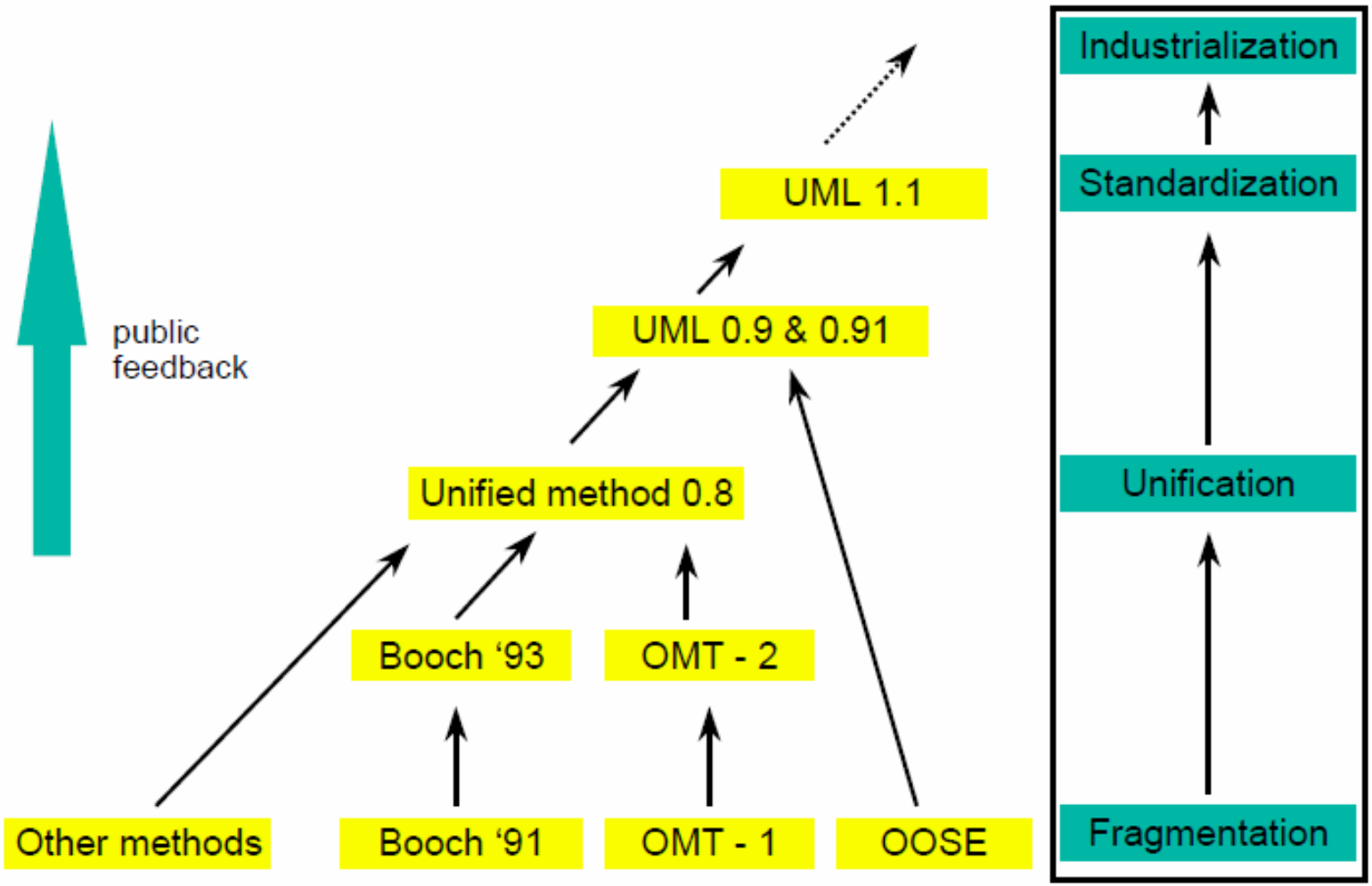
1992 New iterations of methods appear.
Booch '93, OOSE (Jacobson), OMT-2 (Rumbaugh)

1995 Unification, UML 0.9 by Booch, Rumbaugh

1997 Standardization, UML 1.1 by Booch, Rumbaugh, Jacobson

Object Management Group (OMG) adapts UML as OOAD standard

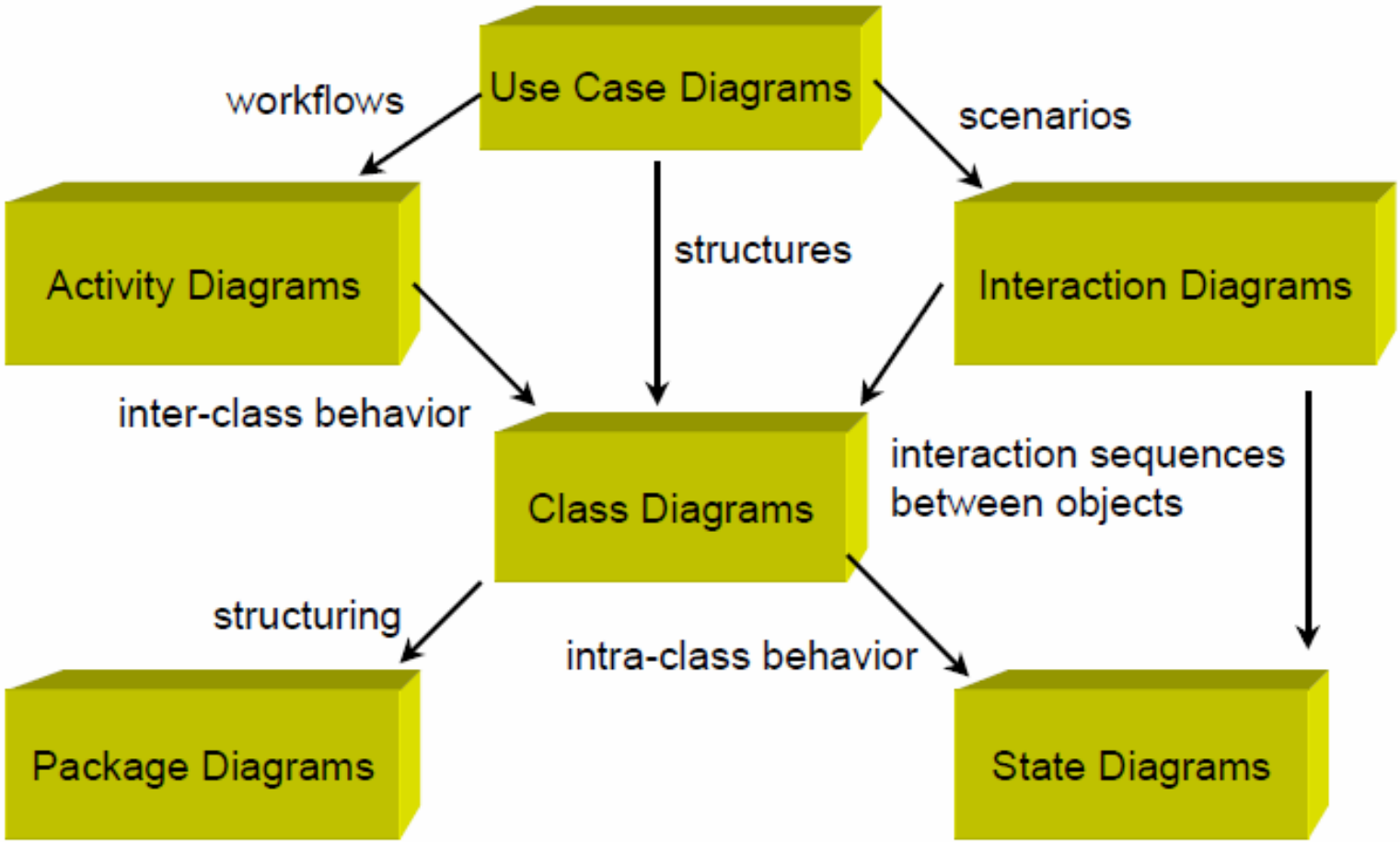
History of UML



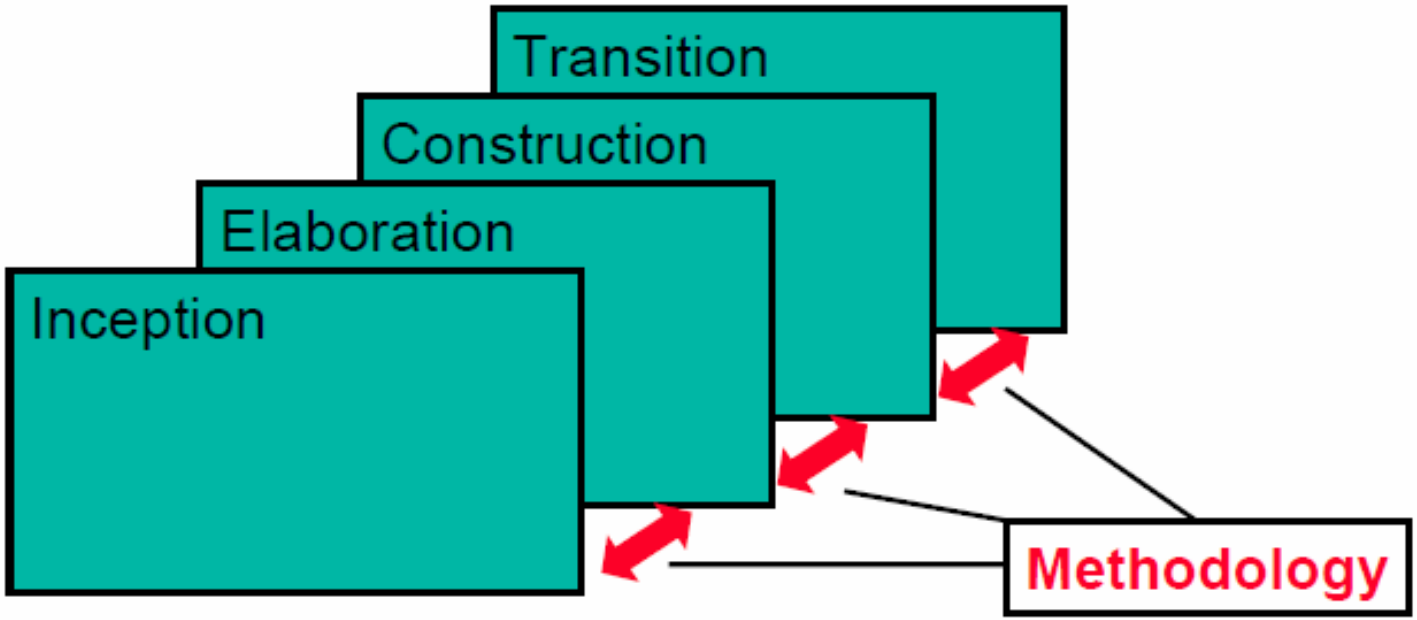
UML Diagrams (1)

○ Use Case Diagrams	Nodes:	Actor, Use (case)
	Links:	Involvement, Extension, Usage
○ Class Diagrams	Nodes:	Class
	Links:	Association, Generalization
○ Interaction Diagrams	Nodes:	Object
	Links:	Message, Lifeline
○ State Diagrams	Nodes:	State, Sub-State
	Links:	Transition
○ Activity Diagrams	Nodes:	Activity
	Links:	Guard, Synchronization
○ Package Diagrams	Nodes:	Package
	Links:	Dependency
○ Deployment Diagrams	Nodes:	Processor, Node
	Links:	Dependency

UML Diagrams (2)



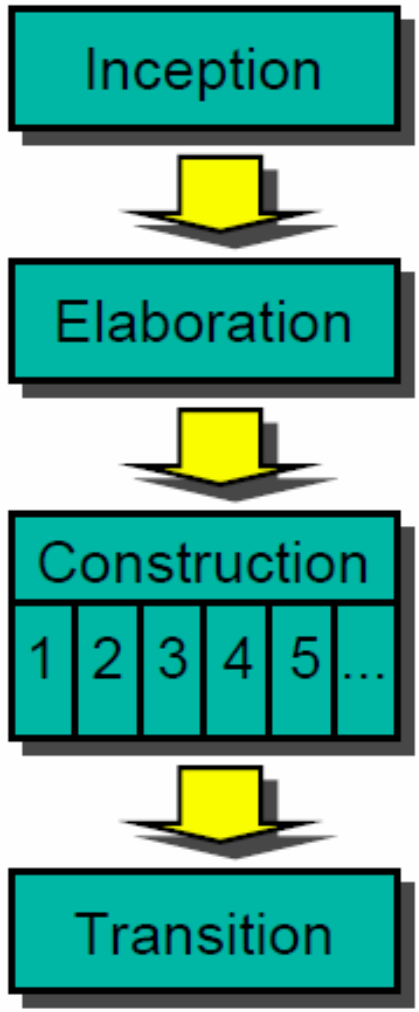
Objectory and UML



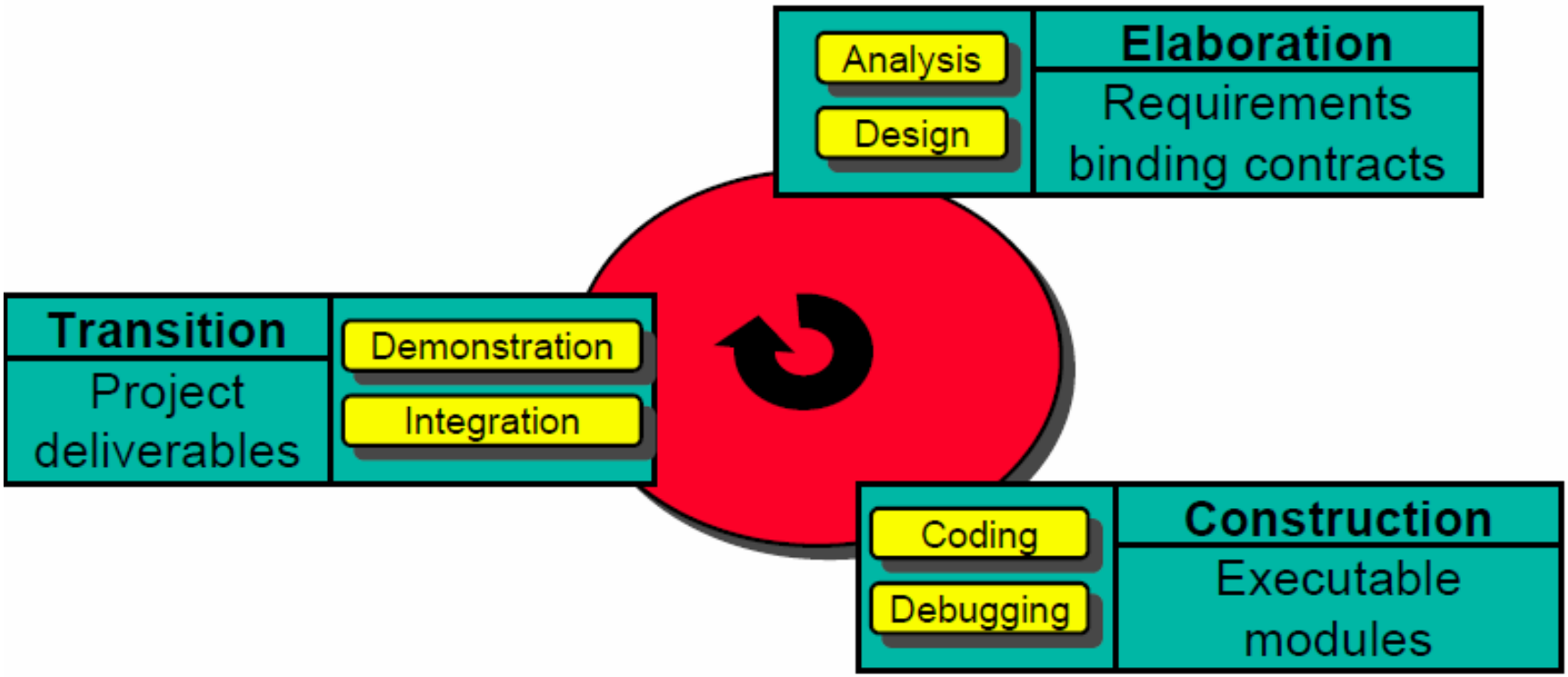
Software development is a process in **phases**.
This process has to follow a **methodology**.
Each phase is supported by graphs & **diagrams**.
There are different kinds of documents and various usage of them.
UML is an essential **language** for diagrams, offering computer support as well as the right patterns for the various stages of refinement and viewpoints.

Objectory: The UML Software Development Process

- Inception **establishes** the **business rationale** for the project and **decides** on the **scope** of the project.
- Elaboration is the phase where you **collect** more detailed **requirements**, do high-level **analysis** and **design** to establish a baseline **architecture** and create the **plan** for construction.
- Construction is an **iterative** and **incremental** process. Each iteration in this phase builds production-quality software **prototypes**, tested and integrated as subset of the requirements of the project.
- Transition contains beta **testing**, performance **tuning** and user **training**.



Objectory: Incremental Iterations



First Step: Inception

- Inception can take many **forms**:
  - For some projects it is a **chat** at the coffee machine.
  - For bigger projects it is a full-fledged feasibility **study** that takes months.
- During the inception phase you work out the **business case** for the project:
  - Derive how much the project will **cost**.
  - Estimate how much **profit** it will bring in.
- Some **initial analysis** is required to get a sense of the project's **scope** and **size**.
- Inception should be a **few days** of work to consider if it is worth doing a few months of work of deeper investigation during elaboration.
- At the point of inception the project **sponsor** agrees to no more than a serious look at the project:

Do we go ahead with the project?

Second Step: Elaboration

- Starts after you have received the "go-ahead to start the project" **agreement**.
- At this stage you typically have only a **vague** idea of the **requirements**.

  "We are going to build the next generation customer support system for the Watts Galore Utility Company. We intend to use object-oriented technology to build a more flexible system that is more customer oriented - specifically, one that will support consolidated customer bills".

- Elaboration is the point where you want better **understanding** of the problem:
  - **What** is it you are actually going to build?
  - **How** are you going to build it?
  - What **technology** are you going to use?
- Elaboration includes to have a careful and thorough look at the possible **risks** in your project:

What are the things that could derail you?

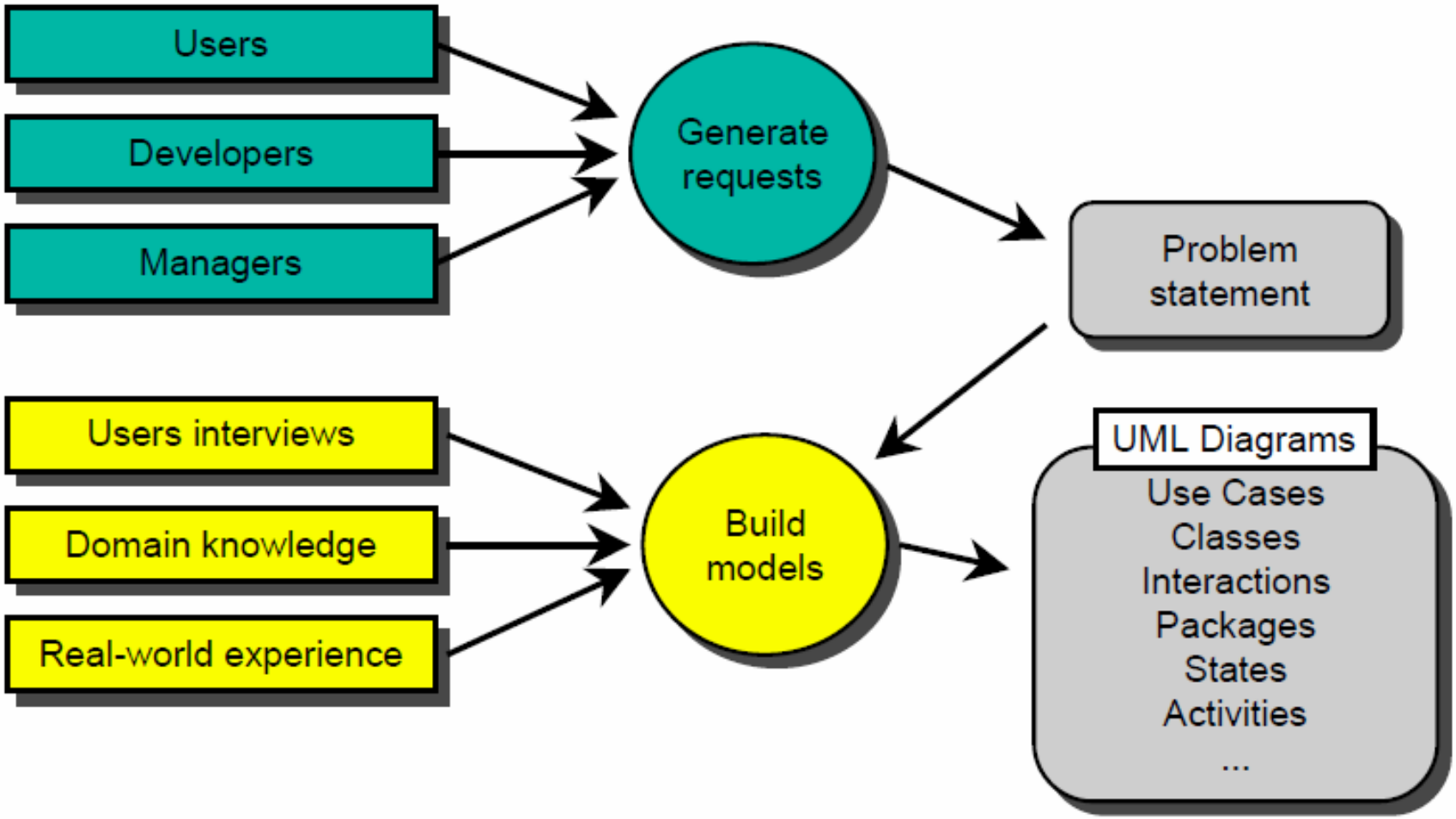
Elaboration: Systems Analysis

Rationale: Finding and **fixing** a **fault** after software delivery is 100 times more **expensive** than finding and fixing it during systems analysis or early design phases.

- The **goal** of analysis is to develop a **model** of what the system will do.
- The analysis should include information required to **understand** what is meaningful from a **real world** system.
- The **client** of a system should **understand** the analysis **model**.
- The analysis phase delivers a **base** from which further **details** are derived in the design phase.
- Analysis provides the **requirements** and the real-world environment in which the software system will exist.

Object-oriented analysis forces a **seamless** development **process** with no discontinuities because of **continuos refinement** and progressing from analysis through design to implementation.

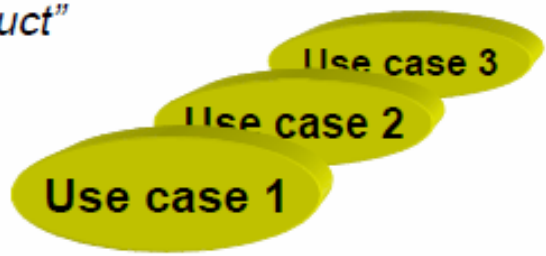
Analysis: Actors, Steps, Deliverables



Elaboration: Requirements Capture

Identify typical **use cases** (see chapter 2.1) of the system you are going to build.

- For a person using a **database** a **typical use case** would be:
  - *"list all customers who have ordered a certain product"*
  - *"create a list with my top 10 customers"*
  - *"I want fax-letters to be sent automatically"*



- A **developer** responds with specific **cost estimates**:
  - *"The top 10 customer list can be developed in a week."*
  - *"Creating the auto-fax function will take two months."*
- User and developer **negotiate** about the **priorities**:
  - Developer: *"I could start with the sold - products list."*
  - User: *"I definitely need the top 10 customers list first."*

Elaboration: Planning

Schedule use cases to specific **iterations** and dates of **delivery**.

- The **users** should indicate the level of **priority** for each use case.
  - *"I absolutely must have this function for any real system."*
  - *"I can live without this function for a short period."*
  - *"It is an important function, but I can survive without it for a while."*
- The **developers** should consider the **architectural** risk.
  - Do not omit use cases which later cause you to do a lot of rework to fit them in.
  - Concentrate to the use cases which are technologically most challenging.
- The **developers** should be aware of the **schedule** risks.
  - *"I'm pretty sure I know how long it will take."*
  - *"I can estimate the time only to the nearest man-month."*
  - *"I have no idea."*

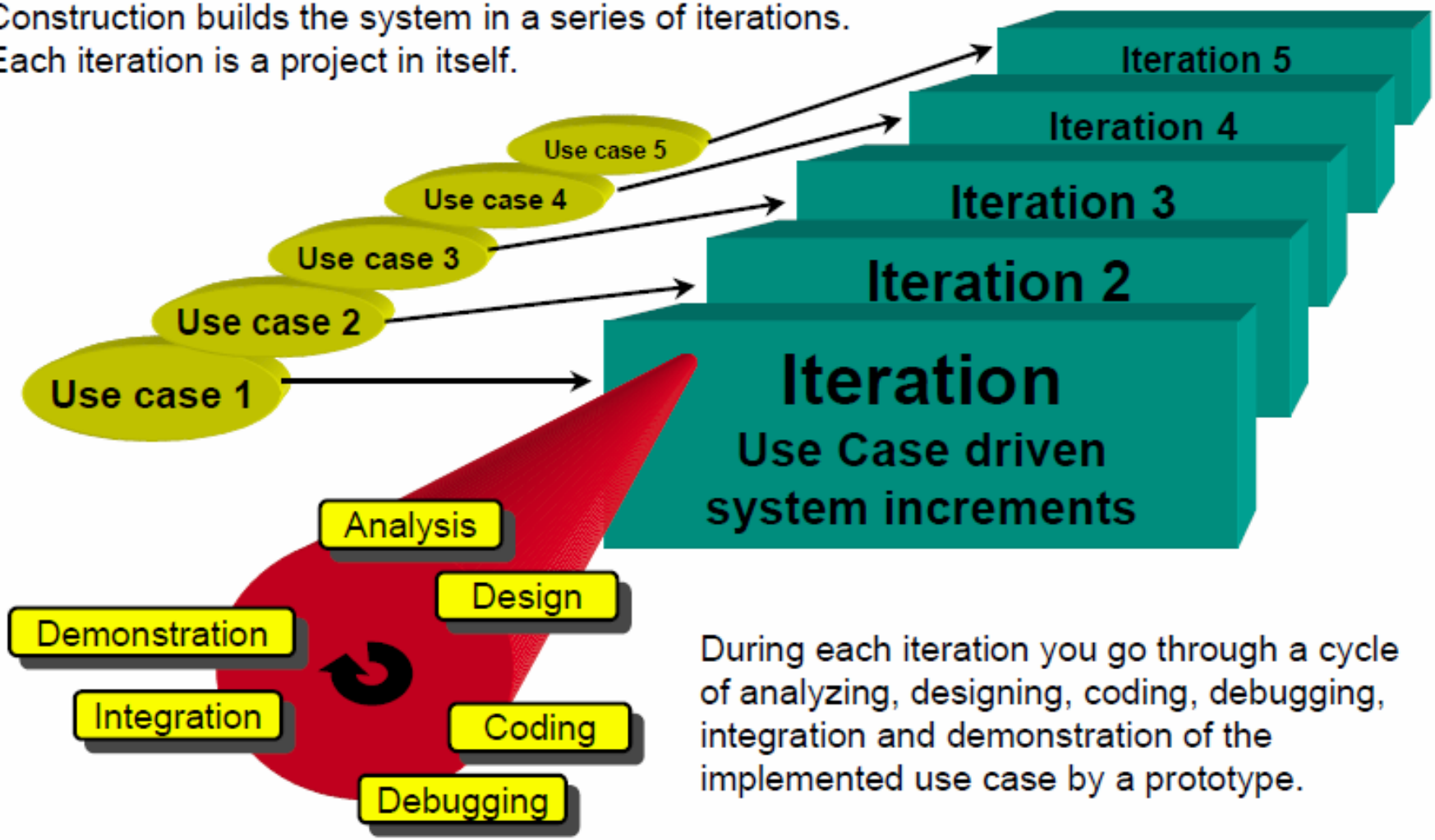
Planning: Estimate

- Once the use cases are assigned, the **length** of each **iteration** should be estimated to the nearest person-week.
- In performing this estimate assume you **need** to do
  - analyzing
  - designing
  - coding
  - unit testing
  - integration
  - documentation

The estimates should be done by the **developers**, **not** by the **managers**.

Third Step: Construction

Construction builds the system in a series of iterations.
Each iteration is a project in itself.

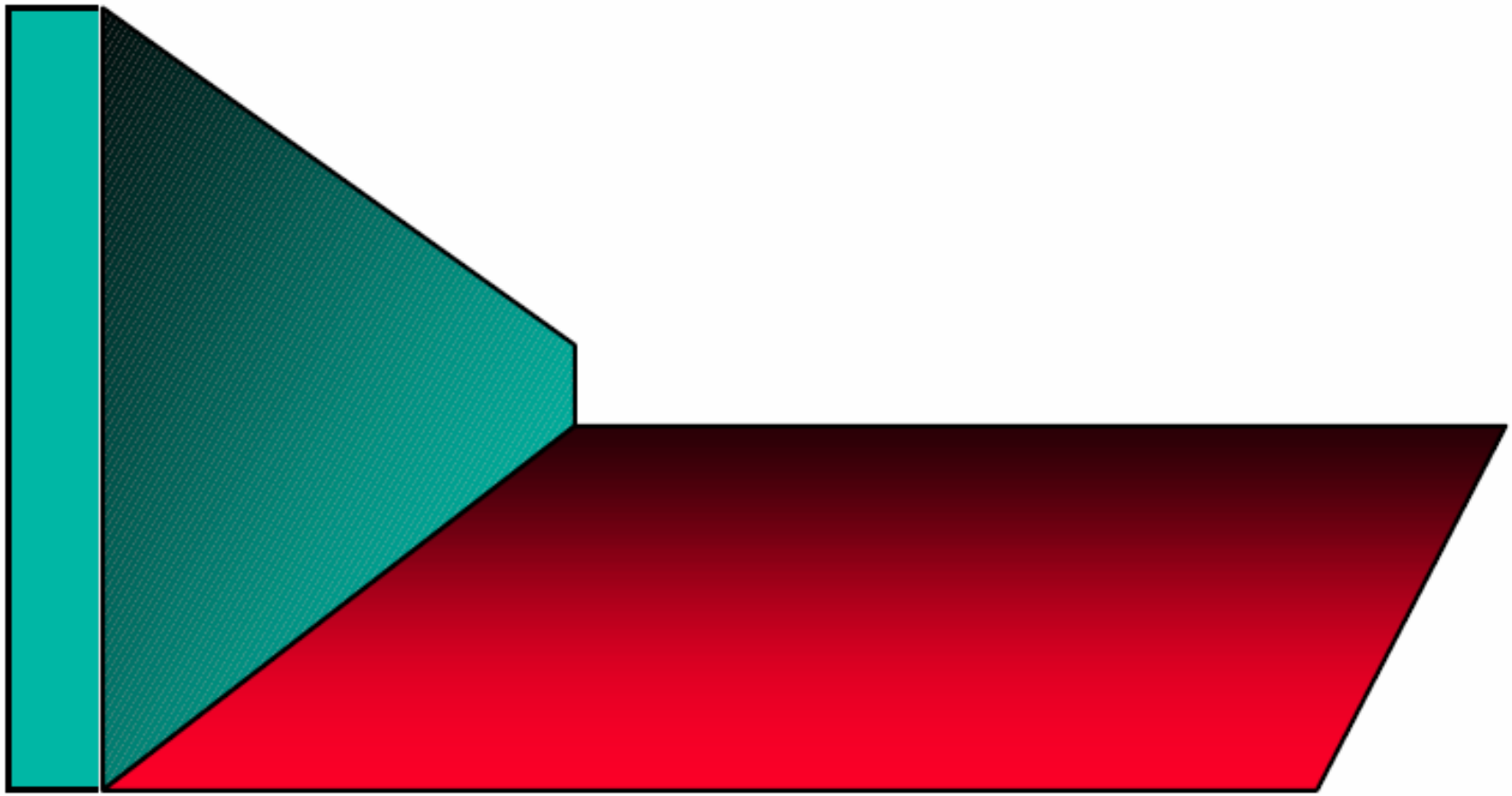


During each iteration you go through a cycle of analyzing, designing, coding, debugging, integration and demonstration of the implemented use case by a prototype.

Construction: Iterations

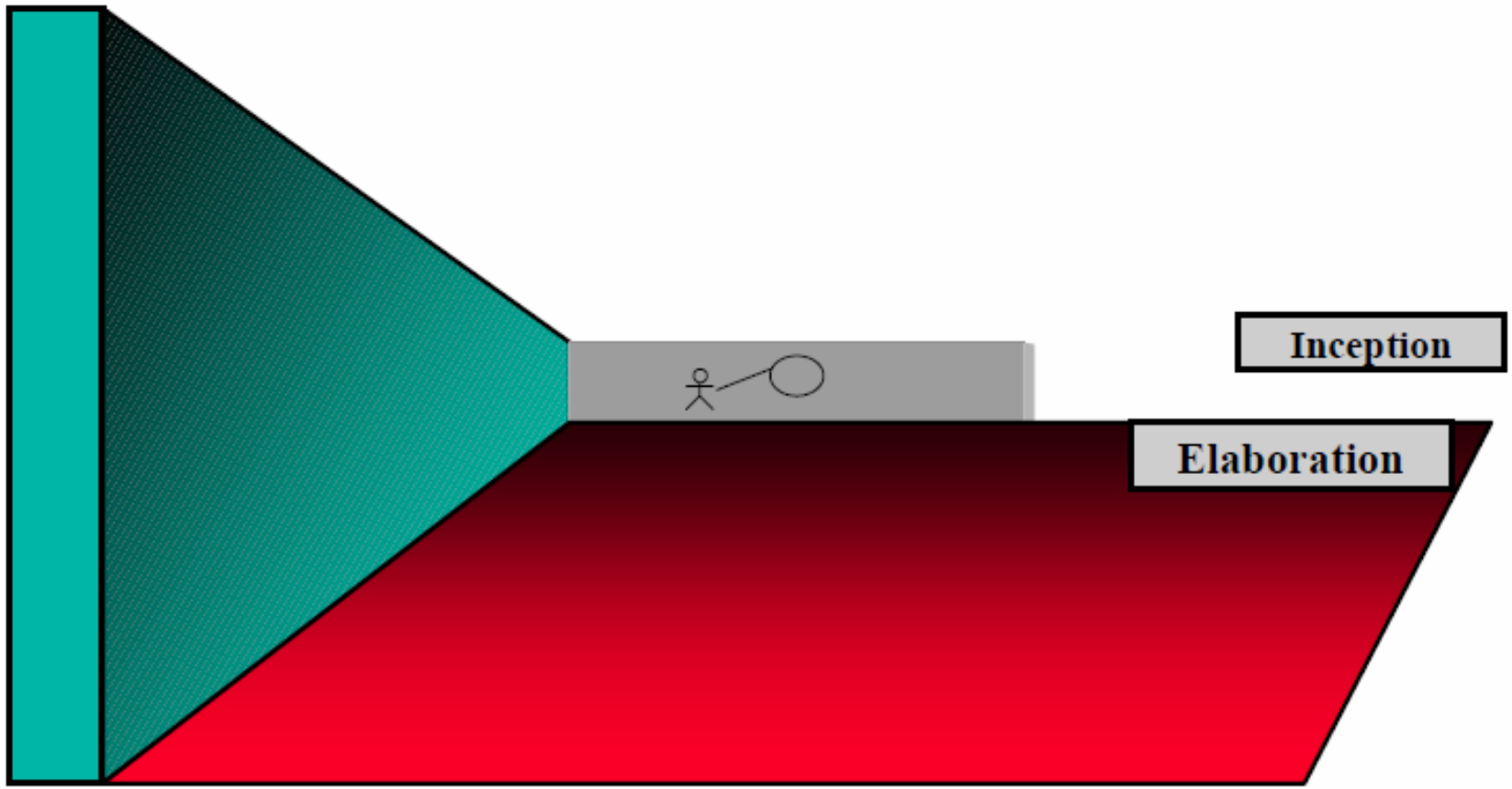
- Finish the iteration with a **demo** to the user and perform system **tests** to confirm that the use cases have been built correctly.
- Iterations within a construction are both, incremental and iterative.
  - Iterations are **incremental** in function. Each iteration builds on the use cases implemented in the previous iteration
  - They are **iterative** in terms of the code base which will be rewritten to make it more flexible.
- Do not underestimate the **testing** phase.
  - Write test code.
  - Separate the test into unit and test code.
  - Unit tests should be written by the developers.
  - Apply **all** tests after each iteration.

Diagrams and Process



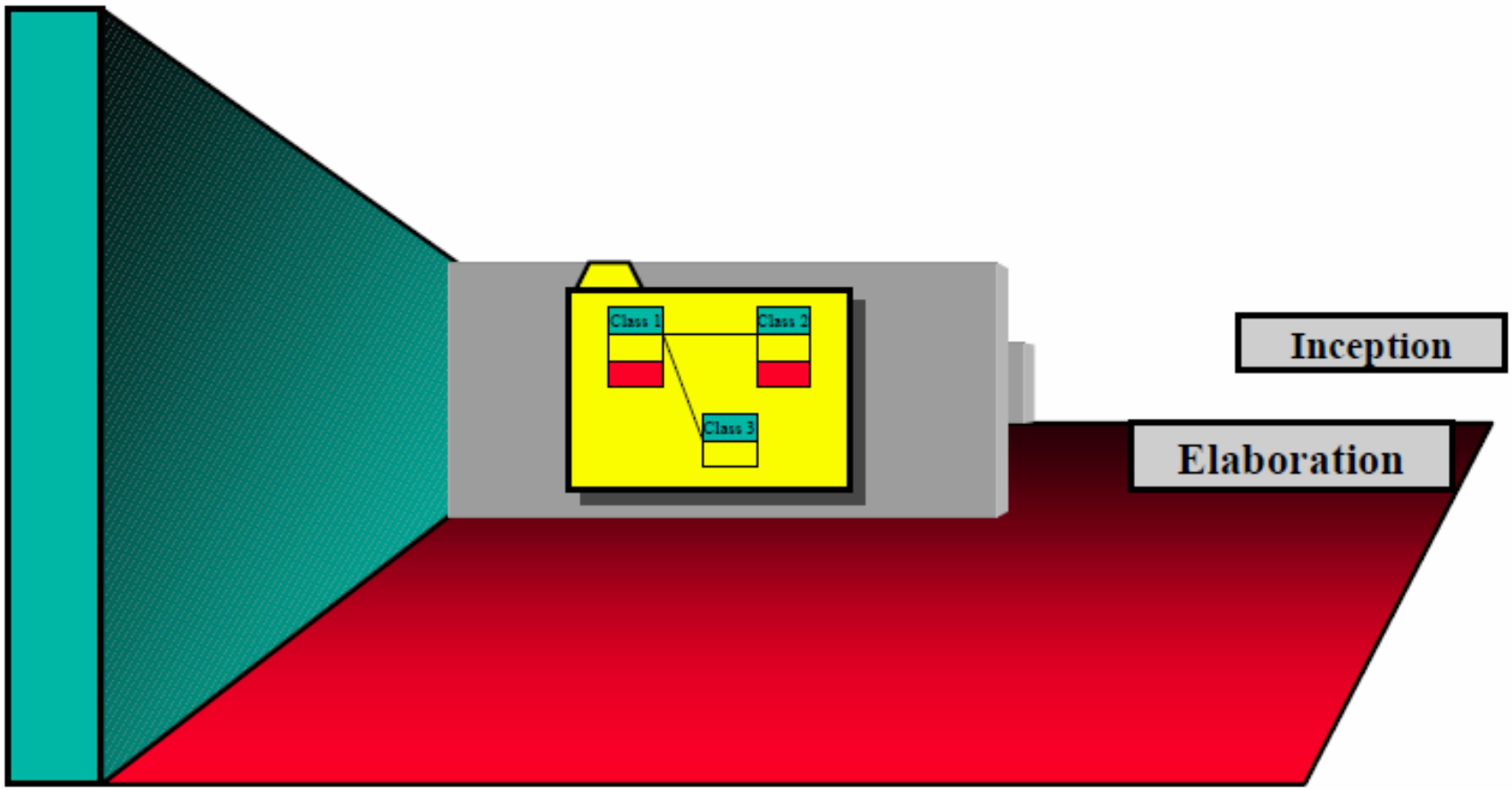
Diagrams and Process

Use Case Diagrams



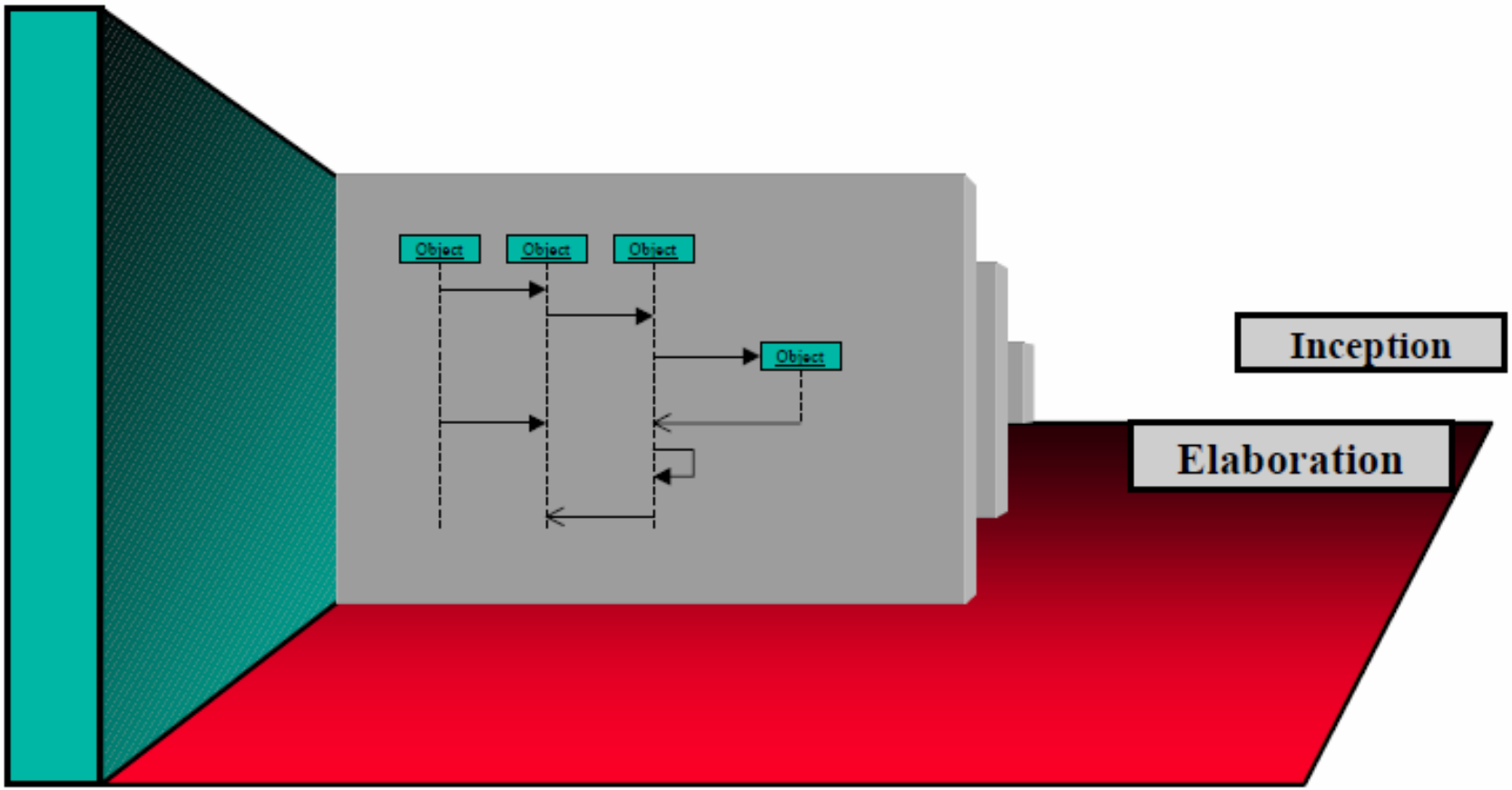
Diagrams and Process

Class & Package Diagrams



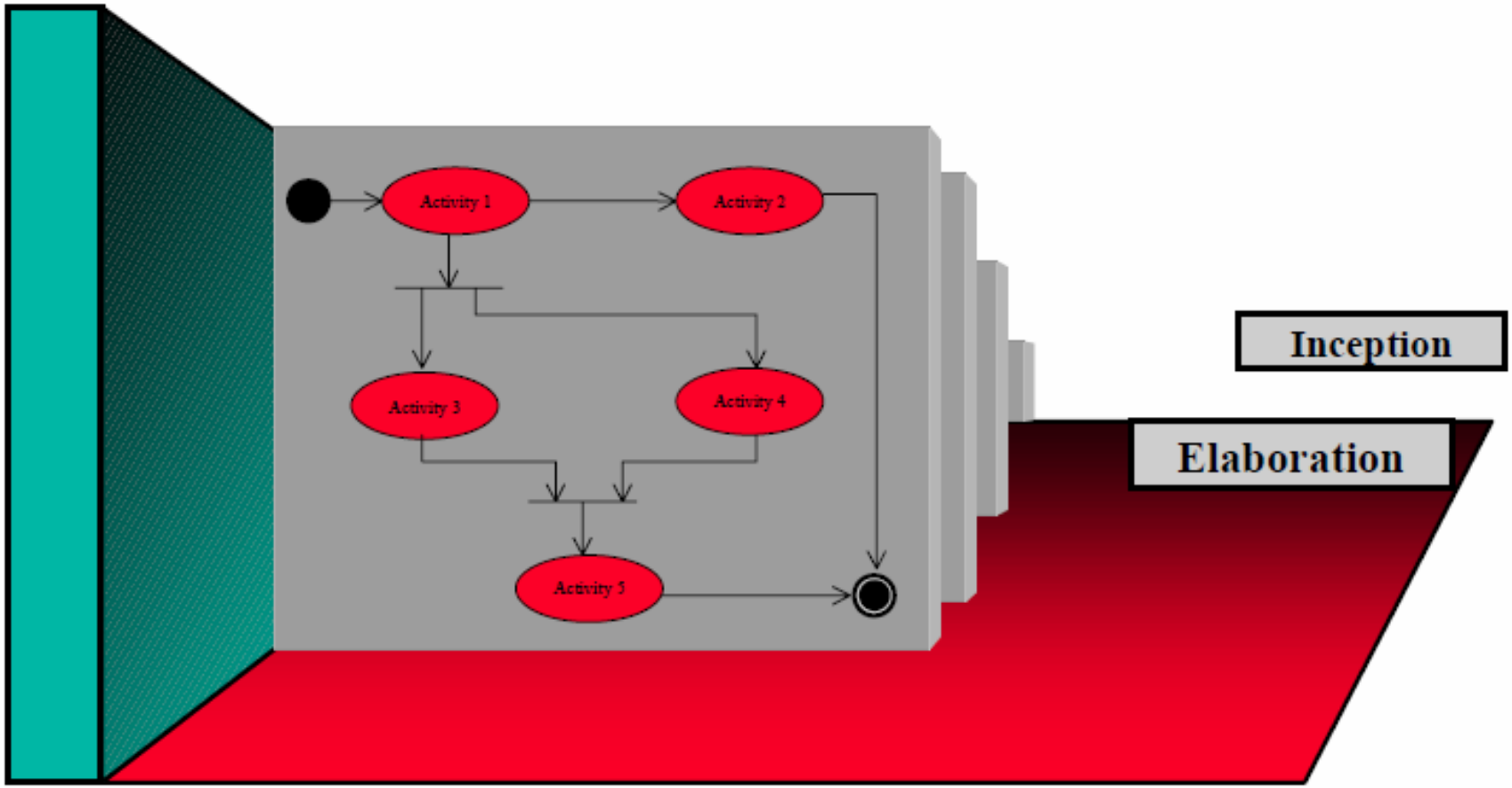
Diagrams and Process

Interaction Diagrams (Scenarios)



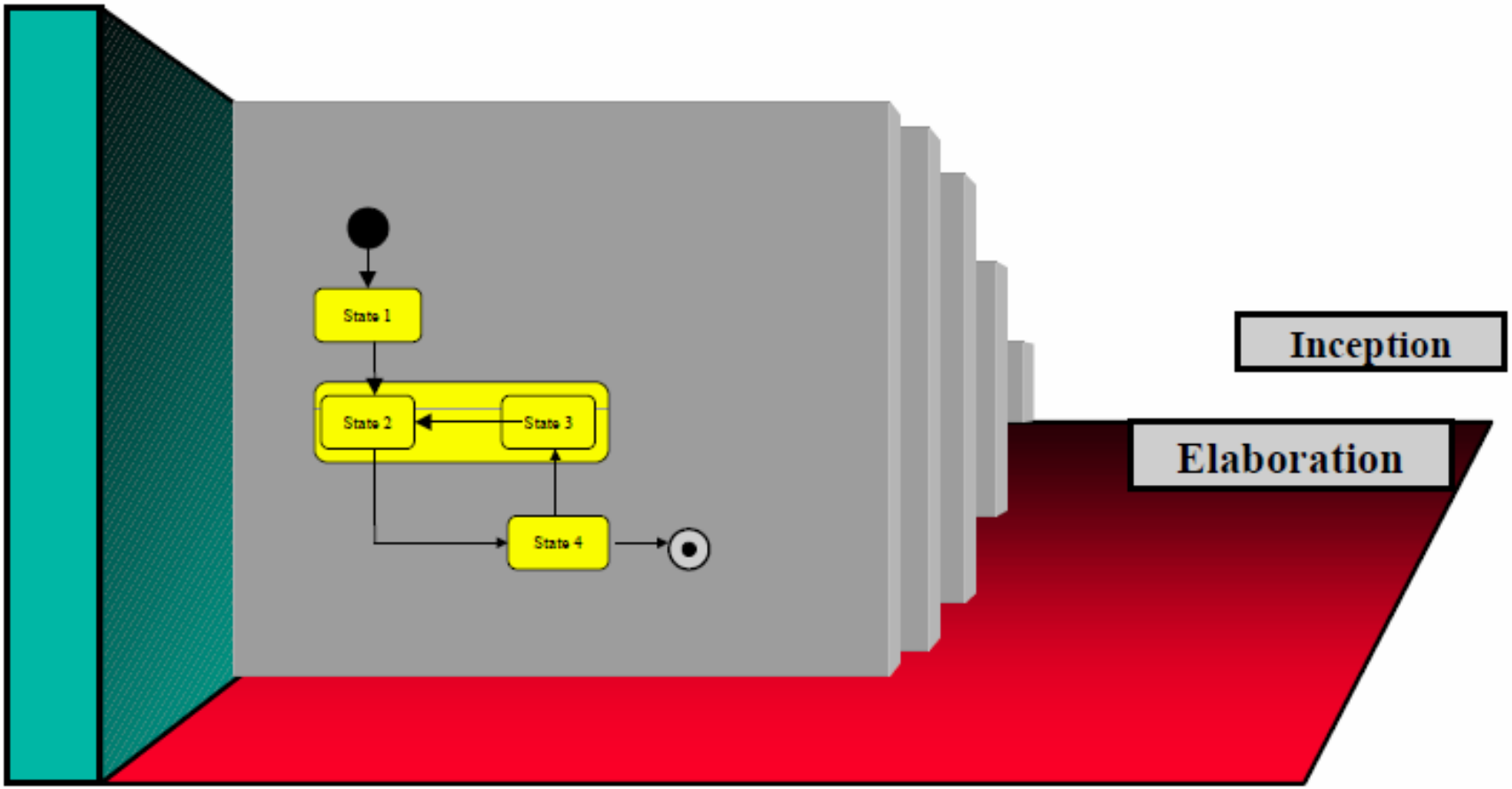
Diagrams and Process

Activity Diagrams (Workflow, Interclass Behavior)



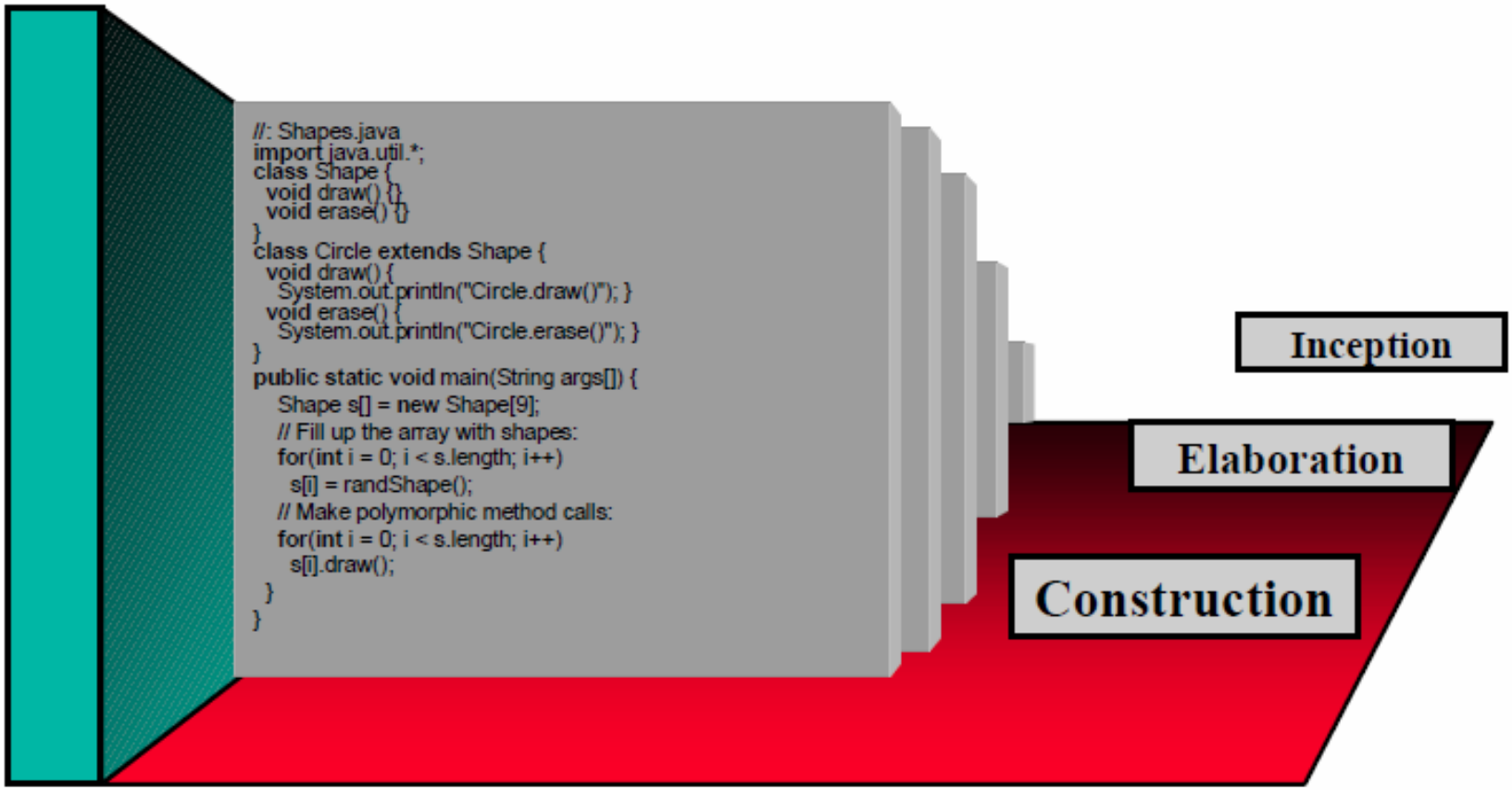
Diagrams and Process

State Transition Diagrams (Intraclass Behavior)



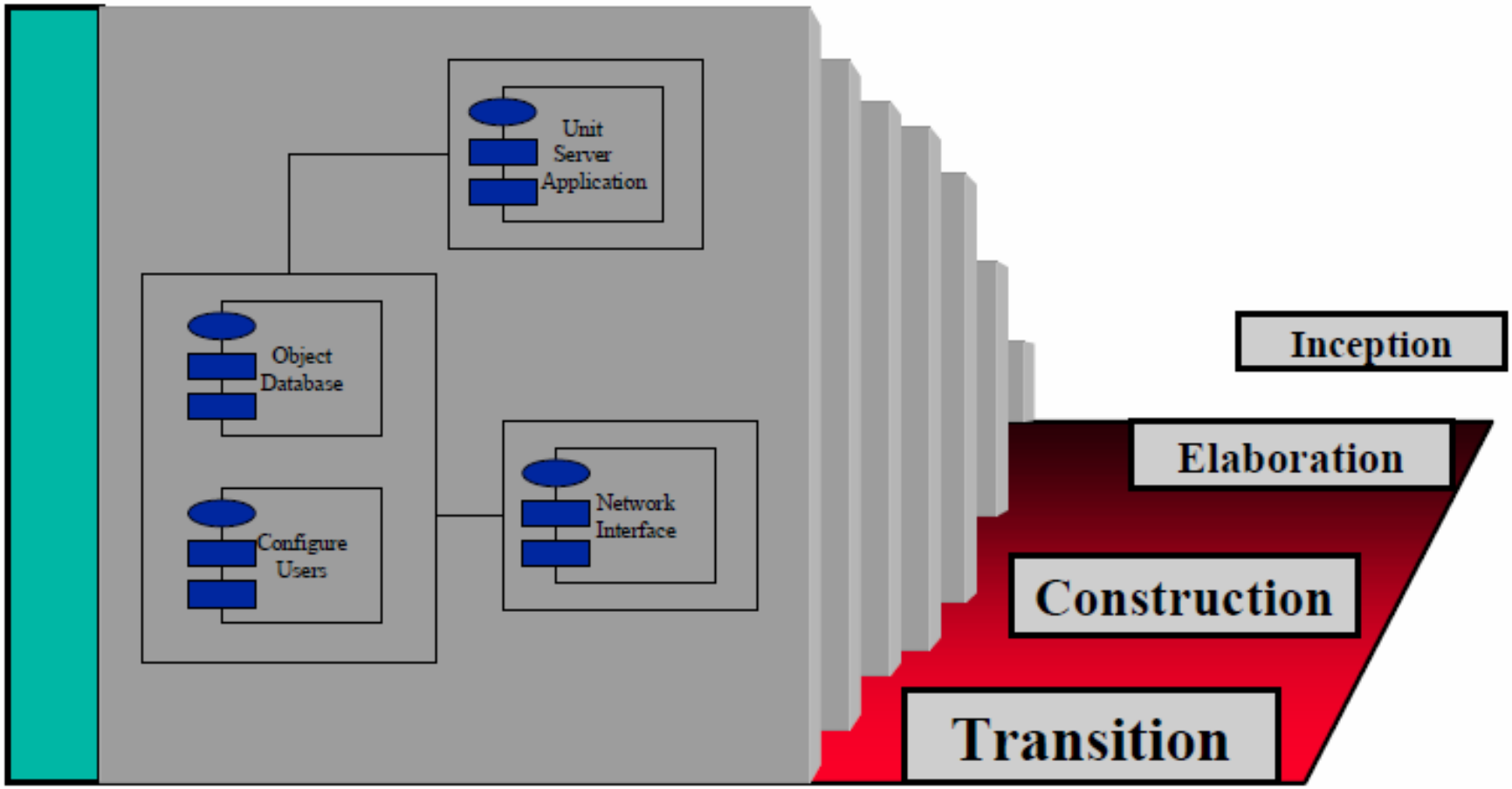
Texts and Process

Source Code



Diagrams and Process

Deployment Diagrams



Any Questions?

Source: