SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

ARTIFICIAL INTELLIGENCE IN AGILE SYSTEMS

SYLLABUS

UNIT - I

Introduction to AI techniques, Intelligent Agents, Problem Solving with AI, Intelligent Agents, Structure of Agents, Agile Alliance, Principles of Agile Practices, Practices of Extreme Programming, Planning –Initial Exploration, Release Planning , Iteration planning , Task planning, Challenges in Traditional Systems, Real time applications of AI in Agile systems, Test driven development, Acceptance Test, Serendipitious Architecture, Serendipitious decoupling

SYLLABUS

UNIT - II

Symptoms of poor design , Principles of a good design, Types of design patterns, Strategy design Pattern, Real time Applications of Strategy design Pattern, Mediator design pattern, Real time Applications of Mediator design pattern, Singleton design pattern, Real time Applications of Singleton design pattern, Factory design pattern, Real time Applications of Factory design pattern, Façade design Pattern, Case study: Identify which pattern is applicable for the given case study and justify, Case Study : U.S. Healthcare analytics - for the payer market, Case Study : ML to identify anomalies in claims adjudication and payments system

SYLLABUS

UNIT - III

Agile Approach, AI process for business today, Agile design Example, Agile approach in AI, Organizing Agile for AI Data Scientist, Organizing Agile for AI Data Engineers, Organizing Agile for AIBusiness Analysts, Need for Agile in AI, Contrasting Machine Learning and AI, Advantages of Agile AI, Agile Framework for AI Projects, Case study: Identify which pattern is applicable for the given case study and justify, Case Study : Prediction of disease patterns and proactive care, Case Study : Identify fraud waste abuse using patterns in claims

SYLLABUS

UNIT - IV

Relevance feedback: Rocchio algorithm, Probabilistic relevance feedback, Probability ranking principle, Binary Independence Model, Bayesian network for text retrieval, Evaluation of relevance feedback strategies, Pseudo relevance and indirect relevance feedback, Query Expansion and its types, Query drift, Global methods for Query reformulation

SYLLABUS

UNIT - V

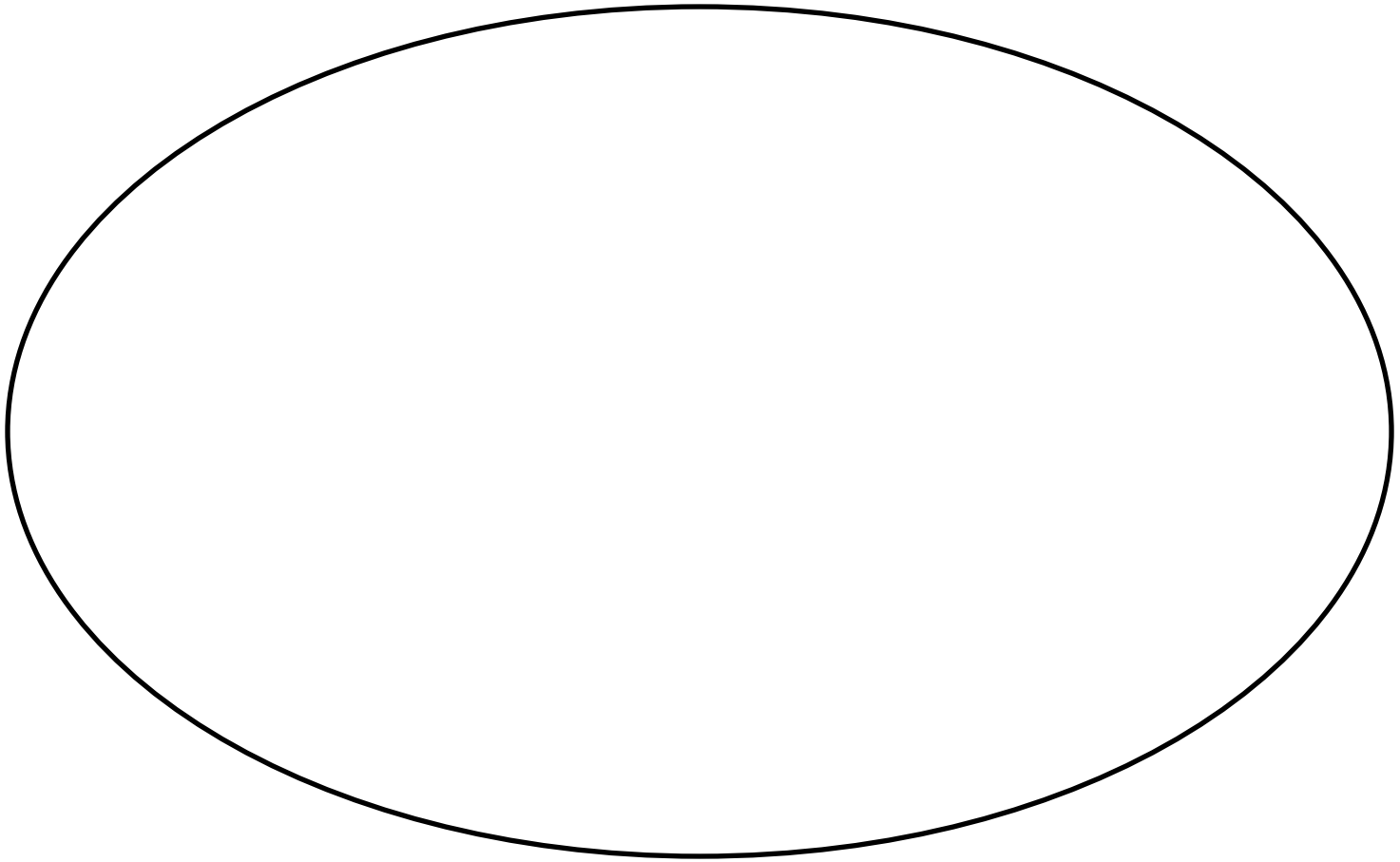
Learning from Real-Time, Big Data, Applications of AI in health care, Realizing the Potential of AI in Healthcare, Evolution of Data and Its Analytics, Real time Challenges of Big Data, Impact of Data in Future, Ethics of Artificial Intelligence and Machine Learning, Prediction Ethics, Preventing Algorithms from Becoming Immoral, Real time applications of Agile systems, AI and Agile systems in health care, Future of Health care

UNIT - I
INTRODUCTION

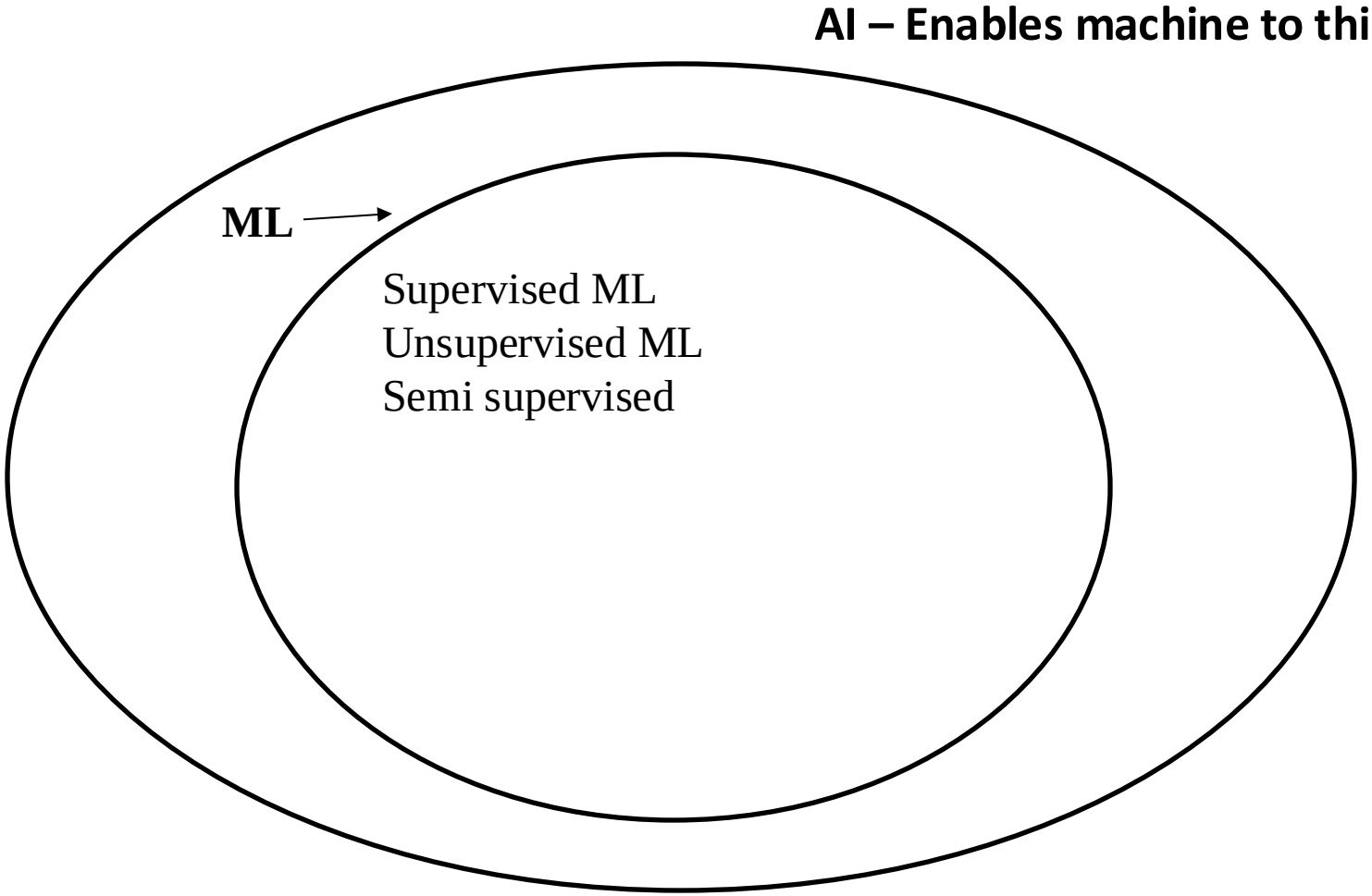
- Artificial intelligence (AI) is the ability of machines to replicate or enhance human intellect, such as reasoning and learning from experience

AI vs ML vs DL vs DS

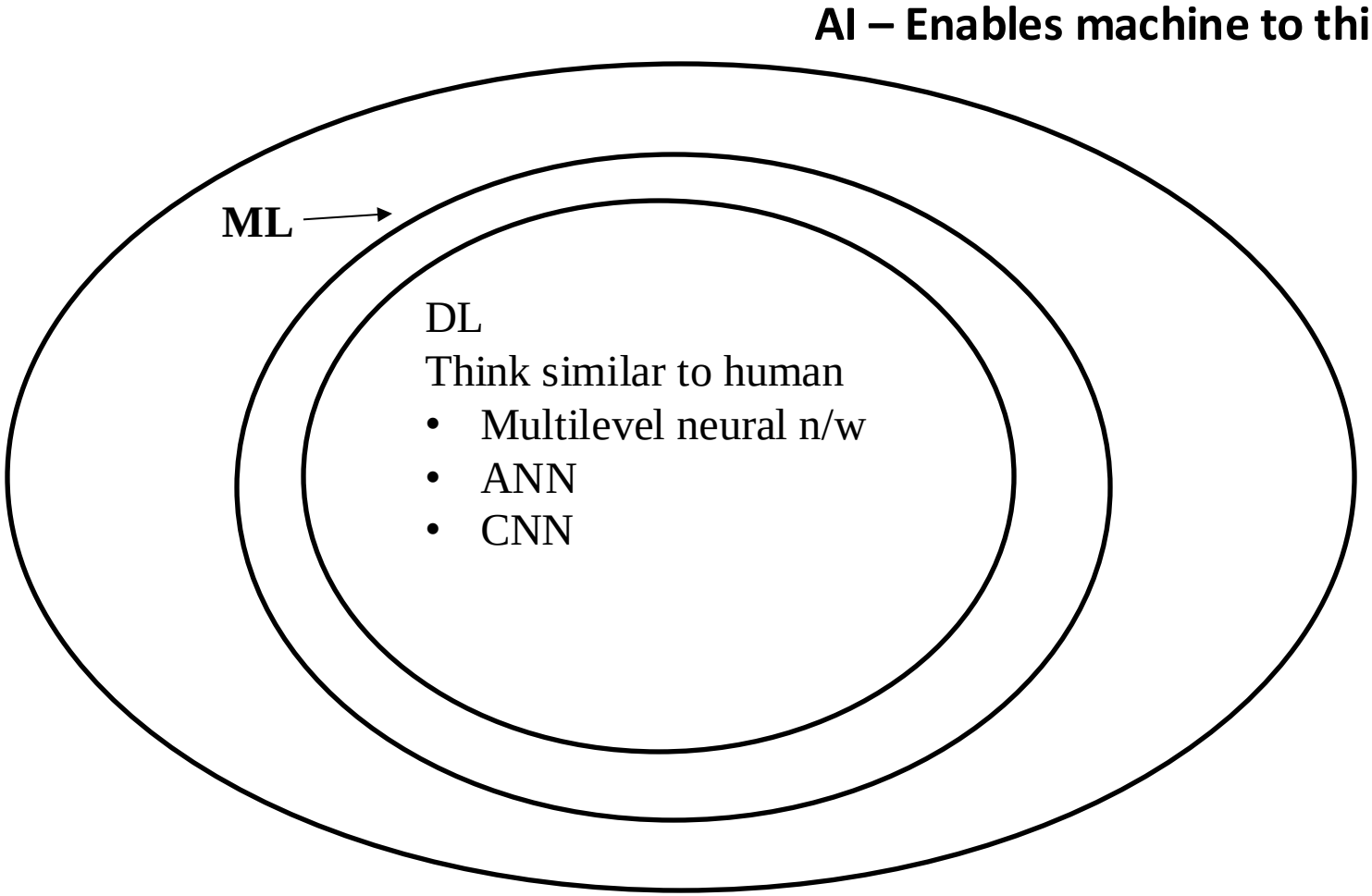
AI – Enables machine to think



AI vs ML vs DL vs DS

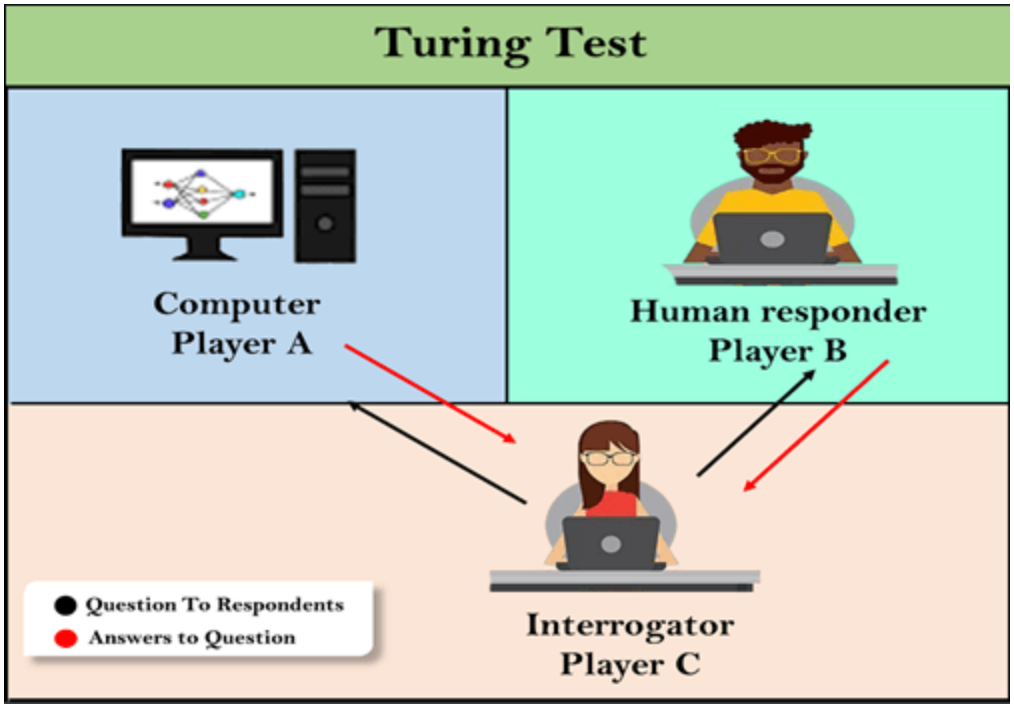


AI vs ML vs DL vs DS



History of Artificial Intelligence

- 1943 - Warren McCulloch and Walter pits proposed a model of artificial neurons
- 1950 – Turing Test by Alen turing



History of Artificial Intelligence (Contd..)

- Cloud Shanon – information theory for possibility of chess playing on computers
- 1956 -- John McCarthy at the Dartmouth Conference. AI coined as an academic field
- 1966 – Researches emphasized developing algorithms which can solve mathematical problems. First chatbot named Eliza developed
- Heuristic search – A* , AO* , game tree search
- Knowledge Emphasis (1966 -1974)
- Uses procedural and declarative knowledge

History of Artificial Intelligence (Contd..)

- Knowledge based system (1969 -1999)
- 1997 – IBM deep blue beats
- 2002 – AI in home Appliances
- 2006 – AI in business
- 2011 – IBM introduced NLP
- 2012 – Google now –Google Android app
- 2014 – Chatbot – Euume Goost man
- 2018 – IBM Computer debate with two master debaters

Advantages of AI

- **High Accuracy with less errors**
- **High-Speed**
- **High reliability**
- **Useful for risky areas**
- **Digital Assistant**
- **Useful as a public utility**

Disadvantages of AI

- **High Cost**
- **Can't think out of the box**
- **No feelings and emotions**
- **Increase dependency on machines**
- **No Original Creativity**

AI Techniques

- It is an ability to design smart machines or to develop software applications that can self-learn and imitate the traits of the human mind with the help of reasoning, sensory applications, planning, optimal decision making and problem-solving techniques.

AI Techniques

- Heuristics
- Support Vector Machines
- Artificial Neural Networks
- Markov Decision Process
- Natural Language Processing

AI Techniques -Heuristics

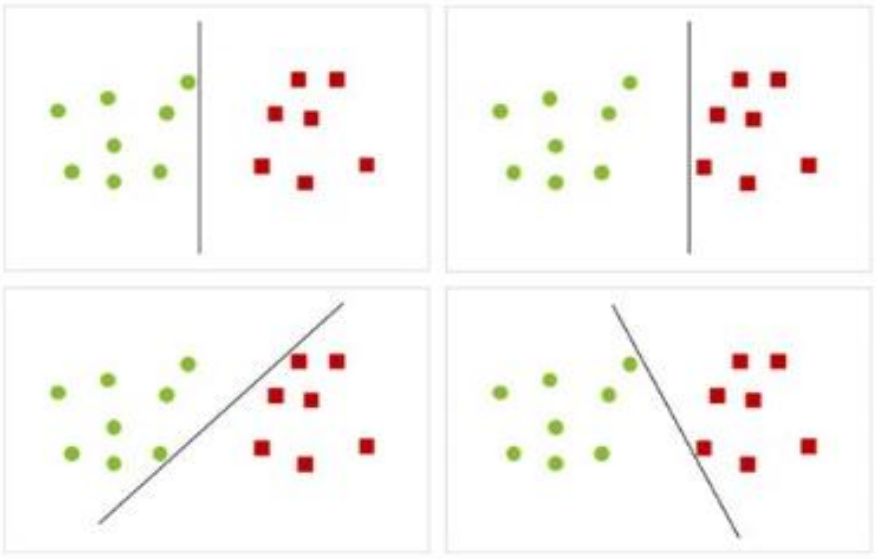
- Suppose we have coins with the following denominations: 5 cents, 4 cents, 3 cents, and 1 cent, and that we need to determine the minimum number of coins to create the amount of 7 cents.
- In order to solve this problem we can use a technique called “Heuristics”

AI Techniques - SVM

- The question whether an email is spam or not is an example of a classification problem.
- In these types of problems, the objective is to determine whether a given data point belongs to a certain class or not.
- After first training a classifier model on data points for which the class is known (e.g. a set of emails that are labeled as spam or not spam), you can then use the model to determine the class of new, unseen data-points.
- A powerful technique for these types of problems is Support Vector Machines4 (SVM)

AI Techniques - SVM

- In this example, the green circles and the red squares could represent two different segments in a total set of customers (e.g. high potentials and low potentials), based on all kinds of properties for each of the customers



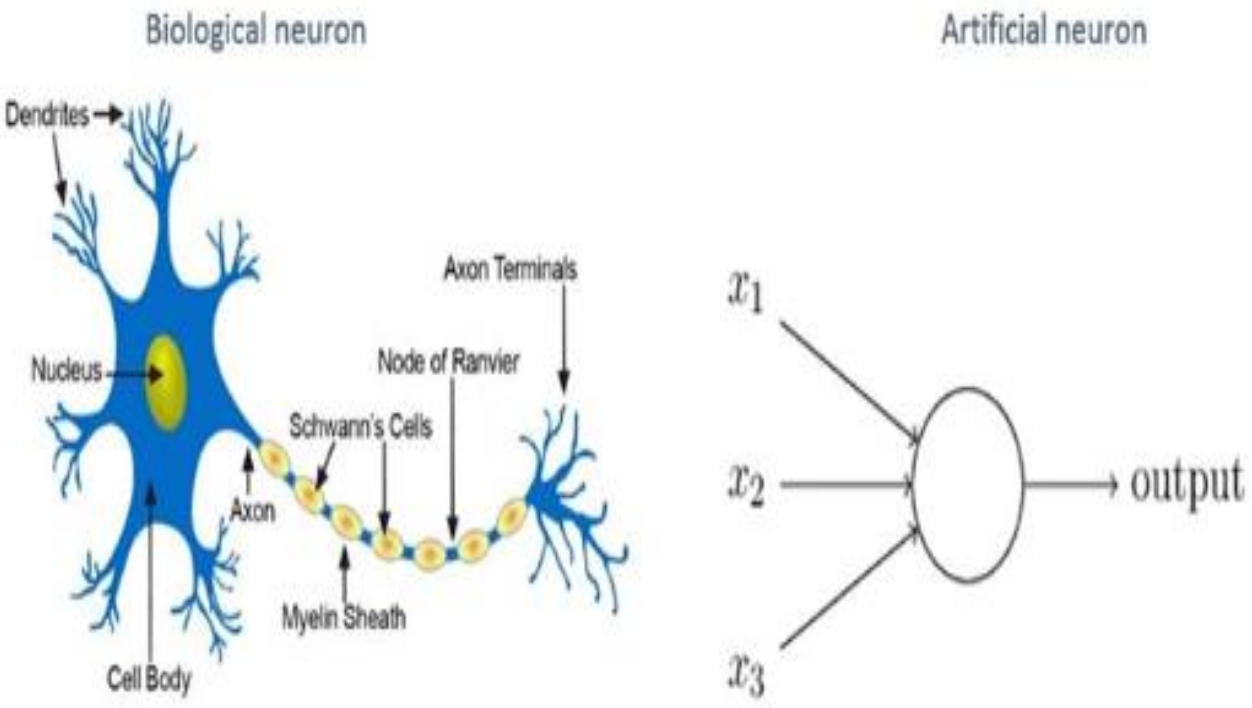
Artificial Neural Networks

- Animals are able to process (visual or other) information from their environment and adapt to change.
- They use their nervous system to perform such behavior.
- Their nervous system can be modeled and simulated and it should be possible to (re)produce similar behavior in artificial systems.

Artificial Neural Networks

- Artificial Neural Networks (ANN) can be described as processing devices that are loosely modeled after the neural structure of a brain.
- The biggest difference between the two is that the ANN might have hundreds or thousands of neurons, whereas the neural structure of an animal or human brain has billions

Artificial Neural Networks



Markov Decision Process

- A Markov Decision Process (MDP) is a framework for decision-making modeling where in some situations the outcome is partly random and partly based on the input of the decision maker.
- Another application where MDP is used is optimized planning.
- The basic goal of MDP is to find a policy for the decision maker, indicating what particular action should be taken at what state.

Markov Decision Process

- An MDP model consists of the following parts
- A set of possible states
- A set of possible actions
- Transition probabilities
- Rewards

Natural Language Processing

- Natural Language Processing (NLP) is used to refer to everything from speech recognition to language generation, each requiring different techniques.
- A few of the important techniques will be explained below, i.e. Part-of-Speech tagging, Named Entity Recognition, and Parsing.

Agents

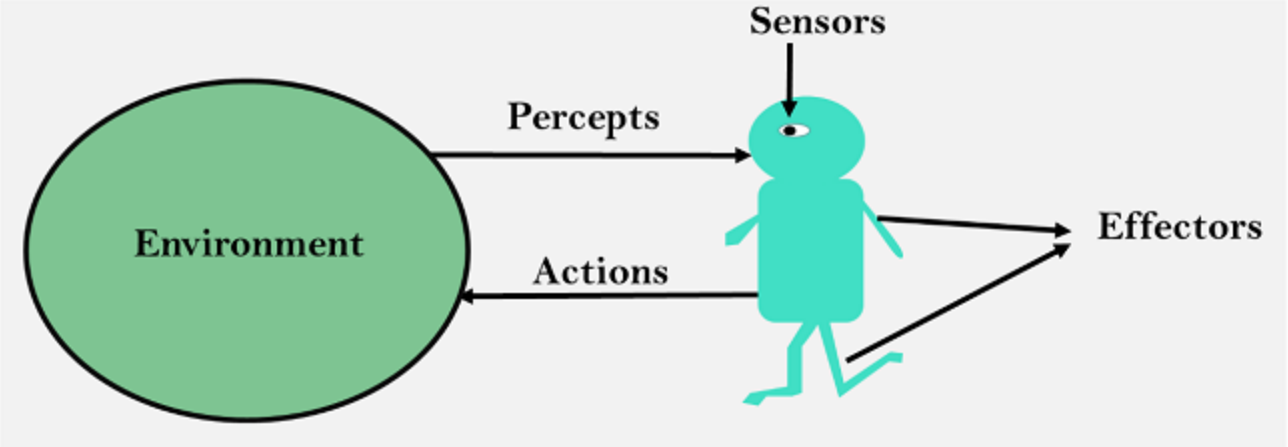
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

- Human agent:
  - eyes, ears, and other organs for sensors;
  - hands, legs, mouth, and other body parts for actuators

- Robotic agent:
  - cameras and infrared range finders for sensors
  - various motors for actuators

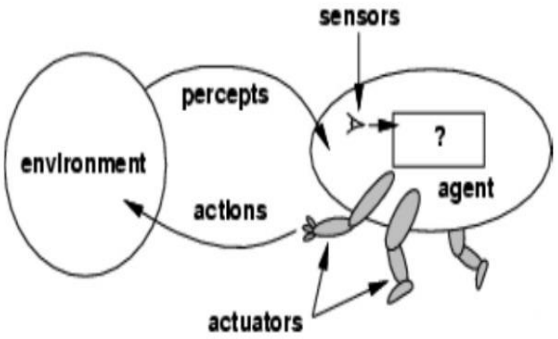
Agent - Components

- The agents sense the environment through sensors and act on their environment through actuators



- Sensors
- Actuators
- Effectors

Agents and environments

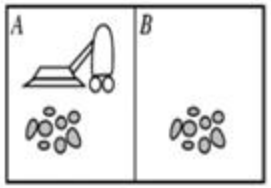


- The agent function maps from percept histories to actions:

$$[f: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

- The agent program runs on the physical architecture to produce *f*
- agent = architecture + program

Vacuum-cleaner world



- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left*, *Right*, *Suck*, *NoOp*
- *Agent's function → look-up table*
  - *For many agents this is a very large table*

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

Stuart J. Russell, Peter Norwig , Artificial Intelligence –A Modern approach

AI Representation - Vacuum Cleaner

Performance: Cleanness,Efficiency,Battery life,Security

Environment: Room, Table, Wood floor, Carpet

Actuators: Wheels, Brushes, Vacuum Extractor

Sensors: Camera, Dirt detection sensor, Bump Sensor, Infrared Wall Sensor

AI Representation - Self driving cars

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer

Intelligent Agent

- Entity that works without assistance, interprets inputs, senses the environment, make choices and acts to achieve the goal

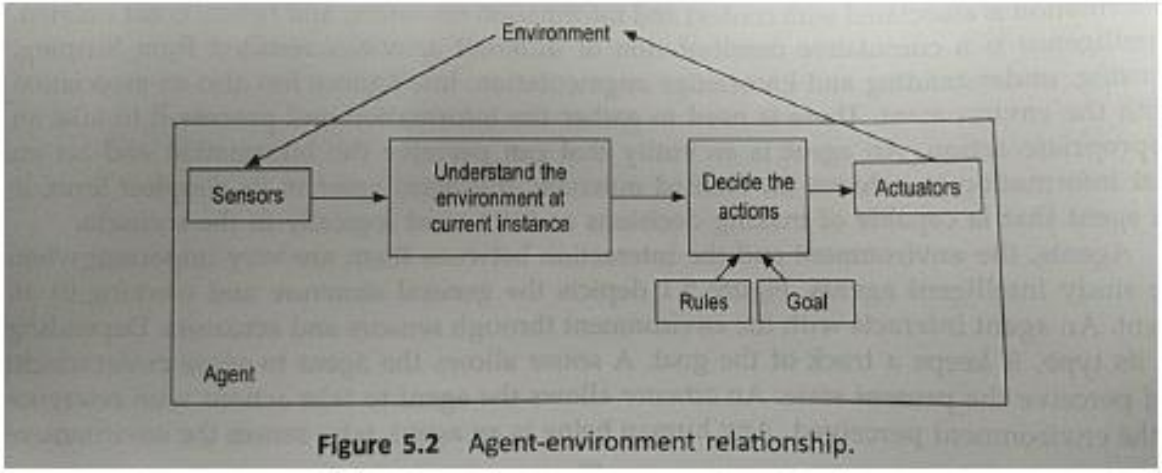


Figure 5.2 Agent-environment relationship.

- Decisions are based on Set of Rules (if-then)
- Intelligent autonomous in Nature, Good Observant to the environment

Intelligent Agents

- Perception is the process of acquiring, interpreting, selecting, and organizing sensory information
- Percept Sequence - Complete history of everything
- Agent Function is generally implemented by an internal program called agent program
- Mapping of Agent Function is

$$P \rightarrow A$$

- Agent consists of architecture and program

Sensors → Agent Programs → Actuators

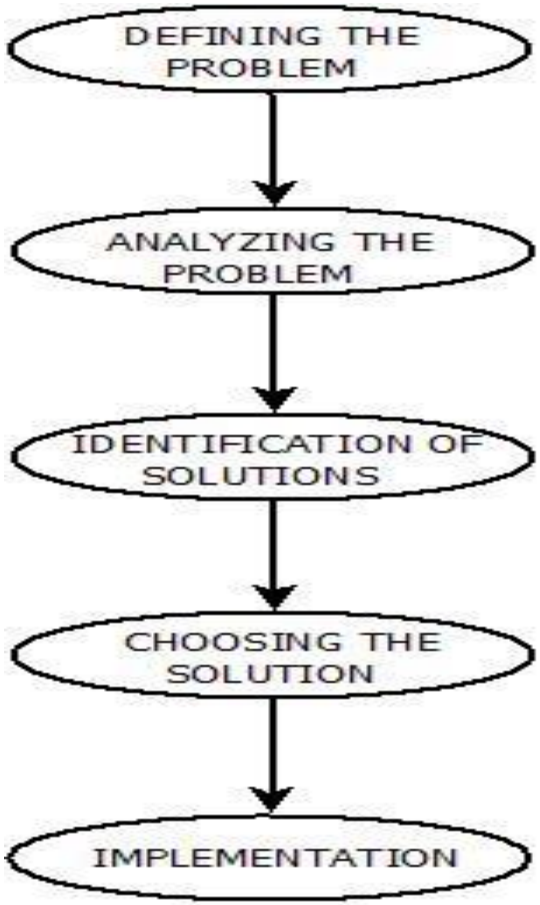
SRM
INSTITUTE OF SCIENCE & TECHNOLOGY

Flexibility and Intelligent Agent

- Intelligence demands flexibility and agents do need that
- Agents need to be
  - Responsive (Timely fashion reaction)
  - Pro-active (Opportunistic and goal-directed behaviour)
  - Social (Interaction with other agents)
  - Mobility (to accumulate the knowledge)
  - Veracity (Truthful)
  - Benevolence (Avoiding conflict)
  - Rationality (Act to maximize the expected performance)
  - Learning (Increase in learning increases the performances)
- For designing an intelligent agent, PEAS (P - Performance, E - Environment, A - Agent's actuators, S - Sensors) are necessary

Problem Solving In AI

- **Problem**
- **Problems are the issues which comes across any system. A solution is needed to solve that particular problem.**
- **Steps : Solve Problem Using Artificial Intelligence**
- **The process of solving a problem consists of five steps. These are:**



Problem solving process

- Problem solving-process of generating solutions for the given situation
- Problem is defined,
    1. in a context
    2. has well defined objective
    3. solution has set of activities
    4. uses previous knowledge and domain knowledge

Primary objective-problem identification

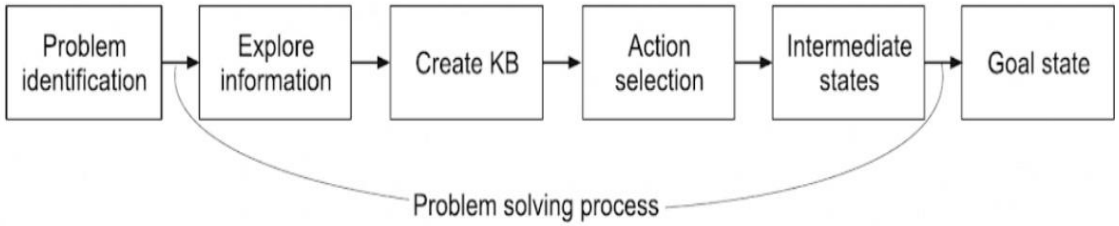


Figure 2.1 Problem-solving process.

Problem Formulation and Representation

- Example: Water jug problem-find out a way to empty 2 galloon jug and fill 5 galloon jug with 1 galloon of water

states: Amount of water in the jugs

actions: 1. empty the big jug

2. empty the small jug

3. pour water from small jug to big jug

4. . pour water from big jug to small jug

Goal: 1 galloon of water in big jug and empty the small jug

path cost: number of actions(minimum number of actions->better solution)

Representation: jugs(b,s), where b-amount of water in bigger jug, s- b-amount of water in smaller jug

initial state: (5,2)

goal state: (1,0)

operators: i) empty the jug

ii) fill the jug

Problem Formulation and Representation

- Solution:

  initial state: (5,2)

  goal state: (1,0)

  operators:

  i) empty big(remove water from big jug)

  ii) empty small(remove water from small jug)

  iii) big is empty(pour water from small jug to big jug)

  iv) small is empty(pour water from big jug to small jug)

  actions of sequence: 2,4,2,4,2

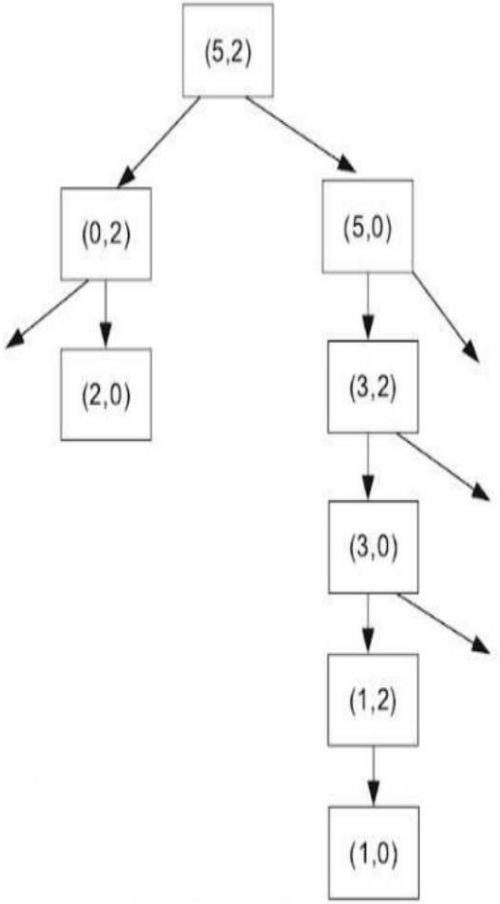
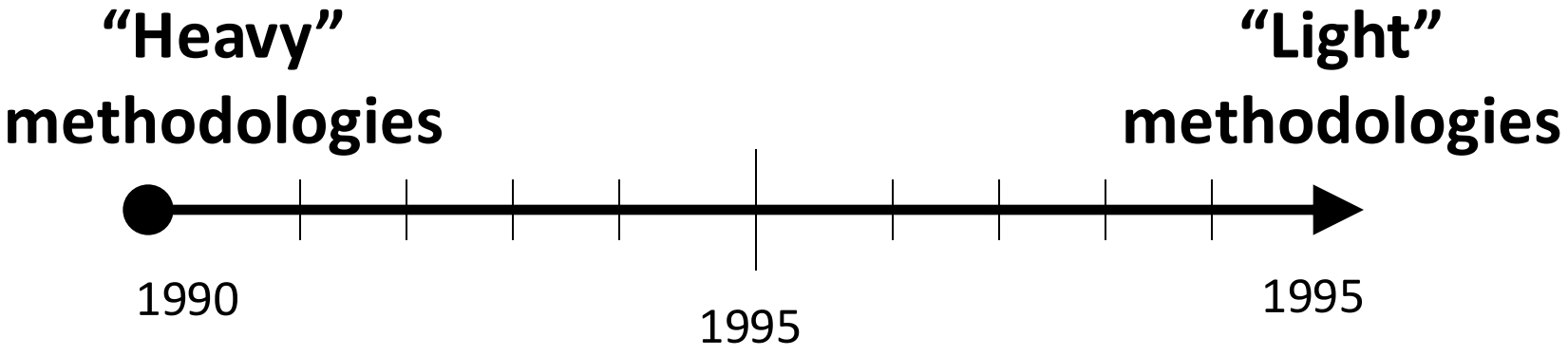


Figure 2.7 Water jug problem.

Agile Alliance

- The Agile Alliance is a non-profit organization that supports individuals and organizations who use agile approaches to develop software
- Schwaber is one of the leaders of the agile software development movement started Agile Alliance

History of the Agile Alliance



History of the Agile Alliance

"Light" Methodologies:

- Extreme Programming (XP)
- SCRUM
- Feature-Driven Design (FDD)
- Adaptive Software Development
- Crystal
- Pragmatic Programming
- DSDM
- Etc.

History of the Agile Alliance

2000,2001 big years for Agile Alliance:

- Meeting of XP and “light” proponents in spring of 2000
- Bob Martin suggests a second meeting in early 2001
- Wiki created late 2000
- February 2001 Agile Manifesto Created

History of the Agile Alliance

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:

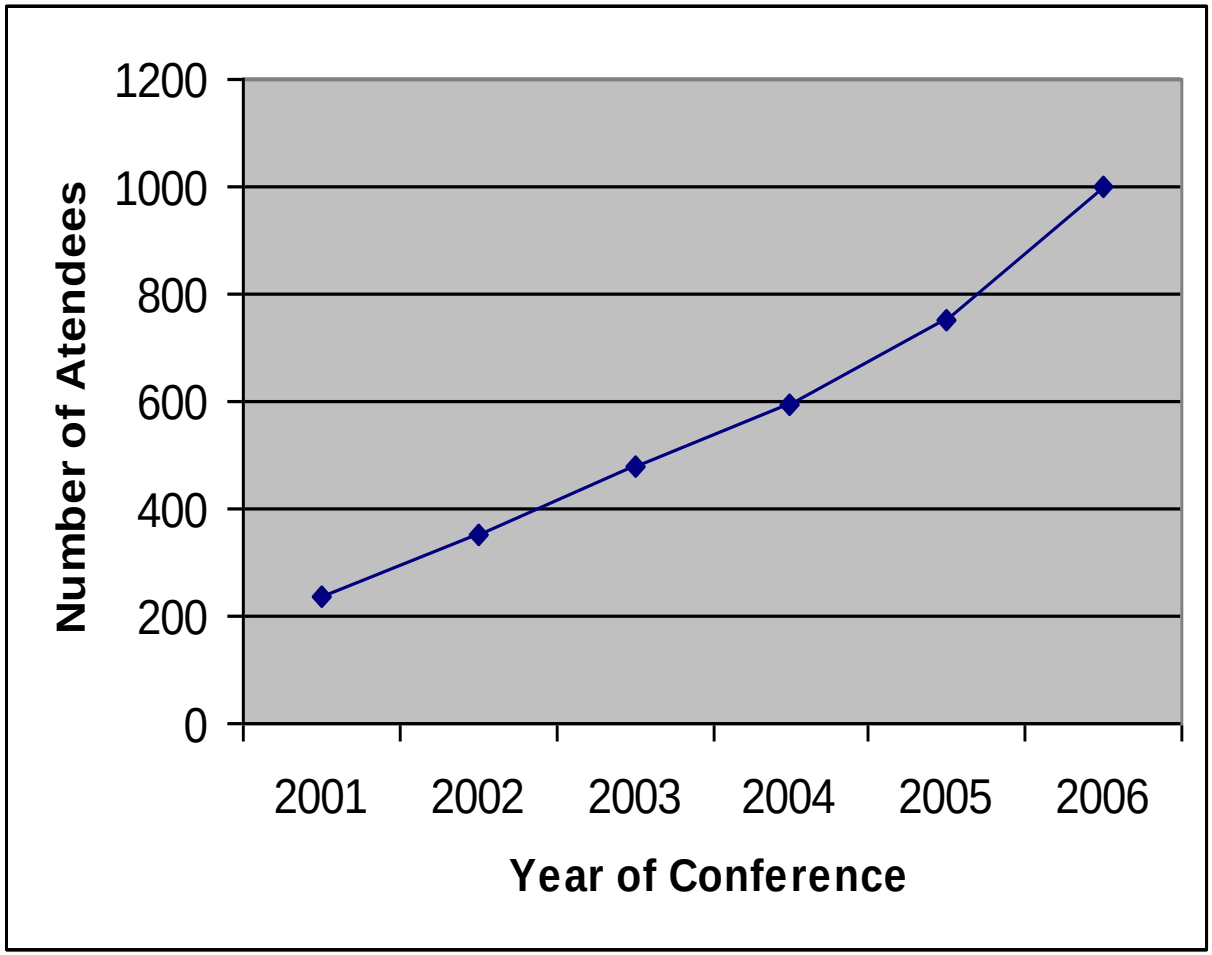
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

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

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

What is the Agile Alliance today?

- 2001,2002 XP Universe
- 2003,2004 XP Universe
- 2003,2004 Agile Development Conference
- 2005,2006 Agile200x



Provided by Ted Little, Agile2006 Program Director

What is the Agile Alliance today?

Programs the Agile Alliance is active in today:

- Agile Narratives Program
- Academic Research Program
- Agile Seminars Program
- Conference Sponsorship Program
- Speaker Reimbursement Program

The Agile Software Development Methodology

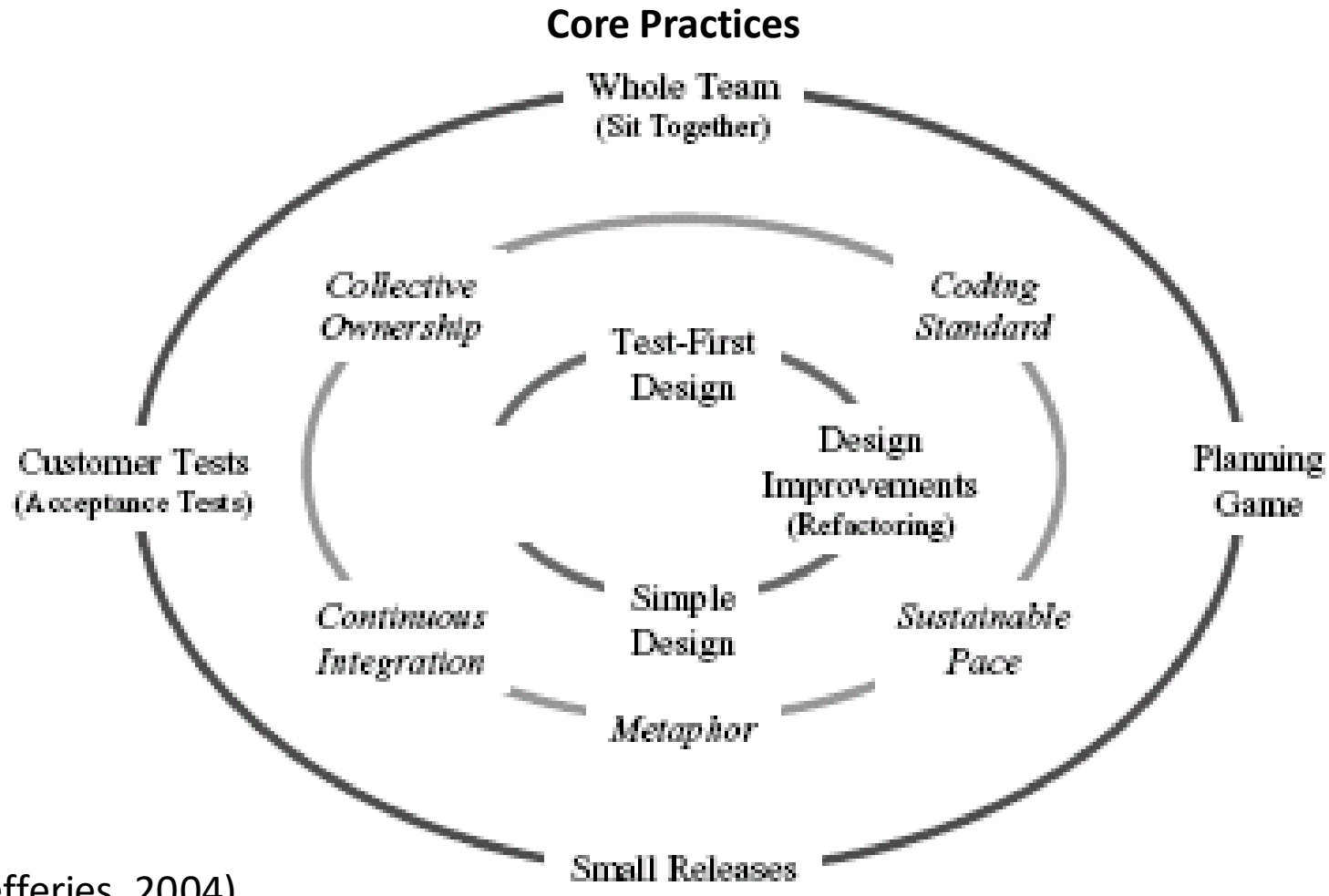
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

XP, how it looks when the methodology is actually used

The Agile Software Development Methodology

"Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation" (Jefferies,2004)

The Agile Software Development Methodology



(Jefferies, 2004)

Principles of Agile Practices

- Early and continuous delivery of valuable software.
- Embrace change.
- Frequent delivery.
- Cooperation.
- Autonomy and motivation.
- Better communication.
- Working software.
- Stable work environments.

Early and continuous delivery of valuable software

- Agile aims to deliver a functioning product in the very first development iteration. It will be a long way from being finished; but it just has to give the customer enough of an idea so that developers can receive valuable feedback. In doing so, agile projects can tailor the product as it is being developed to deliver something that satisfies customers' needs.
- Agile takes large tasks and breaks them up into smaller, more manageable chunks. The interactive nature of agile not only improves project development but also service delivery.

Embrace change

- Project development is becoming increasingly unpredictable. Markets are becoming more complex and the number of diverse products available to consumers grows every day. It's nearly impossible to predict what the final requirements of a project will be and as such, development projects are an increasingly risky.

Frequent delivery

- This principle may closely mirror the first, but while the first states that products should be delivered *early*, this principle goes a little more into detail as to why products should be delivered *continuously*.
- Smaller, more frequent releases mean less chance of error. Frequent releases give consumers more opportunity to provide feedback which, in turn, aids developers in correcting errors that might otherwise have derailed a project. If developers only receive feedback every few months, any errors made early in development may have since become much more difficult and costly to fix

Cooperation

- Agile requires that stakeholders, customers and developers work in unison (often in the same room) to achieve project goals.
- This reduces the risk inherent to project development, by helping improve communication and cooperation

Autonomy and motivation

- Owners and managers need to create an environment that rewards success, fosters healthy working relationships and helps improve employees' work/life balance.
- Give developers the tools and motivation to succeed and you'll be rewarded with easier project development and better returns on investment

Better communication

- Technology provides businesses a hundred different ways to communicate with employees, but none will ever be as good as face-to-face communication.
- 2020 has drastically changed the way we work, with more employees working remotely than ever before, businesses are becoming reliant on communication tools such as Skype and Microsoft Teams.

Working software

- A working product is more valuable than a checklist.” Requirements analysis documents, models and mock-ups may be useful, but they aren’t very useful if you can’t convert this information into a working product.

Stable work environments

- Agile promotes the idea of sustainable development. In short, this means that with the correct implementation of agile methods, there should never be the need for developers to work ridiculous hours just to meet deadlines.

Quality assurance

- Many businesses prioritize speed or quantity over quality. In some cases, this makes a lot of sense. Sometimes customers don’t care as much for quality so whether or not a product works.
- But, if development teams neglect quality for too long, their ability to adapt the product to suit current consumer demands diminishes and it becomes less agile

Simplicity

- This principle should be an ongoing effort. Agile teams need to recognize that there is always room for improvement. Technology provides us more varied, easier ways of doing things. Project management research helps managers and business continuously improve their practices to suit modern trends etc.
- Being agile requires businesses hold weekly or even daily meetings. A large part of these should be dedicated towards finding new and better ways of accomplishing tasks

Self-organizing teams

- Agile teams however, are comprised of multiple individuals, who share a wide variety of skills in several disciplines.
- Furthermore, agile development teams often include stakeholders, managers and consumers as core team-members. This allows them to work independently as a unit without the need to look to others for assistance

Reflection and adjustment

- This final principle is almost a proof-of-concept principle that indicates whether or not agile methodologies have been adequately incorporated into business strategy.
- Reflection on past success or failures and responsibly changing approaches to compensate is what makes agile so successful.

practices of extreme programming

- Key Practice #1 – Pair Programming
- Key Practice #2 – Planning Game
- Key Practice #3 – Continuous Process
- Key Practice #4 – Coding Standards
- Key Practice #5 – Sustainable Pace
- Key Practice #6 – Test Driven Development (TDD)

What is Planning?

- The task of coming up with a sequence of actions that will achieve a goal is called planning.
- Planning Environments
  1. Classical Planning Environments
     - Fully observable, deterministic, finite, static and discrete.
  2. Non classical Planning Environments
     - Partially observable, stochastic with different algorithms and agent designs.

Planning –Initial Exploration

- Design exploration software takes simulation to the next level by allowing users to determine appropriate values of variables that yield product designs that result in exceptional performance.

PLANNING PROBLEM

The planning problem is actually the question how to go to next state or the goal state from the current state. It involves two things 'how' and 'when'.

- The planning problem is defined with:

1. Domain model
2. Initial state
3. Goal state (next state)

The domain model defines the actions along with the objects. It is necessary to specify the operators too that actually describe the action. Along with this, information about actions and state constraints while acting should also be given. This entirely formulates the domain model.

The initial state is the state where any action is yet to take place (the stage when the exam schedule is put up!).

The final state or the goal state is the state which the plan is intended to achieve.

Planning vs. Problem solving

- Planning agent is very similar to problem solving agent
  - Constructs plans to achieve goals, then executes them
- Planning is more powerful because of the representations and methods used
- Search - proceeds through *plan space* rather than *state space*
- Sub-goals - planned independently, it reduce the complexity of the planning problem

	Problem Sol.	**Planning**
States	data structures	logical sentences
Actions	code	preconditions/outcomes
Goal	code	logical sentences
Plan	sequence from s_0	constraints on actions

Planning Agents

- **problem-solving agents** are able to plan ahead - to consider the consequences of *sequences* of actions - before acting.
- **knowledge- based agents** can select actions based on explicit, logical representations of the current state and the effects of actions.
  - This allows the agent to succeed in complex, inaccessible environments that are too difficult for a problem-solving agent

– **Problem Solving Agents + Knowledge-based Agents = Planning Agents**

Release Planning

- Agile release planning is a valuable technique for building customer-centric products and projects. It helps achieve your product roadmap through, planning poker Agile, iterative sprint planning, and daily standup meetings where you review and incorporate feedback to ensure incremental product improvements over time.

Release Planning

- An Agile release plan is a dynamic document that breaks down how and when the organization releases product features or functionalities. This plan prioritizes feedback from previous iterations and sets out the scope, timeline, and resources for each release.
- Agile release plans help teams decide how much functionality can be provided and how long it will take to develop. You can think of it as a prioritized list of functionalities to be delivered to the market, for an improved customer experience, over time.

Iteration planning

- Iteration Planning is an event where all team members determine how much of the Team Backlog they can commit to delivering during an upcoming Iteration
- Iteration planning determines the work that the team commits to be completed in the iteration by adjusting the predicted velocity and managing the number and priority of assigned, deferred, and/or new stories. The Iteration Planning Meeting is usually facilitated by the Team Agility Coach.

Task Planning

- Agile planning is a project management style with an incremental, iterative approach. Instead of using in an in-depth plan from the start of the project—which is typically product related—Agile leaves room for requirement changes throughout and relies on constant feedback from end users
- A task is a single unit of work broken down from a user story. A task is usually completed by just one person.

Challenges in Traditional Systems

- Customer feedback and testing take place at the end of the project, leaving less space for the developer to correct the misunderstood requirements or the end-user to understand the flow. Involves heavy weighted documentation, wasting lot of time and resources

Challenges in Traditional Systems

- Two key problem areas for traditional systems engineering have been identified:
- The first is the growing complexity of modern systems
- The second is the well-documented system of systems problem.
- In both cases, systems engineering tools and processes seem to be unable to keep up with demands

Real time applications of AI in Agile systems

- Personalized Shopping
- AI-powered Assistants
- Fraud Prevention
- Administrative Tasks Automated to Aid Educators
- Creating Smart Content
- Voice Assistants
- Personalized Learning
- Autonomous Vehicles

Test driven development

- Test Driven Development (TDD) is a software development practice that focuses on creating unit test cases before developing the actual code.
- It is an iterative approach that combines programming, the creation of unit tests, and refactoring

Test driven development

- TDD in agile is a framework that emphasizes the creation of unit test cases prior to writing the real code.
- It is an iterative process that incorporates programming, unit testing, and refactoring.
- It is inevitable for mistakes to occur throughout the design and coding phases of software development

Acceptance Test

- Acceptance Test -Driven Development, or ATDD, is a software development methodology, often associated with agile methodologies, that fosters collaboration between developers, testers and business stakeholders, and in which test automation plays a major role.

Acceptance Test

- UAT, or user acceptance testing, is the final stage in the software testing process.
- It is typically performed by the end-users or client to determine whether an application or feature fulfills its purpose.
- UAT must be completed before the software can be released to the market.

Types of Acceptance Test

- Alpha & Beta Testing.
- Contract Acceptance Testing.
- Regulation Acceptance Testing.
- Operational Acceptance testing.

Serendipitious Architecture

- The pressure that the acceptance tests placed on the architecture of the payroll system.
- The very fact that we considered the tests first led us to the notion of an API for the functions of the payroll system.
- Clearly, the UI will use this API to achieve its ends. Note also that the printing of the paychecks must be decoupled from the Create Paychecks function.
- These are good architectural decisions.

Serendipitious Architecture

First we add two employees.

Add employees.		
id	name	salary
1	Jeff Languid	1000.00
2	Kelp Holland	2000.00

Next we pay them.

Create paychecks.	
pay date	check number
1/31/2001	1000

Make sure 20% straight tax was removed.

Inspect paychecks.		
id	gross pay	net pay
1	1000	800
2	2000	1600

Serendipitious decoupling

- Decoupling is important because it makes the code easier to understand and maintain.
- Software development is not a solo sport; many developers will work on the same code base. Thus, making the code easier to read is beneficial for all involved. Using decoupling can make your code more readable and, therefore, more understandable for others (and yourself).

Serendipitious decoupling

- The benefits of decoupling are many, but here are just a few:
- You can create more modular code, which makes it easier to work with and test independently
- It reduces coupling between components and makes it possible for you to change one part without affecting another part (or many parts)
- It makes it easier for other people to understand how your application works

Serendipitious decoupling

- **How Can I Decouple My Code?**
- Use a dependency injection container.
- Use inversion of control.
- Use a service locator.
- Use a factory method.
- Object-oriented programming uses interfaces or abstract classes to create contracts for interacting with other types instead of using concrete dependencies directly in your codebase.

Serendipitious decoupling

- Use an adapter to provide a consistent interface to third-party libraries, making them more accessible for the rest of your codebase to interact with and leaving much more room for change in the future (ex: changing the data source library).
- Use a proxy to give a uniform interface to third-party libraries, making them more accessible to the rest of your codebase and allowing for much more flexibility in the future (ex: changing the data source library).