The background is a light blue gradient with several realistic water droplets of various sizes scattered across it. The droplets have highlights and shadows, giving them a 3D appearance.

Introduction To Software Engineering

Presented by:

Basma Hussien Mohamed

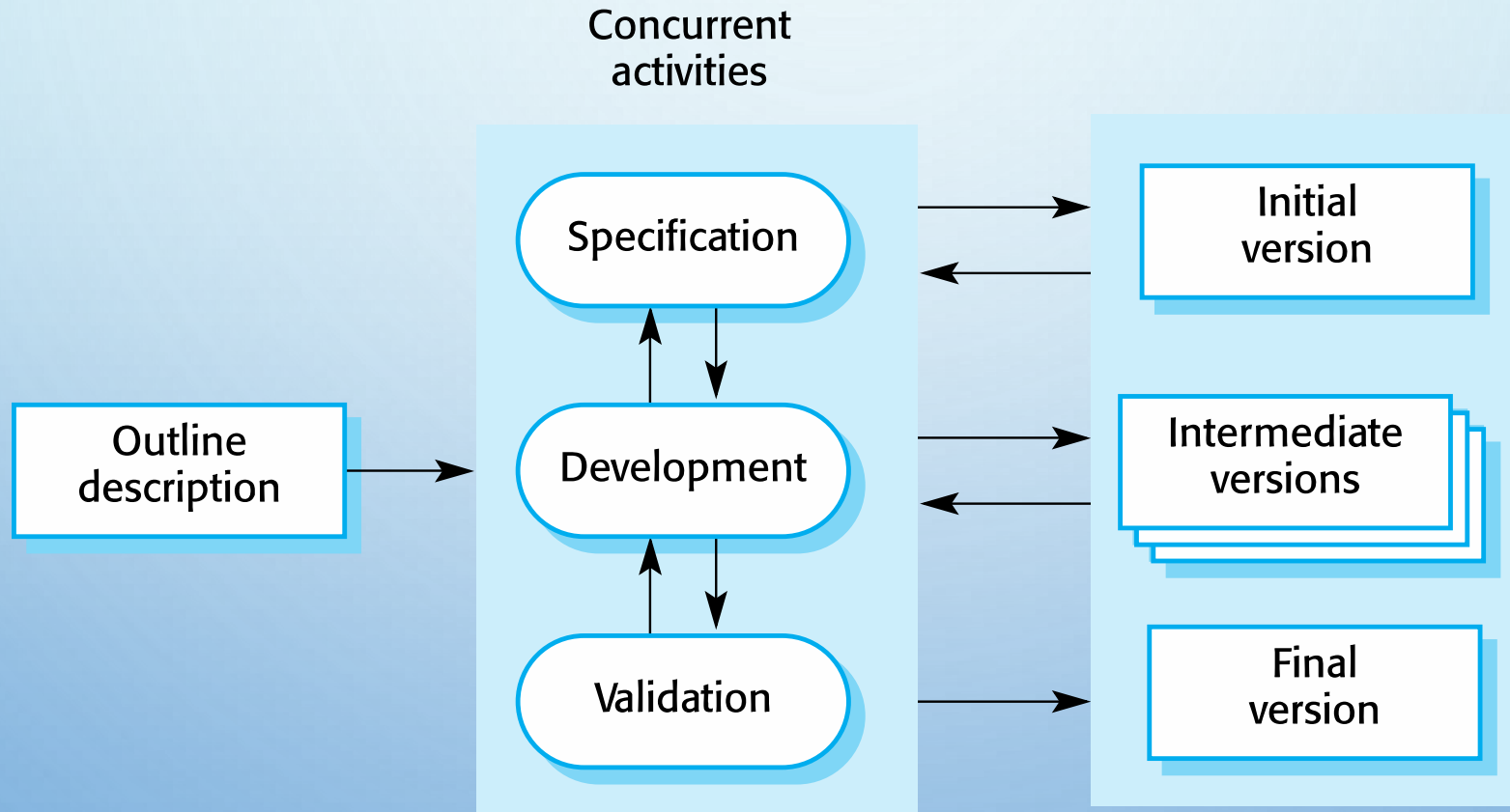
Yesterday Assignments

- 3 Case Studies
- V-Model
- Spiral Model

The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The text is centered in the middle of the slide.

Incremental Development Summary

Incremental development



Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
 - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
 - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
 - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- The process is not visible.
 - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
 - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure.
 - Incorporating further software changes becomes increasingly difficult and costly.

Incremental development problems (Cont.)

- Most systems require a set of basic facilities that are used by different parts of the system.
 - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
 - However, this **conflicts** with the procurement model of many organizations, where the complete system specification is part of the system development **contract**.

Iteration vs Increment

Iteration vs Increment

- When discussing iterative and incremental development, the terms *iteration* and *increment* are often used freely and interchangeably. However, **They are NOT synonyms.**
- ***Iteration*** refers to the cyclic nature of a process in which activities are repeated in a structured manner.
- ***Increment*** refers to the quantifiable outcome of each iteration.

Iteration vs Increment

- Iterations can offer a development process two key things:
 - iterative refinement, where the process improves what already exists and is being done.
 - and incremental development, where the process results in progress against project objectives.
- Additionally, the term *increment* has the obvious implication that there should be more of something at the end of an iteration than there was at the start “new release”.
- Without a clear notion of an increment, iterations are likely to just go in circles!!
- **So, we develop software *iteratively* and release *incrementally* in various sizes over time.**

The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The title text is centered in the middle of the slide.

Reuse-Oriented Software Engineering

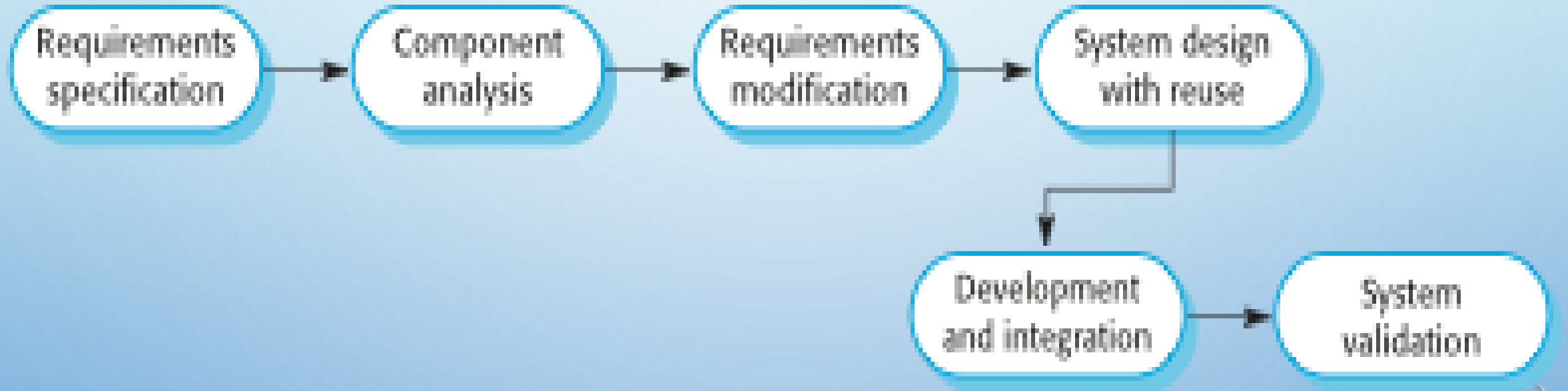
Reuse-Oriented Software Engineering (Integration and configuration)

- Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems.
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system

Types of reusable software

- Web services that are developed according to service standards and which are available for remote invocation.
- Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.

Reuse-oriented software engineering



Advantages and disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But...
- requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements


The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The text is centered in the middle of the slide.

Lets Make An Activity...

Process activities

Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of **specification**, **development**, **validation** and **evolution** are organized differently in different development processes.
- For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The droplets have highlights and shadows, giving them a 3D appearance.

1. Software Specifications (Analysis)

Software specification

Is the process of establishing what services are required and the constraints on the system's operation and development.

Software specification (Cont.)

- **Software Requirements**

- Description of features and functionalities of the target system.
- Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

- **Requirement Engineering**

- The process to gather the software requirements from client, analyze and document them is known as requirement engineering.
- The **goal** of requirement engineering is **to develop and maintain sophisticated and descriptive SRS** 'System Requirements Specification' document.

Requirements Engineering

Requirements engineering process:

A. Requirements elicitation and analysis

- What do the system stakeholders require or expect from the system?

B. Requirements specification

- Defining the requirements in detail “output SRS sheet”

C. Requirements validation

- Checking the validity of the requirements

A. Requirements Elicitation

- **Requirements elicitation “Not Gathering” and analysis:**
 - What do the system stakeholders require or expect from the system?

Why Not Requirements Gathering?

Requirements Gathering



- Like collecting sea shells
- Take what you see
- More reactive, less proactive

VS.

Requirements Elicitation



- Like archeology
- Planned, deliberate search
- More proactive, less reactive

Requirements Elicitation (Cont.)

- **Requirements Elicitation Techniques:**

- Brainstorming
- Document Analysis
- Focus Groups
- Interviews
- Observation
- Prototyping
- Survey/Questionnaire

Requirement Elicitation Techniques

- **Interviews**

- Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:
 - **Structured (closed) interviews**, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
 - **Non-structured (open) interviews**, where information to gather is not decided in advance, more flexible and less biased.
 - **Oral interviews (Ex. Phone or Video Calls)**
 - **Written interviews**
 - **One-to-one interviews** which are held between two persons across the table.
 - **Group interviews** which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.
- You may record interviews for further documentation.

Requirement Elicitation Techniques (Cont.)

- **Surveys**

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

- **Questionnaires**

- A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.
- A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

- **Task “Document” analysis**

- Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Requirement Elicitation Techniques (Cont.)

- **Domain Analysis “Focus Group”**

- Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

- **Brainstorming**

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

- **Observation**

- Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Requirement Elicitation Techniques (Cont.)

- **Prototyping**

- Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product.
- It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements.
- The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

B. Requirements specification

- **Requirement Specification Sheet:**

- **SRS** is a document created by system analyst after the requirements are collected from various stakeholders.
- SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.
- **The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.**

Requirements specification (Cont.)

- SRS should come up with following features:
 - User Requirements are expressed in natural language.
 - Technical requirements are expressed in structured language, which is used inside the organization.
 - Design description should be written in Pseudo code.
 - Format of Forms and GUI screen prints.
 - Conditional and mathematical notations for DFDs etc.

C. Requirements validation

- **Software Requirement Validation:**

- After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly.
- **Requirements can be checked against following conditions -**
 - If they can be practically implemented
 - If they are valid and as per functionality and domain of software
 - If there are any ambiguities
 - If they are complete
 - If they can be demonstrated

Software Requirements Characteristics

- Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.
- A complete Software Requirement Specifications must be:
 - Clear
 - Correct
 - Consistent
 - Modifiable
 - Verifiable
 - Prioritized
 - Unambiguous
 - Credible source

Software Requirements Categories

- Broadly software requirements should be categorized in **two categories**:

1. Functional Requirements

- Requirements, which are related to functional aspect of software fall into this category.
- They define functions and functionality within and from the software system.

- **EXAMPLES -**

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Software Requirements (Cont.)

2. Non-Functional Requirements

- Requirements, which are not related to functional aspect of software, fall into this category. They are expected or **implicit characteristics of software**, which users make assumption of.
- Non-functional requirements include -
 - Security
 - Logging
 - Storage
 - Configuration
 - Performance
 - Cost
 - Interoperability
 - Flexibility
 - Disaster recovery
 - Accessibility

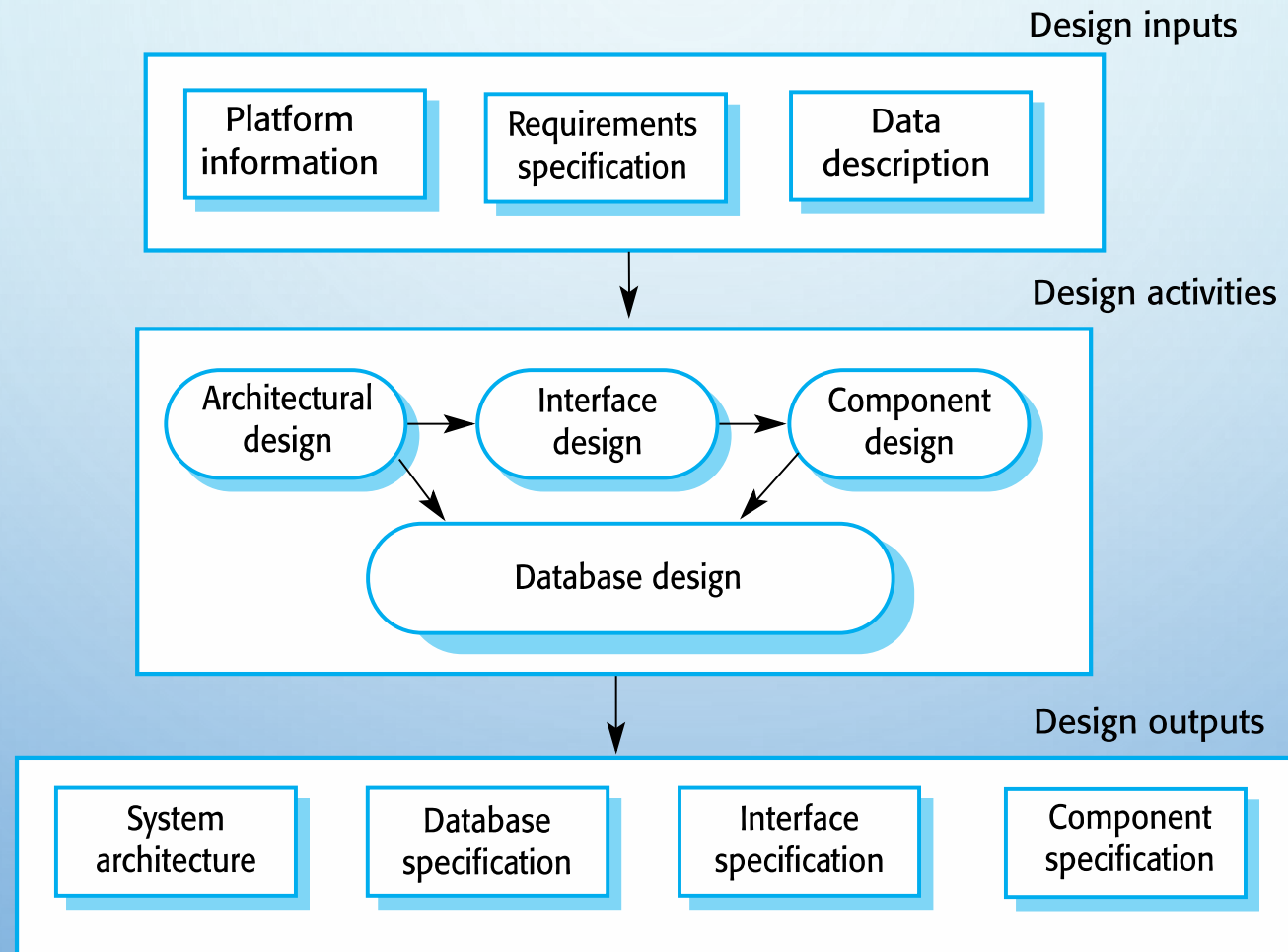


2. Software Design & Implementation

2. Software Development (Design and Implementation)

- The process of converting the system specification into an executable system.
- **Software design:**
 - Design a software that meets the specification;
- **Implementation:**
 - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

A general model of the design process



Design activities

- *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- *Database design*, where you design the system data structures and how these are to be represented in a database.
- *Interface design*, where you define the interfaces between system components.
- *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.

System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- **Debugging** is the activity of finding program faults and correcting these faults (may be thought of as a test activity).

3. Software Verification & Validation (V & V)



Verification vs validation

- **Verification:**

"Are we building the product right"?

- The software should conform to its specification and it is “**error free**”.



- **Validation:**

"Are we building the right product"?!"

- The software should do what the user really requires, it must be Valid against user requirements.



3. Software Validation

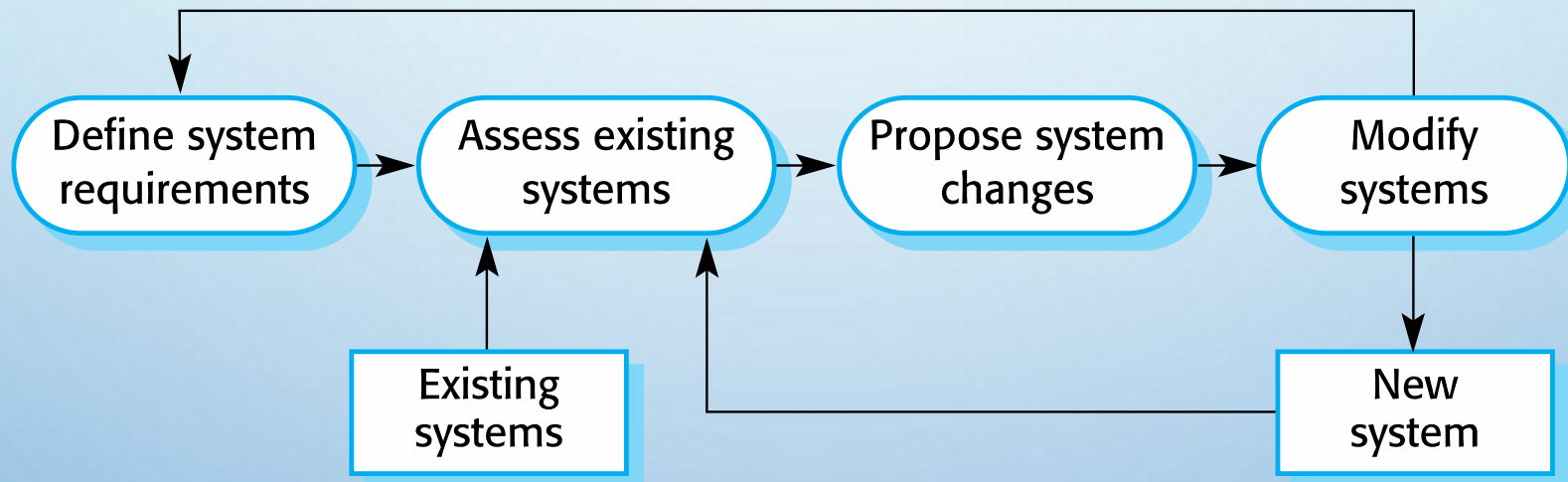
- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- **Testing** is the most commonly used V & V activity.

4. Software Evolution

4. Software evolution

- Software is flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Day2 Assignment

CASE Tools

CASE Tools

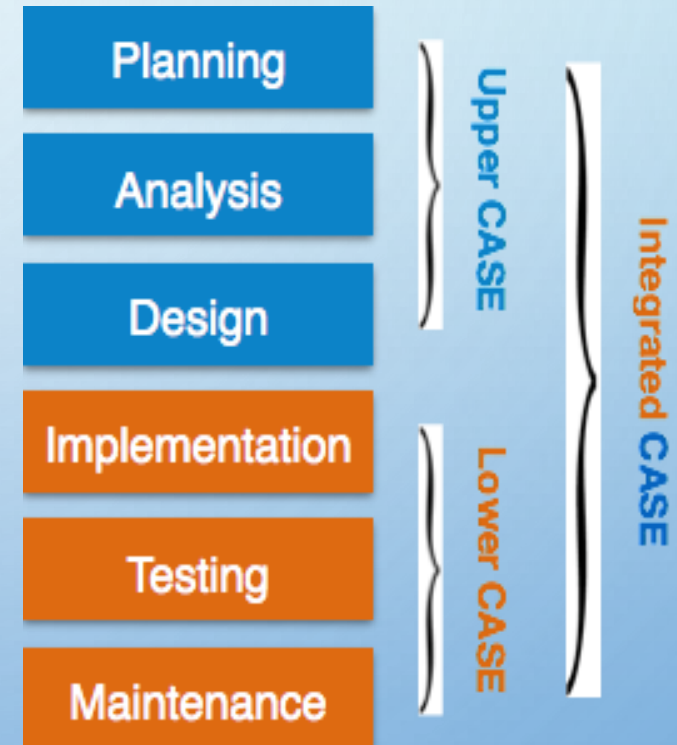
- CASE stands for **C**omputer **A**ided **S**oftware **E**ngineering. It means, development and maintenance of software projects with help of various automated software tools.
- CASE tools are set of software application programs, which are used to automate SDLC activities.
- CASE tools are used by software project managers, analysts and engineers to develop software system.

CASE Tools

- There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.
- The scope of CASE tools goes throughout the SDLC.

Components of CASE Tools (Cont.)

- **Upper CASE Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower CASE Tools** - Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated CASE Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.



CASE Tools Types

- **Diagram tools**

- These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. **For example, Flow Chart Maker tool** for creating state-of-the-art flowcharts.

- **Process Modeling Tools**

- Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. **For example, EPF Composer**

CASE Tools Types (Cont.)

- **Documentation Tools**

- Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.
- Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual.

For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

CASE Tools Types (Cont.)

- Analysis Tools

- These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.

- Design Tools

- These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules.

For example, Animated Software Design

CASE Tools Types (Cont.)

- Configuration Management Tools

- An instance of software is released under one version. Configuration Management tools deal with
 - Version and revision management
 - Baseline configuration management
 - Change control management
- CASE tools help in this by automatic tracking, version management and release management.

For example, Fossil, Git, Accu REV.

- Change Control Tools

- These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

CASE Tools Types (Cont.)

- **Programming Tools**

- These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. **For example, Cscope to search code in C, Eclipse.**

- **Prototyping Tools**

- Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. **For example, Serena prototype composer, Mockup Builder.**

CASE Tools Types (Cont.)

- Web Development Tools

- These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

- Quality Assurance Tools

- Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter.

CASE Tools Types (Cont.)

- Maintenance Tools

- Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

What is Agile?



Backlog		To Do	In Progress	Testing	Done
<div>Feature 10 hrs HIGH</div>	<div>Bug Fix 2 hrs Medium</div>	<div></div>	<div></div>	<div></div>	<div></div>
<div>Update 4 hrs Low</div>	<div>Research 3 hrs Medium</div>	<div></div>			<div></div>
<div>Content 2 hrs HIGH</div>		<div></div>			
		<div></div>			

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
 - Focus on the code rather than the design
 - Are based on an iterative approach to software development
 - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- **The aim of agile methods is to reduce overheads in the software process (e.g. by *limiting documentation*) and to be able to respond quickly to changing requirements without excessive rework.**



Agile manifesto

4 values

+

12 Principles

Agile Values *(from the Agile Manifesto)*

The **Agile Manifesto** establishes a set of **four values** that are **people-centric** and **results-driven**:

Individuals and Interactions		<i>over</i>	Processes and Tools
Working Software		<i>over</i>	Comprehensive Documentation
Responding to Change		<i>over</i>	Following a Plan
Customer Collaboration		<i>over</i>	Contract Negotiation

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

(www.agilemanifesto.org)

Agile 12 Principle

- 1. Customer satisfaction through early and continuous software delivery** – Customers are happier when they receive working software at regular intervals, rather than waiting extended periods of time between releases.
- 2. Accommodate changing requirements throughout the development process** – The ability to avoid delays when a requirement or feature request changes.
- 3. Frequent delivery of working software** – Scrum accommodates this principle since the team operates in software sprints or iterations that ensure regular delivery of working software.
- 4. Collaboration between the business stakeholders and developers throughout the project** – Better decisions are made when the business and technical team are aligned.
- 5. Support, trust, and motivate the people involved** – Motivated teams are more likely to deliver their best work than unhappy teams.
- 6. Enable face-to-face interactions** – Communication is more successful when development teams are co-located.

Agile 12 Principle(Cont.)

- 7. Working software is the primary measure of progress** – Delivering functional software to the customer is the ultimate factor that measures progress.
- 8. Agile processes to support a consistent development pace** – Teams establish a repeatable and maintainable speed at which they can deliver working software, and they repeat it with each release.
- 9. Attention to technical detail and design enhances agility** – The right skills and good design ensures the team can maintain the pace, constantly improve the product, and sustain change.
- 10. Simplicity** – Maximize work that should NOT be done. Develop just enough to get the job done for right now.
- 11. Self-organizing teams encourage great architectures, requirements, and designs** – Skilled and motivated team members who have decision-making power, take ownership, communicate regularly with other team members, and share ideas that deliver quality products.
- 12. Regular reflections on how to become more effective** – Self-improvement, process improvement, advancing skills, and techniques help team members work more efficiently.

Agile method applicability

- Product development where a software company is developing **a small or medium-sized product** for sale.
- Custom system development within an organization, where there is a clear commitment from the **customer to become involved** in the development process and where **there are not a lot of external rules and regulations** that affect the software.
- Because of their focus on small, **tightly-integrated teams**, there are problems in scaling agile methods to large systems.

Problems with agile methods

- It can be difficult to keep the *interest of customers* who are involved in the process.
- *Team members* may be *unsuited to the intense involvement* that characterizes agile methods.
- *Prioritizing* changes can be difficult where there are *multiple stakeholders*.
- *Maintaining simplicity* requires extra work.
- *Contracts* may be a problem as with other approaches to iterative development.

Plan-driven vs Agile

Agile vs Waterfall - Four Massive Differences

Complete IT Professional

www.completeitprofessional.com

AGILE vs. WATERFALL











PROS

-  Customer Approval During All Stages
-  Great for Quick Launches
-  Prioritized by Business Value
-  Customer Involvement Makes Project User Focused

-  Early Agreement on Deliverables
-  No Need for Customer Involvement During Development
-  Full Scope Known in Advance
-  Known Deliverables Reduce Chance of "Piecemeal Effect"

CONS

-  Disadvantages When Team is Dedicated Full-time on The Project
-  Customer May Not Have Time to be Involved
-  Customer May Redefine Scope
-  Quick Launches Can Cause Incomplete Tasks

-  Customer Only Sees Final Product and Could be Unhappy
-  Customer Has Trouble Visualizing Project in Early Stage
-  Late Changes Cause Going Over Budget
-  Late Changes Extend Project Timeline

WHAT SHOULD FACTOR INTO YOUR DECISION

Customer
Preference

Project
Size

Customer
Budget

Time To
Market

Customer
Availability

The background is a light blue gradient with several realistic water droplets of various sizes scattered across the surface. The droplets have highlights and shadows, giving them a three-dimensional appearance.

Agile Software Development



Agile

The diagram features a large umbrella with a dark grey/black canopy and a light grey underside. The word 'Agile' is written in large, bold, black letters on the top right of the canopy. The umbrella's handle is a thin grey rod that extends from the bottom of the canopy down to a brown, textured grip at the bottom right. Several agile frameworks are listed around the umbrella, with some having lines pointing to the canopy. The frameworks listed are: Crystal (top right), Feature-Driven Development (FDD) (middle right), Dynamic Systems Development Methodology (DSDM) (bottom right), Lean (middle left), Scrum (bottom left), Kanban (bottom left), and Extreme Programming (XP) (bottom left).

Crystal

**Feature-Driven
Development (FDD)**

**Dynamic
Systems
Development
Methodology
(DSDM)**

Lean

Scrum

Kanban

**Extreme
Programming (XP)**

The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The text "Extreme Programming (XP)" is centered in the middle of the slide.

Extreme Programming (XP)

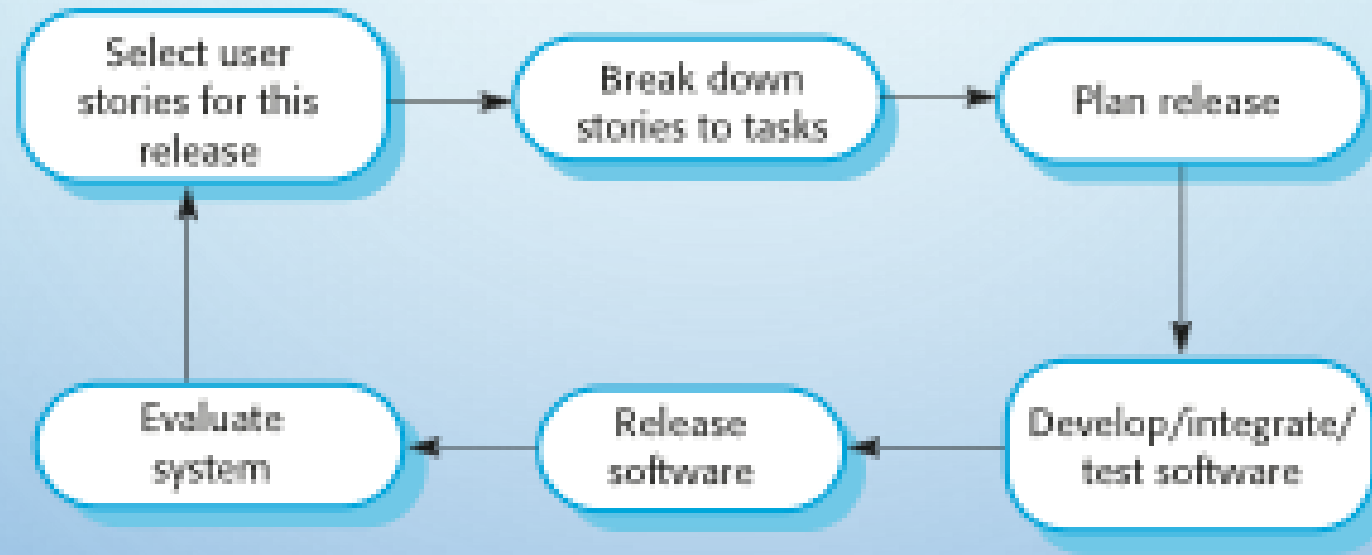
Extreme programming

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
 - New versions may be built several times per day;
 - Increments are delivered to customers every 2 weeks;
 - All tests must be run for every build and the build is only accepted if tests run successfully.

XP and agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

The extreme programming release cycle



Extreme programming practices (a)

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (b)

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and change

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it **proposes constant code improvement (refactoring) to make changes easier “increments”** when they have to be implemented.

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- **However**, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Testing in XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- **XP testing features:**
 - Test-first development.
 - User involvement in test development and validation.
 - Automated test harnesses are used to run all component tests each time that a new release is built.

Test-first development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
 - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

Customer involvement

- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution.

Test case description for dose checking

Test 4: Dose checking

Input:

1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:

1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:

OK or error message indicating that the dose is outside the safe range.

Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- It serves as an informal review process as each line of code is looked at by more than 1 person.

Pair programming

- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- This helps develop common ownership of code and spreads knowledge across the team.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

The background is a light blue gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The text is centered in the middle of the slide.

Software engineering Ethics

Software engineering ethics

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Ethical principles

1. **PUBLIC** - Software engineers shall act consistently with the public interest.
2. **CLIENT AND EMPLOYER** - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **PRODUCT** - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT** - Software engineers shall maintain integrity and independence in their professional judgment.
5. **MANAGEMENT** - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **PROFESSION** - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **COLLEAGUES** - Software engineers shall be fair to and supportive of their colleagues.
8. **SELF** - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Ethical dilemmas

- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.
- Disagreement in principle with the policies of senior management.

Thank you 😊