UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4351 Fall 2023

Software Engineering

M & W 4 to 5:30 PM

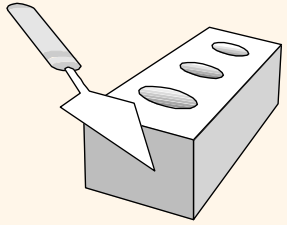
Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE

COSC 4351

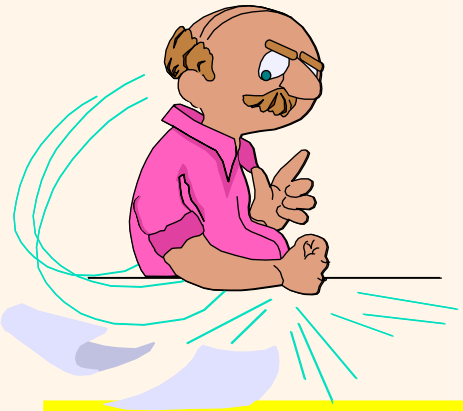
4 to 5:30



PLEASE

LOG IN

CANVAS



Youyi [A-L]

Kevin [M-Z]

Please close all other windows.

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	
08.30.2023 (W 4 to 5:30) (4)		Tutorials 1 on WHAT tools (UML & ERD Modeling)		UML Modeling CANVAS Assignment	
09.06.2023 (W 4 to 5:30) (5)		Lecture 3: Software Development Process			
09.11.2023 (M 4 to 5:30) (6)		EXAM 1 REVIEW (CANVAS) (ZyBook)	Download ZyBook: Sections 1-5		
09.13.2023 (W 4 to 5:30) Optional (7)					Q & A Set 1 topics.
09.18.2023 (M 4 to 5:30) (8)					EXAM 1 (CANVAS) (ZyBook)

Class 3

SDLC Models

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment

CLASS PARTICIPATION 20 points

20% of Total

One Page Summary of SDLC MODELS Papers

CLASS PARTICIPATION 20% Module | Available until Aug 28 at 4:00pm | Due Aug 28 at 4pm | 100 pts

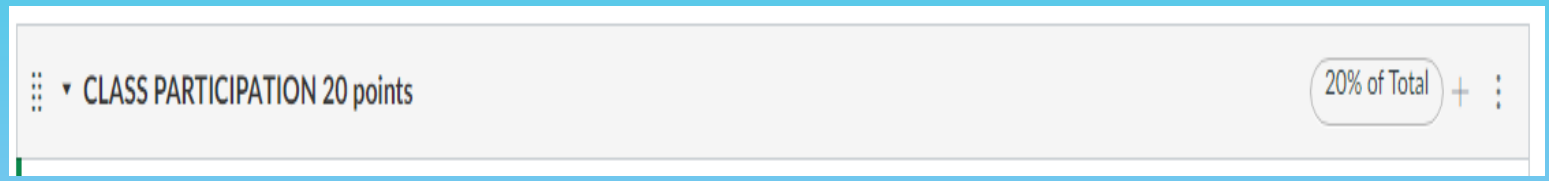
One Page Summary of SDLC MODELS Papers

Published | Edit

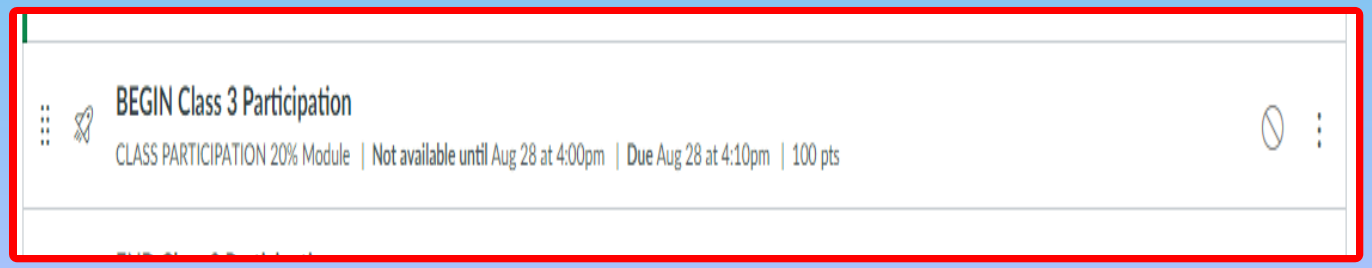
Please attach your electronic version.

From 4:00 to 4:10 – 10 minutes.

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	



PASSWORD: CLASS 3



VH, publish.

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	

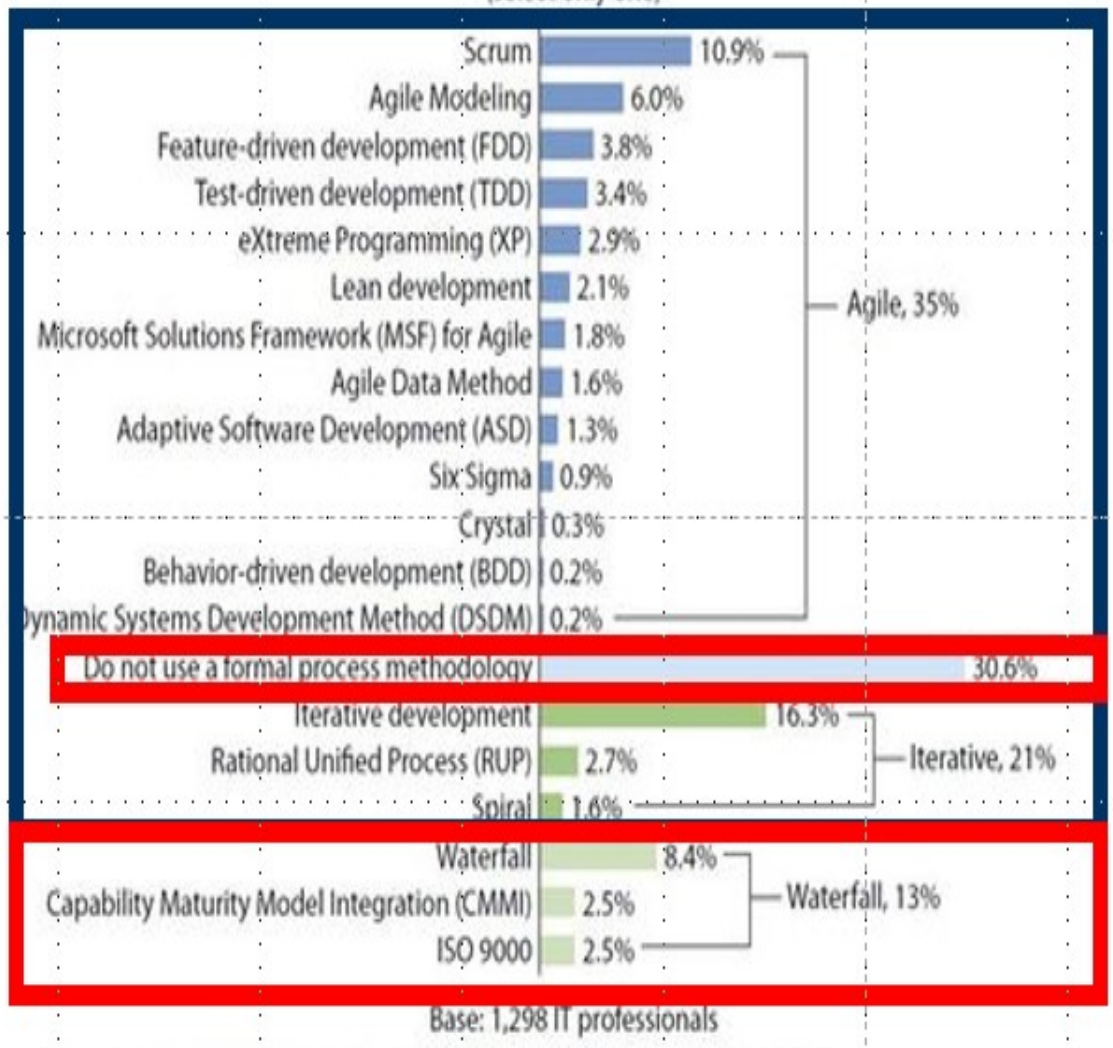
From 4:10 to 4:55 – 45 minutes.

COSC 4351 Company Fundamental Software Engineering

SDLC Models

2011

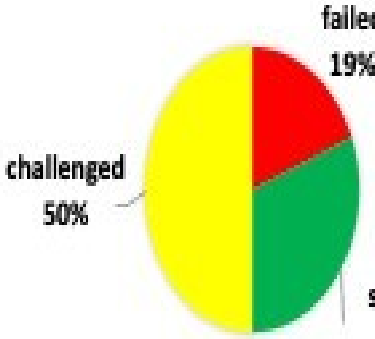
Figure 1 Agile Adoption Has Reached Mainstream Proportions



Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009

Project Success
Quick Reference Card

Based on CHAOS 2020: Beyond Infinity Overview. January'2021, QRC by Henny Portman



Modern measurement
(software projects)

Good Sponsor, Good Team, and Good Place are the only things we need to improve and build on to improve project performance.

2022

The Good Place is where the sponsor and team work to create the product. It's made up of the people who support both sponsor and team. These people can be helpful or destructive. It's imperative that the organization work to improve their skills if a project is to succeed. This area is the hardest to mitigate, since each project is touched by so many people. Principles for a Good Place are:

- The Decision Latency Principle
- The Emotional Maturity Principle
- The Communication Principle
- The User Involvement Principle
- The Five Deadly Sins Principle
- The Negotiation Principle
- The Competency Principle
- The Optimization Principle
- The Rapid Execution Principle
- The Enterprise Architecture Principle

Successful project Resolution by Good Place Maturity Level:

highly mature	50%
mature	34%
moderately mature	23%
not mature	23%

The Good Team is the project's workhorse. They do the heavy lifting. The sponsor breathes life into the project, but the team takes that breath and uses it to create a viable product that the organization can use and from which it derives value. Since we recommend small teams, this is the second easiest area to improve. Principles for a Good Team are:

- The Influential Principle
- The Mindfulness Principle
- The Five Deadly Sins Principle
- The Problem-Solver Principle
- The Communication Principle
- The Acceptance Principle
- The Respectfulness Principle
- The Confrontationist Principle
- The Civility Principle
- The Driven Principle

Successful project Resolution by Good Team Maturity Level:

highly mature	66%
mature	46%
moderately mature	21%
not mature	1%

The Good Sponsor is the soul of the project. The sponsor breathes life into a project, and without the sponsor there is no project. Improving the skills of the project sponsor is the number-one factor of success – and also the easiest to improve upon, since each project has only one. Principles for a Good Sponsor are:

- The Decision Latency principle
- The Vision Principle
- The Work Smart Principle
- The Daydream Principle
- The Influence Principle
- The Passionate Principle
- The People Principle
- The Tension Principle
- The Torque Principle
- The Progress Principle

Successful project Resolution by Good Sponsor Maturity Level:

highly mature	67%
mature	33%
moderately mature	21%
not mature	18%

22 software development trends for 2022

https://blog.codacy.com/software-development-trends-2022/

Believe it or not, the year 2022 is right around the corner! So what does next year have in store for the rapidly changing, ever-evolving software world? From code reviews to DevOps, software testing, and tech companies' culture, here are our 22 software development trends for 2022 🔮

#1 – A rise of automated code reviews

Tech companies increasingly see a well-defined code review process as a fundamental part of the software development process. Code reviews are among the best ways to improve code quality. Automated code review tools are increasing in popularity as more companies start to include them in their code review process, allowing developers to spend more time building new features instead of on code reviews. We can expect solutions like Codacy to see an increase in adoption in 2022.

#2 – A greater focus on software quality standards

Software solutions are increasingly embedded in our day-to-day lives and in most of the devices we use. As a result, there is a growing need for software to follow quality standards like the ones proposed by ISO. Examples include the ISO/IEC 25010 on software product quality or the ISO/IEC 27001 on information technology. Companies are also starting to see the advantages of ISO certification, like improved quality, more efficient processes, increased reputation, and enhanced client satisfaction.

Believe it or not, the year 2022 is right around the corner! So what does next year have in store for the rapidly changing, ever-evolving software world? From code reviews to DevOps, software testing, and tech companies' culture, here are our 22 software development trends for 2022 🔮

#3 – A pragmatic focus on coding standards

As companies and teams grow, they start to have a list of rules and guidelines for writing code. This list also incorporates language conventions and style consistency. Having a clear standard can help current developers, and it is also relevant for the onboarding of new developers. In 2022, we expect more companies to see the advantages of coding standards, not motivated by growing pains but by understanding its need from the early stages.

#4 – A steady replacement of legacy systems

We all know that legacy systems increase a company's risk of attacks or data breaches since security vulnerabilities are easier to exploit. As a result, companies are more focused on intentionally and periodically evaluating and modernizing their systems and technology stack and we anticipate this tendency to increase in 2022.

Believe it or not, the year 2022 is right around the corner! So what does next year have in store for the rapidly changing, ever-evolving software world? From code reviews to DevOps, software testing, and tech companies' culture, here are our 22 software development trends for 2022 🔮

#13 – A closer look into ethical AI

While AI is growing and it provides a high number of benefits, it also faces challenges and risks when it comes to data security, data privacy, and ethical uses of data. So, the tech industry will continue developing ethical principles, guidelines, and regulations to ensure we can expect ethical behaviors from the AIs the developers are building.

#15 – A run for the win from Python

Python's popularity has been increasing in the last few years, and we expect it to continue to grow in 2022. It is becoming the language of choice for developers building applications with AI and ML-based features, and with Artificial Intelligence and Machine Learning becoming more common, so is Python.

Believe it or not, the year 2022 is right around the corner! So what does next year have in store for the rapidly changing, ever-evolving software world? From code reviews to DevOps, software testing, and tech companies' culture, here are our 22 software development trends for 2022 🔮

#19 – An advanced CI/CD pipeline

An advanced CI/CD pipeline will enable continuous building, testing, and deploying iterative code changes. This reduces the chances of deploying new code based on a buggy or failed previous version. The pipeline will also minimize human intervention and, at the same time, help teams simplify and scale the test automation process.

CCNET

#20 – A growing need for outsourcing

It is increasingly challenging for companies in all industries to find talented IT professionals who can develop high-quality solutions within budget. So, several companies are outsourcing their development base, and we expect this tendency to increase in 2022. Some of the benefits of this approach include cost efficiency, faster turnaround, lower risk, and freeing up business resources.

#21 – A higher transparency about software

More clients and customers are demanding to know how companies make their software. This higher demand creates a need for more transparency regarding software development. Tech companies need to answer how they built the software and how long they will maintain it. Plus, product roadmaps will become more widely available for the public, with clients having a direct say on the developed features. At Codacy, we already have our product roadmap available for you to check out, and you can submit your ideas to our development team.

Agile

Software Development Life Cycle (SDLC) Models

"You've got to be very careful if you don't know where you're going, because you might not get there."

Yogi Berra

Yogi Berra	
	
Catcher / Outfielder / Manager	
Born: May 12, 1925 (age 88) St. Louis, Missouri	
Batted: Left	**Threw:** Right
MLB debut	
September 22, 1946 for the New York Yankees	
Last MLB appearance	
May 9, 1965 for the New York Mets	
Career statistics	
Batting average	.285
Home runs	358
Runs batted in	1,430
Teams	
As player	
• New York Yankees (1946–1963)	

Capability Maturity Model (CMM)

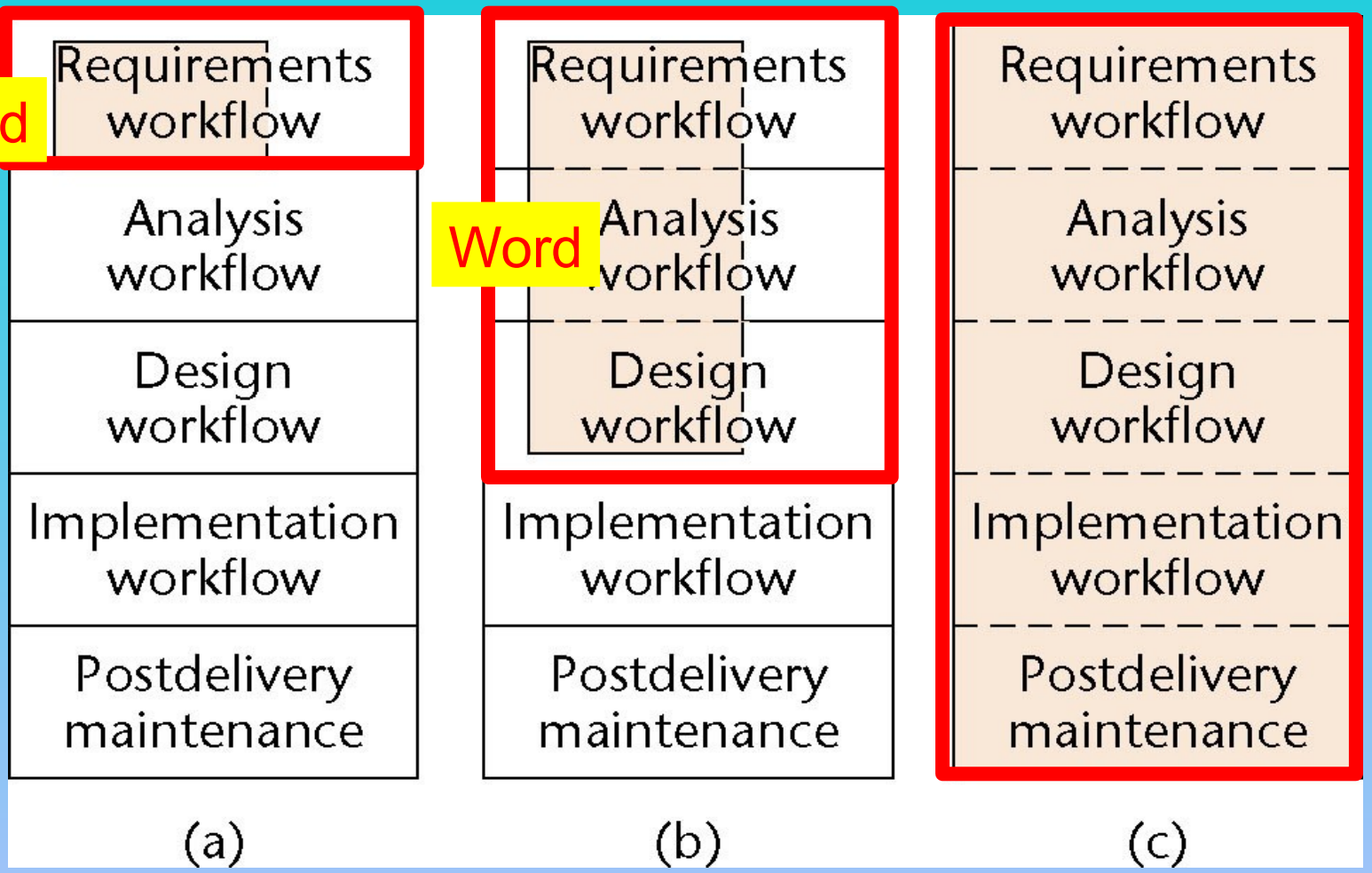
A benchmark for measuring the **Maturity** of an organization's **software Process**

CMM defines **5 levels of Process Maturity** based on certain Key Process Areas (**KPA**)

ISO/IEC 25010 on software product quality

ISO/IEC 27001 on information technology

Taxonomy of CASE (Software assisting with Software Development)



(a) **Tool** (b) **Workbench** (c) **Environment**

CMM Levels & (KeyPprocessAareas s)

Level 5 – Optimizing (< 1%)
- -- Process change management
- -- Technology change management
- -- Defect prevention

Level 4 – Managed (< 5%)
- -- Software Quality management - SQA
- -- quantitative Process management

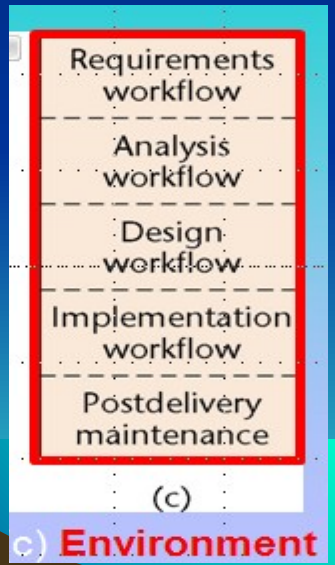
Level 3 – Defined (< 10%)
- -- peer Reviews
- -- intergroup coordination - CCNET
- -- software project engineering
- -- integrated software management - SVN
- -- training program - TUTORIALs
- -- organization Process definition
- -- organization Process focus

Level 2 – Repeatable (~ 15%)
- -- software configuration management - SVN
- -- Software Quality Assurance - SQA
- -- software project tracking and oversight - MS PROJECT
- -- software project Planning- SPMP
- -- Requirements management - SRS

Level 1 – Initial (~ 70%)
- -- no Process - adhoc

What is the effect of introducing **CASE Environments** within organizations with a **low Capability Maturity Level**?

CASE Environments require a **Process**, thus it requires a **Capability Maturity Level** of at least **2**.



You have just purchased Antedeluvian Software Developers, an organization on the verge of bankruptcy because the company is at **CMM Maturity Level 1**. What is the first step you will take to restore the organization to profitability?

Provide **Software Engineering** education for all employees, including managers. This is a necessary prerequisite for advancement to **Maturity Level 2**.

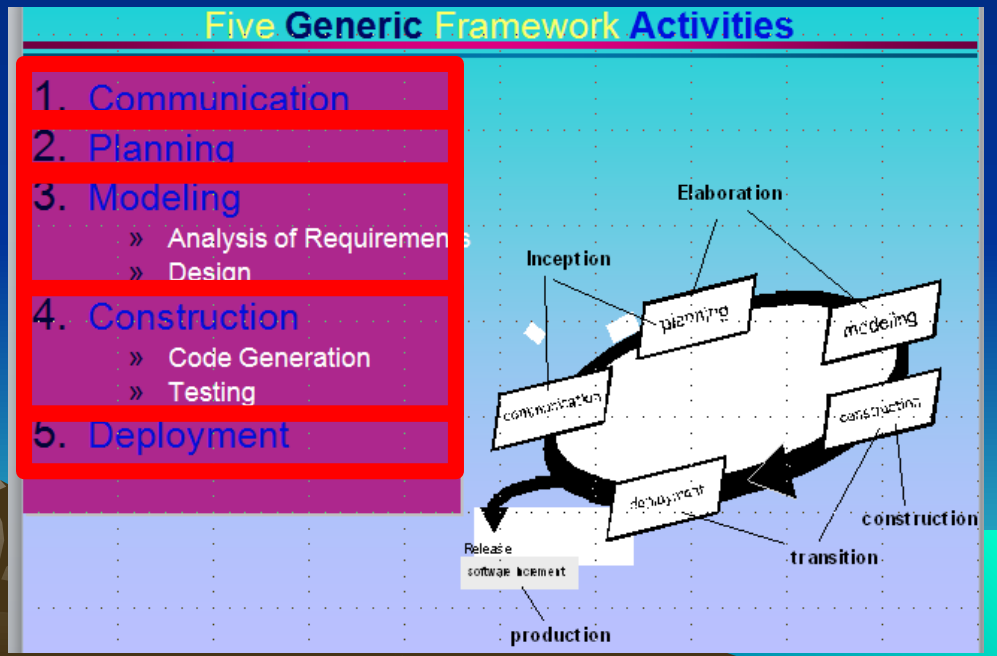
Level 2 – Repeatable (~ 15%)

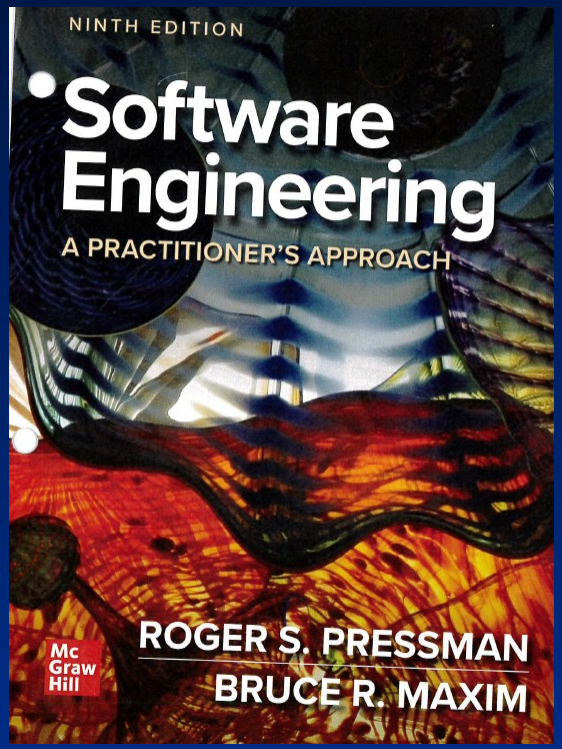
- software configuration management - **SVN**
- Software Quality Assurance - **SQA**
- SOFTWARE PROJECT tracking and oversight - **MS PROJECT**
- SOFTWARE PROJECT Planning- **SPMP**
- Requirements management - **SRS**

What would you say?

SDLC Model

A **Framework** that **describes the Activities** performed at each stage of a **software development project / project**.





24.1.3 The Process

A software process (Chapters 2 through 4) provides the framework from which a comprehensive plan for software development can be established. A small number of framework activities are applicable to all software projects, regardless of their size or complexity. Even agile developers follow a change-friendly process (Chapter 3) to impose some discipline on their software engineering work. A number of task sets—tasks, milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

Framework Activities

- work tasks
- work products
- milestones & deliverables
- SQA checkpoints

Umbrella Activities

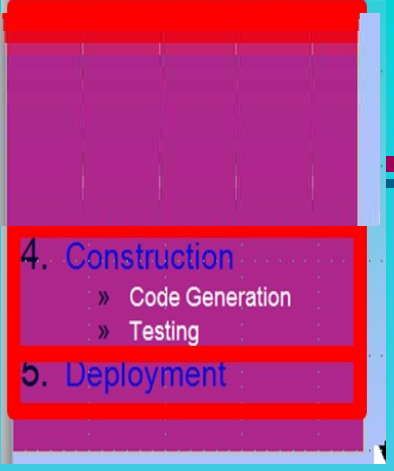
Prescriptive Process Models
SDLC

- **Prescriptive Process Models** advocate an orderly approach to **Software Engineering**

That leads to a few questions …

- If **Prescriptive Process Models strive for structure and order**, are they **inappropriate** for a **software** world that **thrives on change**?

- Yet, **if we reject traditional Process Models (and the structure and order they imply) and replace them with something less structured,** do we make it **impossible** to achieve **coordination** and **coherence** in **software** work?

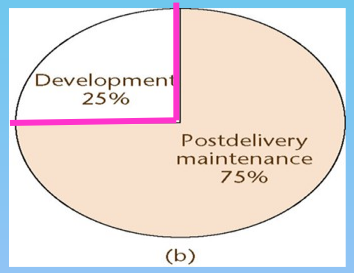


Code-and-Fix **Model**

- The **easiest** way to develop **software**
- The **most** expensive way

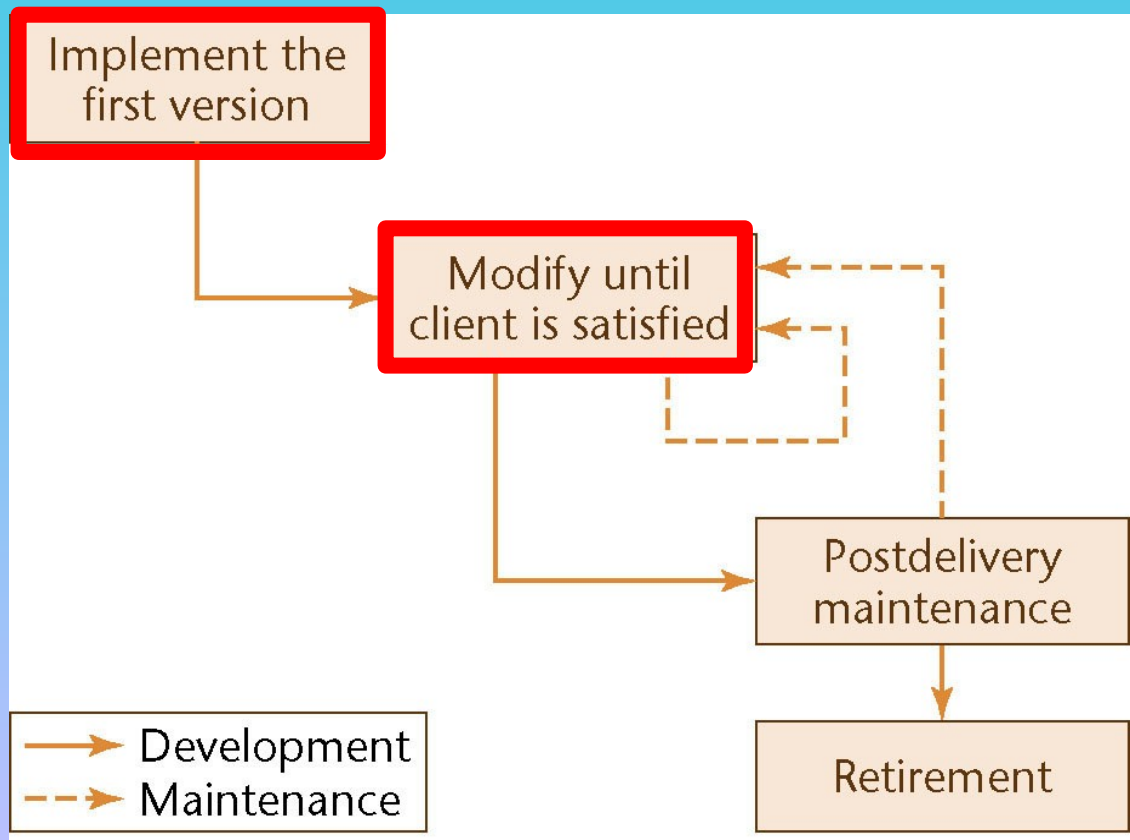
No **Specifications/ Analysis**

- **Maintenance** nightmare



No **Design**

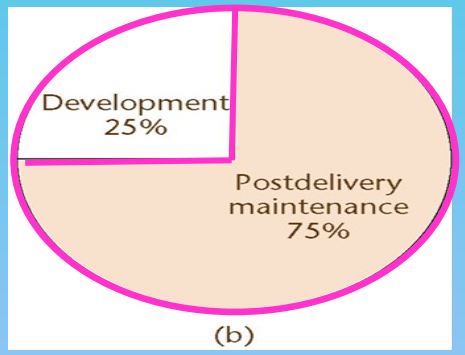
- **Maintenance** nightmare

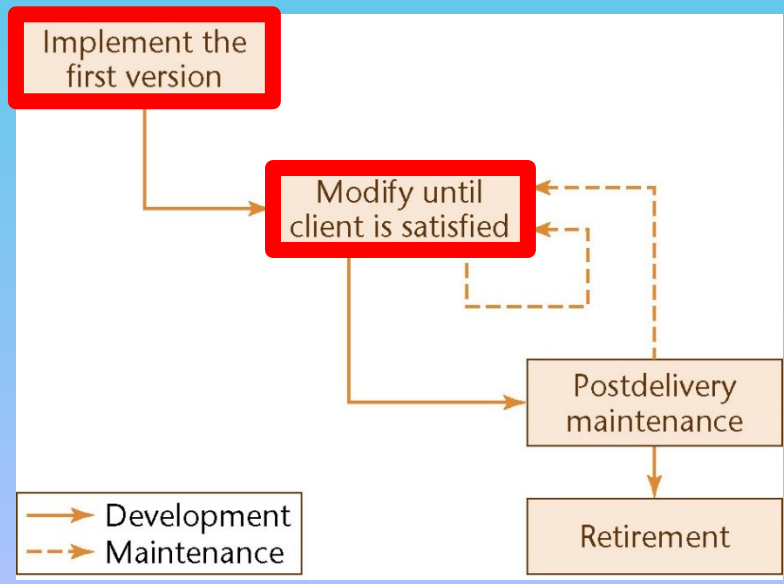


Describe a risk inherent in using the Code-and-Fix Life-Cycle Model.

There is a risk that development will be exceedingly expensive.

Also, maintenance of a software product can be extremely difficult without Specification or Design documents, and the chances of a regression fault occurring are considerably greater, thereby increasing the risk of not being able to maintain the system at all.





IBM OS/360

- grew from 1 MLoC to 8 MIOC in 3 years
- Induced 2 defects for any 1 removed

What would you say?

Prescriptive Process Models

- Linear Models
- Iterative Models
- Evolutionary Models
- Specialized Models
- Adaptive Models
- .....
- .....
- .....

Prescriptive Process Models

Linear Models

Iterative Models

Evolutionary Models

Specialized Models

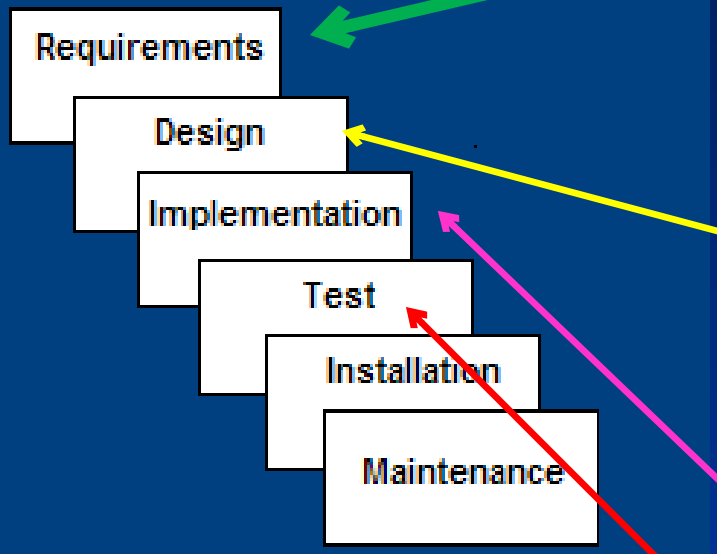
Adaptive Models

. . . . . .

. . . . . .

. . . . . .

Linear Models: The Waterfall



- **Requirements (& Specification) Analysis WHAT** – defines needed information, function, behavior, performance and interfaces.
- **Design HOW** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation
- **Testing**

Waterfall Strengths

Easy to understand, **Easy** to use

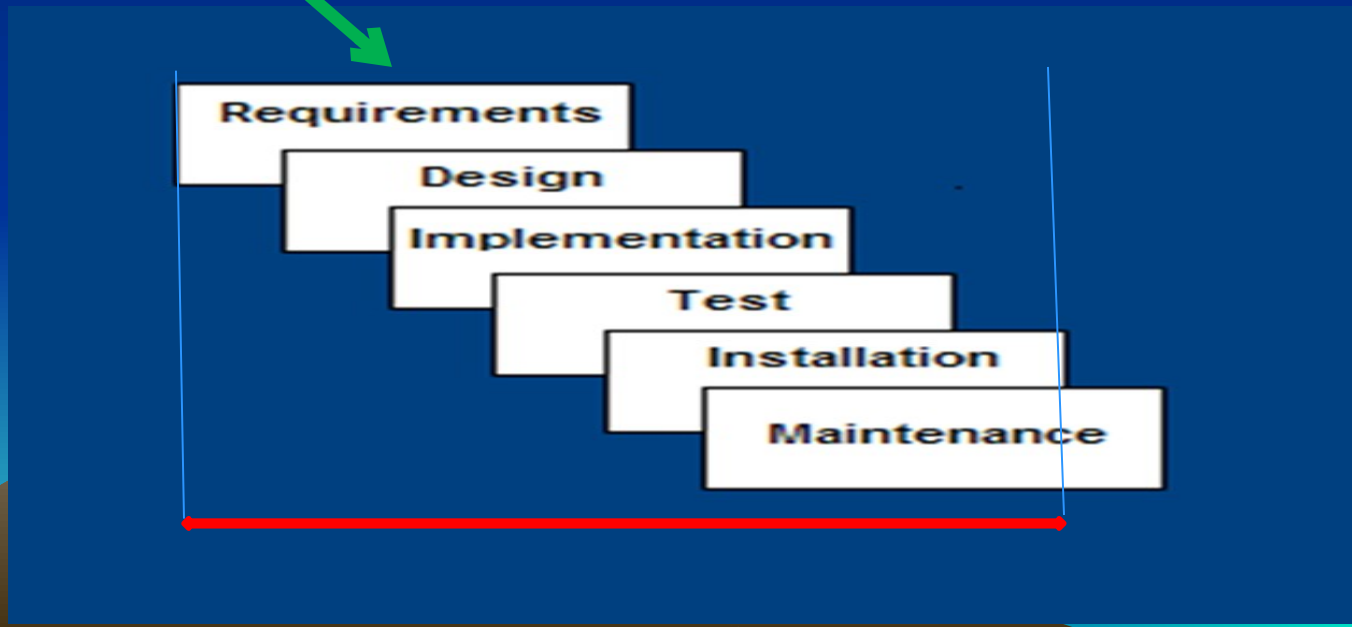
Provides **structure** to inexperienced staff

Milestones are well understood

Sets **Requirements** **stability**

Good for **management control** (plan, staff, track)

Good when **quality** is more important than **cost** or **schedule**



Waterfall Deficiencies

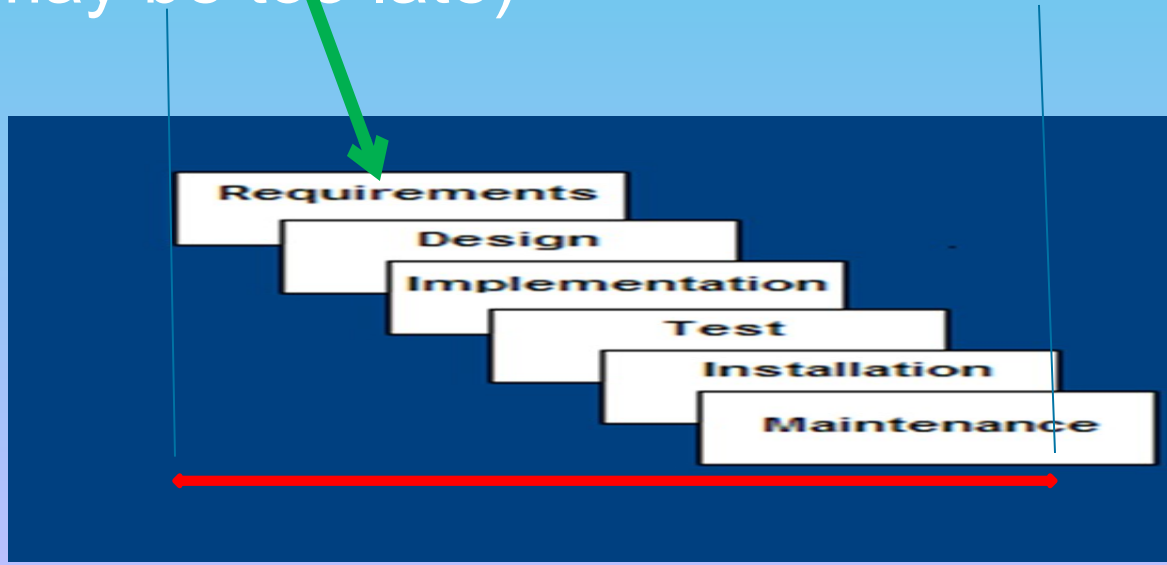
All **Requirements** must be known **upfront**

Deliverables created for each **phase** are considered **frozen** – **inhibits flexibility**

Does not reflect problem-solving nature of software development – **iterations of phases**

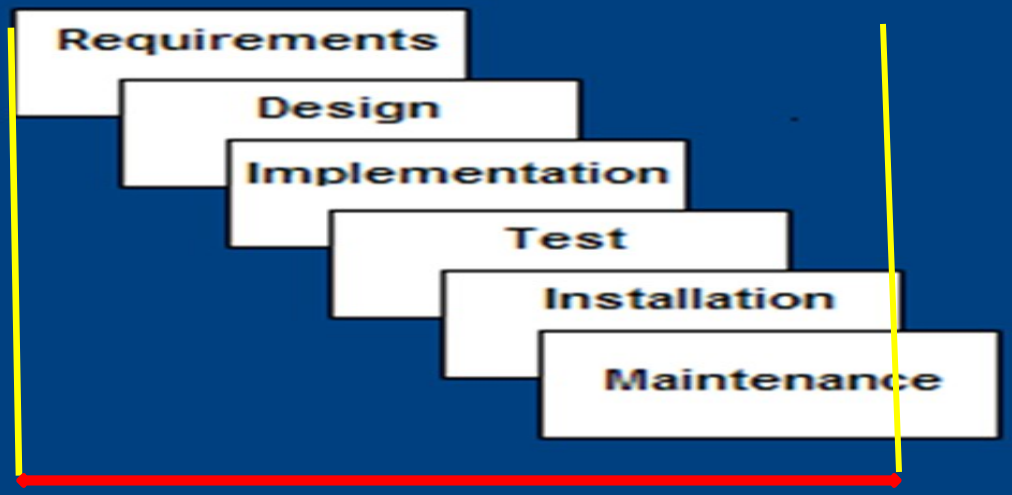
Integration is one **big bang** at the end

Little opportunity for **Customer** to preview the **system** (until it may be too late)



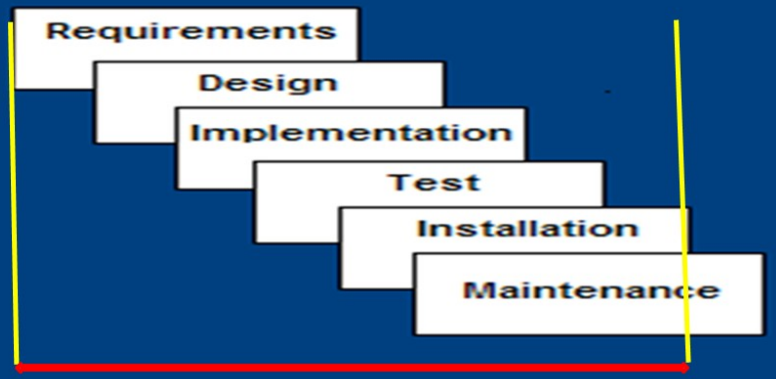
When to use the Waterfall Model

- **Requirements** are very **well known**
- **Product** definition is **stable**
- Technology is **understood**



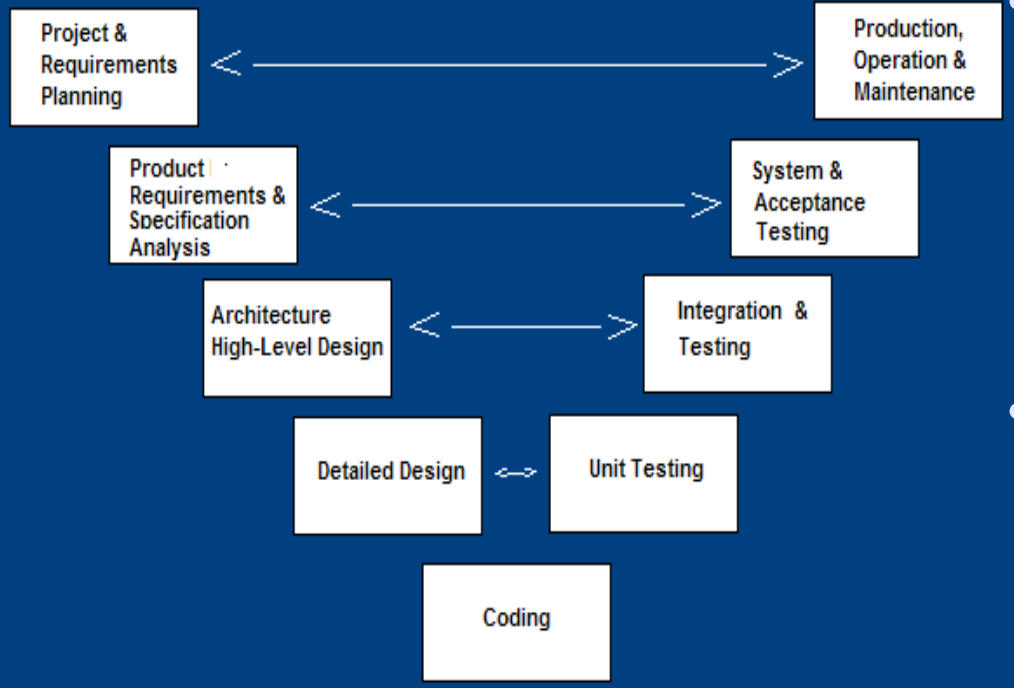
Describe a risk inherent in using the **Waterfall Software Development Life-Cycle Model**.

There is a risk that **by the time (long) the product is delivered** the **Client's needs might have changed** (**software** automates the **Domain** that is constantly evolving).



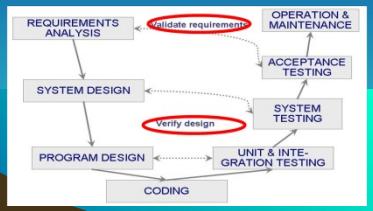
What would you say?

Linear Models: V-Shaped Model



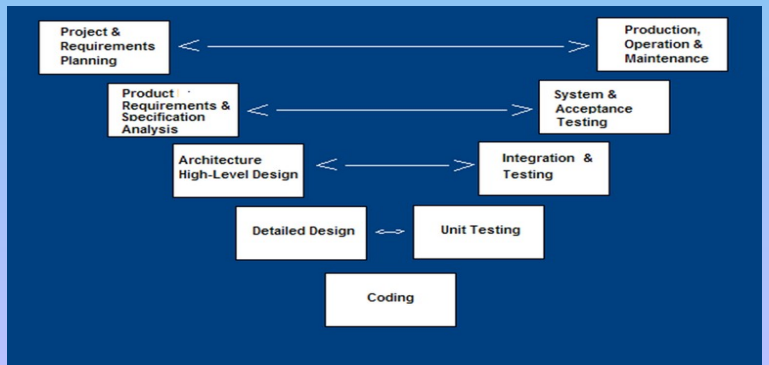
- A variant of the **Waterfall** that emphasizes the **verification** and **validation** of the **product**.
- **Testing** of the **product** is **planned in parallel** with a corresponding phase of **development**

- **Verification** – did I built the WRIGHT **SYSTEM**?
  - **Testing** at the **end of each Workflow**
- **Validation** - is the **SYSTEM** WRIGHT?
  - **Testing** at the **end of the PROJECT** (far too late)

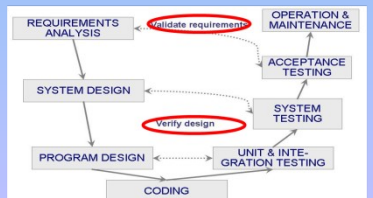


V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system **WHAT**
- Architecture or High-Level Design – defines how software functions fulfill the design **HOW**
- Detailed Design – develop algorithms for each architectural component **HOW – "blue print"**
- Production, operation and maintenance – provide for enhancement and corrections
- System and **Acceptance Testing** – check the entire software system in its environment
- Integration **Testing** – check that modules interconnect correctly
- Unit **Testing** – check that each module acts as expected
- Coding – transform algorithms into **software**



- **Verification** – did I built the WRIGHT SYSTEM?
  - **Testing** at the **end of each Workflow**
- **Validation** - is the SYSTEM WRIGHT?
  - **Testing** at the **end of the PROJECT** (far too late)



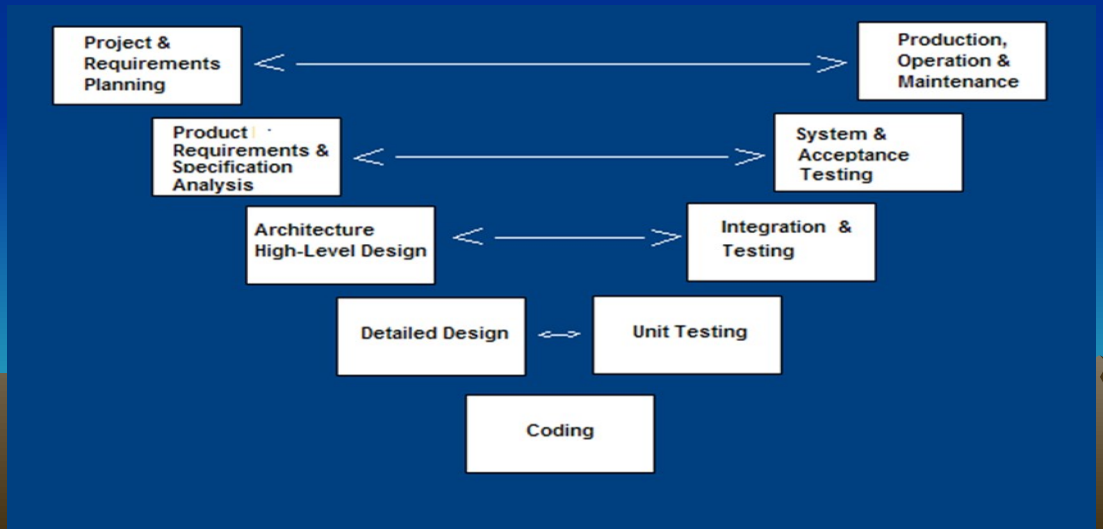
V-Shaped Strengths

Emphasize planning for verification and validation of the **product** in early stages of **product** development

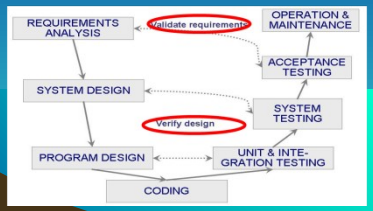
Each deliverable must be Testable

Project management can track progress by milestones

Easy to use



- Verification – did I built the WRIGHT SYSTEM?
  - Testing at the **end of each** Workflow
- Validation - is the SYSTEM WRIGHT?
  - Testing at the **end of the** PROJECT (far too late)



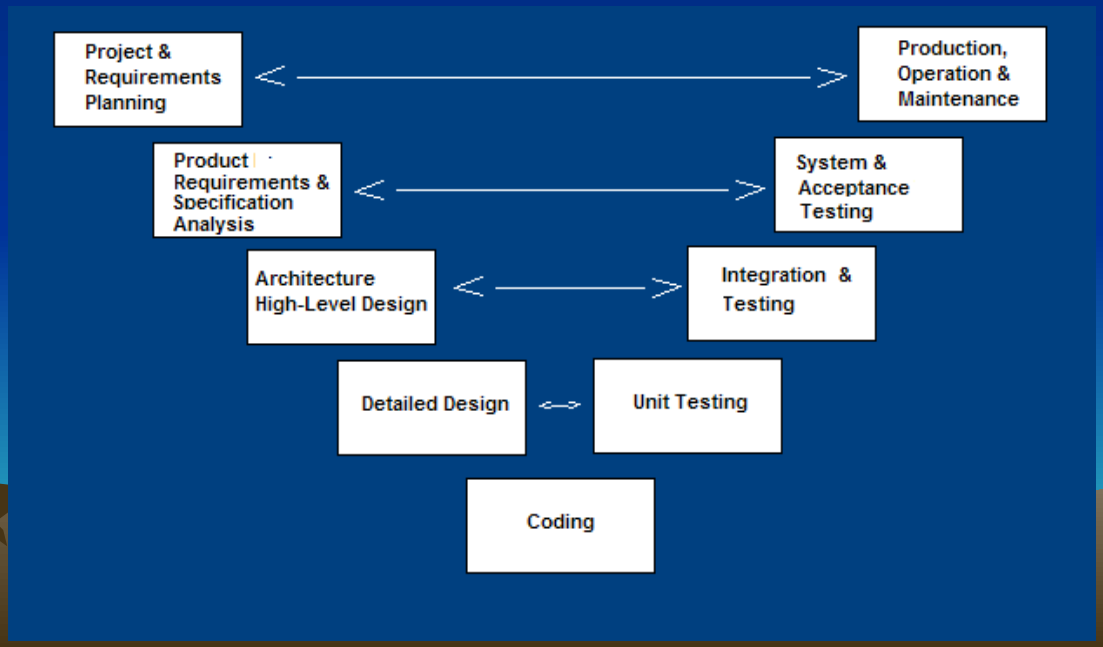
V-Shaped Weaknesses

Does not easily handle concurrent events

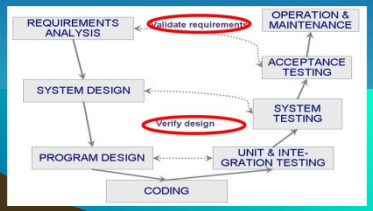
Does not handle iterations

Does not easily handle dynamic changes in Requirements

Does not contain risk analysis activities



- Verification – did I built the WRIGHT SYSTEM?
  - Testing at the **end of each Workflow**
- Validation - is the SYSTEM WRIGHT?
  - Testing at the **end of the PROJECT** (far too late)

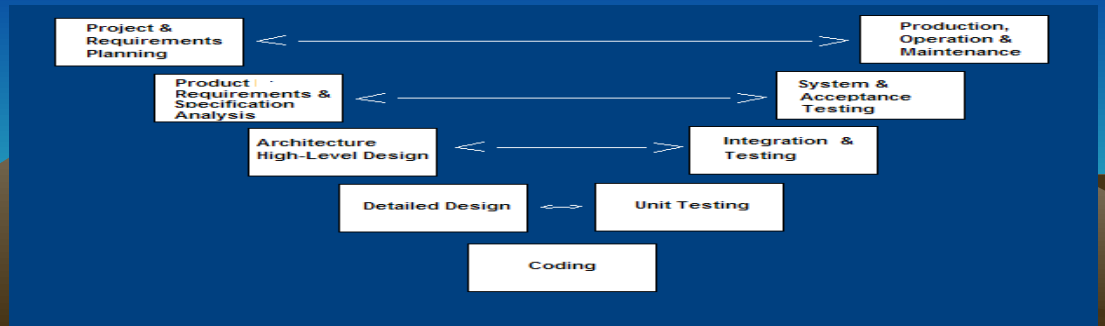


When to use the V-Shaped **Model**

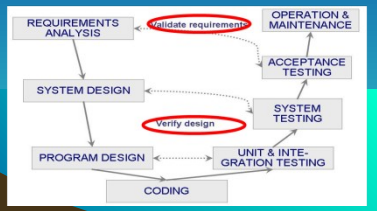
Excellent choice for systems requiring high **reliability** – hospital patient control applications

All **Requirements** are known up-front

Solution and technology are known



- Verification – did I built the WRIGHT SYSTEM?
  - Testing at the **end of each** Workflow
- Validation - is the SYSTEM WRIGHT?
  - Testing at the **end of the** PROJECT (far too late)



Two-Dimensional **Model**?

- The **Waterfall Model**
- One-Dimensional

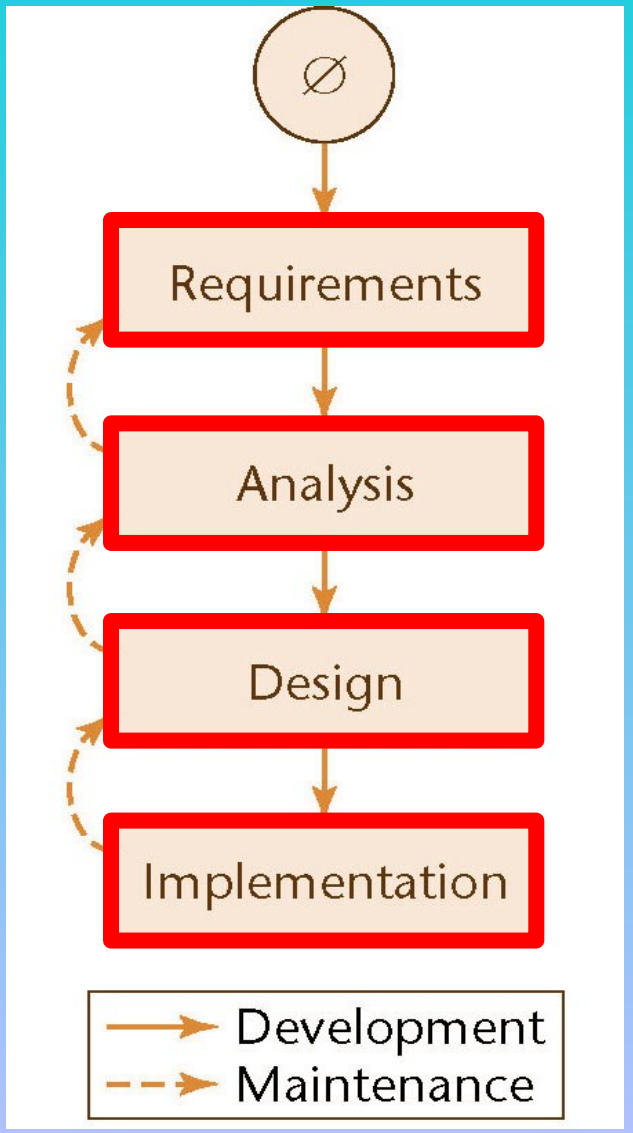
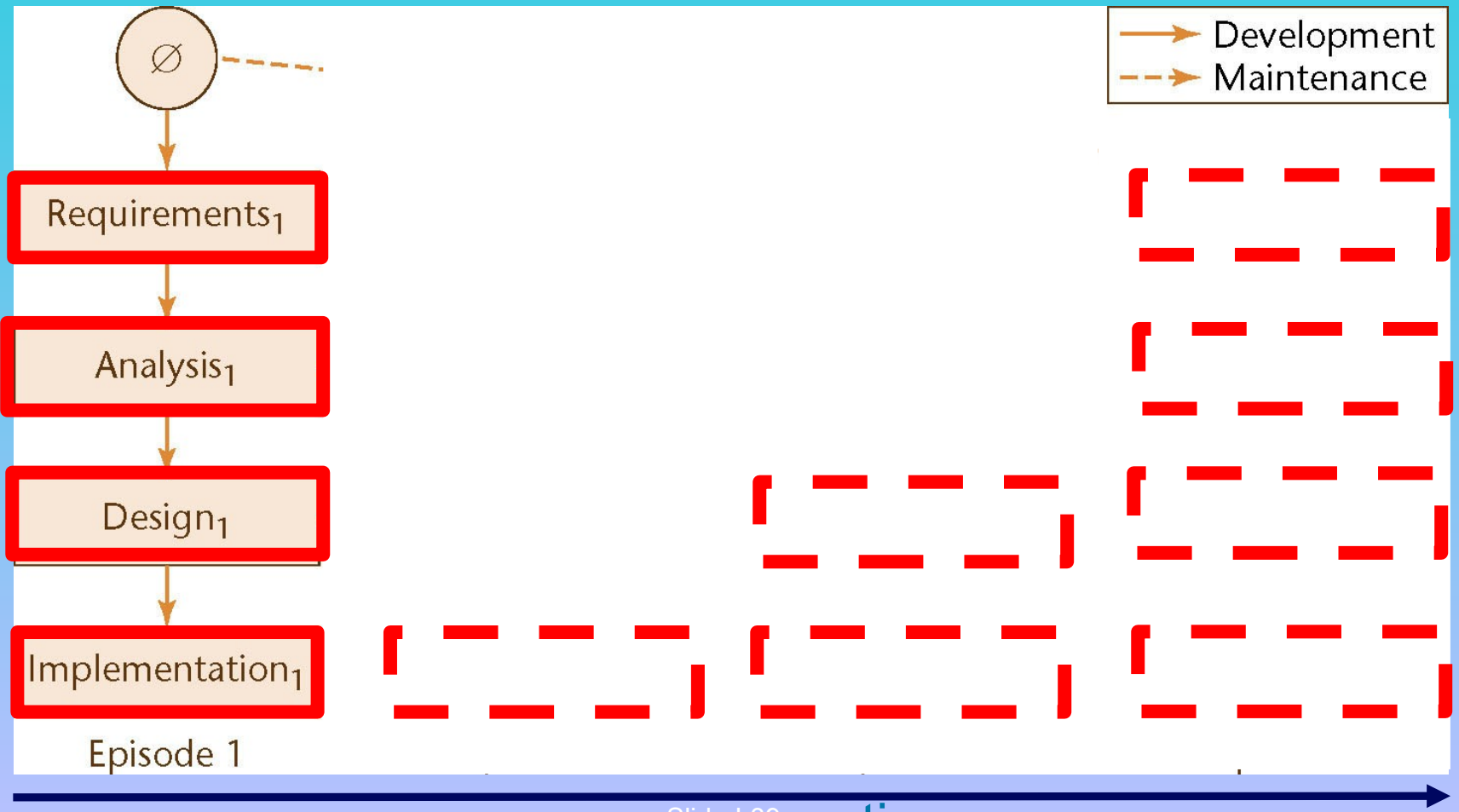


Figure 2.3 (again)

Why a Two-Dimensional **Model**?

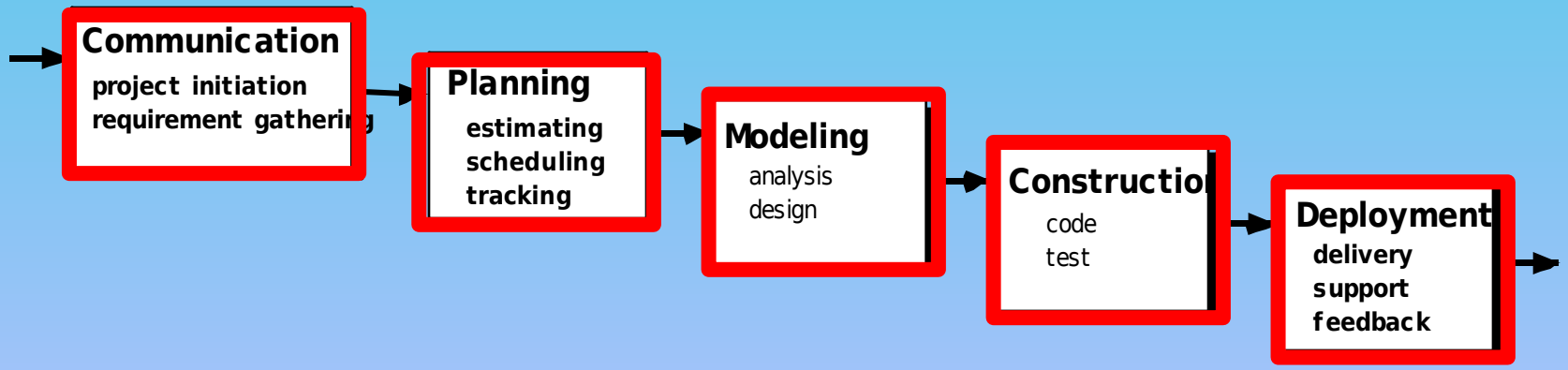
- **Evolution tree Model**
- **Two-Dimensional**



Why a Two-Dimensional **Model**?

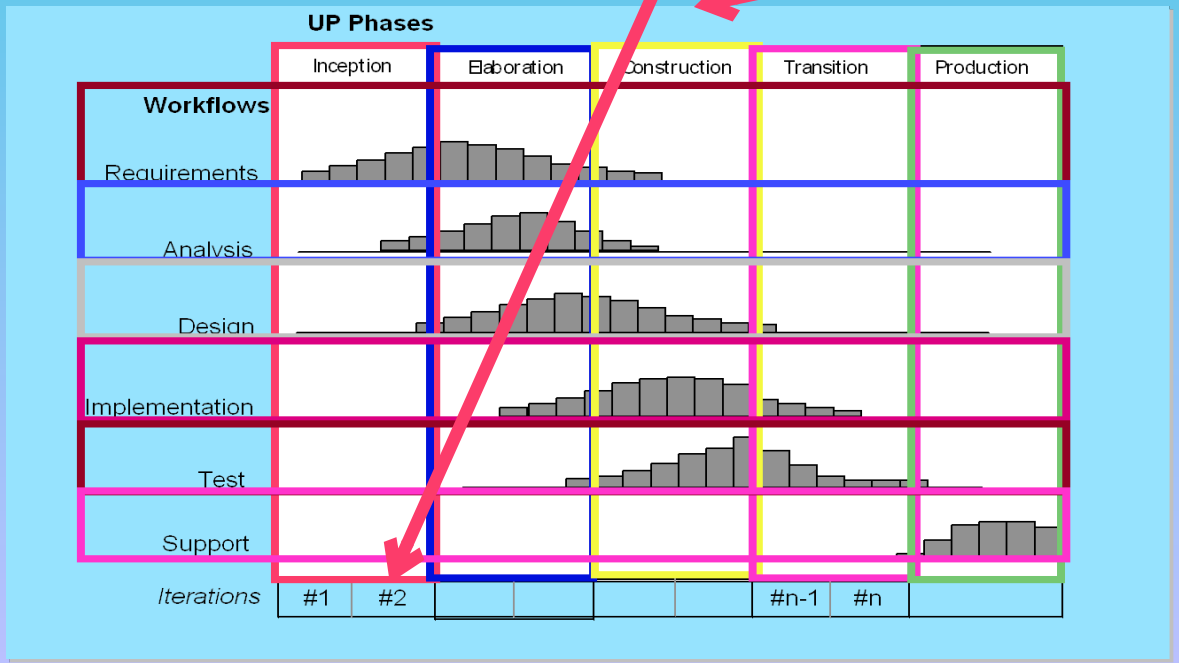
Are all the additional complications of the **two-Dimensional Model necessary**?

In **an ideal world**, each workflow would be completed before the next workflow is started



Why a Two-Dimensional **Model**?

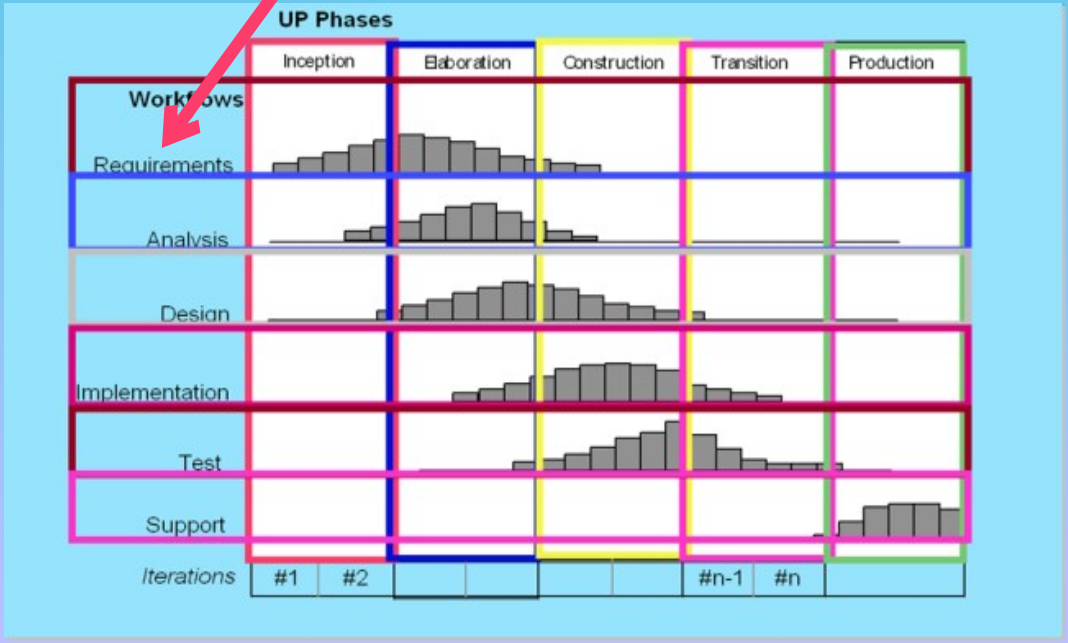
- **In reality**, the development **task is too big** for this
- As a consequence of **Miller's Law** (**5+/-2 – stepwise refinement**)
  - The development task has to be divided into **Increments**
  - Within each **Increment**, **Iteration** is performed until the task is complete

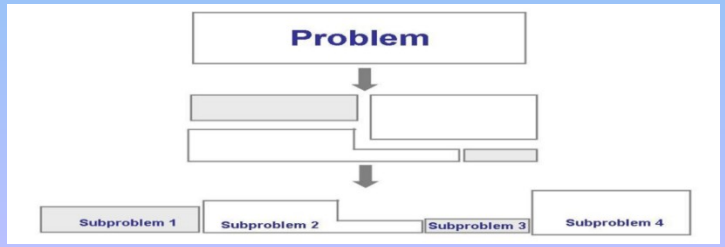


Why a Two-Dimensional **Model**?

At the beginning of the process, there is not enough information about the software product to carry out the Requirements workflow

- Similarly for the **other core workflows**





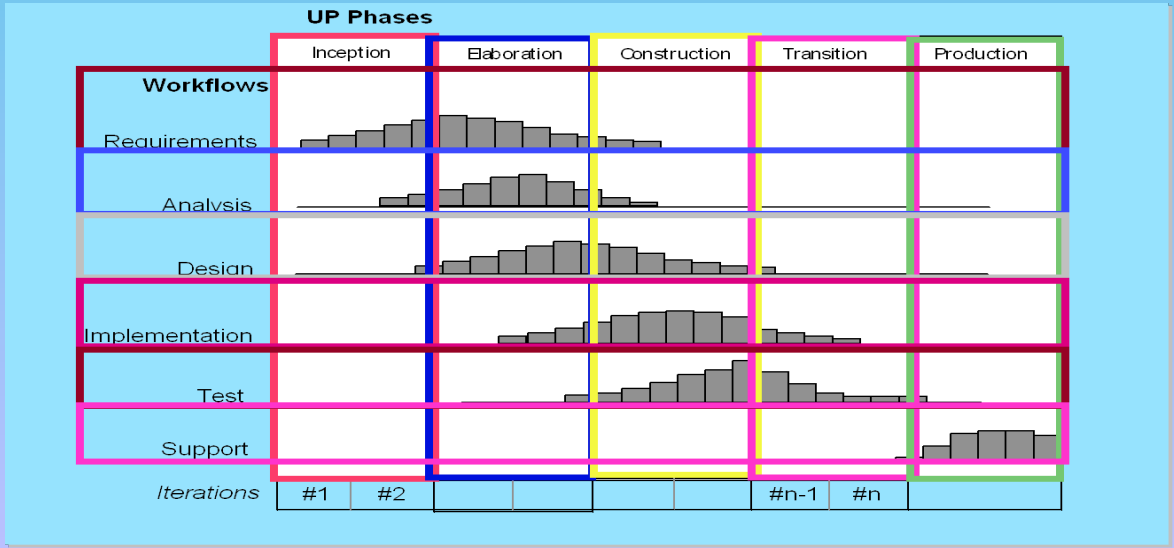
Why a Two-Dimensional **Model**?

Handles the inevitable changes well

- The **moving target problem**
- The **inevitable mistakes**

It is the best solution:

- It provides a Framework for Incrementation and Iteration



Prescriptive Process Models

Linear Models

Iterative Models

Evolutionary Models

Specialized Models

Adaptive Models

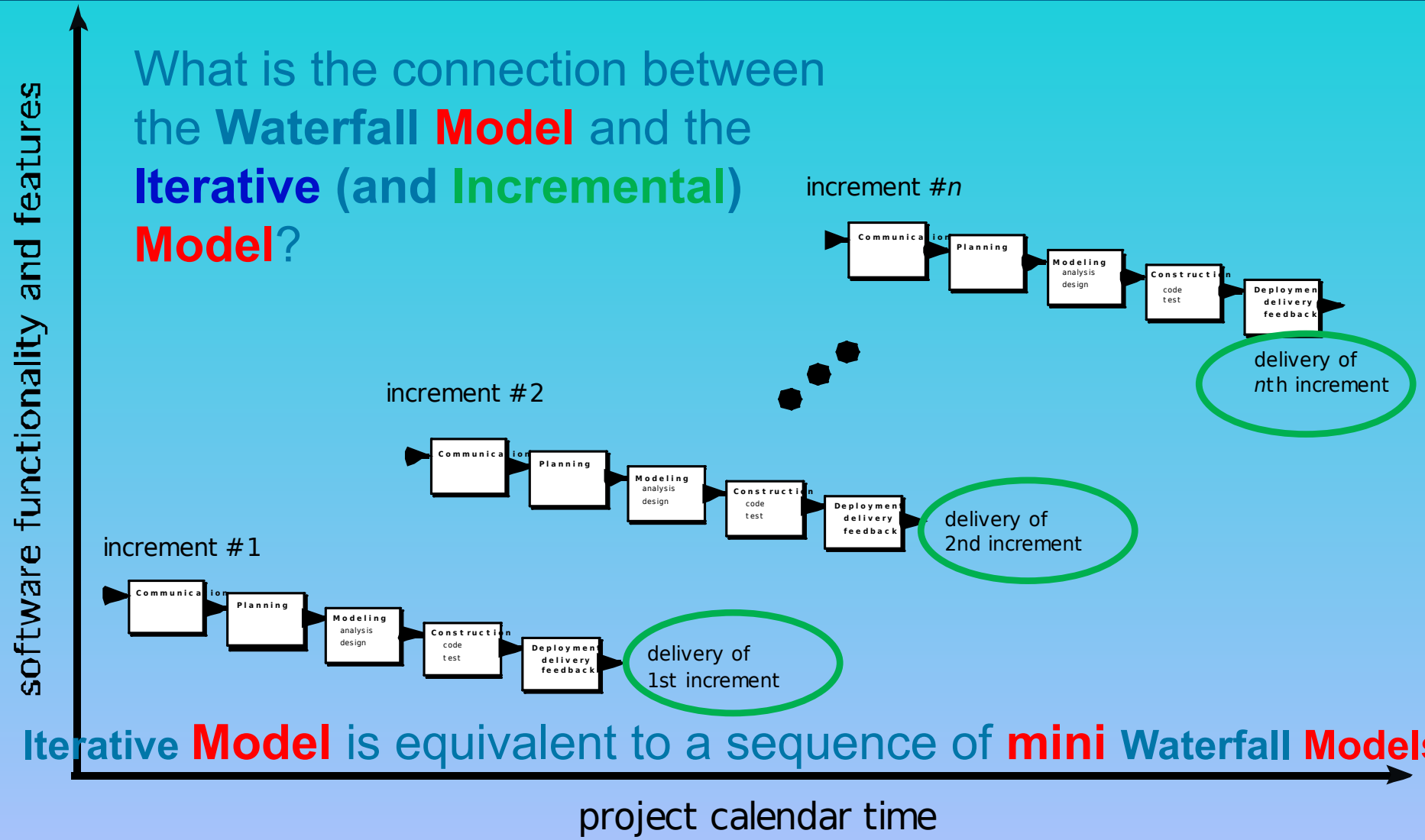
. . . . . .

. . . . . .

. . . . . .

Iterative Model: The Incremental Model

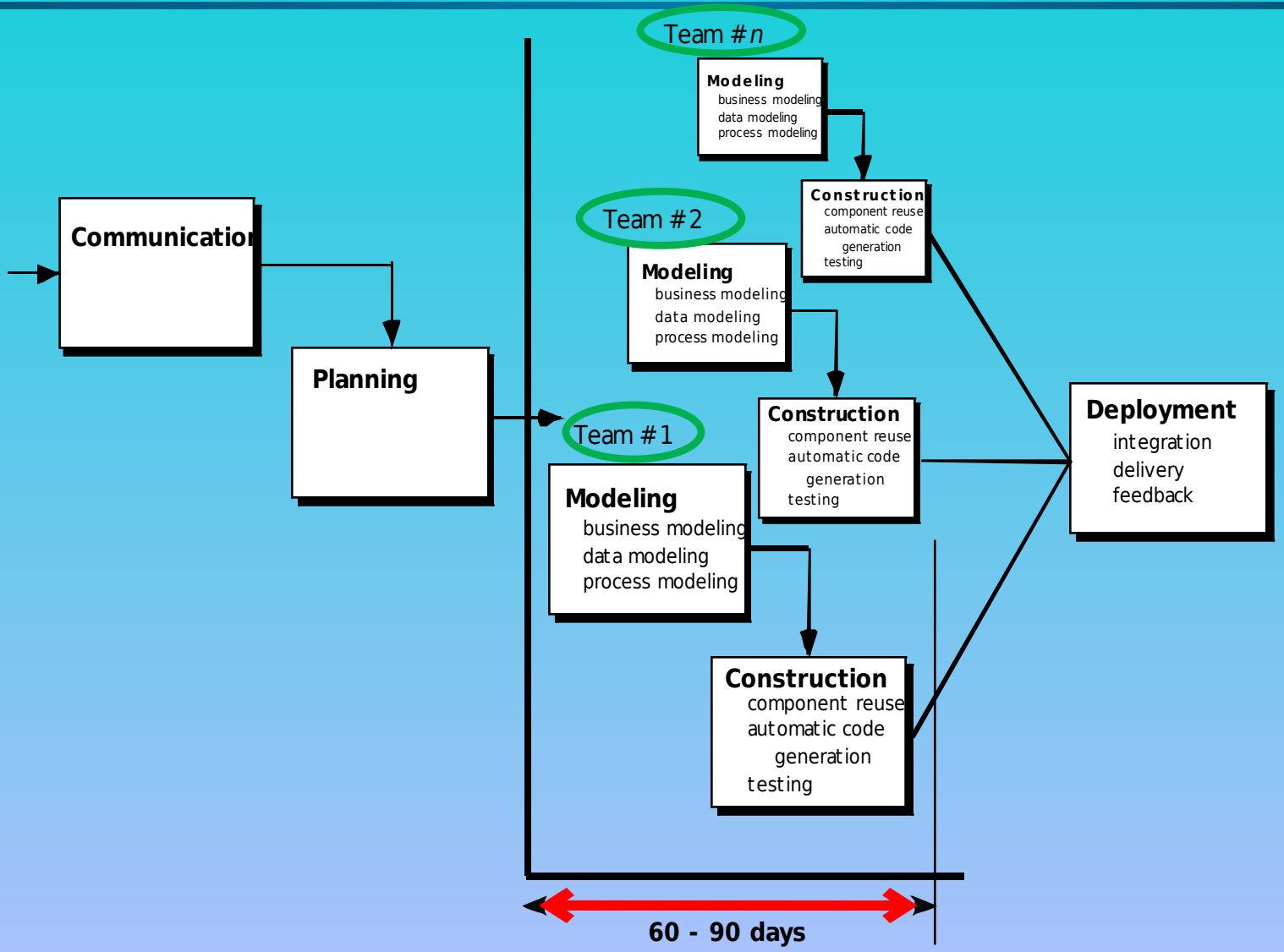
What is the connection between the **Waterfall Model** and the **Iterative (and Incremental) Model**?



Iterative Model is equivalent to a sequence of **mini Waterfall Models**

Delivers **software** in small but usable **pieces**, each **piece** builds on **pieces already delivered**

Iterative Model: The RAD (Rapid Application Development) Model



Heavy use of **Reusable** software components; extremely short development cycle

Iterative Model: The RUP (Rational Unified Process) Model

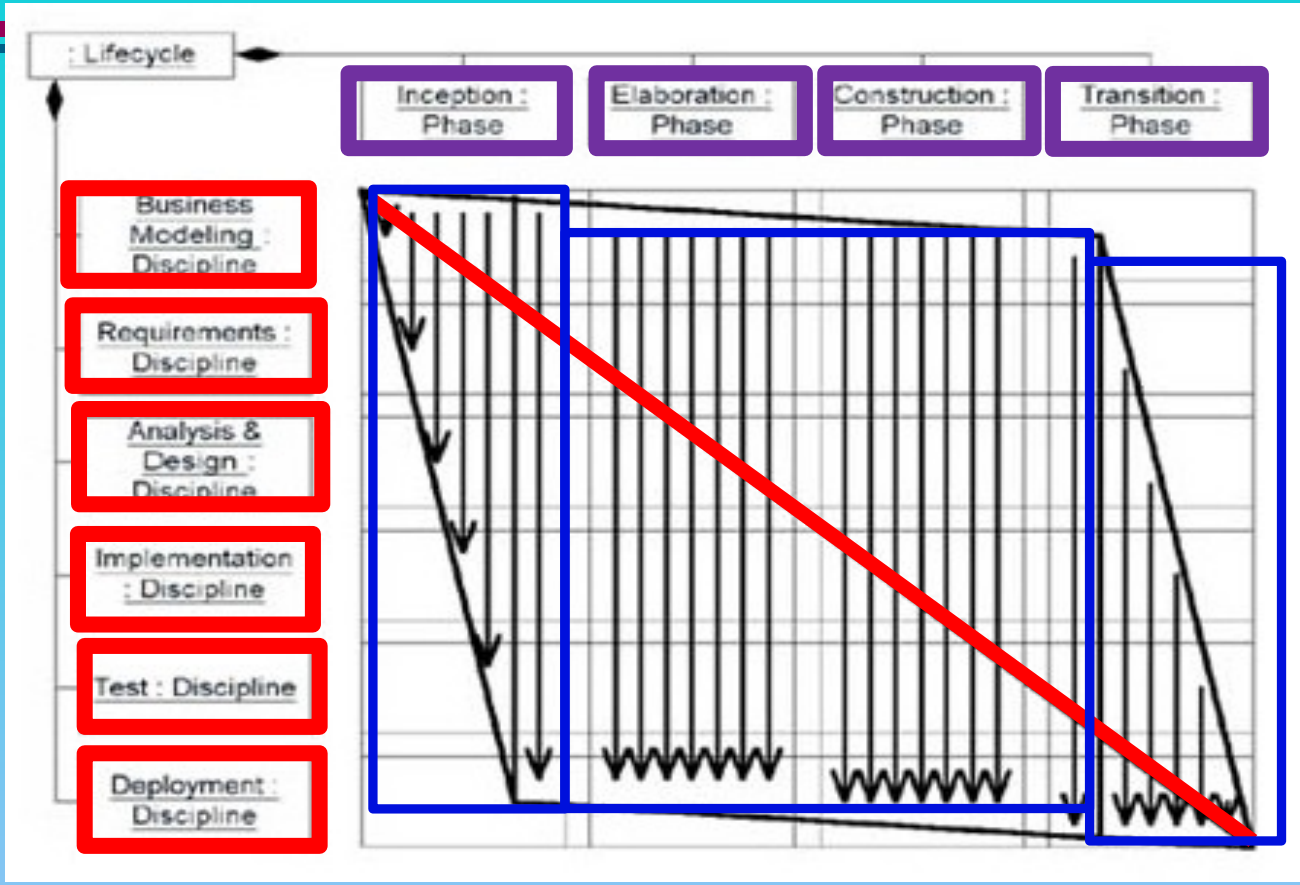
Iterative Waterfall

Iterative development offers significant advantages over traditional, **Linear** development because it has proven to be **impossible to understand all risks, costs, implementation details, and future User preferences in advance** and in detail through big design up front .

It is also well known that there is a point of diminishing returns in **estimating** whereby the **accuracy of estimates** falls off sharply when one puts "too much" effort into refining an estimate in advance. **Our understanding of these details must be improved over time** through **Incremental delivery** of subsets of a **solution**.

This is really the only practical way to build a high-quality solution.

The RUP (Rational Unified Process) Model



Generally, an **effort ramps up** at the start of a development cycle, reaches an **optimum** where all core disciplines are being performed in parallel and all supporting disciplines are operating as appropriate, and then **ramps down** at the end of the development cycle. As **Iterations** execute, their content involves **collaborations among roles**, **activities**, and **artifacts** where activities are related via a **producer-consumer relationship** and may overlap in time such that a **consumer** activity may start as soon as its inputs from **producer** activities are sufficiently mature.

Prescriptive Process Models

Linear Models

Iterative Models

Evolutionary Models

Specialized Models

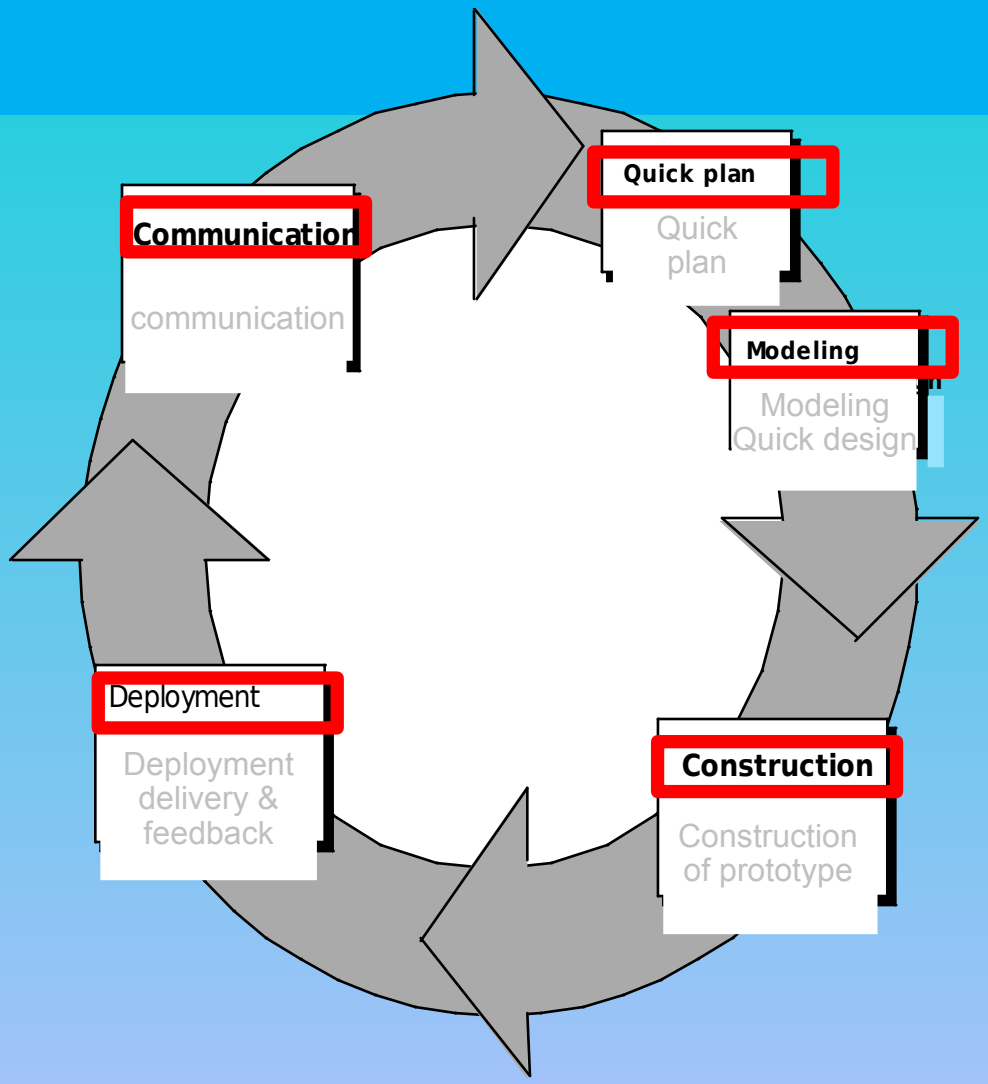
Adaptive Models

. . . . . .

. . . . . .

. . . . . .

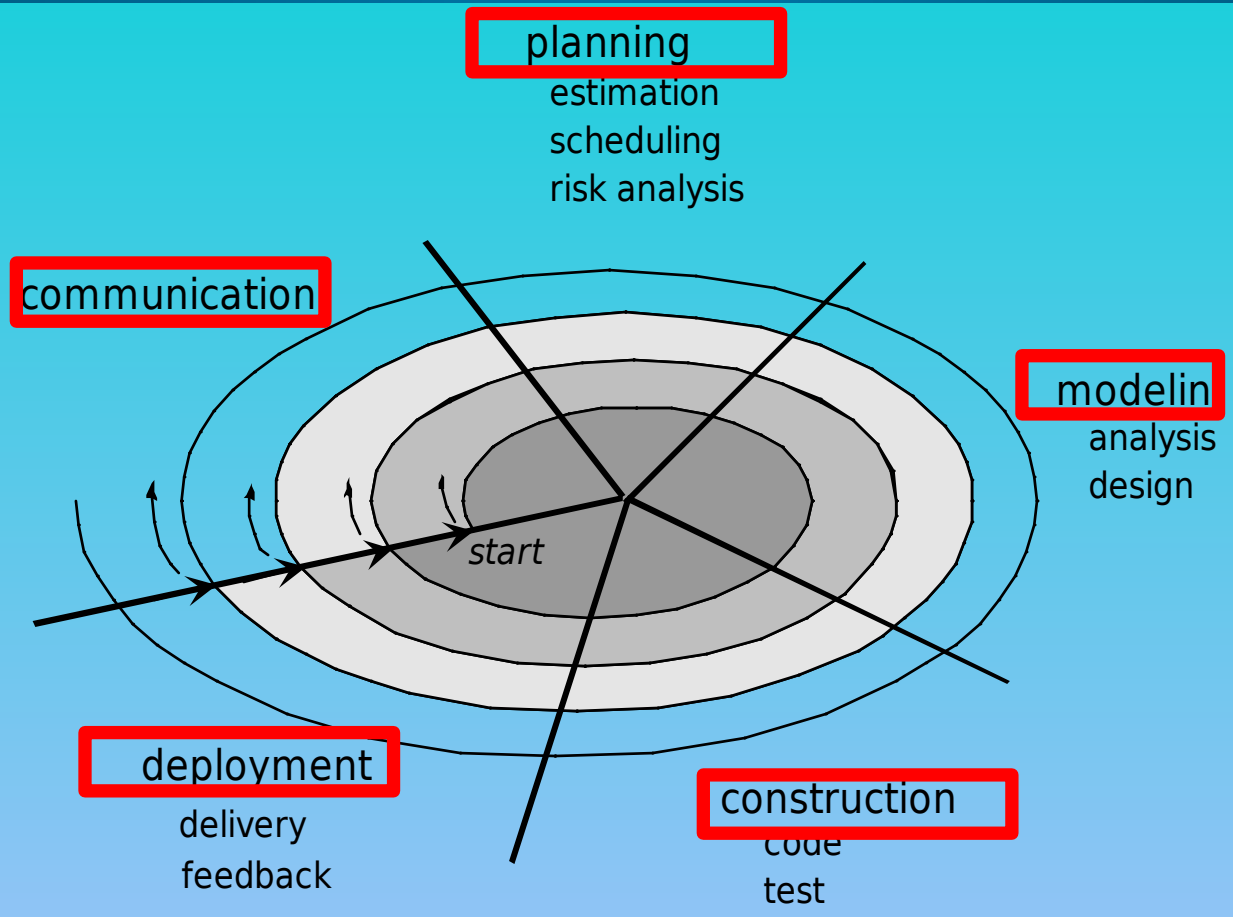
Evolutionary Models: Prototyping Model



Good first step when **Customer** is clueless about the details,

Developer needs to resist pressure to extend prototype into a production product

Evolutionary Models: The Spiral Model

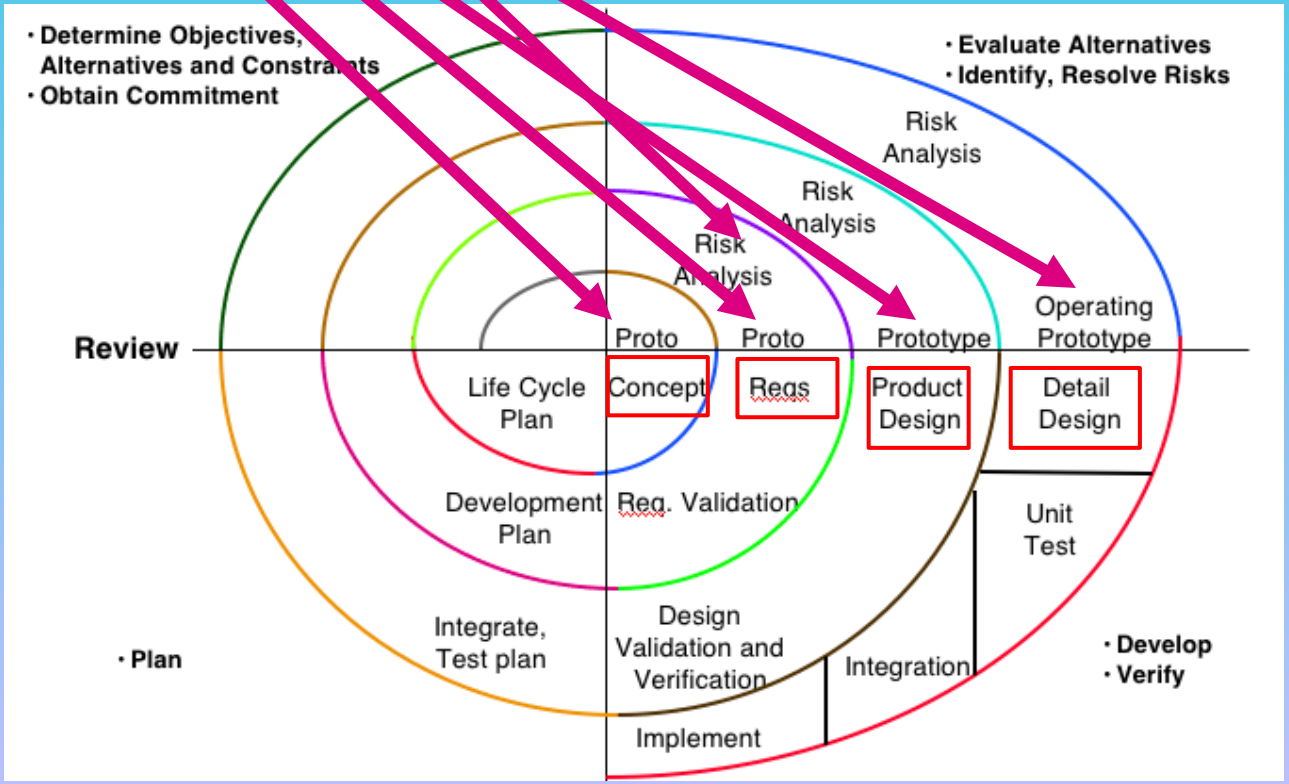


http://sunset.usc.edu/publications/TECHRPTS/1998/usccse98-512/usccse98-512.pdf

Couples **Iterative** nature of prototyping with the **controlled** and **systematic** aspects of the **Linear Sequential Model**

The Spiral **Model**

- Important features
  - Risk Analysis is explicitly shown in the **Model** at each stage
  - Prototyping and Iterative development "fit" in this **Model**
  - Repetition of activities clearly in **Model**
    - Suggests that alternatives can be explored

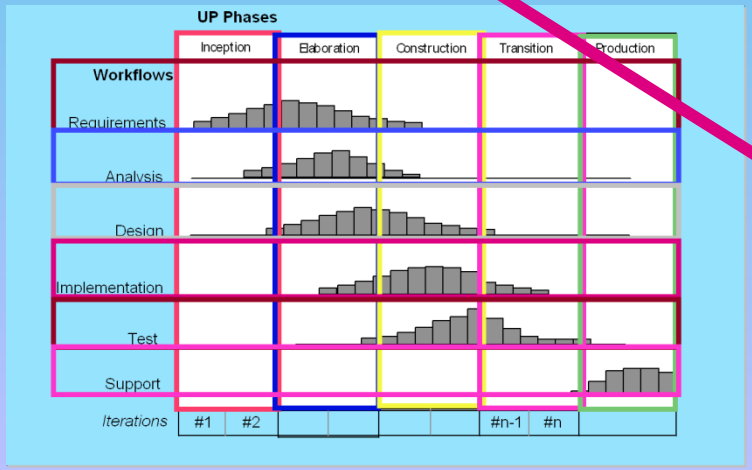


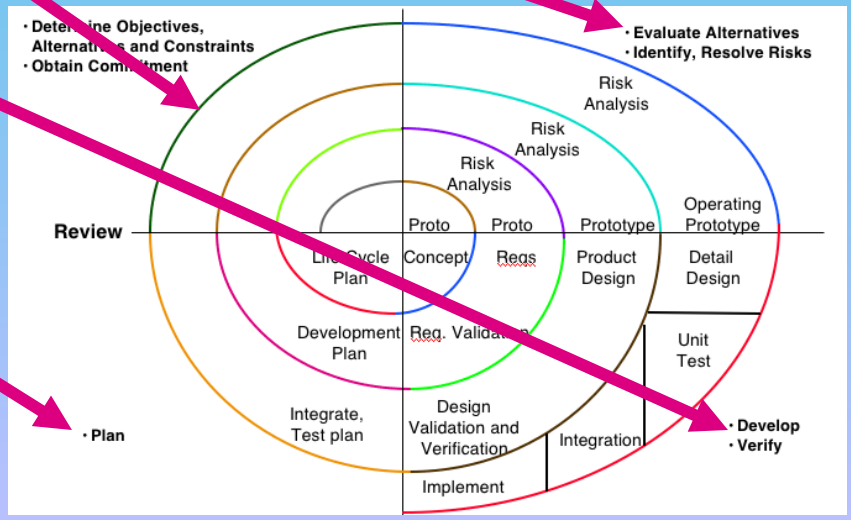
Features of the Spiral Process Model

Each cycle around the spiral can be like a Phase

Each cycle has four stages

1. Determine Objectives, Constraints
2. Identify and manage Risks. Explore Alternatives
3. Develop and Verify next stage or level of the **Product**
   - » Depending on the spiral, "**Product**" might be a **Requirements** document, a **High-Level Design**, Code, etc.
4. Review results and Plan for next stage
   - » May include getting **Client/Customer** feedback





Is the Spiral **Model** the Answer?

For some situations, yes. (There is no **one** answer.)

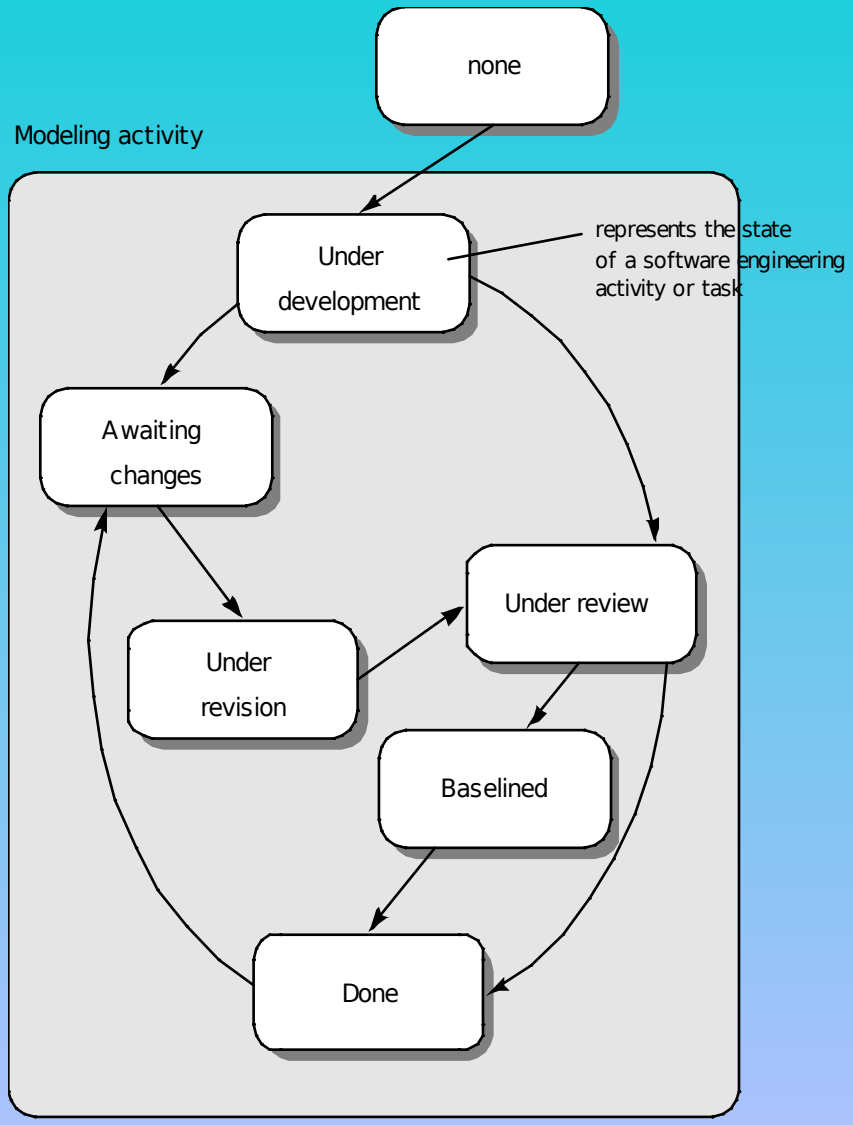
Original study that analyzed the **Spiral Model** says:

- Intended for **internal** development of large systems
  - "**internal**": **Developers** and **Clients** in same organization
- Intended for large-scale systems where cost of not doing risk management is high

But, the **Spiral Model** is important

- Historically
- And as an illustration of many desired features (**Iteration**) of a **software** development **process (SDLC)**

Evolutionary Models: Concurrent Model



Similar to Spiral Model often used in development of client/server Applications

Prescriptive Process Models

Linear Models
Iterative Models
Evolutionary Models
Specialized Models
Adaptive Models

. . . . . .

. . . . . .

. . . . . .

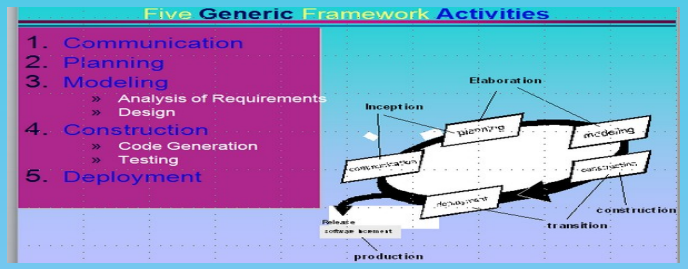
Still Others: Specialized Process Models

AOSD (Aspect Oriented Software Development)—provides a **Process** and **methodological approach** for defining, specifying/analyzing, designing, and constructing *aspects*

Specialized Process Models: Unified Process (UP) Model

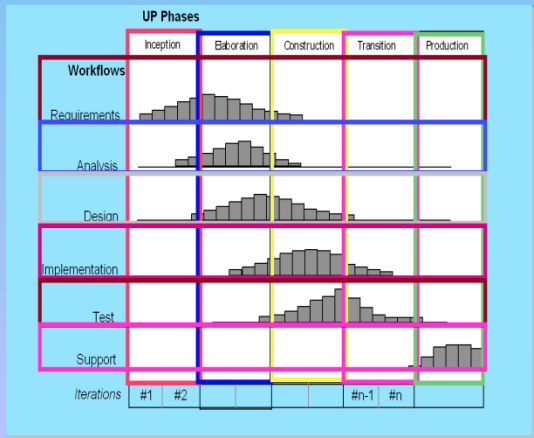
Use-Case driven, architecture centric, **Iterative**, and **Incremental** **software** Process

Attempts to draw on best features of traditional **software** **Process** Models and implements many features of **Agile** **software** development



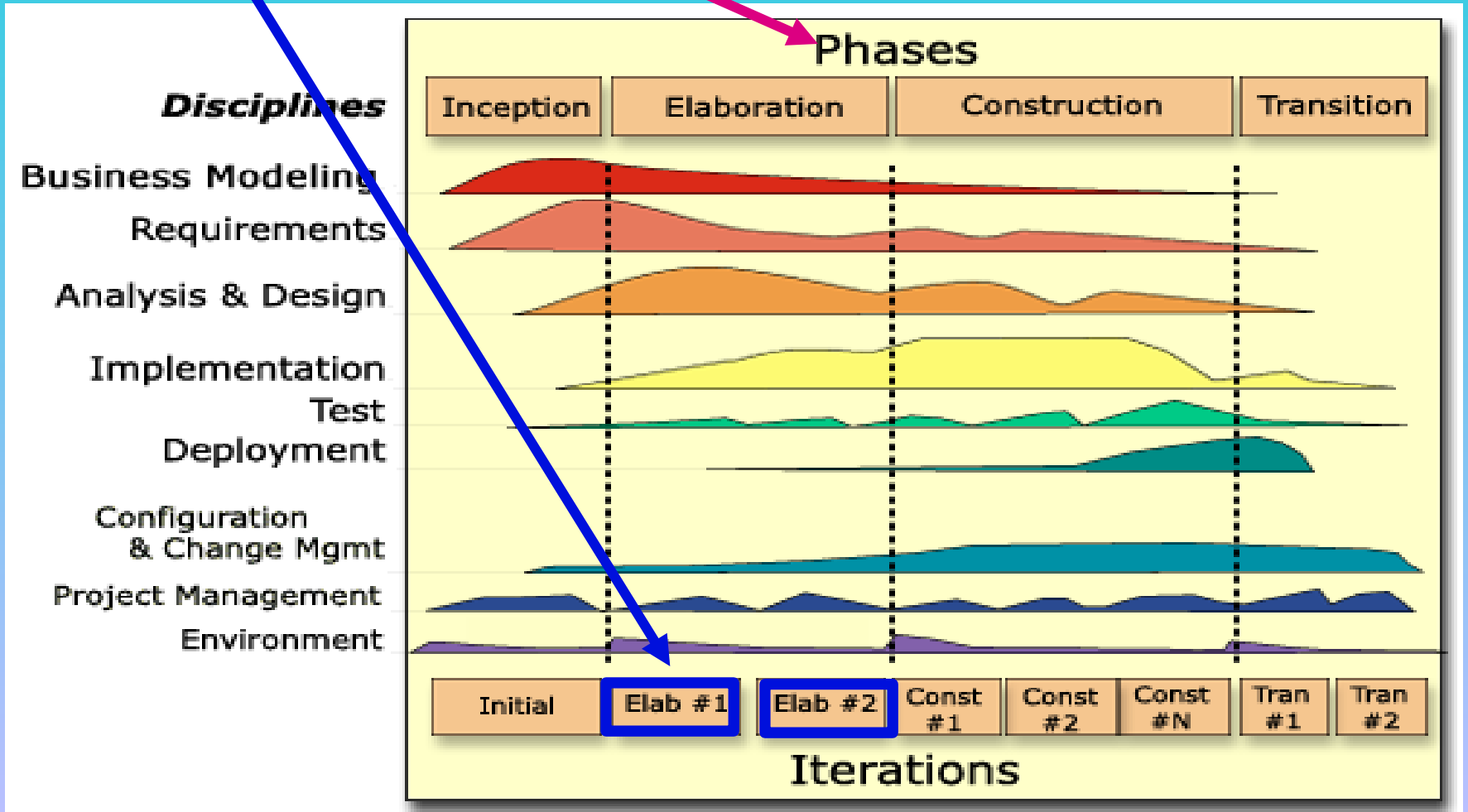
Phases

- Inception phase (**Customer** communication and planning)
- Elaboration phase (communication and modeling)
- Construction phase
- Transition phase (**Customer** delivery and feedback)
- Production phase (**software** monitoring and support)



The Phases of the Unified Process

- The Increments are identified as Phases
- Iterations for each Phase



Unified Process (UP) Work Products

Inception Phase

- Vision document
- Initial **Use-Case Model**
- Initial project glossary
- Initial business case
- Initial risk assessment
- **Project plan** (phases and iterations)
- Business model
- Prototypes

Elaboration Phase

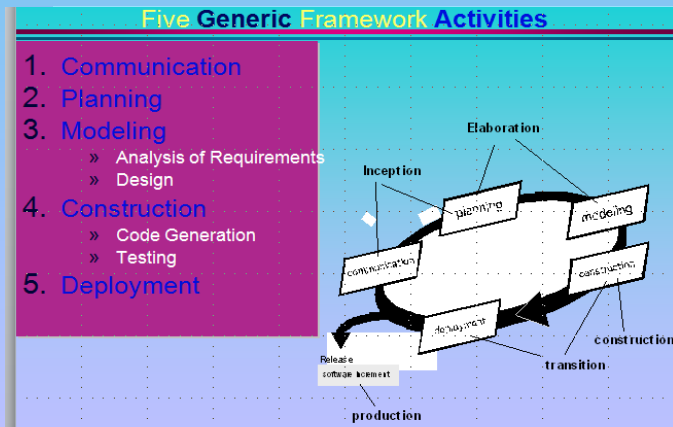
- **Use-Case Model**
- Functional and non-functional **Requirements**
- **Analysis OO Class Model**
- Software architecture description
- Executable architectural prototype
- Preliminary design model
- Revise risk list
- Project plan (iteration plan, workflow, milestones)
- **Preliminary user manual**

Construction Phase

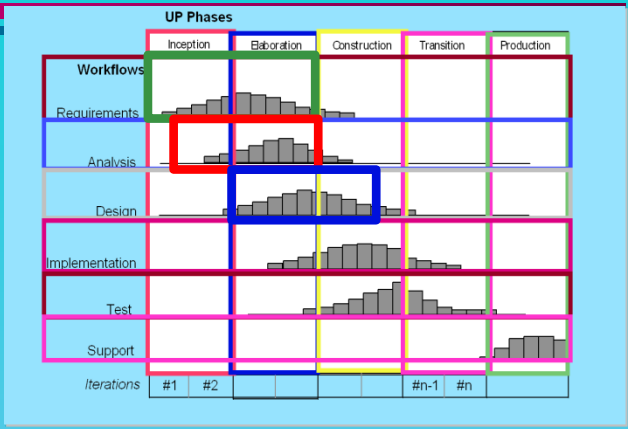
- **Design OOD Model – "blue print"**
- **Software components**
- Integrated software increment
- Test plan
- **Test cases**
- Support documentation (user, installation, increment)

Transition Phase

- Delivered **software increment - "house"**
- **Beta test reports**
- User feedback



Unified Process (UP) Workflows



Requirements Workflow

- Goal is to answer 4 questions
  - » What are the relevant characteristics of **Customer** domain?
  - » What will the **software product** accomplish?
  - » What features and functions will be used to accomplish it?
  - » What constraints will be placed on the **system**?
- **Requirements** are determined **Iteratively** during Inception and Elaboration Phases
- **Software team** must learn enough to establish **project** scope, set an **Iterative** and **Incremental** **project** plan, and develop **Use Cases recognizable to End-users**

Analysis Workflow WHAT

- Begins during Inception phase and culminates in the Elaboration phase
- Goal is to perform architectural analysis and produce the work products required by the analysis model
- The **Analysis OO Model** uses abstractions that are recognizable to the **Customer** or **End-user**

Design Workflow HOW – "blue print"

- Begins in last part of Elaboration phase and continues to first stages of Construction phase
- Objective of Design is to transform Analysis model (the **WHAT**) into a **Design OO Model** (the **HOW** "**blue print**")
- Modeling moves from the problem (**Customer**) domain into solution (**engineer**) domain

Unified Process (UP) Workflows

Implementation Workflow

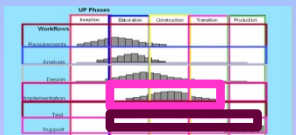
- Work begins in Elaboration phase and dominates Construction phase
- Incorporates the software engineering tasks required to translate design Classes "**blue print**" into **executable software** components "**house**", **testing** and **integrating** these components, and deliver those required for the planned **Increment**
- Build/**Reuse** decisions made during this phase

Testing Workflow

- During Elaboration the intent of Testing Workflow is to exercise the executable architecture and demonstrate its **viability**
- During Construction testing workflow focuses on **testing software components** as they are integrated in the build and **testing integrated software** before releasing to **Customer**
- During Transition testing the Customers or End-users assume the responsibility for **Acceptance Tests**

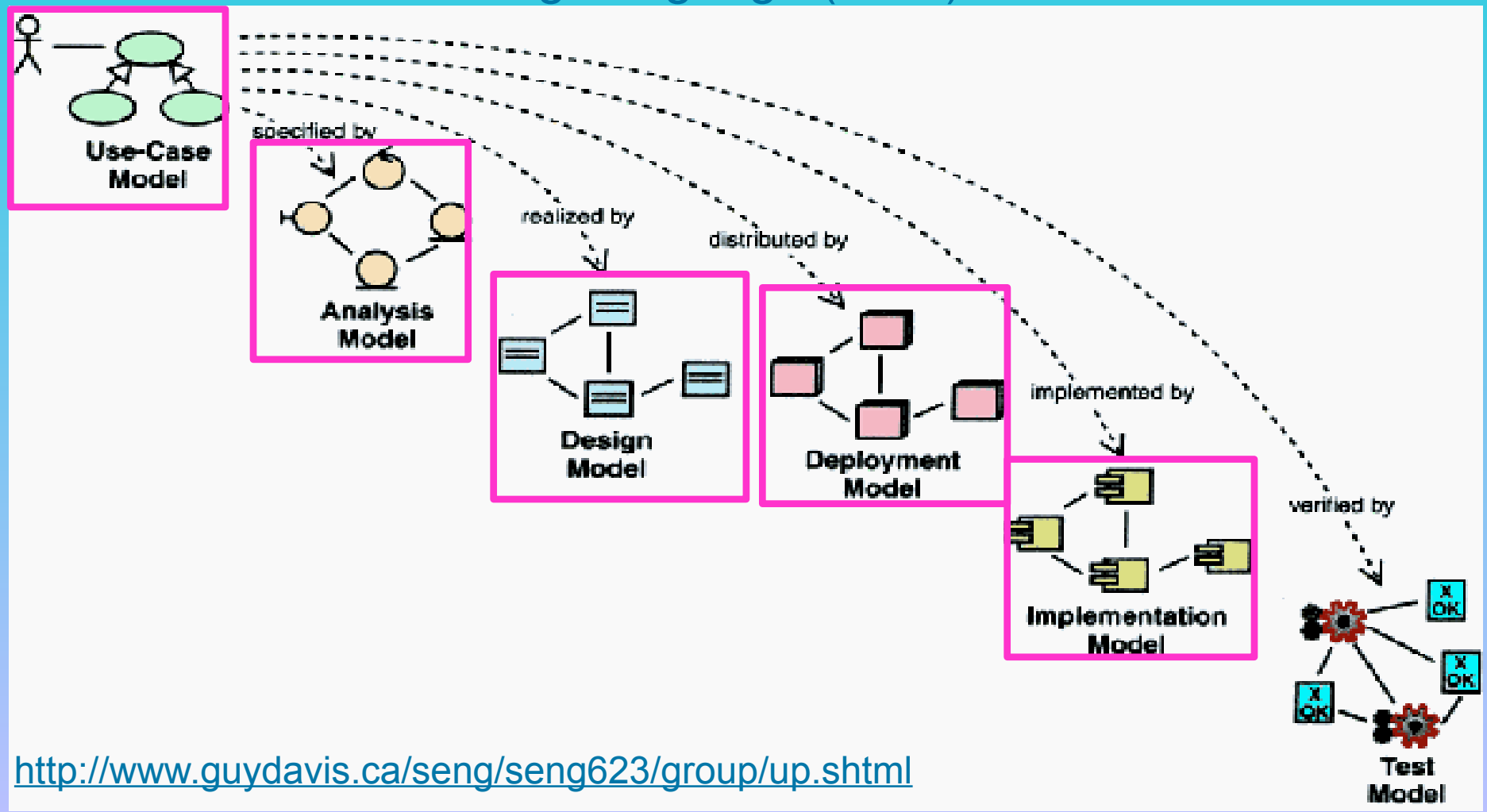
Project Management Workflow

- Spans all software **Increments**
- Focus is on planning, risk management, project tracking and control
- **Phase** plan provides a rough estimate of the effort required to accomplish workflow across each **UP** phase, the major milestones for each phase and **Increment**, and the number of increments required
- **Iteration** plan defines the tasks associated with analysis, design, implementation, testing, and deployment for one software **Increment**



Unified Process

Unified Process—a "Use-Case driven, architecture-centric, **Iterative** and **Incremental**" **software** Process closely aligned with the Unified Modeling Language (UML)



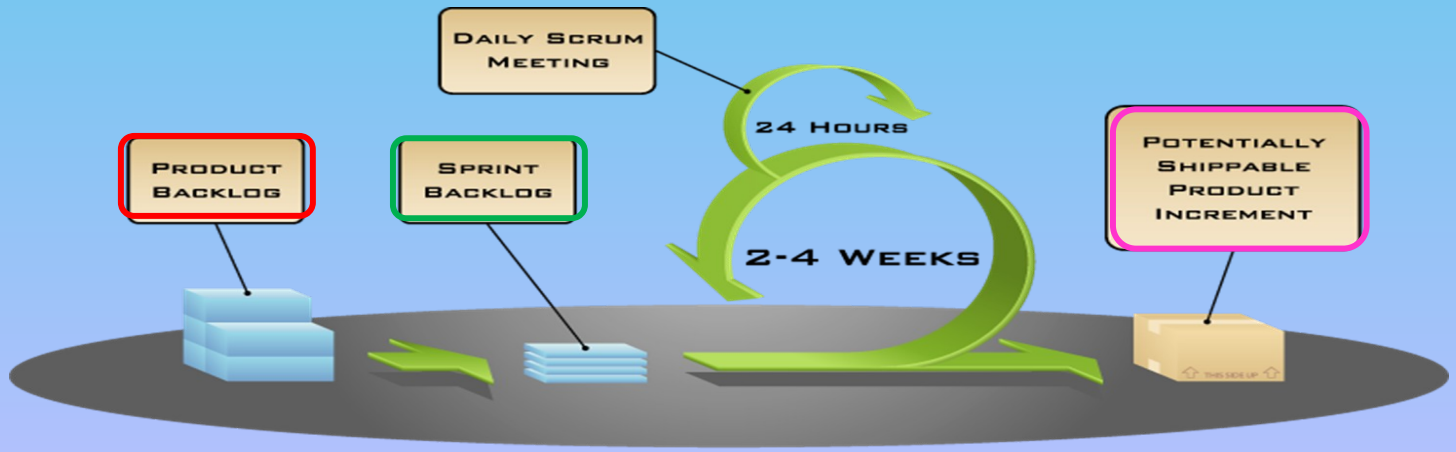
http://www.guydavis.ca/seng/seng623/group/up.shtml

Specialized Process Models: Scrum Model

Originally proposed by Schwaber and Beedle

Scrum—distinguishing features

- Development work is partitioned into "packets"
- Testing and documentation are on-going as the **product** is constructed
- Work occurs in "sprints" and is derived from a "backlog" of existing **Requirements**
- Daily meetings are very short and sometimes conducted without chairs
- "demos" are delivered to the **Customer** with the **time-box** allocated



http://ahmed-nasr.blogspot.com/2009/05/agile-scrum-flow-chart.html

Virtual Agile Scrum Certification Training from Improving.

1/24/2022

Upcoming Classes

DATE	COURSE	LOCATION	
Feb 08 - Feb 10 4:00 PM - 9:00 PM GMT	Professional Scrum Product Owner (PSPO) Certification Course	Online	Register
Feb 15 - Feb 16 3:00 PM - 11:00 PM GMT	Professional Scrum Master (PSM) Certification Course	Online	Register
Feb 15 - Feb 17 12:00 PM - 5:00 PM GMT	Professional Scrum Product Owner (PSPO) Certification Course	Online	Register

4/11-12/2023

April and May Guaranteed-to-Run Agile, Scrum Certification Training from Improving. - Hilford, Victoria - Outlook - Google Chrome

about:blank

Delete | Archive | Reply | Reply all | Forward | Read / Unread | Categorize | Flag

April and May Guaranteed-to-Run Agile, Scrum Certification Training from Improving.

Guaranteed-to-Run Scrum Certification Classes at Improving

Professional Scrum Product Owner - Advanced (PSPO) Certification course

Guaranteed to run!

April 11-12, 2023
Normally $1,495
Now **$1,095** (discount code:PSPO-Corp)
$995 for Improving Alumnus (Improving-Alumni)

The course goes beyond the topics explored in the Professional Scrum Product Owner (PSPO) class by deepening participants' understanding of the role through the exploration of the many stances of a ***professional*** Product Owner. This course is an interactive, experiential workshop where attendees explore topics through a series of exercises and discussions.

Students should have at least one year of Product Owner experience and practical knowledge of Scrum in order to maximize the benefit from discussions and exercises. It is recommended that participants first take the Professional Scrum Product Owner (PSPO) class, but is not required.

Register here

Agile Leadership Workshop

in-person at

improving
It's what we do.

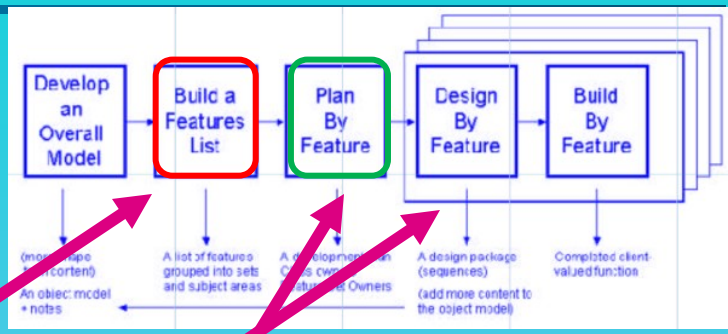
Specialized Process Models: Crystal Model

Proposed by Cockburn and Highsmith

Crystal—distinguishing features

- Actually a family of process **Models** that allow "maneuverability" based on problem characteristics
- Face-to-face communication is emphasized
- Suggests use of "reflection workshops" to review the work habits of the Team

Specialized Process Models: Feature Driven Development Model



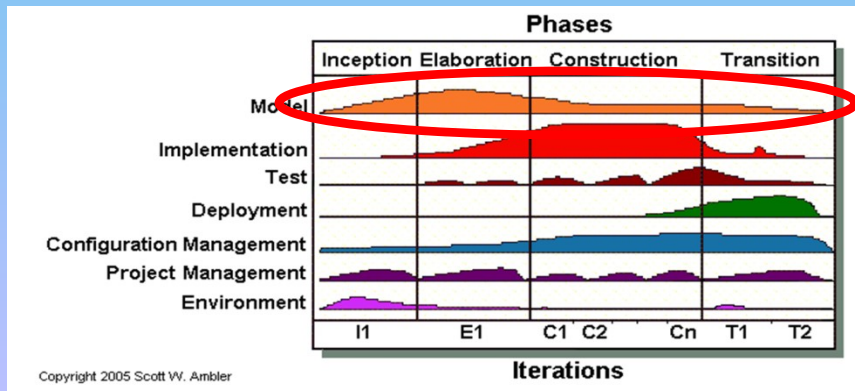
Originally proposed by Peter Coad

FDD—distinguishing features

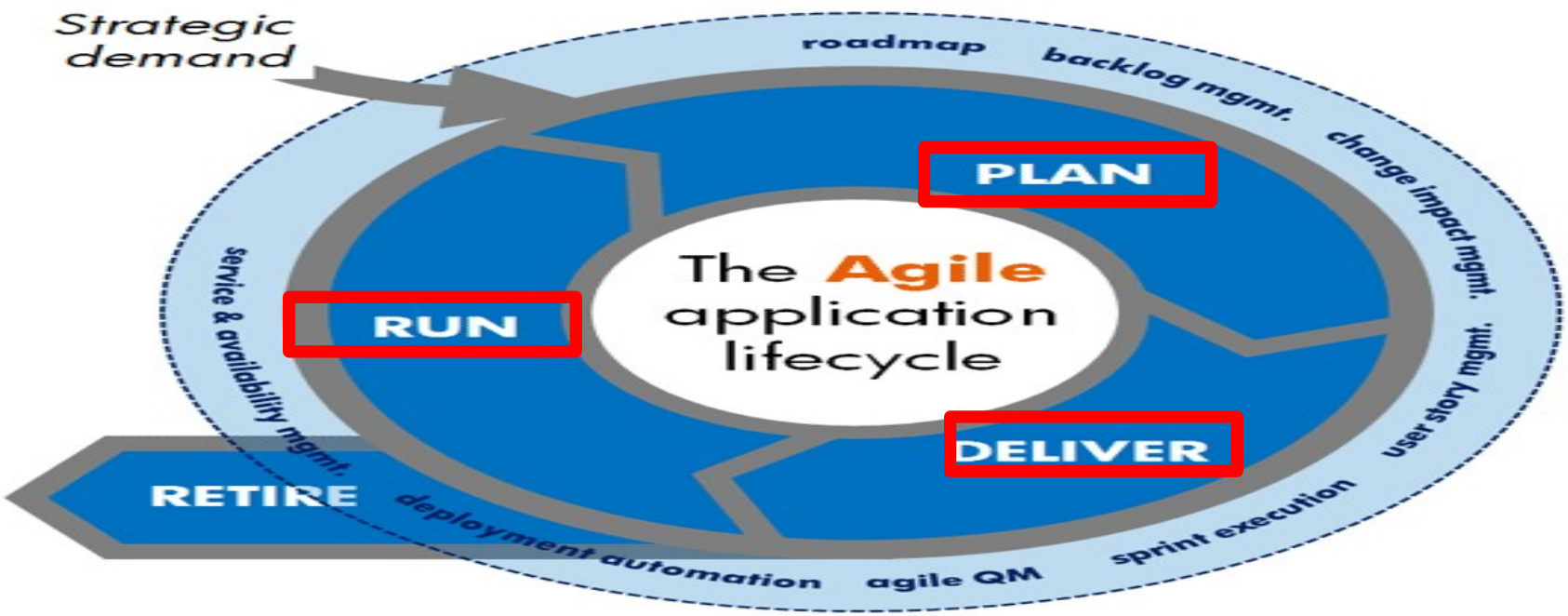
- Emphasis is on defining "features"
  - » a *feature* "is a **Client**-valued **function** that can be implemented in two weeks or less."
- Uses a feature template
  - » **<action> the <result> <by | for | of | to> a(n) <object>**
- A Features List is created and "Plan By Feature" is conducted
- Design and Construction merge in FDD

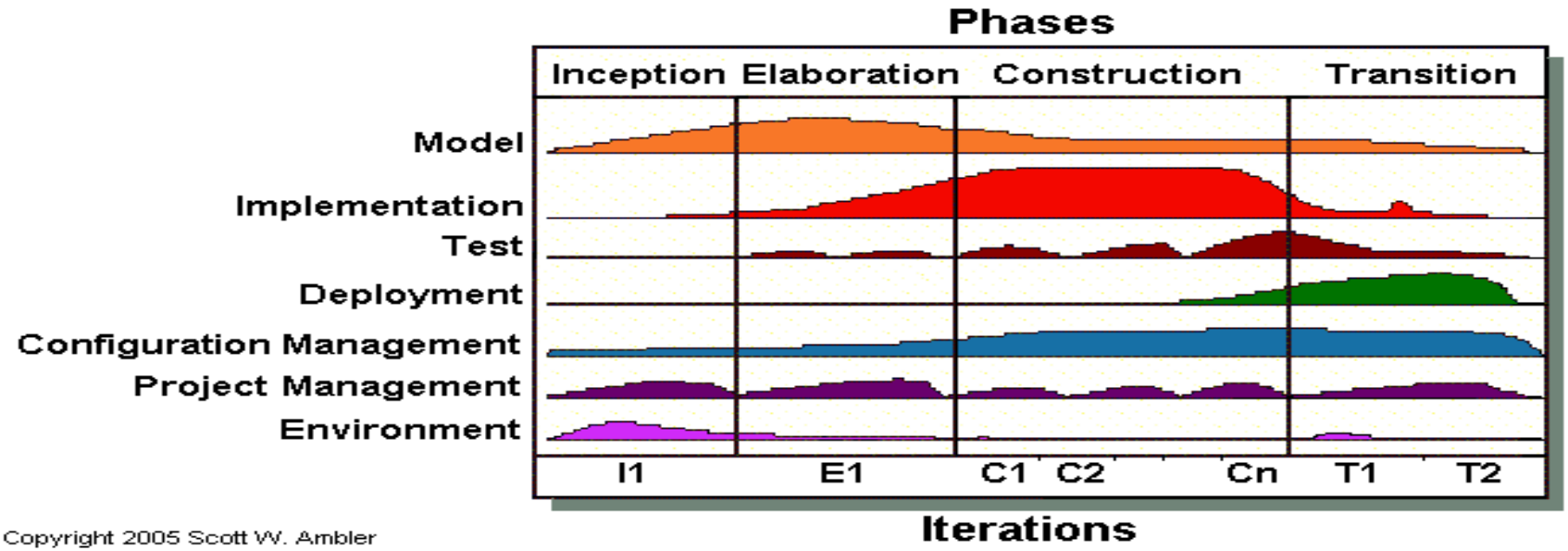
Specialized Process Models: Agile Model

- Originally proposed by Scott Ambler
- Suggests a set of **Agile Model principles**
  - **Model** with a purpose
  - Use multiple **Models**
  - **Travel light**
  - **Content** is more important than representation
  - Know the **Models** and the Tools you use to create **them**
  - **Adapt** locally



Agile Unified Process (AUP)





The Manifesto for Agile Software Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** **over processes and tools**
- ***Working software*** **over comprehensive documentation**
- ***Customer collaboration*** **over contract negotiation**
- ***Responding to change*** **over following a plan**

That is, while there is value in the items on the right, we value the items on the left more."

Kent Beck et al

Describe the sort of **product** that would be an ideal application for an **Agile Process**.

A small **product**, particularly one in which the **Requirements** are vague.

Now describe the type of situation where an **Agile Process** is inappropriate.

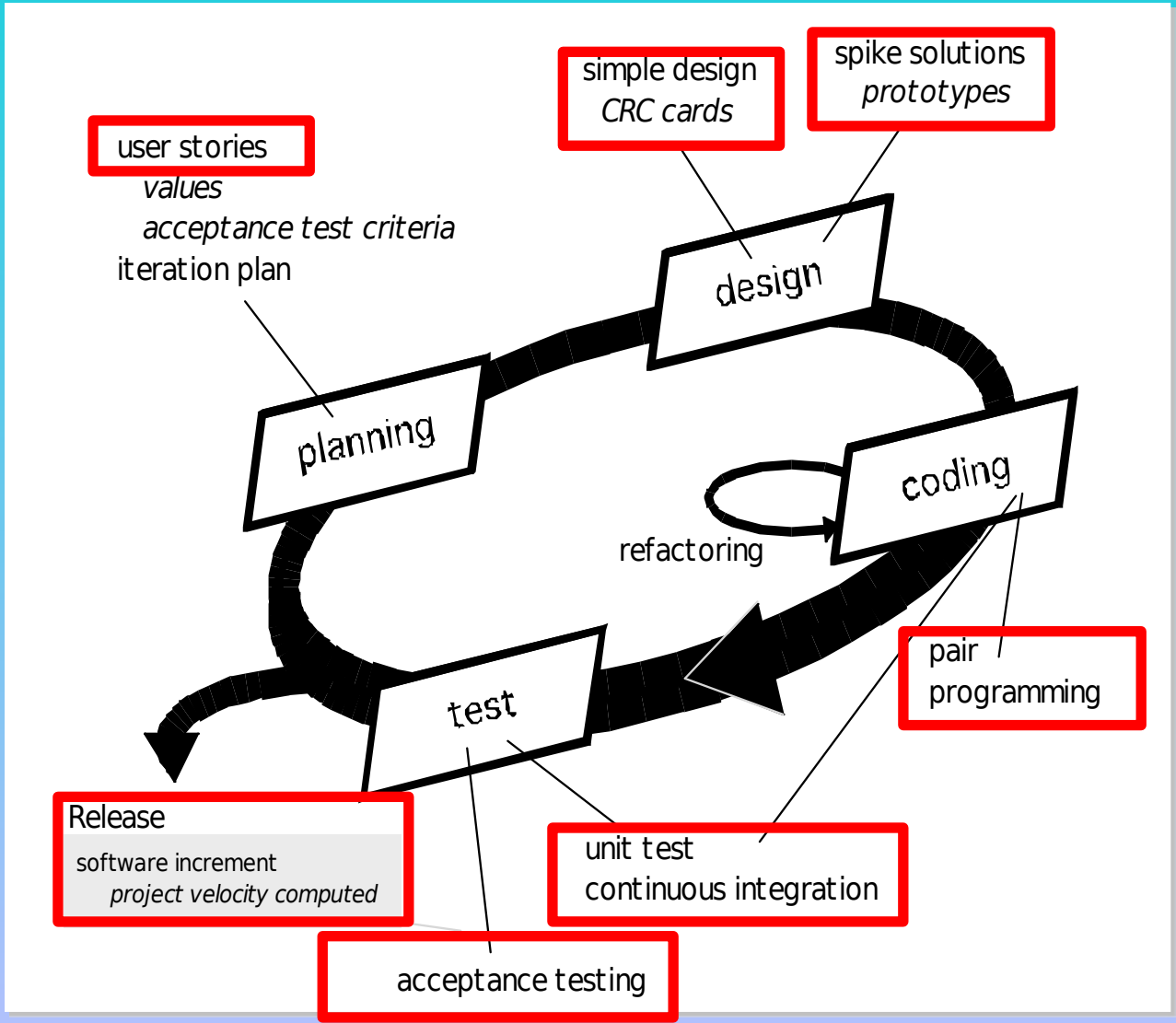
A medium or large scale **product**.

What would you say?

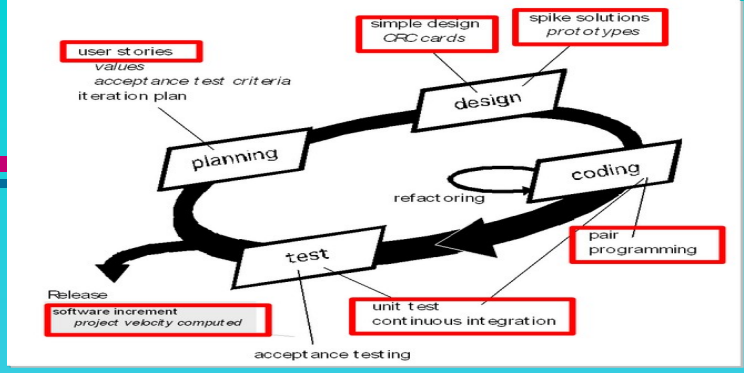
Specialized Process Models: EXtreme Programming (XP) Model

- Widely used **Agile Process**, proposed by Kent Beck
- XP Planning
  - Begins with the creation of "user stories" **UML Use Case Description**
  - Agile **team** assesses each **story** and assigns a **cost**
  - **Stories** are grouped to form a deliverable Increment
  - A commitment is made on **delivery date**

EXtreme Programming (XP) Model



EXtreme Programming (XP)



XP **Design**

- Follows the KIS**S** principle
- Encourage the use of CRC cards (**C**lass-**R**esponsibility-**C**ollaboration)
- For difficult design problems, creation of "spike solutions"—a design prototype
- Encourages "refactoring"—an Iterative refinement of the internal **program Design**

XP **Coding**

- Recommends the construction of a unit test *before* coding commences (**TDD**)
- Encourages "pair programming"

XP **Testing**

- All unit tests are executed **daily** (**Continuous Integration**)
- "Acceptance tests" defined by **Customer**; executed to assess **Customer** functionality

After the first Increment "project velocity" is used to help define subsequent **delivery dates** for other Increments

Describe some features of the Extreme Programming (XP) Life-Cycle Model.

Extreme programming [Beck, 2000] is a somewhat controversial new approach to software development based on the iterative-and-incremental model. The first step is that the software development team determines the various features (**stories**) the client would like the product to support. For each such feature, the team informs the client how long it will take to implement that feature and how much it will cost. This first step corresponds to the requirements and analysis workflows of the iterative-and-incremental model (Figure 2.4).

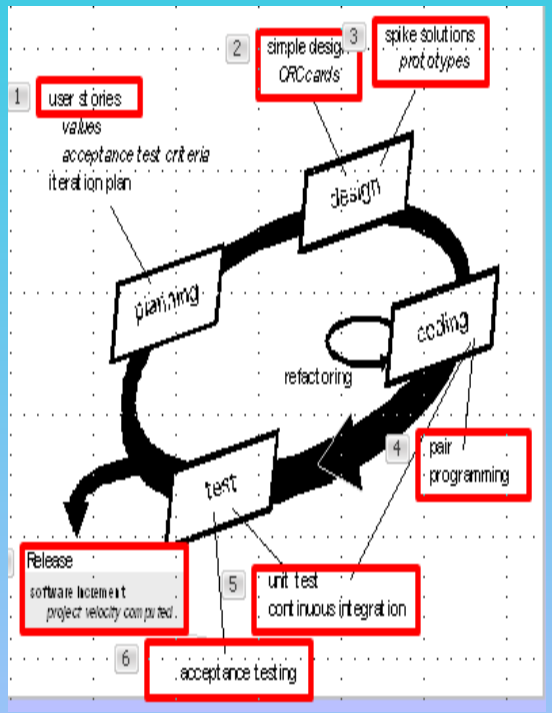
The client selects the features to be included in each successive build using cost–benefit analysis (Section 5.2), that is, on the basis of the duration and the cost estimates provided by the development team as well as the potential benefits of the feature to his or her business. The proposed build is broken down into smaller pieces termed **tasks**. A programmer first [illegible] test cases for a task; this is termed **test-driven development** (TDD). [illegible] mers work together on one computer (**pair programming**) [Williams, Kessler, Cunningham, and Jeffries, 2000], implementing the task and ensuring that all the test cases work correctly. The two programmers alternate typing every 15 or 20 minutes; the programmer who is not typing carefully checks the code while it is being entered by his or her partner. The task is then integrated into the current version of the product. Ideally, implementing and integrating a task should take no more than a few hours. In general, a num[illegible] implement tasks in parallel, so integration is essentially continuous [illegible] change coding partners daily, if possible; learning from the other team members increases everyone's skill level. The TDD test cases used for the task are retained and utilized in all further integration testing.

Some drawbacks to pair programming have been observed in practice [Drobka, Noftz, and Raghu, 2004]. For example, pair programming requires large blocks of uninterrupted time, and software professionals can have difficulty in finding 3- to 4-hour blocks of time. In addition, pair programming does not always work well with shy or overbearing individuals, or with two inexperienced programmers.

UML

TDD

ccnet



What would you say?

Prescriptive Process Models

Linear Models
Iterative Models
Evolutionary Models
Specialized Models
Adaptive Models

. . . . . .

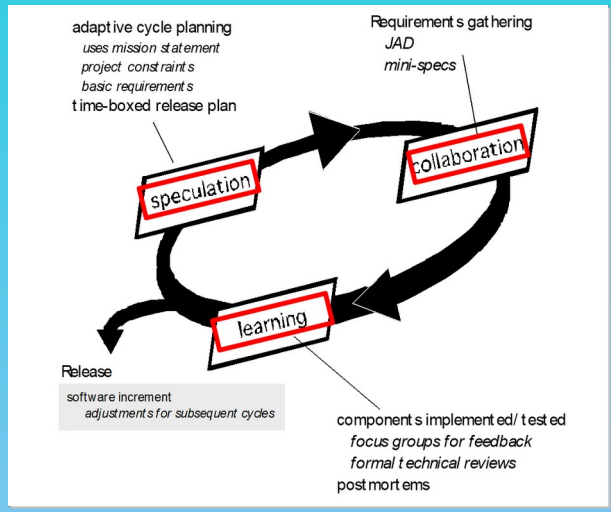
. . . . . .

. . . . . .

Specialized Process Models: Adaptive Software Development

- Originally proposed by Jim Highsmith

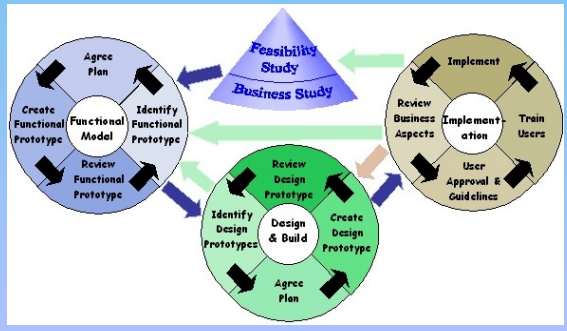
- ASD — distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing"
  - Explicit consideration of risks
  - Emphasizes collaboration for Requirements gathering
  - Emphasizes "learning" throughout the **process**



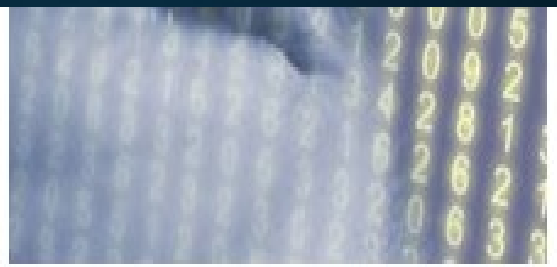
Specialized Process Models: Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)

- DSDM—distinguishing features
  - Similar in most respects to **XP** and/or **ASD** (**A**daptive **S**oftware **D**evelopment)
  - Nine guiding principles
    - » **Active User** involvement is imperative.
    - » **DSDM teams** must be empowered to make decisions.
    - » The focus is on frequent delivery of **products**.
    - » Fitness for business purpose is the essential criterion for Acceptance of **deliverables**.
    - » Iterative and Incremental development is necessary to converge on an accurate business solution.
    - » All changes during **development** are reversible.
    - » **Requirements** are baselined at a high level
    - » **Testing** is integrated throughout the life-cycle (**TDD**).



http://beta.codeproject.com/KB/architecture/dsdm.aspx?msg=629667



He was quoting from the Standish CHAOS Report that comes out every couple of years and documents the success and failure rates of IT projects. The CHAOS reports have been published since 1994, the same year DSDM appeared and when many agile methods were getting going. Each year the results vary slightly, but the general theme is that many IT projects are challenged and results like the following are typical:

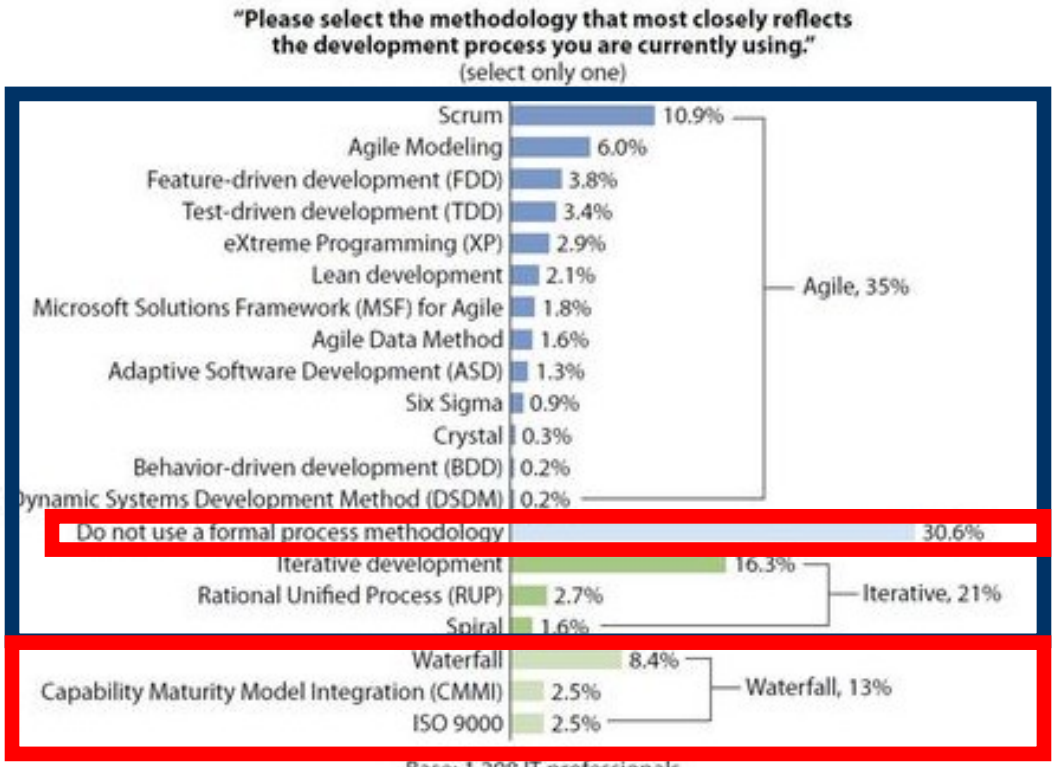
1994

* 32% Successful (On Time, On Budget, Fully Functional)
* 44% Challenged (Late, Over Budget, And/Or Less than Promised Functionality)
* 24% Failed (Cancelled or never used)
* 61% Feature complete

It is interesting then that Ken attributes the poor success rates of IT projects since the start of agile to be a PMI problem. You would think that with the rise of agile methods and the success of all these Scrum, XP, FDD, and DSDM projects we hear about, that these statistics would have turned right around!

control project managers? Well, not if you believe Forester Research and Dr Dobb's. They indicate that agile methods are now used more than any other approach for IT projects. With 35% of companies using agile and a further 21% using some kind of iterative approach and only 13% using a waterfall approach.

Figure 1 Agile Adoption Has Reached Mainstream Proportions



Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009

2011

So if 56% of projects, back in 2009, were already getting the adaptive feedback benefits of an iterative approach why are the CHAOS report findings not getting any better? Also the increase in agile continues and last year Gartner predicted *"By 2012 agile development methods will be utilized in 80% of all software development projects"*. So why the poor results, and tell me again why the failures we see are PMI problems caused by "their predictive approach" when it appears these approaches are already in the minority?

NonPrescriptive Process Models

Open Source Models

. . . . . .

. . . . . .

. . . . . .

NonPrescriptive Process Models – Open Source

- Postdelivery Maintenance Life Cycle **Model**

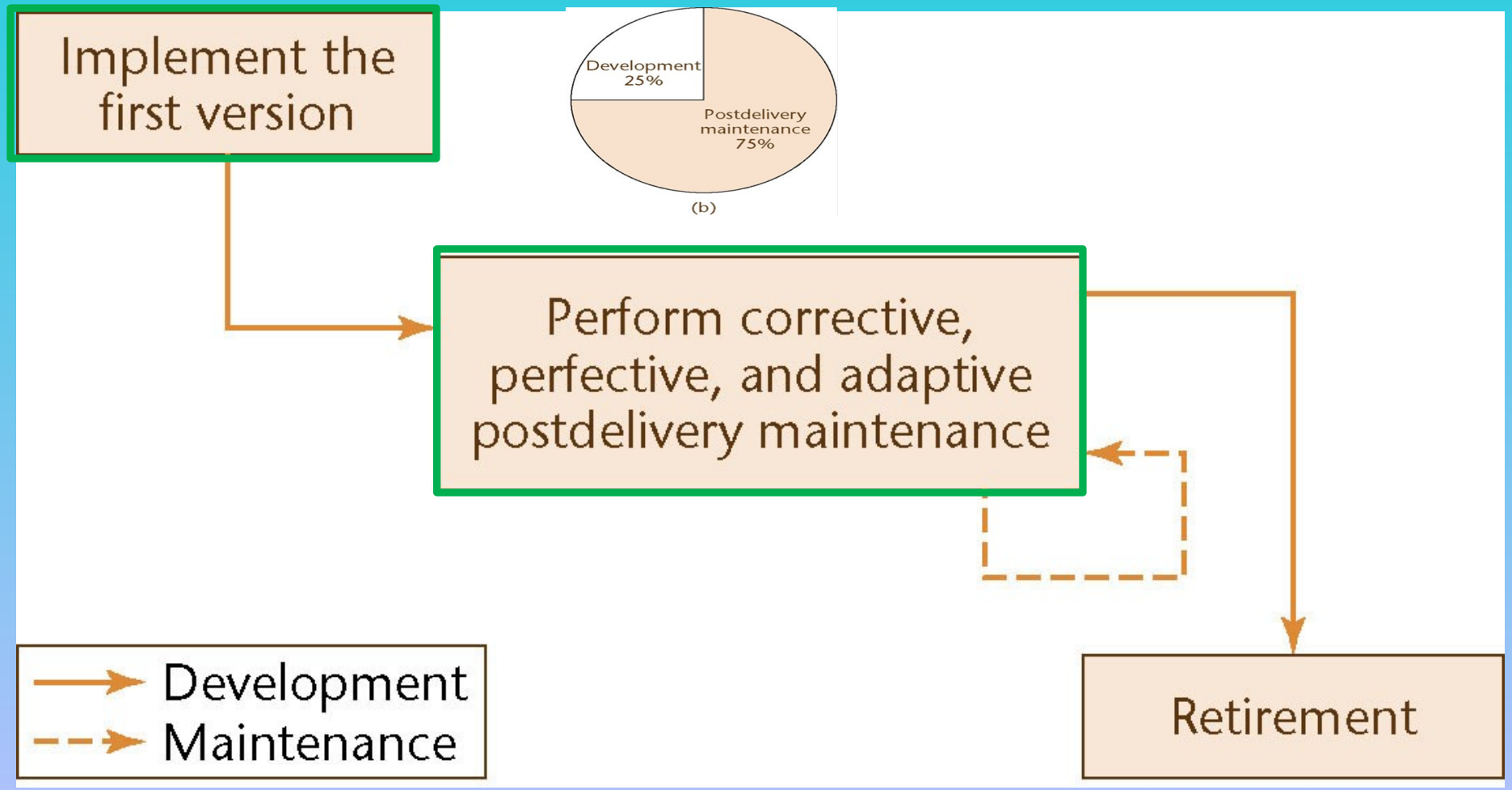
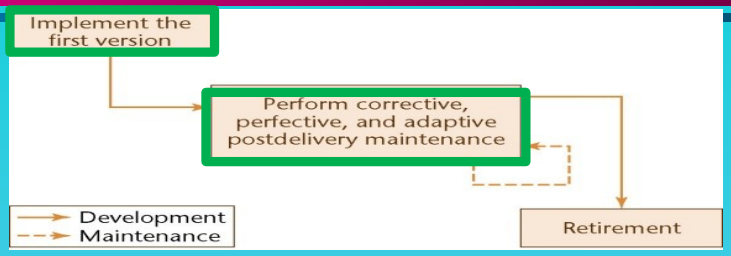
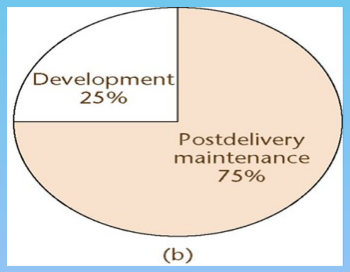


Figure 2.11

NonPrescriptive Process Models – Open Source



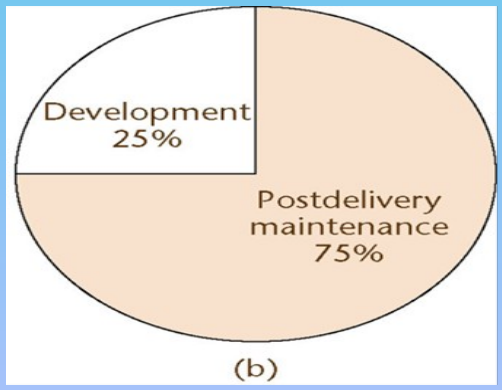
- Two informal phases
- First, **one individual** builds an initial first version
  - Made available via the Internet (e.g., SourceForge.net)
- Then, if there is sufficient interest in the project
  - The initial version is widely downloaded
  - **Users** become **co-developers**
  - The product is extended



- Key point: **Individuals** generally work voluntarily on an open-source project in their spare time

NonPrescriptive Process Models – Open Source

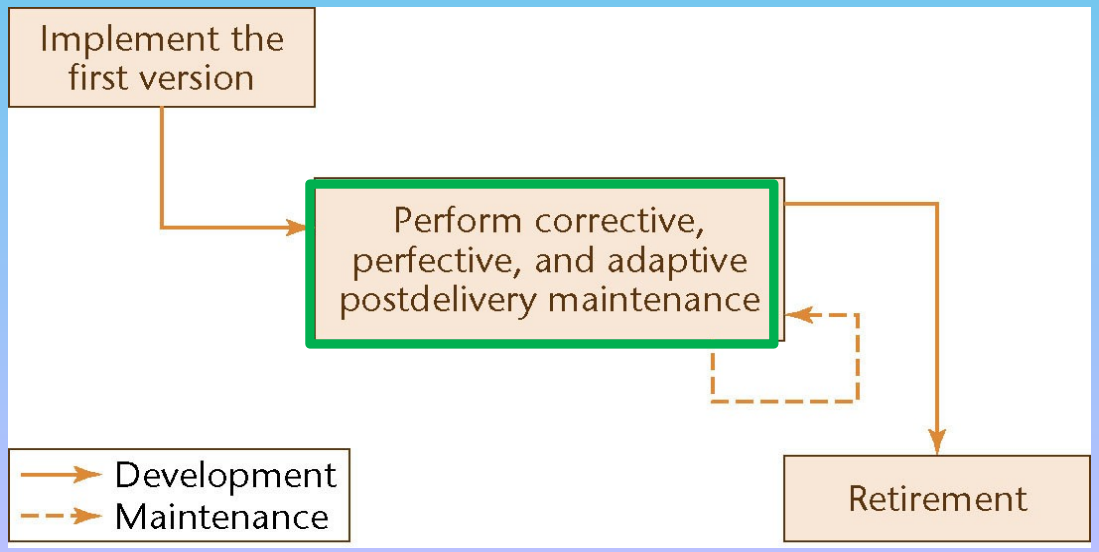
- Open-Source software is generally maintained by unpaid **volunteers**
  - **Users** are strongly encouraged to submit defect reports, both failure reports and fault reports



NonPrescriptive Process Models – Open Source

Consequently, in an open-source project, there are generally no **Specifications /Analysis (WHAT)** and no **Design (HOW)**

How have some Open-Source projects been so successful without **Specifications/Analysis** or **Designs**?



NonPrescriptive Process Models – Open Source

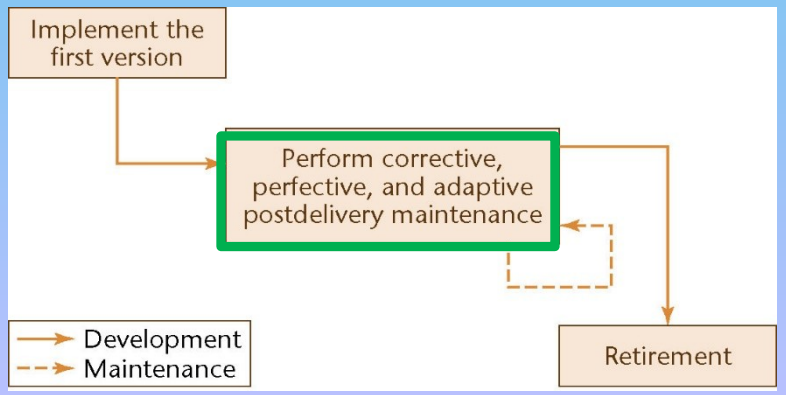
- The Open-Source Life Cycle **Model** is **restricted** in its applicability

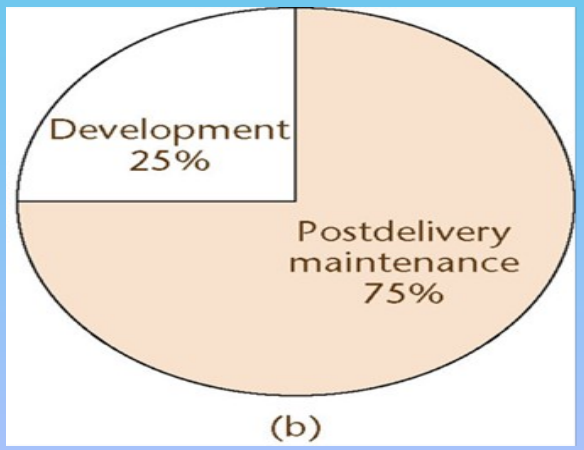
- It can be extremely successful for infrastructure projects:
  - Operating Systems (**Linux**, OpenBSD, Mach, Darwin)
  - Web Browsers (**Firefox**, **Netscape**)
  - Compilers (**gcc**)
  - Web Servers (**Apache**)
  - DataBase Management Systems (**MySQL**)

Describe a risk inherent in using the Open-Source Life-Cycle Model.

There is a risk that the project may not attract a team to work on the project.

Also, even if a start is made, **the members of the team may lose interest** during the course of the project.





What would you say?

From 4:10 to 4:55 – 45 minutes.

COSC 4351 Company Fundamental Software Engineering

SDLC Models

END

At 4:55 PM.

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	

TUTORIALS

TUTORIAL 1 on UML MODELING

- Tutorial 1 on UML MODELING.pptx
- Movie Company System Requirements.doc
- TA for UML USE CASE Diagram MODEL Movie Company System.doc
- TA for UML MVC CLASS Diagram MODEL Movie Company System.doc

TUTORIAL 1 on ERD MODELING

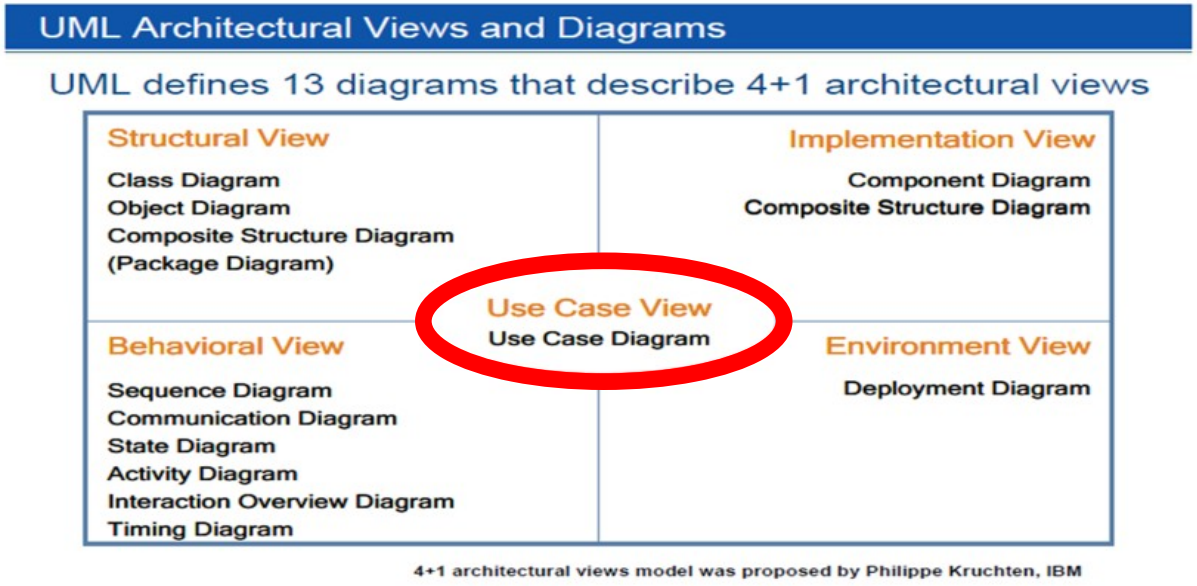
- TUTORIAL 1 on ERD - WHAT DATA.ppt
- Template DB Team Project with Line Numbers for ERD Modeling.pdf

VH, publish.

The Unified Modeling Language

The **Unified Modeling Language** (**UML**) has become a **standard** notation for **Object Oriented software Analysis (OOA) & Design (OOD – "blue print")**

It includes various types of **UML Diagrams** (**Models**) which use **specific icons and notations**



Movie Company System

Requirements

The Movie Company wants to keep a catalog of its artists and their discs (maximum 30).

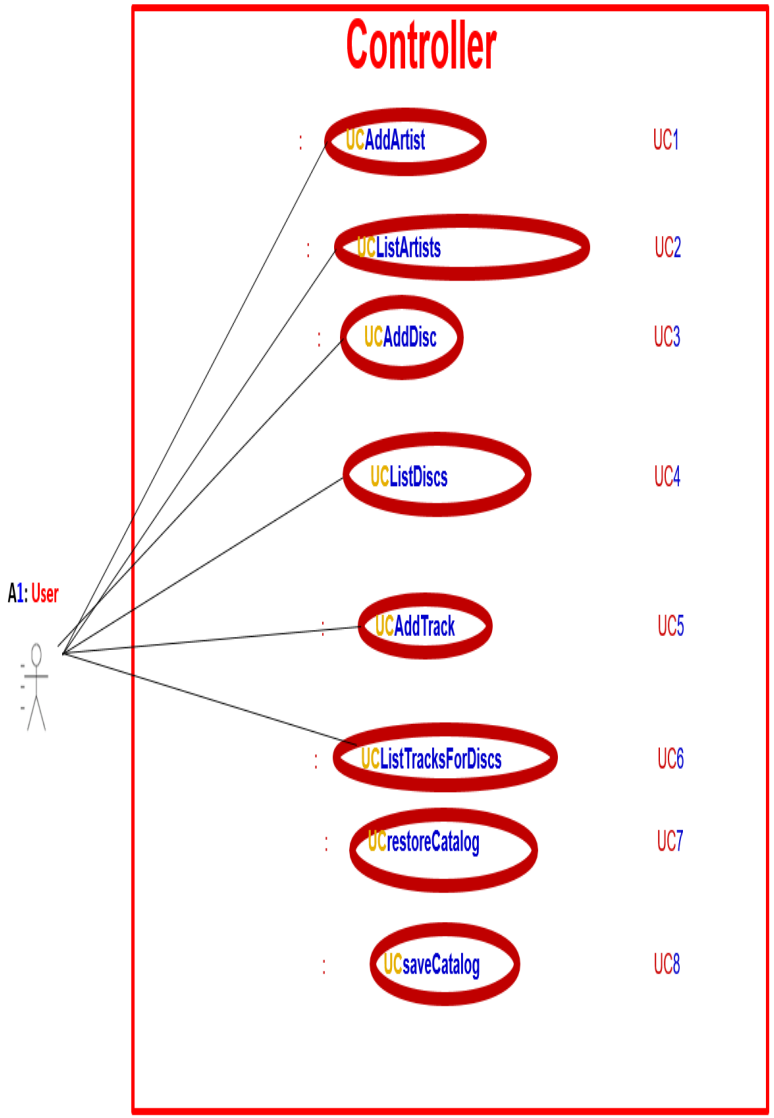
Artists have a name. The Movie Company can *add an artist* or get a *listing of all its artists*.

Each artist produces music Discs. A Disc is associated with an Artist. A Disc is cataloged with a disc catalogNumber. The Movie Company can *add a disc* or get a *listing of all its discs*.

A Disc has a number of Tracks. The Track have a title and a duration time. A Disc has a catalogNumber , a title, the number of tracks. The Movie Company can *add a track* or get a *listing of all tracks for a given disc*.

Movie Company System

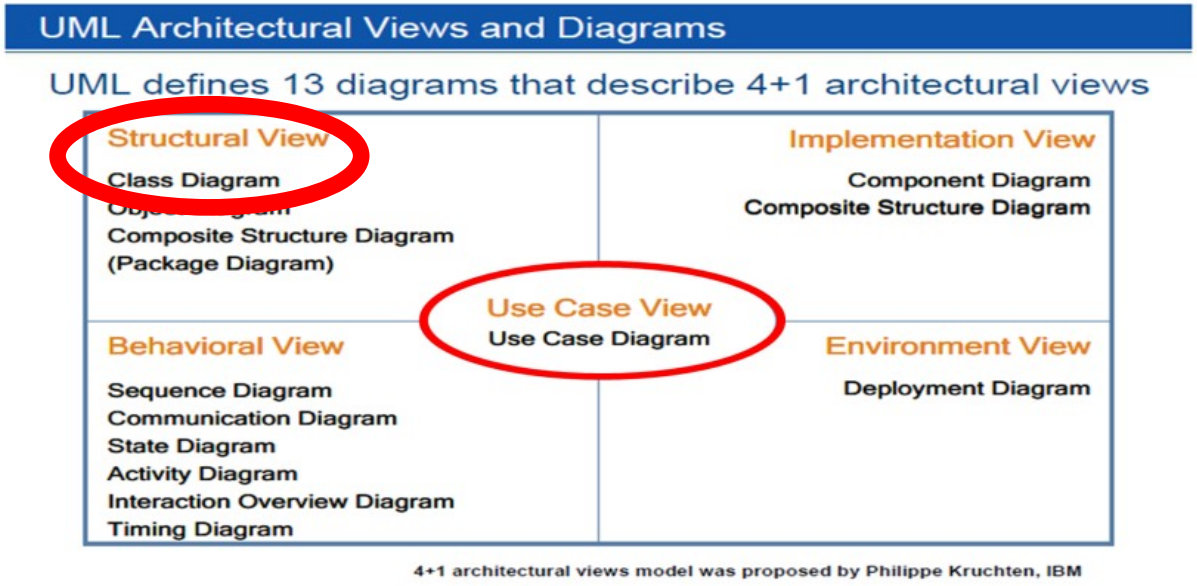
UML USE CASE MODEL



The Unified Modeling Language

The **Unified Modeling Language** (**UML**) has become a **standard** notation for **Object Oriented software Analysis (OOA) & Design (OOD – "blue print")**

It includes various types of **UML Diagrams** (**Models**) which use **specific icons and notations**



Movie Company System

Requirements

The Movie Company wants to keep a catalog of its artists and their discs (maximum 30).

Artists have a name. The Movie Company can *add an artist* or get a *listing of all its artists*.

Each artist produces music Discs. A Disc is associated with an Artist. A Disc is cataloged with a

disc catalogNumber. The Movie Company can *add a disc* or get a *listing of all its discs*.

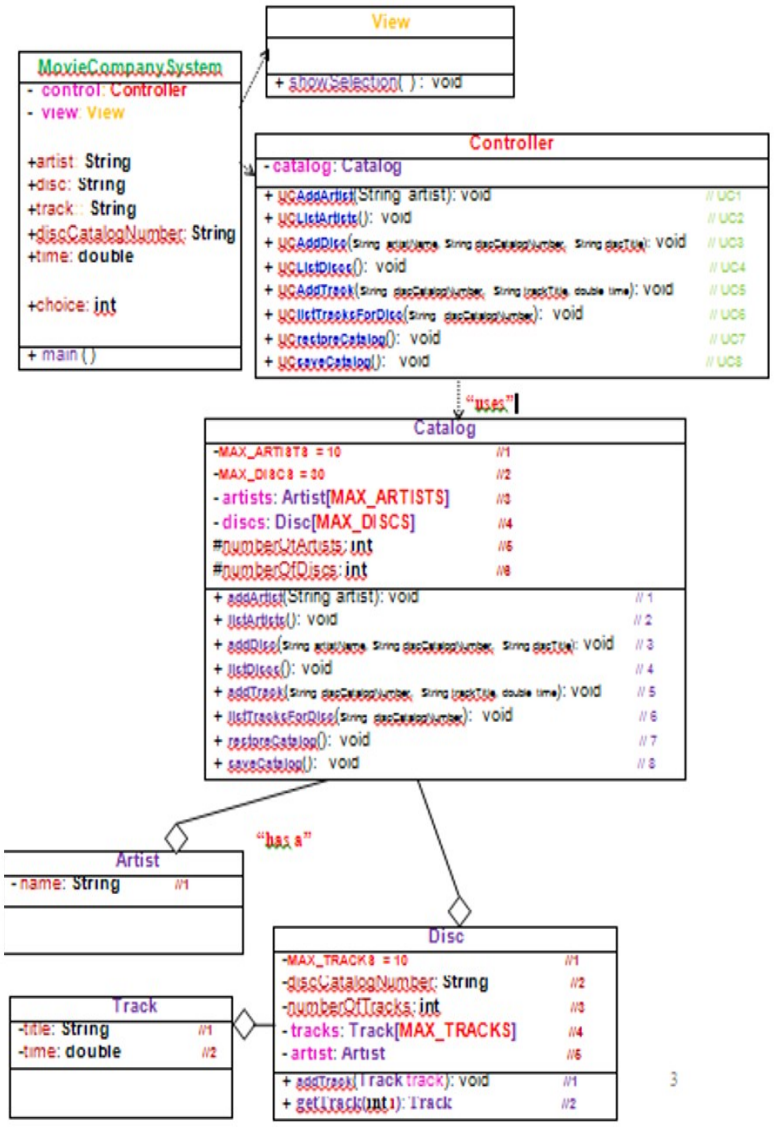
A Disc has a number of Tracks. The Track have a title and a duration time. A Disc has a

catalogNumber , a title, the number of tracks. The Movie Company can *add a track* or get a

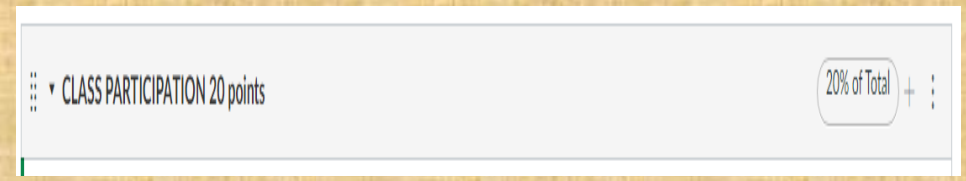
listing of all tracks for a given disc.

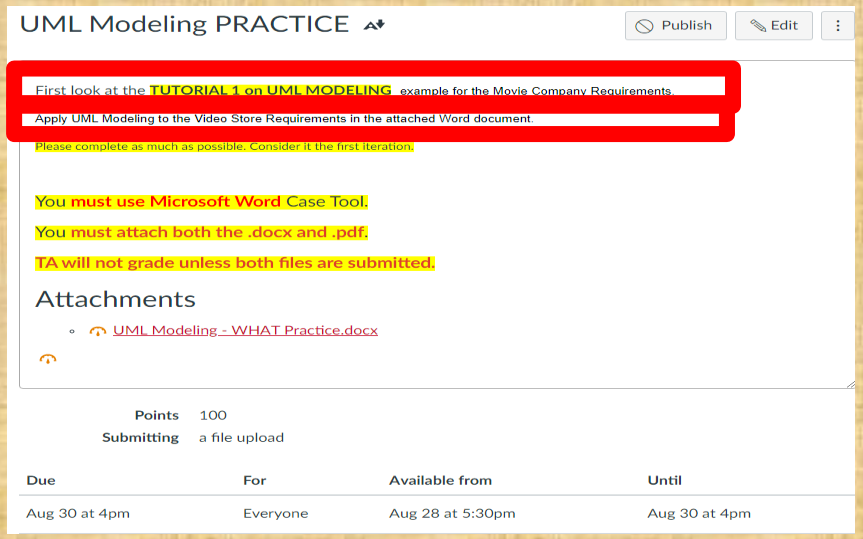
Movie Company System

UML MVC CLASS MODEL



08.30.2023 (W 4 to 5:30) (4)		Tutorials 1 on WHAT tools (UML & ERD Modeling)		UML Modeling CANVAS Assignment	





Due August 30, 4:00 PM

Grade:

Class Participation 20%

COSC 4351

Name: _____

UML Modeling – WHAT Practice

Apply the example from UML Modeling TUTORIAL to these requirements. Please complete as much as possible. Consider it the first iteration.

You must use Microsoft Word Case Tool.

DVD Collection Application Requirements:

A video store wishes to automate their processes related to their collection of DVDs. The data for each DVD will consist of title, a category, running time, year of release, and price. These DVDs are stored in Catalog.txt.

First iteration

3. display the collection of DVDs sorted by year

The client is estimating that there will not own more than 100 DVDs.

Catalog.txt

Amadeus,Drama,160 Mins.,1984, 14.83

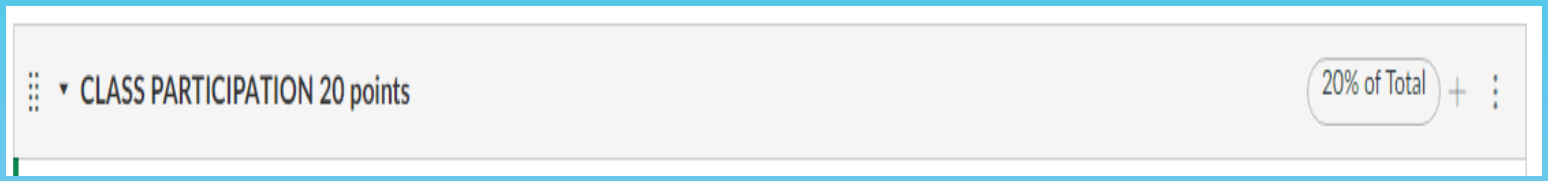
As Good As It Gets,Drama,139 Mins.,1998,11.3

VH, next UML & ERD Modeling

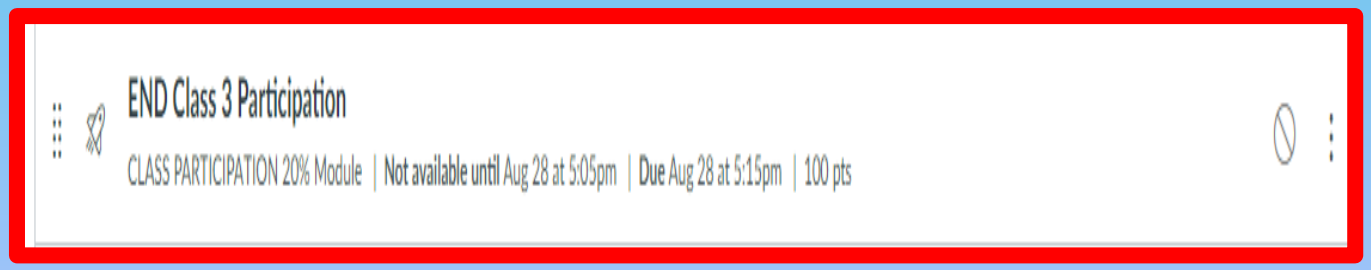
08.30.2023 (W 4 to 5:30) (4)		Tutorials 1 on WHAT tools (UML & ERD Modeling)		UML Modeling CANVAS Assignment	

From 5:05 to 5:15 – 10 minutes.

08.28.2023 (M 4 to 5:30) (3)	Review Tutorials 1 on WHAT tools (UML & ERD Modeling)	Lecture 2: SDLC		SDLC Papers Summary (1 Page) CANVAS Assignment	



PASSWORD: CLASS 3



VH, publish.

VH Download Meeting Participants

At 5:15 PM.

End Class 3

VH, upload Class 3 to CANVAS.

VH, Download Attendance Report
Rename it:
8.28.2023 Attendance Report FINAL