CSIT314 Software Development Methodologies

Software process models and ethics

This lecture …

- Software process models
  - Waterfall model
  - Prototyping model
  - Iterative/incremental model
  - Spiral model
  - Rational Unified Process
  - Agile methods

Acknowledgement: some materials are adapted from Chapter 2 - Ian Sommerville (2010), *Software Engineering,* 9th Edition, Addison-Wesley.

The software process

- A **structured set of activities** required to develop a software system

- Many different processes but they all involve:
  - Requirements specification- defining what the system should do;
  - Design & Implementation - defining the organization of the system and implementing the system
  - Verification & Validation (V & V) - checking that it conforms to the specification and does what the customer wants
  - Maintenance and Evolution - changing the system in response to changing customer needs.

- A software process model is an **abstract representation of a process.** It presents a description of a process from some particular perspective

Waterfall model

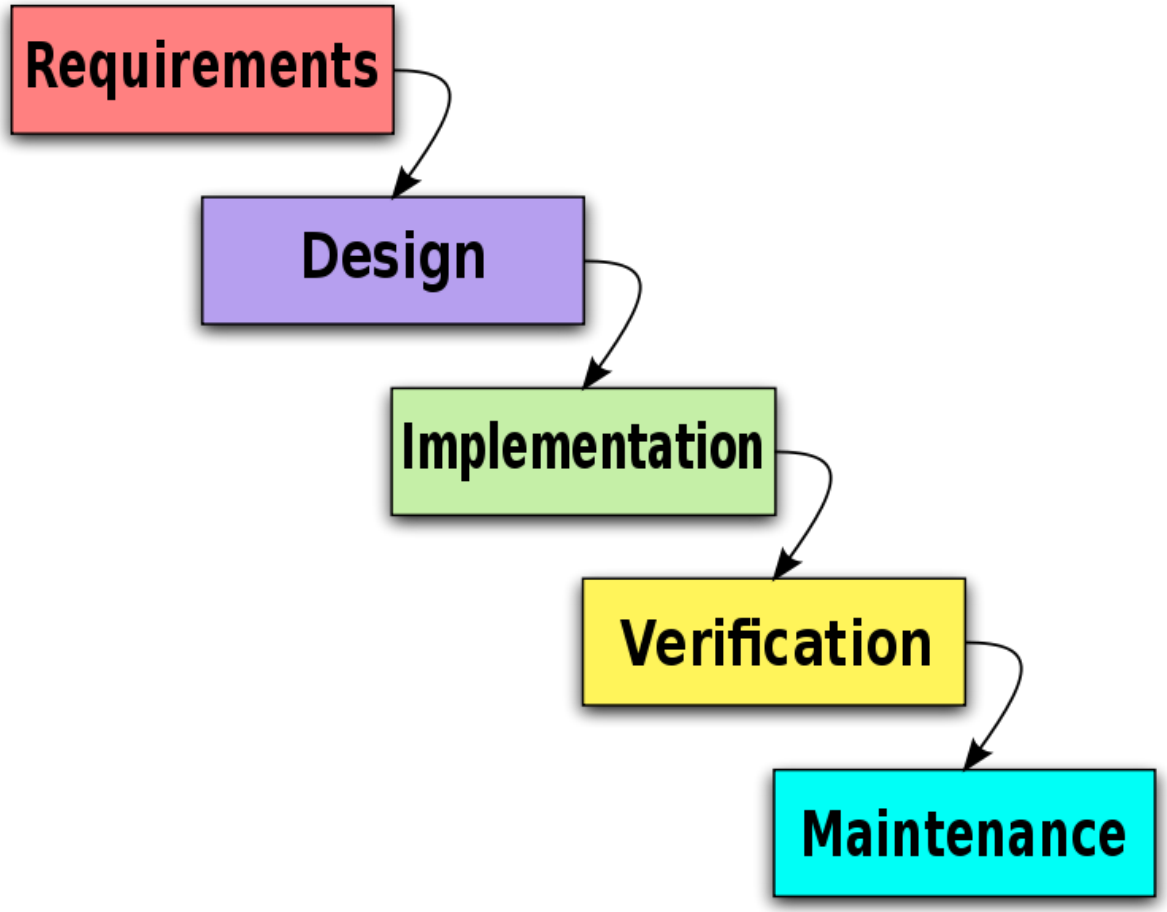
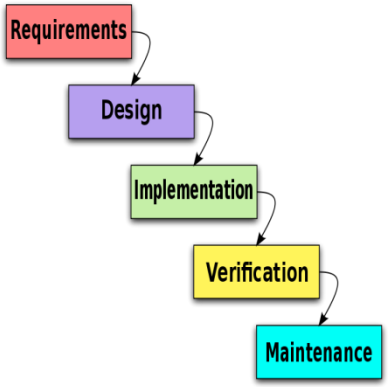


Image source: http://en.wikipedia.org/wiki/Waterfall_model

Waterfall model (cont.)



- The waterfall model is a sequential software development process.
  - a phase has to be *complete* and absolutely *correct* before moving onto the next phase.
- Progress flows from the top to the bottom, like a waterfall.
- Has its origins in the manufacturing and construction industries and adapted for software development since 1970s.
- First described for software development by **Winston W. Royce** in 1970

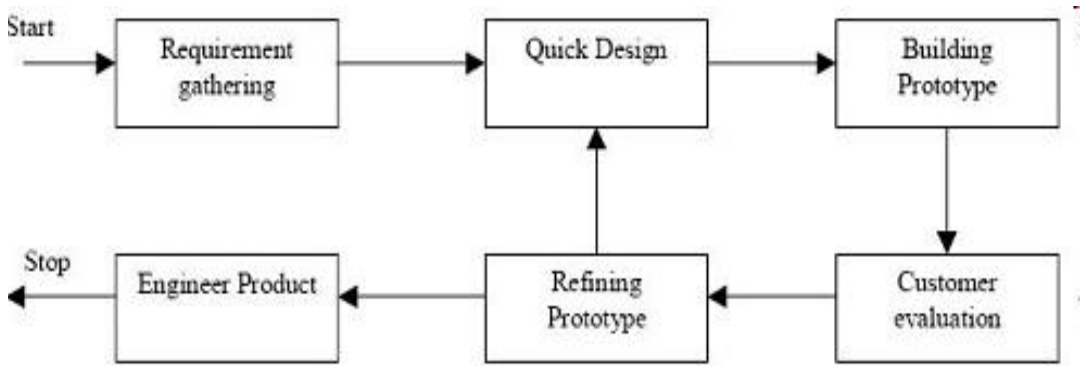
Waterfall model limitations

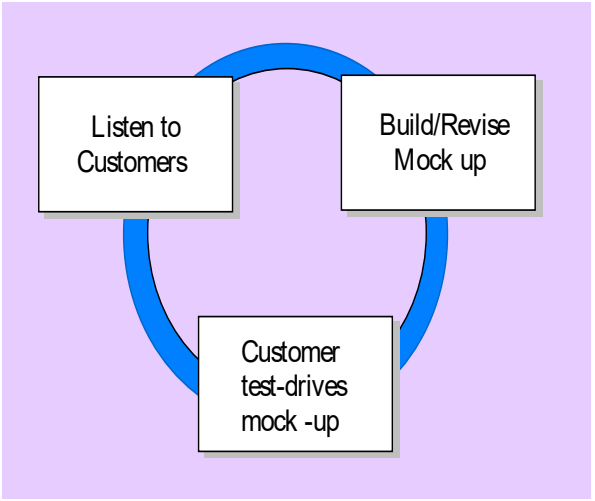
- The difficulty of responding to changing customer requirements.
- Designers will have to fully predict problem areas of the system and produce a *correct* design before implementation is started
- This model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.

The Rise and Fall of Waterfall

https://vimeo.com/18951935

Prototyping model





- gather requirements
- quick design focusing on what will be visible to user – input & output formats
- build a prototype, i.e. a working version of the system
- prototype evaluated and requirements refined

- process iterated until customer & developer satisfied
  - then throw away prototype and rebuild system to high quality

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;

Prototyping model
Benefits

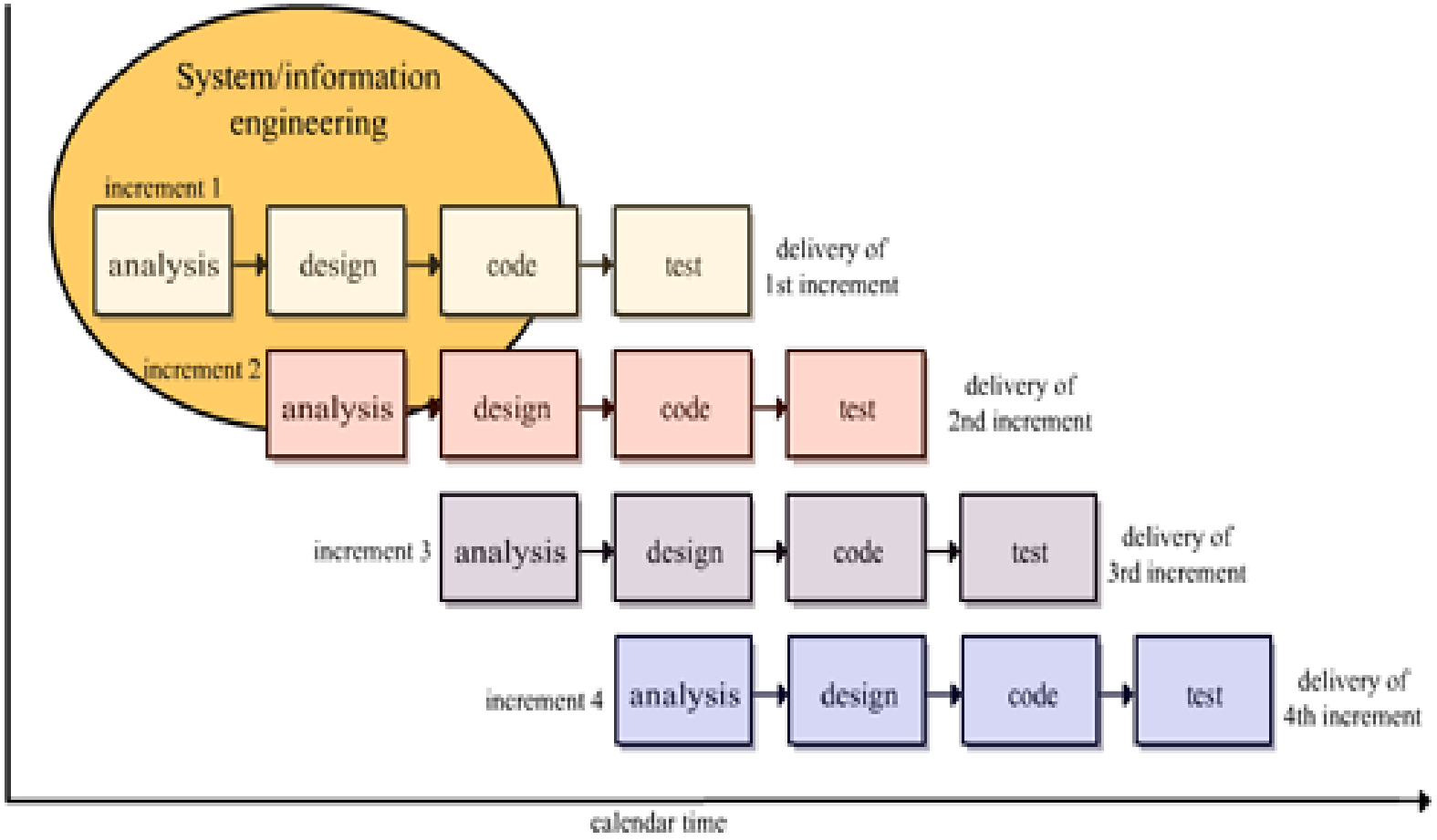
- Users are actively involved in the development
- Reduced time and costs
  - a working model of the system is provided, the users get a better understanding of the system being developed.
  - Prototyping can improve the quality of requirements and specifications provided to developers.
  - Quicker user feedback is available leading to better solutions.
- Best projects to use prototyping
  - Human-Computer Interface

Prototyping model Limitations

- User confusion of prototype and completed system
  - Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished.
- Developer attachment to prototype:
  - Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture
- Excessive development time of the prototype:
  - A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex
- Expense of implementing prototyping:
  - the start up costs for building a development team focused on prototyping may be high

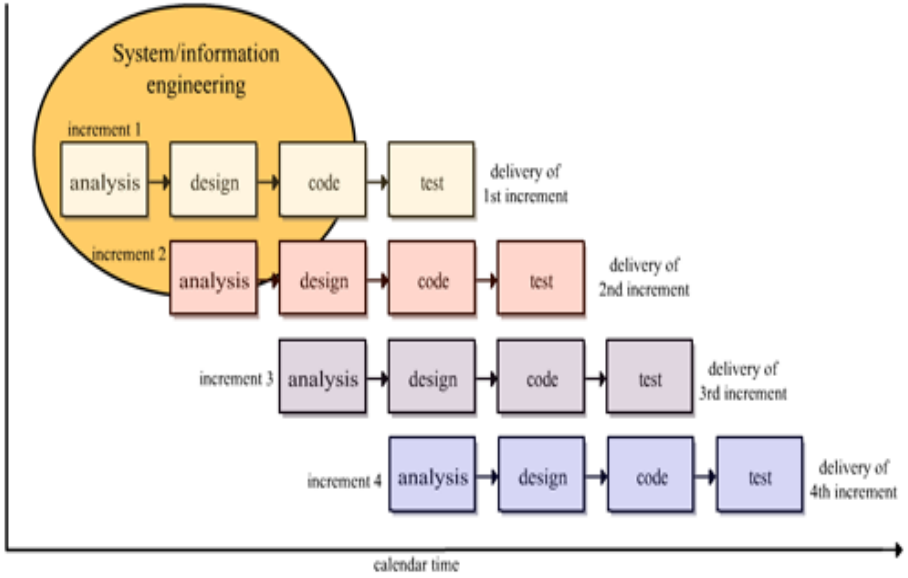
Source: http://en.wikipedia.org/wiki/Software_prototyping

Incremental and Iterative model

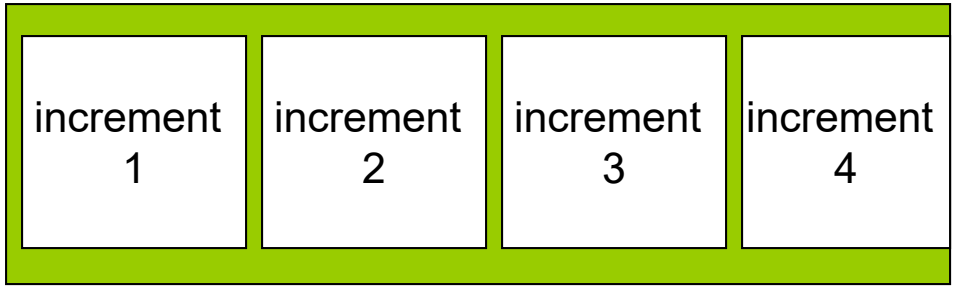


Incremental and Iterative model

- Develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- At each iteration, design modifications are made and new functional capabilities are added.



The software system



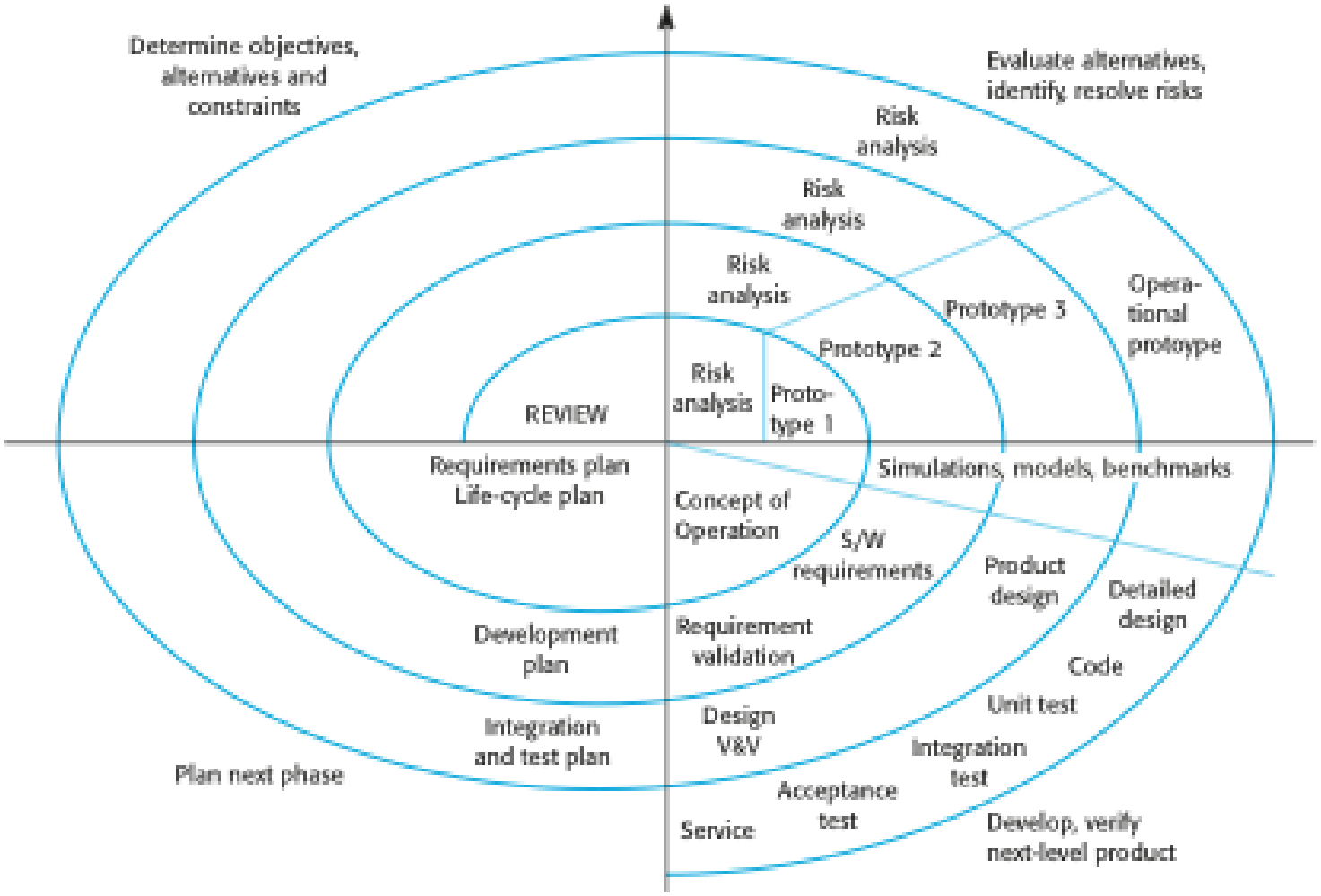
Incremental approach:benefits

- Feedback from early stages used in developing latter stages
- The cost of accommodating changing customer requirements is reduced.
- 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
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Possible disadvantages of incremental delivery

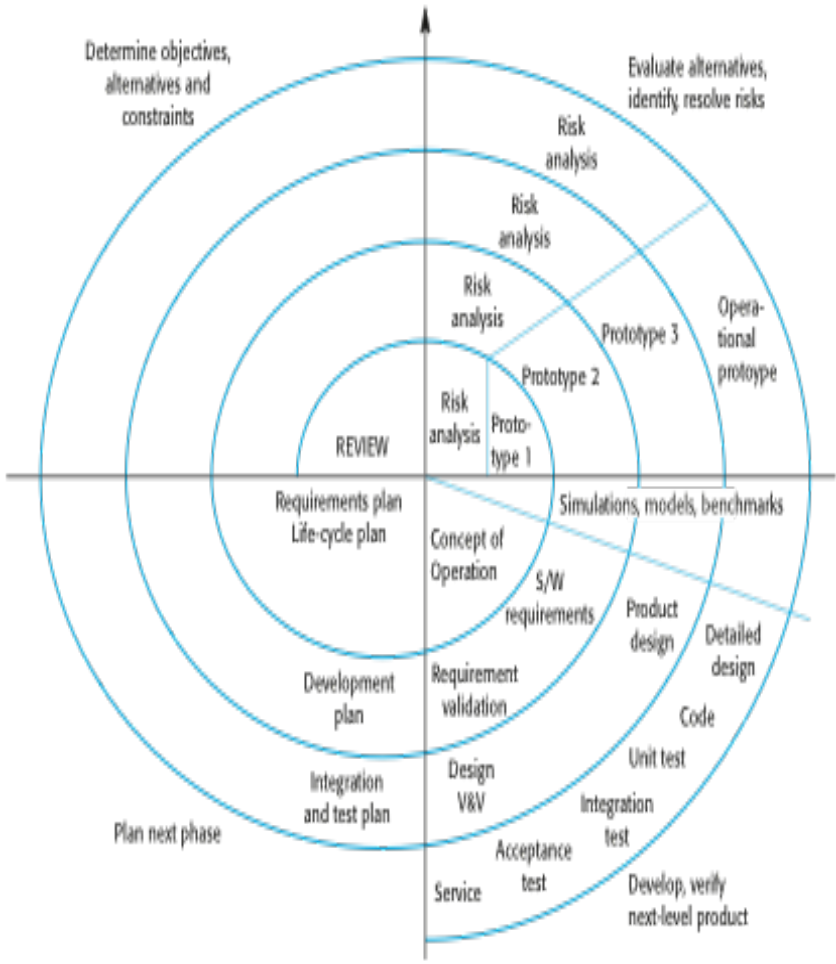
- Loss of economy of scale
  - some costs will be repeated
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure.
  - Incorporating further software changes becomes increasingly difficult and costly.

The Spiral Model



The Spiral Model

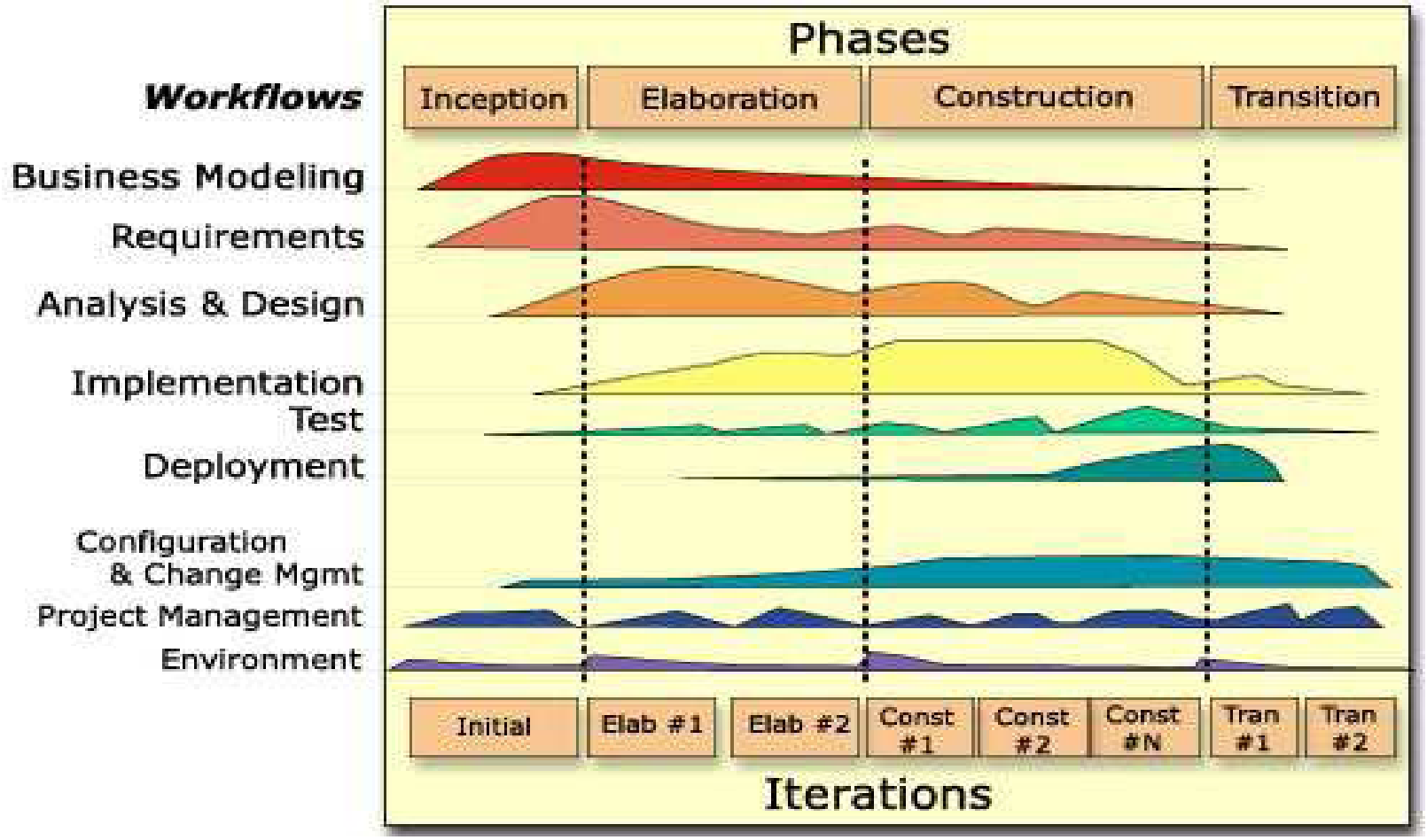
- A combination of prototyping with iterative model
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- Risks are explicitly assessed and resolved throughout the process.



Spiral model usage

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- The spiral model is mostly used in large projects.
  - Game development is a main area where the spiral model is used and needed
  - Military projects, e.g. Future Combat Systems program.
    - it had a 2 year iteration (spiral).

The Rational Unified Process



Note: We will later explore Unified Process in depth.

Agile methods

- In the 1990s, the IT industry faces substantial frustration:
  - Most project followed the waterfall model
  - Big time gap between business requirements and the delivery of software that answered those needs => cancelling many projects.
  - Business environments and requirements quickly changed and the final product did not meet the then current needs.
  - The waterfall model did not take advantage of how quickly software could be altered.

Agile methods (cont.)

- Agile refers to a set of methods and practices under which:
  - Requirements and solutions evolve through the **collaborative effort** of self-organizing cross-functional teams and their customers and users.
  - adaptive planning, evolutionary development, early delivery, and continuous improvement are advocated.
  - Rapid and flexible response to change is encouraged
- Agile methodologies are mostly based on the *values* and *principles* expressed in the Agile Manifesto.

Check out this video https://www.youtube.com/watch?v=oHBs4wxiSpQ

Agile principles

1. The **highest** priority is to satisfy the customer through **early** and **continuous delivery** of valuable software.
2. Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the **shorter** timescale.

Source: www.agilemanifesto.org

Agile principles (cont.)

4. **Business** people and developers must **work together daily** throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and **support** they need, and **trust** them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation.

Source: www.agilemanifesto.org

Agile principles (cont.)

7. **Working software** is the primary measure of progress.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to **technical excellence and good design** enhances agility.

Source: www.agilemanifesto.org

Agile principles (cont.)

10. **Simplicity**--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing** teams.
12. At regular intervals, the team **reflects** on how to become more effective, then **tunes** and **adjusts** its behaviour accordingly.

Source: www.agilemanifesto.org

Agile software development methods

- **Scrum**
- **Extreme Programming**
- Dynamic Systems Development Methods
- **Kanban**
- Lean software development
- Etc.

Scrum


24 hours
Sprint
2-4 weeks
Sprint goal
Return
Sprint
backlog
Potentially shippable
product increment
Cancel
Coupons
Gift wrap
Coupons
Product
backlog

Sprints

- Scrum projects make progress in a series of “sprints” (i.e. iterations)
- Typical duration is 2–4 weeks or a calendar month at most
- Product is designed, coded, and tested during the sprint

Product backlog

This is the product backlog

- A typical Scrum backlog comprises the following different types of items:
  - Features
  - Bugs
  - Technical work (e.g. “Upgrade all developers' workstations to Windows 7”)
  - Knowledge acquisition (e.g. “researching various JavaScript libraries and making a selection.”
- **Prioritized** by the product owner
- Reprioritized at the start of each sprint

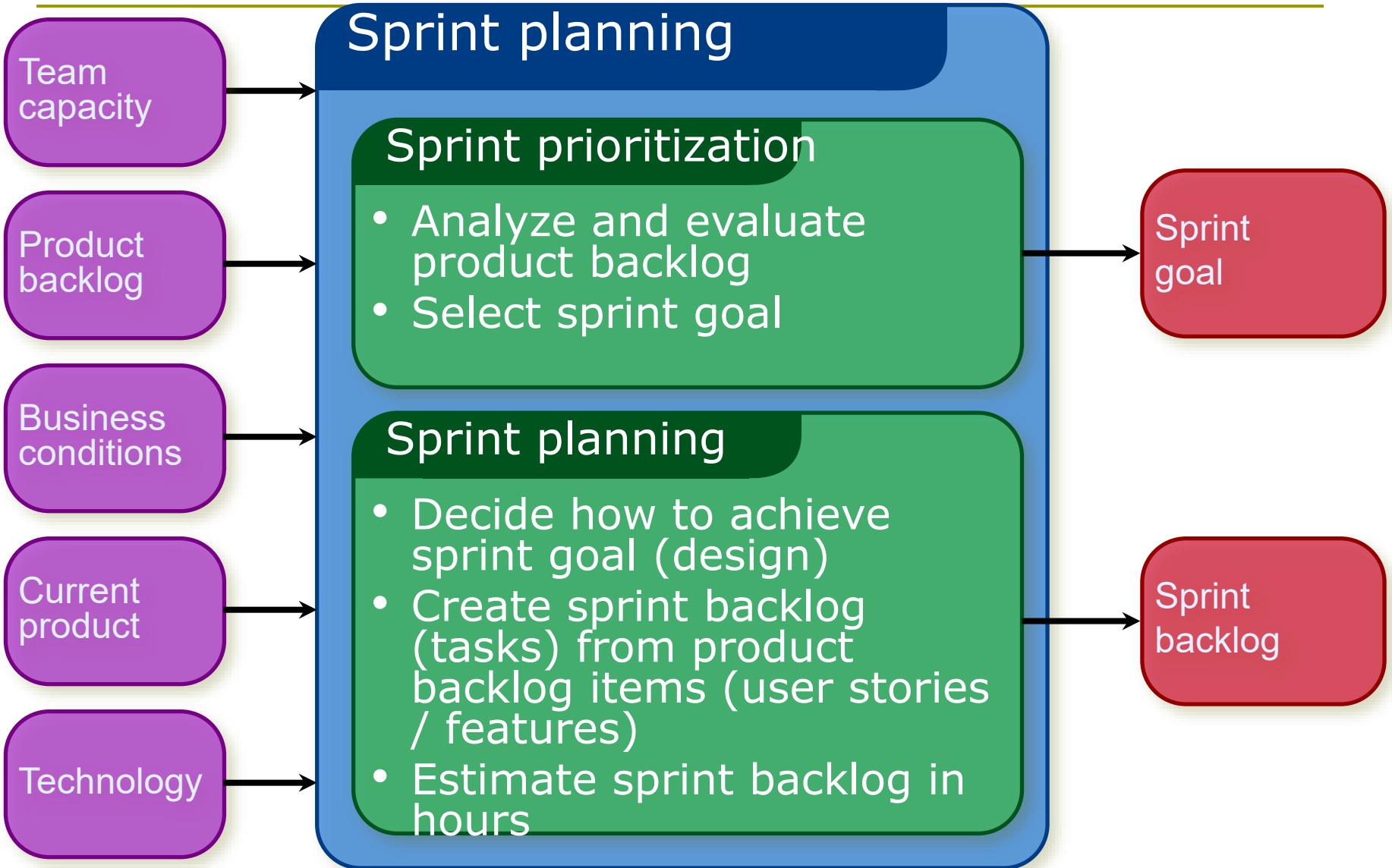
User stories

- **User stories** are short, simple descriptions of a feature told from the perspective of a user or customer of the system who desires the new capability.

- User stories typically follow a simple template:
  - *As a < type of user >, I want < some goal > so that < some reason >.*

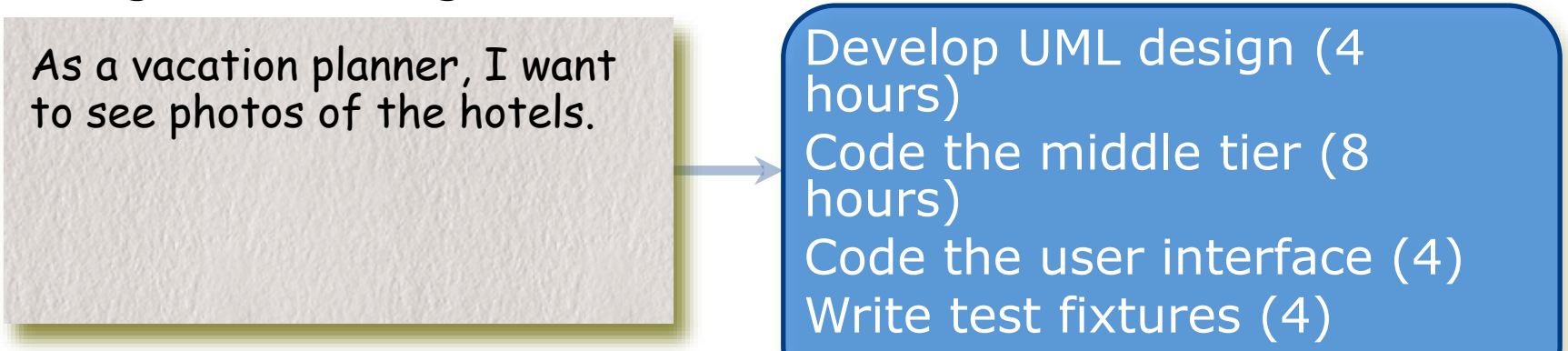
- Examples:
  - As a site visitor, I can read current news on the home page.
  - As a trainer, I can create a new course or event. This includes the following information: name, description (HTML), trainer names (multiple selection from a list), start date, end date, venue name (HTML) and address, contact name, contact phone, contact email, a link for more information, and a link to register. For a certification course the name of the class is a dropdown list; for others, it is free text.

Sprint planning



Sprint planning

- Team decides on the **goal** for a particular sprint, and then selects items from the product backlog they commit to complete to achieve the **sprint's goal**.
- Sprint backlog is created
  - Tasks are identified and each is estimated (e.g. 1-16 hours)
  - Collaboratively, not done alone by the ScrumMaster
- High-level design is considered



User story

Tasks (Example only)

Note: effort estimation can be in **story points**.

Scrum Team

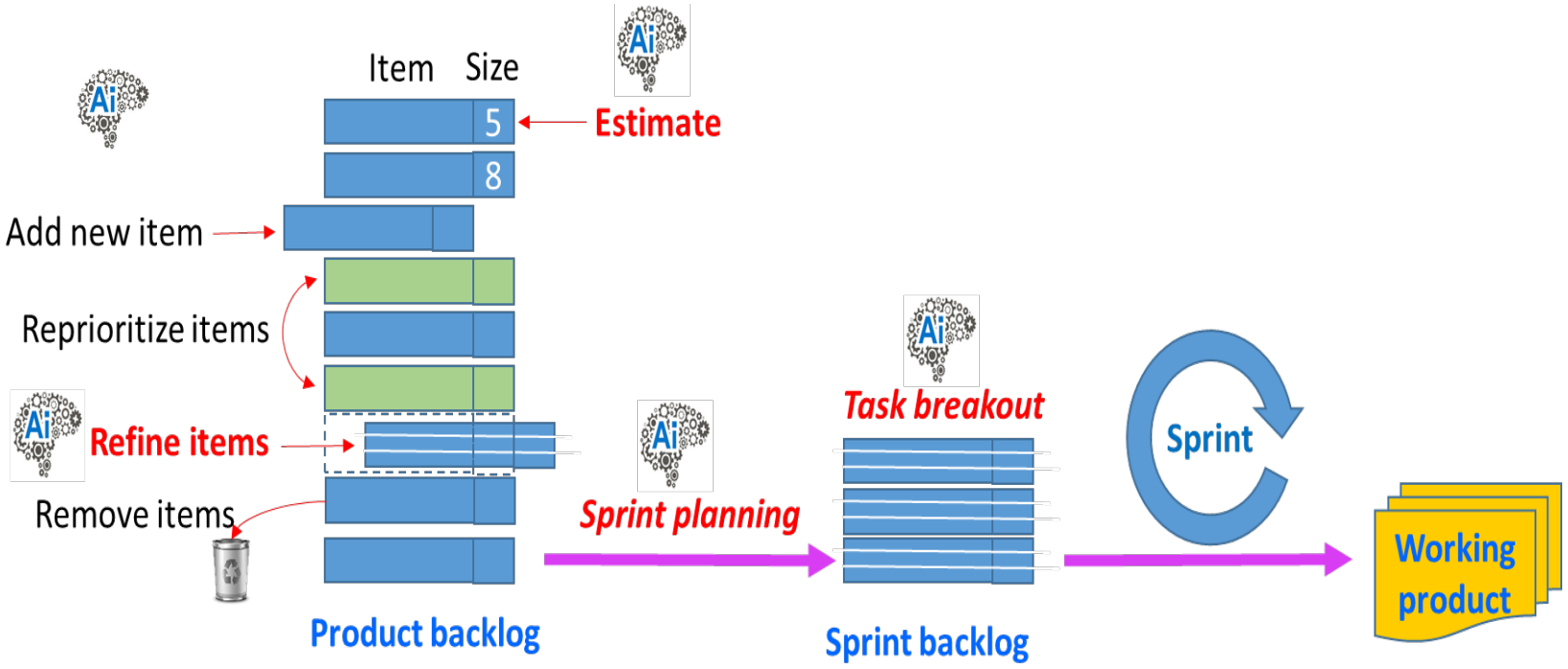
- Product owner:
  - Define the **features** of the product
  - Decide on **release date and content**
  - **Prioritize** features according to market value
  - Adjust features and priority every iteration, as needed
  - Accept or reject work results
- Scrum Master
  - Represents management to the project, ensure that the team is fully functional and productive.
  - Responsible for enacting Scrum values and practices
- The team
  - Self-organizing and cross-functional: Programmers, testers, user experience designers, etc.

Scrum meetings

- Sprint planning meeting:
  - https://youtu.be/2A9rkiIcnVI
  - https://youtu.be/GivcWpDRlD4
- Daily Scrum meeting:
  - https://www.youtube.com/watch?v=GzQjGhD5tSU
- Sprint review meeting
  - https://youtu.be/cbJinz6TieI
- Sprint retrospective meeting
  - https://youtu.be/n_iu8kuA0XE

AI for Agile Software Development

- Read these articles:
  - https://techxplore.com/news/2019-01-framework-ai-powered-agile.html
  - https://blog.developer.atlassian.com/artificial-intelligence-for-issue-analytics-a-machine-learning-powered-jira-cloud-app/



What is ethics?

- Dictionary definition
  - *"moral principles that govern a person's behaviour or the conducting of an activity."*

  See this video https://ethicsunwrapped.utexas.edu/glossary/ethics

- In some views, ethics has 3 components:
  - **Values**: good things we strive for, desire and seek to protect.
  - **Principles**: outlining how we may or may not achieve our values (telling us what's right or wrong).
  - **Purpose**: our reason for upholding our values and principles.

(Source: https://ethics.org.au)

Why are ethics important in software development?

- Software are everywhere in people's lives, across almost all industry sectors (e.g. business, government, medicine, education, transport, agriculture, manufacture, etc.) in our society.

- Software can do good or cause harm
  - **Question: What are software apps that can do good or cause harm?**

- Any software development activities (requirements, design, implementation, testing and maintenance) can lead to those consequences.

Software and its consequences

- See this video https://youtu.be/CQ4irwe3ZDk about Volkswagen emissions scandal and answer the following questions:
  - What was the scandal?
    - "software-based defeat device"
  - Why was it an ethical issue?

See more at https://www.bbc.com/news/business-34324772

ACM Code of Ethics for Software Development

- **Principle 1: PUBLIC** - *act consistently with the public interest*, e.g.:
  - Accept **full responsibility** for their own work.
  - Approve software only if they have a well-founded belief that it is **safe**, **meets specifications**, **passes appropriate tests**, and does not diminish quality of life, diminish privacy or harm the environment.
  - **Disclose** any actual or potential **danger** to the user, the public, or the environment.
  - Consider issues of **physical disabilities**, allocation of resources, economic disadvantage and other factors that can diminish access to the benefits of software.

ACM Code of Ethics for Software Development

- **Principle 2: CLIENT AND EMPLOYER** - *act in a manner that is in the best interests of their client and employer, consistent with the public interest*, e.g.:
  - Provide service in their areas of **competence**, being honest and forthright about any **limitations** of their experience and education.
  - Not knowingly use software that is obtained or retained either illegally or unethically.
  - Keep private any **confidential information** gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law.

ACM Code of Ethics for Software Development

- **Principle 3: PRODUCT -** *ensure that their products and related modifications meet the highest professional standards possible, e.g.:*
  - Strive for **high quality**, **acceptable cost** and a **reasonable schedule**, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.
  - Identify, define and address **ethical**, economic, cultural, legal and environmental issues related to work projects.
  - Ensure **adequate testing**, **debugging**, and **review** of software and related documents on which they work.

ACM Code of Ethics for Software Development

- **Principle 4: JUDGMENT –** *maintain integrity and independence in their professional judgment*, e.g.:
  - Only **endorse** documents either prepared under their supervision or within their areas of competence and with which they are in agreement.
  - Maintain **professional objectivity** with respect to any software or related documents they are asked to evaluate.
  - **Disclose** to all concerned parties those **conflicts of interest** that cannot reasonably be avoided or escaped.

ACM Code of Ethics for Software Development

- **Principle 5: MANAGEMENT -** subscribe to and promote an ethical approach to the management of software development and maintenance, e.g.:
  - Ensure **good management** for any project on which they work, including effective procedures for promotion of quality and reduction of risk.
  - Ensure that there is a **fair agreement** concerning **ownership** of any software, processes, research, writing, or other **intellectual property** to which a software engineer has contributed.

ACM Code of Ethics for Software Development

- **Principle 6: PROFESSION -** advance the integrity and reputation of the profession consistent with the public interest, e.g.:
  - Take **responsibility** for **detecting**, **correcting**, and **reporting** errors in software and associated documents on which they work.

ACM Code of Ethics for Software Development

- **Principle 7: COLLEAGUES -** be fair to and supportive of their colleagues, e.g.:
  - **Credit** fully the work of others and refrain from taking undue credit.
  - **Review** the work of others in an objective, candid, and properly-documented way.

ACM Code of Ethics for Software Development

- **Principle 8: SELF -** shall participate in **lifelong learning** regarding the practice of their profession and shall promote an ethical approach to the practice of the profession, e.g.:
  - **Further their knowledge** of developments in the analysis, specification, design, development, maintenance and testing of software and related documents, together with the management of the development process.
  - **Improve their ability** to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.