Hợp tác Đức
DEUTSCHE ZUSAMMENARBEIT

Implemented by
giz Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH

develoPPP
Where business meets development.

DVET

BOSCH
Sáng tạo vì cuộc sống

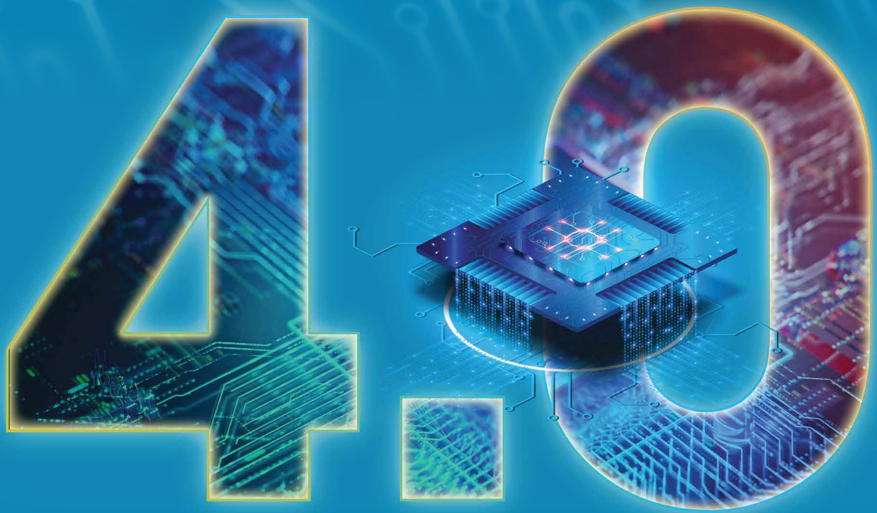
rexroth
A Bosch Company

LILAMA2
COLLEGE

Development Partnership with the Private Sector

Integrating requirements of Industry 4.0 in TVET

Training Module Industry 4.0



Programming of Mechatronic Systems

develoPPP programme of the German Federal Ministry for Economic Cooperation and Development

Hợp tác Đức
DEUTSCHE ZUSAMMENARBEIT

Implemented by giz Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH

develoPPP Where business meets development.

DVET

BOSCH Sáng tạo vì cuộc sống

rexroth A Bosch Company

LILAMA2 COLLEGE

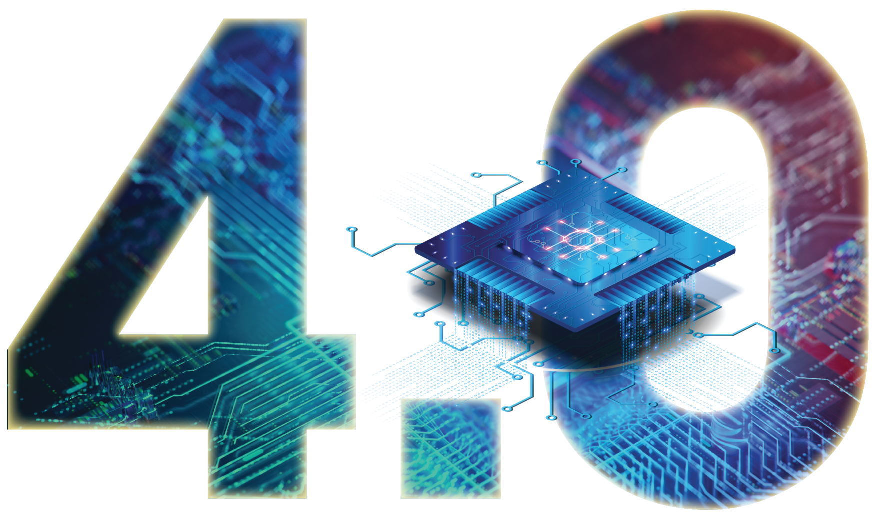
Development Partnership with Private Sector

Integrating requirements of Industry 4.0 in Technical and Vocational Education and Training

Bosch Rexroth - GIZ - LILAMA 2

Training Module

Industry



Programming of Mechatronic Systems

Development Partnership with Private Sector
Integrating requirements of Industry 4.0 in TVET
Bosch Rexroth - GIZ - LILAMA 2

Training Module
Industry 4.0 - Programming of Mechatronic Systems
1st Edition

Training Materials – Exercises – Project Works – Solutions

Editorial Board

Stefan Paschek, Kieu Tan Thoi, Phan Hong Phuong
Nguyen Trong Tin, Le Van Hung
Frank Schulze, Ralf Hill

Contributions

LILAMA 2 International Technology College
Technology Engineering Faculty
Training Department
Quality Assurance Department

GIZ Programme Reform of TVET in Viet Nam
Pham Thi Thanh Truc
Nguyen Minh Cong

Translation English language to Vietnamese
Nguyen Minh Ngoc
Tran Simon Trung Hieu

The book ***Training Module Industry 4.0 – Programming of Mechatronic Systems is*** *officially published by the cooperation partners of the development partnership*
"Integrating requirements of Industry 4.0 in TVET".
It was disseminated to eleven partner TVET Institutes of the ***Programme Reform of TVET in Viet Nam*** *and made available to other relevant stakeholders in TVET and in manufacturing industry.*
The book can be copied or downloaded on
www.tvet-vietnam.org
free for academic purpose and research without commercial interest. For any other purpose of use and duplication, inquire ***Programme Reform of TVET in Viet Nam*** *for further information and permission.*

Address: *#1, Lane 17, Ta Quang Buu Street, Bach Khoa Ward, Hai Ba Trung District, Hanoi, Viet Nam*
Tel: *+84 (0) 24 39 74 64 71*
Fax: *+84 (0) 24 39 74 65 70*
Website: *www.tvet-vietnam.org*
www.giz.de/vietnam

FOREWORD

Foreword to Industry 4.0 stakeholders in TVET and manufacturing industry

Technical and Vocational Education and Training (TVET) plays a crucial role in the development of a skilled and competent workforce for advanced industrial sectors. The adaptation of knowledge and skills of the Vietnamese workforce to the new competence requirements on digitalisation and Industry 4.0 is a challenge that needs to be addressed by TVET institutes in close cooperation with the business sector.

The cooperation partners Bosch Rexroth AG, LILAMA 2 International Technology College (LILAMA 2), with support of the Directorate of Vocational Education and Training (DVET), and the Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH tackled the challenge to qualify technicians and engineers who meet the requirements of Industry 4.0 at LILAMA 2 and other TVET institutes and implemented the development partnership ***"Integrating requirements of Industry 4.0 in TVET".***

Development partnerships with the Private Sector (DPP) promote private-sector activities where entrepreneurial opportunities and development policy potential meet and are supported by **the www.develoPPP.de** programme of the German Federal Ministry for Economic Cooperation and Development (BMZ).

The main activities of the cooperation partners comprised: raising awareness on necessary changes in the TVET system in Viet Nam; analysing the current and future demand of the industry; developing and integrating Industry 4.0 training modules into initial training programmes and further training courses; developing technical and didactical competencies of TVET teachers and in-company trainers as well as regular trainees and technical company staff; disseminating initial and further training programmes to other TVET institutes and to the TVET system in Viet Nam.

In this regard, national and international Industry 4.0 experts and curriculum designers developed the training module "Programming of Mechatronic Systems" with integrated IT-security and further trained teachers of LILAMA 2 and other partner TVET institutes of the Vietnamese-German *Programme Reform of TVET in Viet Nam* as master trainers and multipliers to the TVET system. The competencies mediated in the module orient on the latest Industry 4.0 qualifications of the revised German training ordinance for 3.5-year training in electronic and mechatronic occupations and fulfill the Industry 4.0 qualification demand of national and international companies in Viet Nam.

The Industry 4.0 training module complies to Circular 03/2017 of the Ministry of Labour, Invalids and Social Affairs and resembles the structure of the initial training programmes for intermediate and college levels, developed in the frame of the *Programme Reform of TVET in Viet Nam*. They are available for free download at **www.tvet-vietnam.org**. The teaching and learning material build up on these programmes and can be offered as an elective training module for college graduates of *Industrial Electronic Technician, Mechatronic Technician and Electronic Technician for Energy and Building Technology*. Furthermore, the module can be offered to industry technicians and engineers as further training on applied Industry 4.0 applications for in the field of electronics, mechatronics and automation.

TVET teachers are recommended to implement the training module **"Programming of Mechatronic Systems"** as hybrid learning or blended learning, in which trainees learn via electronic and online media as well as traditional face-to-face teaching and training.

On behalf of the cooperation partners, we would like to express our utmost gratitude for the assistance of all parties to develop the training module "Programming of Mechatronic Systems" and wish a successful implementation and dissemination.

Dr Juergen Hartwig
Director
Programme "Reform of TVET in Viet Nam"

TABLE OF CONTENTS

ABBREVIATIONS 8

A. TRAINING MODULE 9

Module name and code and scheduled time 9
I. Module classification and characteristics 9
II. Module objectives 10
III. Module content 12
1. General content classification and time allocation 12
2. Detailed content 13
Training Unit 1:Object Oriented Programming 13
Training Unit 2: Microcontroller Programming 16
Training Unit 3:Database Systems 18
Training Unit 4: Data visualisation with dashboards 20
IV. Requirements for module implementation 22
V. Assessment content and methodology 25
VI. Guidelines for professional module implementation 28

B. TRAINING UNITS 30

TRAINING UNIT 1: Object Oriented Programming 30
1. Object Oriented Programming 30
1.1. Fundamentals of programming 30
1.2 Adapting software modules 39
1.3 Basics of Object Oriented Programming 47
1.4. Extended concepts 57
2. Unified Modeling Language (UML) 60
2.1 What is UML? 60
2.2 Class diagrams to describe software requirements and documentation 61
2.3 Display and application of class relations 64
2.4 Display of inheritance trees for software documentation 65
2.5 Sequence diagrams to describe communication processes of software (PC and mechatronic systems) 66
2.6 Use Case Diagram for high level documentation of use cases 67
2.7 Implementation of UML diagrams 69
3. Analyse tasks and find solutions 73
4. Test software and rollout 75
4.1 Clarification of the V – Model 75
4.2. Software Testing 78
4.3 Software test plan 84

TRAINING UNIT 2: Microcontroller Programming 94
1. Microcontroller programming basics 94
1.1. Microcontroller Basics 94
1.2. Basic terms of programming 99
1.3. Difference between microcontroller and PLC 101
2. Microcontroller programming and hardware control 108
2.1. Basics of microcontroller programming 108
2.2. Hardware control 122

TABLE OF CONTENTS

3. Create measuring programmes 133
 3.1. Introduction to state machines 133
 3.2. Measuring programme 140
4. Communication methods and data exchange 142
 4.1. HTTP Requests 142
 4.2. Server Communication 146
 4.3. Publisher subscriber commuinication 155
Project Microcontroller : Data acquisition in I4.0 Setup 160

TRAINING UNIT 3: Database Systems 181
1. Database systems basics 181
 1.1 Basics for usage of databases 181
 1.2 Structure of database 191
2. Handling of databases on server level 194
3. Userprogram with databases 197
Project: Database Systems: Data acquisition in I4.0 Setup 204

TRAINING UNIT 4: Data Visualisation with Dashboards 225
1. Dashboards basics 225
 1.1. Defining the usage and tasks of dashboards 225
 1.2. Theoretical foundations for the creation of web applications 227
 1.3. Comparison of different creation methods 227
 1.4. Design guidelines for dashboards 231
 1.5. Dashboard limitations 237
 1.6. Explanation of the procedure for creating a dashboard 239
 1.7. Security aspects for dashboards 242
 1.8. Installation of necessary libraries 243
 1.9. Explanation of the necessary programming concepts, functions and classes to display dashboards 245
 1.10. Clarification of the necessary web-based programming components for dashboard programming 247
2. Possiblities for data visualization 249
 2.1. Fundamentals of data visualization 249
 2.2. Placement of diagrams in dashboards 254
 2.3. Conversion of data from data source to data sink in the diagram 254
3. Create and deploy dashboards 257
 3.1. Implementation of a suitable program logic to create a dashboard 257
 3.2. Consideration of user authentication options for security aspects 259
 3.3. Creating a visualization program 260

APPENDIXES 270
1. Project regions and partnering TVET institutes 270
2. List of charts 271
3. List of tables 271
4. List of illustration 271
5. List of figures 272

REFERENCES 276

ABBREVIATIONS

BMZ	Federal Ministry for Economic Cooperation and Development
DPP	Development Partnerships with the Private Sector
DVET	Directorate of Vocational Education and Training
GIZ	Deutsche Gesellschaft für Internationale Zusammenarbeit GmbH
I 4.0	Industry 4.0
LILAMA 2	LILAMA 2 International Technology College
TVET	Technical and Vocational Education and Training

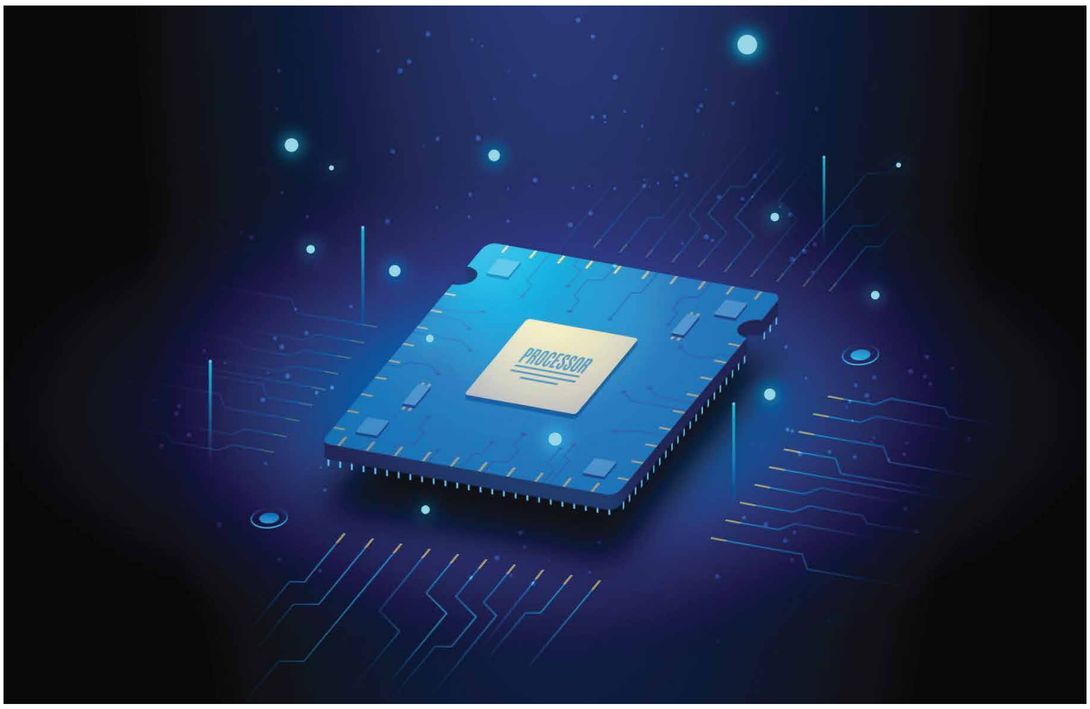


Illustration 1: Realistic microchip processor

A. TRAINING MODULE

(According circular No. 03/2017 / TT-BLĐTBXH March 1, 2017 of the Ministry of Labour, Invalids and Social Affairs)

Module name: Industry 4.0 **-** Programming of Mechatronic Systems

Module code: MD I 4.0 - Programming

Scheduled time: 320 hours

Theory: 80h

Practice/Laboratory/Discussion/Assignments: 220h

Examinations / Assessments: 20h

I. Module classification and characteristics:

Classification:

- Training module for higher Industry 4.0 qualification – International College Level. Admission requirement: National College Diploma in electronics or mechatronics occupations or 3 years proven professional work experience in the field and industry.

Characteristics:

- The module orients on German training and examination standard. It mediates digtalisation topics and advanced Industry 4.0 competencies as higher qualification in the fields of electronics, automation and mechatronics. The module is structured in 4 training units that contents build up on each other’s. It can be offered as supplementary training module for for college graduades of initial training programmes as well as offered to the industry in hybrid short-term trainings for technicians, engineers and in-company trainers.
- The trainees learn professional I 4.0 terminology acquire knowledge skills on programming in the industry 4.0 environment - an ever-increasing networking of industrial plants and an ever-increasing urge to store and display production and environmental data in a meaningful way. The trainees learn to implement consequential duties and tasks of electronic and meachatronic technicians in the future.
- Topics such as data acquisition, programming of software modules with object-oriented languages are dealt with in detail. The trainees learn various possibilities to create measurement and control programs with

microcontroller. They integrate the programmes into an I4.0 environment. They learn different possibilities to store data in databases, to evaluate and to visualize data in dashboards for knowledge processing. Aspects of correct software documentation as well as the creation of requirements and specifications, the correct design of databases and data security aspects are considered.

- After completing this module, the trainees will be able to identify possible applications for Industry 4.0, to work out proposals for solutions and to implement them. They will be able to form an interface between software development and production and to link both worlds synergetically.

II. Module objectives:

Knowledge: The trainees know and are familiar with:

- The historical developments in the field of digitization
- The definitions of the most important terms in the field of digitization / industry 4.0
- The clear distinction between Industry 3.0 and Industry 4.0
- Different application scenarios for Cyber-Physical Systems
- New IT-relevant object-oriented programming languages
- The structure and content of requirement and functional specifications
- Methodical approaches to testing and documenting software modules
- The dangers and risks for networked systems and adequate security measures
- Relational database systems and learn a database language
- The handling and application areas of microcontrollers in the I4.0 environment
- Options for data storage and retention as well as for the visualization and presentation of system-relevant measured variables in dashboards

Skills: The trainees are able to:

- Analyse technical systems and equipment and identify potential for data acquisition and the integration of additional sensor technology
- Build, modify and test networked systems
- Operate networked systems and carry out maintenance and optimization work
- Analyse a technical problem and develop a solution taking into account the prevailing conditions.
- Adapt and document software modules and integrate them into existing systems

- Design test plans and test the modified software modules under operating conditions
- Carry out systematic error/fault analyses and prepare comprehensive documentation of the entire procedure
- Integrate technical security measures into IT systems, inform the users of these systems about the correct behaviour and log the measures carried out in accordance with operational and legal requirements.
- Check the effectiveness of the implemented security measures, monitor compliance with data protection regulations and report security-relevant incidents
- Develop and implement data backup concepts
- Select the right procedures to suit the situation and complete software tests before deployment in the production system
- Create possibilities for process and data monitoring in compliance with company security guidelines
- Use ready-made software libraries and modify them according to the situation, while adhering to the operational guidelines of software versioning

Autonomy and responsibility: The trainees are able to:

- Inform themselves independently about emerging technologies and acquire the knowledge necessary for application in an industrial context.
- Search for freely available and operational software libraries and to adapt them to the given tasks
- Identify possible uses for Industry 4.0 applications in their companies, to work out proposals for solutions and to implement them

III. Module Content:

1. General content classification and time allocation:

Nr.	Modular Training Units	Scheduled time (hours)			
		Total	Theorie	Practice/ Laboratory/ Discussion/ Assignments	Examination/ Assessment
1	**1.** Object Oriented Programming 1.1. Programming basics 1.2. Unified Modeling Language (UML) 1.3. Aanalysing tasks and finding solutions 1.4. Test and rollout of Software 1.5. Practice your English	80	20	55	5
2	**2.** Microcontroller Programming 1.1. Microcontroller programming basics 1.2. Microcontroller programming and hardware control 1.3. Create measuring programs 1.4. Communication methods and data exchange 1.5. Practice your English	80	20	55	5
3	**3.** Database Systems 1.1. Database systems basics 1.2. Handling of databases on server level 1.3. Userprogram with databases 1.4. Practice your English	80	20	55	5
4	**4.** Data visualization with Dashboards 1.1. Dashboards basics 1.2. Possiblitiess for data visualization 1.3. Create and deploy dashboards 1.4. Practice your English	80	20	55	5
	Total hours:	320	80	220	20

2. Detailed content

Training Unit 1:

Object-Oriented Programming Time: 80 hours

Objective: The trainees:

- Know the difference between procedural and object-oriented programming
- Know the most important terms of object-oriented programming
- Have learned an object-oriented programming language
- Can use libraries and adapt ready-made classes to their needs
- Are able to analyse a technical problem and develop a solution taking into account the prevailing conditions.
- Are able to adapt and document software modules and integrate them into existing systems.
- Design test plans and test the modified software modules under operating conditions
- Carry out systematic error/fault analyses and prepare comprehensive documentation of the entire procedure

Content:

1.1.Programming Basics

1.1.1.Fundamentals of programming

1.1.1.1.Basic programming terms

1.1.1.2.Basics of pocederal programming

1.1.1.3.Introduction to Variables, Arrays, Conditions, Loops and Functions

1.1.1.4.Difference between IEC61131 and standard programming languages

1.1.1.5.Getting to know Integrated Development Environments (IDE)

1.1.1.6.Software documentation and comments

1.1.2.Adapting software modules

1.1.2.1.Include modules from predefined or external libraries

1.1.2.2.Understanding the program code

1.1.2.3.Documentation and commenting of program codes

1.1.2.4.Understanding the mechanism function call and modular programming

1.1.2.5.Interpretation of function documentation and transfer parameters

1.1.3.Basics of object-oriented programming

1.1.3.1.Procedural programming vs. object-oriented programming

- 1.1.3.2.Difference between classes and objects
- 1.1.3.3.Security aspect access rights for classes
- 1.1.3.4.Creation of constructor/structure for classes
- 1.1.3.5.Creating base classes for inheritance
- 1.1.3.6.Understanding inheritance structures and mechanisms
- 1.1.3.7.Use of the most important object-oriented structures
- 1.1.3.8.Installation and use of libraries

1.1.4.Extended concepts

- 1.1.4.1.Implementation of file input and output using object-oriented structures
- 1.1.4.2.Preparation and local visualization of data using object-oriented methods
- 1.1.4.3.Creation of graphical user interfaces

1.2.Unified Modeling Language (UML)

1.2.1.UML Basics

1.2.2.Class diagrams to describe software requirements and documentation

- 1.2.2.1.Representation of attributes and methods
- 1.2.2.2.Representation of possibilities for data encapsulation

1.2.3.Display and application of class relations

- 1.2.3.1.Association
- 1.2.3.2.Aggregation
- 1.2.3.3.Composition
- 1.2.3.4.Inheritance

1.2.4.Display of inheritance trees for software documentation

1.2.5.Sequence diagrams to describe communication processes of software (PC and mechatronic systems)

1.2.6.Use Case Diagram for high level documentation of use cases

1.2.7.Implementation of UML diagrams

- 1.2.7.1.Creation of classes using UML diagrams, as well as deriving UML diagrams from existing classes
- 1.2.7.2.Implementation of sequence diagrams for defined tasks
- 1.2.7.3.Creating Use Case Diagrams for sample applications

1.3.Analyse tasks and find solutions

1.3.1.Analysis of technical orders and development of solutions

1.3.1.1.Analysing customer requirements with regard to the required function

1.3.1.2.Clarify specifications in exchange with customers (internal/external)

1.3.1.3.Analyse processes, interfaces and environmental conditions as well as the initial state of the systems, determine and document requirements for software modules

1.3.1.4.Analyse and prepare data flows

1.3.1.5.Create requirement specification

1.4.Test software and rollout

1.4.1.Clarification of the V – Model

1.4.2.Draft test plan according to the operational test and release procedure, in particular defining procedures, standard and limit values of operating parameters and generating test data

1.4.3.Create a test plan based on requirements (specification sheet, legal or operational specifications)

1.4.4.Simulate technical environmental conditions

1.4.5.Testing software modules

1.4.6.System tests and component tests Perform tests in the system under operating parameters

1.4.7.Software Troubleshooting

1.4.7.1.Analysing malfunctions, systematic troubleshooting, possibly adapting requirements and specifications with documentation

1.4.8.Documentation and Release

1.4.8.1.Basics of code and software documentation

1.4.8.2.Software versioning

1.4.8.3.Release strategies

Training Unit 2:

Microcontroller Programming Time: 80hours

Objective: The trainees:

- Know the difference between using a microcontroller and a PLC
- Are able to control actuators and sensors and connect them to the microcontroller
- Can save measured values in log files
- Are able to search and find errors in a structured way
- Are able to commission, configure and parameterize a microprocessor
- Create state machines to implement measuring programs
- Are able to integrate software modules into a sequence program
- Are familiar with the possibilities for data exchange between microprocessors
- Know possibilities for connecting microprocessors to higher-level IT systems

Content:

2.1.Microcontroller programming basics

2.1.1.Microcontroller Basics

2.1.2.Basic terms of programming

2.1.3.Difference between microcontroller and SPS

Operating systems for microcontrollers

2.2.Microcontroller programming and hardware control

2.2.1.Basics of microcontroller programming

2.2.1.1.Installation of necessary software packages

2.2.1.2.Programming to control microcontroller peripherals

2.2.1.3.Software architecture in the microcontroller environment

2.2.2.Hardware control

2.2.2.1.Control of external hardware (actuators and sensors)

2.2.2.2.Use of pulse width modulation for hardware control

2.2.2.3.Use of ADC (analog to digital converter) for reading analog signals

2.2.2.4.Measurement conversion from digital value to physical quantity

2.2.2.5.Creation of log data with sensor signals

2.3.Create measuring programmes

2.3.1.Introduction to state machines

2.3.1.1.Fundamentals of state machines

2.3.1.2.Implementation of state machines for microcontrollers

2.3.1.3.Software design of a sequence program via state machines

2.3.2.Measuring programme

2.3.2.1.Implementation of a measuring program for automated acquisition of sensor values

2.3.2.2.Creation of a program to save sensor values in log files

2.3.2.3.Extension by alarm possibility to receive an alarm if a limit value is exceeded

2.3.2.4.Graphical representation of the log file

2.4.Communication methods and data exchange

2.4.1.HTTP Requests

2.4.1.1.Basics to webbased communication

2.4.1.2.Server-side setup of a web service for recording and evaluating requests

2.4.1.3.Creation of client sided communication program using HTTP requests

2.4.2.Client Server Communication

2.4.2.1.Basics to Client Server bases communication architectures

2.4.2.2.Server-side creation of a communication program for data acquisition and data backup using web sockets

2.4.2.3.Client-side creation of a measurement and communication program, for connection establishment and data transmission

2.4.3.Publisher Subscriber Communication

2.4.3.1.Publisher Subscriber Communication Architecture Basics

2.4.3.2.Design of communication architecture

2.4.3.3.Creating Publish and Subscriber Side Measurement and Backup Modules

2.4.4.Implementation of a program for communication and data exchange of two microcontrollers with provided function blocks

Training Unit 3:

Database Systems Time: 80hours

Objective: The trainees:

- Have a basic understanding of databases and related system concepts
- Are able to set up and configure a database server
- Know the basic command sequences for manipulating databases
- Are able to develop programs which write measured values into databases and read them from databases

Content:

3.1.Database systems basics

3.1.1.Basics for usage of databases

3.1.1.1.Basic concepts of databases

3.1.1.2.Necessary software components (database server)

3.1.1.3.Manual configuration of database server settings

3.1.1.4.Creation of a database under consideration of security aspects

3.1.2.Theoretical principles for databases

3.1.2.1.Structure of databases

3.1.2.2.Introduction to relational database models

3.1.2.3.Getting to know the Entity Relationship Diagram for database design and documentation

3.2.Handling of databases on server level

3.2.1.Database manipulation

3.2.1.1.Creation of a database on a database server

3.2.1.2.Correct creation of tables and database entries for relational databases

3.2.1.3.Getting to know the basic commands for manipulating databases

3.2.1.4.Execution of join commands to join tables within databases

3.3.Userprogram with databases

3.3.1.Database creation

3.3.1.1.Getting to know classes and functions to connect to database servers

3.3.1.2.Creating user programs to create databases on database servers

3.3.1.3.Implementation of user programs to store and read data on database servers

3.3.2.Database user programme

3.3.2.1.Analyse the current state of an existing mechatronics system and develop a concept to make the system I4.0 suitable (sensors and actuators)

3.3.2.2.Creating a database structure via ER diagram

3.3.2.3.Creating UML Use Case, Class and Sequence diagrams to grasp the system requirements

3.3.2.4.Implementation of a software structure using UML to link a measuring program with a database

3.3.2.5.Programming a measuring program to store measured values in a database

3.3.2.6.Creating an evaluation program to read out locally measured values from a database

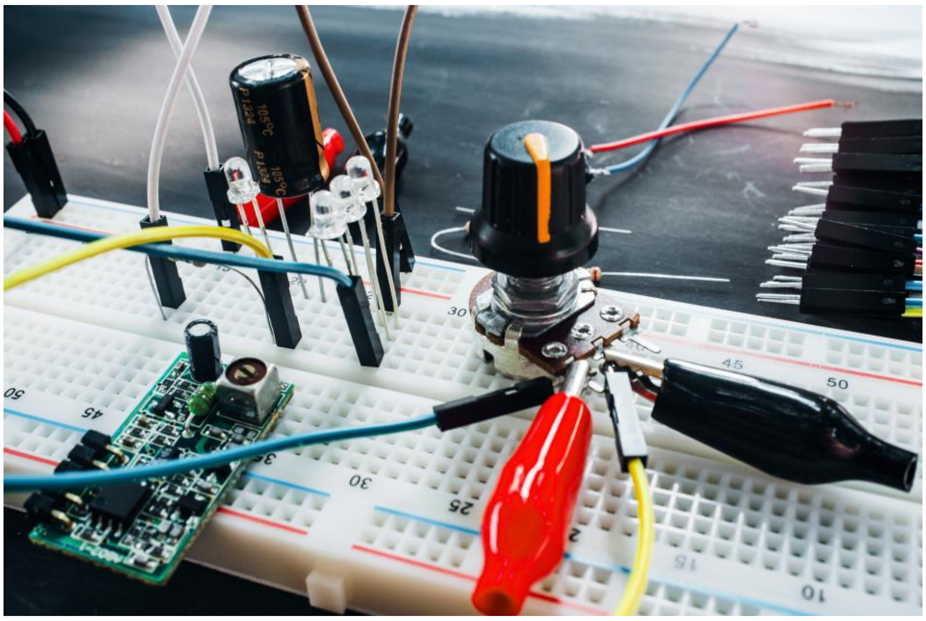


Illustration 2: Electronic board components

Training Unit 4:

Data visualisation with dashboards Time:80 hours

Objective: The trainees:

- Learn how to handle large amounts of data
- Are able to use the right visualization method for different requirements
- Are familiar with programming dashboard applications in combination with databases
- Can deploy dashboards in the local network and on publicly accessible servers

Content:

4.1.Dashboards basics

4.1.1.Defining the usage and tasks of dashboards
4.1.2.Theoretical foundations for the creation of web applications
4.1.3.Comparison of different creation methods
4.1.4.Design guidelines for dashboards
4.1.5.Dashboard limitations
4.1.6.Explanation of the procedure for creating a dashboard
4.1.6.1.Comparison of prefabricated dashboard solutions to self-implemented solutions
4.1.7.Security aspects for dashboards
4.1.7.1.Local deployment vs internet deployment
4.1.8.Installation of necessary libraries
4.1.9.Explanation of the necessary programming concepts, functions and classes to display dashboards
4.1.10.Clarification of the necessary web-based programming components for dashboard programming

4.2.Possiblities for data visualization

4.2.1.Fundamentals of data visualization
4.2.1.1.Display of different diagram types with application areas
4.2.1.2.Definition of the necessary data structures to use daigaramm types
4.2.1.2.1.Continuous data vs. features
4.2.1.3.Placement of diagrams in dashboards
4.2.1.4.Conversion of data from data source to data sink in the diagram
4.2.1.4.1.Data preparation, pre-filtering, source selection

4.3.Create and deploy dashboards

4.3.1.Implementation of a suitable program logic to create a dashboard

4.3.1.1.Implementation of static dashboards without data update

4.3.1.2.Implementation of dynamic dashboards with update interval

4.3.1.2.1.Considering network traffic

4.3.2.Consideration of user authentication options for security aspects

4.3.2.1.User name, passwort

4.3.3.Creating a visualization program

4.3.3.1.Creating a measurement program to store sensor data in a database

4.3.3.2.Implementing a dashboard to continuously read and graphically display measured values from the database

4.3.3.3.Generation of a limit value mechanism to detect exceeding of a limit value

4.3.3.4.Generation of an Alerting Mechanism to give warnings when limit values are exceeded

Illustration 3: Data visualizer graphic

IV. Requirements for module implementation

1. Vocational classroom/ training workshop

Vocational Classroom:

- Offers barrier-free access and workplaces, complies with occupational safety regulations, technical directions, and legal regulations.
- Provides sufficient workspace for the number of trainees as well as operational IT infrastructure with PC workstations with appropriate programming software and internet connection
- Training workshop:
- Workshop for electronics/mechatronics industrial project works
    - Offers barrier-free access and workplaces, complies with occupational safety regulations, technical directions, and legal regulations.
    - Provides sufficient workspace and machine workplaces for the number of trainees
- Barrier-free toilet rooms, bathrooms, and changing rooms, separately for women and men

2. Equipment and machinery:

- Stationary machine tools
- Analog and digital measuring tools
    - Length measuring tools (/callipers)
    - Angle measuring tools (yardsticks)
    - Test gauges
    - Bipolar voltage tester, multimeter,
    - Current clamp, installation tester
- Devices
    - Industrial plants (complete systems and plants for production or of processes) such as process automation plant for the production of fluids and substances, bottle processing systems, collection tables, production stations, test stations
    - Process engineering, manufacturing engineering, control engineering demonstrators as well as I4.0 demonstrators, functional and accessible for mounting microcontroller sensor technology.
- Others
    - Administrator access to PC and microcontroller to perform necessary software installation.

3. Teaching and learning materials, tools, consumption materials:

- Hand tools
- Pliers (crimping pliers, side cutters, pointed pliers, wire stripper)
- Wrench assortment(s) (hexagon/hexagon socket)
- Cable knife, scissors
- Pegboards

- Industrial components of automation technology
  - Flexible assembly racks made of aluminum profiles for the construction of subtasks in automation technology
  - Pneumatic and electropneumatic components
  - Hydraulic and electrohydraulic components,
  - Electrical drives such as three-phase asynchronous motor, servo motor, stepper motor
  - PLC compact units (networkable and with AI/AO), modular PLC (networkable and with AI/AO), power supply units depending on load sizes
  - PLC modules and network materials for ASi and PROFI bus, PROFINET and Ethernet, if necessary, also addressing devices
  - Router and IOT gateways for connection to industry 4.0
  - Powerful notebook or desktop PC, user software for drawing and simulation, PLC software
  - Microcontroller with operating system, W - Lan module, alternatively also network connection, periphery to control actuators or read in sensors, power connection
  - Microcontroller capable actuators, compatible in voltage range, power stages
  - Microcontroller suitable analog sensors, voltage range must be compatible, different types for implementing example application, at least temperature, distance, humidity, acceleration
  - Network components to connect the microcontroller and IT infrastructure (functional network cables, Wlan routers, or lan switches, servers)
  - Power supply for microcontrollers and actuators adapted to the respective specified performance data

- Auxiliary materials
  - auxiliary and operating materials for the wokassignment and maintenance work in accordance with the practical exercises and work orders, including tests

- Consumtion materials
  - consumables for the work assignment according to the practical exercises and work orders, including tests
  - Connection cable for microcontroller and production engineering

- Protective equipment
  - Personal safety equipment (PSA)
  - work safety clothes, safety shoes, visual cover, hearing protection

- Technical literature and table books
  - Handbook and textbook Mechatronics, Handbook Informatics

- Technical documentation
  - Partial, group and general drawings, layout plans
  - Installation descriptions, maintenance plans, functional descriptions
  - Circuit diagrams, wiring diagrams, work plans
  - nominal value tables, measurement reports, evaluation reports

- Software
  - User software for drawing and simulation,
  - PLC software (TIA-Portal or Step 7)
  - Simulation Software - Automation Technology
  - Learning software for self-study
  - Object-oriented programming language (e.g. Python)
  - Necessary libraries
  - Software for creating SSH connections to microcontrollers (e.g. Putty)
  - Integrated Development Environment THE compatible programming language
  - Relational Database Language
  - Relational database management system (e.g. MySql)
  - Web server (e.g. XAMPP)
  - Software or libraries for creating dashboard applications (e.g. Dash and Things board)

4. Other requirements and conditions: None

V. Assessment content and methodology

1. Content:

- **Knowledge:**
  - To know and observe legal and operational guidelines for quality assurance as well as data protection and IT security when working with and in digital systems - To describe Industry 4.0 and the digitization of production processes as well as data protection and IT security in manufacturing
  - Automation of machine tools and production systems.
  - To know flexible manufacturing plants and systems as well as handling systems and robots for flexible manufacturing plants and to assign them to the intended purpose
  - To Analyse information necessary for order processing, also from digital media and in English
  - Describe business requirements and goals of production and calculate operational parameters
  - to Analyse influences on the production process and to take them into account in the planning
  - Linking the knowledge of manufacturing and process plants with I4.0 knowledge for connecting and extracting process data

- **Skills**
  - to plan production orders customer-specifically as well as to consider technological, business, environmental and safety aspects and IT security
  - maintain, Analyse, save and archive data
  - to know and use assistance, simulation, diagnosis and visualization systems
  - Set up manufacturing processes with conventional and numerically controlled machine tools and/or manufacturing systems Machine tools
  - to monitor, control and optimize manufacturing processes - to guarantee series production readiness of manufacturing processes
  - to detect and eliminate disturbances and errors in the production process
  - to apply operational and customer-specific quality assurance systems
  - to systematically search for and eliminate the causes of quality defects

  - document production processes, quality inspection and faults/ malfunctions
  - Contribute to the continuous improvement of work processes in the operating procedure
  - prepare products and protocols and hand them over to external customers or to the subsequent production area (internal customer)

- **Autonomy and responsibility:**
  - to observe and apply general regulations of occupational safety, health, fire and environmental protection (observation, checklist with 90% correct answers)
  - to Analyse work orders of internal and external customers and to assess the technical and economic feasibility in compliance with safety and environmental protection regulations
  - take into account customer-specific requirements and deadlines and arrange for partial orders
  - to research and evaluate information for job planning in digital networks
  - initiate, monitor and control partial orders
  - to hand over products to external customers or to the subsequent production area (internal customer) and to present work results also with the help of digital media
  - To assume responsibility in the manufacturing process and to be aware of product liability in the context of business relations with customers.
  - To communicate and cooperate in interdisciplinary teams
  - to use energy and material under economic and environmental aspects and to dispose of materials and substances in an environmentally friendly way
  - Ensure learning time and learning creativity (observation, checklist). - To participate actively in the lessons (more than 80% in theoretical and 100% in practical lessons).

2. Methods

- The assessment is based on the project work carried out and products manufactured by the trainee/learner and is carried out in accordance with the provisions on the minimum knowledge and skills required for graduates of intermediate and/or college level in the profession.

- **Knowledge:**
  - The trainee's/learner's skills and behaviour are determined based on oral and written tests such as quizzes, technical discussions, and multiple-choice questions, as well as through integrated theory - practice exercises or practical exercises during the implementation of the teaching units of the module. The evaluations are calculated according to the valid point rules

- **Skills:**
  - On the basis of practical exercises, project work and company work orders, the practical performance of the trainee/learner is assessed with regard to the following criteria with the aid of evaluation sheets/scales:
    + occupational safety
    + Organization of the workplace
    + Technical standards
    + Planning and implementation
    + target time
    + self-assessment

- **Autonomy and responsibility:**
  - Following attitudes and characters of the trainees/learners are determined and evaluated by observation over the entire training period: work, learning and cooperation ethics, regulation and regulation morale, diligence, conscientiousness, discipline, ability to work in a team, punctuality, independence, sense of responsibility, prudence, initiative, active participation in lessons and support/motivation of others in the steering process.

VI. Guidelines for professional module implementation:

1. Scope of application

- Training module for higher Industry 4.0 qualification – International College Level. Admission requirement: National College Diploma in electronics or mechatronics occupations or 3 years proven professional work experience in the industry.

2. Guidelines on teaching and learning methods

- For teachers, lecturers and in-company trainers:
  The responsible teachers (at theTVET institute)/in-company trainers (in the company) observe the following guidelines for the professional implementation of the theory lessons and practice training of the whole module and each single training units:
  - The trainees/learners shall be instructed in detail and comply with regulations on occupational safety, health and environmental protection as well as fire protection and security. Compliance with the regulations must be continuously monitored by the responsible teacher/responsible In-company trainer. The trainee/learner is to be expressly informed and made aware of the appropriate measures and consequences of non-compliance with the regulations.
  - The learning process and learning progress of the trainees/learners shall be continuously monitored and regularly evaluated, in particular the consistent compliance with occupational safety regulations and environmental protection conditions.
  - To ensure the highest possible quality of teaching and training through the reference to the corresponding teaching unit in the planning and implementation of lessons.
  - Within the framework of the practice training units, the necessary work steps shall be carefully explained to the trainee/learner and demonstrated correctly.
  - The personal level of knowledge and skills shall be checked and assessed individually for each practical teaching unit on the basis of regular work reports drawn up by the trainee.
  - The quality of teaching is increased and ensured by the increased use of different teaching and learning methods such as the 4-step method, project method, guiding text, self-study and group work as well as by the efficient use of teaching and learning materials and

other aids.
    - The work results of the trainees/learners are to be evaluated and discussed transparently and together with the trainees/learners by the responsible vocational teacher or by the company trainer.
- For trainees/learners:
    - The trainees/learners are instructed to:
        - strictly follow the instructions of the vocational teachers or the company trainers
        - participate regularly and actively in the lessons and each teaching unit of the training module
        - observe the regulations on occupational safety and health, fire and environmental protection
        - actively contribute to environmental protection
        - observe the teaching and workshop regulations
        - participate attentively in class, take notes and ask questions in case of uncertainty
        - ask questions to the vocational school teachers or the company trainers or to other trainees/learners to ask for support with difficult tasks and to name problems
        - prepare the workplace and keep it clean and tidy
        - prepare, properly handle and maintain the equipment
        - prepare daily and weekly work reports on the theoretical and practical lessons of the module

3. Aspects to be considered:

- The training emphasis of the training module lays on all training units: 1, 2, 3 and 4

4. Further notes and explanations (if any)

B. TRAINING UNITS

UNIT 1: OBJECT- ORIENTED PROGRAMMING

Unit 1: Object-oriented programming

Objective: The trainees:

- Know the difference between procedural and object-oriented programming
- Know the most important terms of object-oriented programming
- Have learned an object-oriented programming language
- Can use libraries and adapt ready-made classes to their needs
- Are able to Analyse a technical problem and develop a solution taking into account the prevailing conditions.
- Are able to adapt and document software modules and integrate them into existing systems.
- Design test plans and test the modified software modules under operating conditions
- Carry out systematic error/fault analyses and prepare comprehensive documentation of the entire procedure

Content:

1. Object-oriented programming

1.1. Fundamentals of programming

a. Basic programming terms

- Algorithm

An algorithm is a set of instructions or rules designed to solve a definite problem. The problem can be simple like adding two numbers or a complex one, such as converting a video file from one format to another.

- Programme

A computer programme is termed as an organized collection of instructions, which when executed perform a specific task or function. A programme is processed by the central processing unit (CPU) of the computer before it is executed. An example of a programme is Microsoft Word, which is a word processing application that enables users to create and edit documents. The browsers that we use are also programmes created to help us browse the internet.

b. Basics of procedural programming

Procedural programming is a programming paradigm, derived from imperative programming, based on the concept of the *procedure call*. Procedures (a type of routine or subroutine) simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a programme's execution, including by other procedures or itself. The first major procedural programming languages appeared circa 1957–1964, including Fortran, ALGOL, COBOL, PL/I and BASIC. Pascal and C were published circa 1970–1972.

Computer processors provide hardware support for procedural programming through a stack register and instructions for calling procedures and returning from them. Hardware support for other types of programming is possible, but no attempt was commercially successful (for example Lisp machines or Java processors).

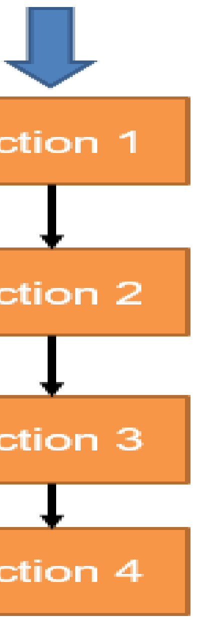


Chart 1: Procedural programming

c. Introduction to Variables, Arrays, Conditions, Loops and Functions

- **Variables**

What is a Variable in Python?
A Python variable is a reserved memory location to store values. In other words, a variable in a python programme gives data to the computer for processing.
Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc.
Variables can be declared by any name or even alphabets like a, aa, abc, etc.

Variable Naming Rules in Python.
Variable name should start with letter(a-zA-Z) or underscore (_).
Valid : age , _age , Age Invalid : 1age.
In variable name, no special characters allowed other than underscore (_).
Valid : age_ , _age
Invalid : age_*
Variables are case sensitive. age and Age are different, since variable names are case sensitive.
Variable name can have numbers but not at the beginning. Example: Age1 5.
Variable name should not be a Python keyword. Keywords are also called as reserved words. Example pass, break, continue. etc are reserved for special meaning in Python. So, we should not declare keyword as a variable name.
How to Declare and use a Variable Let see an example. We will declare variable "a" and print it.
a=100
print (a)

- **Mathematical operations**
  - ✓ a = 4
  - ✓ b = 3
  - ✓ c = a + b = 7
  - ✓ c = a - b =1
  - ✓ c = a * b = 12
  - ✓ c = a / b = 1.33333333
  - ✓ c = a % b = 1 (modulo)
  - ✓ c = a ** b = 64
  - ✓ c = a // b = 1 (integer division**)**
- **Functions**

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
The code block within every function starts with a colon (:) and is indented.
The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

- **Conditions**

Python uses boolean logic to evaluate conditions. The boolean values True and False are returned when an expression is compared or evaluated.
Condition **Change programme flow via condition**
Redirects flow of code execution

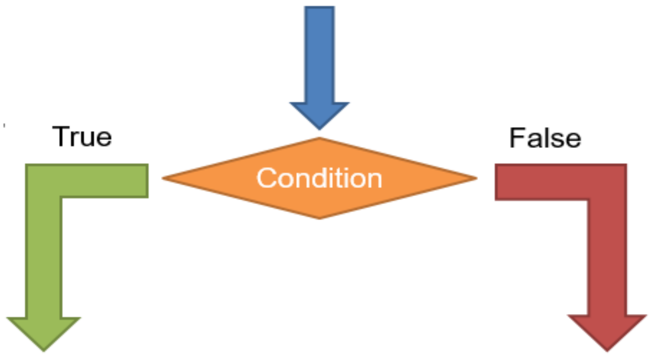


Chart 2: Condition function

Programme can follow true or false path

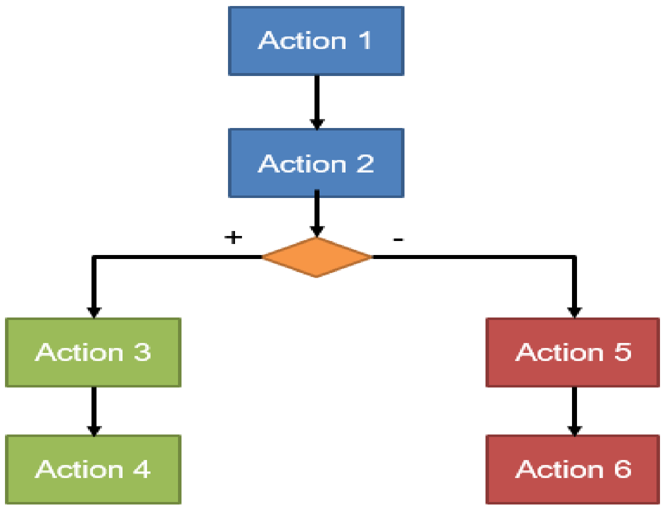


Chart 3: Condition programming path

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- ✓ Equals: a == b
- ✓ Not Equals: a != b
- ✓ Less than: a < b
- ✓ Less than or equal to: a <= b
- ✓ Greater than: a > b
- ✓ Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

- **Loops**

Python has two primitive loop commands:

- ✓ while loops
- ✓ for loops

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- **Arrays**

Arrays are used to store multiple values in one single variable:

Example

Create an array containing car names:

```
cars = ["Ford", "Volvo", "BMW"]
```

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

```
x = cars[0]
```

Example

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

d. Difference between IEC61131 and standard programming languages

- **PLC programming vs PC programming**
  - ➢ PLC programming

    + Defined fixed cycle

    + Real time operating system

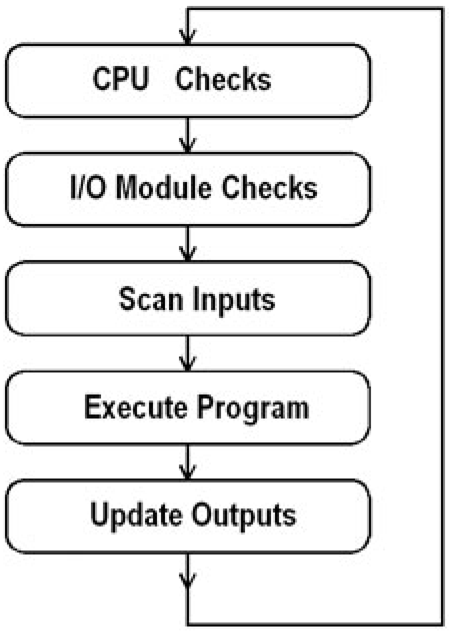


Chart 4: Scan cycle of PLC

PLCs operate by continually scanning programmes and repeat this process many times per second. When a PLC starts, it runs checks on the hardware and software for faults, also called a self-test. If there are no problems, then the PLC will start the scan cycle. The scan cycle consists of three steps: input scan, executing programme(s), and output scan.
Input Scan: A simple way of looking at this is the PLC takes a snapshot of the inputs and solves the logic. The PLC looks at each input card to determine if it is ON or OFF and saves this information in a data table for use in the next step. This makes the process faster and avoids cases where an input changes from the start to the end of the programme.
Execute Programme (or Logic Execution): The PLC executes a programme one instruction at a time using only the memory copy of the inputs the ladder logic programme. For example, the programme has the first input as ON. Since the PLC knows which inputs are ON/OFF from the previous step, it will be able to decide whether the first output should be turned ON.
Output Scan: When the ladder scan completes, the outputs are updated using the temporary values in memory. The PLC updates the status of the outputs based on which inputs were ON during the first step and the results of executing a programme during the second step. The PLC now restarts the process by starting a self-check for faults.

- PC programming

+ No defined cycle
+ Cycle can be programmed
+ Real time depends on operating system and programme design

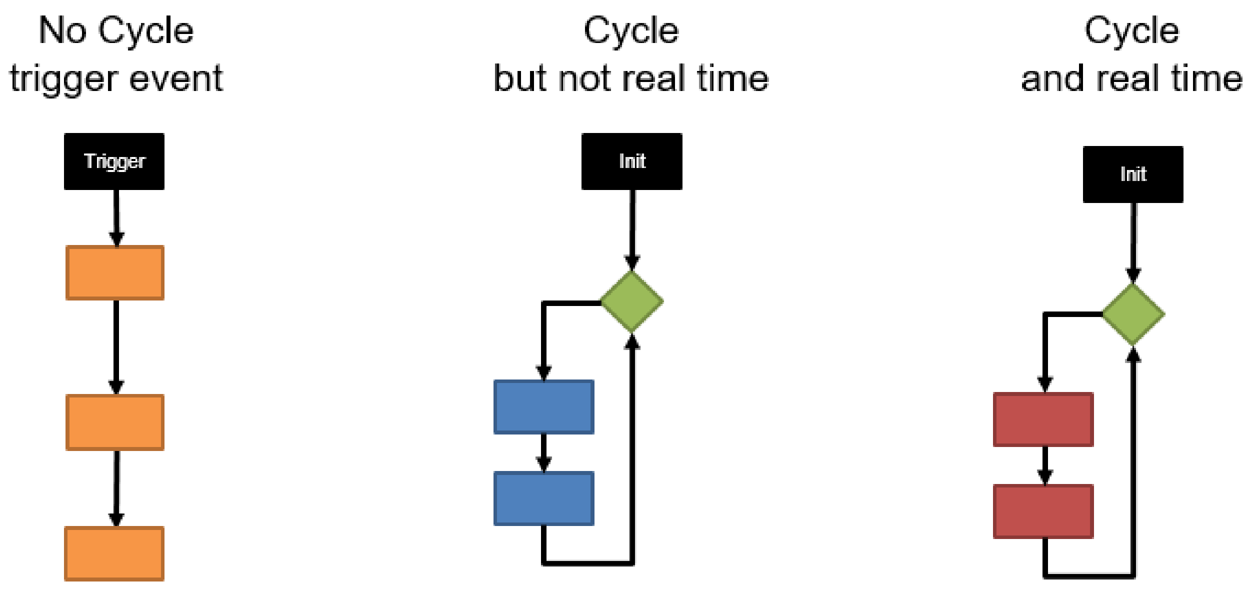


Chart 5: Cycle and real time

e. Getting to know Integrated Development Environments (IDE)

An **integrated development environment** (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger. Some IDEs, such as NetBeans and Eclipse, contain the necessary compiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not.

The boundary between an IDE and other parts of the broader software development environment is not well-defined; sometimes a version control system or various tools to simplify the construction of a graphical user interface (GUI) are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram for use in object-oriented software development.

f. Software documentation and comments

- **Python 3.9**

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general purpose language, meaning it can be used to create a variety of different programmes and isn’t specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most-used programming languages today. A survey conducted by industry analyst firm RedMonk found that it was the most popular programming language among developers in 2020.

- **Pycharm 3.9**

Pycharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains (formerly known as IntelliJ). It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda.
Pycharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license.

1.2. Adapting software modules

a. Include modules from predefined or external libraries

- Open windows system command (Type "cmd" into windows search)

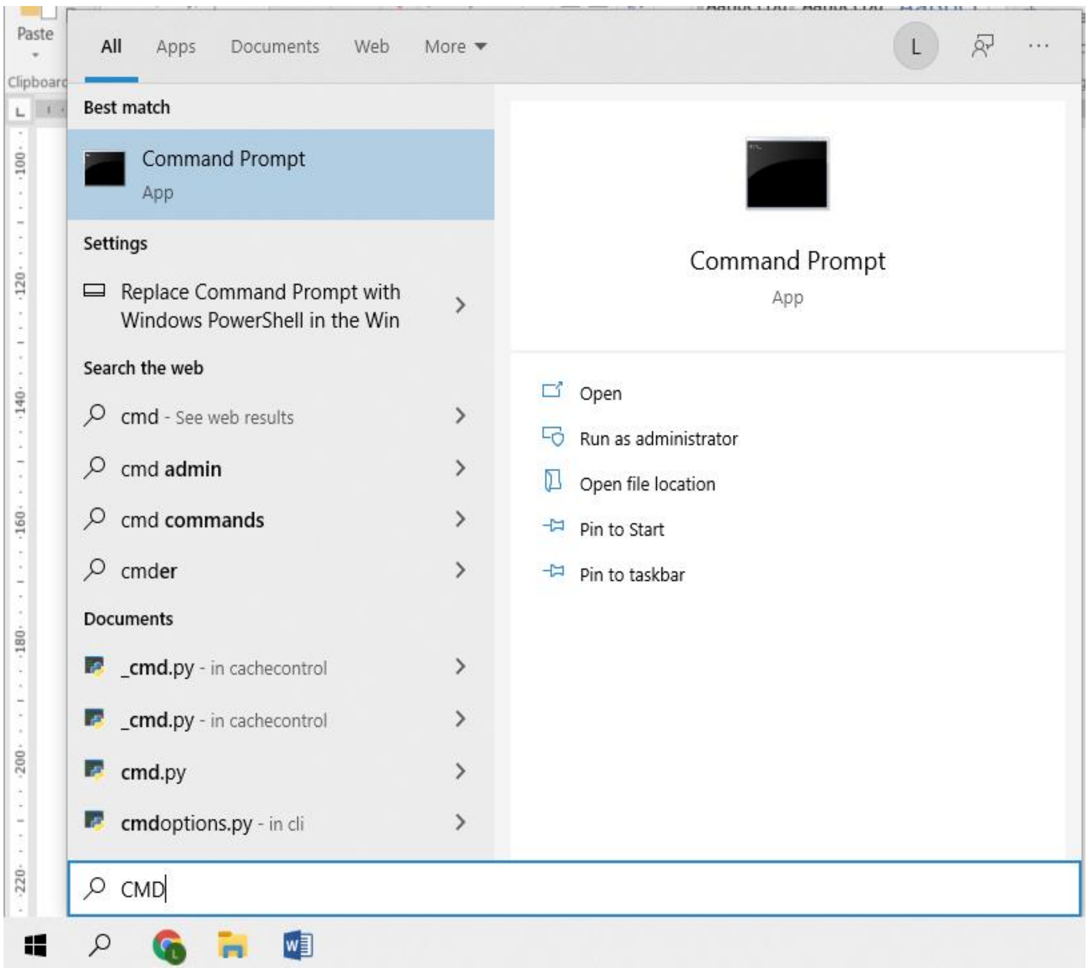
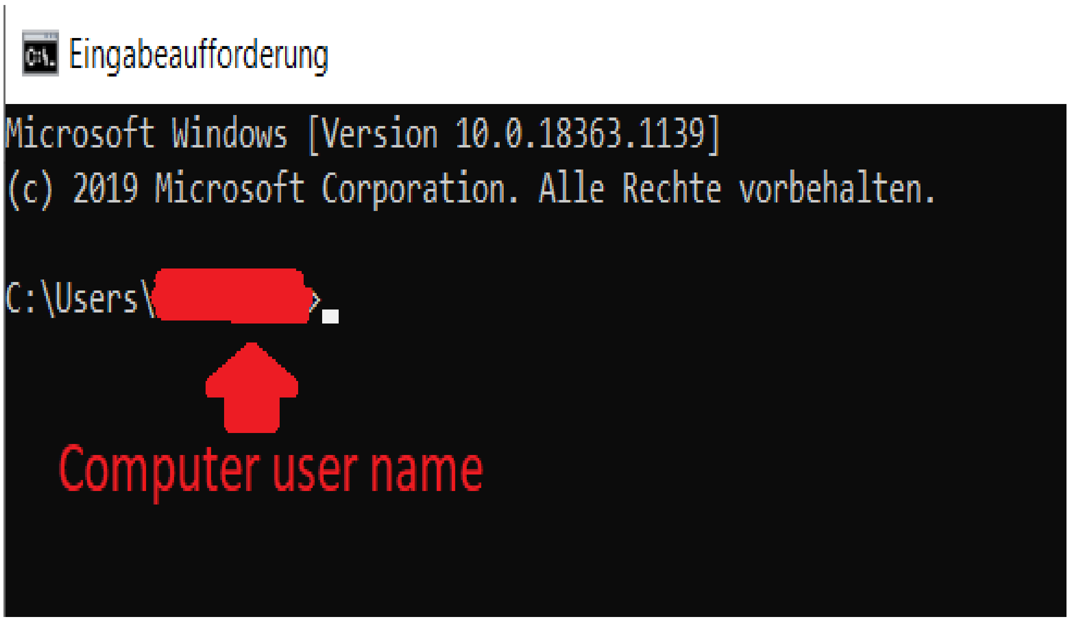


Figure 1: Windows system command interface

- Window should look like this:



```
Eingabeaufforderung
Microsoft Windows [Version 10.0.18363.1139]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\[redacted]>
```

Figure 2: Microsoft windows command window screen

- In system command type (press enter after every command):
  1. pip3 install numpy
  2. pip3 install matplotlib
  3. pip3 install pandas
  4. pip3 install scipy
  5. pip3 install tk

b. Understanding the programme code

- **Simple user interaction**

These tasks are created for a beginner level in programming. Their intends are to get used with a new programming language. Besides the theoretical background knowledge already delivered in the course no additional knowledge is necessary. The tasks are to be solved only with the help of the course material and without the internet, unless stated differently.

- Console Output

Requirements:
The programmes task is to output the user’s family and first name, the age and address.
The programmes output has to be displayed in the console. The data has to be hard coded. No additional user input is required. To change the data the user has to change the programme code.
Required output:
First name: XXX

```
Family name: YYY
Age: DD.MM.YY
Address: XXYY
```

- Sulution:

```
firstName = "XXX"
familyName = "YYY"
Age = 23
Address = "XXXYYY"
print("First name:" + firstName)
print("Family name:" + familyName)
print("Age:" + str(Age))
print("Address:" + Address)
```

- **Console Input**

Requirements:

Update the programme from - so that user input from the console can be used to set the name, address and age on runtime.

Additional information:

- The function **input()** reads in user input from console till the return button is pressed.
- The data is stored as datatype string.
- User interaction will be marked by <<>>

Required output:

```
Please add first name: <<user adds in first name>>
Please add family name: <<user adds in first name>>
Please add age: <<user adds in first name>>
Please add address: <<user adds in first name>>
------------------------------
First name: XXX
Family name: YYY
Age: DD.MM.YY
Address: XXYY
-------------------------------
```

- Solution:

```
firstName = input("Please add first name: ")
familyName = input("Please add family name:  ")
Age = input("Please add age: ")
Address = input("Please add address: ")
print("------------------------------")
print("First name:" + firstName)
print("Family name:" + familyName)
print("Age:" + Age)
print("Address:" + Address)
```

c. Understanding the mechanism function call and modular programming

What is a function in Python?

In Python, a function is a group of related statements that performs a specific task.

Functions help break our programme into smaller and modular chunks. As our programme grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function

```
def function_name(parameters):
            """docstring"""
            statement(s)
```

Above shown is a function definition that consists of the following components.

- Keyword `def` that marks the start of the function header.
- A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of the function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
- An optional `return` statement to return a value from the function.

Example of a function

```
def greet(name):
    """    This function greets to  the person passed in as  a parameter"""
    print("Hello, " + name + ". Good morning!")
```

How to call a function in python?

Once we have defined a function, we can call it from another function, programme, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

Try running the above code in the Python programme with the function definition to see the output.

```
def greet(name):
    """    This function greets to  the person passed in as  a parameter"""
    print("Hello, " + name + ". Good morning!")
greet('Paul')
```

Note: In python, the function definition should always be present before the function call. Otherwise, we will get an error. For example,

```
# function call
greet('Paul')
# function definition
def greet(name):
    """    This function greets to  the person passed in as  a parameter"""
    print("Hello, " + name + ". Good morning!")
# Erro: name 'greet' is not defined
```

The return statement

The return statement is used to exit a function and go back to the place from where it was called.

Syntax of return

```
return [expression_list]
```

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

For example:

```
>>> print(greet("May"))
Hello, May. Good morning!
None
```

Here, `None` is the returned value since `greet()` directly prints the name and no `return` statement is used.

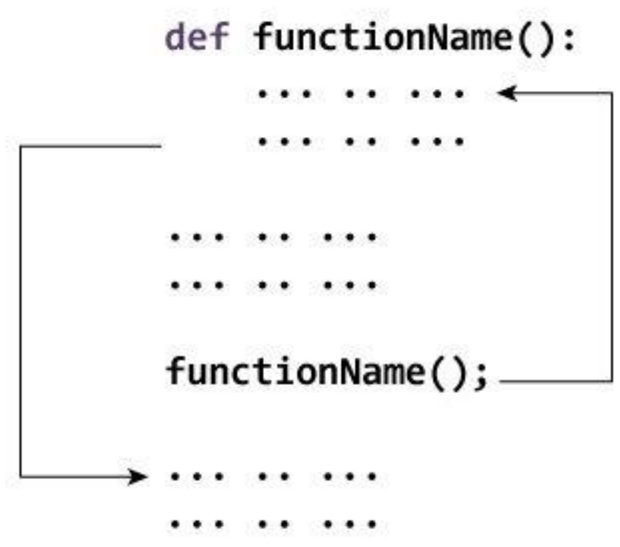
Example of return

```
def absolute_value(num):
    """    This function greets to  the person passed in as  a parameter"""
    if num >= 0:
        return num
    else:
        return -num
print(absolute_value(2))
print(absolute_value(-4))
```

Output

```
2
4
```

How Function works in Python?



- **Introducing Modular Programming**

Modular programming is an essential tool for the modern developer. Gone are the days when you could just throw something together and hope that it works. To build robust systems that last, you need to understand how to organize your programmes so that they can grow and evolve over time. *Spaghetti coding* is not an option. Modular programming techniques, and in particular the use of Python modules and packages, will give you the tools you need to succeed as a professional in the fast changing programming landscape.

In this chapter, we will:

- Look at the fundamental aspects of modular programming
- See how Python modules and packages can be used to organize your code
- Discover what happens when modular programming techniques are not used
- Learn how modular programming helps you stay on top of the development process
- Take a look at the Python standard library as an example of modular programming
- Create a simple programme, built using modular techniques, to see how it works in practice

Let's get started by learning about modules and how they work.

For most beginner programmers, their first Python programme is some version of the famous *Hello World* programme. This programme would look something like this:

```
print("Hello World!")
```

This one-line programme would be saved in a file on disk, typically named something like hello.py, and it would be executed by typing the following command into a terminal or command-line window:

```
python hello.py
```

The Python interpreter would then dutifully print out the message you have asked it to:

```
Hello World!
```

This hello.py file is called a **Python source file**. When you are first starting out, putting all your programme code into a single source file is a great way of organizing your programme. You can define functions and classes, and put instructions at the bottom which start your programme when you run it using the Python interpreter. Storing your programme code inside a Python source file saves you from having to retype it each time you want to tell the Python interpreter what to do.

As your programmes get more complicated, however, you'll find that it becomes harder and harder to keep track of all the various functions and classes that you define. You'll forget where you put a particular piece of code and find it increasingly difficult to remember how all the various pieces fit together.

Modular programming is a way of organizing programmes as they become more complicated. You can create a Python **module**, a source file that contains Python source code to do something useful, and then **import** this module into your programme so that you can use it. For example, your programme might need to keep track of various statistics about events that take place while the programme is running. At the end, you might want to know how many events of each type have occurred. To achieve this, you might create a Python source file named stats.py which contains the following Python code:

```
def init():

def event_occurred(event):

def get_stats():
```

The stats.py Python source file defines a module named stats —as you can see, the name of the module is simply the name of the source file without the .py suffix. Your main programme can make use of this module by importing it and then calling the various functions that you have defined as they are needed. The following frivolous example shows how you might use the stats module to collect and display statistics about events:

```
import stats
stats.init()
stats.event_occurred("meal_eaten")
stats.event_occurred("snack_eaten")
stats.event_occurred("meal_eaten")
stats.event_occurred("snack_eaten")
stats.event_occurred("meal_eaten")
stats.event_occurred("diet_started")
stats.event_occurred("meal_eaten")
stats.event_occurred("meal_eaten")
stats.event_occurred("meal_eaten")
stats.event_occurred("diet_abandoned")
stats.event_occurred("snack_eaten")
for event,num_times in stats.get_stats():
```

We're not interested in recording meals and snacks, of course—this is just an example—but the important thing to notice here is how the stats module gets imported, and then how the various functions you defined within
the stats.py file get used. For example, consider the following line of code:

```
stats.event_occurred("snack_eaten")
```

Because the event_occurred() function is defined within the stats module, you need to include the name of the module whenever you refer to this function.

- **Note**

There are ways in which you can import modules so you don't need to include the name of the module each time. We'll take a look at this in Chapter 3, *Using Modules and Packages*, when we look at namespaces and how
the import command works in more detail.

As you can see, the import statement is used to load a module, and any time you see the module name followed by a period, you can tell that the programme is referring to something (for example, a function or class) that is defined within that module.

Example

```
from math import pi
r = float(input(""))
A = r**2 * pi
print("The area of the circle is: " +str(A))
```

1.3 Basics of Object - Oriented Programming

a. Procedural programming vs. object-oriented programming

Object-oriented programming and procedural programming both are used to develop the applications. Both of them are high-level programming languages. These two are important concepts, and it is also important to know the difference between them.

S.no.	On the basis of	Procedural Programming	Object-oriented programming
1.	Definition	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
2.	Security	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3.	Approach	It follows a top-down approach.	It follows a bottom-up approach.
4.	Data movement	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5.	Orientation	It is structure/procedure-oriented.	It is object-oriented.
6.	Access modifiers	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7.	Inheritance	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8.	Code reusability	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.

S.no.	On the basis of	Procedural Programming	Object-oriented programming
9.	Overloading	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
10.	Importance	It gives importance to functions over data.	It gives importance to data over functions.
11.	Virtual class	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12.	Complex problems	It is not appropriate for complex problems.	It is appropriate for complex problems.
13.	Data hiding	There is not any proper way for data hiding.	There is a possibility of data hiding.
14.	Programme division	In Procedural programming, a programme is divided into small programmes that are referred to as functions.	In OOP, a programme is divided into small parts that are referred to as objects.
15.	Examples	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

Table 1: Comparison of procedural programming vs. object-oriented programming

b. Difference between classes and objects

- **What is Class?**

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object. It provides initial values for member variables and member functions or methods.

- **What is Object?**

An object is nothing but a self-contained component that consists of methods and properties to make a data useful. It helps you to determines the behavior of the class.

For example, when you send a message to an object, you are asking the object to invoke or execute one of its methods.

From a programming point of view, an object can be a data structure, a variable, or a function that has a memory location allocated. The object is designed as class hierarchies.

Here is the important difference between class and object:

Class	**Object**
A class is a template for creating objects in programme.	The object is an instance of a class.
A class is a logical entity	Object is a physical entity
A class does not allocate memory space when it is created.	Object allocates memory space whenever they are created.
You can declare class only once.	You can create more than one object using a class.
Example: Car.	Example: Jaguar, BMW, Tesla, etc.
Class generates objects	Objects provide life to the class.
Classes can't be manipulated as they are not available in memory.	They can be manipulated.
It doesn't have any values which are associated with the fields.	Each and every object has its own values, which are associated with the fields.
You can create class using "class" keyword.	You can create object using "new" keyword in Java

Table 2: Comparison of Class and Object

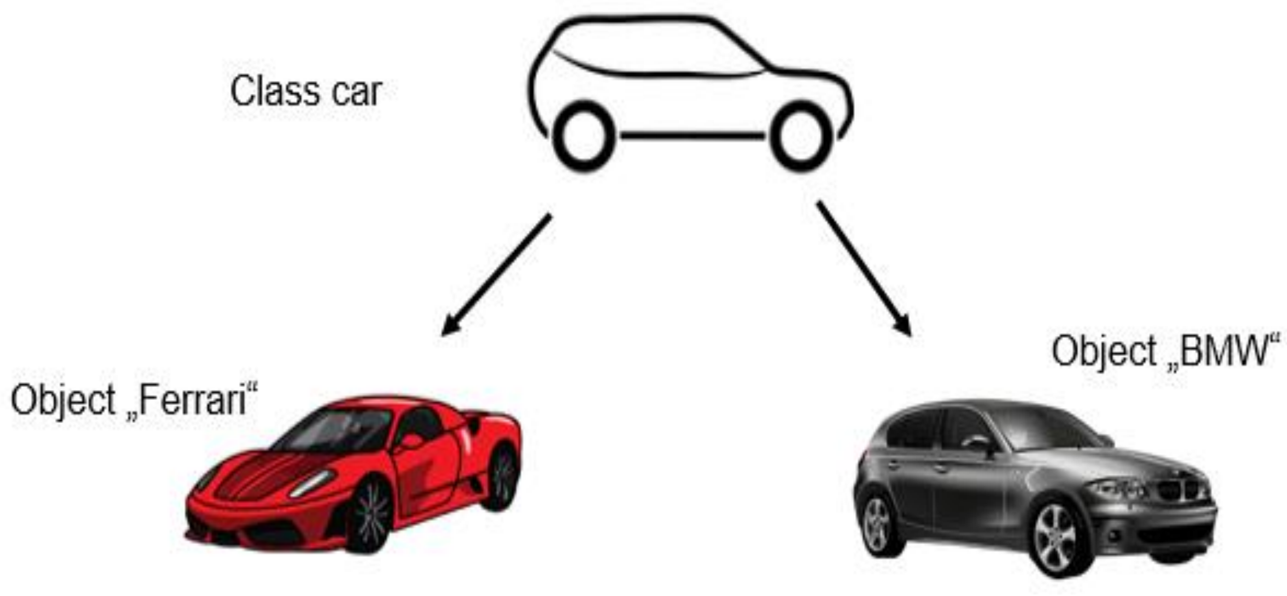
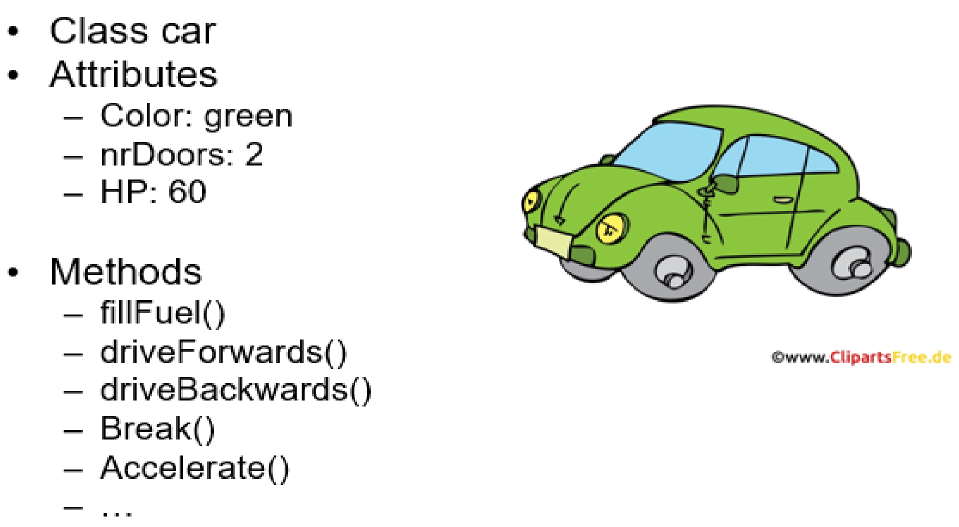


Figure 3: Demonstration between “Class” and “Object”

- Class members
  - Attributes
    - Variables
    - „the objects properties“
    - May serve as a „memory“
  - Methods
    - Functions
    - „defines what a object is capable of doing“
- OOP Example



- Class car
- Attributes
  - Color: green
  - nrDoors: 2
  - HP: 60
- Methods
  - fillFuel()
  - driveForwards()
  - driveBackwards()
  - Break()
  - Accelerate()
  - …

©www.ClipartsFree.de

Figure 4: OOP Example

c. Security aspect access rights for classes

- Accessibility of class members
  - Data encapsulation
  - Controls access on attributes and methods
  - Method of data security
  - Creates defined intersections to use a class Possibilities

    + Public: : Access always possible

    # Private: Access from parent class and derived classes possible

    - Protected: Access only possible from parent class

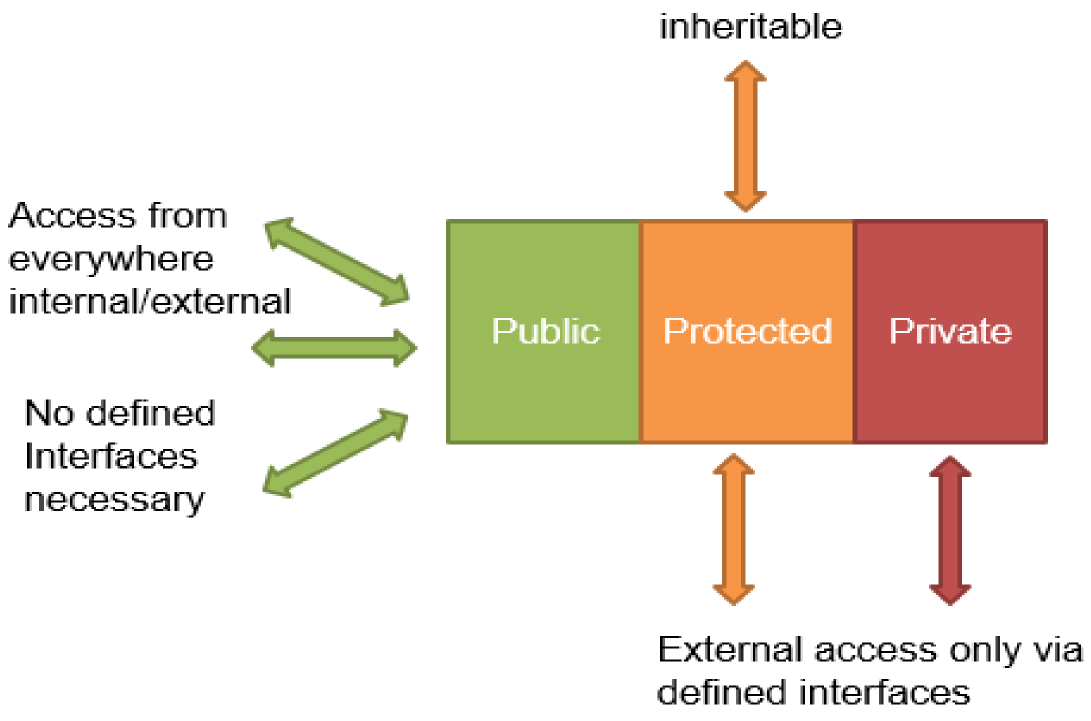
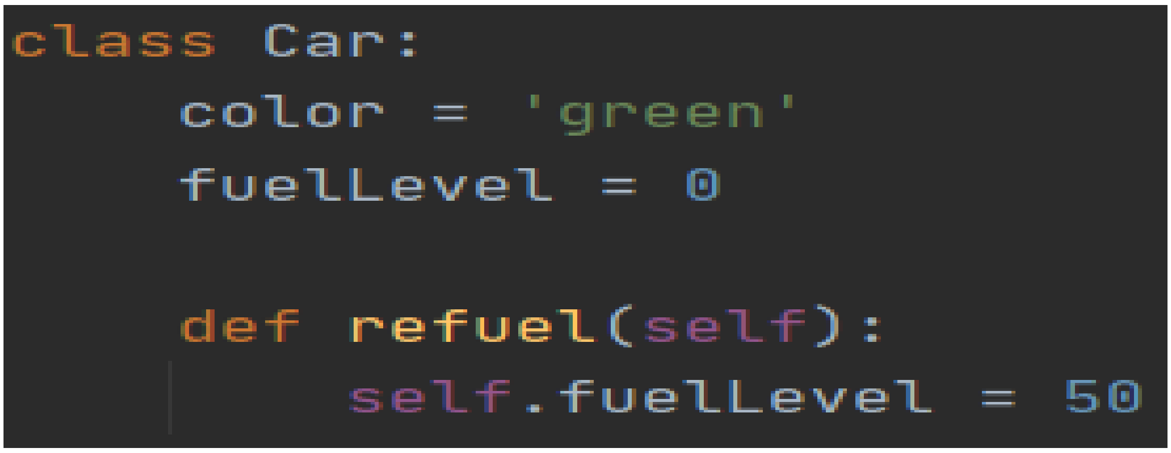

inheritable
Access from everywhere internal/external
No defined Interfaces necessary
Public
Protected
Private
External access only via defined interfaces

Chart 6: Class Possibilities

Get and set method:

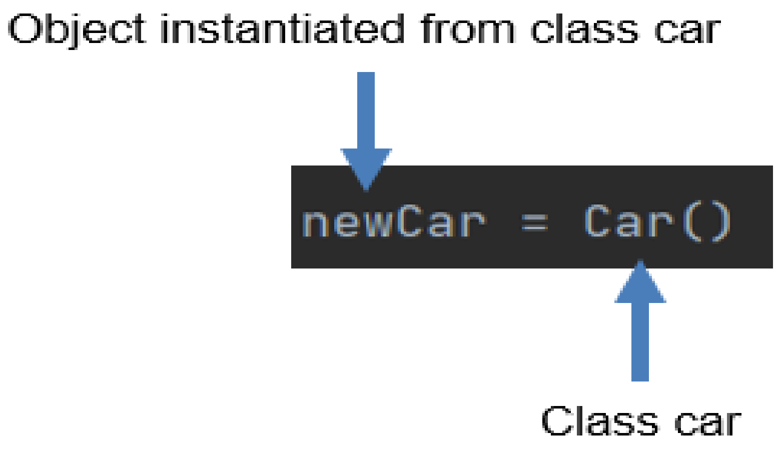
- Public function
- Defined Interface
- Access private and protected members

d. Creation of constructor/structure for classes



```
class Car:
    color = 'green'
    fuelLevel = 0

    def refuel(self):
        self.fuelLevel = 50
```

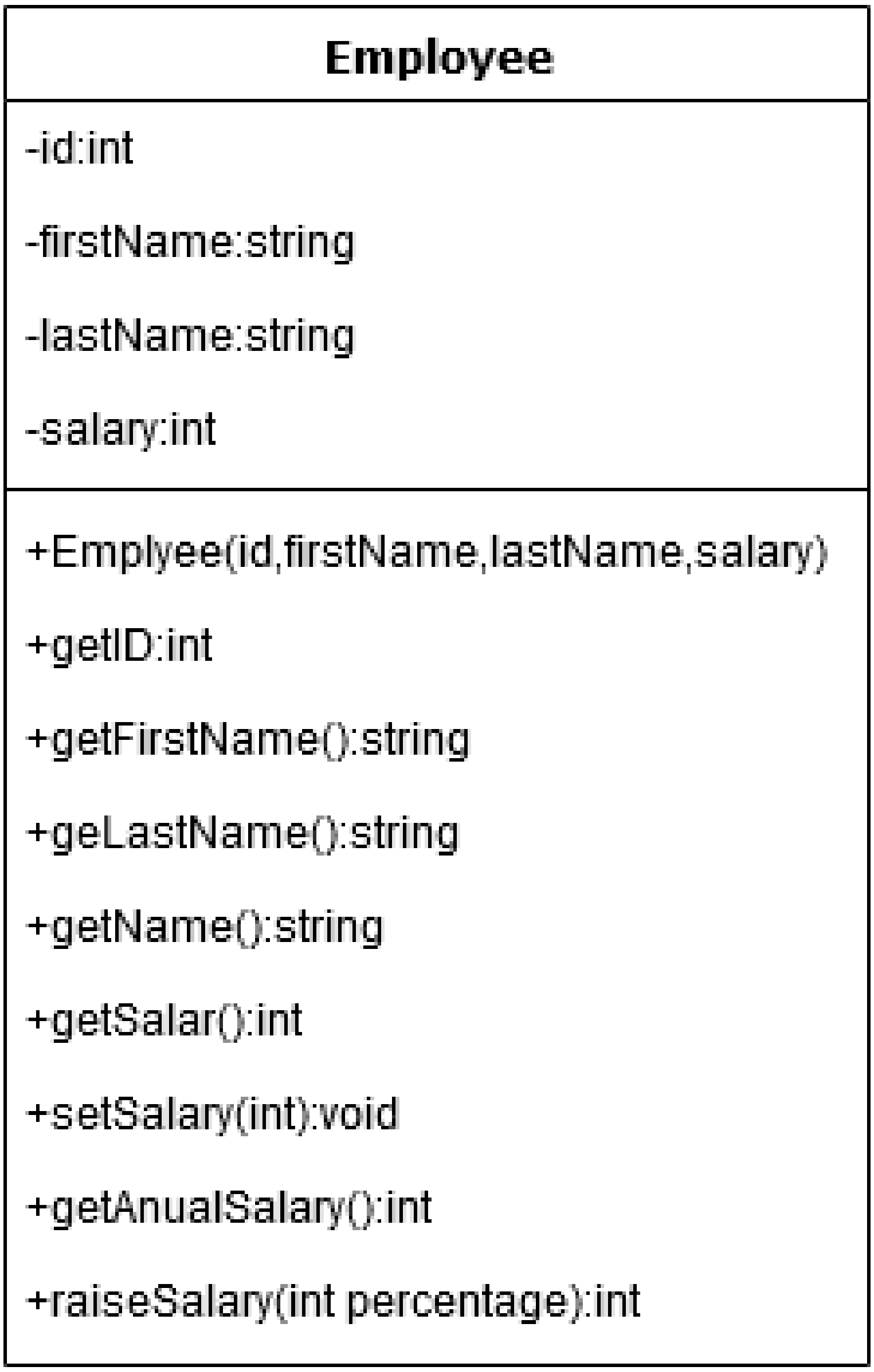
- Example of instantiate object



```
newCar = Car()
```

Example

Create a class using the given UML diagram below. All getter methods return the corresponding attributes. Setter methods with data type void do not return any value. The method getName shall return the full name with “First name last name”. The method raise percentage shall return the new salary raised by the percentage.



Employee

-id:int

-firstName:string

-lastName:string

-salary:int

+Emplyee(id,firstName,lastName,salary)

+getID:int

+getFirstName():string

+geLastName():string

+getName():string

+getSalar():int

+setSalary(int):void

+getAnualSalary():int

+raiseSalary(int percentage):int

Solution:

```
class Employee:
    def __init__(self, id, firstName, lastName, salary):
        self._id = id
        self._firstName = firstName
        self._lastName = lastName
        self._salary = salary

    def getID(self):
        return self._id
```

```
def getFirstName(self):
    return self._firstName

def getLastName(self):
    return self._lastName

def getName(self):
    return self._firstName + " " + self._lastName

Creation of constructor/structure for classes def getSalary(self):
    return self._salary

def setSalary(self, value):
    self._salary = value

def getAnnualSalary(self):
    return self._salary * 12

def raisePercentage(self, percentage):
    self._salary = self._salary * percentage / 100 + self._salary
    return self._salary
```

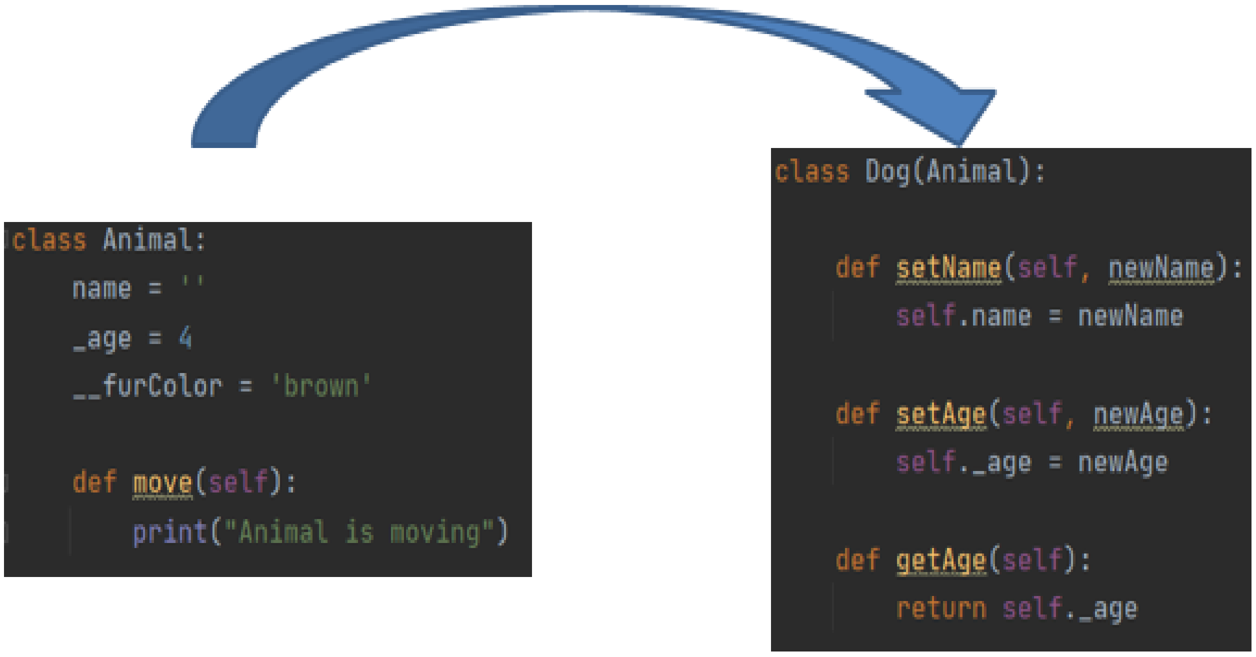
e. Creating base classes for inheritance

- ***Introduction***

Inheritance is one of the most important aspects of Object-oriented programming (OOP). The key to understanding Inheritance is that it provides code re-usability. In place of writing the same code, again and again, we can simply inherit the properties of one class into the other.

f. Understanding inheritance structures and mechanisms

- Child class inherits members from parent class
- Prevent redundant code
- Modularization
- Protected not inherited
    - Inheritance Python



```python
class Animal:
    name = ''
    _age = 4
    __furColor = 'brown'

    def move(self):
        print("Animal is moving")
```

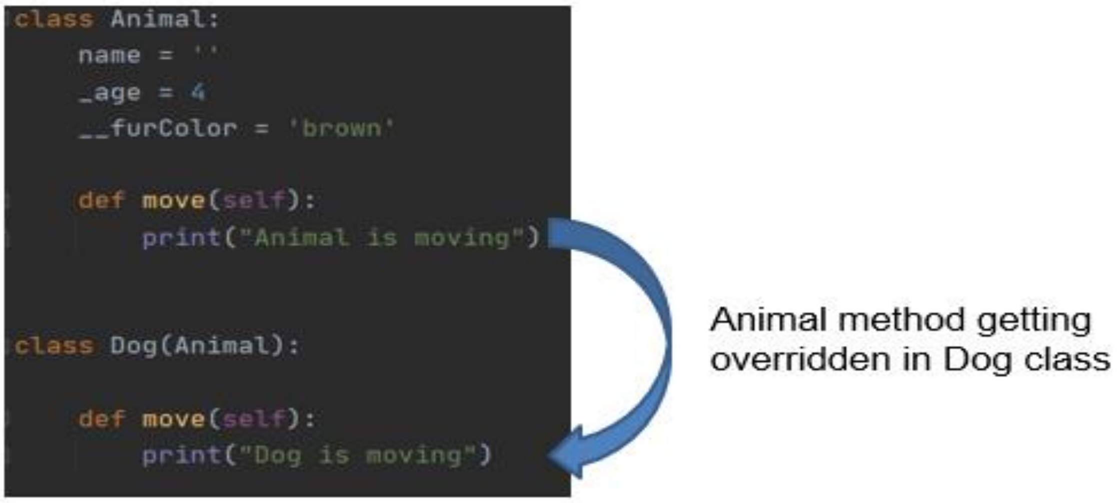
```python
class Dog(Animal):

    def setName(self, newName):
        self.name = newName

    def setAge(self, newAge):
        self._age = newAge

    def getAge(self):
        return self._age
```

Figure 5: Inheritance Python

- Inheritance – Overriding

- Overriding
    1. New of method from base class
    2. Add functionality
    3. Change functionality
    4. Adaptation of methods to new class
- Creation in base class as remembrance

- Example overriding



```python
class Animal:
    name = ''
    _age = 4
    __furColor = 'brown'

    def move(self):
        print("Animal is moving")


class Dog(Animal):

    def move(self):
        print("Dog is moving")
```

Figure 6: Inheritance – Overriding

1.4. Extended concepts

a. Implementation of file input and output using object-oriented structures

- **DataIO**

- **Reading data from**
    1. **Files**
    2. **Streams**
    3. **Save data to files**

Figure 7: Logo of pandas

- **Popular library pandas**
    - **Interne link:** https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/01_table_oriented.html

- **Pandas Dataframe**

- **Structured like table**
    1. **Rows**
    2. **Columns**
- **Accessible via name**
- **Statistical Information**

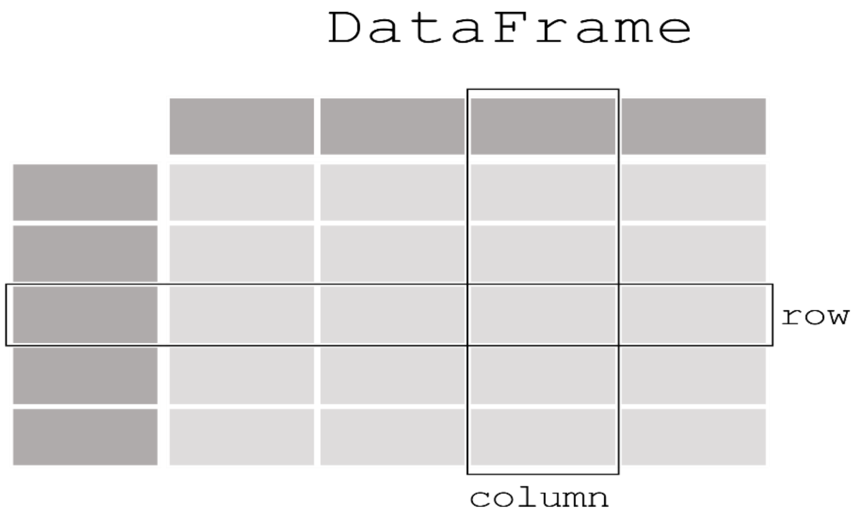
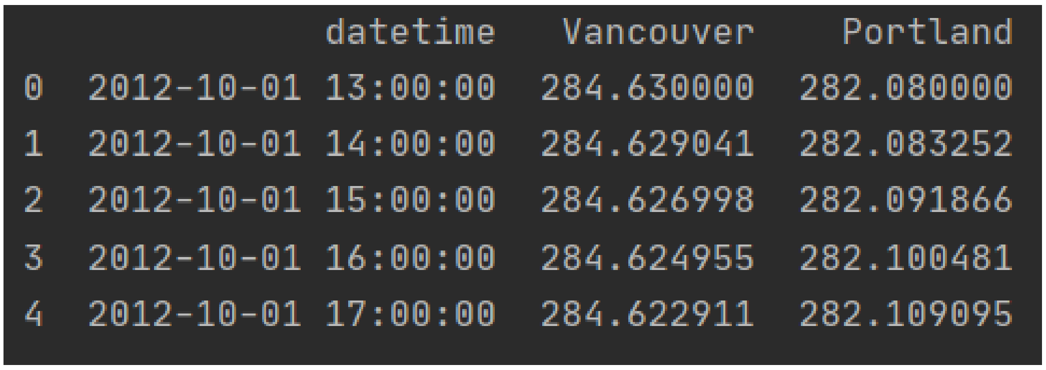


Figure 8: Sample of Pandas Dataframe

- **Sample data frame in python**



```
              datetime   Vancouver    Portland
0  2012-10-01 13:00:00  284.630000  282.080000
1  2012-10-01 14:00:00  284.629041  282.083252
2  2012-10-01 15:00:00  284.626998  282.091866
3  2012-10-01 16:00:00  284.624955  282.100481
4  2012-10-01 17:00:00  284.622911  282.109095
```

Figure 9: Sample of Python Dataframe

- Comma Seperated Values – CSV

- File type especially for sensor data
- Easy file format
    1. Editable per hand
- Values separated via comma
    1. Depends on country setting
    2. German semicolon
    3. American comma
    4. Can be read directly in excel

- Using pandas library

- Installing pandas: pip3 install pandas
- Import pandas as class : import pandas as pd
- Using member functions: df = pd.readcsv(“filename.csv”)
- Store dataframe in variable: df.tocsv(“filename.csv”)

b. Preparation and local visualization of data using object-oriented methods

- **Matplotlib**

- Library for plotting and visualizing data
- Compatible with many other libraries
- Used in many other libraries
- Website: https://matplotlib.org/tutorials/index.html

- **Data visualization**

- Data is displayed in figures Figure
    - Defined area
    - Configurable
    - Contain plot
- Plot
    - Diagram used to display data

- **Parts of figure**

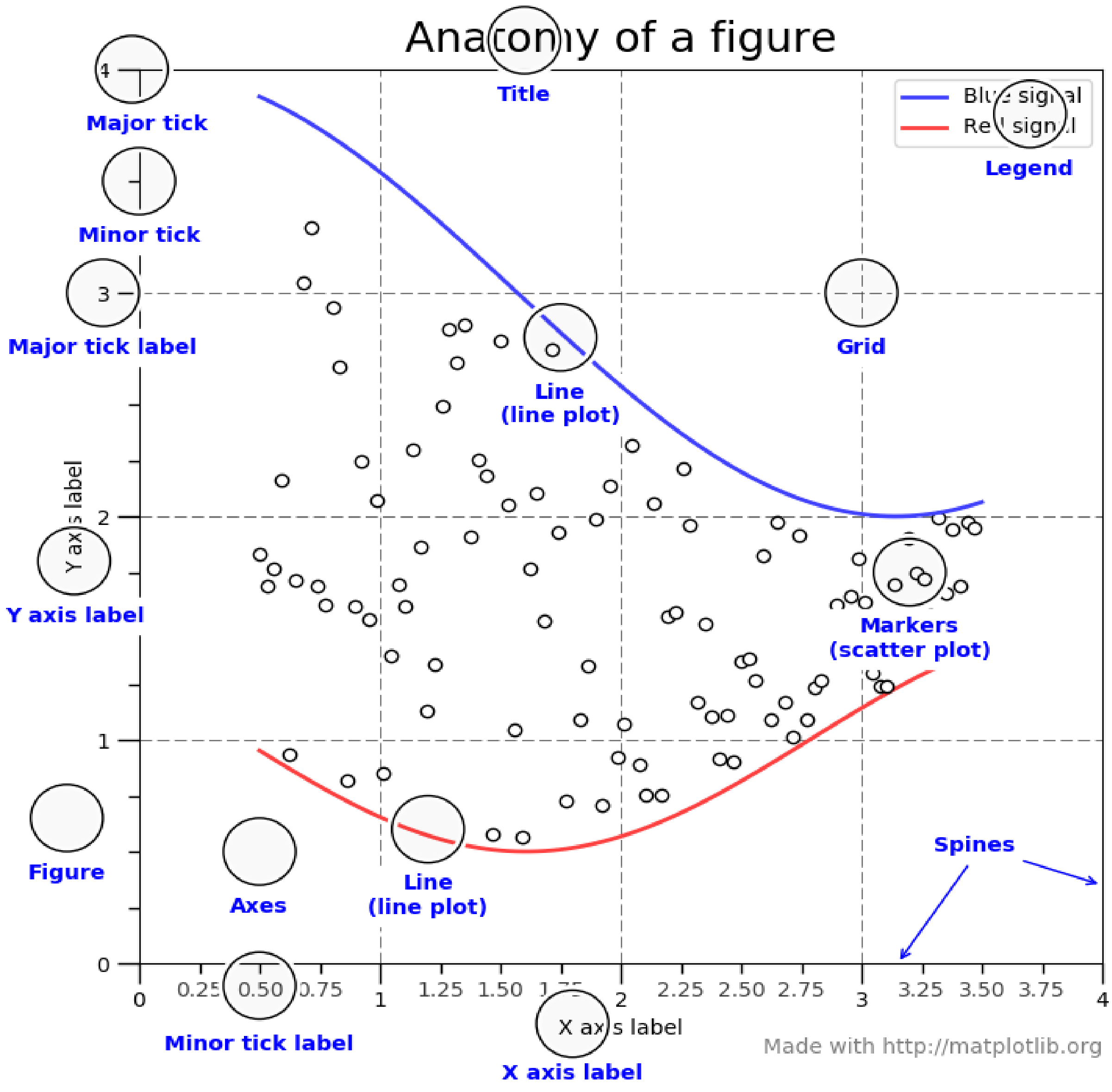


Figure 10: Explanation of parts of figure

- **Creation of graphical user interfaces**

- Installing Matplotlib: pip3 install Matplotlib
- Import pandas as class: import Matplotlib as mpl

- **Example**

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```

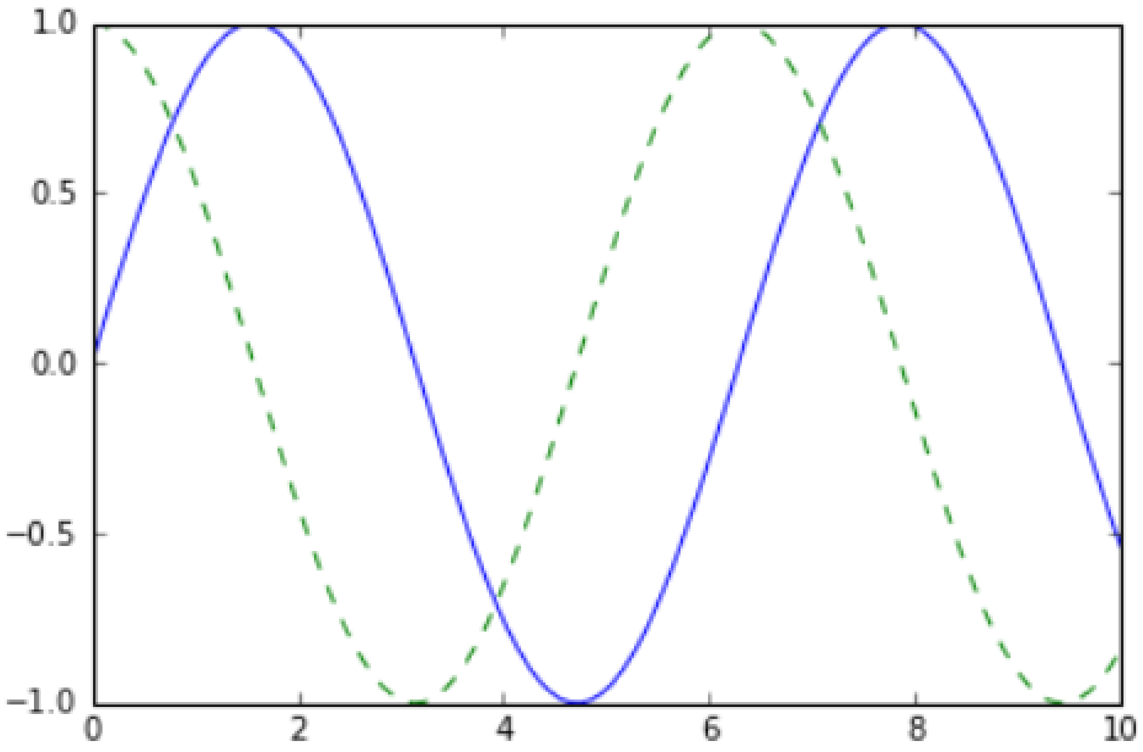


Figure 11: Graphical user interfaces

2. Unified Modeling Language (UML)

2.1 What is UML?

It is the general-purpose modeling language used to visualize the system. It is a graphical language that is standard to the software industry for specifying, visualizing, constructing, and documenting the artifacts of the software systems, as well as for business modeling.

Benefits of UML:
Simplifies complex software design, can also implement OOPs like a concept that is widely used.

It reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
It makes communication more clear and more real.
It helps to acquire the entire system in a view.
It becomes very much easy for the software programmer to implement the actual demand once they have a clear picture of the problem.

Types of UML: The UML diagrams are divided into two parts: Structural UML diagrams and Behavioral UML diagrams which are listed below:

Structural UML diagrams
Class diagram
Package diagram
Object diagram
Component diagram
Composite structure diagram
Deployment diagram
Behavioral UML diagrams
Activity diagram
Sequence diagram
Use case diagram
State diagram
Communication diagram
Interaction overview diagram
Timing diagram

2.2. Class diagrams to describe software requirements and documentation

UML class diagrams: Class diagrams are the main building blocks of every object-oriented method. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the class diagram has an appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in their context. It describes various kinds of objects and the static relationship between them.

The main purpose to use class diagrams are:
This is the only UML that can appropriately depict various aspects of the OOPs concept.

Proper design and analysis of applications can be faster and efficient.
It is the base for deployment and component diagram.
There are several software available that can be used online and offline to draw these diagrams Like Edraw max, lucid chart, etc. There are several points to be kept in focus while drawing the class diagram. These can be said as its syntax:
Each class is represented by a rectangle having a subdivision of three compartments name, attributes, and operation.

There are three types of modifiers that are used to decide the visibility of attributes and operations.
+ is used for public visibility(for everyone)
# is used for protected visibility (for friend and derived)
– is used for private visibility (for only me)

Below is the example of Animal class (parent) having two child class as dog and cat both have object d1, c1 inheriting properties of the parent class.

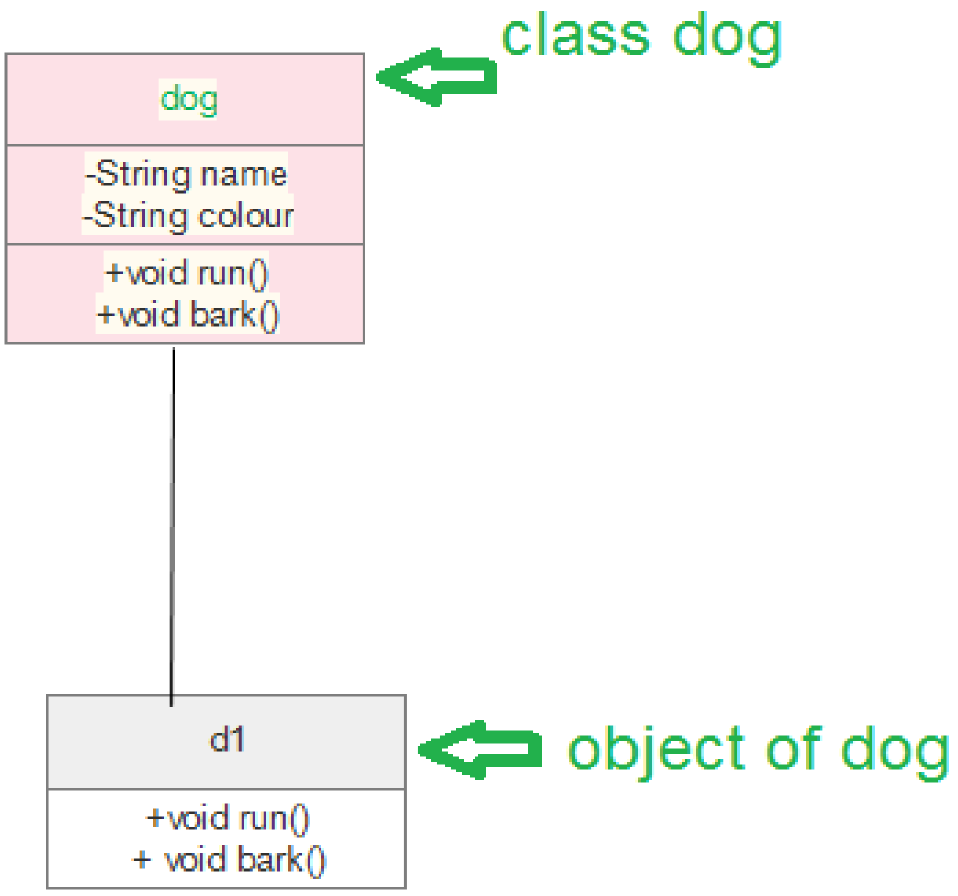


Chart 7: Example of inheriting properties of the parent class

The class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but are also used to construct the executable code for forward and reverse engineering of any system.

The class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, a class diagram is generally used for construction purpose.
In a nutshell, it can be said, class diagrams are used for −

Describing the static view of the system.
Showing the collaboration among the elements of the static view.

Describing the functionalities performed by the system.
Construction of software applications using object-oriented languages.

```
import java.io.*;
class GFG {
public static void main(String[] args)
{
dog d1 = new dog();
d1.bark();
d1.run();
cat c1 = new cat();
c1.meww();
}
}
class Animal {
public void run()
{
String name;
String colour;
System.out.println("animal is running");
}
}
class dog extends Animal {
public void bark()
{
System.out.println("wooh!wooh! dog is barking");
}
public void run()
{
System.out.println("dog is running");
}
}
class cat extends Animal {
public void meww()
{
System.out.println("meww! meww!");
}
}
```

The process to design class diagram: In Edraw max (or any other platform where class diagrams can be drawn) follow the steps:

- Open a blank document in the class diagram section.
- From the library select the class diagram and click on create option.
- Prepare the model of the class on the opened template page.
- After editing according to requirement save it.

There are several diagram components that can be efficiently used while making/editing the model. These are as follows:

- Class { name, attribute, method}
- Objects
- Interface
- Relationships {inheritance, association, generalization}
- Associations {bidirectional, unidirectional}

Class diagrams are one of the most widely used diagrams in the fields of software engineering as well as business modeling.

2.3. Display and application of class relations

Association

An association can be labeled by placing an association name in the middle of the association or by placing the role name either or both ends of the association

If no association name or role name is specified, then the default association name ‘has’ is assumed

Aggregations

Aggregation is an asscociation in which one class belongs to a collection

Example: Order has collection of orderDetails

It is represented by diamond symbol placed next to the aggregate
Aggretion are special asscociations that represent a ‘has a’ or a ‘whole/part’ relationship among peers

Composition

Sometimes an aggregation relation may be a strong aggregation
The parts cannot have a life of their own
It means that if the aggregate is destroyed, then the parts also destroyed
A strong aggregation is also known as a composition
A strong aggregation is represented by a solid diamond symbol

2.4. Display of inheritance trees for software documentation

Inheritance overview: trees. To assess the size and complexity of the system, we build an inheritance tree. We use as node size the number of instance variables (width) and the number of methods (height), while the colour tone represents the lines of code of the class.

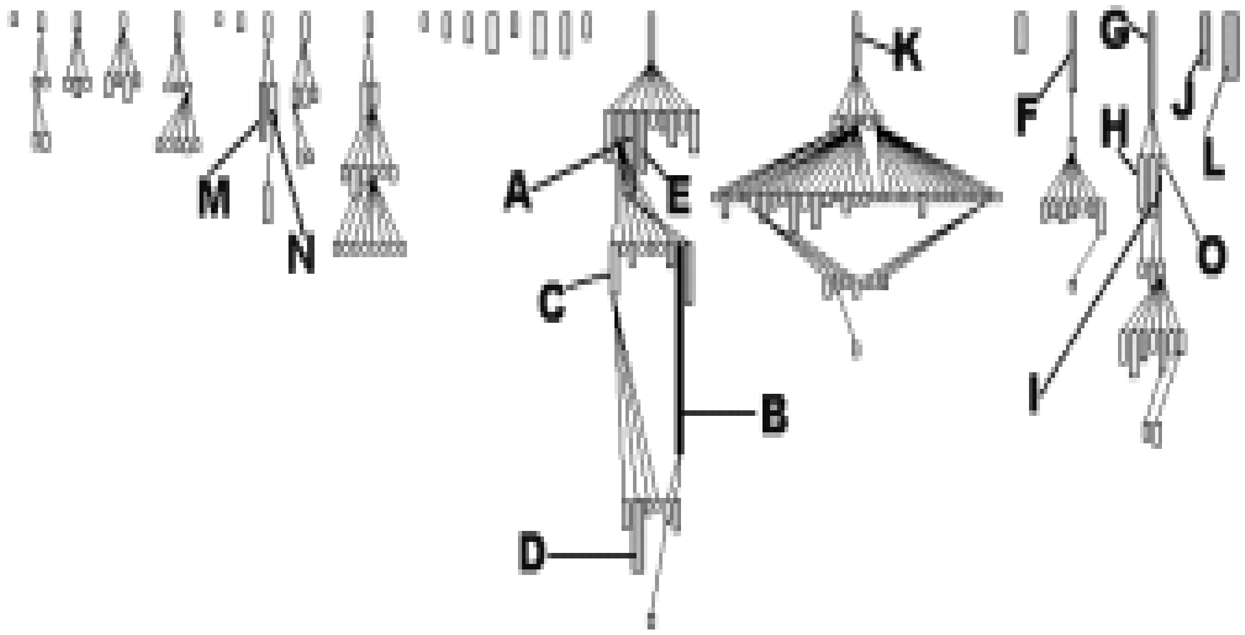


Figure 12. Inheritance Overview via Tree; node width = NIV, node height = NOM and colour = WLOC.

Interpretation. We observe a few main hierarchies with a high proportion of very small classes. Then, using the number of methods per class as a criterium we identify some
candidate classes for further investigation: (1) the smallest class (O) is completely empty and (2) 13 classes are quite large (between 40 and 175 methods) (B).

These 13 classes can be classified according to their po-sition in the inheritance tree: being a leaf (D,I), being on top of a hierarchy (F,G,K), being in the middle of the hier-
archy (A,B) or being alone (E,J,L). Large sibling classes like (H,I) and (N,M) are good candidates for refactoring analysis because some of the code may be moved up in their
common superclass. An example of possible further investigation is the large class called BrowserNavigator that implements 175 meth-ods (marked B) whereas its superclass Navigator (A) al-ready implements 70 methods. Another interesting case is
the class called BRScanner (named L), which implements 49 methods and defines 14 instance variables

2.5. Sequence diagrams to describe communication processes of software (PC and mechatronic systems)

Sequence Diagrams –A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence Diagram Notations
Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

We use actors to depict various roles including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram.

For example – Here the user in seat reservation system is shown as an actor where it exists outside the system and is not a part of the system
Lifelines – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name: Class Name

We display a lifeline in a rectangle called head with its name and type. The head is located on top of a vertical dashed line (referred to as the stem) as shown above. If we want to model an unnamed instance, we follow the same pattern except now the portion of lifeline's name is left blank.

Difference between a lifeline and an actor – A lifeline always portrays an object internal to the system whereas actors are used to depict objects external to the system. The following is an example of a sequence diagram:
Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.
Asynchronous Messages

Delete Message
Self Message.
Reply Message.
Found Message
Lost Message.

Guards – To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

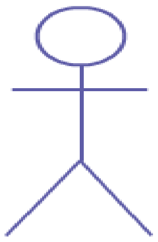
Uses of sequence diagrams –
Used to model and visualise the logic behind a sophisticated function, operation or procedure.
They are also used to show details of UML use case diagrams.
Used to understand the detailed functionality of current or future systems.
Visualise how messages and tasks move between objects or components in a system.

2.6. Use Case Diagram for high level documentation of use cases

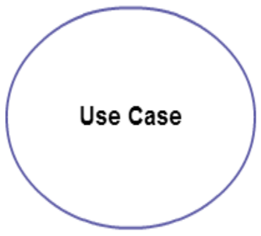
Use case diagram is a behavioral UML diagram type and frequently used to Analyse various systems. They enable you to visualize the different types of roles in a system and how those roles interact with the system. This use case diagram tutorial will cover the following topics and help you create use cases better.
Importance of Use Case Diagrams
As mentioned before use case diagrams are used to gather a usage requirement of a system. Depending on your requirement you can use that data

in different ways. Below are few ways to use them.
To identify functions and how roles interact with them – The primary purpose of use case diagrams.
For a high-level view of the system – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
To identify internal and external factors – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

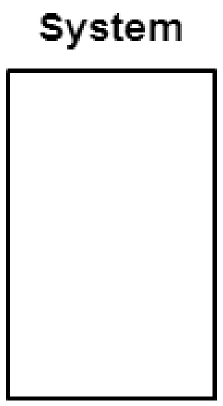
Use Case Diagram objects
Use case diagrams consist of 4 objects.
Actor
Use case
System
Package
The objects are further explained below.



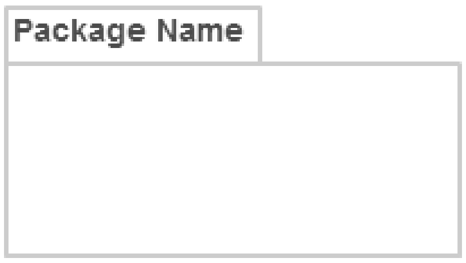
Actor
Actor in a use case diagram is any entity that performs a role in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



Use Case
A use case represents a function or an action within the system. It's drawn as an oval and named with the function.



System
The system is used to define the scope of the use case and drawn as a rectangle. This an optional element but useful when you're visualizing large systems. For example, you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.



Package
The package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are used to group together use cases. They are drawn like the image shown below.

2.7. Implementation of UML diagrams

2.7.1 Creation of classes using UML diagrams

Up to now, you've learned about objects, relationships and guidelines that are critical when drawing use case diagrams. I'll explain the various processes using a banking system as an example.

Identifying Actors
Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you're trying to show in the use case. An example of an external organization can be the tax authority or the central bank. The loan processor is a good example of an external system associated as an actor.

Identifying Use Cases
Now it's time to identify the use cases. A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete function required by an actor. You can extend or include use cases depending on the complexity of the system.
Once you identify the actors and the top level use case you have a basic idea of the system. Now you can fine tune it and add extra layers of detail to it.
Look for Common Functionality to use Include
Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that it's always called when the original use case is executed. (see the diagram for an example).

Is it Possible to Generalize Actors and Use Cases

There may be instances where actors are associated with similar use cases while triggering a few use cases unique only to them. In such instances, you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

One of the best examples of this is “Make Payment” use case in a payment system. You can further generalize it to “Pay by Credit Card”, “Pay by Cash”, “Pay by Check” etc. All of them have the attributes and the functionality of payment with special scenarios unique to them.

Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases, you can use the extend relationship and attach an extension rule to it. In the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

Extend doesn’t always mean it’s optional. Sometimes the use case connected by extending can supplement the base use case. The thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.

2.7.2. Implementation of sequence diagrams for defined tasks

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

A Quick Overview of the Various Parts of a Sequence Diagram

Lifeline Notation

A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

Message Arrows

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received

Synchronous message

Asynchronous message

Return message

Participant creation message

Participant destruction message

Reflexive message

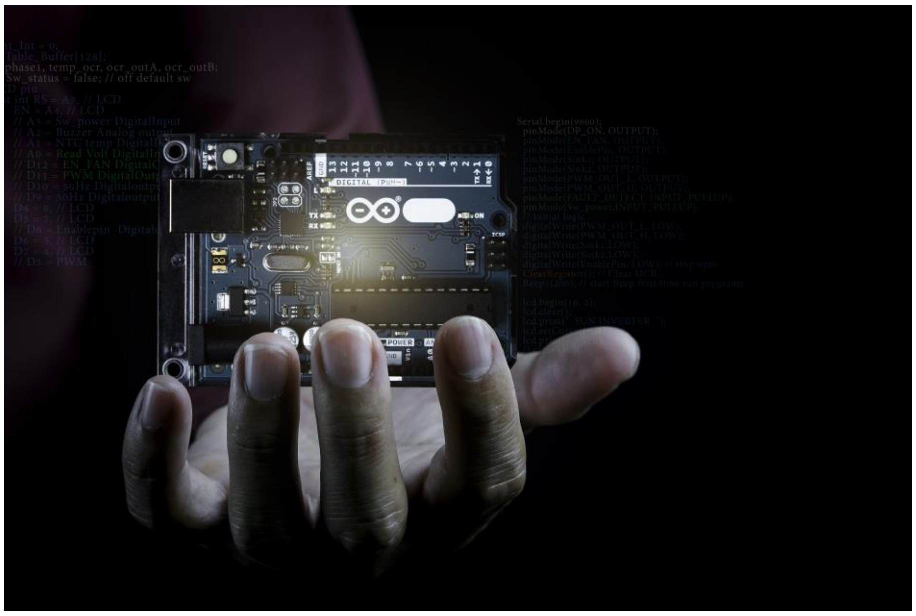


Illustration 4: electronic control board element

2.7.3 Creating Use Case Diagrams for sample applications

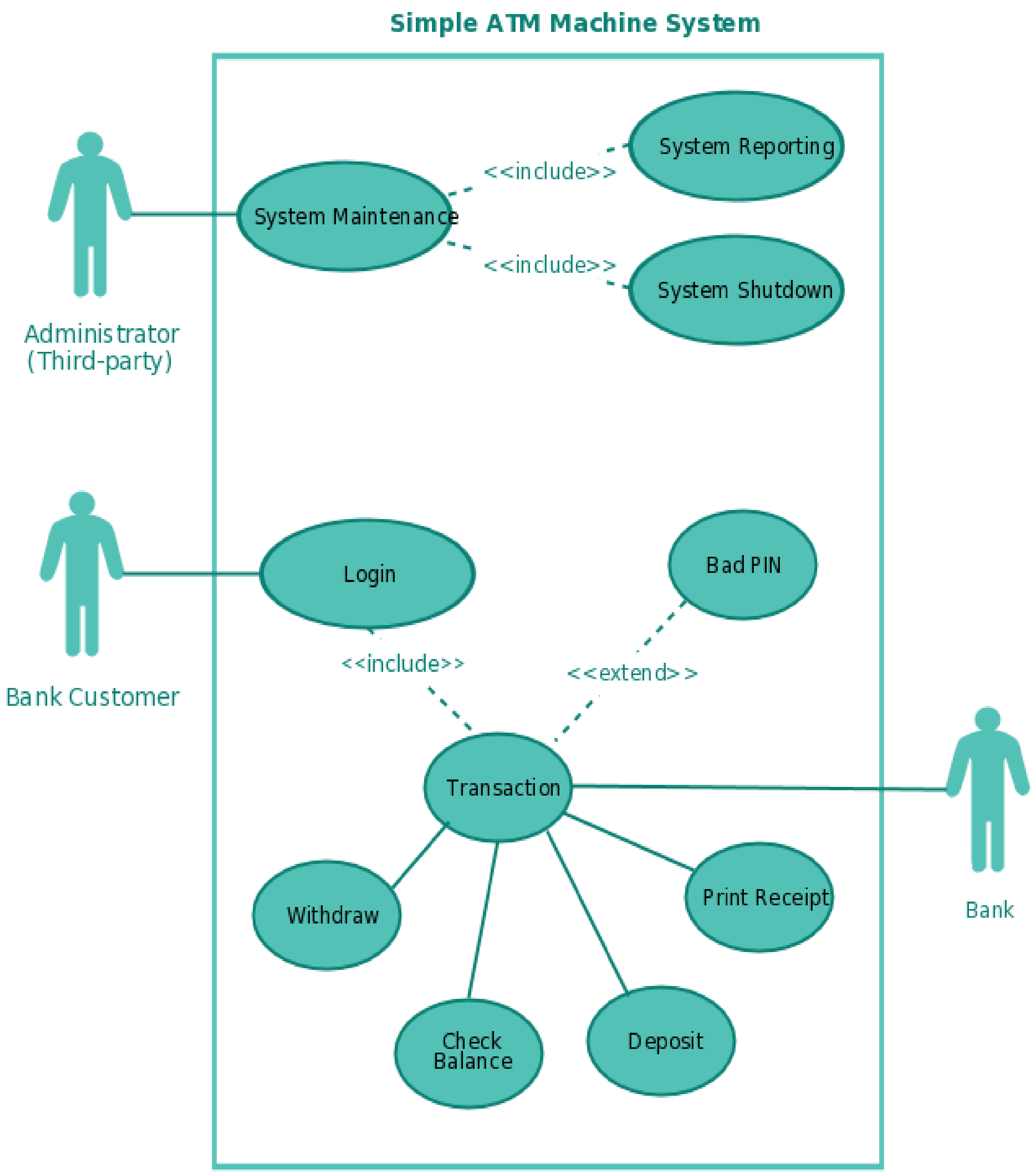


Figure 13: ATM machine system illustration

We've gone ahead and created use case diagram templates for some common scenarios. Although your problem or scenario won't be exactly like this you can use them as a starting point.

3. Analyse tasks and find solutions

Analysis of technical orders and development of solutions:
Analysing customer requirements with regard to the required function, Clarify specifications in exchange with customers.

Example: QR Code Generation using Python

QR code stands for Quick Response Code. QR codes may look simple but they are capable of storing lots of data. Irrespective of how much data they contain when scanned QR code allows the user to access information instantly. That is why they are called Quick Response Code.
These are being used in many scenarios these days. It first appeared in Japan in 1994. QR codes can be used to store(encode) lots of data and that too of various types. For example, they can be used to encode:

i. Contact details
ii. Facebook ids, Instagram ids, Twitter ids, WhatsApp ids, and more.
iii. Event Details
iv. Youtube links
v. Product details
vi. Link directly to download an app on the Apple App Store or Google Play.
vii. They are also being used in doing digital transactions by simply scanning QR codes.
viii. Access Wi-Fi by storing encryption details such as SSID, password, and encryption type.

This list goes on….!

We just saw some advantages of QR codes. Now we will learn here how we can generate QR codes in Python.
For QR code generation using python, we are going to use a python module called **QRcode.**
Link: https://pypi.org/project/qrcode/
Install it using this command: **pip install qrcode**
We will generate a QR Code for encoding the youtube link and we will also explore more. QR code generation is simple. Just pass the text, link, or any content to *'make'* function of QRcode module.

```
import qrcode
img = qrcode.make("https://www.youtube.com/")
img.save("youtubeQR.jpg")
```

On executing this code output is:

You can scan it and verify.

You can see it’s just 3 lines of code to generate this QR Code. One more thing to mention is that it’s not necessary that you always have to give a link to qrcode.make() function. You can provide simple text also.

For example:

You can encode: India is a country with many religions. I love India.

Let’s try it out:

```
import qrcode
img = qrcode.make("India is a country with many religions. I love India.")
img.save("youtubeQR.jpg")
```

Output QR Code for this text is:

Scan it from your mobile and you will get the content.

So this is the one part, which involves generating a QR Code and scanning it. But what if we want to read this QR Code i.e., now we want to know what was encoded in the QR Code without scanning it. For this, we will use OpenCV. OpenCV is a library of programming functions focused on real-time computer vision tasks.

Install opencv: **pip install opencv-python**
Code to decode a QR code back to know the original string.

```
import cv2
d = cv2.QRCodeDetector()
val, _, _ = d.detectAndDecode(cv2.imread("myQRCode.jpg"))
print("Decoded text is: ", val)
```

Output:

```
India is a country with many religions. I love India.
```

4. Test software and rollout

4.1. Clarification of the V – Model

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

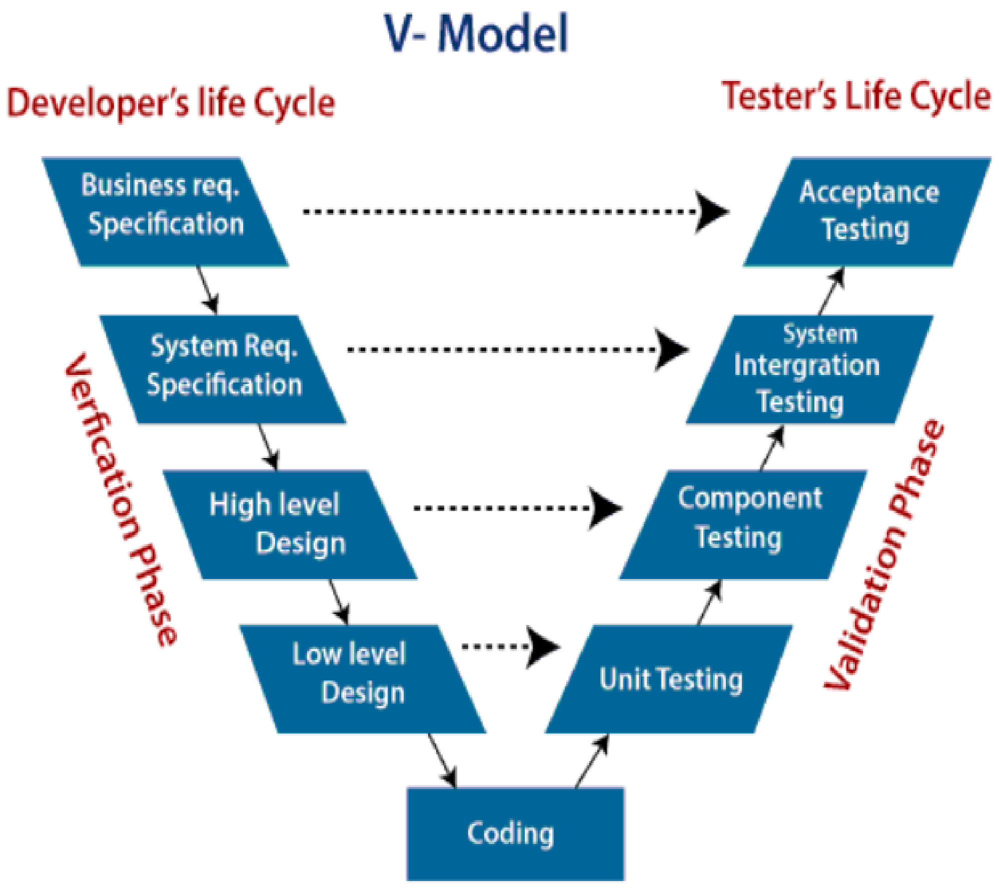


Figure 14: Illustration of Verification and Validation Model

Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.
Validation: It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.
So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

There are the various phases of Verification Phase of V-model:

i. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
ii. **System Design:** In this stage system engineers Analyse and interpret the business of the proposed system by studying the user requirements document.
iii. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
iv. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
v. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

i. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a programme module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.

ii. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.

iii. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.

iv. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

- Easy to Understand.
- Testing Methods like planning, test designing happens well before coding.
- This saves a lot of time. Hence a higher chance of success over the waterfall model.
- Avoids the downward flow of the defects.
- Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

- Very rigid and least flexible.
- Not a good for a complex project.
- Software is developed during the implementation stage, so no early prototypes of the software are produced.
- If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

4.2. Software Testing

What is Software Testing

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

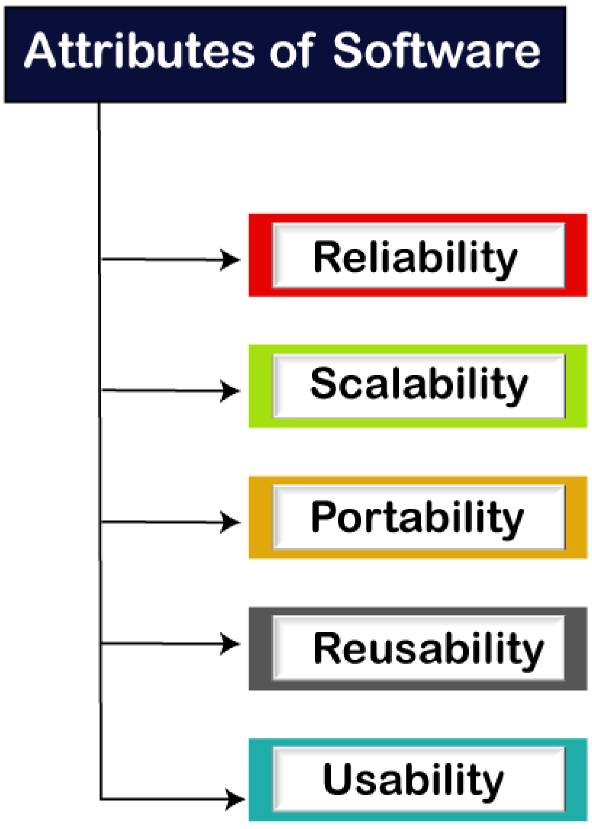


Chart 8: Process of software testing

Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

What is Testing

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application.

The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audience.

Testing Goals

- Assess, achieve and preserve quality and safety
- Find as many critical faults as possible
- Find faults as early as possible
- Avoid higher costs from late bug searching
- In practice, testing is the only way to demonstrate that the system works

Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:

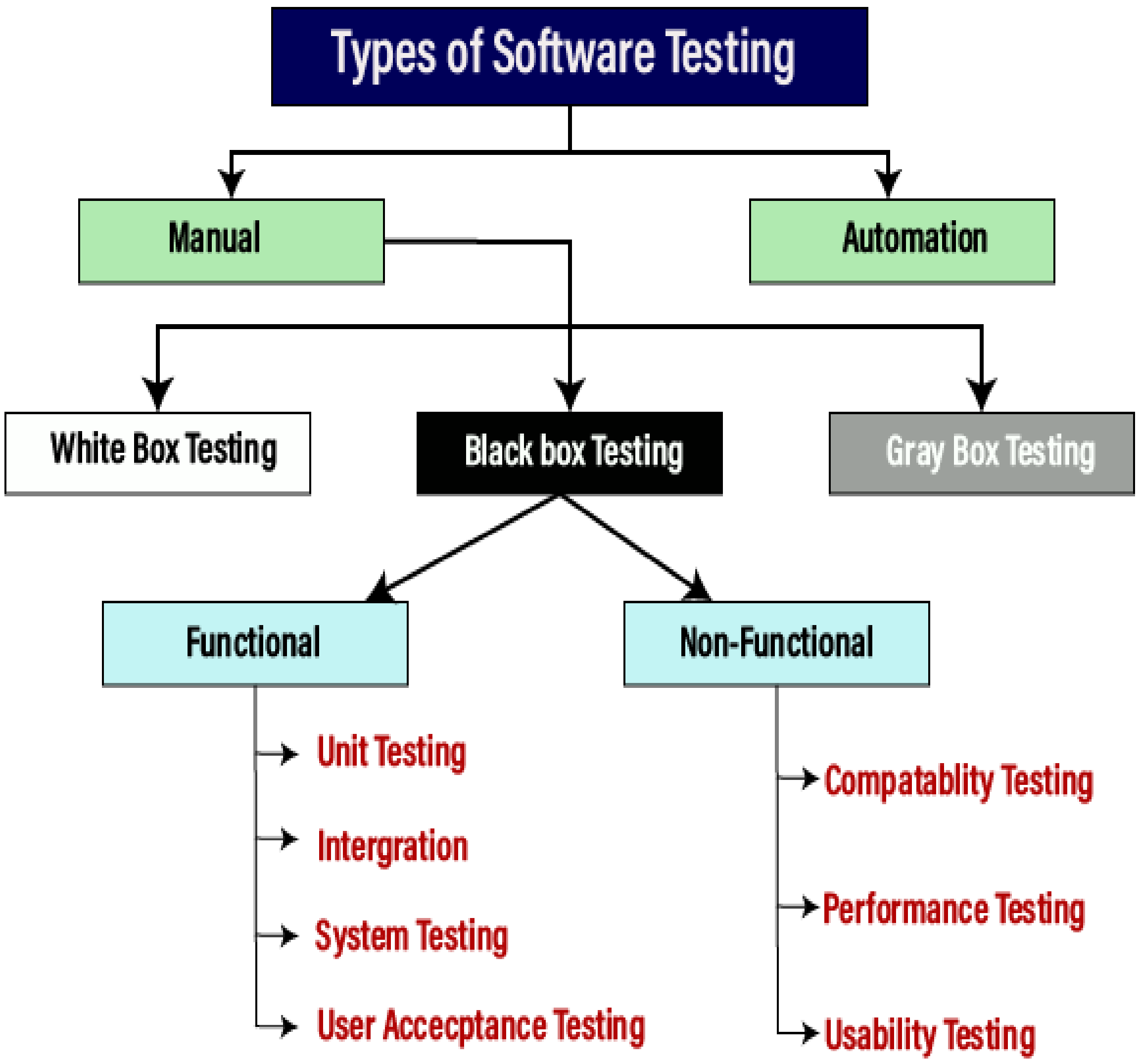


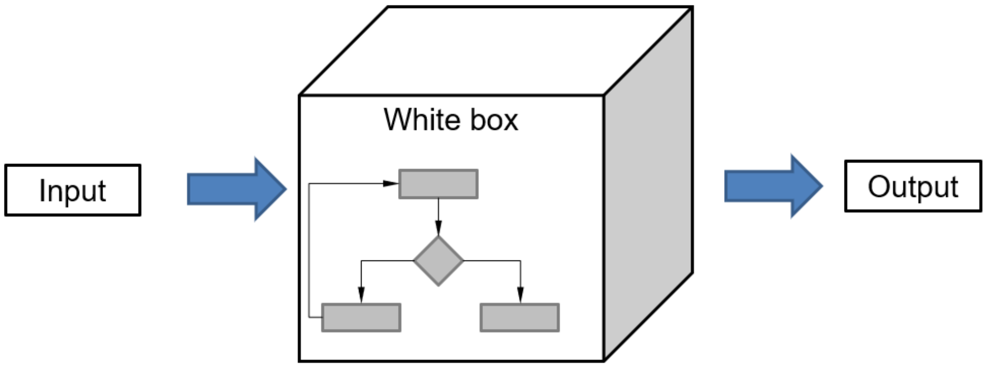
Chart 9: Types of software testing

Manual testing

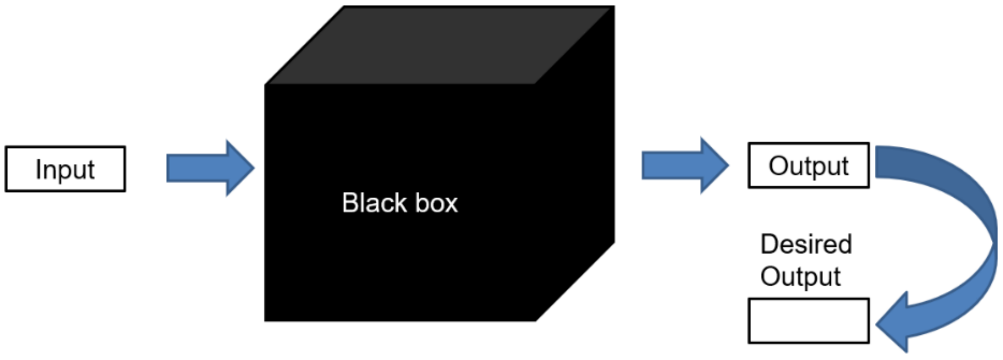
The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

- White box testing



- Black box testing



- Gray box testing

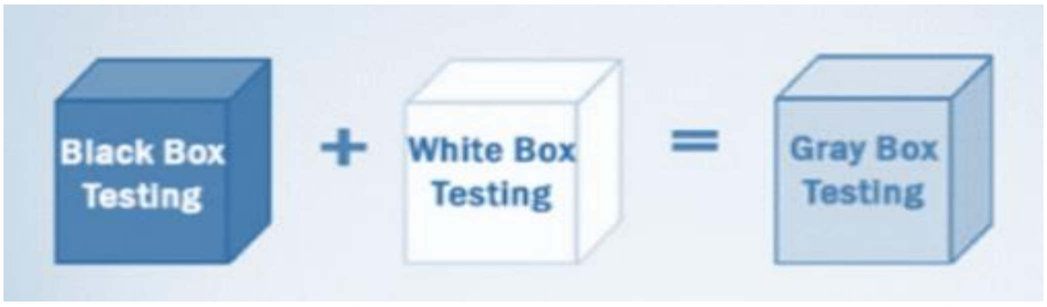


Figure 15: Different kinds of manual testing

Automation testing

Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

Bugs Classification

- Requirement, features, functional bugs
- Structural bugs
- Data bugs
- Coding bugs
- Interface, integration and system bugs
- Test and test design bugs

Static vs dynamic testing

- Static testing
  - Checking Requirements and Code
  - No execution of code
- Benefits
  - Early error detection
  - Reduction of later testing
  - Entire team involved
- Disadvantage
  - Problematic with complex interactions of system components
  - Some errors can only be detected during execution

Dynamic testing

  - Error detection via execution of programme
  - Creation of specific test cases
- Benefits
  - Execution of test object
  - Interaction of system components is tested
  - Additional tests, such as performance, load …
- Disadvantages
  - Needs executable test object and environment
  - Only programmes that will get executed are tested
  - Only errors effects are detected, error causes need to be found in a separate step (debugging)

Why software testing

The High Cost of Software Defects

While it’s always valuable to find software defects, finding them early in the development lifecycle is how organizations derive the most value from their static analysis investment. The following chart shows cost of finding defects relative to the Software Development Life-cycle (SDLC).

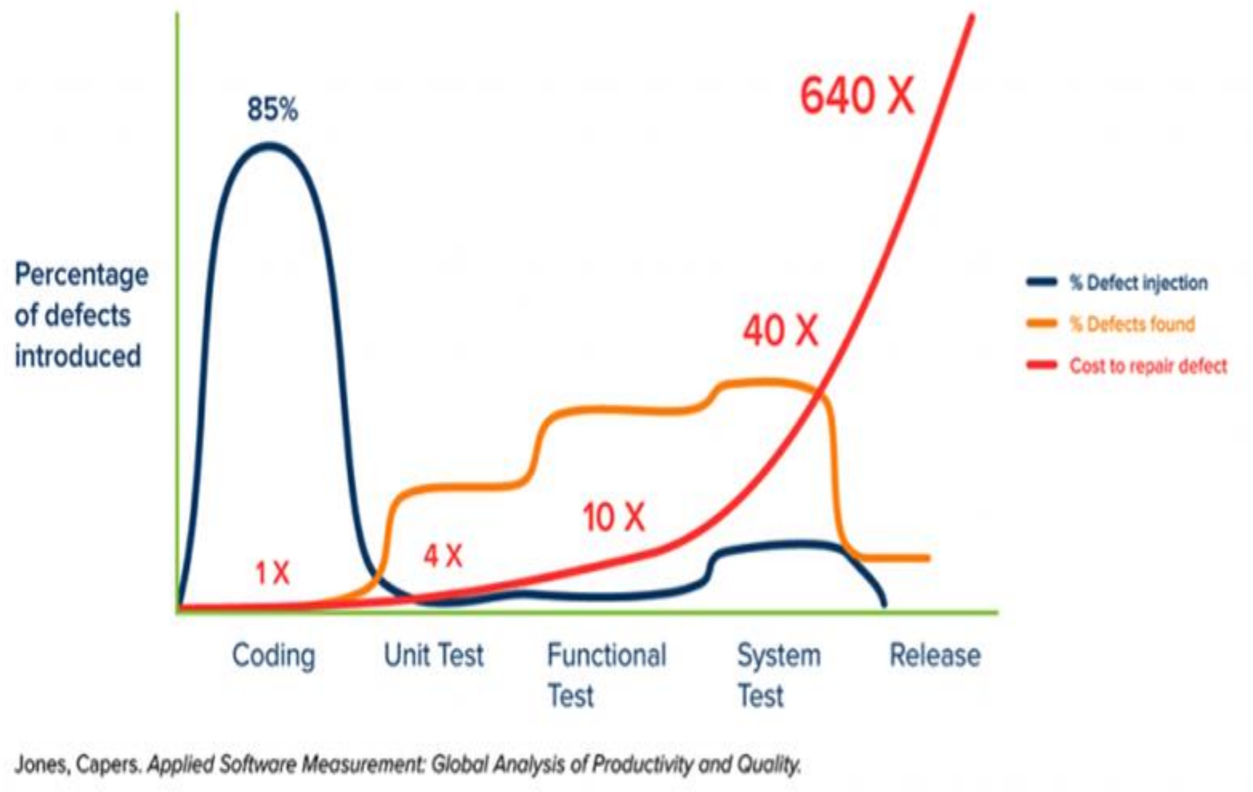


Figure 16: Illustration of high cost of software defects

In their case study, highlights the value of finding errors in the coding stage. Leveraging static analysis enabled to find programming errors before the software hits production, saving costs associated with retesting, recertifying, and redeployment. Most importantly, though, early defect detection keeps in good standing with their customers. Their proactive approach to catching as many bugs as possible early on enables the company to quickly deliver the high-quality software their customers have come to expect, while avoiding costs associated with late-stage defect detection.

4.3 Software test plan

What is a software test plan?

A test plan is a document that sets out the scope, approach, and schedule of intended testing activities. The test plan may also list the resources the software tester needs to function effectively.

The test plan usually includes the following information:

i. The overall objective of the testing effort.
ii. A detailed outline of how testing will be conducted (the test approach).
iii. The features, applications, or components to be tested.
iv. Detailed scheduling and resource allocation plans for testers and developers throughout all stages of testing.

What are the objectives of a software test plan?

The primary objective for a test plan is to produce documentation that describes how the tester will verify that the system works as intended. The document should describe what needs to be tested, how it will be tested, and who's responsible for doing so.

By writing up a test plan, all team members can work in unison and communicate their roles to one another. You should consider creating some SMART objectives for your test plan.

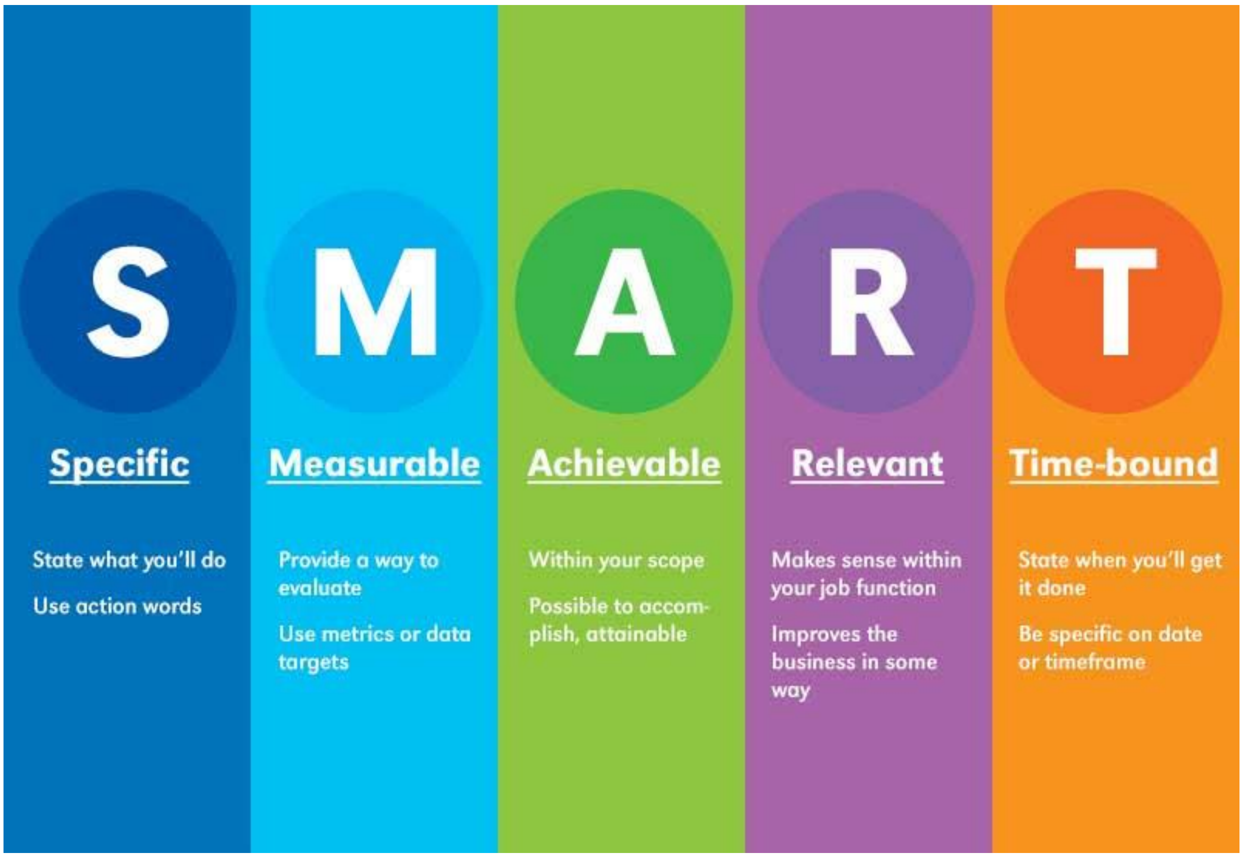

S
M
A
R
T
Specific
State what you'll do
Use action words
Measurable
Provide a way to evaluate
Use metrics or data targets
Achievable
Within your scope
Possible to accom-plish, attainable
Relevant
Makes sense within your job function
Improves the business in some way
Time-bound
State when you'll get it done
Be specific on date or timeframe

Figure 17: Objectives of software test plan

What is a test case?

A test case is documentation created by the software tester that contains detailed information on what the test should accomplish. It's an essential part of recording information about testing activities and results.

Test cases are used in conjunction with test plans. A test case should include the following information.

i. A unique name or number to identify it.
ii. The features, applications, or components covered by the test case.
iii. Specific data values required for input fields and button controls to be tested.
iv. The predicted results of actions taken during testing (the expected outcome).
v. A description of the actual results following each action taken during testing (the actual outcome).
vi. An indication of whether or not the test case was successful.
vii. Any errors discovered.

What's the importance of a test plan?

A test plan is the foundation of every testing effort. It helps set out how the software will be checked, what specifically will be tested, and who will be performing the test. By creating a clear test plan all team members can follow, everyone can work together effectively.

Whether you're building an app or developing open-source software, a test plan is essential to delivering the final result.

A high-quality plan helps to identify risk areas, determine the order of testing activities, and allocate resources efficiently. The test plan becomes a useful reference document that can be referred back to throughout the product's development cycle.

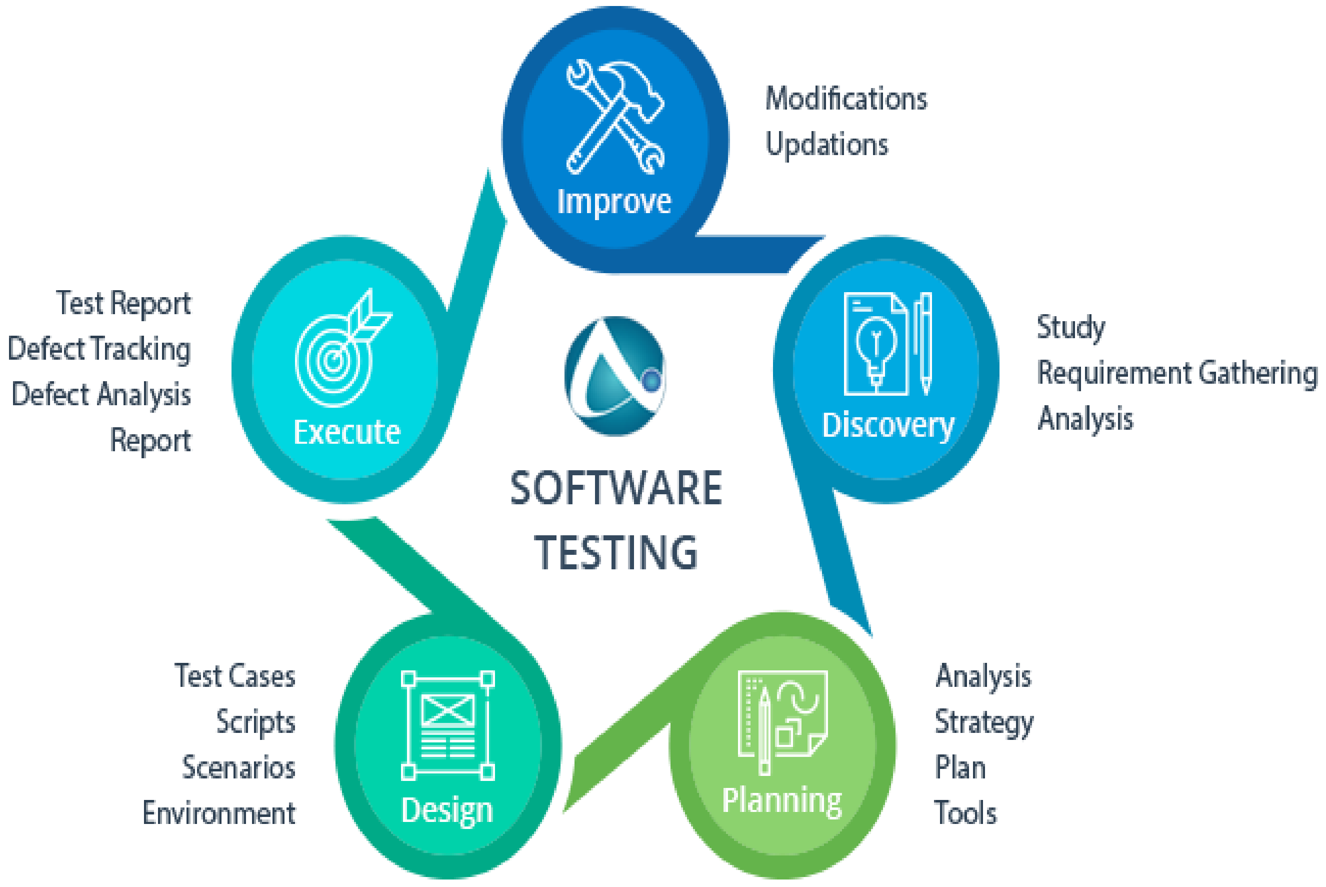


Figure 18: Software testing plan

How to keep the audience in mind when creating a test plan

Before you begin creating your test plan, you'll need to identify your intended consumers and make sure their needs are being met. This will improve the quality of your test plan tenfold.

Here are the main things to ensure your test plan is:

- **Concise.** Your test plan should be no longer than one page with bullet points.
- **Organized.** Make sure all the information is logically grouped.
- **Readable.** The document should be easy to read, avoiding technical language where possible.
- **Flexible.** Your test plan should be adaptable and not fixed in stone. You want to create documentation that won't hold you back if new information comes along or changes need to be made.
- **Accurate.** Make sure all the information you've included is accurate.

How to write a test plan

This might be the first job on your software developer CV, and if that's the case, you may need a cheat sheet to successfully write your initial test plan.
Luckily, we have you covered. This section will provide you with 14 essential things to include in your software test plan as part of the QA process.

1) Learn about the software

Before testing starts, it's important to learn everything you can about the software. Ask questions about how it was developed to learn about its intended purpose, how it works, and to garner information that might help you understand its functionality.
By understanding your software properly, you can create test cases that are relevant and useful for testing your product.

2) Define the scope of testing

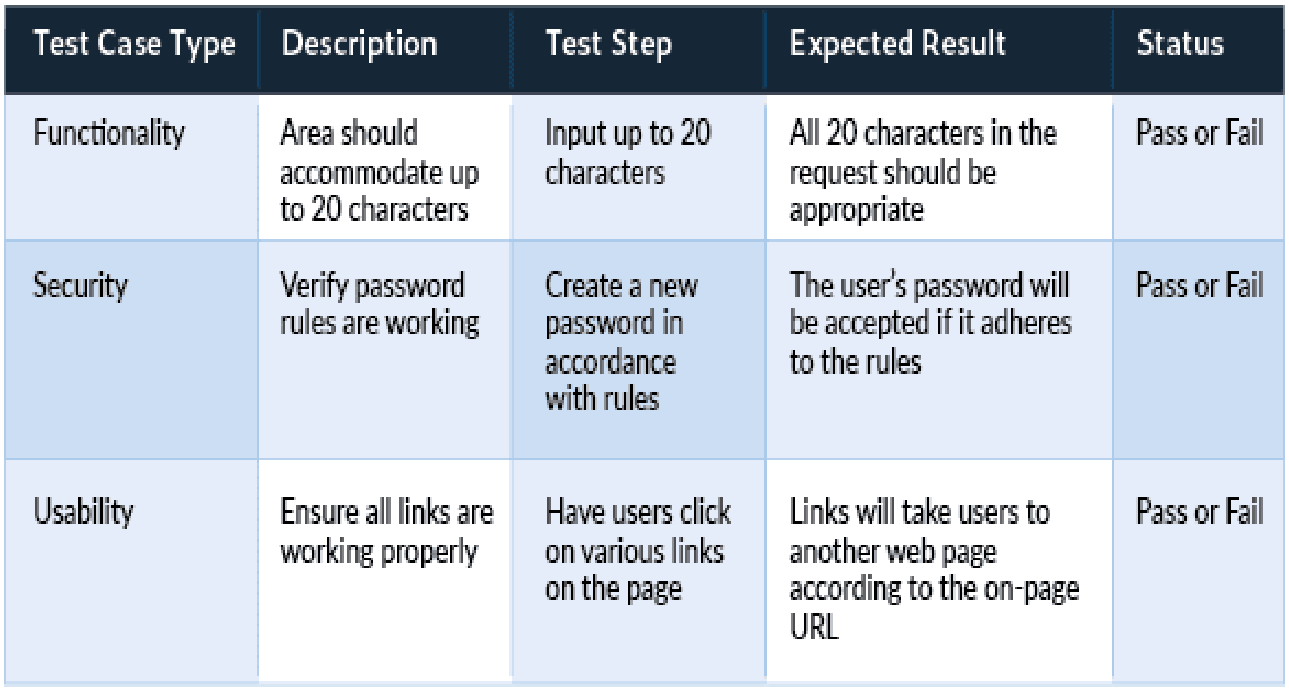
There's no point in creating testing documents that are longer than the product itself. Before anything else, establish what exactly will be tested during the process, which modules or functions need to be covered in detail, and any other essential aspects you should know about.

3) Create test cases

One of the main tasks when developing a software testing document is creating test cases. A test case is a document that describes the steps taken to carry out your testing. It should include information such as:

- What needs to be tested
- How it will be tested
- Who will do the testing
- Expected results

Here’s a simple spreadsheet for setting up test cases:



Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Area should accommodate up to 20 characters	Input up to 20 characters	All 20 characters in the request should be appropriate	Pass or Fail
Security	Verify password rules are working	Create a new password in accordance with rules	The user’s password will be accepted if it adheres to the rules	Pass or Fail
Usability	Ensure all links are working properly	Have users click on various links on the page	Links will take users to another web page according to the on-page URL	Pass or Fail

Figure 19: Test case for software

4) Develop a test strategy

The test strategy defines how you plan to implement testing. Your testers should all be working off the same game plan, so make sure every member of the team is aware of what they're supposed to be doing at any given time.

5) Define the test objective

Each test case should be linked to a test objective. The objective ensures every action is relevant and contributes toward making your software more valuable for customers. Test objectives can include things like:

- Testing known features
- Testing newly implemented features
- Performing exploratory tests
- Ensuring stability throughout the product lifecycle

6) Choose testing tools

You'll need to make sure you have the right software testing solution to perform your testing activities. Some of these tools might be software-based, while others may require physical resources like test machines. It's important to choose appropriate tools for each specific job and not to rely on a one-size-fits-all solution.

7) Find bugs early

Leave time in your planning document for 'bug fixing' sessions. These allow you to identify problems with the software early on before they become too problematic or expensive to fix. This makes them easier and cheaper to tackle. Check out any app security measures, use every feature, and seek out what doesn't work well.

8) Define your test criteria

This should be part of the test case, but it's good to break it down separately. Test criteria are essentially your objectives broken down into smaller parts. They include specific information about how each objective will be met, which helps you track your testing progress.

Suspension criteria are criteria that need to be met before testing can stop. For example, you may want to suspend testing if a certain number of bugs have been found or if the software is unable to run due to performance issues.

Exit criteria are criteria that need to be met before testing can finish. For example, the test case should finish once each objective has been met and all bugs have been resolved.

9) Resource planning

In your software testing document, include a resource plan that lists the number of people required for the testing process. This should detail what each person's role is and any training they'll require to fulfill it effectively.

10) Plan your test environment

In your test plan, include information about the environment where testing will take place, such as:

- Test hardware required for product testing.
- Sizing requirements for software and servers.
- Platforms supported by the product.
- Other essential information related to the environment that might affect your testing process.

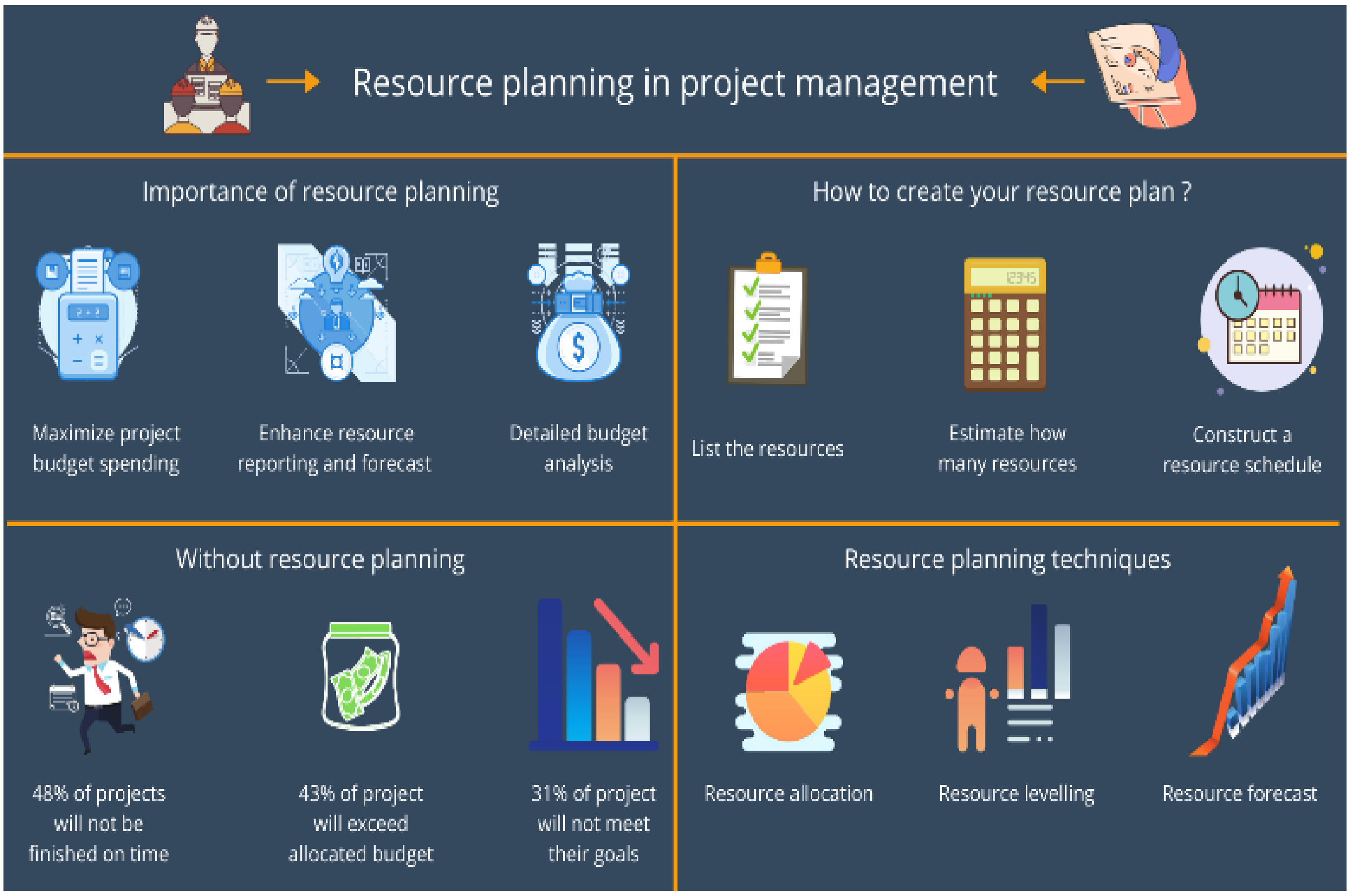

Resource planning in project management
Importance of resource planning
Maximize project budget spending
Enhance resource reporting and forecast
Detailed budget analysis
How to create your resource plan ?
List the resources
Estimate how many resources
Construct a resource schedule
Without resource planning
48% of projects will not be finished on time
43% of project will exceed allocated budget
31% of project will not meet their goals
Resource planning techniques
Resource allocation
Resource levelling
Resource forecast

Figure 20: Resource planning in project management

11) Plan test team logistics

Test management is one of the most important parts of implementing process. If you're not able to communicate with your testers effectively, their progress will suffer and your testing document won't be as useful as it could be.

12) Schedule & estimation

In your test plan, include a schedule that allows you to outline specific testing milestones and deadlines. Milestones may include the initial release of the product, internal testing sessions, public beta tests, or any other key points in time where your team needs to focus their efforts on testing.

13) Test deliverables

Your testing document should include a list of all the deliverables required for testing. These should be linked to the steps in your schedule so everyone knows exactly when they need to be ready for action.

14) Test automation

If your software is particularly complex and requires a vast number of test cases, you may want to consider software test automation.

Automating the process means testers can accomplish more in less time, which boosts productivity and significantly reduces the overall cost of testing. You might even be able to utilize a mobile bot to speed up testing activities.

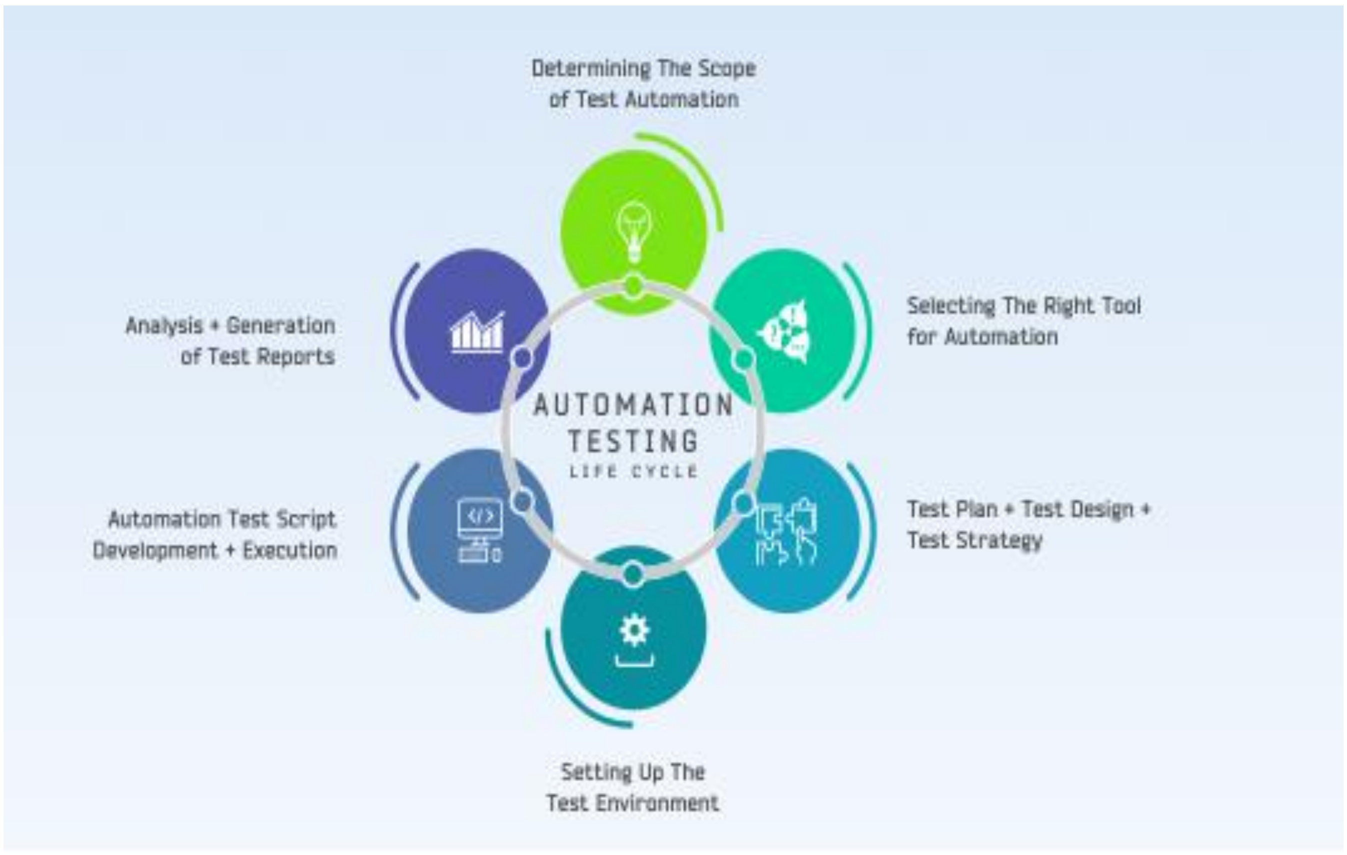

Determining The Scope of Test Automation
Selecting The Right Tool for Automation
Test Plan + Test Design + Test Strategy
Setting Up The Test Environment
Automation Test Script Development + Execution
Analysis + Generation of Test Reports
AUTOMATION TESTING LIFE CYCLE

Figure 21: Life Cycle of automation testing

Exercise: Compare Black Box Testing vs. White Box Testing vs. Grey Box Testing

Index	Black Box Testing	White Box Testing	Grey Box Testing
1	Knowledge of internal working structure (Code) is not required for this type of testing. Only GUI (Graphical User Interface) is required for test cases.	Knowledge of internal working structure (Coding of software) is necessarily required for this type of testing.	Partially Knowledge of the internal working structure is required.
2	Black Box Testing is also known as functional testing, data-driven testing, and closed box testing.	White Box Testing is also known as structural testing, clear box testing, code-based testing, and transparent testing.	Grey Box Testing is also known as translucent testing as the tester has limited knowledge of coding.
3	The approach towards testing includes trial techniques and error guessing method because tester does not need knowledge of internal coding of the software.	White Box Testing is proceeded by verifying the system boundaries and data domains inherent in the software as there is no lack of internal coding knowledge.	If the tester has knowledge of coding, then it is proceeded by validating data domains and internal system boundaries of the software.
4	The testing space of tables for inputs (inputs to be used for creating test cases) is pretty huge and largest among all testing spaces.	The testing space of tables for inputs (inputs to be used for creating test cases) is less as compared to Black Box testing.	The testing space of tables for inputs (inputs to be used for creating test cases) is smaller than Black Box and White Box testing.

Index	Black Box Testing	White Box Testing	Grey Box Testing
5	It is very difficult to discover hidden errors of the software because errors can be due to internal working which is unknown for Black Box testing.	It is simple to discover hidden errors because it can be due to internal working which is deeply explored in White Box testing.	Difficult to discover the hidden error. Might be found in user level testing.
6	It is not considered for algorithm testing.	It is well suitable and recommended for algorithm testing.	It is not considered for algorithm testing.
7	Time consumption in Black Box testing depends upon the availability of the functional specifications.	White Box testing takes a long time to design test cases due to lengthy code.	Test cases designing can be done in a short time period.
8	Tester, developer and the end user can be the part of testing.	Only tester and developer can be a part of testing; the end user can not involve.	Tester, developer and the end user can be the part of testing.

Table 3: Comparison of Black Box, White Box and Grey Box Testing

UNIT 2: MICROCONTROLLER PROGRAMMING

Objective: The trainees:

- Know the difference between using a microcontroller and a PLC
- Are able to control actuators and sensors and connect them to the microcontroller
- Can save measured values in log files
- Are able to search and find errors in a structured way
- Are able to commission, configure and parameterize a microprocessor
- Create state machines to implement measuring programmes
- Are able to integrate software modules into a sequence programme
- Are familiar with the possibilities for data exchange between microprocessors
- Know possibilities for connecting microprocessors to higher-level IT systems

Content:

1. Microcontroller programming basics

1.1. Microcontroller Basics

a. What is a Microcontroller?

A Microcontroller is a VLSI (Very Large Scale Integration) Integrated Circuit (IC) that contains electronic computing unit and logic unit (combinedly known as CPU), Memory (Program Memory and Data Memory), I/O Ports (Input / Output Ports) and few other components integrated on a single chip.

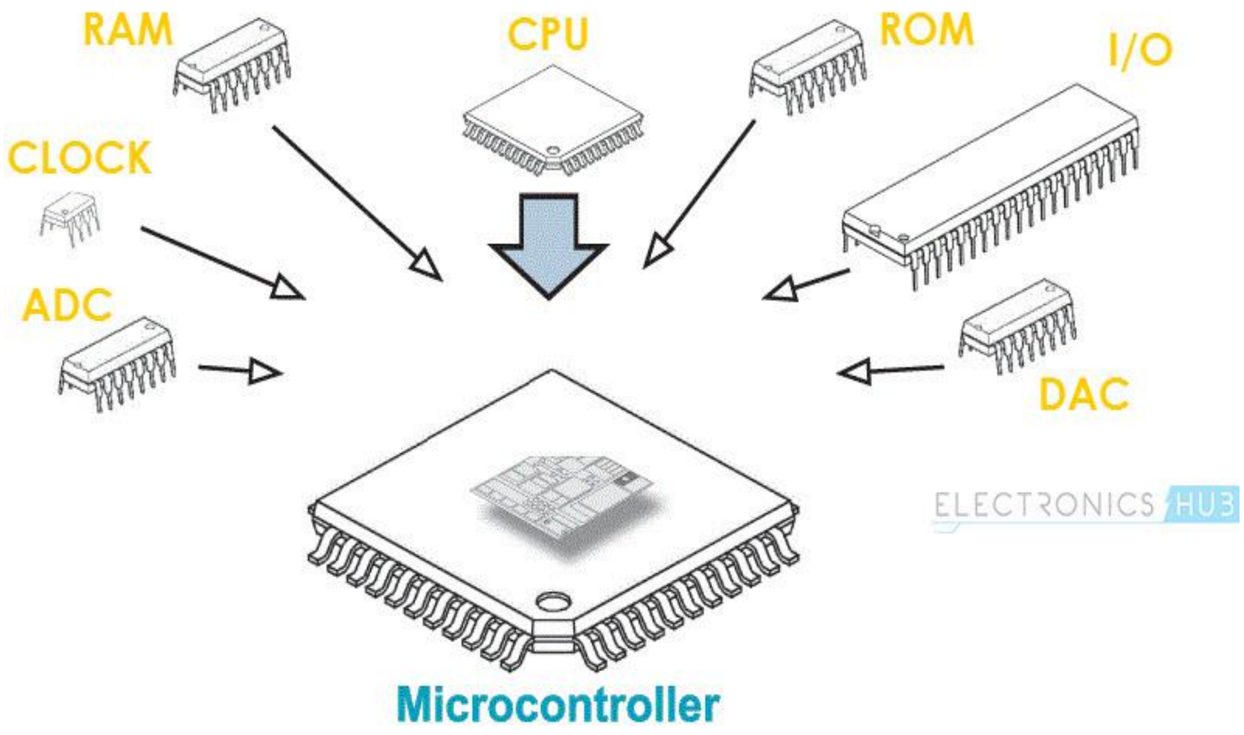


Figure 22: Illustration of Microcontroller

Sometimes, a Microcontroller is also called as a Computer-on-a-Chip or a Single-Chip-Computer. Since the Microcontroller and its supporting circuitry are often embedded in the device it controls, a Microcontroller is also called as an Embedded Controller.

Microcontrollers are omnipresent. If a device or an application involves measuring, storing, calculating, controlling or displaying information, then device contains a Microcontroller in it. Let us see some of the areas where microcontrollers are used.

The biggest user of Microcontrollers is probably the Automobiles Industry. Almost every car that comes out of the assembly factory contains at least one Microcontroller for the purpose of engine control. You can find many more Microcontrollers for controlling additional systems.

Consumer Electronics is another area which is loaded with Microcontrollers. Microcontrollers are a part of Digital Cameras, Video Camcorders, CD and DVD Players, Washing Machines, Ovens, etc.

Microcontrollers are also used in test and measurement equipment like Multimeters, Oscilloscopes, Function Generators, etc. You can also find microcontrollers near your desktop computer like Printers, Routers, Modems, Keyboards, etc.

The above definitions of the Microcontroller might seem complicated or confusing to newbies in Electronics or Embedded Systems but the concept will become clear as we move forward.
First, we will see the Rise of Microcontrollers, where you can find how the development to the Microcontroller took place.

b. Rise of Microcontrollers

Microprocessor, the invention that took the field of computation by storm. A Microprocessor is an Integrated Circuit (IC) that contains the Central Processing Unit (CPU). The earliest known Microprocessors are the Intel’s 4004 and the Texas Instruments’ TMS1000.
Since then, the computational power, complexity and power consumption kept on increasing in order to provide ultimate performance (Power Consumption must be discussed separately due to developments such as Low Power VLSI etc.).

For a Microprocessor to work, it needs a bunch of supporting hardware that can be found on a mother board. The hardware includes memory, ICs for peripheral devices, etc.
In the beginning itself, the Microprocessors ability to control other **electronic equipment** like Photocopiers is realized. The emphasis here is not on the computational power of the Microprocessor but rather on a control mechanism with less complex hardware and increased reliability.
This requirement paved way for integrating the minimum hardware required for complete functioning of a Processor on to a single chip i.e. same chip as the processor, to be precise.
This is the rise of Microcontrollers, an Integrated Circuit, which contains all the functions and hardware in order to make a complete computer system. Here, the computational power of the device is of less importance than the integration of all the components, including memory.

c. Basics of Microcontrollers

Basically, a Microcontroller consists of the following components.

- Central Processing Unit (CPU)
- Programme Memory (ROM – Read Only Memory)
- Data Memory (RAM – Random Access Memory)
- Timers and Counters
- I/O Ports (I/O – Input/Output)
- Serial Communication Interface
- Clock Circuit (Oscillator Circuit)
- Interrupt Mechanism

Most modern Microcontrollers might contain even more peripherals like **SPI (Serial Peripheral Interface)**, I2C (Inter Integrated Circuit), ADC (**Analog to Digital Converter**), DAC (Digital to Analog Converter), CAN (Controlled Area Network), USB (Universal Serial Bus), and many more.

The CPU (Central Processing Unit) in a Microcontroller performs the arithmetic, logic, math and data-oriented function, similar to CPU in the Microprocessor. The difference between a Microprocessor and Microcontroller is that a Microprocessor need to be interface with external memory and other I/O Interfaces to work as a computer whereas, a Microcontroller has all the required peripherals on the same chip as the CPU.

The integration of features like ADC, DAC etc. on the same chip as the CPU makes it more efficient and cheaper than to use a separate ADC Chip.

Developing a Computer Controlled System involves design of the Hardware and also writing an efficient Software Programme. Since a Microcontroller has all the hardware, that are required to make a computer controlled system on a single chip, using a Microcontroller will drastically reduce the efforts and time spent on hardware design and wiring.

d. Basic Structure of a Microcontroller

You might have seen the basic structure of a Microcontroller many times. If you have already seen the structure of Microcontroller and the **basic components** of a Microcontroller before, then consider this as a revision. If you haven't seen it, then it is very important to get an idea about the basic structure of a Microcontroller.

The following image shows the Basic Structure of a Microcontroller.

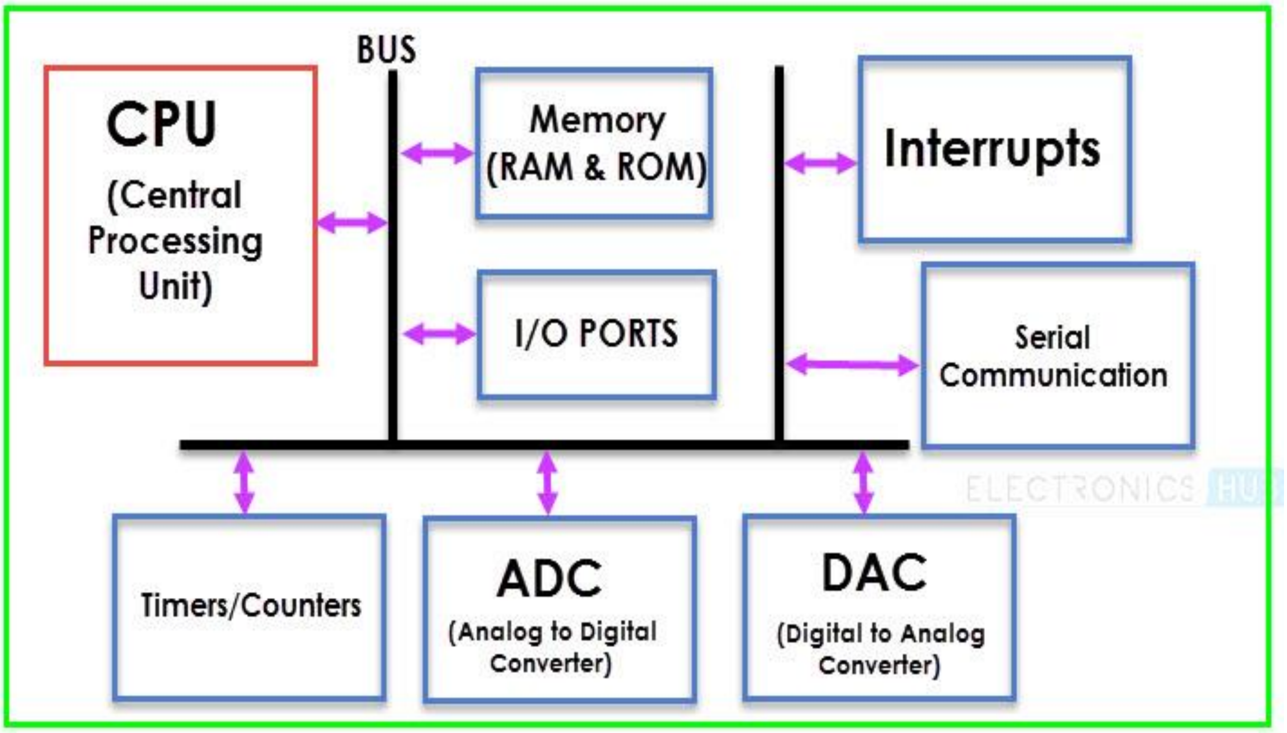

BUS
CPU (Central Processing Unit)
Memory (RAM & ROM)
Interrupts
I/O PORTS
Serial Communication
ELECTRONICS HUB
Timers/Counters
ADC (Analog to Digital Converter)
DAC (Digital to Analog Converter)

Figure 23: Basic Structure of a Microcontroller

From the above image, you can understand that the three important (or major) components of a Microcontroller are:

- The CPU (Central Processing Unit)
- The Memory and
- The I/O Ports

This doesn't mean that other components are of less importance. But these can be considered as supporting devices. We will now see each of the Basic Components of a Microcontroller mentioned in the above structure.

e. Advantages of Microcontrollers

- A Microcontroller is a true device that fits the computer-on-a-chip idea.
- No need for any external interfacing of basic components like Memory, I/O Ports, etc.
- Microcontrollers doesn't require complex operating systems as all the instructions must be written and stored in the memory. (RTOS is an exception).
- All the Input/Output Ports are programmable.
- Integration of all the essential components reduces the cost, design time and area of the product (or application).
-

f. Disadvantages of Microcontrollers

- Microcontrollers are not known for their computation power.
- The amount of memory limits the instructions that a microcontroller can execute.
- No Operating System and hence, all the instruction must be written.

g. Applications of Microcontrollers

There are huge number of applications of Microcontrollers. In fact, the entire embedded systems industry is dependent on Microcontrollers. The following are few applications of Microcontrollers.

- Front Panel Controls in devices like Oven, washing Machine etc.
- Function Generators
- Smoke and Fire Alarms
- Home Automation Systems
- Automatic Headlamp ON in Cars
- Speed Sensed Door Locking System

In this tutorial/article, we have seen the basics of Microcontrollers, Basic Structure of a Microcontroller, different components of a Microcontroller, advantages, disadvantages and applications of Microcontrollers.

1.2. Basic terms of programming

a. What is a programming term?

Programming terms are a collection of words representing scientific and technological concepts. Usually, they will be used in the fields of programming and represent the exact concept of expertise.

For the most part, programming terms are not expressive and must be approved and promulgated by the competent authority.

b. General term

- Software Engineering: Kỹ thuật phần mềm.
- HDSE (Higher Diploma in Software Engineering): Chứng Chỉ kỹ sư phần mềm Quốc tế.
- Structured Programming: Lập trình cấu trúc .
- OOP (Object-Oriented Programming): Lập trình hướng đối tượng.
- Programmer: Lập trình viên.
- Programming: Lập trình.
- Programme: Chương trình.
- Tester: Người kiểm thử chương trình.
- Designer: Chuyên viên thiết kế.
- Developer: Người phát triển phần mềm.
- Project: Dự án.
- Project Manager (PM): Quản lý dự án.
- Coder: Người viết Code.

c. Glossary of source code

- Source code: Mã nguồn.
- Open source: Mã Nguồn mở.
- Code: Mã.
- Design: Thiết kế.
- Source file: File nguồn.
- Library: Thư viện.
- Header file: File chứa các nguyên mẫu hàm.
- Implementation File: File chứa nội dung thực thi, mã lệnh của các hàm.

d. Terminology of translation tools and programmes

- Run: Chạy chương trình.
- Debug: Gỡ rối, sửa lỗi.
- Error: Lỗi.
- Compile error: Lỗi khi dịch chương trình.
- Runtime error: Lỗi khi chạy chương trình.

- Integrated-Development-Environment (IDE): Môi trường tích hợp phát triển.
- Compiler: Trình biên dịch.
- Compile: Dịch chương trình.
- Interpreter: Trình thông dịch.
- Line: dòng.
- Editor: Trình soạn thảo.

e. Terminology when writing code

- Operator: Toán tử .
- Function: Hàm .
- Character: Ký tự .
- Digits: Chữ số .
- Argument: Đối số.
- Selection: Chọn lựa, rẽ nhánh.
- Statement: Câu lệnh .
- Declaration: Khai báo.
- Initialization: Khởi tạo.
- Definition: Định nghĩa.
- Condition: Điều kiện.
- Control structure: Cấu trúc điều khiển.
- Value: Giá trị.
- Syntax: Cú pháp.
- Function Call: Lời gọi hàm.
- Expression: Biểu thức.
- Operand: Toán hạng .
- Dynamic Variable: Biến động .
- Memory leak: Lỗi xảy ra khi con trỏ ra khỏi phạm vi khi chưa giải phóng bộ nhớ.
- Pointer: Con trỏ .
- Reference: Tham chiếu.
- Parameter: Tham số .
- Prototype: Nguyên mẫu hàm .
- Comment: Ghi chú, chú thích .
- Code block: Khối lệnh .
- Assign: Gán .
- Allocate (memory): Cấp phát bộ nhớ.
- Deallocate (memory): Giải phóng và thu hồi bộ nhớ.
- Dynamic Memory: Bộ nhớ động .
- Static Memory: Bộ nhớ tĩnh.
- Static variable: Biến tĩnh .

1.3. Difference between microcontroller and PLC

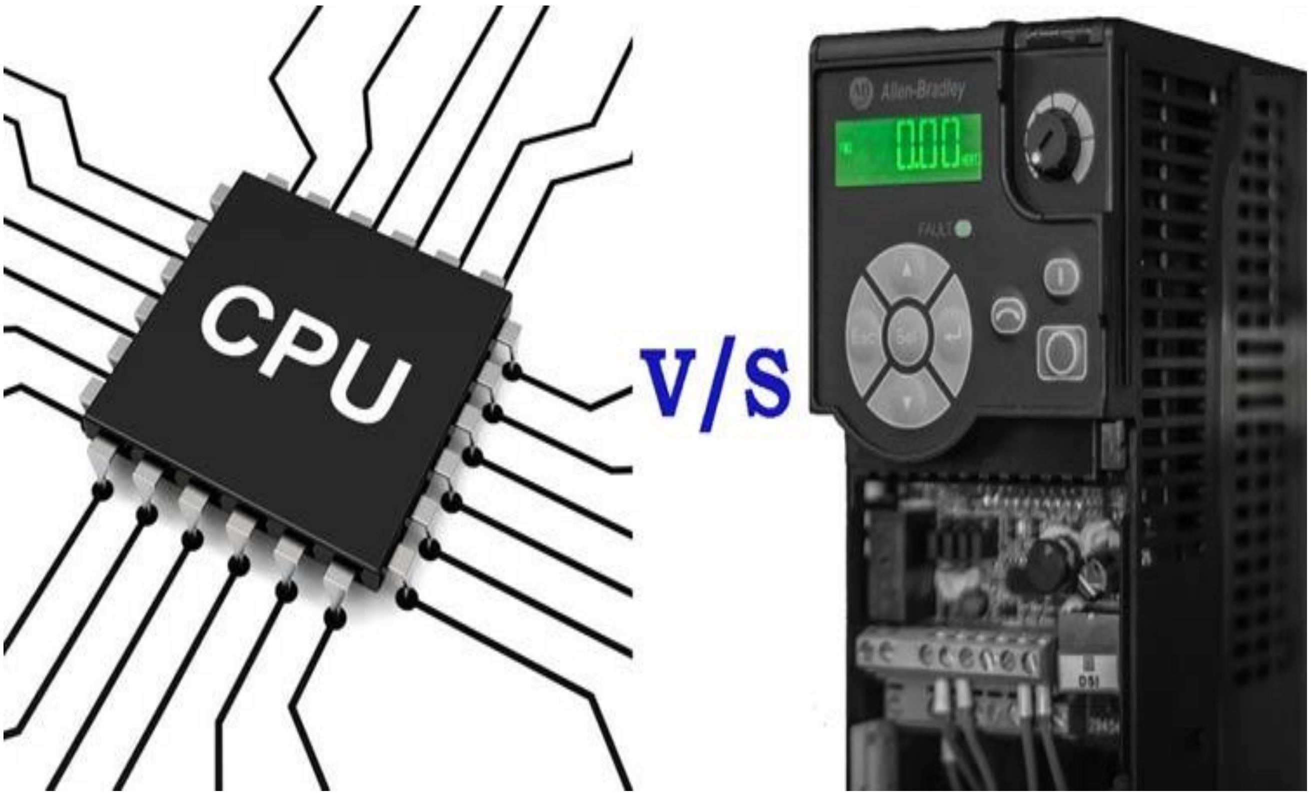


Figure 24: Microcontroller and PLC

a. Architecture

PLCs Architecture:
PLCs generally can be referred to as a high-level microcontroller. They are essentially made up of a processor module, the power supply, and the I/O modules. The processor module consists of the central processing unit (CPU) and memory. In addition to a microprocessor, the CPU also contains at least an interface through which it can be programmed (USB, Ethernet or RS232) along with communication networks. The power supply is usually a separate module, and the I/O modules are separate from the processor. The types of I/O modules include discrete (on/off), Analog (continuous variable), and special modules like motion control or high-speed counters. The field devices are connected to the I/O modules.

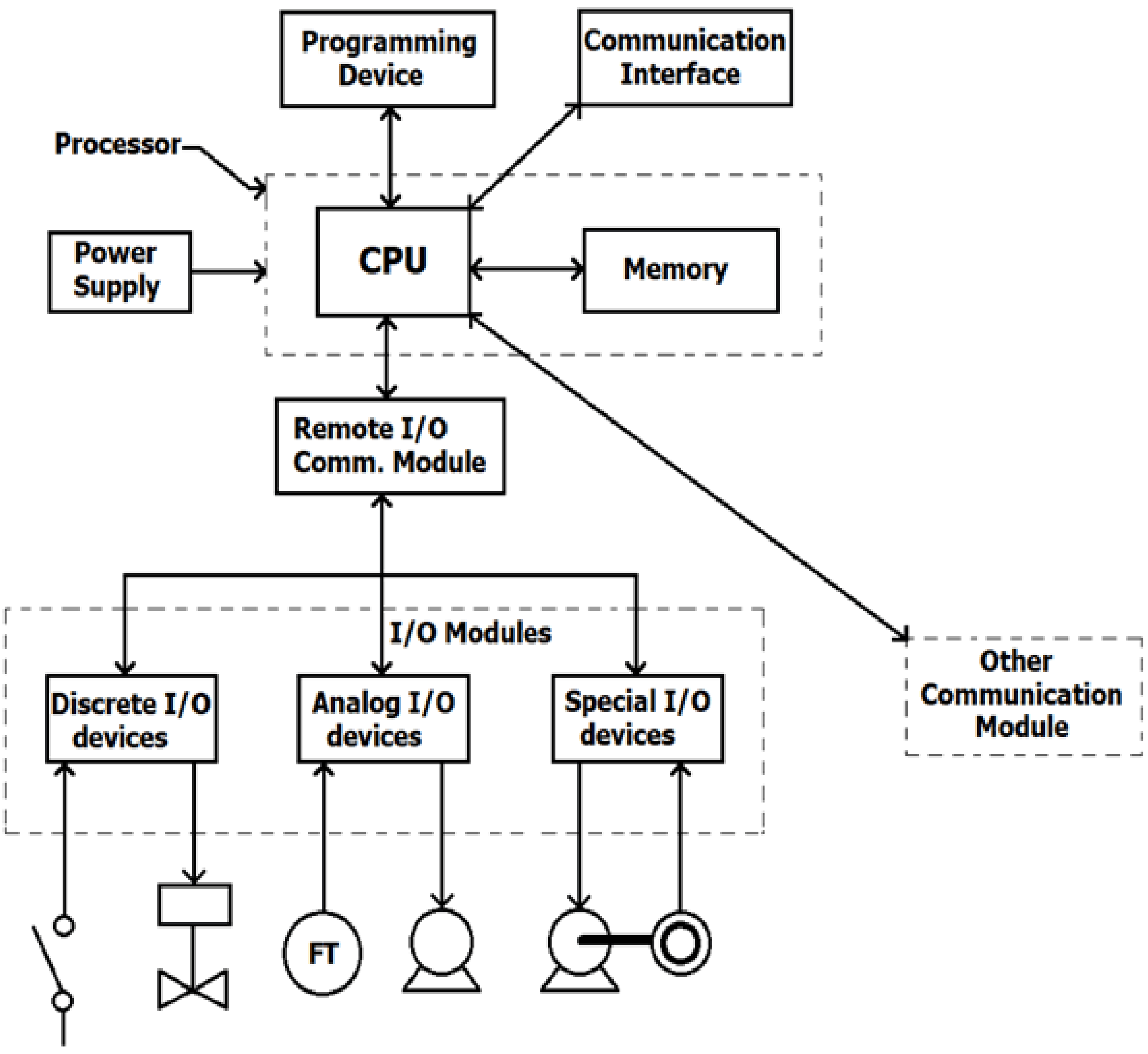


Figure 25: PLCs Architecture

Depending on the amount of I/Os modules possessed by the PLC, they may be in the same enclosure as the PLC or in a separate enclosure. Certain small PLCs called nano/micro PLCs usually have all their parts including power, processor etc. in the same enclosure.

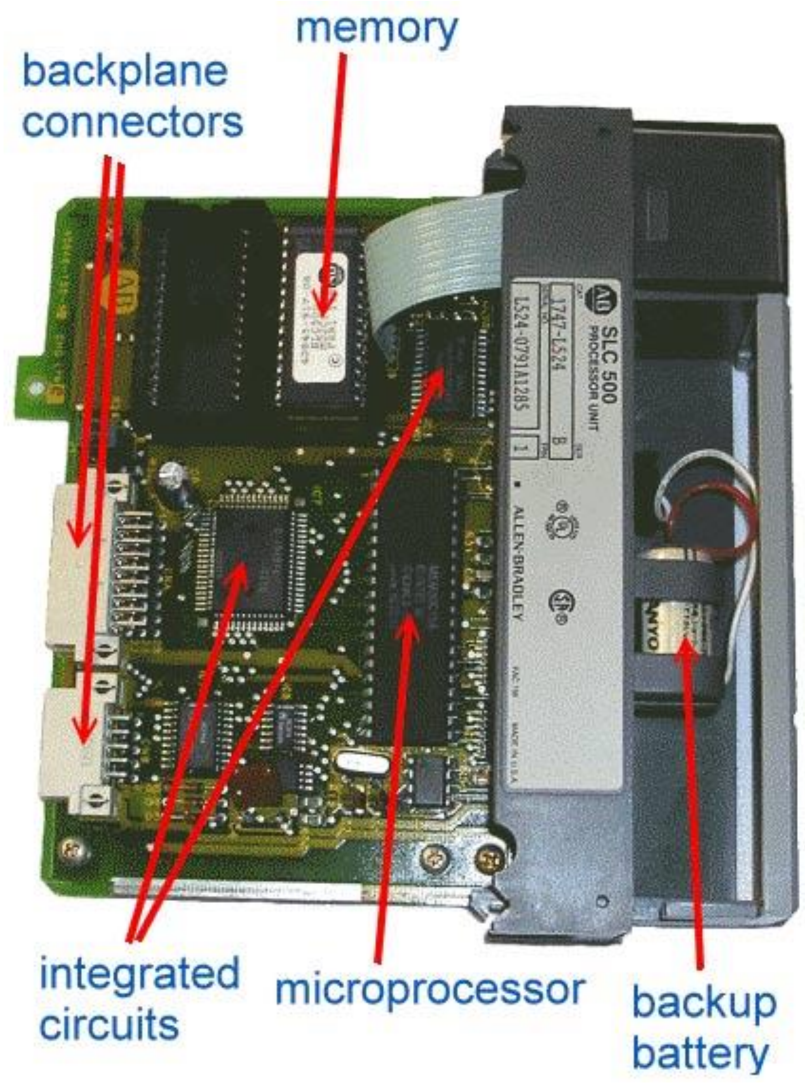


Figure 26: PLC hardware structure

Microcontroller's Architecture

The architecture of PLCs described above is somewhat similar to the microcontrollers in terms of constituents, but the microcontroller implements everything on a single chip, from the CPU to the I/O ports and interfaces required for communication with the outside world. Architecture of the microcontroller is shown below.

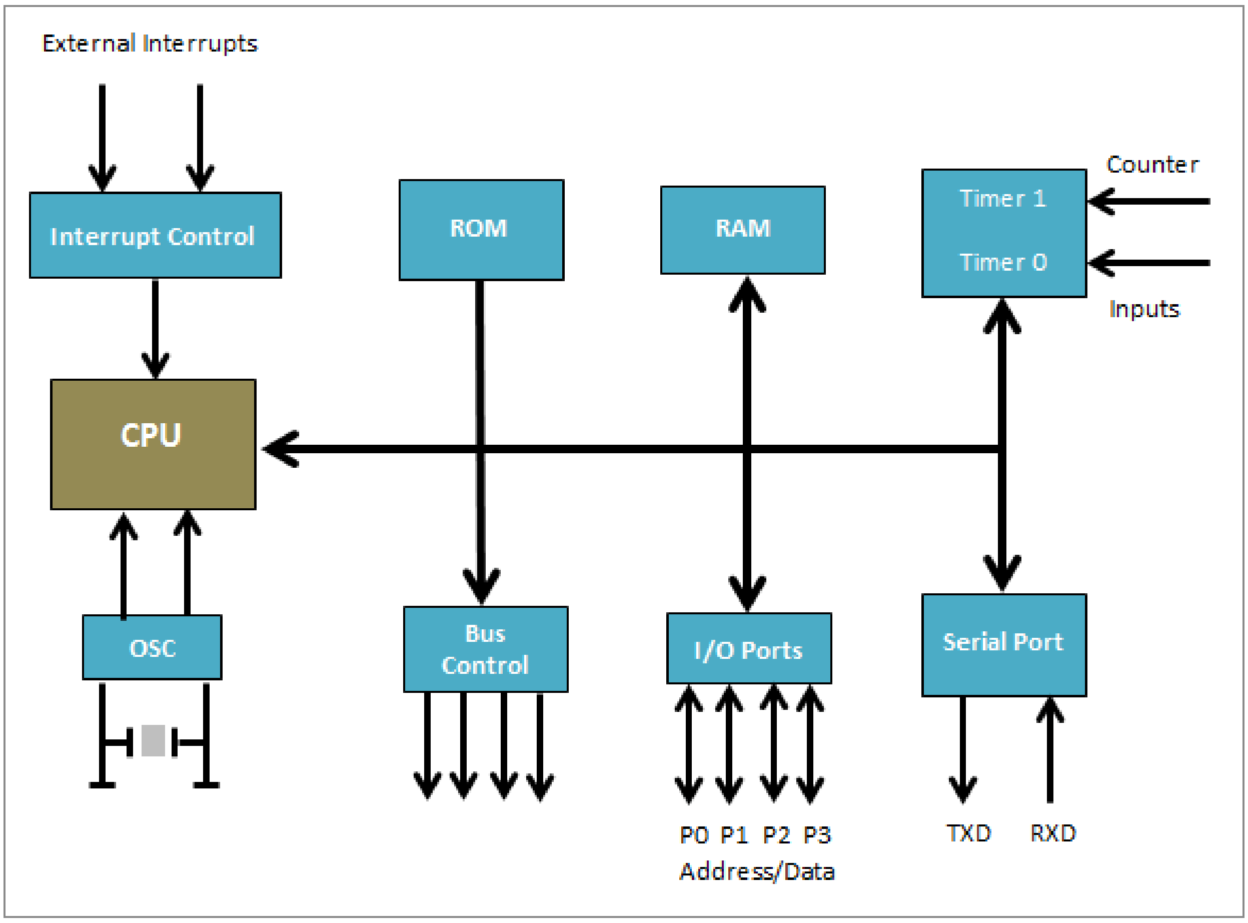


Figure 27: Microcontroller's Architecture

Just like the microcontroller has diverse architecture from the AVR architecture to the 8051 architecture, PLCs likewise have variations in their design which supports the configuration and desire of a particular manufacturer but they generally all adhere to the industry standard (IEC 61131-3) for PLCs. This standard fosters interoperability between modules and parts.

b. Interfaces

PLCs are standard designed to interface with industrial grade sensors, actuators, and communication modules and are thus given current and voltage ratings which are often incompatible with microcontrollers without extra hardware.

PLCs usually use Ethernet, and several variations of the RS- serial series like RS-232, RS-485 for communication. The advent of the industrial **internet of things** nowadays, is creating a surge in the number of connected PLC devices capable of transmitting data over wireless communication interfaces.

As mentioned earlier, they come in different sizes, from small devices (with few IO pins/modules) which are referred to as building blocks to large, giant rack mounted PLCs with hundreds of IOs.

Microcontrollers as well have sensors, actuators, and modules designed to meet their specific needs which might be difficult to interface with a PLC. They are however usually designed to handle processing of only a few 100 IOs. While several techniques can be explored to increase the IOs of the microcontroller, this are still possible with PLCs and is thus not unique to the microcontrollers, asides from the fact that it increases the entire project budget.

c. Performance, Sturdiness and Reliability

This is by far the point under which the PLC distinguishes itself the most. As mentioned initially, the PLC was designed for use in industrial setups and was thus fortified to be able to withstand several adverse conditions associated with that environment like, extreme temperature ranges, electrical noise, rough handling and high amount of vibration. PLCs are also a good example of real time operation system due to their ability to produce outputs within the shortest time possible after evaluating an input. This is very important in industrial system as timing is a huge part of the manufacturing plant/process.

Microcontrollers however are less sturdy. By design they were not designed to serve as standalone devices like PLCs. They were designed to be embedded in a system. This provides an explanation for their less sturdy look compared to PLCs. For these reasons, microcontrollers may fail when deployed in certain scenarios as the chips are fragile and can easily be damaged.

d. Skill Requirement for Use

One of the key attributes of the PLC is the low technical knowledge required for programming, and generally operating it. The PLC was designed to be use by both the highly skilled automation experts and factory technicians who have little or no formal training. It is relatively easy to troubleshoot and diagnose faults. Modern PLC devices usually come with a display screen that makes things easier to monitor without sophisticated tools.

Microcontrollers on the other hand however, require skilful handling. Designers need to have a good knowledge of electrical engineering principles and programming to be able to design complementary circuits for the microcontroller. Microcontrollers also require special tools (e.g Oscilloscope) for fault diagnosis and firmware trouble shooting. Although several simplified platforms like the Arduino currently exists, it is still a lot more complex than the plug and play PLCs both from connection stand point, programming standpoint, and ease of use.

e. Programming

For the sake of simplicity and ease of use by all knowledge classes, PLCs were originally designed to be programmed using a visual of programming that mimics the connections/schematics of relay logic diagrams. This reduced the training requirements for existing technicians. The primary, most popular programming language used for PLCs are the Ladder Logic and instruction list programming language. Ladder logic uses symbols, instead of words, to emulate the real world relay logic control, which is a relic from the PLC's history. These symbols are interconnected by lines to indicate the flow of current through relay, like contacts and coils. The number of symbols has increased tremendously over the years enabling engineers to easily implement high level of functionalities.

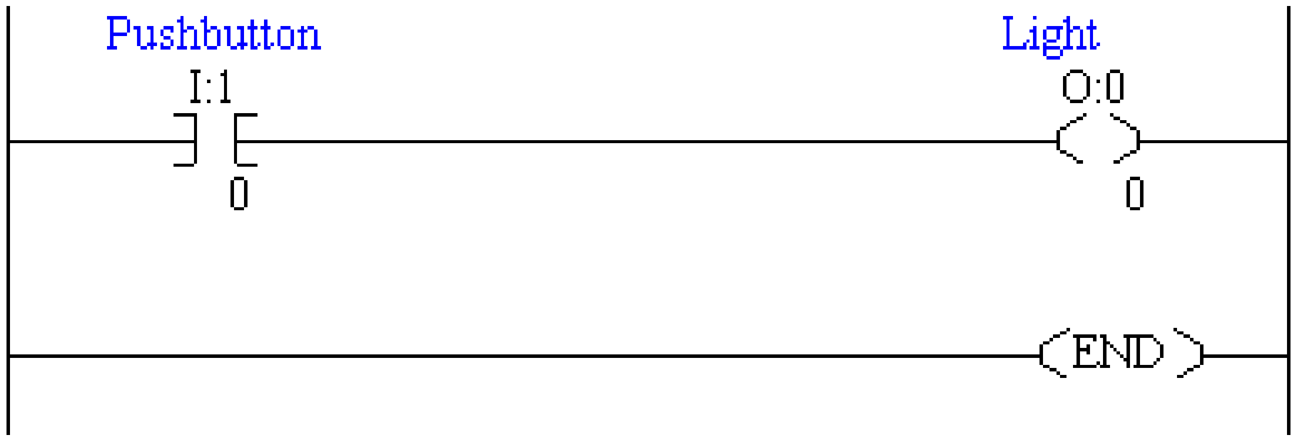


Figure 28: Ladder logic/diagram

An example of a ladder logic/diagram based code is shown above. It usually looks like a ladder which is the reason behind its name. This simplified look makes PLCs very easy to programme such that if you can analyse a schematic, you can programme PLCs.

Due to the recent popularity of modern high level programming languages, PLCs are now being programmed using these languages like C, C++ and basic but all PLCs generally still adhere to the industry IEC 61131/3 control systems standard and support the programming languages stipulated by the standard which include; Ladder Diagram, Structured Text, Function Block Diagram, Instruction List and Sequential Flow Chart.

Modern day PLC are usually programmed via application software based on any of the languages mentioned above, running on a PC connected to the PLC using any of, USB, Ethernet, RS232, RS-485, RS-422, interfaces.

Microcontrollers on the other hand are programmed using low level languages like assembly or high level languages like C and C++ among others. It usually requires a high level of experience with the programming language being used and a general understanding of the principles of firmware development. Programmers usually need to understand concepts like data structures and a deep understanding of the microcontroller architecture is required to develop a very good firmware for the project.

Microcontrollers are usually also programmed via application software running on a PC and they are usually connected to that PC via an additional piece of hardware usually called a programmers**.**

The operation of programmes on the PLC is however very similar to that of the microcontroller. The PLC uses a dedicated controller as a result they only process one programme over and over again. One cycle through the programme is called a **scan** and it's similar to a microcontroller going through a loop.
An operating cycle through the programme running on PLC is shown below.

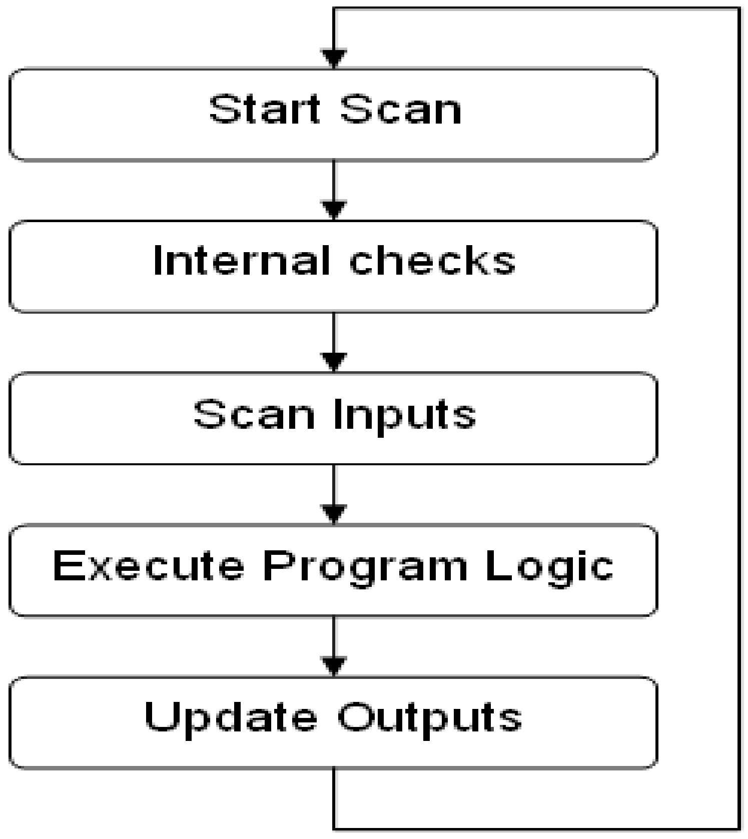


Chart 10: Operating cycle on PLC

f. Applications

PLCs are the primary control elements used in industrial control systems. They find application in the control of industrial machines, conveyors, robots and other production line machineries. They are also used in SCADA based systems and in systems that require a high level of reliability and ability to withstand extreme conditions. They are used in industries including;

1. Continuous bottle filling system
2. Batch mixing system
3. stage air conditioning system
4. Traffic control

Microcontrollers on the other hand find application in everyday electronic devices. They are the major building blocks of several consumer electronics and smart devices.

2. Microcontroller programming and hardware control

2.1. Basics of microcontroller programming

2.1.1. Installation of necessary software packages

First, we need to download the zip file "raspbian-buster.zip" used to install the Linux Raspbian operating system according to the following link https://www.raspberrypi.org/downloads/raspbian/, to your computer count.
Next we download the balenaEtcher software for Windows operating system according to the following link, https://www.balena.io/etcher/, to the computer.

This software is used for the purpose of copying OS Image files to the memory card. In addition, we can also use other software to copy the OS Image file to the memory card to help install the Linux operating system for the Raspberry Pi embedded computer.

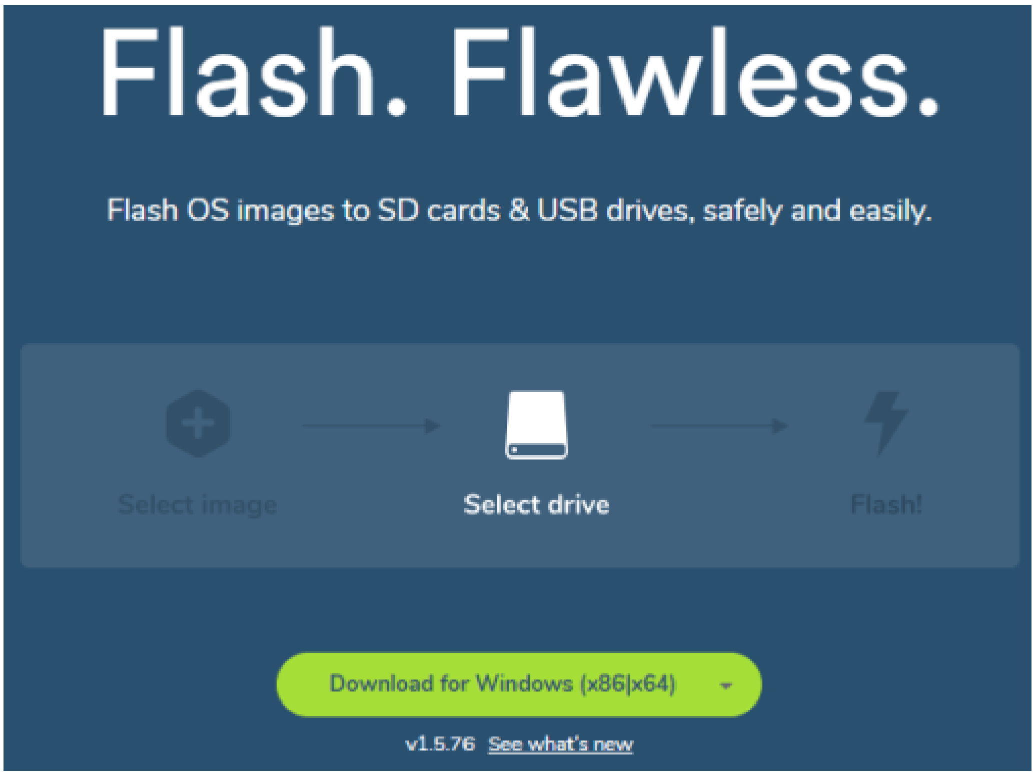
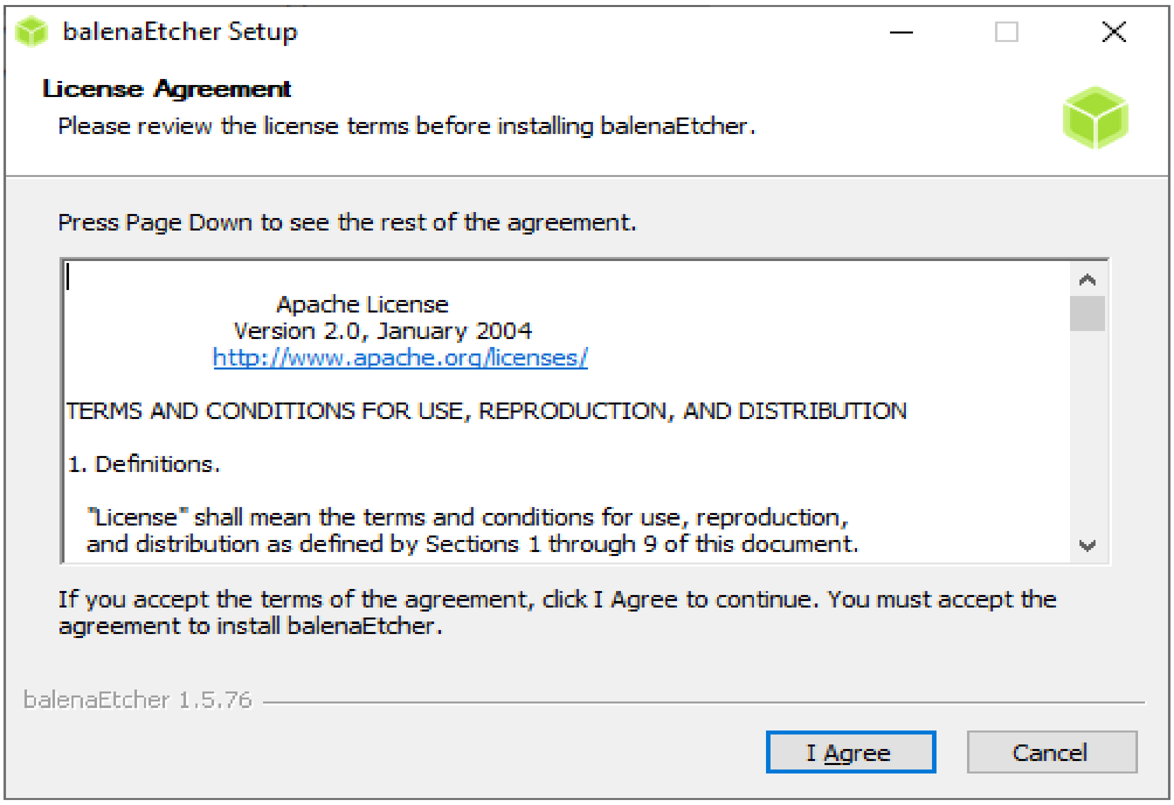

Flash. Flawless.
Flash OS images to SD cards & USB drives, safely and easily.
Select image
Select drive
Flash!
Download for Windows (x86|x64)
v1.5.76 See what's new

Figure 29: Instruction on software download

Double click on the newly downloaded file to start the installation of the balenaEtcher software. An installation window will appear on the screen and we click on the "I Agree" button to continue, as shown below.



License Agreement
Please review the license terms before installing balenaEtcher.

Press Page Down to see the rest of the agreement.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

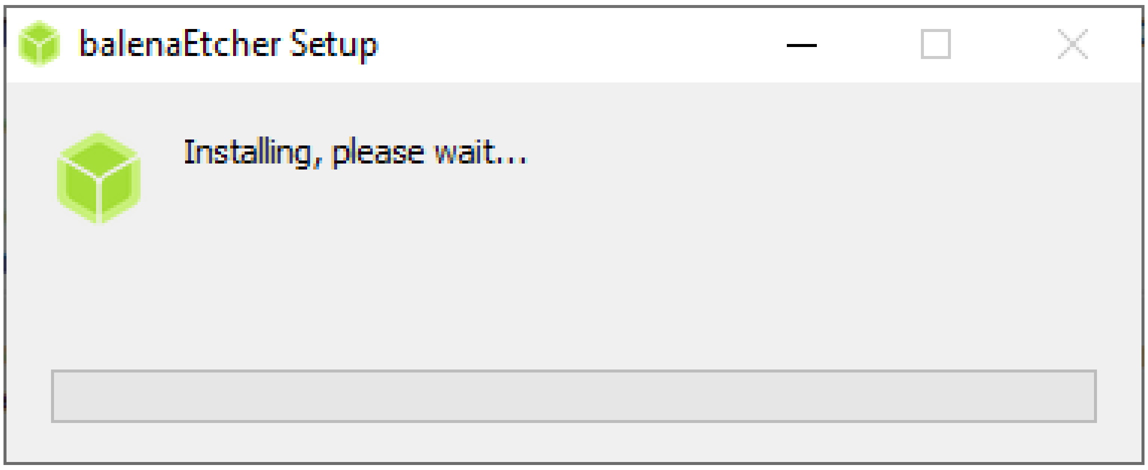
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install balenaEtcher.

balenaEtcher 1.5.76

Figure 30: Instruction on balenaEtcher Setup 1

Wait a while for the balenaEtcher software installation to complete.


balenaEtcher Setup

Installing, please wait...

Figure 31: Instruction on balenaEtcher Setup 2

After installing the balenaEtcher software for Windows operating system on the computer, continue, we will use this software to copy the OS Image file to the memory card. First, start the VirtualBox software as shown in the image below.

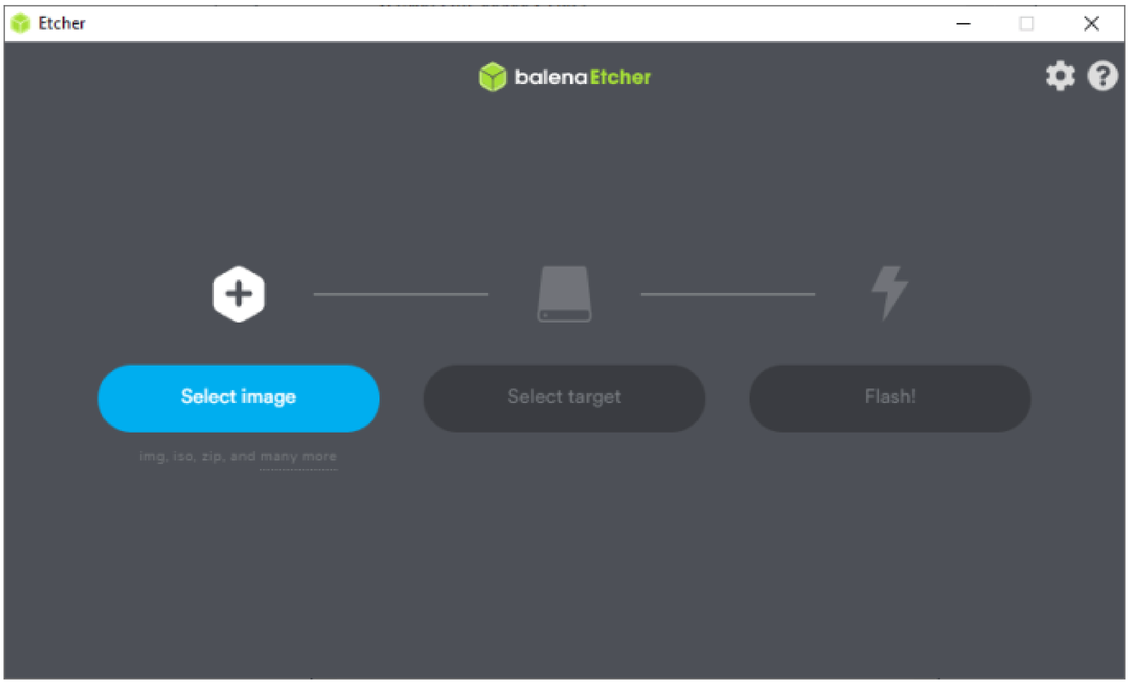


Figure 32: Instruction on balenaEtcher Setup 3

Next, insert the memory card into the reader on the computer, click the "Select Image" button and select the OS Image file "2019-09-26-raspbian-buster.zip" that was downloaded earlier, as shown in the illustration. under.

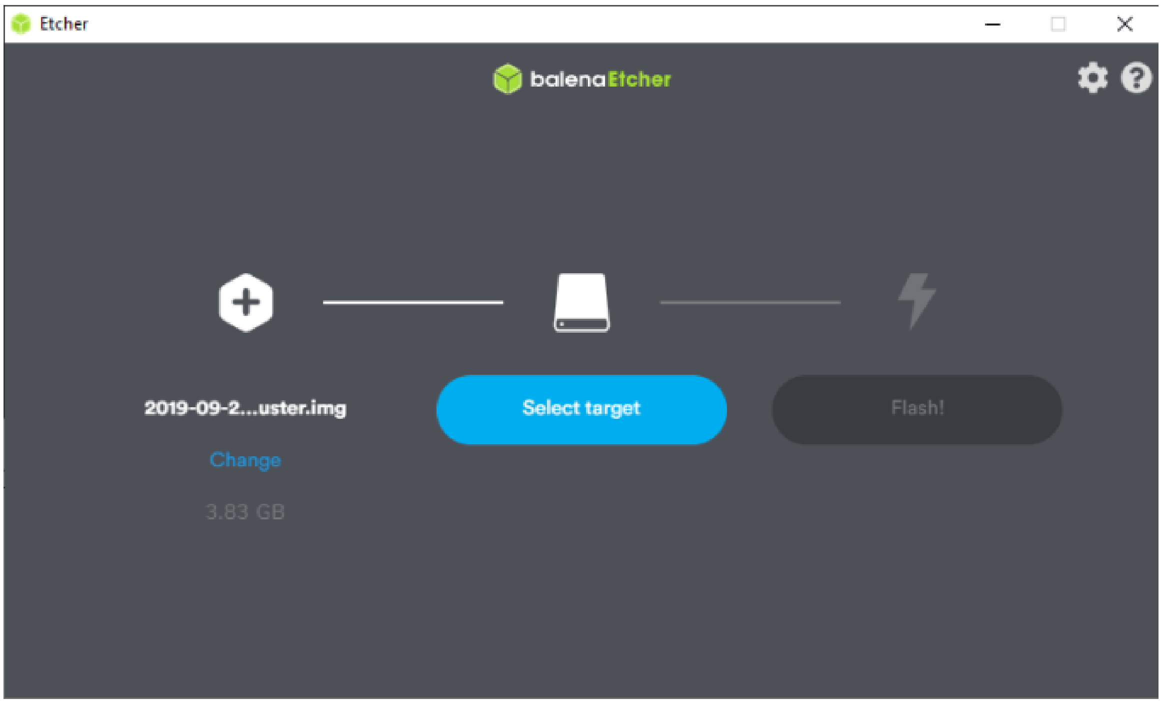


Figure 33: Instruction on balenaEtcher Setup 4

Next click on the "Select target" button to select the exact memory card to be used.

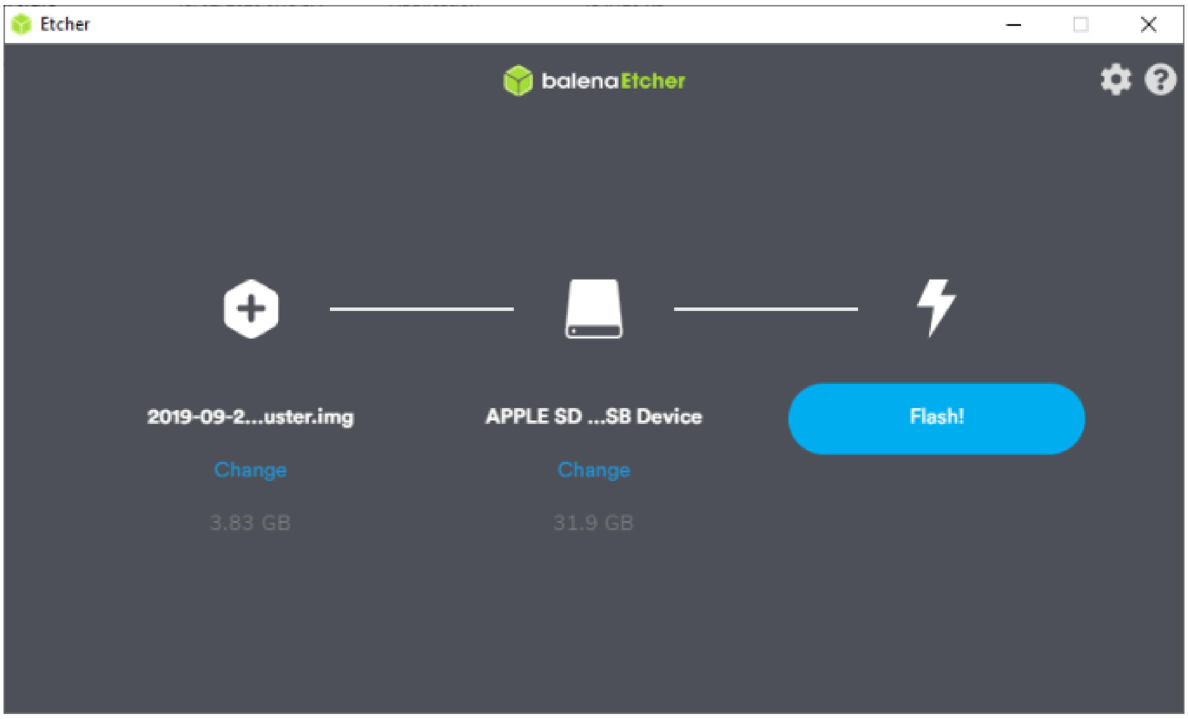


Figure 34: Instruction on balenaEtcher Setup 5

Finally click on the "Flash!" button. to copy the OS Image file to the memory card. Wait a while for the process to complete.

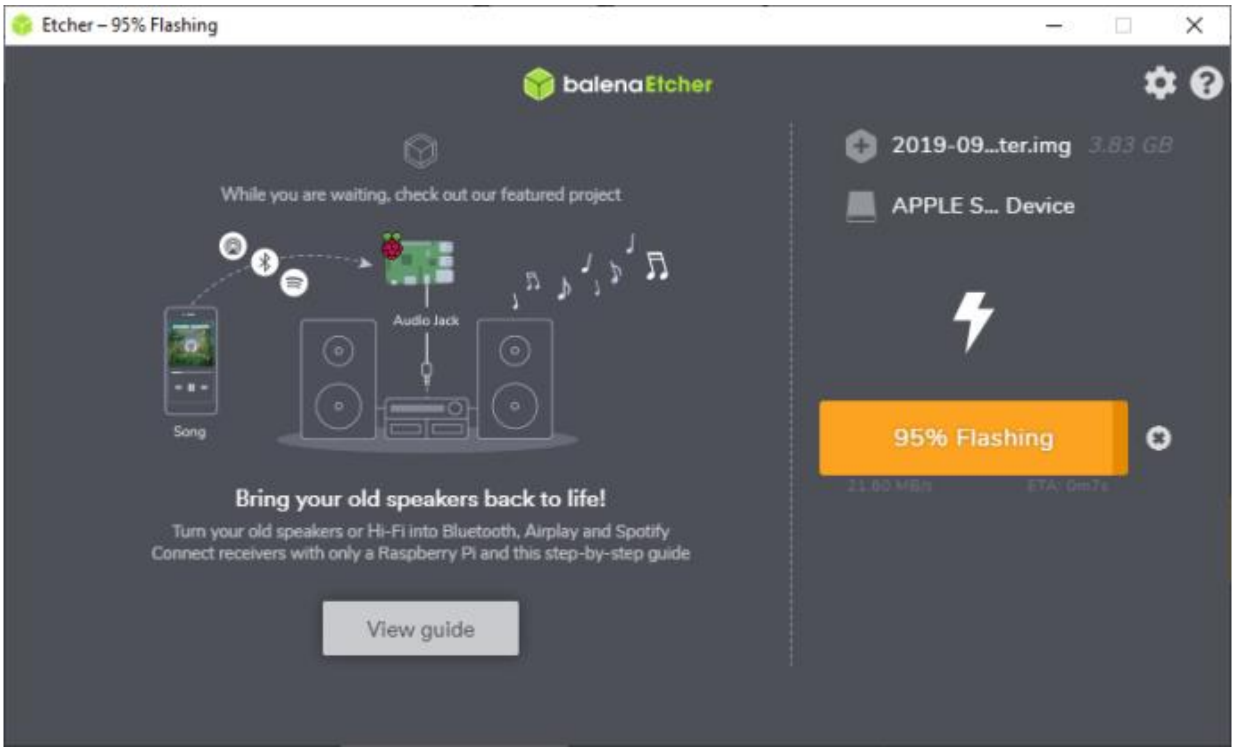
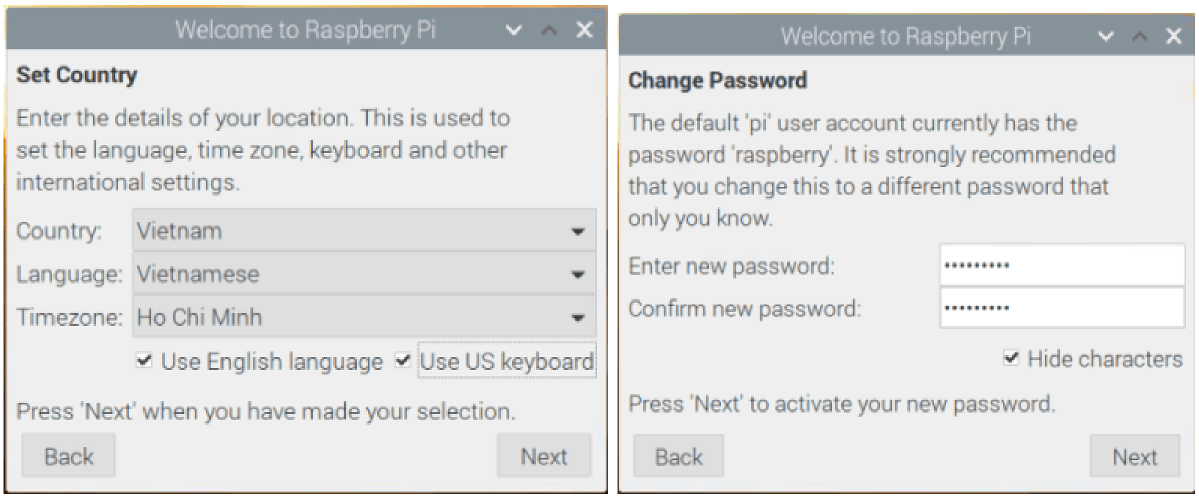

Etcher – 95% Flashing
balenaEtcher
While you are waiting, check out our featured project
Audio Jack
Song
Bring your old speakers back to life!
Turn your old speakers or Hi-Fi into Bluetooth, Airplay and Spotify Connect receivers with only a Raspberry Pi and this step-by-step guide
View guide
2019-09...ter.img 3.83 GB
APPLE S... Device
95% Flashing

Figure 35: Instruction on balenaEtcher Setup 6

Here are the complete steps to copy the Linux operating system to the memory card. Next, we just need to plug this memory card into the Raspberry Pi embedded computer and turn on the power to start the machine, continue with some initial setup operations for the Raspberry Pi such as time zone, country, language, username , login password, communication network connection, etc.


Welcome to Raspberry Pi

Set Country

Enter the details of your location. This is used to set the language, time zone, keyboard and other international settings.

Country: Vietnam

Language: Vietnamese

Timezone: Ho Chi Minh

✓ Use English language ✓ Use US keyboard

Press 'Next' when you have made your selection.

Back Next

Welcome to Raspberry Pi

Change Password

The default 'pi' user account currently has the password 'raspberry'. It is strongly recommended that you change this to a different password that only you know.

Enter new password: ••••••••••

Confirm new password: ••••••••••

✓ Hide characters

Press 'Next' to activate your new password.

Back Next

Figure 36: Instruction on Rasperry Pi setup 1

After the initial setup is complete, we need to proceed to reboot the Raspberry Pi and start being able to use the Linux operating system on the Raspberry Pi embedded computer.

Install Python software on Linux operating system

To be able to open the Terminal window quickly from the GUI, we press the key combination "Ctrl + Alt + T". Then do typing in the Linux command line to update the latest Linux OS version and install the Python software.

To update to the latest Linux desktop version, enter the following command:

```
~$ sudo apt-get update
```

To install the Python software, enter the following command line:

```
~$ sudo apt-get install python
```

2.1.2. Programming to control microcontroller peripherals

- **Event driven**
  - Programme on micro controller only gets executed when event happens
  - Detecting trigger on interrupt pin
  - Low power applications
  - Time of execution usually unclear

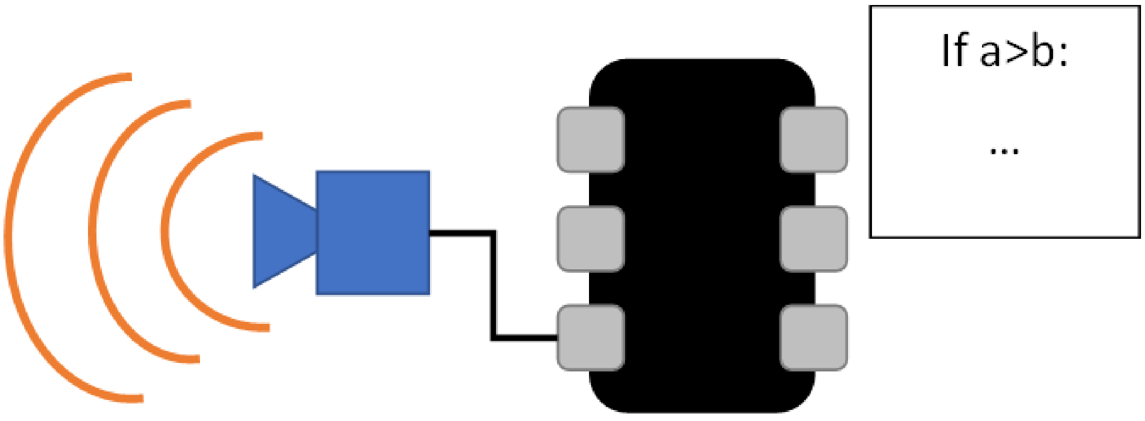


Figure 37: Illustration of event driven

- **Delay driven**
  - Programme on µC gets executed periodical
    - With software delay no multitasking capabilitys
  - Software delays still need computation time
  - Pure delay driven dose not fullfill realtime requirements

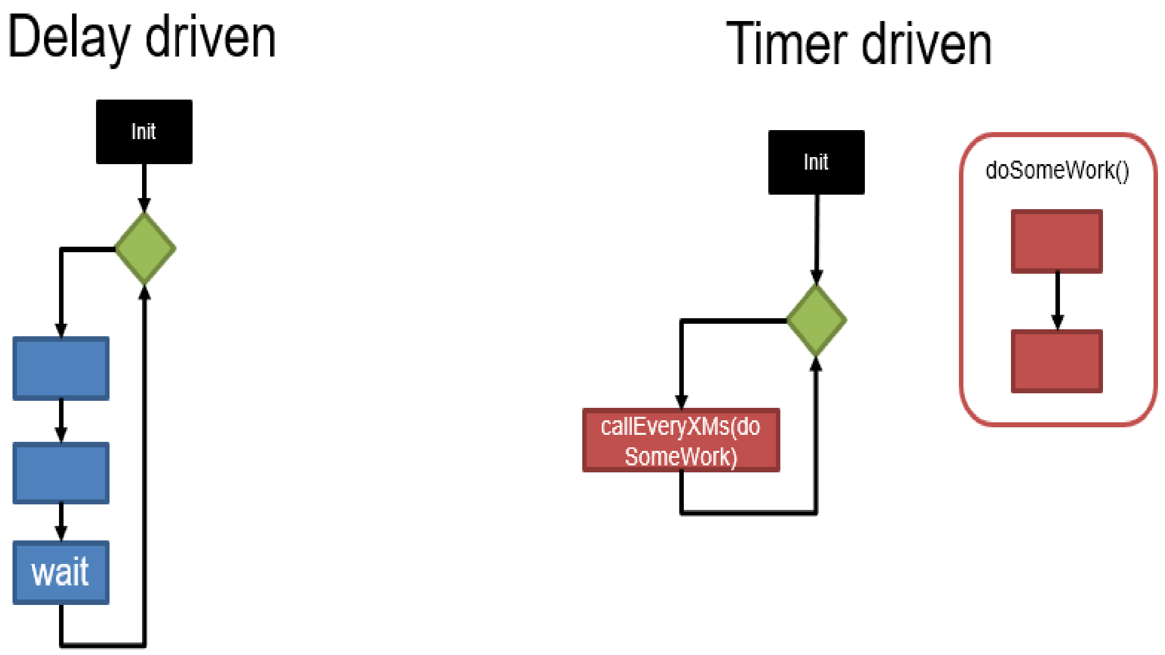
```
While True:
        …
        …
        …
    Wait(1second)
```

- Timer driven

- Progam on µC gets executed periodically
  - Timer used to call task after defined period of time
- Task has to be executed within task period
  - Including execution time of code
  - Delay from hardware access
  - Delay from communication

```
Funciton doSomeWork():
        …
        …
        …
callEveryXMs(doSomeWork())
```

- Delay driven vs Timer driven



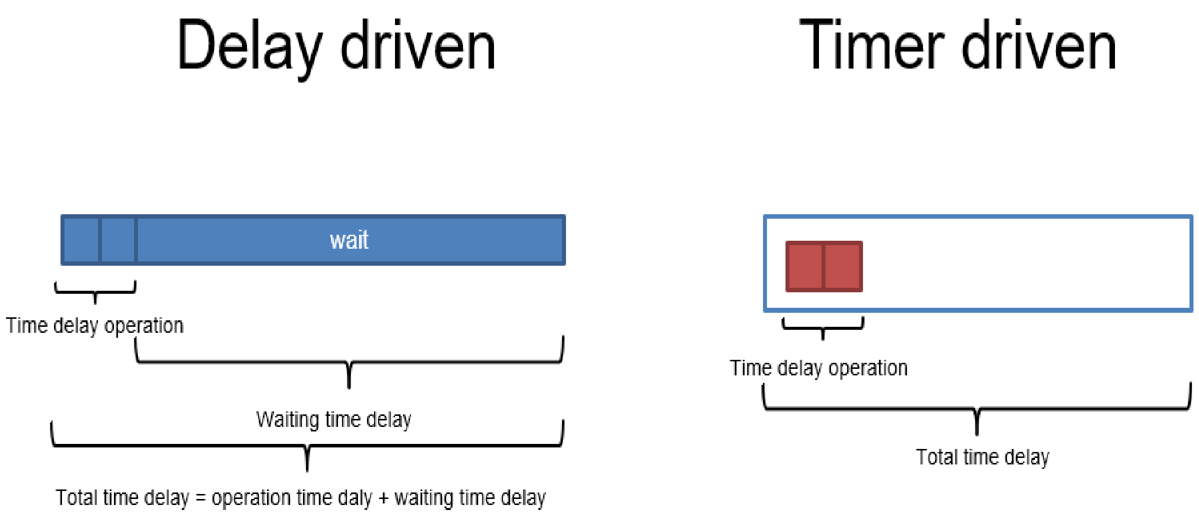


Figure 38: Comparison of Delay driven vs Timer driven

Exercise: Digital access of dual LED

Requirements:

Microcontroller libraries provide easy access to digital input and output ports. In this task the usage of digital outputs shall be trained. In this example, a dual colour led will be accessed via the digital output ports.

Dual colour LEDs, housed in a 3mm epoxy package, emit two light colours, often red and green. Two-colour LEDs have 3 lines, i.e. 2 connections, and one common cathode and common anode. The 2 connections are arranged antiparallel in the circuit and are cathode/anode connected. Positive voltage is applied to one of the two terminals, then it emits the corresponding light colour.

If the polarity of the voltage is reversed the other light colour will light up and only one of the connections can be voltage received. This type of LED is often used as indicator lights for a wide variety of devices, including televisions, digital cameras and Remote controls.

Your task is to create a programme that follows this sequence:

1. No LED shines
2. Red LED shines green dose not shine
3. Green LED shines red dose not shine
4. Both LED shine
5. Back to 1

Between every step in the sequence there shall be a delay of 1 second. The times library will provide you with the functionality for creating a delay. You can use any valid GPIO pin. Ensure that the software pin corresponds to the hardware pin. To find valid pins study the GPIO modules description.

Steps:

1. Connect to your microcontroller
2. Store the corresponding hardware pin numbers in a variable
3. Setup
   a. Initialize the GPIO board
   b. Set the corresponding GPIO pins as output
   c. Set the output state of the GPIO pins to low
4. In the main loop
   a. Implement the sequence of LED from before
5. Prepare a destroy routine in case the programme execution gets stopped

Reference solution:

Solution in pseudo code (Pin numbers depend on hardware implementation)

```
• redPin = Nr1
• greenPin =Nr2
• In setup function
    o Set redPin and greenPin as output
    o Set redPin and greenPin to state low
• In main loop
    o While true
        ▪ Set redPin and greenPin to low
        ▪ Delay 1second
        ▪ Set redPin high, set greenPin low
        ▪ Delay1second
        ▪ Set redPin low, set greenPin high
        ▪ Delay 1 second
        ▪ Set redPin and greenPin to high
        ▪ Delay 1 second
```

2.1.3. Software architecture in the microcontroller environment

Software Architecture

Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
  - Selection of structural elements and their interfaces by which the system is composed.
  - Behavior as specified in collaborations among those elements.
  - Composition of these structural and behavioral elements into large subsystem.
  - Architectural decisions align with business objectives.
  - Architectural styles guide the organization.

Microcontroller Operating Systems

- Requirements:
  - Possilbity to start, stop and supervice tasks
  - Precice controll of calling
  - Multitasking support
  - Fullfill real time requirements

Real time

- Real time according to DIN 44300 Teil 9(original):

„Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“

- Real time according to DIN 44300 Teil 9 (translated):

"Real-time means the operation of a computer system in which programmes for processing accruing data are constantly operational in such a way that the processing results are available within a specified period of time. Depending on the application, the data can be generated according to a temporally random distribution or at predetermined points in time."

Real time operating systems (RTOS)

- RTOS
  - Manages secure processing of requests from an application program or the arrival of signals via hardware interfaces within a period that can be determined in advance
  - Period can be high or low depending on the applications requirements
  - Important is to finish execution within the periods duration
- Soft RTOS
  - OS used when failing of execution in duration leads to problematic system states
  - Deadline has to be kept most of the times

Figure 39: Raspberry Pi4

- Hard RTOS
  - OS used when failing of execution in duration leads to catastrophic events
  - Deadline always has to be kept

Figure 40: XDK sensor

Hard vs Soft RTOS

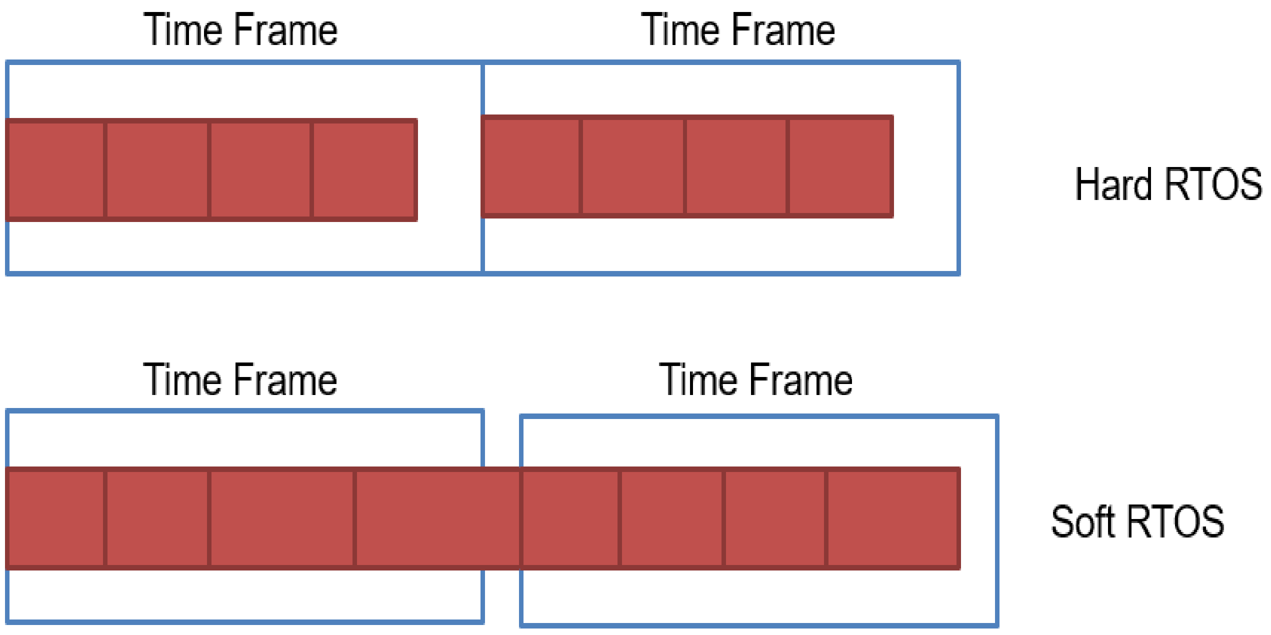


Figure 41: Illustration of Hard vs Soft RTOS

Basic structure µC

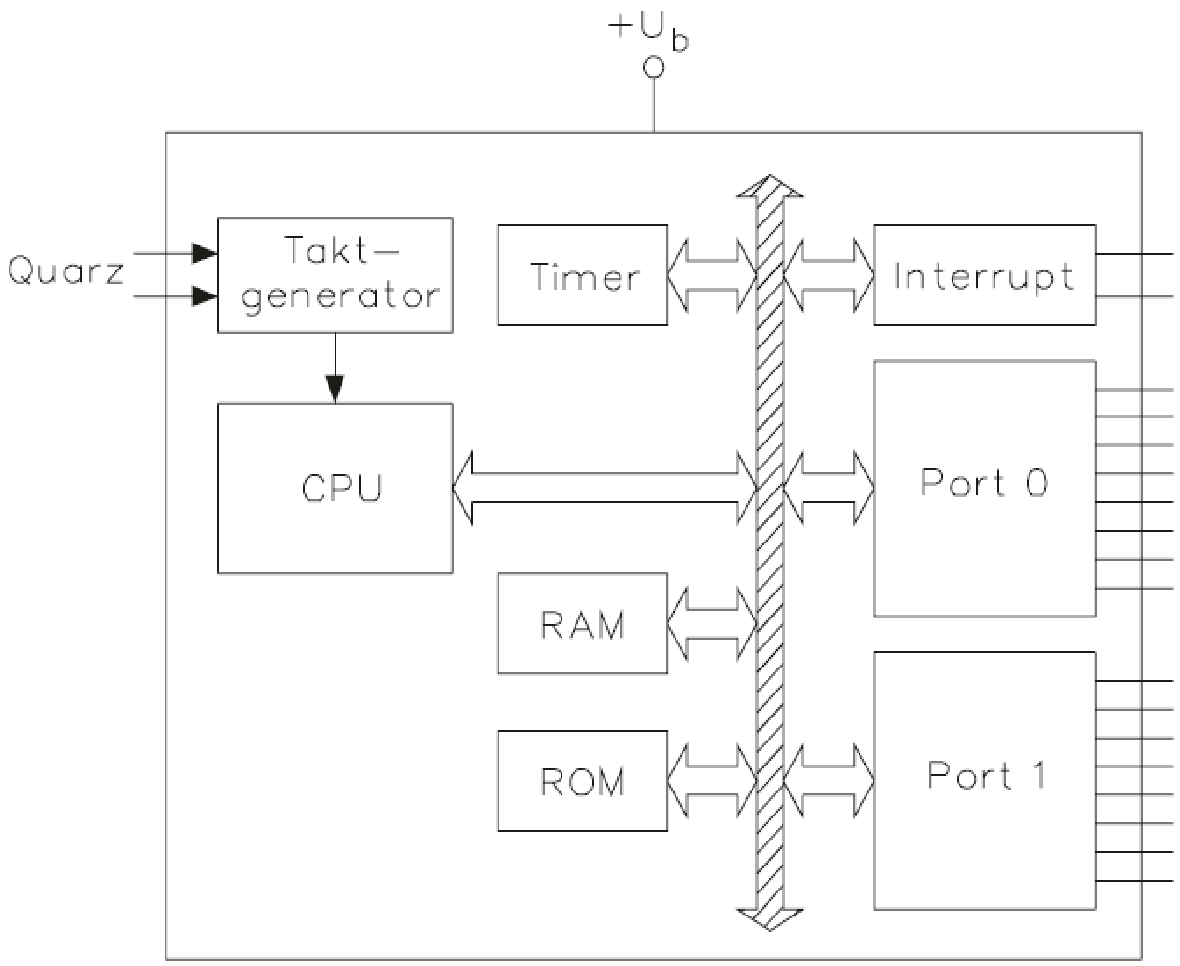


Chart 11: : I/O structure of the microcontroller

Functional units µC

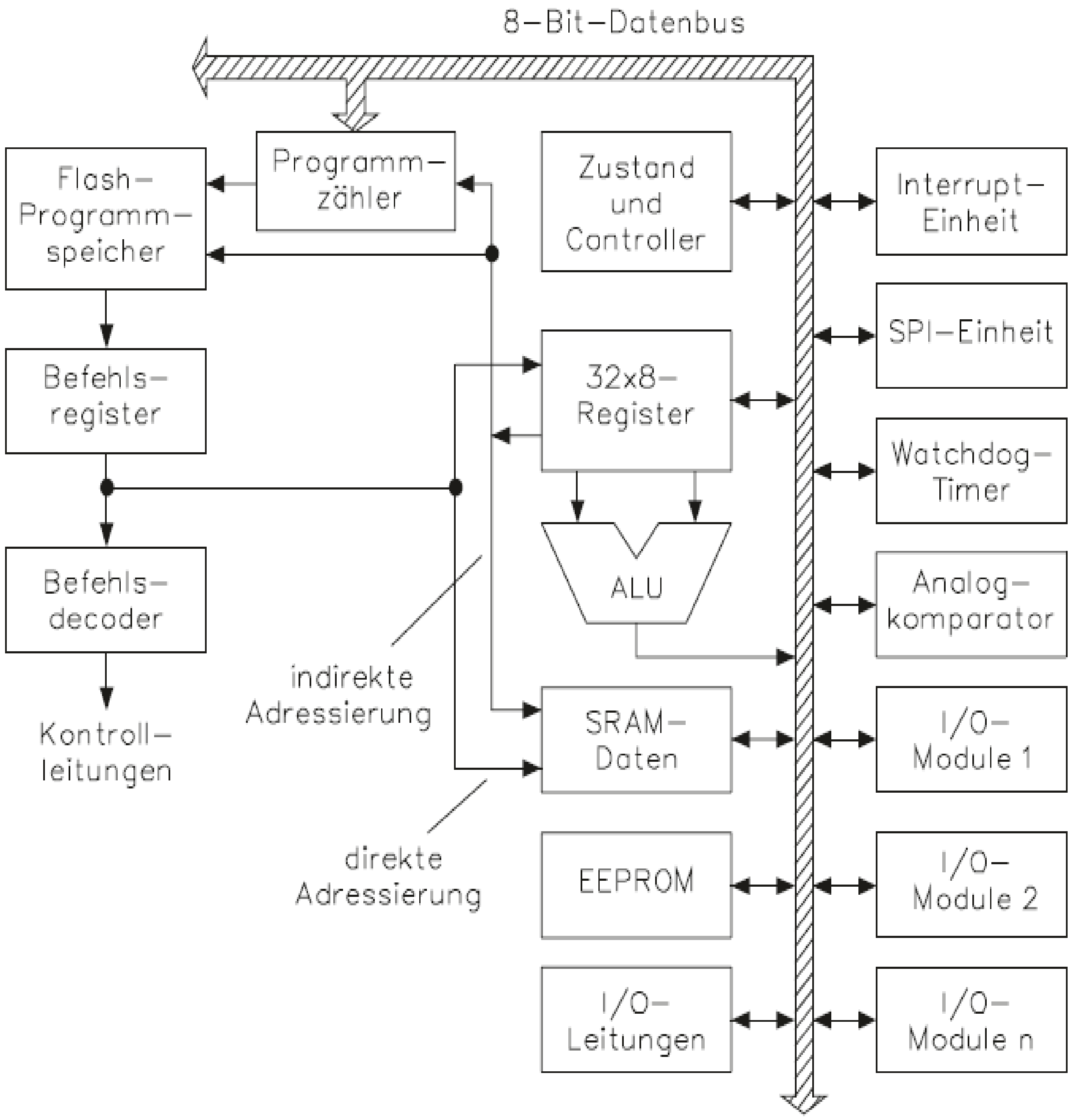


Chart 12: Function block diagram of microcontroller

Memory distribution

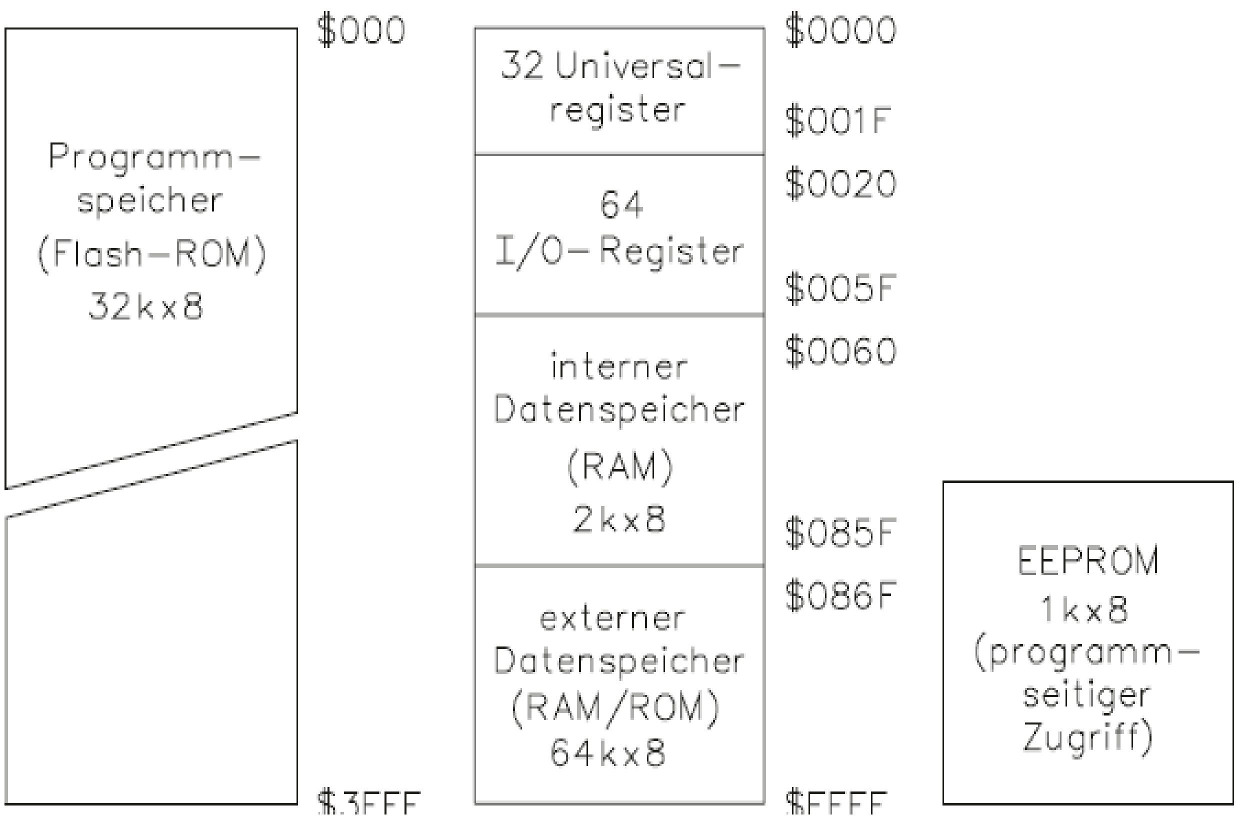


Chart 13: Memory distribution in microcontrollers

2.2. Hardware control

This chapter will provide tasks for basic hardware control. The real implementation will depend on which type of hardware is available. The tasks can be altered depending on which hardware is provided. Originally, this exercise were designed for microcontroller, which use 3.3V output voltage. If a microcontroller with 5V output voltage is used the basic task instructions can be used without change, but the usage of the correct sensor and actuators which are 5V compatible has to be ensured. Do not use 3.3V hardware on 5V hardware and vice versa.

2.2.1. Control of external hardware (actuators and sensors)

Reading in digital input

There are two main ways on how to detect changes on hardware and react to them. The first one is via polling and the second one via interrupt.

Detection via polling

Requirements:

When a digital input is detected via polling the user needs to create a programme where the microcontroller sequentially reads in the digital input and detects if its set to high or low. This way of programming will cost computation power and energy with every iteration. For simple programmes where energy efficiency is not mandatory this technique can be used.

Create a programme that sets the dual LED to green colour if a button is not pressed and to red colour if the button is not pressed. Use board GPIO pins to your desire.

Steps:

1. Wire the button and the LED with the board
2. Store the corresponding hardware pin numbers in a variable
3. Setup
    a. Initialize the GPIO board
    b. Set the corresponding LED GPIO pins as output
    c. Set the output state of the LED GPIO pins to low
    d. Set the corresponding button GPIO pin as input
        i. Set the internal pull up resistor to pull up "pull_up_down=GPIO.PUD_UP"
4. Loop
    a. Read in the digital input
    b. Check if button was pressed
    c. Implement the logic above depending on the result

Reference solution:

Solution in pseudo code (Pin numbers depend on hardware implementation)

```
• redPin = Nr1
• greenPin =Nr2
• btnPin = Nr3
• In setup function
    o Set redPin and greenPin as output
    o Set redPin and greenPin to state low
    o Set btnPin as input and activate the pullup resistor
• In main loop
    o While true
        ▪ btnVal = digitalRead(btnPin)
        ▪ if btnVal == pressed
            • set redPin to high
            • set greenPin to low
        ▪ else
            • set redPin to low
            • set greenPin to high
        ▪ delay 100ms
```

Detection via interrupt

Requirements:

When a digital input is detected via interrupt then it’s not necessary to manually call the read in function. Instead, an interrupt function is created that listens to a rising or a falling edge (depending in implementation) on the corresponding pin. If the state changes then the interrupt function will call a callback function to be executed. This style of programming permits very power and computation efficient programming.

Upgrade the programme from detecion via polling for reading in via interrupt. Use the “GPIO.add_event_detect(PINNr, GPIO.BOTH, callback=detect, bouncetime=200)”. The bouncetime is set to 200 to avoid getting wrong interrupt calls by the buttons bouncing.

The interrupts callback function is called detect. This function shall implement the same detection logic as detecion via polling.

Steps:

1. Wire the button and the LED with the board
2. Store the corresponding hardware pin numbers in a variable
3. Setup
    a. Initialize the GPIO board
    b. Set the corresponding LED GPIO pins as output
    c. Set the output state of the LED GPIO pins to low
    d. Set the corresponding button GPIO pin as input
        i. Set the internall pull up resistor to pull up “pull_up_down=GPIO.PUD_UP”
    e. Set the interrupt detection function according to above requirements
4. Outside of setup
    a. Create a function “detect”
        i. Read in the digital input
        ii. Check if button was pressed
        iii. Implement the logic above depending on the result
5. Loop
    a. Create a while true loop
        i. Do nothing in the loop “pass”

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)

- redPin = Nr1
- greenPin =Nr2
- btnPin = Nr3
- In setup function
    - Set redPin and greenPin as output
    - Set redPin and greenPin to state low
    - Set btnPin as input and activate the pullup resistor
    - Create interrupt service routine, attach function detect set bounce time to 200ms, activate falling and rising edge detection

```
• In function detect
    o btnVal = analogeRead(btnPin)
        ▪ if btnVal == pressed
            • set redPin to high
            • set greenPin to low
        ▪ else
            • set redPin to low
            • set greenPin to high
• In main loop
    o While true
        ▪ pass
```

2.2.2. Use of pulse width modulation for hardware control

Requirements:

Pulse width modulation is a very common method to for example dim light or control engines speed. In this task a RGB LED shall be used. RGB LED emits light in different colours. In its transparent or semi-transparent plastic housing, with four pins, it is equipped with three LEDs: red, green and blue. With different luminance's, the three primary colours mix to different colours and by controlling the circuit, you can make the RGB LED emit colourful light.

Create a programme the sets the RGB LED to 10 different colour patterns. Let the led switch between the patterns with 300ms delay. Figure displays the RGB colour spectrum here you can take it as a refence to decide which colours you like to choose.

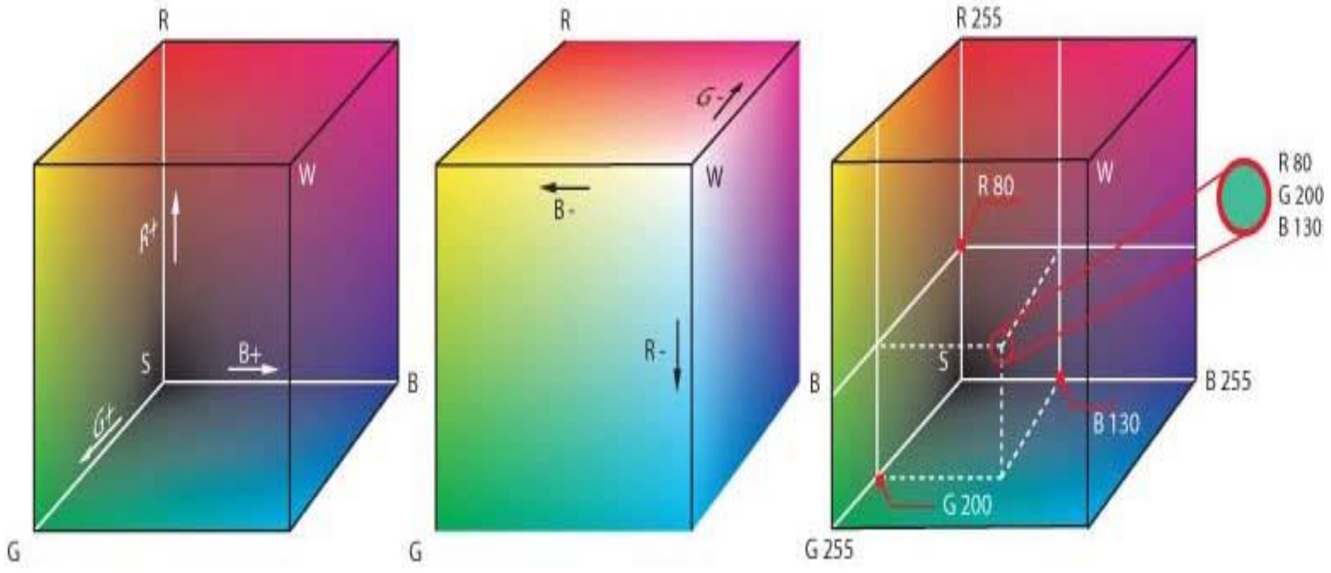


Figure 42: Pulse width modulation

Usually, RGB is defined between 0 and 255 for each colour. Depending on the used library it is possible that the PWM either gets 0 to 100 where 0 means no Pulse and 100 means full dc output, or 0 and 255, where 0 means no pulse and 255 means 100% output. Consider to research which type of input for the PWM of your microcontroller is needed. If possible, use 2000hz as PWM frequency.

Steps:

1. Wire RGB LED to microcontroller
2. Decide on colour patterns
3. Save colour patterns in list
4. Setup
    a. Set the corresponding LED GPIO pins as output
    b. Set the output state of the LED GPIO pins to low
    c. Set the selected pins to PWM pins
5. Loop
    a. Create a for loop
    b. Loop through every colour combination
    c. Use the "ChangeDutyCycle(dc)" to change the colour to the desired value
    d. Use a 300ms delay

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)

- redPin = Nr1
- greenPin =Nr2
- bluePin =Nr2
- colourListR = [cR1, …, cR10]
- colourListG = [cG1, …, cG10]
- colourListB = [cB1, …, cB10]
- In setup function
    - redPWM = set redPin, greenPin and bluePin as output
    - greenPWM = set redPin, greenPin and bluePin to state high
    - bluePWM = set redPin, greenPin and bluePin as PWM pins with 2000hz
    - Initiate PWM with 100% duty cycle

```
• In main loop
    o While true
        ▪ For i = 0 to 9
            • Set redPWM duticycle to colourListR[i]
            • Set greenPWM duticycle to colourListG[i]
            • Set bluePWM duticycle to colourListB[i]
            • Delay 300ms
```

2.2.3. Use of ADC (analog to digital converter) for reading analog signals

Reading in sensor values

Microcontroller or extension boards often have the possibly to read in sensor values via analogue to digital converters. These converters convert (if they are a voltage interface) a voltage into a number representation. The resolution depends on the type of ADC converter. Nowadays 8 – 12 Bit converters are common.

In this example a thermistor will be used as a temperature sensor. A temperature sensor records the temperature and converts it into output signals. Temperature sensors can be divided into two types by material and component characteristics: Thermistor and thermocouple. A thermistor is one of the earlier types, made of semiconductor materials and usually negative temperature coefficient (NTK), whose resistance decreases with increasing temperature. Since the resistance changes acutely with temperature changes thermistors are the most sensitive temperature sensors.

Read in the voltage of the temperature sensor. Calculate the thermistors value based on the voltage read from it. You can assume that the thermistor is in a voltage divider as in 43.

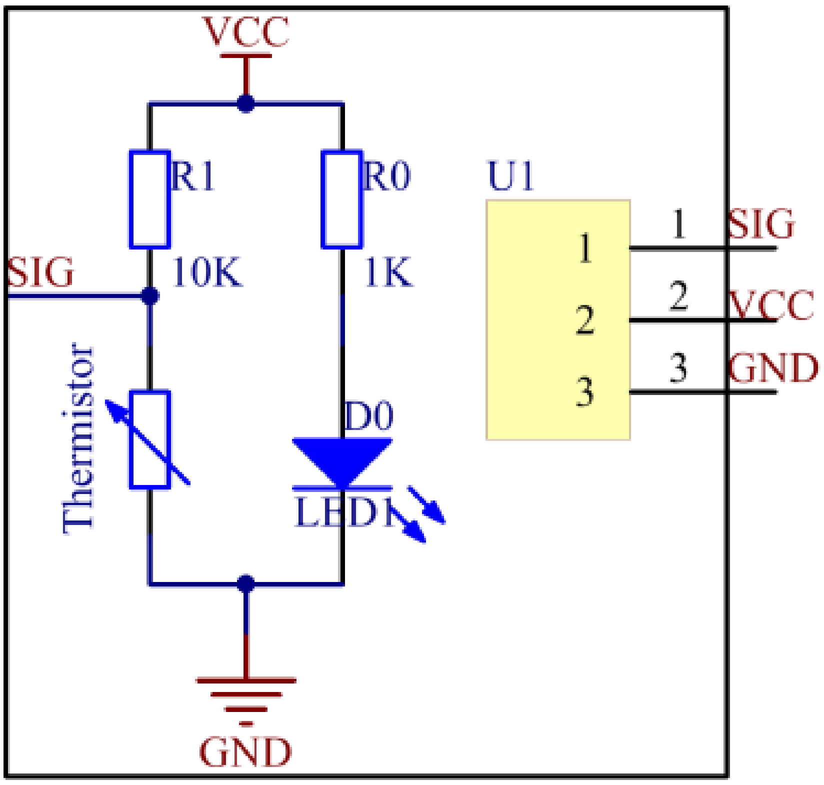


Figure 43: Sensor Set 2.0 für Raspberry Pi 4 Modell B

The Thermistors base resistor value can be read out of the datasheet. Further, use the datasheet and or internet to research a equation to calculate the temperature from the calculated thermistor resistance.
Steps:

1. Wire the thermistor to the microcontroller board
2. Setup
    a. Initiate the ADC converter
    b. Setup the ADC pin as input
3. Loop
    a. Reading the thermistor value
    b. Calculate the thermistor voltage (careful if 5V or 3.3V)
    c. Based on the thermistor voltage calculate the thermistor resistance
    d. Research a method to calculate the thermistors value based on the datasheet and or internet research (example approximation with datasheet values)
    e. Print temperature values
    f. Delay of 500ms between every measurement

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)
Voltage and resolution depends on the hardware

```
• aInPin = Nr1
• In setup function
    o Initiate ADC
    o Set  aInPin as input
• In main loop
    o While true
        ▪ adcVal = analogeRead(aInPin)
        ▪ voltageVal =  Voltage * float(adcVal) /resolution
        ▪ Rt = 10000 * voltageVal / (5 - voltageVal)
        ▪ temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
        ▪ temp = temp - 273.15
        ▪ print(temp)
        ▪ delay 500ms
```

2.2.4. Measurement conversion from digital value to physical quantity
Distance measurement
Requirements:

The provided code reads in a ultrasonic sensor. At first the provided code shall be understood. Afterwards the programme is to be updated so that if a distance of a object is below 7cm then the warning message: “Danger object is too close” shall be outputted.

Code:

```
import RPi.GPIO as GPIO
import time
TRIG = 11
ECHO = 12
# Setup Pins
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
#Calculate Distance
def calcDistance():
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)
    GPIO.output(TRIG, 1)
    time.sleep(0.00001)
    GPIO.output(TRIG, 0)
    while GPIO.input(ECHO) == 0:
        a = 0
    time1 = time.time()
    while GPIO.input(ECHO) == 1:
        a = 1
    time2 = time.time()
    #calculate time between start and stop
    timeDel = time2 - time1
    # convert time to distance
    return timeDel * 340 / 2 * 100
def loop():
    while True:
        dis = calcDistance()
        time.sleep(0.3)
          def destroy():
    GPIO.cleanup()
if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

Steps:

1. Understand provided code
2. Update code so that distances below 7cm output a warning

Reference solution

Added code segment is marked in yellow.

```
import RPi.GPIO as GPIO
import time
TRIG = 11
ECHO = 12
# Setup Pins
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
#Calculate Distance
def calcDistance():
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)
    GPIO.output(TRIG, 1)
    time.sleep(0.00001)
    GPIO.output(TRIG, 0)
    while GPIO.input(ECHO) == 0:
        a = 0
    time1 = time.time()
    while GPIO.input(ECHO) == 1:
        a = 1
    time2 = time.time()
    #calculate time between start and stop
    timeDel = time2 - time1
    # convert time to distance
    return timeDel * 340 / 2 * 100
def loop():
    while True:
        dis = calcDistance()
if dis < 7:
print("Danger object is too close")
        time.sleep(0.3)
          def destroy():
    GPIO.cleanup()
if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

3. Create measuring programmes

3.1. Introduction to state machines

3.1.1. Fundamentals of state machines

A state machine is a system that can be described in terms of a set of states that the system may enter. The next state reached depends on the inputs and the present state. The outputs also depend on inputs and the present state. A state may be a set of values measured at various points in a circuit. A simple flip-flop has two states in which it can exist. A large majority of practical state machines use locked flip-flops as the storage elements. The code that defines each state then corresponds directly to the code contain by the flip-flops. The general model of the sequential machine is shown in figure below.

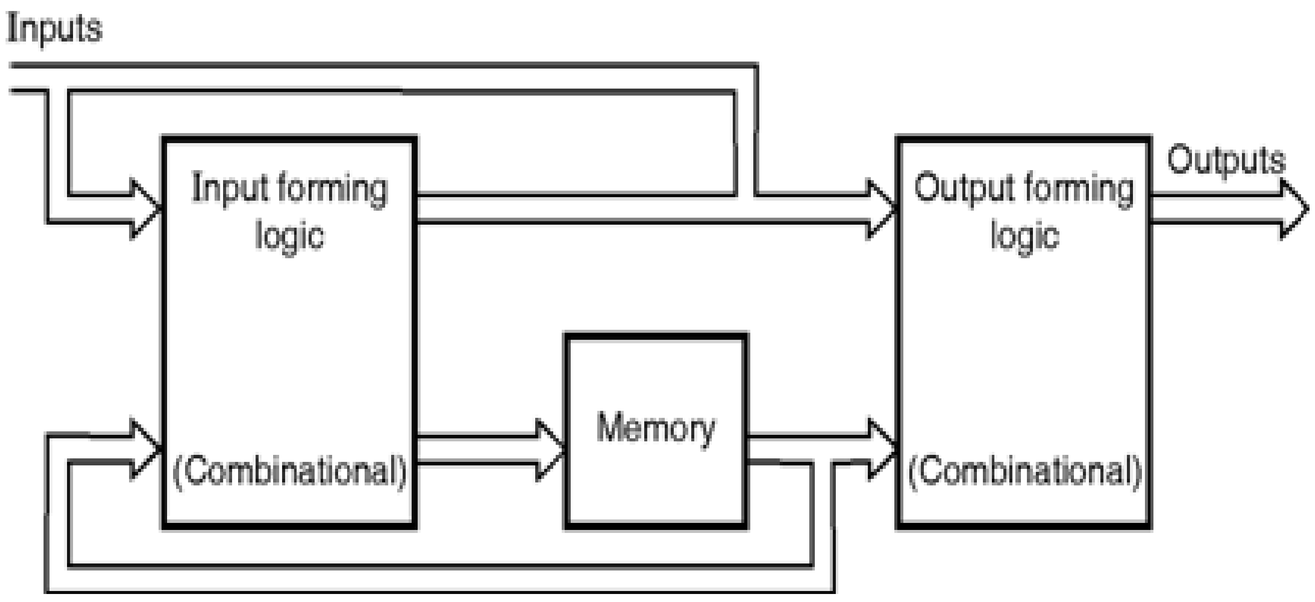


Figure 44: Fundamentals of state machines

General model of sequential circuit

This model is also called the Mealy machine after the man who first proposed the model. The input forming logic (IFL) and the output forming logic (OFL) sections are made up of combinational logic circuits. The memory section contains the state of the system. A slight variation of the Mealy machine is the Moore machine which uses only the memory to drive the OFL. In this case, the output is a function of only the state of the system. When a variable is clock-driven, it is considered a synchronous variable. If a machine changes state in response to the clock and all inputs are synchronous, we will classify that circuit as a synchronous system. If the state changes occur in response to the clock, but one or more inputs are not clock-driven, the machine will be called a mainly synchronous system. If the state changes are input-driven rather than clock-driven, the system is an asynchronous one.

3.1.2. Implementation of state machines for microcontrollers

State machines are useful tools that in the right application can simplify designing microcontroller firmware. They allow you to create an event-driven system that can change its response to inputs based on its internal state. The example below introduces one way to structure a state machine in a microcontroller environment.

Consider an example system that contains a microcontroller connected to an LED and a push button. In this example system one press of the button turns on the LED, a second push of the button will make the LED blink, and if the button is pressed again the LED will turn off. Also, our system must turn off the LED after a period of inactivity. If the button hasn't been pressed in the last 10 seconds, the LED will turn off.

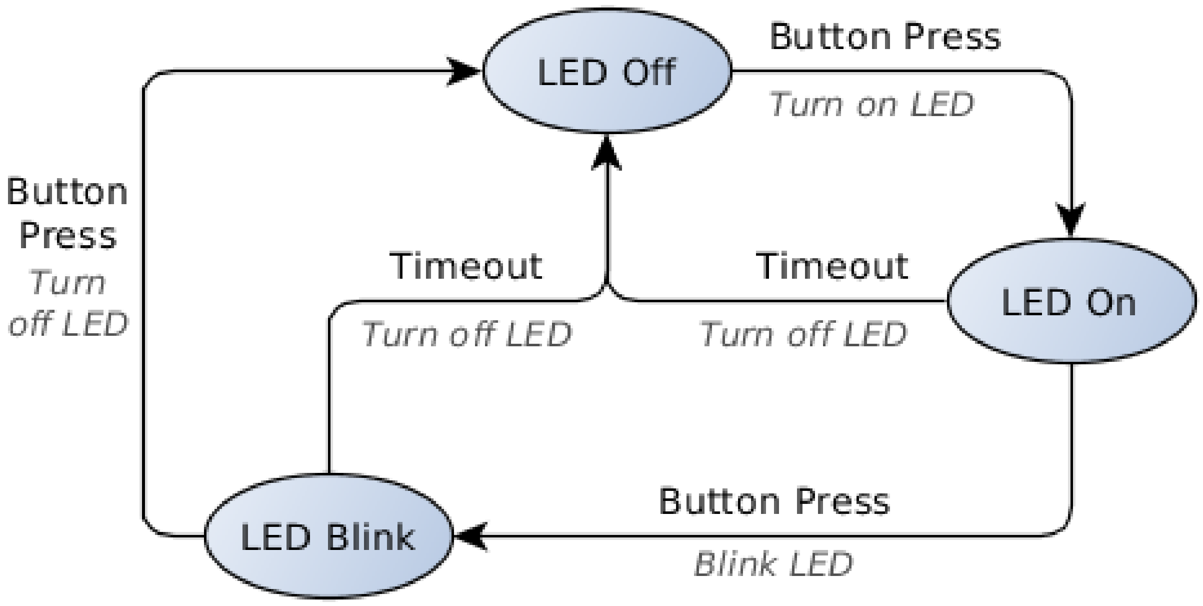


Figure 45: Example system state diagram

In our example, every time the button is pressed, the system must take one of three possible actions (turn on the LED, blink the LED, or turn off the LED). The response to a button press depends on the current status of the LED, which can be affected by another event in the system (the 10-second timeout). We must therefore create a system that keeps track of the LED and uses its status to decide what to do in response to an input.

A state machine will generate an output determined by both the current internal state of the system and the input it receives. By changing the state, the system can generate a different output given the same input as before. If we use the states to keep track of the LED, then we can determine which output to generate when the button is pressed.

State Machine Structure

For microcontroller applications, let's use the definition of a finite-state machine. A finite-state machine has a known set of inputs, outputs, and states. The state diagram in Figure 1 contains all the elements we need to describe our state machine; let's define the four different elements of a state machine.

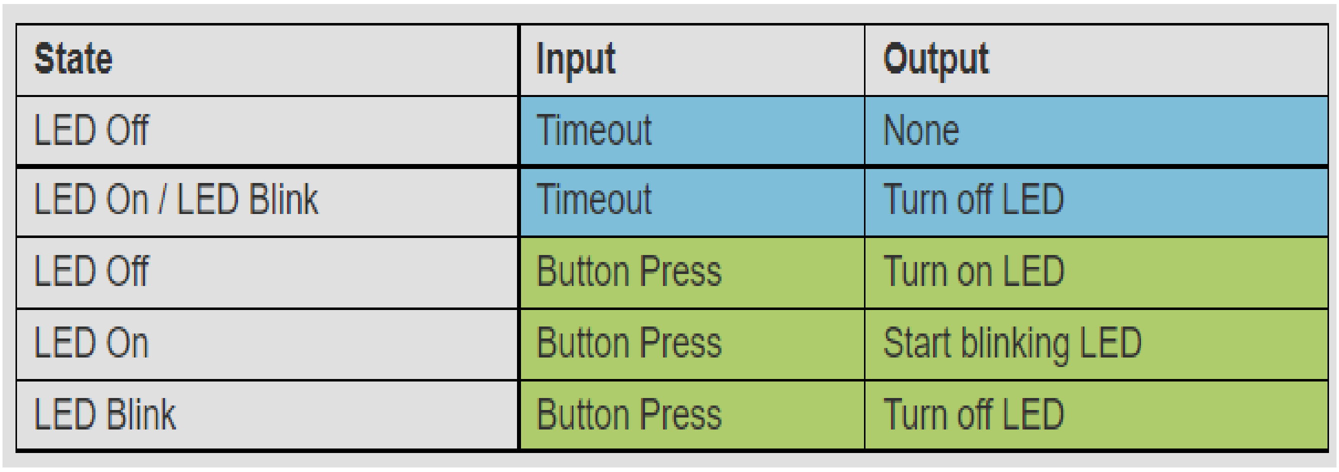
Inputs—any event that requires our system to generate an output or change its behaviour is an input. Our example has two inputs: a 10-second timeout and a button press. In our state diagram, the inputs are listed above the arrows connecting the states.

State transitions—the arrows in the state diagram represent state transitions. These define when our system will change its behaviour by changing its internal state. State transitions can only be triggered by an input. In our example, we trigger a state transition every time an input produces an output that changes the status of the LED. This way the system's state always keeps track of the LED. The state transitions also define how our system is allowed to change behaviour. For example, there is no arrow connecting the LED off state and LED blink state, therefore the LED can never change directly from the off state to the blink state.

Outputs—actions that need to be taken by the system in response to an input are outputs. In Figure 1, the outputs are listed in italics under the state transition arrows. Like state transitions, our system can only generate an output following an input event. Our system has three outputs: making the LED turn on, blink or turn off.

States—the circles in the state diagram represent the states. A state is a list of rules that tells the system what to do when an input event occurs. When an input occurs, the system will look at the set of rules defined in its current internal state and look up which output or state transition it needs to generate, or whether it should do nothing at all in response to the input. Our states are the three different LED behaviours we outlined earlier; the state machine is only allowed to exist in one of these states.

But how do we decide what our states will be, or even how many we should have? Each state is a different set of rules that define how the system will respond to the inputs, so all the states need to cover the different output behaviours we need. Let's take a look at the input-output relationships of our example in a table instead of the state diagram above.



State	Input	Output
LED Off	Timeout	None
LED On / LED Blink	Timeout	Turn off LED
LED Off	Button Press	Turn on LED
LED On	Button Press	Start blinking LED
LED Blink	Button Press	Turn off LED

The button press input can produce three possible outputs based on our example description. Because the output is dependent on the state and the input, the only way one input is allowed to produce three possible outputs is to have three states. A timeout event results in two possible actions, but these can be included in our three other states.

Finally, it's important to define the initial state of our system. This is the state that the system will start in on power-up, or if a reset occurs. Our state machine will always start in the LED off state.

3.1.3. Software design of a sequence programme via state machines

The Hardware

Before we implement the different elements of our state machine in code, let's look at the hardware we'll use.

The example hardware can be put together with an Arduino, a momentary switch and a capacitor to debounce the switch. The switch is connected across the digital 2 and 4 pins on the Arduino (PD2/INT0 and PD4, respectively on the Atmega) with a suitable valued debounce capacitor (100nF) in parallel. Pin 4 is set as a sink and pin 2 is set up as an input with internal pull-up enabled. The example also uses the on-board LED connected to pin 13 (PB5/SCK on the Atmega).

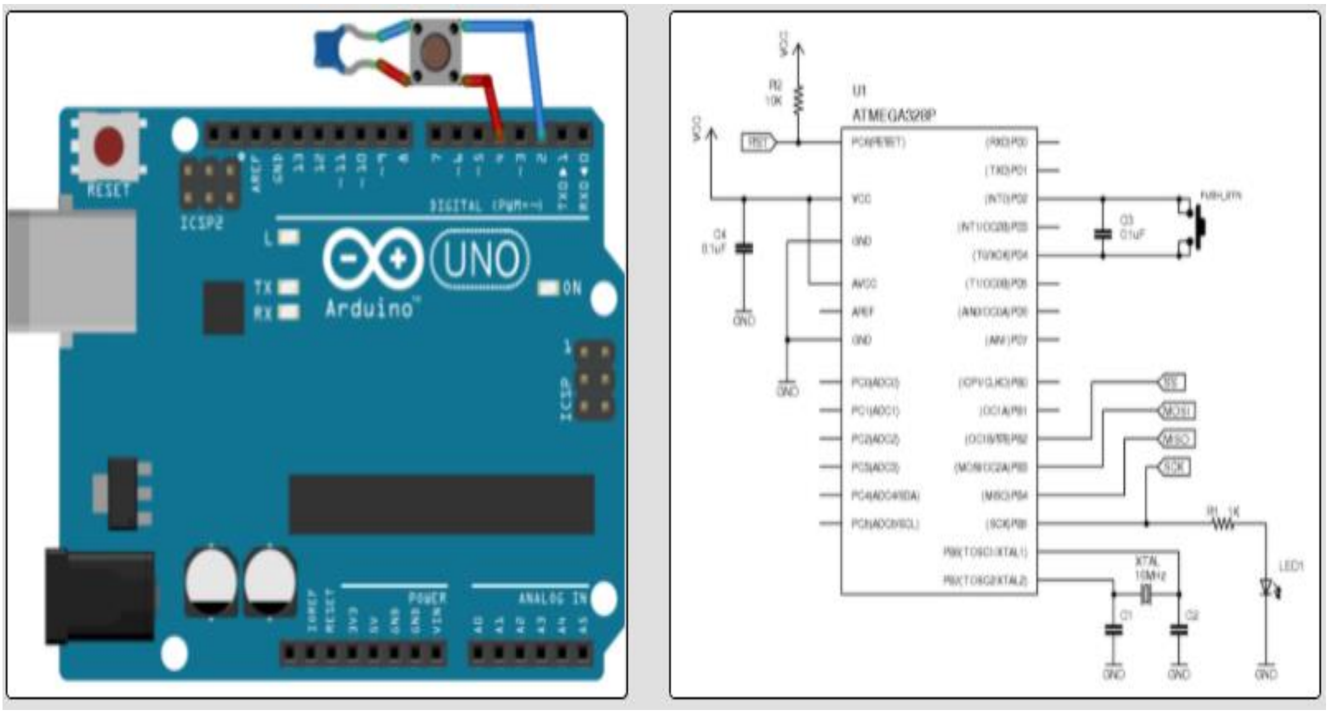


Figure 46: Hardware used to implement our example

Alternatively, the example hardware can be built on a breadboard. The schematic on the right shows the relevant Atmega328 circuit used in this example.

Programming a State Machine

Based on our definitions of a state machine, we need to programme each state as a list of instructions to execute (outputs, state transitions) for each possible input, and then have the code navigate to the correct set of instructions. One simple way to do this is to use nested switch statements as shown below.

```
/*  Example of nested switch statements     */
switch(system_state){
    case led_off:
        switch(system_input){
            case button_press:
                turn the led on;        // Output
                system_state = led_on;  // State transition
                break;
            case timeout:
                break;                  // do nothing
        }
        break;
    case led_on:
        switch(system_input){
            case button_press:
                blink the led;              // Output
                system_state = led_blink;   // State transition
                break;
            case timeout:
                turn off the led;           // Output
                system_state = led_off;     // State transition
                break;
        }
        break;
    case led_blink:
        switch(system_input){
            case button_press:
                turn off led;           // Output
                system_state = led_off; // State transition
                break;
            case timeout:
                turn off led;           // Output
                system_state = led_off; // State transition
                break;
        }
        break;
}
```

Application exercises: Simple state machine

Requirements:

Create a simple state machine that consists of four states. The states shall be called “S1”, “S2”, “S3”, and “S4”. The state S1 will lead to S2, S2 to S3, S3 to S4 and S4 back to S1 again. The transition shall only happen when a button is pressed. While in a new state the programme shall output the name of the state continuously. Ensure that the state only gets changed once per button press.

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)

```
• activeState = “S1”
• btnPin = Nr1
• btnPressed = 0
• In setup function
    o Set btnPin as input and activate the pullup resistor
•
• While true
    o If activeState == “S1”
        ▪ Print(“S1”)
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = “S2”
    o Elif activeState == “S2”
        ▪ Print(“S2”)
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = “S3”

    o Elif activeState == “S3”
        ▪ Print(“S3”)
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = “S4”
        ▪
    o Elif aciveState == “S4”
        ▪ Print(“S4”)
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = “S1”
    o Delay 200ms
```

3.2. Measuring programme

Requirements:

A real measurement programme usually does not start immediately when the microcontroller is connected to power. Instead it will be run in a state machine. With a initial state “init” where the programme waits for a user input to start. A measurement state “measure” and usually also a error state “error” in case that any errors happen. More complicated measurement programmes will have more states. Such as different types of measurements and so on.

In this task a simple measurement programme shall be implemented. The programme shall wait dormant in the init state till the user presses a button. When entering the init state the programme shall initially print “Read for measurement please press the button”. If a button is pressed then the programme shall print “Starting to measure” once. Then the programme shall read in a sensor of your choosing and print the sensor value. The programme shall make 100 measurements with a delay of 500ms between them. After all measurements are finished the programme shall print “measurement finished” and return to the init state. If the user presses the button again during the measurement then the programme shall stop the measurement state and instead go into the error state. There the programme shall output once “Error happened during measurement please reset”. If the user presses again the button it will return to the init state.

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)

```
• activeState = "S1"
• btnPin = Nr1
• btnPressed = 0
• entry = 1
• In setup function
    o Set btnPin as input and activate the pullup resistor
    o Initiate ADC
    o Set  aInPin as input
    o Cnt = 0

• While true
    o If activeState == "init"
        ▪ If entry == 1
            • Print("Read for measurement please press the
              button")
            • entry = 0
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = "measure"
            • entry = 1
    o Elif activeState == "measure"
        ▪ If entry == 1
            • Print("Starting to measure")
            • entry = 0
            • cnt = 0
        ▪ sensVal = analogeRead(aInPin)
        ▪ print(sensVal)
        ▪ cnt = cnt+1
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = "error"
        ▪ if cnt >= 99
```

```
            • aciveState = “init”
    o Elif activeState == “error”
        ▪ If entry == 1
            • Print(“Error happened during measurement please
              reset”)
            • entry = 0
        ▪ btnVal=digitalRead(btnPin)
        ▪ if btnVal ==  pressed
            • btnPressed = 1
        ▪ Elif btnPressed == 1 and btnVal == not pressed
            • bnPressed = 0
            • activeState = “init”
        ▪
    o Delay 500ms
```

4. Communication methods and data exchange

4.1. HTTP Requests

4.1.1. Basics to webbased communication

HTTP is a set of protocols designed to enable communication between clients and servers. It works as a request-response protocol between a client and server.
A web browser may be the client, and an application on a computer that hosts a website may be the server.
So, to request a response from the server, there are mainly two methods:

GET : to request data from the server.

POST : to submit data to be processed to the server.

Here is a simple diagram which explains the basic concept of GET and POST methods.

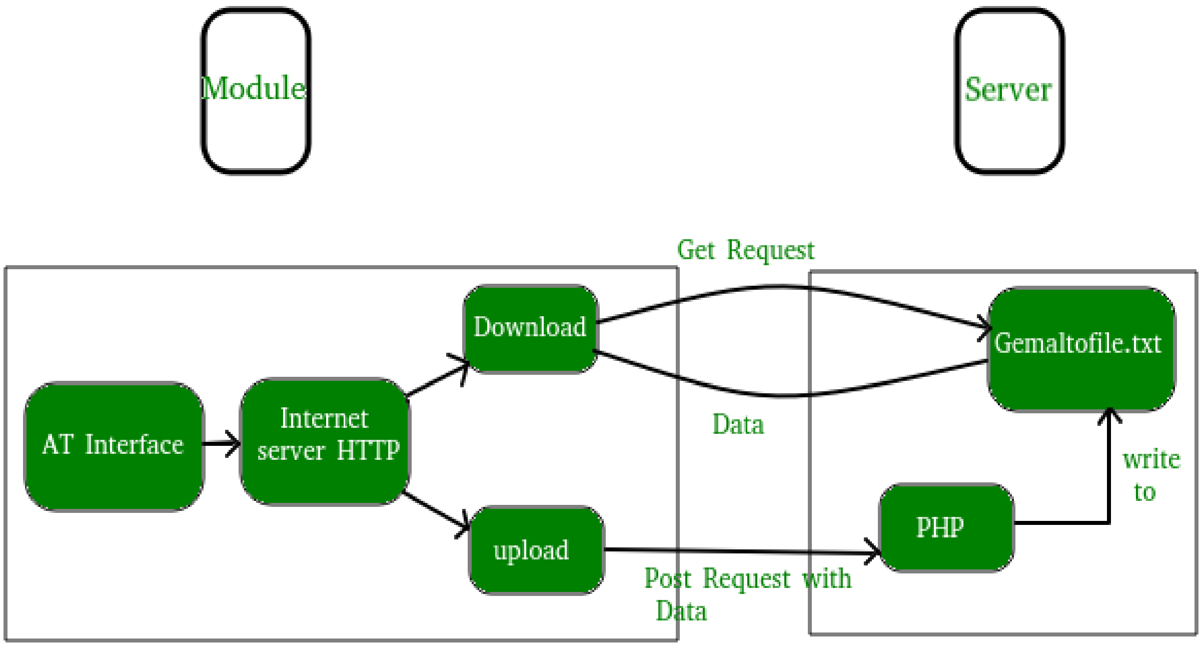


Figure 47: Webbased communication

Now, to make HTTP requests in python, we can use several HTTP libraries like:

httplib
urllib
requests

The most elegant and simplest of above listed libraries is Requests. We will be using requests library in this article. To download and install Requests library, use following command:

pip install requests

Making a Get request

```
# importing the requests library
import requests
# api-endpoint
URL = "http://maps.googleapis.com/maps/api/geocode/json"
# location given here
location = "delhi technological university"
# defining a params dict for the parameters to be sent to the API
PARAMS = {'address':location}
# sending get request and saving the response as response object
r = requests.get(url = URL, params = PARAMS)
```

```
# extracting data in json format
data = r.json()
# extracting latitude, longitude and formatted address
# of the first matching location
latitude = data['results'][0]['geometry']['location']['lat']
longitude = data['results'][0]['geometry']['location']['lng']
formatted_address = data['results'][0]['formatted_address']
 # printing the output
print("Latitude:%s\nLongitude:%s\nFormatted Address:%s"
      %(latitude, longitude,formatted_address))
```

The above example finds latitude, longitude, and formatted address of a given location by sending a GET request to the Google Maps API. An API (Application Programming Interface) enables you to access the internal features of a programme in a limited fashion. And in most cases, the data provided is in **JSON(JavaScript Object Notation)** format (which is implemented as dictionary objects in Python!).

Now, in order to retrieve the data from the response object, we need to convert the raw response content into a JSON type data structure. This is achieved by using json() method. Finally, we extract the required information by parsing down the JSON type object.
Making a POST request

```
# importing the requests library
import requests
# defining the api-endpoint
API_ENDPOINT = "http://pastebin.com/api/api_post.php"
# your API key here
API_KEY = "XXXXXXXXXXXXXXXXX"

# your source code here
source_code = '''
print("Hello, world!")
a = 1
b = 2
print(a + b)
'''
# data to be sent to api
data = {'api_dev_key':API_KEY,
        'api_option':'paste',
        'api_paste_code':source_code,
        'api_paste_format':'python'}
# sending post request and saving response as response object
r = requests.post(url = API_ENDPOINT, data = data)
# extracting response text
pastebin_url = r.text
print("The pastebin URL is:%s"%pastebin_url)
```

4.1.2. Server-side setup of a web service for recording and evaluating requests.

Creation of client sided communication programme using HTTP requests

Requirements

In this task a microcontroller shall read in values of a sensor of your choosing. The microcontroller shall read in a measurement every 500ms. The sensor data shall be sent directly towards a computer via HTTP – Request. The computer shall have a measurement software running for reading in the sensor’s values, converting them to physical values and printing them on a line graph.

Steps:

1. Create a measurement programme on a microcontroller that reads in every 500ms values from a sensor
2. Update the programme to send out the measured values via HTTP request to a pc
3. Set up a server programme on a pc to read in the values transmitted via HTTP request
4. Upgrade the pc programme to display the received values in a line diagram

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)
Further this example needs a http request library to be includes (example: import requests)
Also a http server needs to be running (example: using the flask library)
Client:

- aInPin = Nr1
- In setup function
  - Initiate ADC
  - Set aInPin as input
  - url = set URL as string
- In main loop
  - While true
    - adcVal = analogeRead(aInPin)
    - physicalVal = convertToPhysical(adcVal)
    - data = {‘SensVal’, physicalVal}
    - request.post(url, data)
    - delay 500ms

Server:

- rcvData = []
- In callback function during receiving
  - rcvData.append(request.form[‘SensVal’])
  - plot rcvData without blocking

4.2. Server Communication

4.2.1. Basics to Client Server bases communication architectures

When using computers, each of us has a need to connect to collect and share information. To meet that need, a computer network or a network system (computer network or network system) was born.

On a network, computers can take on one of the following three roles:

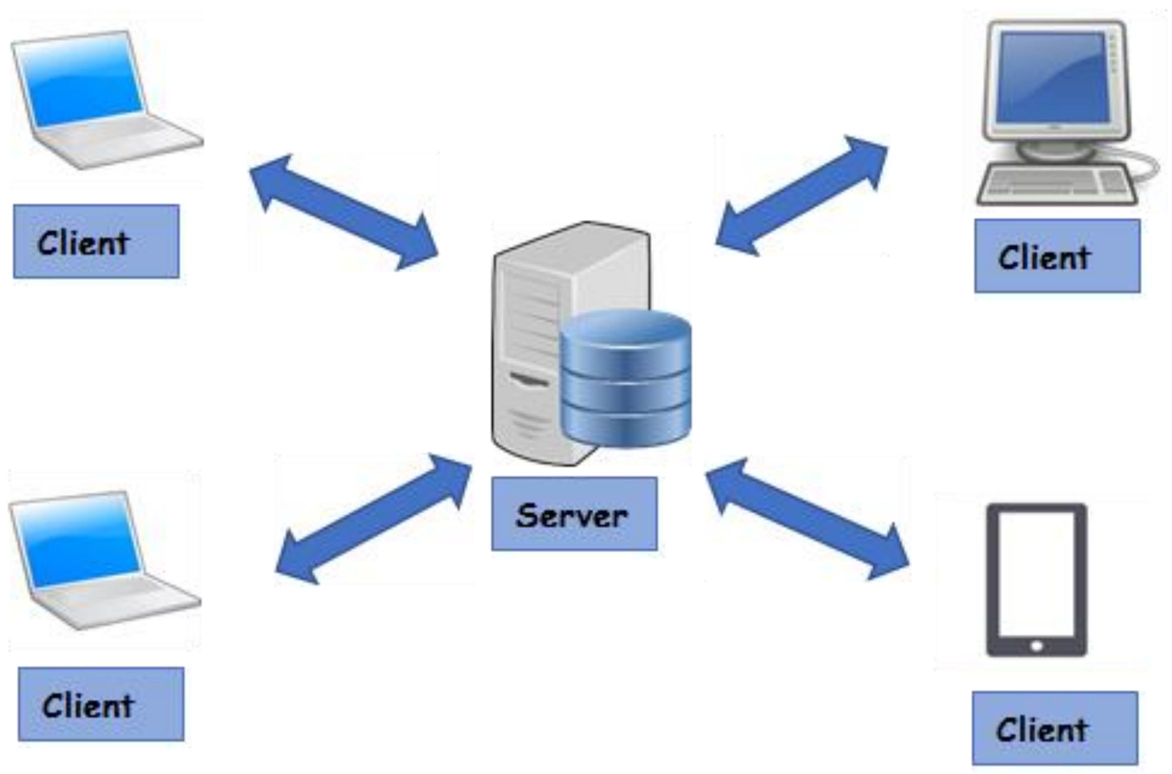


Figure 48: Client Server Communication

• Computer plays the role of server - Server: A computer capable of providing resources and services to other workstations in the network. The server plays a supporting role for the operations on the client workstation to take place more efficiently.

• Computers act as workstations - Client: As workstations, they will not provide resources to other computers, but only use resources provided from the server. A client in one model can be a server for another, depending on the user's needs.

• Computer plays the role of Peer: Both use resources from the provider server, and also provide resources to other computers in the network.

The computer roles that provide for the network are different, so of course there are many different computer network models: Client Server, Peer-to-Peer and Hybrid. In which, the client server network model is the most widely used.

What is the Client Server model?
The client server network model is a computer network model in which the child computers act as a client, they are responsible for sending requests to the server. Let the server process the request and return the result to that client.

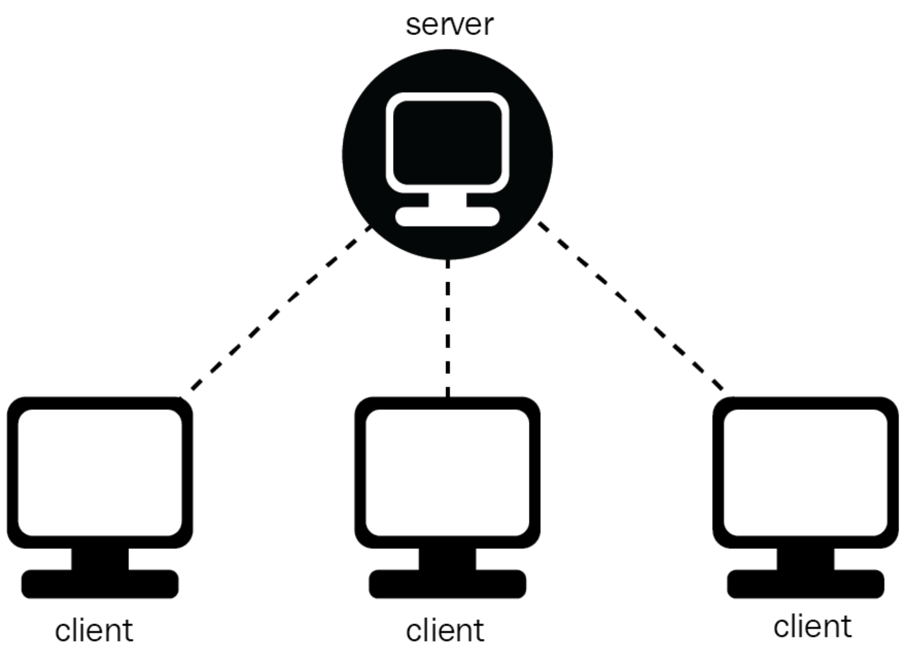


Figure 49: Client Server model

Working principle of Client Server model
In the Client Server model, the server accepts all valid requests from anywhere on the network, and then returns the results to the computer that sent the request.
Computers are considered as clients when they are responsible for sending requests to servers and waiting for a response to be sent.

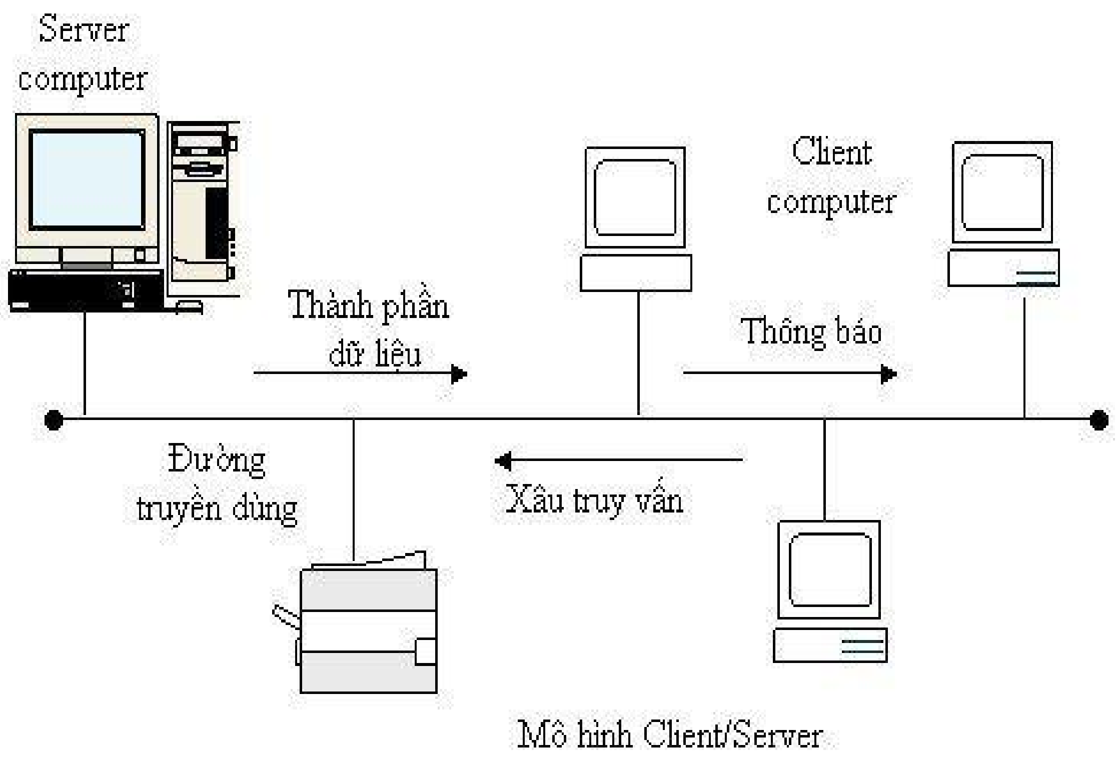


Figure 50: Working principle of Client Server model

In order for the client and the server to communicate with each other, there must be a certain standard between them, and that standard is called a protocol. Some standard protocols are widely used today such as TCP/IP, OSI, ISDN, X.25, Lan-to-Lan, .. Then, if the client wants to get information from the server, they must according to a protocol that the server offers. If the request is accepted, the server collects the information and returns the result to the requesting client. Because normally, the server is always in a state of being ready to receive requests from clients, so as long as the client sends a signal and accepts the request, the server will return the result in the shortest time possible.

Advantages and disadvantages of the client server model

Advantages

• The client server model makes it possible to work on any computer that supports communication protocols. This standard protocol also helps manufacturers integrate into many different products without any difficulty.
• There can be many server chapters doing the same service, they can be on more than one computer or one computer.
• Client server model only carries the characteristics of the software, but is not related to the hardware, in addition to the only requirement that the server must have a higher configuration than the clients.
• Client server supports users with a variety of services and convenience by remote access that the old models do not have.
• The client-server network model provides an ideal foundation, allowing the integration of modern techniques such as object-oriented design models, expert systems, and geographic information systems (GIS).

Disadvantages of client-server model:

Due to the need to exchange data between two different computers in two geographically distant areas, the issue of information security is sometimes not very secure. This is the only downside of this model.

4.2.2. Server-side creation of a communication programme for data acquisition and data backup using web sockets, Client-side creation of a measurement and communication programme, for connection establishment and data transmission

Websockets – Visualization server

Requirements:

In this task the microcontroller acts as client and sends its measurement data periodically to the pc who acts as server. The microcontroller shall send every 200ms a sensor measurement to the server via WebSocket communication.
The computer shall act as a server, for every incoming measurement the computer shall receive the measurement data and print it display it accordingly on a visualization.
Steps:

1. Prepare the client communication programme on the microcontroller
2. Create a measurement programme based on the microcontroller
3. Update the programme to send data over websockets
4. Create the server sided webserver communication
5. Create a visualization based on the received sensor data

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation) Further this example needs a websocket library to be includes (example: import socket)

Client:

```
• aInPin = Nr1
• In setup function
    o Initiate ADC
    o Set  aInPin as input
    o host = host address as string
    o port = port to be used as int
• In main loop
    o Socket.connect(host,port)
    o While true
        ▪ adcVal = analogeRead(aInPin)
        ▪ physicalVal = convertToPhysical(adcVal)
        ▪ data = convert to string physicalVal
        ▪ data = encody to bytes data
        ▪ socket.sendall(data)
        ▪ delay 200ms
```

Server:

```
• rcvData = []
• host = host address as string
• port = port to be used as int
• socket.bind(host,port)
• socket.listen()
• conn, addr = socket.accept()
• while true
    o data = conn.recv(1024)
    o if data:
        ▪ data = float(data.decode(‘utf-8))
        ▪ rcvData.append(data)
        ▪ plot rcvData without blocking
```

***Websockets** - Measurment server*

Requirements:

The requirement in this task is to set up a client server based websocket communication system. The microcontroller shall be configured as a measurement server. The PC shall operate as a client. Whenever the client sends a measurement request to the server the server shall measure a value from a sensor. This value shall be send back to the client. The client then shall save the value in a csv file. Further the client shall have a graphical user programme implemented. So that the user can request a measurement by pressing a button.

Steps:

1. Create a measurement function on the microcontroller which can read in sensor values the function shall be called "readInSensor()" and it shall return the sensor value
2. Create the server functionality on the microcontroller
   a. When a client request is received the server shall call the function "readInSensor()"
   b. The read in value shall be send via socket communication back to the client that requested the value
3. Create the client based websocket functionality
4. Create a Graphical User Interface on the client PC
   a. One button shall be used to create a client request for the server to return a sensor value
5. Update the client programme to save the received values in a .csv file

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)
Further this example needs a websocket library to be included (example: import socket)
Also, a library for creating graphical user interfaces (example: tkinter) and a library to write data to a .csv file needs to be included (pandas)

Client:

```
•  host = host address as string
•  port = port to be used as int
•  open window
•  create button object on window with callback function
•  in button callback function:
     o  socket.connect(host,port)
     o  socket.sendall(“newData”)
     o  data = s.recv(1024)
     o  data = data.decode(‘utf-8)
     o  writeToCSV(data)
```

Server:

```
•  aInPin = Nr1
•  host = host address as string
•  port = port to be used as int
•  In setup function
     o  Initiate ADC
     o  Set  aInPin as input
     o  host = host address as string
     o  port = port to be used as int
     o  socket.bind(host,port)
•  while true
     o
     o  socket.listen()
     o  conn, addr = socket.accept()
     o  data = conn.recv(1024)
     o  adcVal = analogeRead(aInPin)
     o  physicalVal = convertToPhysical(adcVal)
     o  data = convert to string physicalVal
     o  data = encody to bytes data
     o  conn.sendall(data)
```

Websockets - Control panel

Requirements:

In this task the microcontroller shall be used as server and controlled via a graphical user interface created on the pc which acts as a client. The graphical user interface shall have two buttons one to increase the brightness of a LED and one to reduce the brightness of the LED. The LED itself shall be controlled on the microcontroller via PWM. Alternatively to the LED if available a motor shield and a dc motor can be controlled. The communication shall be done via Websockets. Every time the buttons on the GUI on the computer are pressed a command to the microcontroller shall be send. The microcontroller shall receive the command and either increase or decrease the brightness of the LED

Steps:

1. Create the client based communication software
2. Create the server based communication software
3. Update the client software to dim a LED
    a. Create a function "dimLED(value)" which receives a dimming value for the LED and sets the PWM according to the value. No return value is needed
    b. Consider to ensure that no value above the max value of the PWM is inputted
4. Create a server sided GUI which with two buttons to send commands to increase or decrease the PWM value

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)

Client:

- host = host address as string
- port = port to be used as int
- dc = input()
- dc.encode()
- socket.connect(host,port)
- socket.sendall(dc)

Server:

- LEDPin = Nr1
- host = host address as string
- port = port to be used as int
- In setup function
  - Initiate ADC
  - ledPWM = set LEDPin as output
  - Initiate PWM with 0% duty cycle
  - host = host address as string
  - port = port to be used as int
  - socket.bind(host,port)
- while true
  - socket.listen()
  - conn, addr = socket.accept()
  - data = conn.recv(1024)
  - data = data.decode(“utf-8”)
  - data = float(data)
  - set ledPWM duticycle to data

4.3. Publisher subscriber commuinication

4.3.1. Publisher Subscriber Communication Architecture Basics

The publish/subscribe pattern (also known as pub/sub) provides an alternative to the traditional client-server architecture. In the client-client model, the client communicates directly with the endpoint. The pub / sub model separates the client that sends the message (publisher-publish) from the client or the client that receives the message (subscriber-subscriber). Publishers and subscribers never communicate directly. In fact, they are not even aware that the other exists. The connection between them is handled by a third party (broker-broker). The broker's job is to filter all incoming messages and deliver them correctly to subscribers. So let's dive a little deeper into some general aspects of pub/sub.

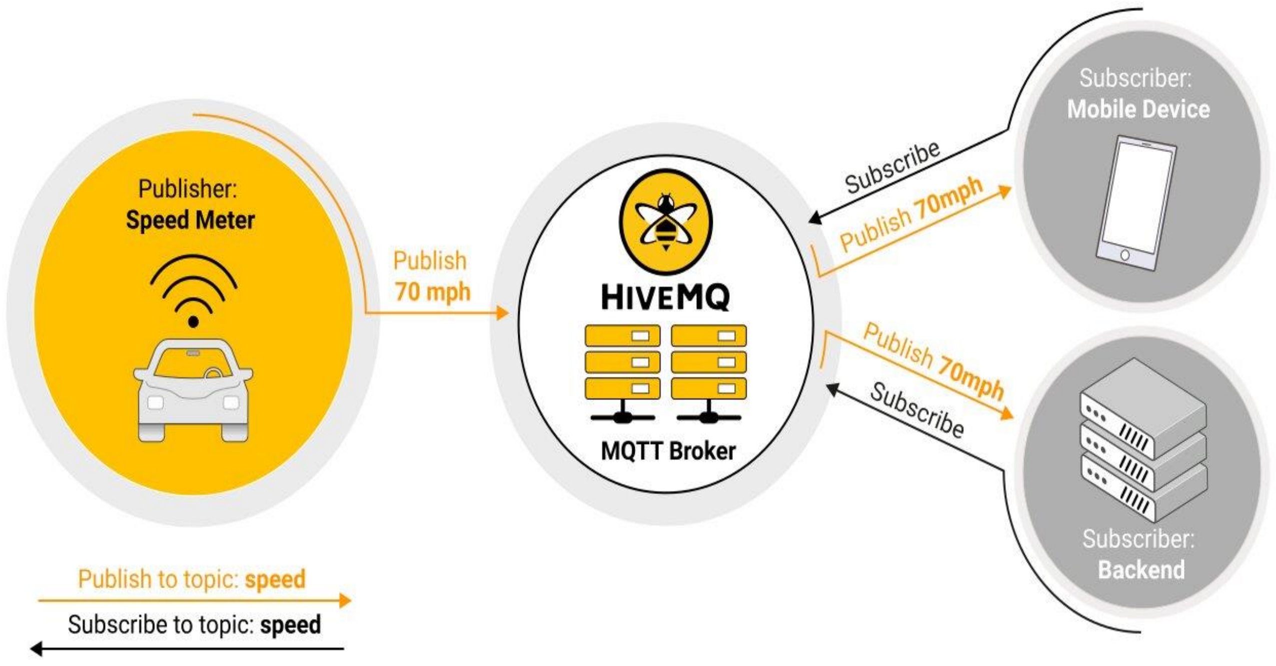


Figure 51: Publish/subscribe pattern

The most important aspect of pub/sub is the separation of the message publisher from the receiver (subscriber). This separation has several dimensions:

- Spatial separation: Publishers and subscribers do not need to know each other (e.g. do not exchange IP addresses and ports).
- Time Separation: Publishers and Subscribers don't need to run at the same time.
- Split Synchronization: Operations on both components need not be interrupted during publish or receive.

In a nutshell, the pub/sub model eliminates direct communication between message publishers and recipients/subscribers. The broker's message filtering allows to control which clients/subscribers receive which messages. Decoupling has three dimensions: space, time, and synchronization.

4.4. Implementation of a programme for communication and data exchange of two microcontrollers with provided function blocks

Simple Communication

Requirements:

In this task the MQTT communication protocol shall be used to transfer data from a publisher to a subscriber via a broker. The microcontroller shall be a publisher and read in the temperature via a temperature sensor. The publisher shall publish on a topic called “TempSens1”. The publisher shall read in temperature values every second publish them onto the named topic. The computer will be working as subscriber. The subscriber will read in the values from the topic “TempSens1” and print them in the consol. Further the computer will also host the required broker.

Steps:

1. Setup a broker on the computer and run the broker
2. Create a measurement programme on the microcontroller to read in the temperature sensor measurement and test it
3. Include the necessary libraries on the microcontroller and create the topic to be published to
4. Include the necessary libraries on the computer and create subscribe to the topic where the sensor values are published at
5. Update the programme to read the published values and print them out on the console

Reference solution

Solution in pseudo code (Pin numbers depend on hardware implementation)
Further a mqtt library needs to be installed (example phao)
Broker:
Start the broker

Publisher:

- aInPin = Nr1
- client = mqtt.Client()
- In setup function
    - Initiate ADC
    - Set aInPin as input
    - host = host address as string
    - port = port to be used as int

- In main loop
    - Connect to client with host adress and port
    - Client.loop_start()
    - While true
        - adcVal = analogeRead(aInPin)
        - physicalVal = convertToPhysical(adcVal)
        - client.publish(“/TempSens1”, physicalVal)

Subscriber:

- on_connect function:
    - client.subscribe(“/TempSens1”)
- on_message function:
    - print(str(msg.payload))
- client = mqtt.Client()
- client.on_connect = on_connect
- client.on_message = on_message
- serverAdr = server address as string
- port = port to be used as int
- connect to server with serverAdr and port
- client.loop_forever()

Illustration 5: CPU processor lighting circuits

Teamwork multiple sensor nodes

Requirements:

In this task the MQTT communication protocol shall be used to send sensor data with at least two different microcontrollers as sensor nodes. This is a group exercise with at least 2 people. Every participant shall take one node and or computer. The participants have to communicate and distribute tasks within the group. The types of used sensors and the topic names for every node can be chosen as desired but have to be different for every node. The subscriber again shall be a computer. The subscriber shall subscribe to both topics, read in the values of both topics and print the received values and the topic name on the console.

Steps:

1. Decide which participant takes which node/computer
2. Decide which sensors shall be used and which names the topics shall get
3. Implement the logic of every node and of the subscriber
    a. Reading in sensor values
    b. Publishing to topic
    c. Subscribing to topic
    d. Printing on console

Solution in pseudo code (Pin numbers depend on hardware implementation)
This example can be solved exactly as Simple communication only with adding several publishers and topics

Project Microcontroller : Data acquisition in I4.0 Setup

1. Introduction

All I4.0 applications have one base product which they rely on, which his data. Production data is already most valuable to companies for optimization and quality measures. Yet company's around the world still have classical production plants without any or only proprietaries data acquisition possibilities. One of the big challenges in future will be to identify and measure production and environmental data and store it. For companies to produce their new product "data" and to lift their productions into the realm of I4.0 it is mandatory to create awareness and skill in measuring important physical values and sending them via networks to storage stations. In this project a full-scale data acquisition scenario will be implemented.

Requirements:

- Experience in object-oriented programming
- UML
    - Use case diagram
    - Class diagram
    - Sequence diagram
- Flow chart
- Software testing
- Finished theoretical and practical part of course LC2

1.1. Additional information – for teachers

If other process stations are used then described below adapt the description to your existing system. This project was written in a general hardware independent way in terms or used sensors. Some of the exercises will provide tasks to identify sensor parameters from the datasheet. These need to be adjusted to the used hardware. Further only use sensors where you can also provide a datasheet.

2. Basic setup

There are two classical production stations. Both production stations are functional and operate without modern ways of data acquisition. Today production data is of great value for every company and delivers the potential for optimization. It will be your task to upgrade the classical production stations into a I4.0 set up. Since selecting the right sensors, creating measurement programmes and sending the data via computer networks to server stations is the basis of I4.0 this project main focus will be on setting up a functional base framework. In this project the entire workflow from system and requirement identification up to practical implementation and testing will be done. The Basic setup will be the extend a existing processing station by measuring temperature and acceleration data. In the following sections the process stations will be explained.

2.1. CPSi 4.0

The process station CPSi is displayed in below. The process station can be controlled via microcontroller and system buttons. In this example only the control with system buttons shall be considered. When a system button is pressed the processstation starts by ejecting a cube from the cube magazine. The cube is tested in several stations. After the final station the cube is either ejected with a pneumatic cylinder or passed through.

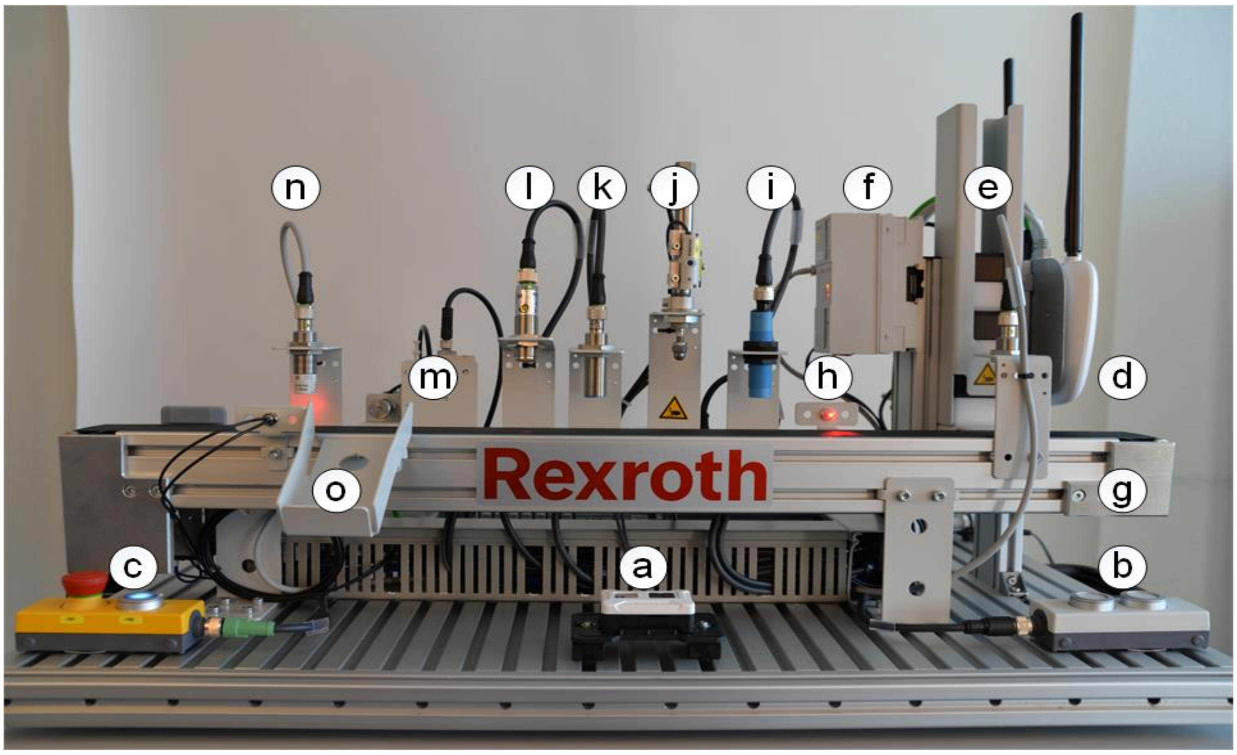


Figure 52: CPS i4.0 training system (front view)

a	Microcontroller (XDK)	i	Capacitive sensor
b	System buttons	j	Position sensor (on contour cylinder)
c	Emergency stop	k	Inductive sensor
d	WLAN router	l	Light barrier
e	Cube magazine	m	Pneu. cylinder and 5/2 directional control valve
f	XM22 (PLC)	n	RFID antenna
g	Conveyor belt	o	Collecting ramp
h	Light conductor		

2.1.1. System description

Before the test, the cubes are stored in a magazine (fig. 8 - 0). The magazine is manually refilled with new cubes. The buttons can be used to select a cube type and start an order. At the height of the second cube in the magazine, there is a mechanical switch. The switch triggers when no cube is pressed against it (NC - normally closed). Provided that:

- there are at least 2 cubes in magazine,
- emergency stop released,
- light conductor unobstructed,
- conveyor belt at a standstill,
- all cylinders in basic position,

... the valve operates the magazine.

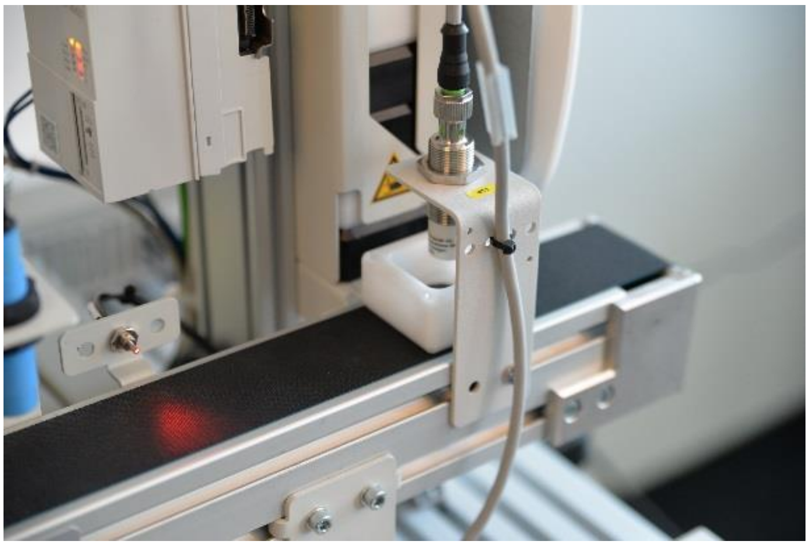


Figure 53: Cubes at start under RFID antenna

The valve is located on the back of the magazine together with a pneumatic cylinder. The cylinders are operated using compressed air with at least 1 bar. The compressed air is distributed to the valves by a compressed air distribution system. Depending on the valve position, the valves feed the compressed air into the cylinder chambers. The compressed air supply to the cylinder chambers can be regulated via throttles at the inlet points. When the piston is extended, it pushes one cube half onto the conveyor belt. The position of the cylinders is detected via reed contacts. The switches trigger with magnetic materials. There are two reed contacts on the magazine cylinder which detect the basic position and the working position of the piston.

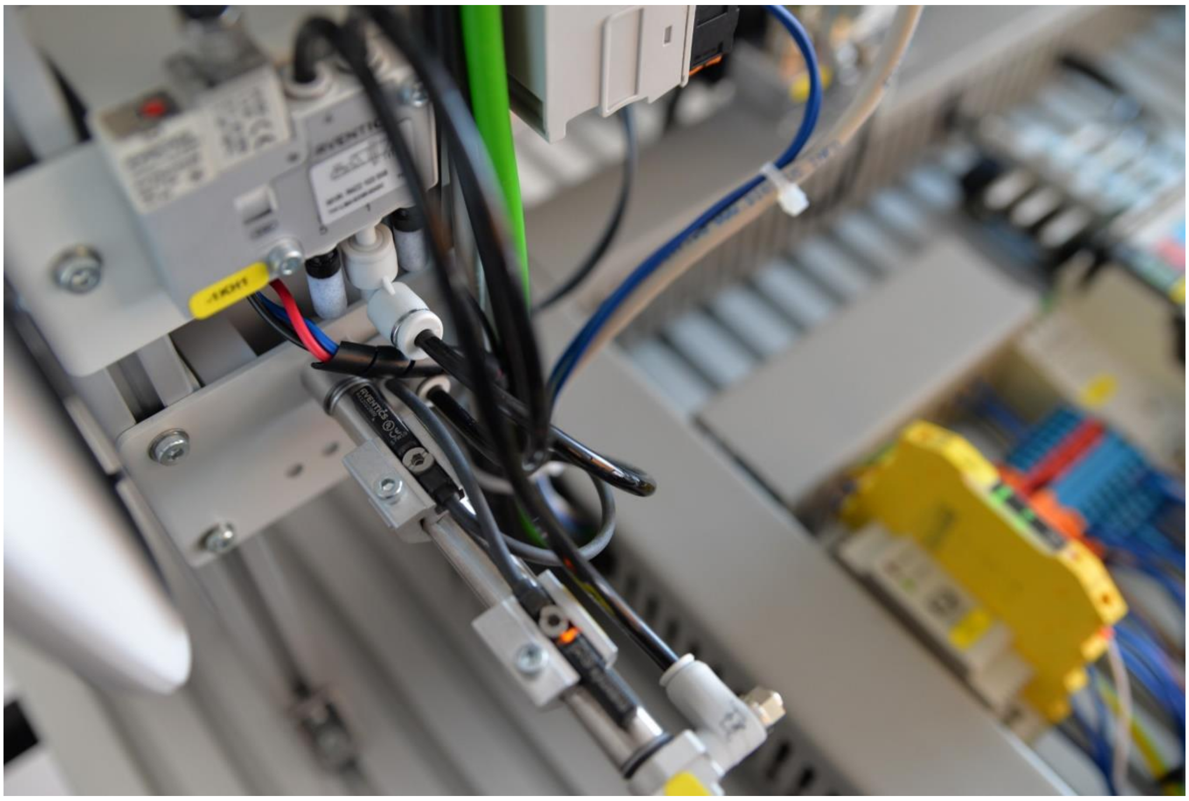


Figure 54: Magazine cylinder with two reed contacts

A DC motor is flange-mounted on the other end of the conveyor belt. The motor drives the conveyor belt. The direction of rotation is determined by two relays. Before the conveyor belt starts, the magazine cylinder returns to the basic position and the RFID antenna above the cube half writes a code number for the cube type on the RFID tag in the cube.

The cubes move in the process direction from right to left. They pass several stations on the conveyor belt. Depending on the process at the station, the conveyor belt is stopped or the workpiece is evaluated as it passes through.

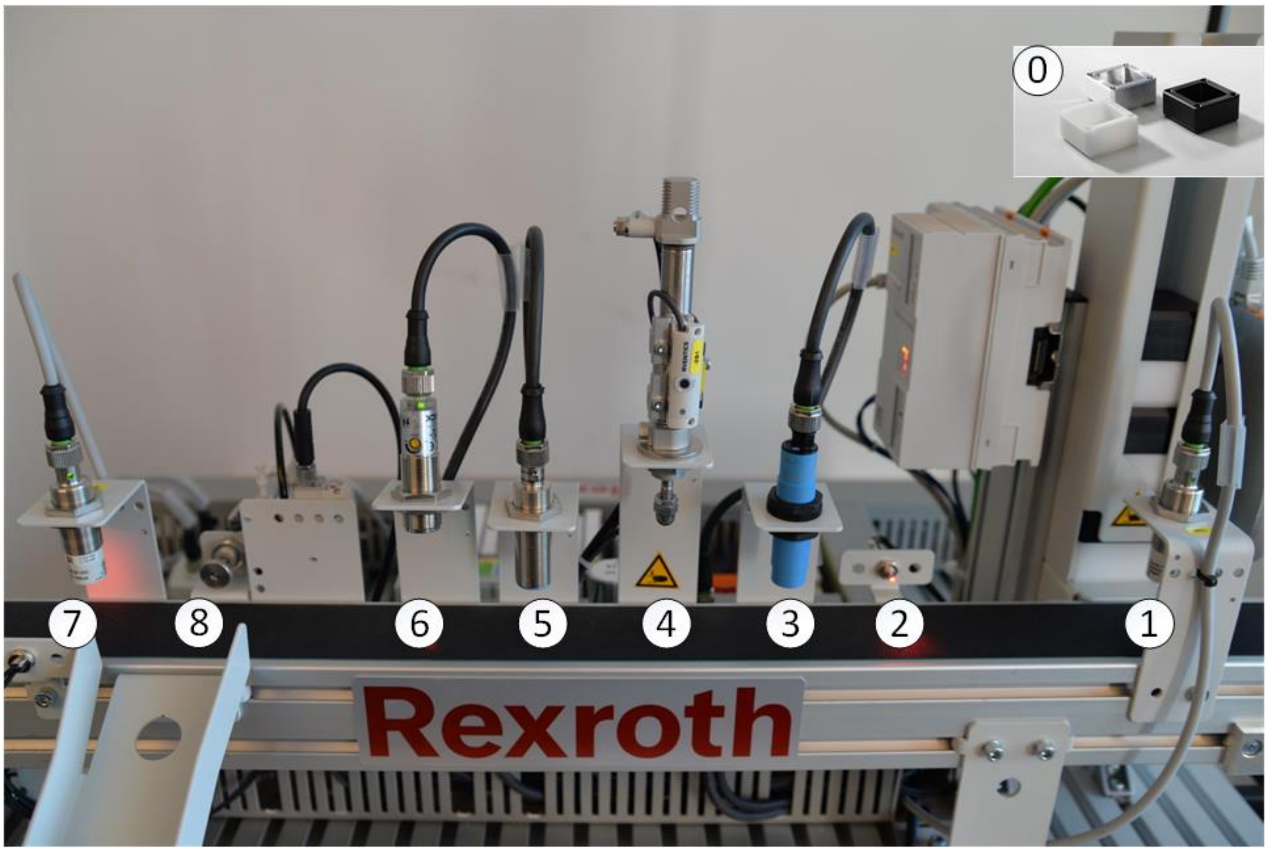


Figure 55: CPSi4.0 workpiece positions Stations

1. **RFID antenna:** Code number is written on the tag in the cube.
2. **Light conductor:** Detects cube position (lateral)
3. **Capacitive sensor:** Detects cube position (top)
4. **Contour cylinder:** Piston moves on workpiece and the position sensor measures the distance covered - inference to orientation (base / cover)
5. **Inductive sensor:** Detects metallic material - inference to material
6. **Light barrier:** Detects bright objects - inference to colour
7. **RFID antenna:** Code number is read from tag
8. **Ejection cylinder:** Eject position for wrong cubes

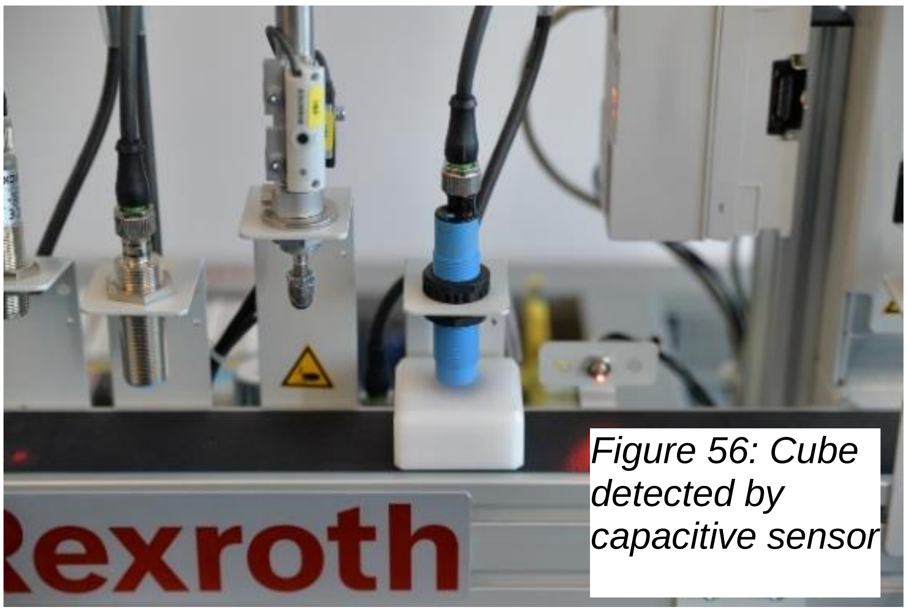


Figure 56: Cube detected by capacitive sensor

The workpiece first passes a light conductor and then a capacitive sensor. The workpiece is detected laterally or from above. Through the feedback, the programme knows when and where a cube is on the conveyor belt. After the sensors have been triggered, the conveyor belt stops so that the workpiece is positioned under the contour cylinder. The redundant detection ensures that the cube half is positioned exactly.

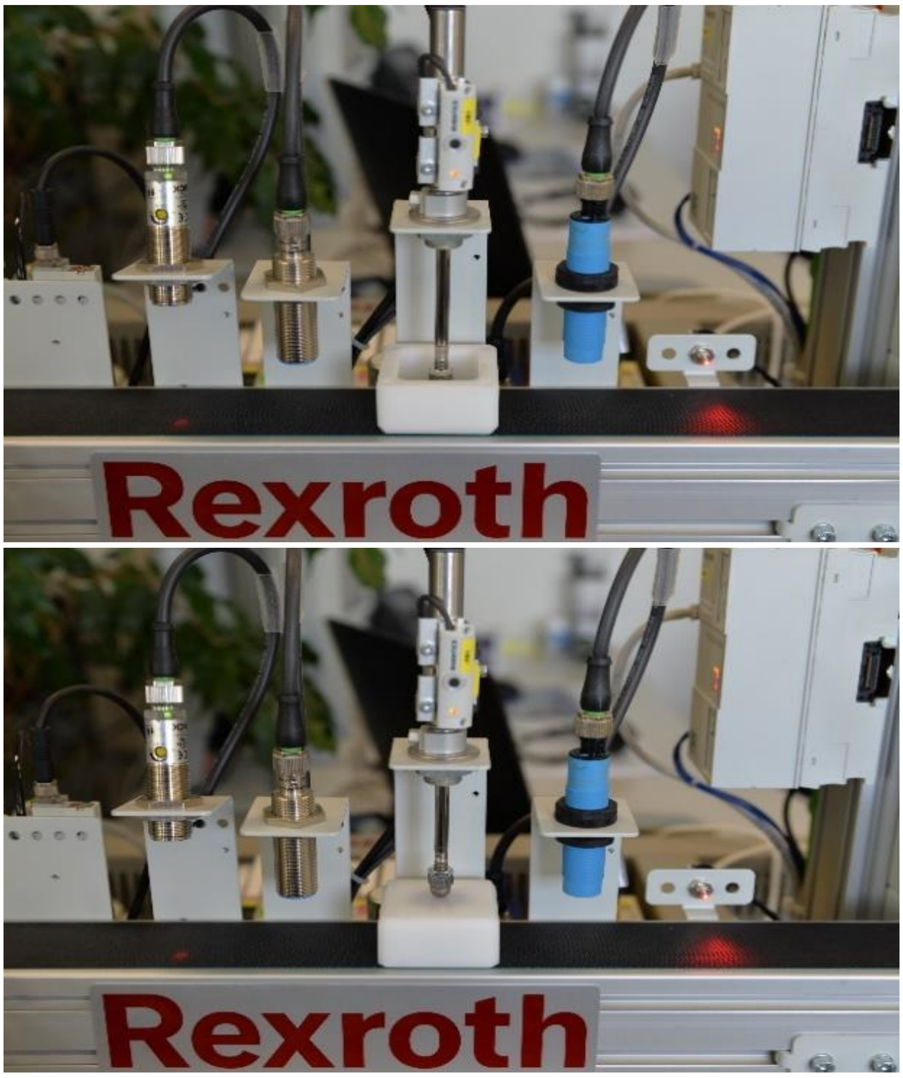


Figure 57: Contour check cylinder with position sensor for base part (above) and cover part (below)

If the conveyor belt stops here, the cylinder piston extends. This cylinder is controlled by a pneumatic valve for closed-loop control (fig. 2 - h). With a base part, the cylinder extends completely. A reed contact will detect this position. With a cover part, the piston presses on the cube. This process is limited in time. After the time has elapsed, the piston retracts. The distance covered is recorded by a position sensor. This sensor is also mounted on the cylinder housing. The workpiece orientation is deduced from the return value.

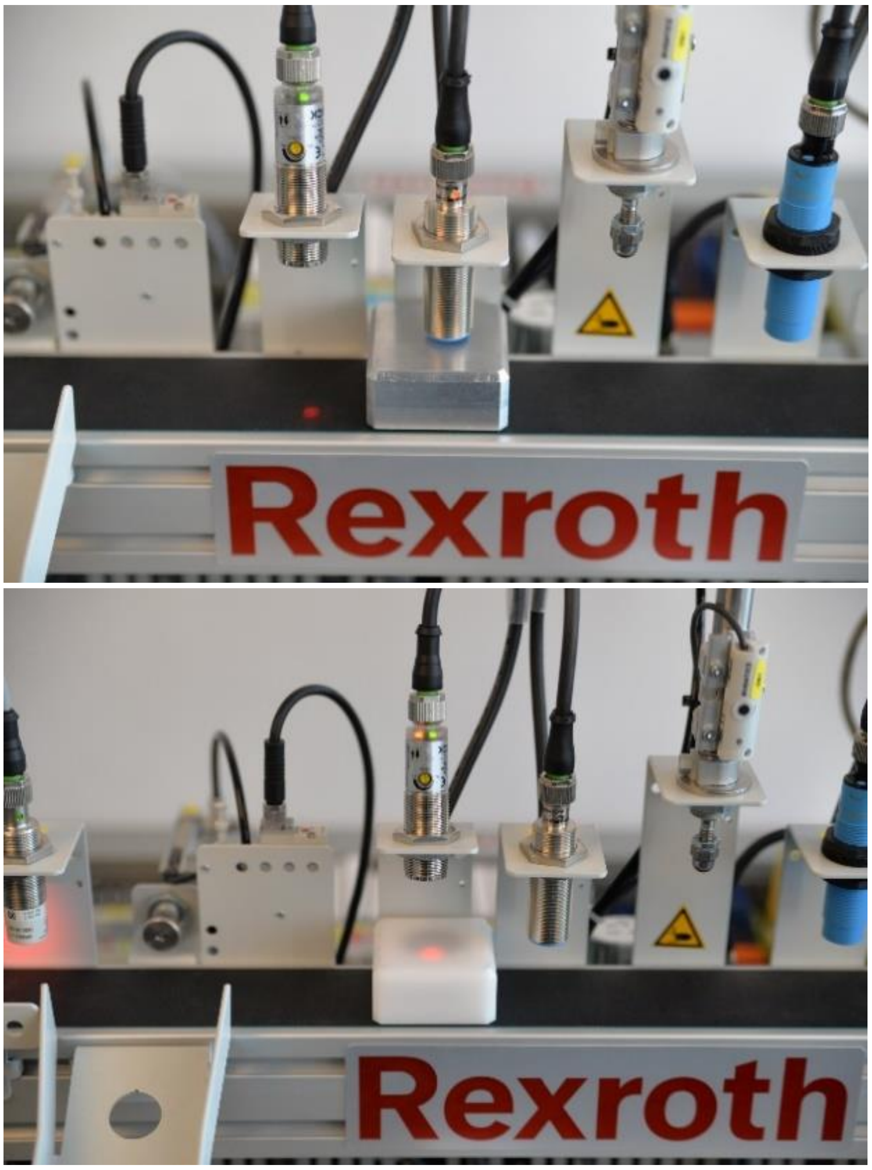
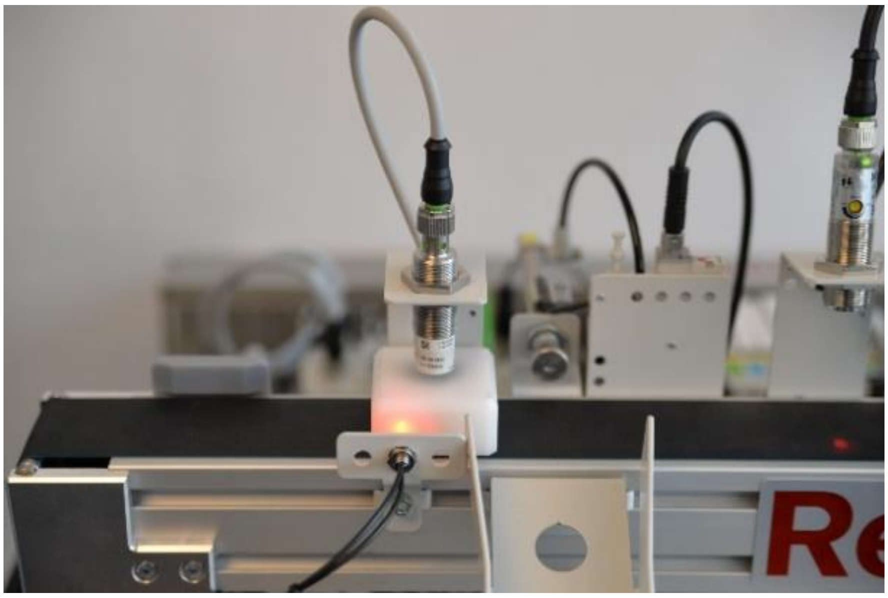


Figure 158: Inductive sensor with aluminum cube (above) and light barrier with plastic cube (below)

If the piston is in the basic position, the conveyor belt continues to move. The workpiece passes an inductive sensor and a light barrier. The inductive sensor only responds to metallic objects. The light barrier detects bright objects (white or silver). An illuminated LED at the top of the sensors indicates that the sensors have detected something. The conveyor belt does not have to stop here. The information of orientation, material and colour are saved.



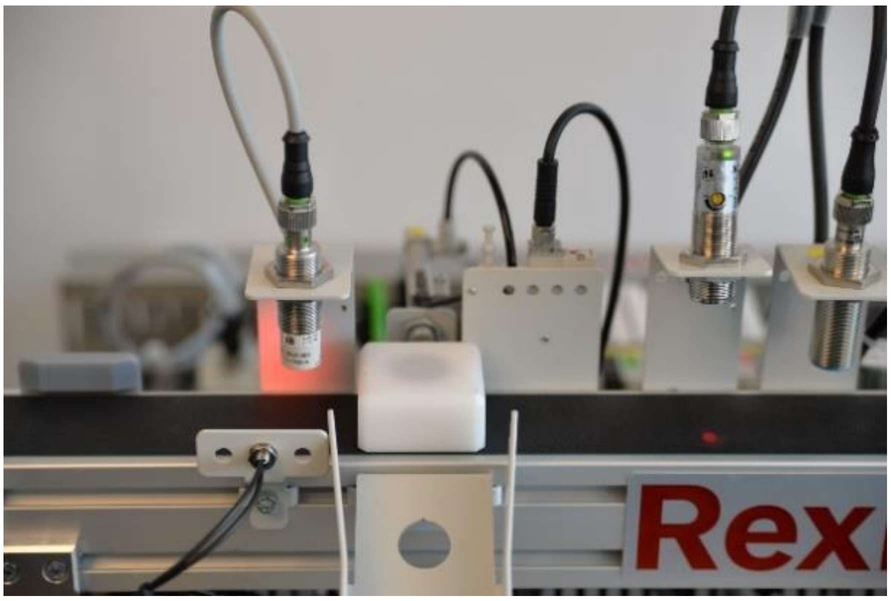


Figure 159: Light conductor stops cube under RFID antenna (above) and wrong cube in ejection position (below)

The workpiece stops at the second light conductor. This is positioned opposite the second RFID antenna. The code number on the tag (cube type) is compared with the saved properties. If there is a match, the cube half moves to the gray mark (end of conveyor belt). In the event of an error, the conveyor belt reverses to the ejection cylinder and is pushed onto the ejection ramp. If there are enough cubes in the magazine, the process is automatically repeated.

2.2. Bottling Station

The bottling station is a purely mechanical station. Sofar there are no means of data acquisition present. The bottling station takes bottles, cleans them in the cleaning station, fills them in the filling station and finally puts the cap on the bottle in the capping station.

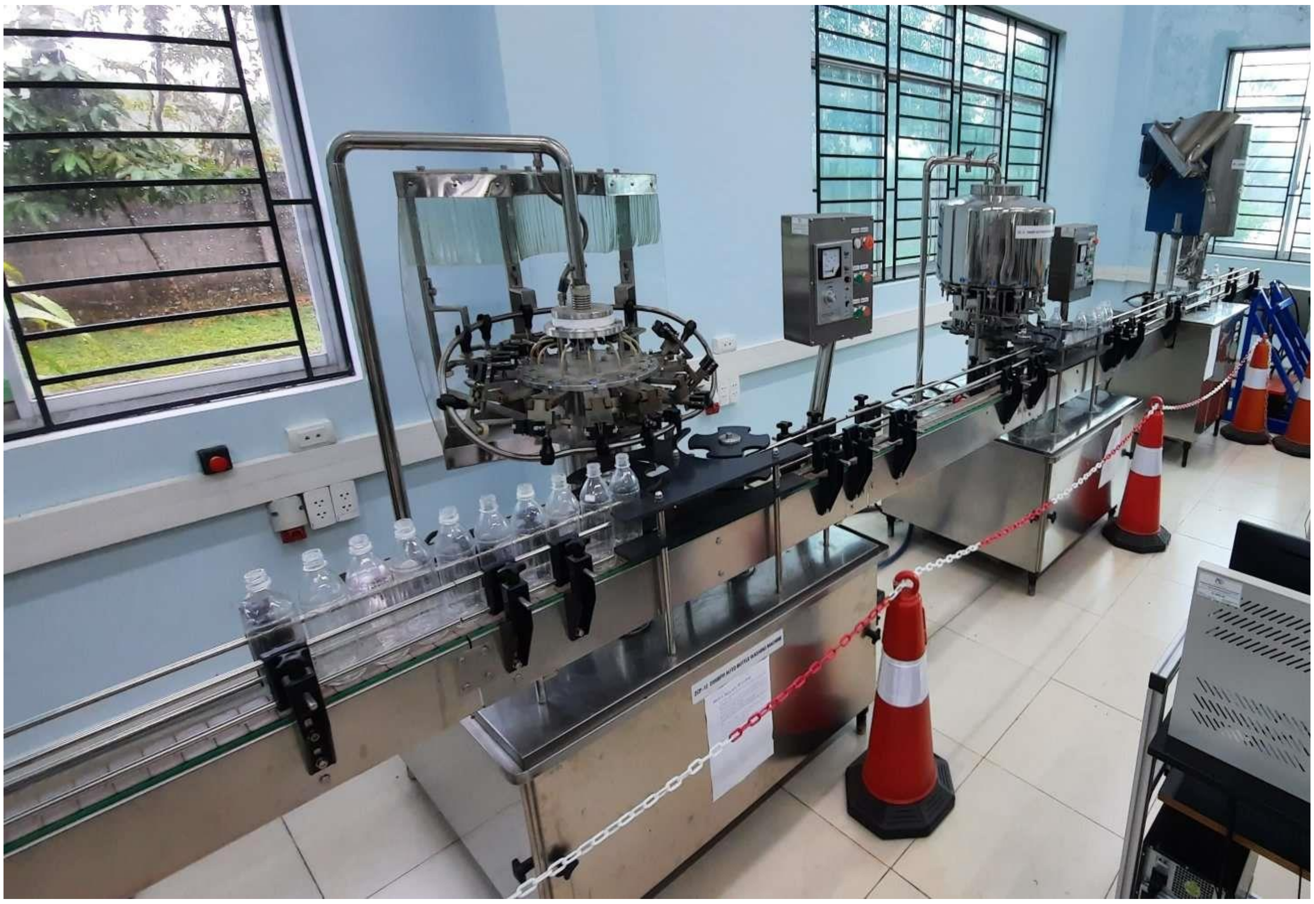


Figure 160: Bottling station full setup, cleaning(left), filling (middle) and capping (right)

After the bottling process has finished new bottles need to be put at the input stream to restart the process.

The cleaning station is the first process entered by a bottle in the entire bottling station. The bottles in the stations instream position will be mechanically gripped on their head, moved upwards and cleaned. The entire cleaning process will be conducted within one rotation of the cleaning station. After cleaning has finished the bottle is released in the out-stream position. There is no check for correct placement of bottles on the instream position, nor at the out-stream position. Therefore, it is not possible to detect bottles which got flipped over. Which in term means all bottles have to be placed correctly at the conveyor belt before the process is started.

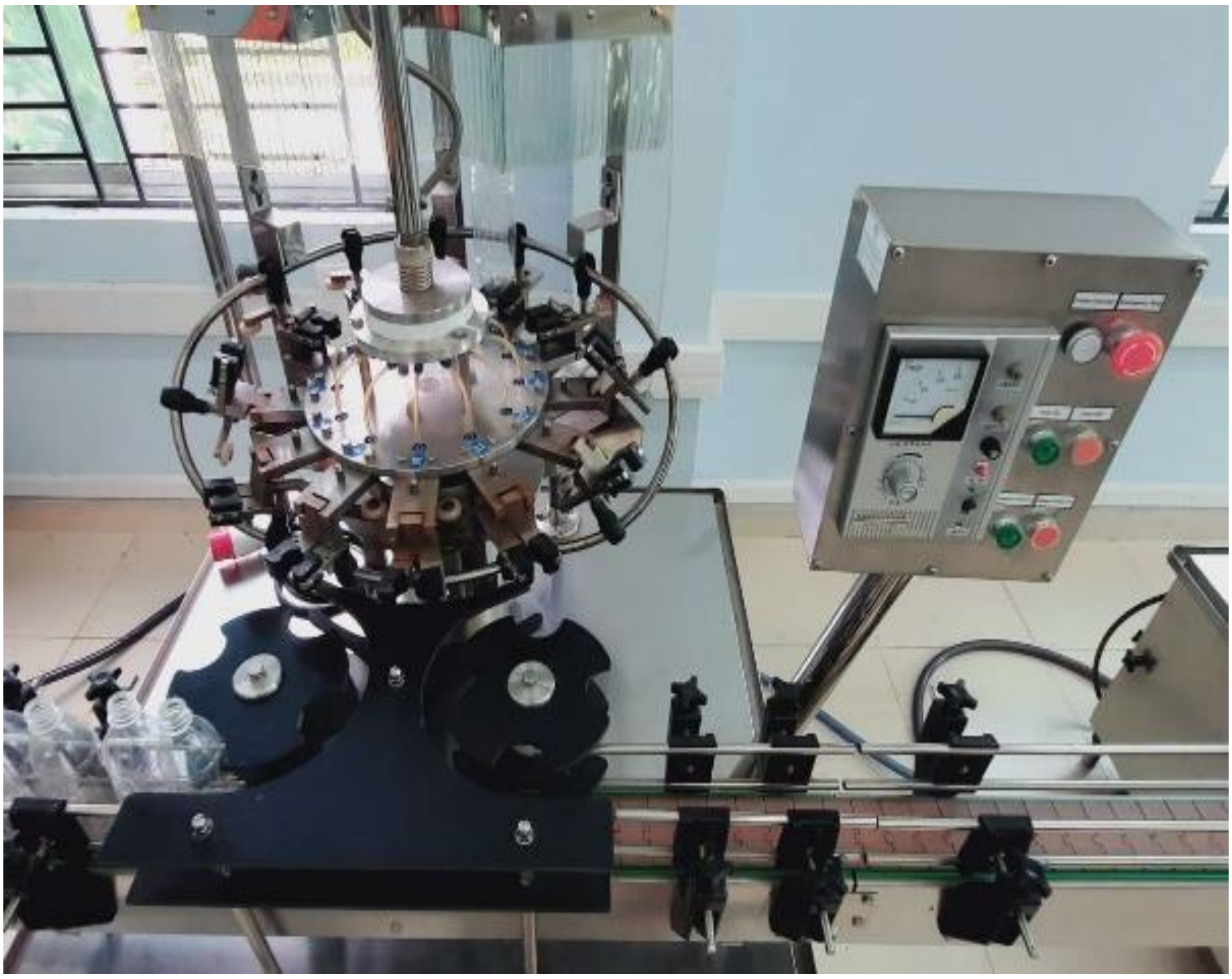


Figure 161: Cleaning station

The filling station mechanically takes in a bottle and mounts it on a predefined position. If a bottle gets mounted the filling valve will be opened and liquid will be filled inside the bottle. The filling process will be automatically initiated if a bottle is at the input position and the rotation process is initiated. After the rotation process has finished and the bottle is at the stations out stream position the valve will be automatically sealed again. On the in stream position its assumed that the bottles already enter sequentially and are positioned correctly. There is no testing or ejection system if a bottle has fallen over.

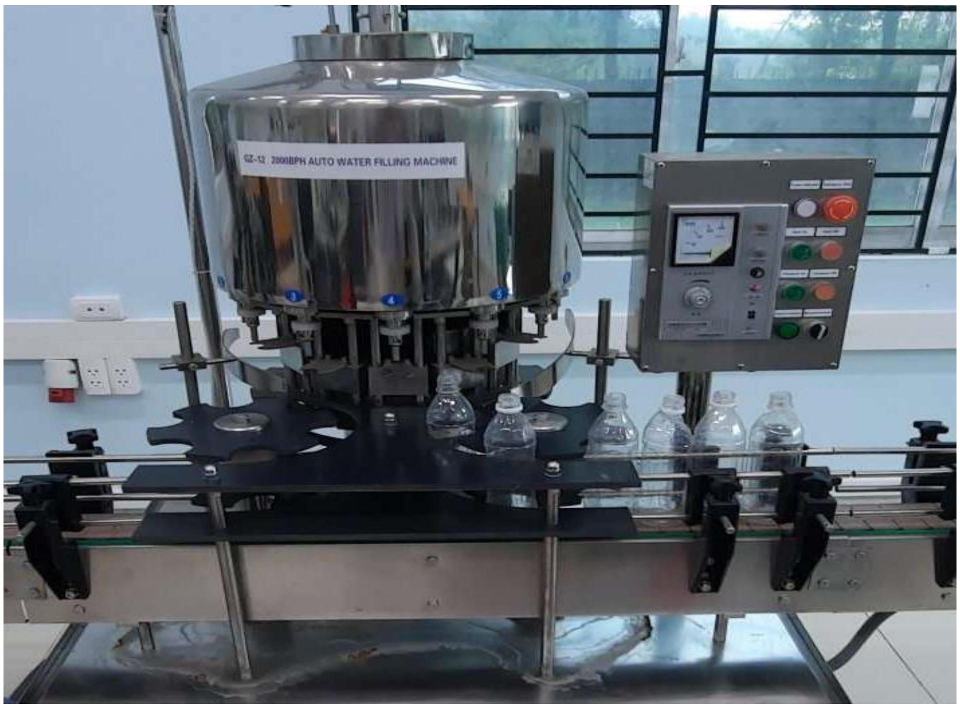


Figure 162: Filling station

After the bottles have been cleaned and filled with water they will be passed to the final station, the capping station. Here the bottles again will be picked sequentially. In one process they will be rotated and a cap will be mounted at the bottles head. Afterwards the bottles will be passed to the stations down stream position. There is no checking if the cap was placed correctly on the bottle head. Neither will here be any checks if the bottles arrive in correct position in the in-stream position. Also, after the bottles have been capped there is no check if the bottles are positioned correctly at the out-stream position. Further the caps need to be manually filled in the caps storage

All three stations deliver possilbitys to mount sensors and measure ambient environmental values.

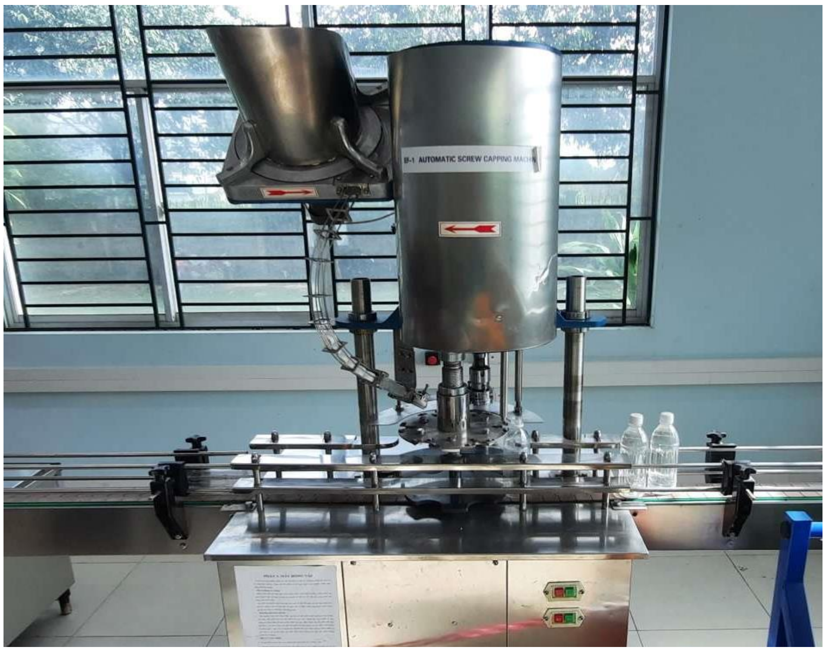


Figure 163: Capping station

3. Hardware selection – general microcontroller

To correctly measure and analyse physical values sensors are required. The existing processing station shall be upgraded by two types of sensors. One is a temperature sensor which has a thermistor. The other one is a accelerometer, the other one is a temperature sensor. The temperature sensor shall track the ambient room temperature. These sensors are often used on different locations in the production to track the temperature influence on processes. The accelerometer is used to track the acceleration of components. Further it can also be used to track vibrations and even be used to detect certain faults such as unbalance.

3.1. Identify sensor parameters

In a first step identify the provided temperature sensor and accelerometer in the datasheet and write its type down:

Temperature sensor	Accelerometer

Thermistor type sensors have several important parameters look up in the datasheet and fill out the missing parameters in the table. Data sheets usually contain several different categories of the same sensor. Make sure to only write the parameters of your sensor:

Parameter	Value
R25	
R/R25	
α	

Search the parameters in the datasheet of the accelerometer and list them in the table below:

Parameter	Value
highest scale range	
smallest scale factor	
Biggest scale factor	

3.2. Calculate sensor physical value

The provided thermistor is a NTC type. Therefore, the thermistors resistance will decrease with falling temperature. Linear interpolation is one method to calculate the thermistors temperature based on the measured resistance. To create a linear interpolation the np.interp function can be used. The function will be explained below.

3.2.1. Visualize curve

In a first step create a programme to visualize the resistance curve based on the datasheet in a range of 0°C to 50°C. Since we want to calculate the temperature based on the resistance make sure to use the temperature on the y – axis and the resistance on the x axis. In the table below a sample plot with different values is provided. Insert a picture of your plot in the column "Your solution". Make sure that your own solution has the same colours, labels and title as the sample plot. Further upload the entire picture and code on the provided task.

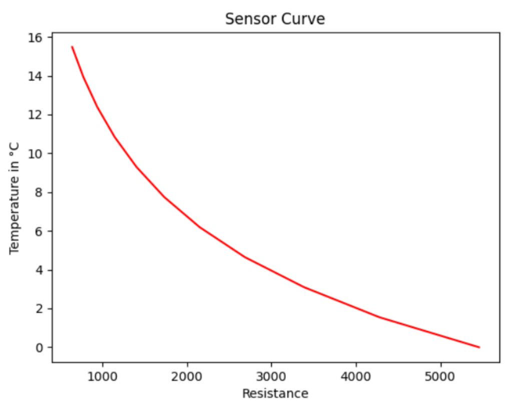
Sample plot	Your solution
	

Figure 164: Visualise curve

3.2.2. Linear interpolate sensor value

A linear interpolation function makes a linear interpolation between provided points (it fits a line between the points). This technique is very commonly used to calculate sensor values without the usage of equations. The np.inpterp function provides a linear interpolation.

Create a programme to interpolate the thermistors temperature values based on given temperature values. The programme shall be tested by creating test data. Use the function np.linespace(start, stop, nrSteps) to create test values for the resistance. Let start be 3000, stop be 3000 and 50 steps. Plot the resulting graph in the table below. Insert a picture of your plot in the column "Your solution". Make sure that your own solution has the same colours, labels and title as the sample plot. Further upload the entire picture and code on the provided task.

Description:

x=np.linspace(start,stop,nrSteps)

- Start… first value to be created
- Stop… last value to be created
- nrSteps amount of steps between first and last value

y=np.interp(x,Xp,Yp)

- x … x value where interpolation shall be calculated
- Xp … array of provided points in x direction from data sheet
- Yp … array of provided points in y direction from data sheet

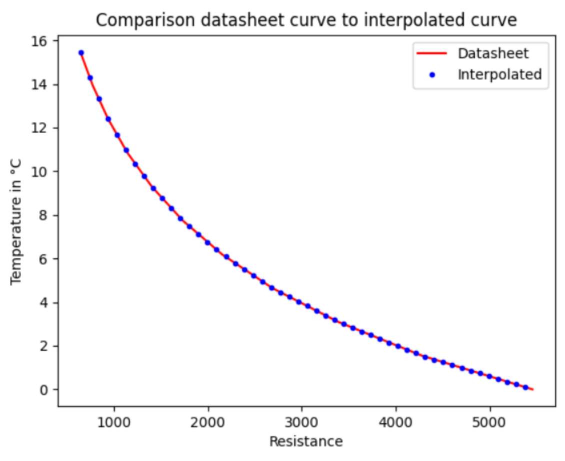
Sample plot	Your solution
	

Figure 165: Linear interpolate

4. Measurement programme – basic microcontroller

After selecting the hardware, a simple measurement programme shall be implemented. The measurement programme shall read in the temperature sensors values and the acceleration sensors values and print them directly on the console. Every measurement shall be taken with a interval of 2 seconds. Enter a screenshot of the console output in the table below. Upload your code and a screenshot of the console output.

The required output shall be:
“Temperature sensor: XXX°C”
“AccelerometerX: XXXg”
“AccelerometerY: YYYg”
“AccelerometerZ: ZZZg”

Console output:

5. Measurement programme – Bosh XDK

The bosh XDK is a robust microcontroller platform with a wide sensor array. It is commonly used in industrial applications especially for data tracking. The bosh XDK delivers a wide sensor array, a integrated wifi module, a sd card reader to store data on a memory card, freely configurable buttons and a usb connector to connect the XDK to a PC for flashing new programmes and directly outputting values in the console of its own IDE.

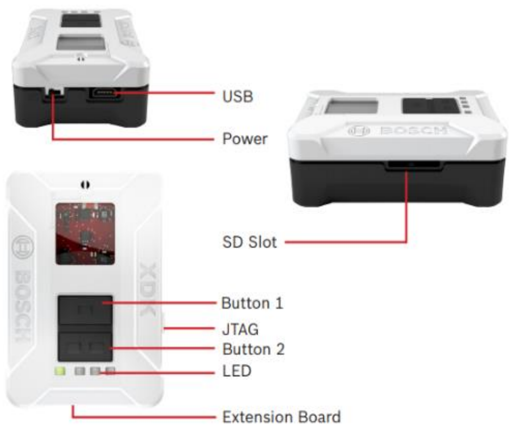


Figure 166: XDK sensor structure

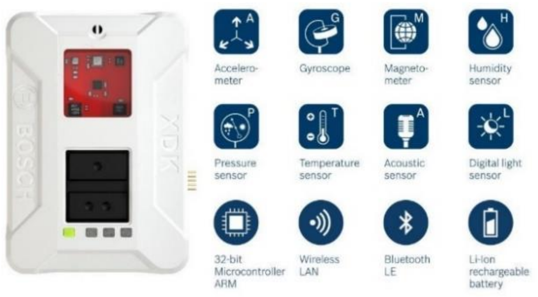


Figure 167: XDK sensor function

The bosh XDK further permits its user to access many types of sensors. Users can measure acceleration, temperature, humidity, air pressure and many more. Further the XDK uses “FreeRTOS” as realtime operating system. This ensures that precicly timed measurements with defined intervals are possible.

Further bosh uses a own eclipse based IDE called XDK-Workbench. This IDE gives the user the choice to either programme the XDK directly with C, or to use MITA programming language. MITA programming language was developed to lessen the burden of creating I4.0 applications without knowledge of embedded programming. The website: **https://developer.bosch.com/web/xdk/getting-started#1** is provided by bosh and has sample code for reading in sensor values. Part of this task is to read through the descriptions and sample codes and reuse them to create a working programme.

The task of this section is to create a measurement programme that reads in the sensor values of the:

- Accelerometer
- Humidity sensor
- Temperature sensor
- Pressure sensor

The read in values shall be outputted on the console of the XDK – Workbench. The project shall be programmed in MITA programming language. Measurements shall be done every 3 seconds. The desired console output shall be as follows:

```
Acceleration in X: XXXg
Acceleration in Y: YYYg
Acceleration in Z: ZZZg
Humidity: XXX%
Temperature: XXX°C
Air pressure: XXXkPa
```

Enter a screenshot of the console output in the table below. Upload your code and a screenshot of the console output.

Console output:

6. Sensor placement and wiring

After the test programmes were successfully completed the sensors can be placed a desirable location. For the CSPi4.0 station the ambient temperature and the vibrations of the ejection cylinder shall be measured. Same goes for the bottling station. Find a valid location for the sensor placement and document your decision with a picture. Further describe in your own words why you choose this location in the table below:

Picture	Documentation

7. Communication protocol

After the sensors and microcontroller got a valid location for measurement the communication protocol shall be implemented. After implementing one of the solutions take a picture of the console output of the pc and print it in the field below. Further upload the console output and all code.

7.1. Common microcontroller

The before described sensors of the microcontroller shall be used to create a measurement programme. As communication protocol the publisher and subscriber method using the MQTT protocol shall be used.

Steps:

1. Setup a broker on the computer and run the broker
2. Use your measurement programme for the microcontroller to read in the sensor values. Let the microcontroller read in with a interval of 1.5 seconds
3. Include the necessary libraries on the microcontroller and create the topic to be published to.
    a. Topic name temperature sensor
        i. Measurement/Temp
    b. Topic name accelerometer
        i. Measurement/AccelX
        ii. Measurement/AccelY
        iii. Measurement/AccelZ
4. Include the necessary libraries on the computer and subscribe to the topic where the sensor values are published at
5. Print the published values on the computers console

Console output

7.2. XDK

The XDK offers the possibility to transmit data via publisher and subscriber method using the MQTT protocol. Since the XDK offers a great variety of sensors multiple sensors shall be transmitted to the computer.

Steps:

1. Setup a broker on the computer and run the broker
2. Use your measurement programme for the microcontroller to read in the sensor values. Let the microcontroller read in with a interval of 3 seconds
3. Include the necessary libraries on the microcontroller and create the topic to be published to.
    a. Topic name temperature sensor
        i. MeasurementXDK/Temp
    b. Topic name accelerometer
        i. MeasurementXDK/AccelX
        ii. MeasurementXDK/AccelY
        iii. MeasurementXDK/AccelZ
    c. Topic name humidity
        i. MeasurementXDK/Humidity
    d. Topic name pressure
        i. MeasurementXDK/Pressure
4. Include the necessary libraries on the computer and subscribe to the topic where the sensor values are published at
5. Print the published values on the computers console
6. Console output

7.3. Create measurement file

After successfully establishing the connection to the pc update the pc programme to also write the received temperature sensor values int a .csv file. The file shall be named “temperature.csv”. During the measurement the microcontroller shall send every 5 seconds a new measurement. After 2 minutes the measurement shall be stopped Calculate the mean value of all measurements and upload the measurement file.

Mean value:

TRAINING UNIT 3: DATABASE SYSTEMS

Objective: The trainees:

- Have a basic understanding of databases and related system concepts
- Are able to set up and configure a database server
- Know the basic command sequences for manipulating databases
- Are able to develop programmes which write measured values into databases and read them from databases

Content:

1. Database systems basics

1.1. Basics for usage of databases

1.1.1. Basic concepts of databases

A database intends to have a collection of data stored together to serve multiple applications as possible. Hence a database is often conceived of as a repository of information needed for running certain functions in a corporation or organization. Such a database would permit not only the retrieval of data but also the continuous modification of data needed for control of operations. It may be possible to search the database to obtain answers to queries or information for planning purposes.

Purpose of Database

A database should be a repository of data needed for an organization's data processing. That data should be accurate, private, and protected from damage. It should be accurate so that diverse applications with different data requirements can employ the data. Different application programmers and various end-users have different views upon data, which must be derived from a common overall data structure. Their methods of searching and accessing of data will be different.

Advantages of Using Database

Database minimizes data redundancy to a great extent.
The database can control the inconsistency of data to a large extent.
Sharing of data is also possible using the database.
Database enforce standards.
The use of Databases can ensure data security.
Integrity can be managed using the database.
Various Levels of Database Implementation
The database is implemented through three general levels. These levels are:
Internal Level or Physical level
Conceptual Level
External Level or View Level

The Concept of Data Independence

As the database may be viewed through three levels of abstraction, any change at any level can affect other levels' schemas. Since the database keeps on growing, then there may be frequent changes at times. This should not lead to redesigning and re-implementation of the database. The concepts of data independence prove beneficial in such types of contexts.

Physical data independence
Logical data independence
Basic Terminologies Related to Database and SQL
Relation: In general, a relation is a table, i.e., data is arranged in rows and columns. A relation has the following properties:
In any given column of a table, all the items are of the same kind, whereas items in different columns may not be of the same kind.
For a row, each column must have an atomic value, and also for a row, a column cannot have more than one value.
All rows of a relation are distinct.
The ordering of rows in a relationship is immaterial.
The column of a relation are assigned distinct names, and the ordering of these columns is immaterial.
Tuple: The rows of tables in a relationship are generally termed as Tuples.
Attributes: The columns or fields of a table is termed as Attributes.
Degree: The number of attributes in a relation determines the degree of relation. A relation having three attributes is said to have a relation of degree 3.

Cardinality: The number of tuples or rows in a relation is termed as cardinality.

1.1.2. Necessary software component (database server)

Database component including:

- Storage engine
- Query language
- Query processor
- Optimization engine
- Metadata catalog
- Log manager
- Reporting and monitoring tools
- Data utilities

Storage engine

The storage engine is the core component of the DBMS that interacts with the file system at an OS level to store data. All SQL queries which interact with the underlying data go through the storage engine.

Query language

A database access language is required for interacting with a database, from creating databases to simply inserting or retrieving data. A proper DBMS must support one or multiple query languages and language dialects. Structured query language (SQL) and MongoDB Query Language (MQL) are two query languages that are used to interact with the databases.

In many query languages, the query language functionality can be further categorized according to specific tasks:

Data Definition Language (DDL). This consists of commands that can be used to define database schemas or modify the structure of database objects.

Data Manipulation Language (DML). Commands that directly deal with the data in the database. All CRUD operations come under DML.

Data Control Language (DCL). This deals with the permissions and other access controls of the database.

Transaction Control Language (TCL). Command which deals with internal database transactions.

Query processor

This is the intermediary between the user queries and the database. The query processor interprets the queries of users and makes them actionable commands that can be understood by the database to perform the appropriate functionality.

Optimization engine

The optimization Engine allows the DBMS to provide insights into the performance of the database in terms of optimizing the database itself and queries. When coupled with database monitoring tools, it can provide a powerful toolset to gain the best performance out of the database.

Metadata catalog

This is the centralized catalog of all the objects within the database. When an object is created, the DBMS keeps a record of that object with some metadata about it using the metadata catalog. Then, this record can be used to:
Verify user requests to the appropriate database objects
Provide an overview of the complete database structure

Log manager

This component will keep all the logs of the DBMS. These logs will consist of user logins and activity, database functions, backups and restore functions, etc. The log manager ensures all these logs are properly recorded and easily accessible.

Reporting & monitoring tools

Reporting and monitoring tools are another standard component that comes with a DBMS. Reporting tools will enable users to generate reports while monitoring tools enable monitoring the databases for resource consumption, user activity, etc.

Data utilities

In addition to all the above, most DBMS software comes with additional inbuilt utilities to provide functionality such as:
Data integrity checks
Backup and restore
Simple database repair
Data validations
Etc.

1.1.3 Manual configruation of database server setting

You must customize the database server properties and features by setting configuration parameters, create storage spaces, and configure connectivity. You can automate startup.

You customize the database server properties by setting or modifying configuration parameters in the on config file. You can use the IBM® OpenAdmin Tool (OAT) for Informix® to monitor and update your configuration. OAT provides suggestions for configuration parameter values to optimize your database server configuration. The current version of IBM Informix does not use some configuration parameters that are used in earlier versions of the server.

If you choose to configure a database server during installation, many configuration parameter and environment variables are set and a set of storage spaces are created automatically. Alternatively, you can manually configure the database server.

When you start the database server for the first time, disk space is initialized and the initial chunk of the root dbspace is created. Any existing data in that disk space is overwritten. Shared memory that the database server requires is also initialized. When you subsequently start the database server, only shared memory is initialized. Although the root dbspace is the default location of log files and databases, you can store log files and databases in other storage spaces to prevent the root dbspace from running out of space.

Storage space creation and management

You can create multiple storage spaces to store different types of objects, such as, data, indexes, logs, temporary objects, instead of storing everything in the root dbspace. The way that you distribute the data on disks affects the performance of the database server. You can configure the database server to both automatically minimize the storage space that data requires and automatically expand storage space as needed. You can segregate storage and processing resources among multiple client organization by configuring multitenancy.

Automatic performance tuning

You can set configuration parameters and Scheduler tasks to enable the database server to automatically adjust values that affect performance. By default, many automatic tuning configuration parameters and Scheduler tasks are set to solve common performance issues.

Feature configuration

You can configure the database server to support the types of optional functionality that you need.

Connectivity configuration
The connectivity information allows a client application to connect to the database server on the network. You must prepare the connectivity information even if the client application and the database server are on the same computer or node.
Limit session resources
You can limit the resources available to individual sessions to more evenly distribute system usage, and prevent resource monopolization.
Automate startup and shutdown on UNIX
You can modify startup and shutdown scripts on UNIX to automatically start and shut down the database server.
Automate startup on Windows
You can automate startup of the database server on Windows.

1.1.4. Creation of data base under consideration of security aspects

The scope of database security
Overview
All systems have ASSETS and security is about protecting assets. The first thing, then, is to know your assets and their value. In this chapter, concentrate on database objects (tables, views, rows), access to them, and the overall system that manages them. Note that not all data is sensitive, so not all requires great effort at protection. All assets are under threat. The second thing to know is what THREATs are putting your assets at risk. These include things such as power failure and employee fraud. Note that threats are partly hypothetical, always changing and always imperfectly known. Security activity is directed at protecting the system from perceived threats. If a threat is potential, you must allow for it to become an actuality. When it becomes actual there is an IMPACT. Impact you can consider and plan for. But in the worst case, there will be a LOSS. Security activity here is directed at minimising the loss and recovering the database to minimise the loss as well as further protecting from the same or similar threats.

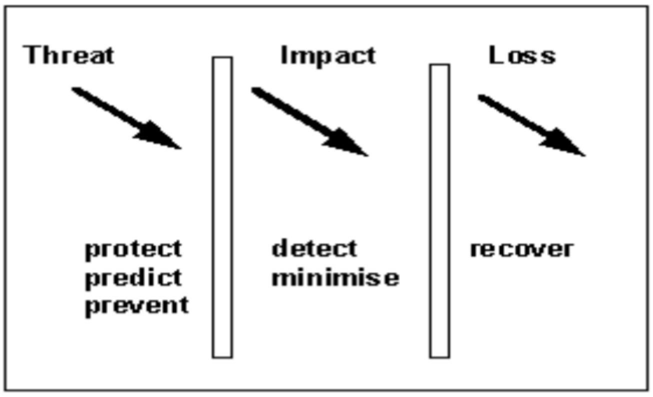


Figure 168: Database security

An outlined development mechanism is:
1. Document assets (what they are, what their value is).
2. Identify treats (what they are, how likely they are, what the impact is if they occur).
3. Associate threats with each asset.
4. Design mechanisms to protect each asset appropriate to its value and the cost of its protection, to detect a security breach against each asset, to minimise the losses incurred and to recover normal operation.

Threats to the database

You will build your security skills from two directions. One is from the appreciation and awareness of changing threats, and the other from the technical remedies to them. Threats include:

- Unauthorised modification: Changing data values for reasons of sabotage, crime or ignorance which may be enabled by inadequate security mechanisms, or sharing of passwords or password guessing, for example.
- Unauthorised disclosure: When information that should not have been disclosed has been disclosed. A general issue of crucial importance, which can be accidental or deliberate Loss of availability: Sometimes called denial of service. When the database is not available it incurs a loss (otherwise life is better without the system!). So any threat that gives rise to time offline, even to check whether something has occurred, is to be avoided. The rest of this section is an overview of the categories of specific regulatory threats to database systems.
- Commercial sensitivity: Most financial losses through fraud arise from employees. Access controls provide both protection against criminal acts and evidence of attempts (successful or otherwise) to carry out acts detrimental to the organisation, whether fraud, extraction of sensitive data or loss of availability.
- Personal privacy and data protection: Internationally, personal data is normally subject to legislative controls. Personal data is data about an identifiable individual. Often the individual has to be alive but the method of identification is not prescribed. So a postal code for a home may in some cases identify an individual, if only one person is living at an address with the postal code. Such data needs careful handling and control. For more information see Data Protection later in the chapter. The issues are too extensive to be discussed here but the implications should be noted. Personal data needs to be identified as such. Controls must exist on the

use of that data (which may restrict ad-hoc queries). Audit trails of all access and disclosure of the information need to be retained as evidence.

- Computer misuse: There is also generally legislation on the misuse of computers. Misuse includes the violation of access controls and attempts to cause damage by changing the database state or introducing worms and viruses to interfere with proper operation. These offences are often extraditable. So an unauthorised access in Hong Kong using computers in France to access databases in Germany which refer to databases in America could lead to extradition to France or Germany or the USA.
- Audit requirements: These are operational constraints built around the need to know who did what, who tried to do what, and where and when everything happened. They involve the detection of events (including CONNECT and GRANT transactions), providing evidence for detection, assurance as well as either defence or prosecution. There are issues related to computer-generated evidence not covered here.

In considerations of logical access to the database, it is easy to lose sight of the fact that all system access imposes risks. If there is access to operating system utilities, it becomes possible to access the disk storage directly and copy or damage the whole database or its components. A full consideration has to take all such access into account. Most analysts would be looking to minimise communications (direct, network and telecommunications) and isolate 5 the system from unnecessary threats. It is also likely that encryption would be used both on the data and the schema. Encryption is the process of converting text and data into a form that can only be read by the recipient of that data or text, who has to know how to convert it back to a clear message. You will find it easier to consider security and auditing as issues separate from the main database functions, however they are implemented. Visualise the security server and audit servers as separate functional modules.

Principles of database security

To structure thoughts on security, you need a model of security. These come in various forms that depend on roles, degree of detail and purpose. The major categories are areas of interest (threats, impact and loss) as well as the actions involved in dealing with them. Security risks are to be seen in terms of the loss of assets. These assets include:

- Hardware
- Software
- Data
- Data quality
- Credibility
- Availability
- Business benefit

Here we are primarily concerned with threats to the data and data quality but, of course, a threat to one asset has consequential impact on other assets. What is always important is that you are very clear on just what asset needs protection. So as a summary: You need to accept that security can never be perfect. There always remains an element of risk, so arrangements must be made to deal with the worst eventuality - which means steps to minimise impact and recover effectively from loss or damage to assets. Points to bear in mind: 6 1. Appropriate security - you do not want to spend more on security than the asset is worth. 2. You do not want security measures to interfere unnecessarily with the proper functioning of the system.

Security models

A security model establishes the external criteria for the examination of security issues in general, and provides the context for database considerations, including implementation and operation. Specific DBMSs have their own security models which are highly important in systems design and operation. Refer to the SeaView model for an example.

You will realise that security models explain the features available in the DBMS which need to be used to develop and operate the actual security systems. They embody concepts, implement policies and provide servers for such functions. Any faults in the security model will translate either into insecure operation or clumsy systems.

Access control

The purpose of access control must always be clear. Access control is expensive in terms of analysis, design and operational costs. It is applied to known situations, to known standards, to achieve known purposes. Do not apply controls without all the above knowledge. Control always has to be appropriate to the situation. The main issues are introduced below.

Authentication and authorisation

We are all familiar as users with the log-in requirement of most systems. Access to IT resources generally requires a log-in process that is trusted to be secure. This topic is about access to database management systems, and is an overview of the process from the DBA perspective. Most of what follows is directly about Relational client-server systems. Other system models differ to a greater or lesser extent, though the underlying principles remain true. For a simple schematic, see Authorisation and Authentication Schematic. Among the main principles for database systems are authentication and authorisation. Authentication 7 The client has to establish the identity of the server and the server has to establish the identity of the client. This is done often by means of shared secrets (either a password/user-id combination, or shared biographic and/or biometric data). It can also be achieved by a system of higher authority which has previously established authentication. In client-server systems where data (not necessarily the database) is distributed, the authentication may be acceptable from a peer system. Note that authentication may be transmissible from system to system. The result, as far as the DBMS is concerned, is an authorisation-identifier. Authentication does not give any privileges for particular tasks. It only establishes that the DBMS trusts that the user is who he/she claimed to be and that the user trusts that the DBMS is also the intended system. Authentication is a prerequisite for authorisation.

Authorisation

Authorisation relates to the permissions granted to an authorised user to carry out particular transactions, and hence to change the state of the database (writeitem transactions) and/or receive data from the database (read-item transactions). The result of authorisation, which needs to be on a transactional basis, is a vector: Authorisation (item, auth-id, operation). A vector is a sequence of data values at a known location in the system. How this is put into effect is down to the DBMS functionality. At a logical level, the system structure needs an authorisation server, which needs to co-operate with an auditing server. There is an issue of server-to-server security and a problem with amplification as the authorisation is transmitted from system to system.

Amplification here means that the security issues become larger as a larger number of DBMS servers are involved in the transaction. Audit requirements are frequently implemented poorly. To be safe, you need to log all accesses and log all authorisation details with transaction identifiers. There is a need to audit regularly and maintain an audit trail, often for a long period.

Access philosophies and management

Discretionary control is where specific privileges are assigned on the basis of specific assets, which authorised users are allowed to use in a particular way. The security DBMS has to construct an access matrix including objects like relations, records, views and operations for each user - each entry separating create, read, insert and update privileges. This matrix becomes very intricate as authorisations will vary from object to object. The matrix can also become very large, hence its implementation frequently requires the kinds of physical 8 implementation associated with sparse matrices. It may not be possible to store the matrix in the computer's main memory.

1.2 Structure of database

Database structure: the building blocks of a database

The next step is to lay out a visual representation of your database. To do that, you need to understand exactly how relational databases are structured.

Within a database, related data are grouped into tables, each of which consists of rows (also called tuples) and columns, like a spreadsheet.

To convert your lists of data into tables, start by creating a table for each type of entity, such as products, sales, customers, and orders. Here's an example:

Each row of a table is called a record. Records include data about something or someone, such as a particular customer. By contrast, columns (also known as fields or attributes) contain a single type of information that appears in each record, such as the addresses of all the customers listed in the table.

First Name	Last Name	Age	ZIP Code
Roger	Williams	43	34760
Jerrica	Jorgensen	32	97453
Samantha	Hopkins	56	64829

Table 5: Data base

To keep the data consistent from one record to the next, assign the appropriate data type to each column. Common data types include:
CHAR - a specific length of text
VARCHAR - text of variable lengths
TEXT - large amounts of text
INT - positive or negative whole number
FLOAT, DOUBLE - can also store floating point numbers
BLOB - binary data

Some database management systems also offer the Autonumber data type, which automatically generates a unique number in each row.
For the purposes of creating a visual overview of the database, known as an entity-relationship diagram, you won’t include the actual tables. Instead, each table becomes a box in the diagram. The title of each box should indicate what the data in that table describes, while attributes are listed below, like this:

Finally, you should decide which attribute or attributes will serve as the primary key for each table, if any. A primary key (PK) is a unique identifier for a given entity, meaning that you could pick out an exact customer even if you only knew that value.

Attributes chosen as primary keys should be unique, unchanging, and always present (never NULL or empty). For this reason, order numbers and usernames make good primary keys, while telephone numbers or street addresses do not. You can also use multiple fields in conjunction as the primary key (this is known as a composite key).
When it comes time to create the actual database, you'll put both the logical data structure and the physical data structure into the data definition language supported by your database management system. At that point, you should also estimate the size of the database to be sure you can get the performance level and storage space it will require

1.2.1. Introduction to relational database model

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.
The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.
The easiest way to understand a database is as a collection of related files. Imagine a file (either paper or digital) of sales orders in a shop. Then there's another file of products, containing stock records. To fulfil an order, you'd need to look up the product in the order file and then look up and adjust the stock levels for that particular product in the product file. A database and the software that controls the database, called a database management system (DBMS), helps with this kind of task

1.2.3. Getting to know the Entity Relationship Diagram for database design and documentation

Entities

Entities are the real-world elements in your system. You could call them the nouns of your database. An ERD shows entities as a rectangle:
For instance, if you're designing a database for an online store, the entities are the product that make up the inventory. Other core entities in your store database will be users and orders.

Relationships
Relationships are the verbs of your ERD and describe how entities are associated with each other. An ERD shows relationships as a labelled diamond on the lines connecting entities:
An online store database has one type of relationship between product and order, and a slightly different relationship between user and order.

Attributes
Attributes are properties or characteristics of entities. You can think of them as adjectives describing the entities in your database. An ERD shows attributes as ovals connected to the relevant entity:
The entities in your online store database will have lots of attributes. To list just a few:
products – name, price and description
users – name, password, address and email address
orders – number of items, date, total amount

2. Handling of databases on server level

2.1. Database manipulation

2.1.1. Creation of a database on a database server

You create a database server by setting mandatory database server properties and then starting the database server.
To create a database server:
Configure the mandatory properties of the database server.
Set configuration parameters in the onconfig file.
Add connectivity information in the sqlhosts file and other connectivity files.
Set environment variables in your environment.
Tip: On Windows operating systems, you can use the Server Instance Manager to configure the mandatory properties of the database server instead of editing the onconfig and sqlhosts file and setting environment variables.

2.1.2. Correct creation of table and database entry for relational database

We can create, read, update and delete (the basic functions of any database) the information in our relational database using a Relational Database Management System (RDBMS). Example of RDBMSs include Oracle, Microsoft SQl Server, MySQL, and PostgreSQL, among many others. Each of these have their pros and cons (and like everything coding-adjacent, their online hyper-partisans), and SQL is not implemented in exactly the same way in each of them. The concepts are the same, but the syntax and keywords may be slightly different, so it is not usually possible to use SQL code written for PostgreSQL in Microsoft SQL Server, for example, without making some modifications.

2.1.3. Getting to know the basic command for manipulating database

All command documentation outlined below describes a command and its available parameters and provides a document template or prototype for each command. Some command documentation also includes the relevant mongosh helpers.

To run a command against the current database, use db.runCommand():

```
db.runCommand( { <command> } )
```

To run an administrative command against the admin database,
use db.adminCommand():

```
db.adminCommand( { <command> } )
```

NOTE

For details on specific commands, including syntax and examples, click on the specific command to go to its reference page.

2.1.4. Excution of join commands to join table within database

We've covered a lot of content in this chapter, from exploring how joins work at a conceptual level, through working with different types of joins, and finally to useful techniques such as aliasing and subqueries.

One of the most important things to remember about how joins work is that we set a condition that compares a value from the first table (usually a primary key), with one from the second table (usually a foreign key). If the condition that uses these two values evaluates to true, then the row that holds the first value is joined with the row that holds the second value.

Let's quickly recap on some of the different types of join we can use:

Join Type	Notes
INNER	Combines rows from two tables whenever the join condition is met.
LEFT	Same as an inner join, except rows from the first table are added to the join table, regardless of the evaluation of the join condition.
RIGHT	Same as an inner join, except rows from the second table are added to the join table, regardless of the evaluation of the join condition.
FULL	A combination of left join and right join.
CROSS	Doesn't use a join condition. The join table is the result of matching every row from the first table with the second table, the cross product of all rows across both tables.

Table 6: Joint functions

When using joins, sometimes our queries can get unwieldy, especially when we're dealing with 2 or more JOINs. To better manage this we can alias table and column names to shorten our query. We can also use aliasing to give more context about the query results.
Finally, the result from a join query can sometimes be obtained using different methods. Subqueries offer another method for us to query the database and retrieve the same results of similar results, as if we had used a JOIN clause.

3. Userprogramme with databases

3.1. Getting to know classes and function to connect database servers

A class is a blueprint of an object. You can think of a class as a concept, and the object is the embodiment of that concept. You need to have a class before you can create an object. So, let's say you want to use a person in your programme. You want to be able to describe the person and have the person do something. A class called 'person' would provide a blueprint for what a person looks like and what a person can do. To actually use a person in your programme, you need to create an object. You use the person class to create an object of the type 'person.' Now you can describe this person and have it do something.

Classes are very useful in programming. Consider the example of where you don't want to use just one person but 100 people. Rather than describing each one in detail from scratch, you can use the same person class to create 100 objects of the type 'person.' You still have to give each one a name and other properties, but the basic structure of what a person looks like is the same.

A function is a combination of instructions that are combined to achieve some result. A function typically requires some input (called arguments) and returns some results. For example, consider the example of driving a car. To determine the mileage, you need to perform a calculation using the distance driven and the amount of fuel used. You could write a function to do this calculation. The arguments going into the function would be distance and fuel consumption, and the result would be mileage. Anytime you want to determine the mileage, you simply call the function to perform the calculation.

3.2. Creating user programmes to create databases on database server

B1: Creating a database

Log in to your OVHcloud Control Panel and select Web Cloud in the top navigation bar. Click Databases in the services bar on the left-hand side, then choose the SQL instance concerned. Click on the Databases tab, then on Add database.
Fill in the fields by following the criteria listed. You can create a user directly by ticking the Create User box.

Database name (obligatory): this will be your database’s name.
Username: This is name of the user that can log in to your database and perform requests (only applicable if the Create User box is ticked).

Rights (only if the box is ticked): the permissions that will be associated with the user on the database. For standard usage, select Administrator. The permissions can be modified as follows.
Password/Confirm password (only if the box is ticked): enter a password, then confirm it.
Finally, click Confirm.

B2: Adding a user

To use an OVHcloud database server, you need to create users with specific rights to connect to a database.
Log in to your OVHcloud Control Panel and select Web Cloud in the top navigation bar. Click Databases in the services bar on the left-hand side, then choose the database name concerned. Next, switch to the Users and rights tab and click Add user.
Enter a “username” and a “password”, then click Confirm.

3.3. Implementation of user programme to store and read on database server

Managing user rights

To allow a user to perform actions on a database, it is necessary to assign permissions to the user.
Log in to your OVHcloud Control Panel and select Web Cloud in the top navigation bar. Click Databases in the services bar on the left-hand side, then choose the database name concerned. Next, switch to the Users and rights tab. Click on the ... button to the right of the user concerned, then on Manage rights. In the left-hand column, Database, you will see a list of the databases on your database server.
The 3 types of permissions proposed are described below:
Administrator: Authorisation of the following queries: Select/Insert/Update/Delete/Create/Alter/Drop
Reading/Writing: Authorisation of the following queries: Select/Insert/Update/Delete
Read: Authorisation of Select queries
None: No database rights
The distribution of rights mentioned above is unique to OVHcloud. This will allow a user with Administrator rights to use DLL (Data_Definition_Language) and DML (Data_Manipulation_Language), while a user with Reading/Writing rights will only use DML.

3.4. Database user Programme

Exercise 1: Requirements: Book and authors

Below a ERM model of a book management system is displayed. Your task is to create the Relational Data Model including all necessary attributes. Also mark primary and foreign keys.

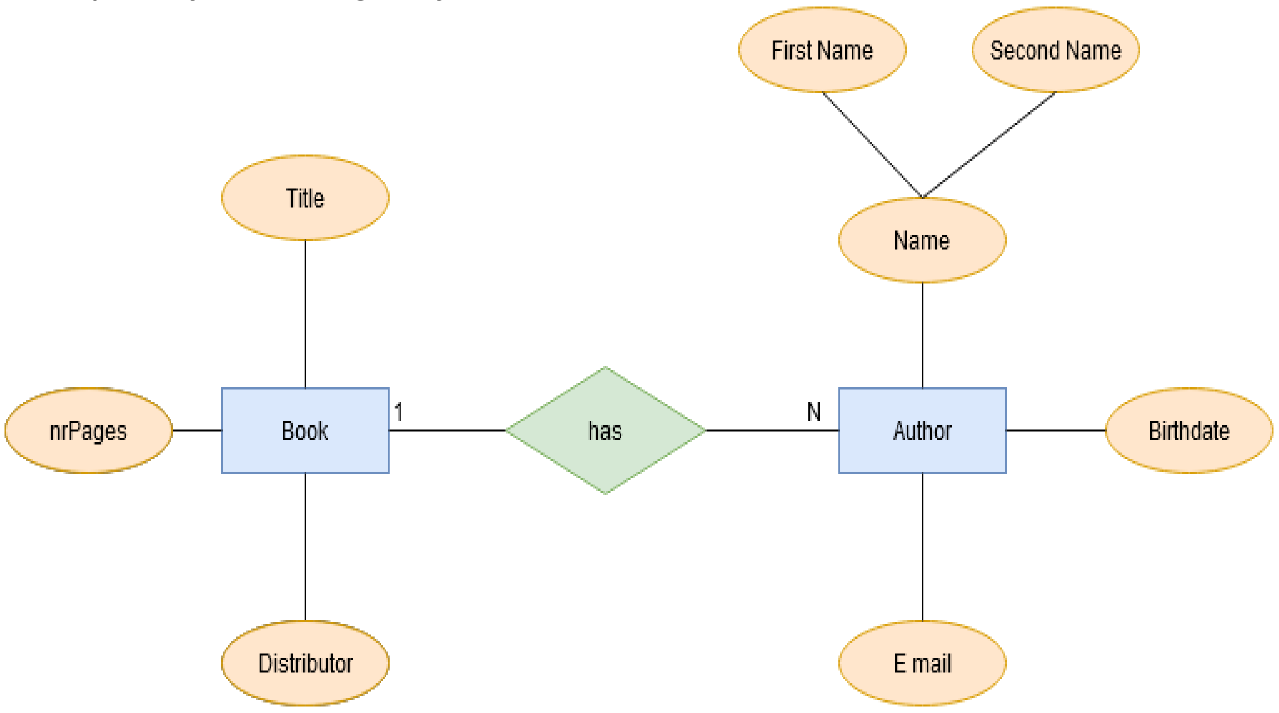


Figure 169: ERM model of a book management system

Relational Data Model:

....

....

....

Exercise 2: Requirements: Art

You are commissioned to design a database in which the most important works of art and their location can be managed. Previously, all information was stored in a single table. The tables design dose not fulfil any requirements necessary for a safe usage of relational data base software. A sample of the table is listed below.

- In a first step a entity relationship diagram shall be drawn to display all entitys and their relationships.
- In a second step the table shall be brought in the N1 form.

Table:

Location	**Country**	**Artist**	**Title**
Prado, Madrid	Spain	Peer Paul Rubens	Die drei Grazien
Louvre, Paris	France	Leonardo Da Vinci	Mona Lisa
Museum of Modern Art, New York City	USA	Vincent van Gogh	'The Starry Night'
Upper Belvedere museum, Vienna	Austria	Gustav Klimt	'The Kiss'
Mauritshuis, The Hague	Netherlands	Johannes Vermeer	'Girl With a Pearl Earring'

Table 7: Artwork database template

ERM

…………………………………………………………………………………………
…………………………………………………………………………………………

Table in N1 form:

…………………………………………………………………………………………
…………………………………………………………………………………………

Exercise 3: Requirements: Car
A car insurance company plans to create a database driven customer tracking system. Sofar customer data was stored without consideration of relational database models. As a first attempt you created a ER model to show the relationship of all entity's and attributes. Explain the customer the tems, entity, relationship and attribute. Explain the customer how the ERM works

ERM

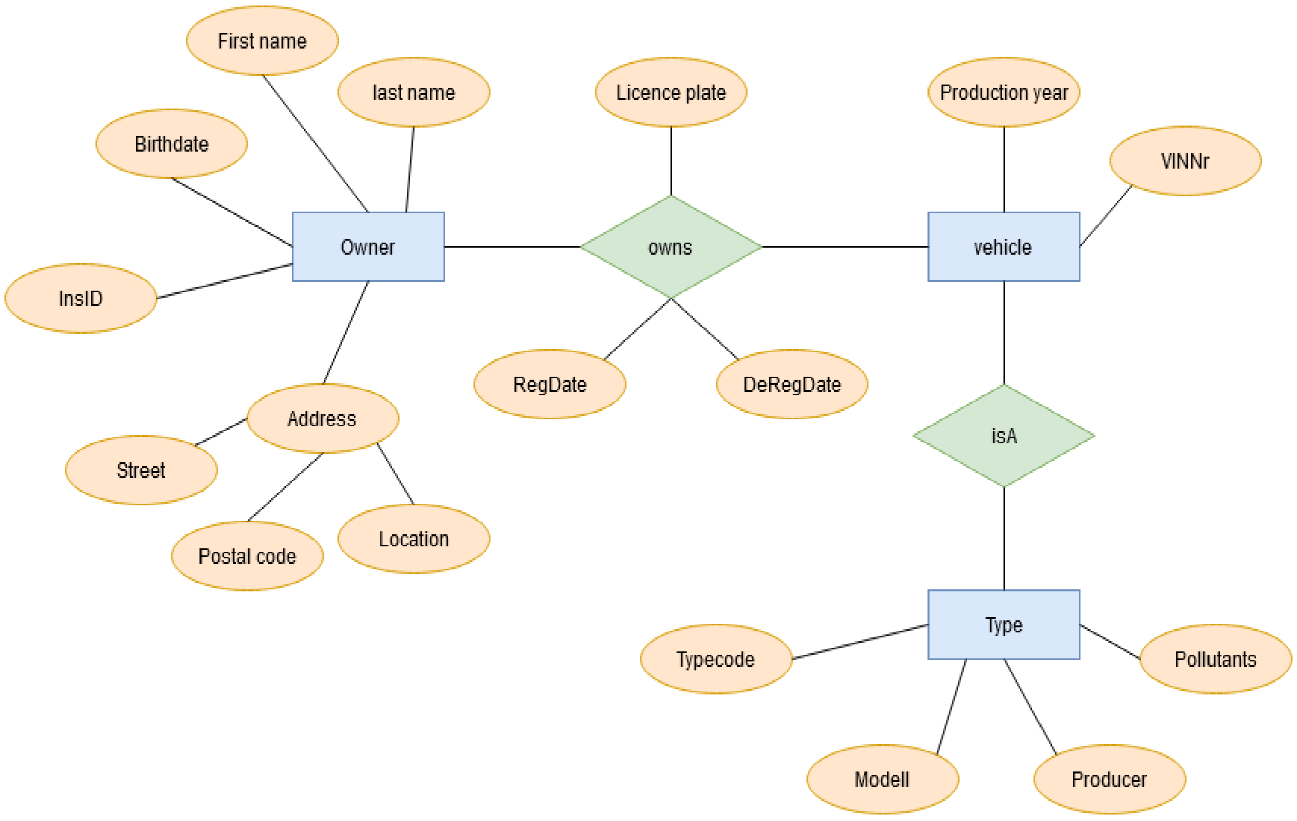


Figure 170: ERM model show all the relationship

Customer explanation:

....
....

Car Table in N1

Requirement

The customer demands a database that fullfills all requirements of N1 form. Further the customer wants to know what N1 form is and how it will benefit him.

Relational database model in N1:

………………………………………………………………………………………
………………………………………………………………………………………

Answers to question:

………………………………………………………………………………………
………………………………………………………………………………………

Car Table in N2

Requirements:

You as experienced database engineer persuaded the customer to bring the database into a higher normalized form. Explain the customer the necessary steps and differences to N1 form if necessary.

Relational databse model:

………………………………………………………………………………………
………………………………………………………………………………………

Differences of N2 to N1 form

………………………………………………………………………………………
………………………………………………………………………………………

Car Table in N3 Form

Requirements:

In a final step you decide to bring the database into N3 form. Explain the customer why I is necessary. Show the customer the finalized relational database model.

Relational database model:

....

Differences of N3 to N2 form

....

Illustration 6: Cloud data storage and secure data storage cloud server

Project: Database Systems: Data acquisition in I4.0 Setup

Introduction

All I4.0 applications have one base product which they rely on, which his data. Production data is already most valuable to companies for optimization and quality measures. Yet company's around the world still have classical production plants without any or only proprietaries data acquisition possibilities. One of the big challenges in future will be to identify and measure production and environmental data and store it. For companies to produce their new product "data" and to lift their productions into the realm of I4.0 it is mandatory to create awareness and skill in measuring important physical values and sending them via networks to storage stations. Relational database systems help to store vast amounts of data in a secure and efficient way. Further they permit a easy to access and maintain solution. In this project a full-scale data acquisition and storage scenario using databases will be implemented.

Requirements:

Experience in object-oriented programming

UML

- Use case diagram
- Class diagram
- Sequence diagram

Flow chart

Software testing

Microcontroller programming

- Hardware control
- Reading in sensors
- Create measurement programmes

Communication methods in computer networks

Finished theoretical and practical part of course LC3

Additional information – for teachers

If other process stations are used then described below adapt the description to your existing system. This project was written in a general hardware independent way in terms or used sensors. Some of the exercises will provide tasks to identify sensor parameters from the datasheet. These need to be adjusted to the used hardware. Further only use sensors where you can also provide a datasheet.

Basic setup

There are two classical production stations. Both production stations are functional and operate without modern ways of data acquisition. Today production data is of great value for every company and delivers the potential for optimization. It will be your task to upgrade the classical production stations into a I4.0 set up. Since selecting the right sensors, creating measurement programmes and sending the data via computer networks to server stations is the basis of I4.0 this project main focus will be on setting up a functional base framework. In this project the entire workflow from system and requirement identification up to practical implementation and testing will be done. The Basic setup will be to extend an existing processing station by measuring physical values. A combination of sensors will be used to count the amount of produced goods. Further a database system shall be set up to store the measured values. A server sided programme shall receive the values via network communication and store them securely in a relational database.

CPSi 4.0

The process station CPSi is displayed in below. The process station can be controlled via microcontroller and system buttons. In this example only the control with system buttons shall be considered. When a system button is pressed the processstation starts by ejecting a cube from the cube magazine. The cube is tested in several stations. After the final station the cube is either ejected with a pneumatic cylinder or passed through.

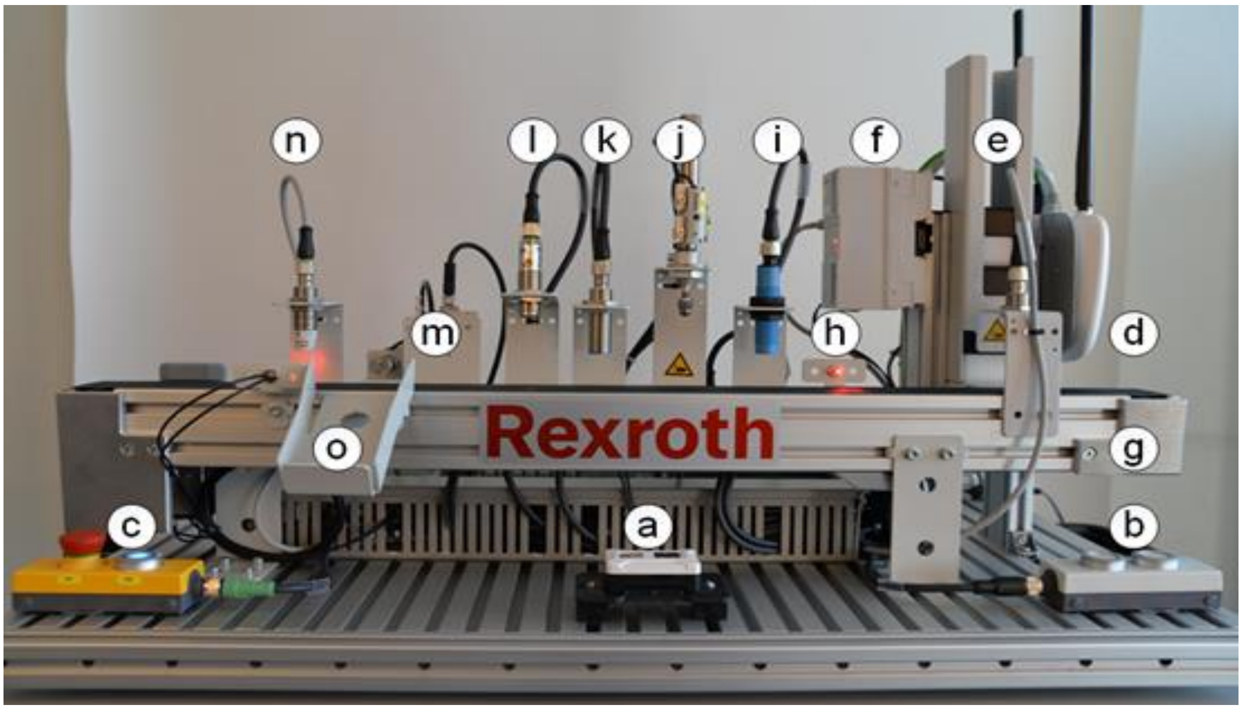


Figure 171: CPS i4.0 training system (front view)

a	Microcontroller (XDK)	i	Capacitive sensor
b	System buttons	j	Position sensor (on contour cylinder)
c	Emergency stop	k	Inductive sensor
d	WLAN router	l	Light barrier
e	Cube magazine	m	Pneu. cylinder and 5/2 directional control valve
f	XM22 (PLC)	n	RFID antenna
g	Conveyor belt	o	Collecting ramp
h	Light conductor		

Table 8: Explanation for Figure 171

System description

Before the test, the cubes are stored in a magazine (fig. 8 - 0). The magazine is manually refilled with new cubes. The buttons can be used to select a cube type and start an order. At the height of the second cube in the magazine, there is a mechanical switch. The switch triggers when no cube is pressed against it (NC - normally closed). Provided that:

there are at least 2 cubes in magazine,
emergency stop released,
light conductor unobstructed,
conveyor belt at a standstill,
all cylinders in basic position,
... the valve operates the magazine.

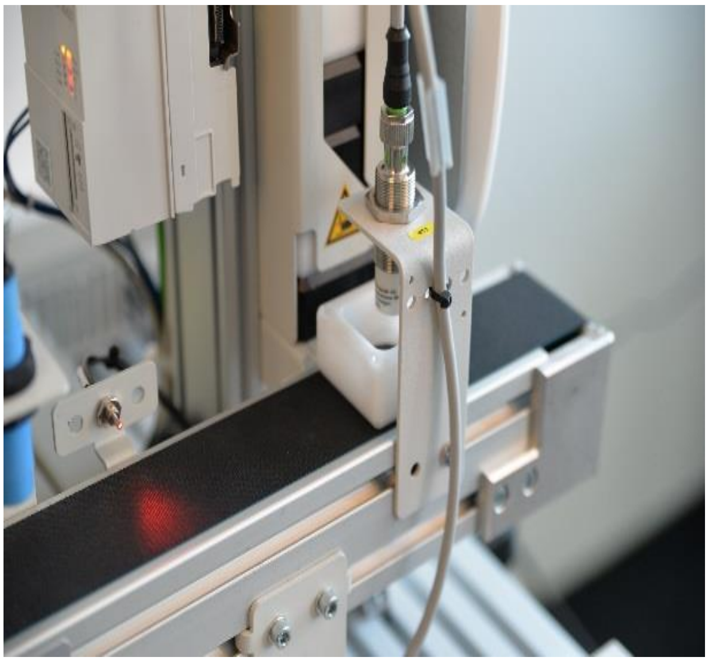


Figure 172: Cubes at start under RFID antenna

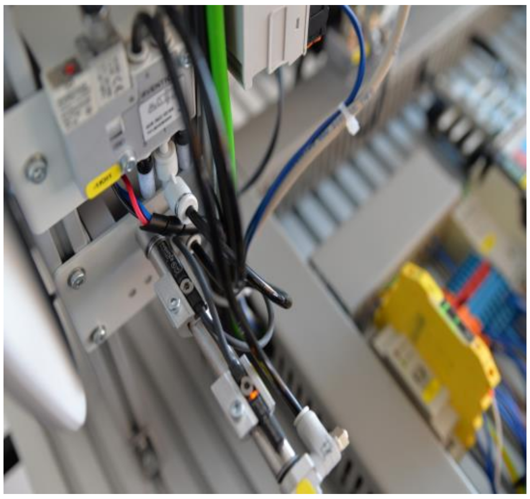


Figure 173: Magazine cylinder with two reed contacts

The valve is located on the back of the magazine together with a pneumatic cylinder. The cylinders are operated using compressed air with at least 1 bar. The compressed air is distributed to the valves by a compressed air distribution system. Depending on the valve position, the valves feed the compressed air into the cylinder chambers. The compressed air supply to the cylinder chambers can be regulated via throttles at the inlet points.

When the piston is extended, it pushes one cube half onto the conveyor belt. The position of the cylinders is detected via reed contacts.

The switches trigger with magnetic materials. There are two reed contacts on the magazine cylinder which detect the basic position and the working position of the piston. A DC motor is flange-mounted on the other end of the conveyor belt. The motor drives the conveyor belt.

The direction of rotation is determined by two relays. Before the conveyor belt starts, the magazine cylinder returns to the basic position and the RFID antenna above the cube half writes a code number for the cube type on the RFID tag in the cube.

The cubes move in the process direction from right to left. They pass several stations on the conveyor belt. Depending on the process at the station, the conveyor belt is stopped or the workpiece is evaluated as it passes through.

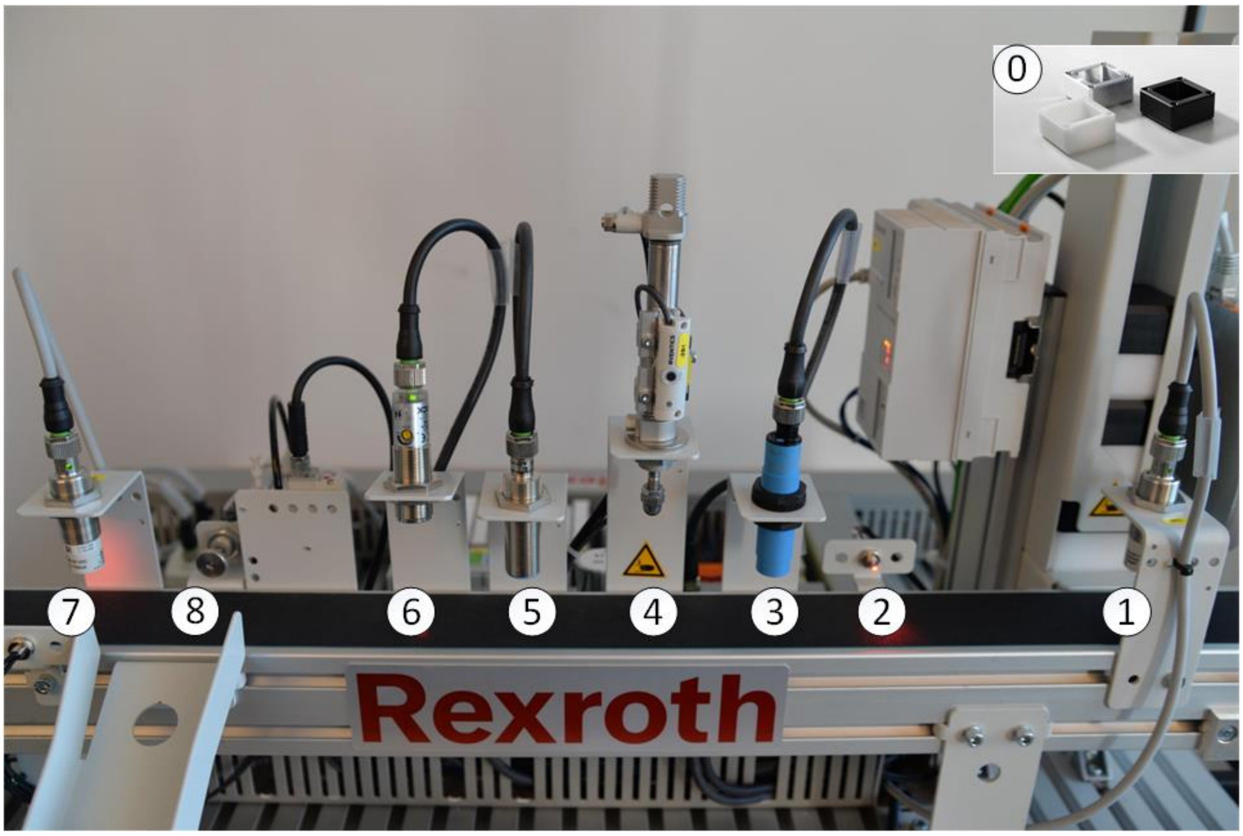


Figure 174: CPSi4.0 workpiece positions

Stations (to fig. 174):
RFID antenna: Code number is written on the tag in the cube.
Light conductor: Detects cube position (lateral)
Capacitive sensor: Detects cube position (top)
Contour cylinder: Piston moves on workpiece and the position sensor measures the distance covered - inference to orientation (base / cover)
Inductive sensor: Detects metallic material - inference to material
Light barrier: Detects bright objects - inference to colour
RFID antenna: Code number is read from tag
Ejection cylinder: Eject position for wrong cubes

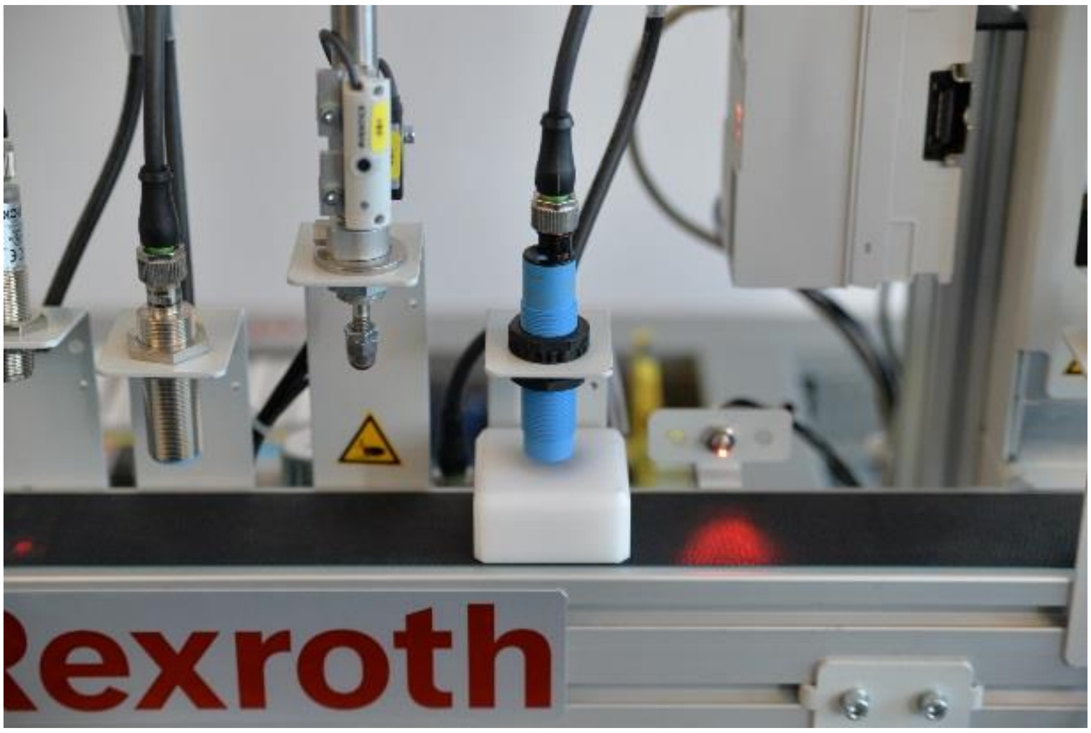


Figure 175: Cube detected by capacitive sensor

The workpiece first passes a light conductor and then a capacitive sensor. The workpiece is detected laterally or from above. Through the feedback, the programme knows when and where a cube is on the conveyor belt. After the sensors have been triggered, the conveyor belt stops so that the workpiece is positioned under the contour cylinder. The redundant detection ensures that the cube half is positioned exactly.

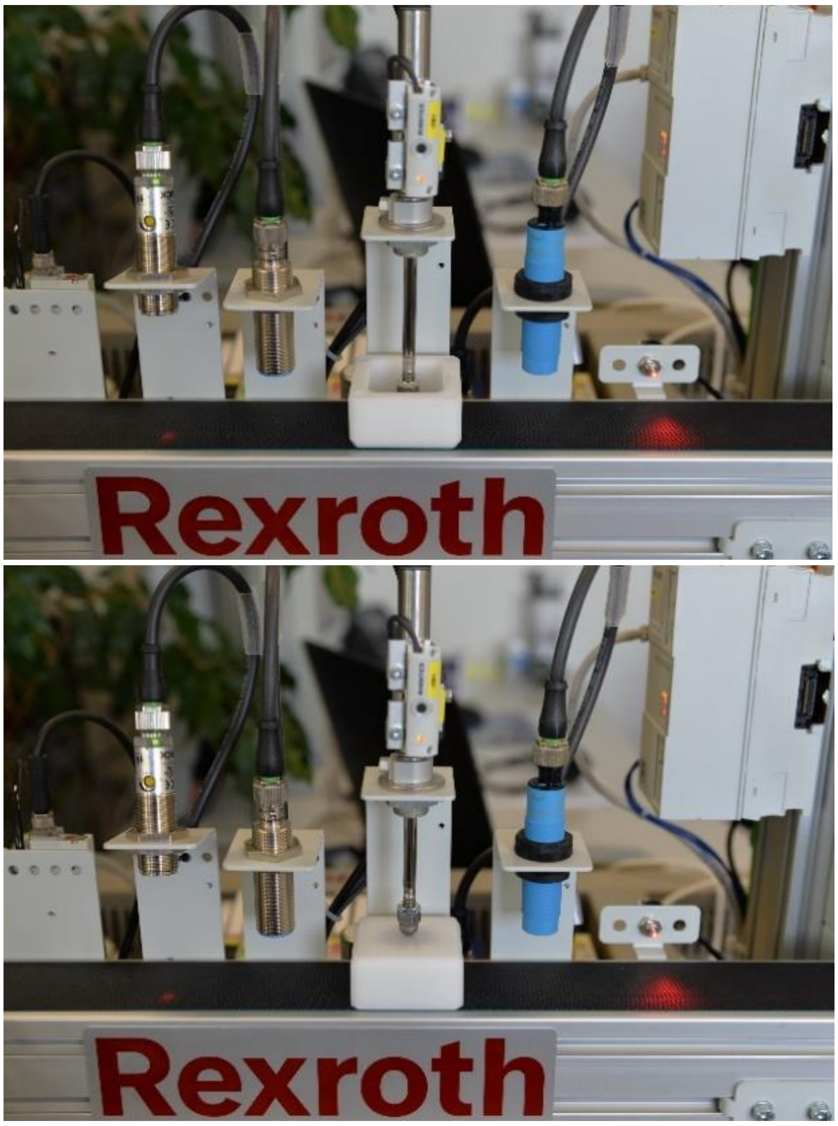


Figure 176: Contour check cylinder with position sensor for base part (above) and cover part (below)

If the conveyor belt stops here, the cylinder piston extends. This cylinder is controlled by a pneumatic valve for closed-loop control (fig. 2 - h). With a base part, the cylinder extends completely. A reed contact will detect this position. With a cover part, the piston presses on the cube. This process is limited in time. After the time has elapsed, the piston retracts. The distance covered is recorded by a position sensor. This sensor is also mounted on the cylinder housing. The workpiece orientation is deduced from the return value.

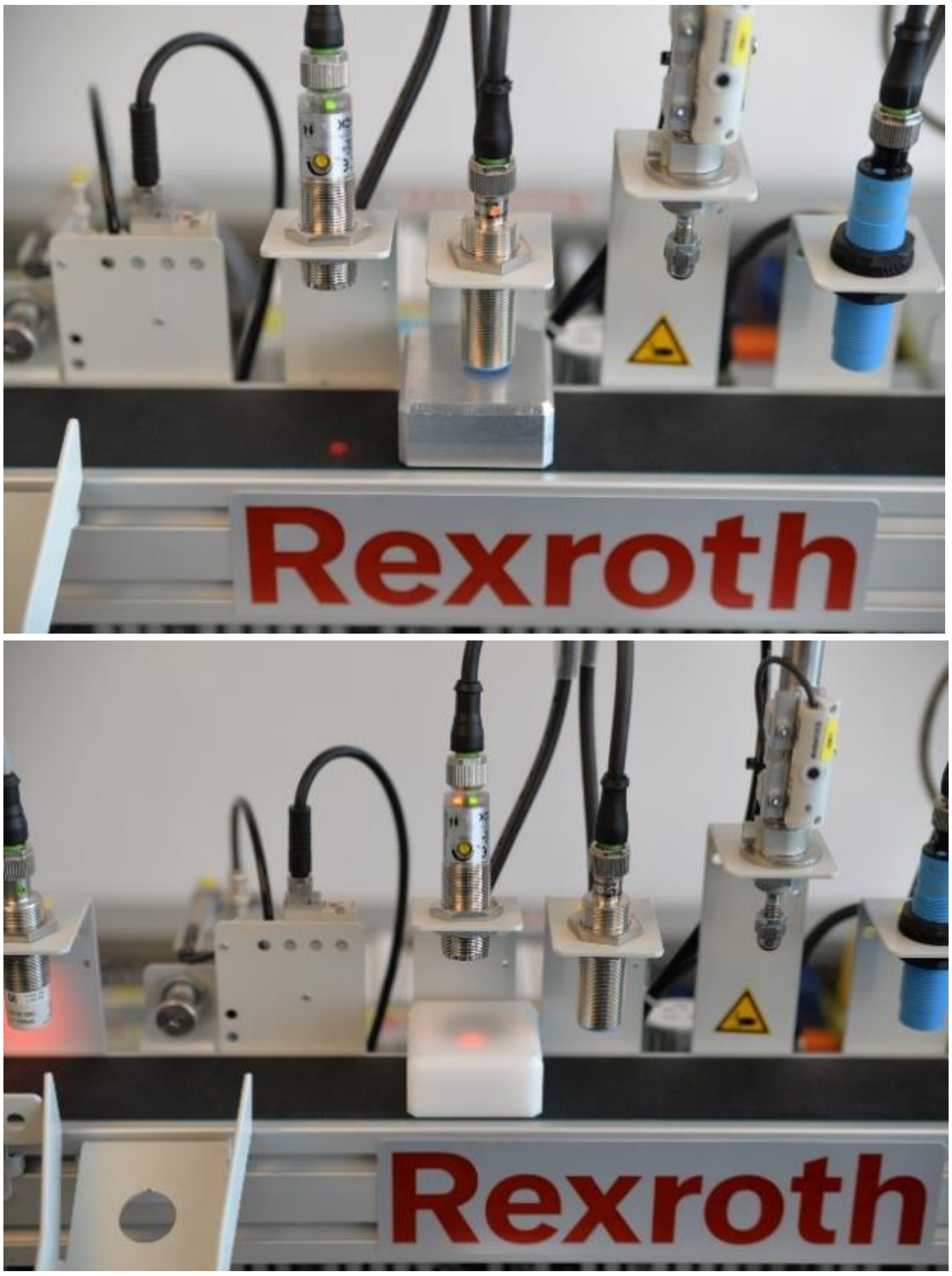


Figure 177: Inductive sensor with aluminum cube (above) and light barrier with plastic cube (below)

If the piston is in the basic position, the conveyor belt continues to move. The workpiece passes an inductive sensor and a light barrier. The inductive sensor only responds to metallic objects. The light barrier detects bright objects (white or silver). An illuminated LED at the top of the sensors indicates that the sensors have detected something. The conveyor belt does not have to stop here. The information of orientation, material and colour are saved.

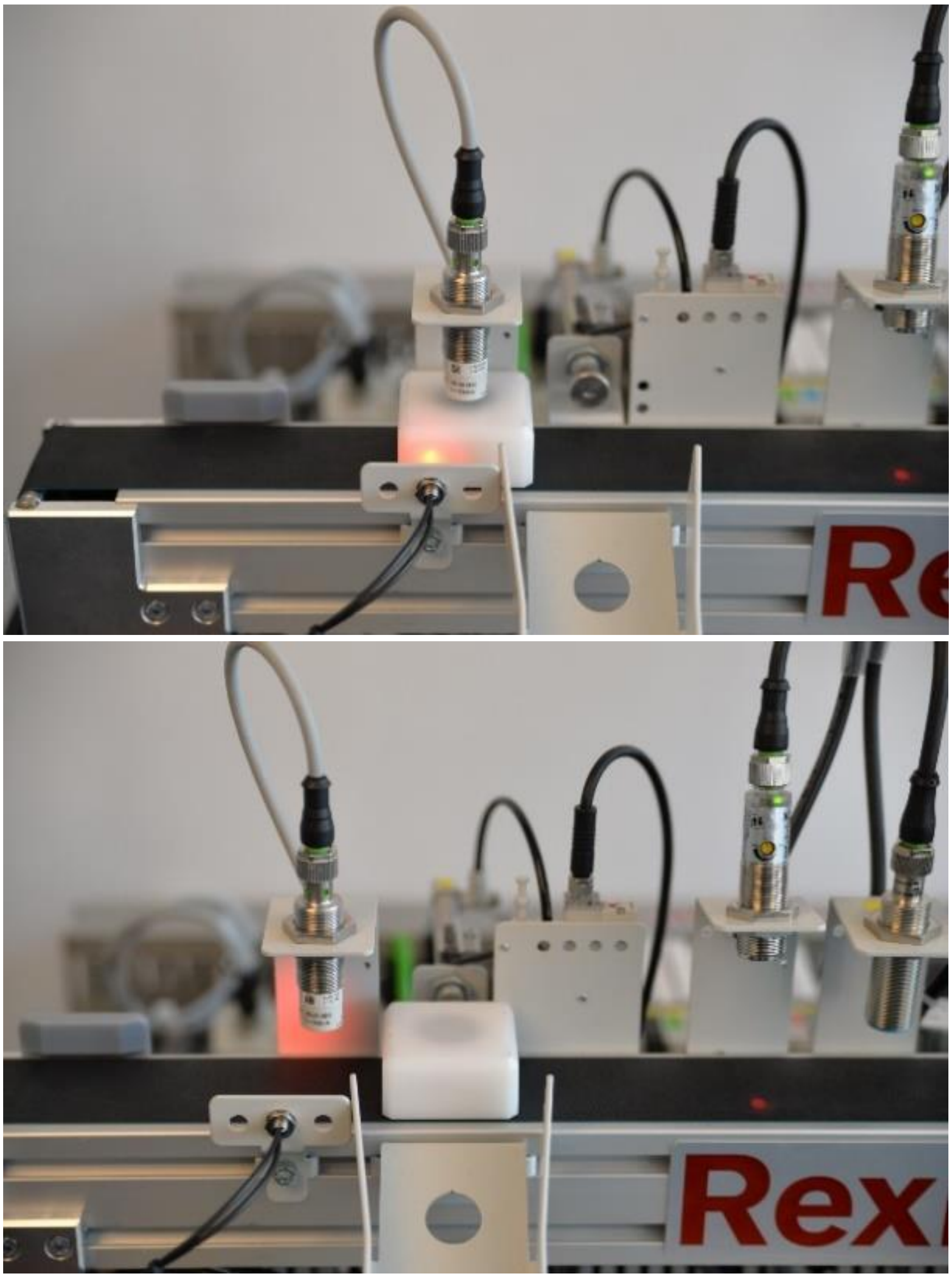


Figure 178: Light conductor stops cube under RFID antenna (above) and wrong cube in ejection position (below)

The workpiece stops at the second light conductor. This is positioned opposite the second RFID antenna. The code number on the tag (cube type) is compared with the saved properties. If there is a match, the cube half moves to the gray mark (end of conveyor belt). In the event of an error, the conveyor belt reverses to the ejection cylinder and is pushed onto the ejection ramp. If there are enough cubes in the magazine, the process is automatically repeated.

Bottling Station

The bottling station is a purely mechanical station. Sofar there are no means of data acquisition present. The bottling station takes bottles, cleans them in the cleaning station, fills them in the filling station and finally puts the cap on the bottle in the capping station.

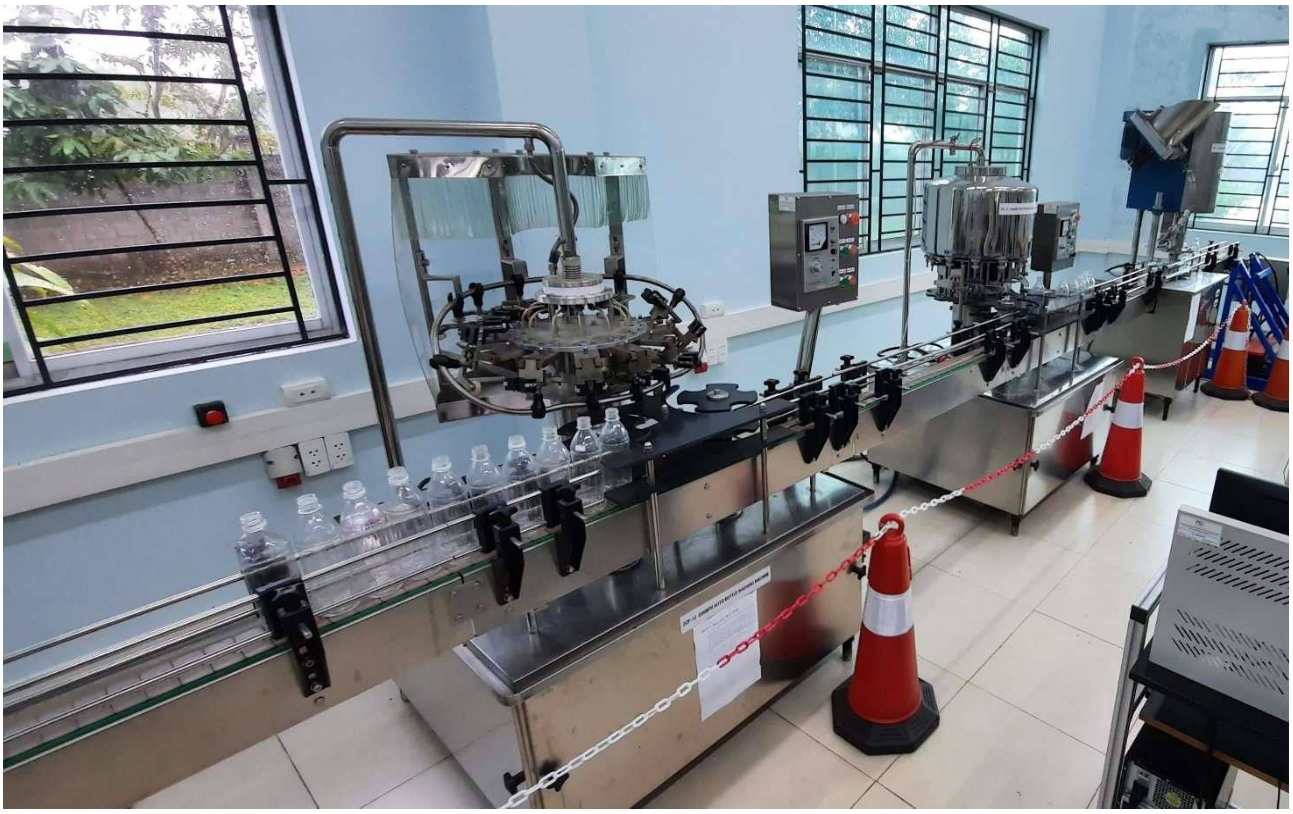


Figure 179: Bottling station full setup, cleaning(left), filling (middle) and capping (right)

After the bottling process has finished new bottles need to be put at the input stream to restart the process. The cleaning station is the first process entered by a bottle in the entire bottling station. The bottles in the stations instream position will be mechanically gripped on their head, moved upwards and cleaned. The entire cleaning process will be conducted within one rotation of the cleaning station. After cleaning has finished the bottle is released in the out-stream position. There is no check for correct placement of bottles on the instream position, nor at the out-stream position. Therefore, it is not possible to detect bottles which got flipped over. Which in term means all bottles have to be placed correctly at the conveyor belt before the process is started.

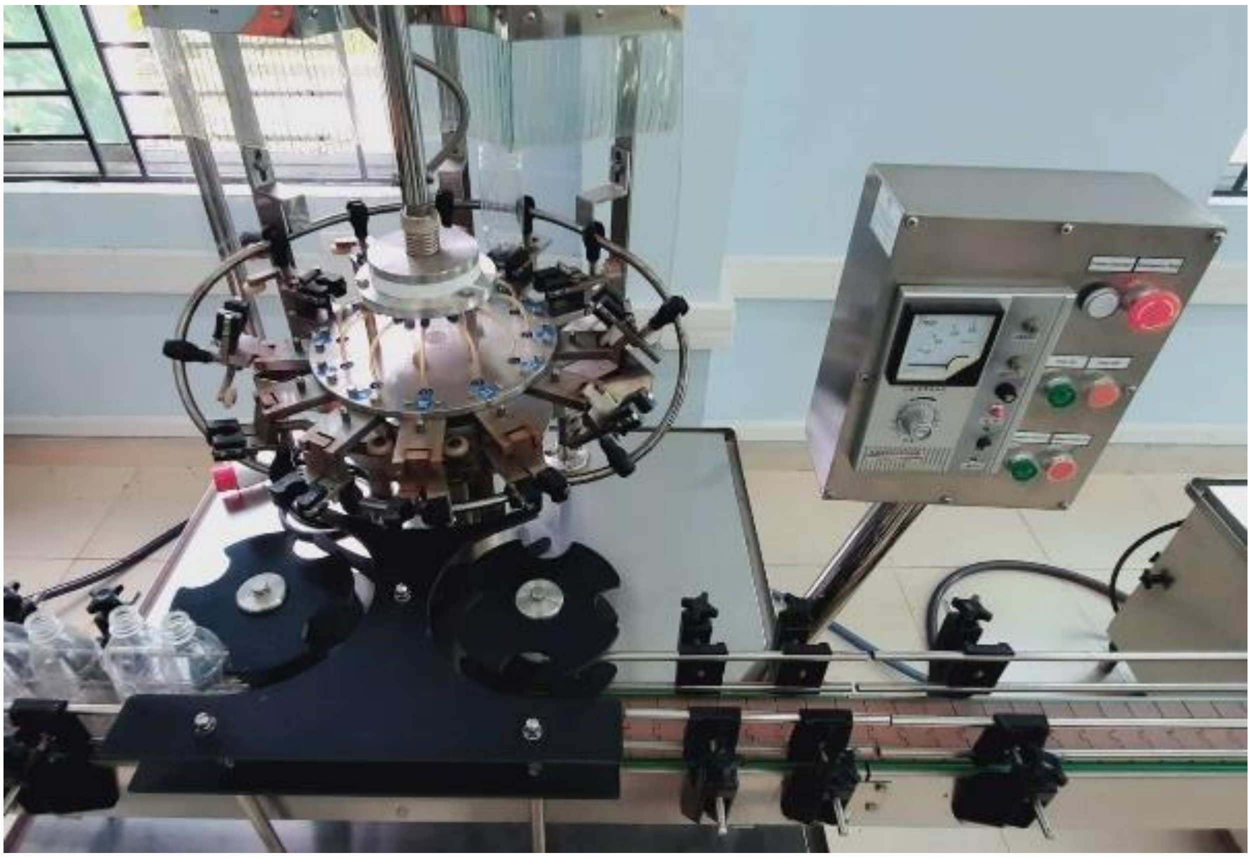


Figure 180: Cleaning station

The filling station mechanically takes in a bottle and mounts it on a predefined position. If a bottle gets mounted the filling valve will be opened and liquid will be filled inside the bottle. The filling process will be automatically initiated if a bottle is at the input position and the rotation process is initiated. After the rotation process has finished and the bottle is at the stations out stream position the valve will be automatically sealed again. On the in stream position its assumed that the bottles already enter sequentially and are positioned correctly. There is no testing or ejection system if a bottle has fallen over.

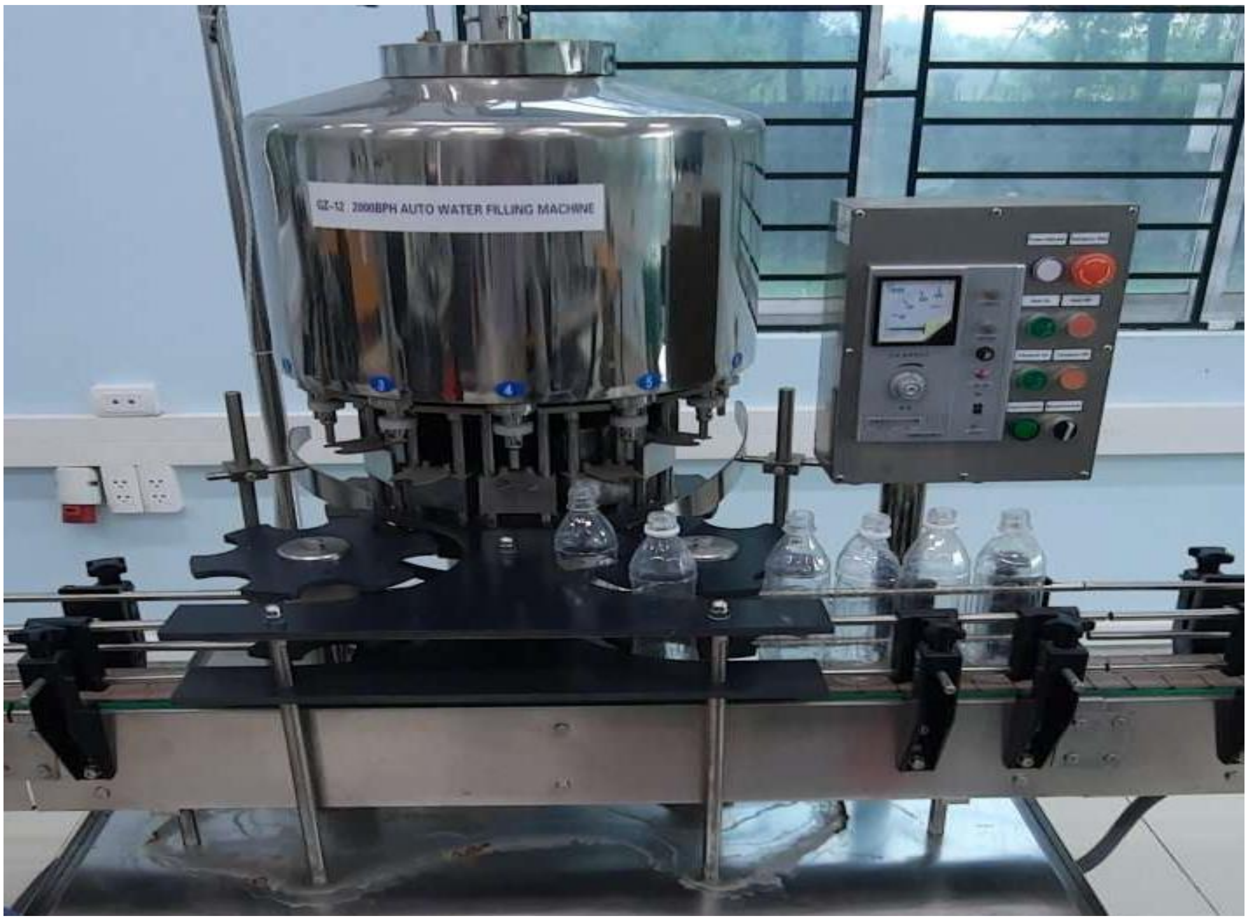


Figure 181: Filling station

After the bottles have been cleaned and filled with water they will be passed to the final station, the capping station. Here the bottles again will be picked sequentially. In one process they will be rotated and a cap will be mounted at the bottles head. Afterwards the bottles will be passed to the stations down stream position. There is no checking if the cap was placed correctly on the bottle head. Neither will here be any checks if the bottles arrive in correct position in the in-stream position. Also, after the bottles have been capped there is no check if the bottles are positioned correctly at the out-stream position. Further the caps need to be manually filled in the caps storage
All three stations deliver possilbitys to mount sensors and measure ambient environmental values.

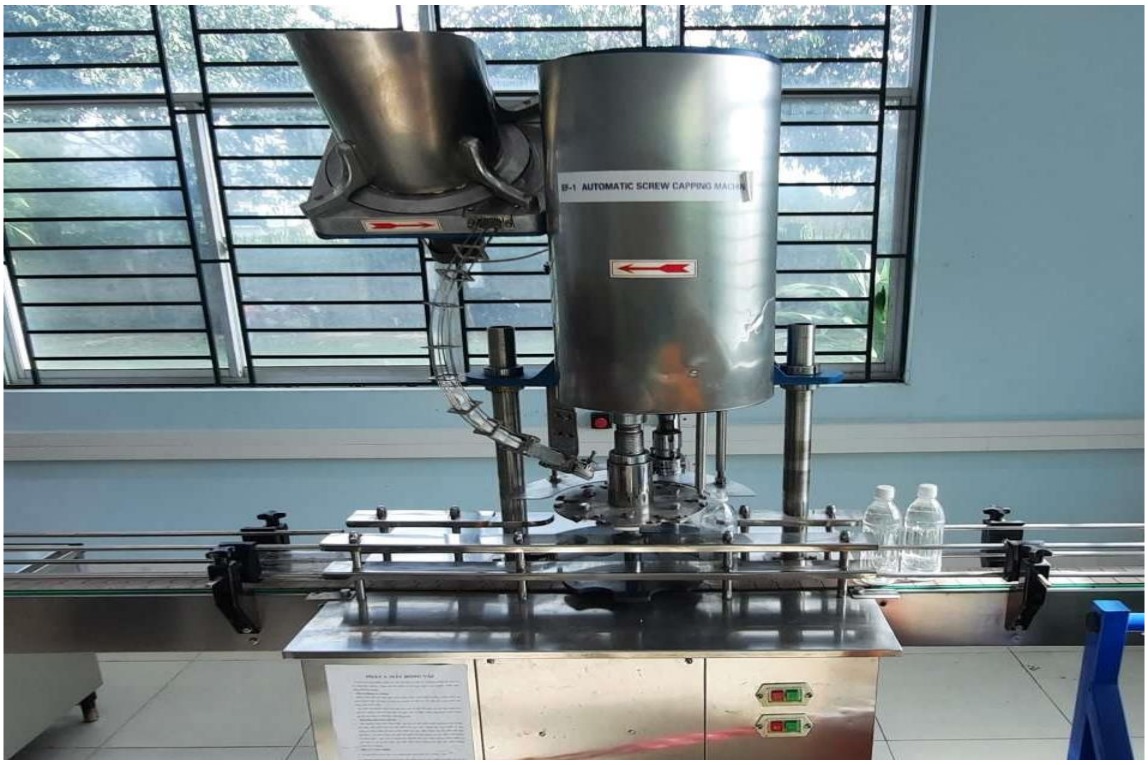


Figure 182: Capping station

Hardware selection – general microcontroller

Object awareness can be measured via different types of sensors. One possibility is the usage of a ultrasonic sensor. These types of sensors are used to measure distances. Their benefit is that they can be used for detection on most surfaces. Depending on the area of application it can be a disadvantage that their detection radius is cone shaped, which can lead to crosstalk phenomena. Further a infrared (IR) sensor will be used. Their benefit is that they have a point shaped detection mechanism. Their disadvantage is that certain surfaces and materials do not reflect the IR signals which causes the sensors to not be able to detect the object.

Identify sensor parameters. In a first step identify the ultrasonic sensor in the datasheet and write its type down:

Ultrasonic sensor Accelerometer

Ultrasonic sensors have several important parameters. Look up in the datasheet and fill out the missing parameters in the table. Make sure to only write the parameters of your sensor:

Parameter	Value
Max range	
Min range	
Max range	

To measure the range with a ultrasonic sensor certain steps are necessary. The steps are written in the datasheet. Summarize all steps here:

Range detection on ultrasonic sensors works via time measurement. To convert the measured time into a distance an equation is provided in the datasheet. Write down the equation here:

Measurement programme – basic microcontroller

After selecting the hardware, a simple measurement programme shall be implemented.
The measurement programme shall read in the ultrasonic sensors values and the IR sensors input values and print them directly on the console. In this case we assume the IR sensor delivers a binary signal if an object got detected. Every measurement shall be taken with a interval of 5 seconds. Enter a screenshot of the console output in the table below. Upload your code and a screenshot of the console output.

The required output shall be:
“Ultrasonic sensor: XXXcm”
“IR sensor detection: XXX”

Console output

Measurement programme – Bosh XDK

The bosh XDK is a robust microcontroller platform with a wide sensor array. It is commonly used in industrial applications especially for data tracking. The bosh XDK delivers a wide sensor array, a integrated wifi module, a sd card reader to store data on a memory card, freely configurable buttons and a usb connector to connect the XDK to a PC for flashing new programmes and directly outputting values in the console of its own IDE.

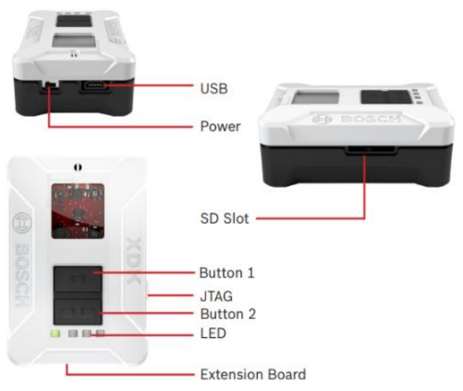


Figure 183: Example of Databoads

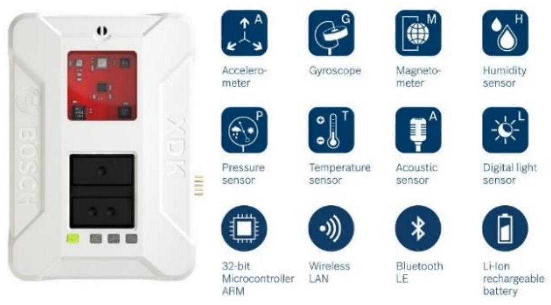


Figure 184: Compare the differences between Reports and Dashboards

The bosh XDK further permits its user to access many types of sensors. Users can measure acceleration, temperature, humidity, air pressure and many more. Further the XDK uses “FreeRTOS” as realtime operating system. This ensures that precicly timed measurements with defined intervals are possible.

Further bosh uses a own eclipse based IDE called XDK-Workbench. This IDE gives the user the choice to either programme the XDK directly with C, or to use MITA programming language. MITA programming language was developed to lessen the burden of creating I4.0 applications without knowledge of embedded programming. The website: https://developer.bosch.com/web/xdk/getting-started#1 is provided by bosh and has sample code for reading in sensor values. Part of this task is to read through the descriptions and sample codes and reuse them to create a working programme.

The task of this section is to create a measurement programme that reads in the sensor values of the:
Accelerometer
Humidity sensor
Temperature sensor
Pressure sensor

The read in values shall be outputted on the console of the XDK – Workbench. The project shall be programmed in MITA programming language. Measurements shall be done every 3 seconds. The desired console output shall be as follows:
Acceleration in X: XXXg
Acceleration in Y: YYYg
Acceleration in Z: ZZZg
Humidity: XXX%
Temperature: XXX°C
Air pressure: XXXkPa
Enter a screenshot of the console output in the table below. Upload your code and a screenshot of the console output.

Console output:

Sensor placement and wiring

After the test programmes were successfully completed the sensors can be placed a desirable location. For the bottling station the sensors have to be placed in a way that the ultrasonic sensor measures if a bottle is present from the side. The IR sensor shall be used to measure if a cap was put on the bottle.

Find a valid location for the sensors to fullfill this requirement. On the CPSi 4.0 station the same procedure shall be conducted. Instead of bottles the position of the blocks shall be figured out. Make sure to adjust the IR sensor so it can detect if the block has the hole on the upside or not. Find a valid location for the sensor placement and document your decision with a picture. Further describe in your own words why you choose this location in the table below:

Picture	Documentation

Communication protocol

After the sensors and microcontroller got a valid location for measurement the communication protocol shall be implemented. After implementing one of the solutions take a picture of the console output of the pc and print it in the field below.

Further upload the console output and all code.

Common microcontroller

The before described sensors of the microcontroller shall be used to create a measurement programme. As communication protocol the publisher and subscriber method using the MQTT protocol shall be used. In this example all measured data shall be converted to a JSON object. This object shall be passed over MQTT. Therefore, there will only be one topic to send the data. The ultrasonic data shall be named “Distance”, the IR sensor data shall be named “Detected”.

Steps:

1. Setup a broker on the computer and run the broker
2. Use your measurement programme for the microcontroller to read in the sensor values. Let the microcontroller read in with an interval of 500 ms
3. Save all measured values in a JSON object, choose the names according to the requirements
4. Include the necessary libraries on the microcontroller and create the topic to be published to.

   a. Topic name

   i. Measurement/States
5. Include the necessary libraries on the computer and subscribe to the topic where the sensor values are published at
6. Print the published values on the computers console
7. Test different sensor interval times

Console output

XDK

The XDK offers the possibility to transmit data via publisher and subscriber method using the MQTT protocol. Since the XDK offers a great variety of sensors ultiple sensors shall be transmitted to the computer. In this example all easured data shall be converted to a JSON object. This object shall be passed over QTT. Therefore, there will only be one topic to send the data. The sensor data shall be named as follows:

For acceleration
 AccelX
 AccelY
 AccelZ
For humidity
 Humidity
 For temperature
Temperature
 For Pressure
 o Pressure
Steps:
1. Setup a broker on the computer and run the broker
2. Use your measurement programme for the microcontroller to read in the sensor values. Let the microcontroller read in with an interval of 3 seconds
3. Include the necessary libraries on the microcontroller and create the topic to be published to.
a. Topic name
i. MeasurementXDK/ States
4. Include the necessary libraries on the computer and subscribe to the topic where the sensor values are published at
5. Print the published values on the computers console

Console output

Create a ERM diagram

After the measurement and communication systems got set up the relational database system can be started. As a first step a ERM diagram shall be created. For the ERM diagram it is necessary to model all parts of the system. The Database shall include the stations name, the sensor names, sensor types, the sensor values, the microcontroller type and the time stamp when the measurement was taken. All names have to be in english. Draw the ERM diagram below:

Setup database

The ERM diagram is the basis of every database. After it was created Create the database itself. The database name shall be either “Bottling Station” or “CSPi4.0” depending on which station you work. All used names shall match the ERM diagram. All tables have to be in 3rd normal form, if necessary normalize the tables. Upload and include a picture of all created tables and rows. Also upload the picture.

Table

Database programme

After the database is set up and the measurement programme works it is ossible to store the measured data inside the database.

Writing to database

Whenever the programme receives a value over MQTT all received sensor values plus the current time stamp of reception shall be saved into the database. Ensure that the right tables and values are send to the database. Further ensure that the time stamp has the right format to fit with your used database. Below a small list of necessary SQL commands is presented. It is also necessary to provide a library to send SQL commands to your database.

Test your setup by creating a programme that reads in values for 5 minutes and save the measured values to the database. Before starting your measurement, programme ensure the stations are running. Upload the code.

SQL commands:

Inserting values into a table

INSERT INTO tableName

VALUES (value1, value2, value3, ...);

Reading row values from table

SELECT colName FROM tableName

Reading from database

After completion load all stored sensor values from the database. Create a visualization of the sensor values. Put every sensor value into a own diagram. As tile choose the sensor name. Insert the diagrams below. Upload the code and a screenshot of the diagrams

TRAINING UNIT 4:

DATA VISUALISATION WITH DASHBOARDS

Objective: The trainees:

- Understand the basics of dashboards
- Distinguish the types of dashboards, distinguish the similarities and differences of dashboards and reports
- Understand the steps to design a complete dashboard
- Know the limits of a dashboard
- Understand the process to create a dashboard
- Can create simple dashboards
- Can create a visualization programme

Content:

1. Dashboards basics

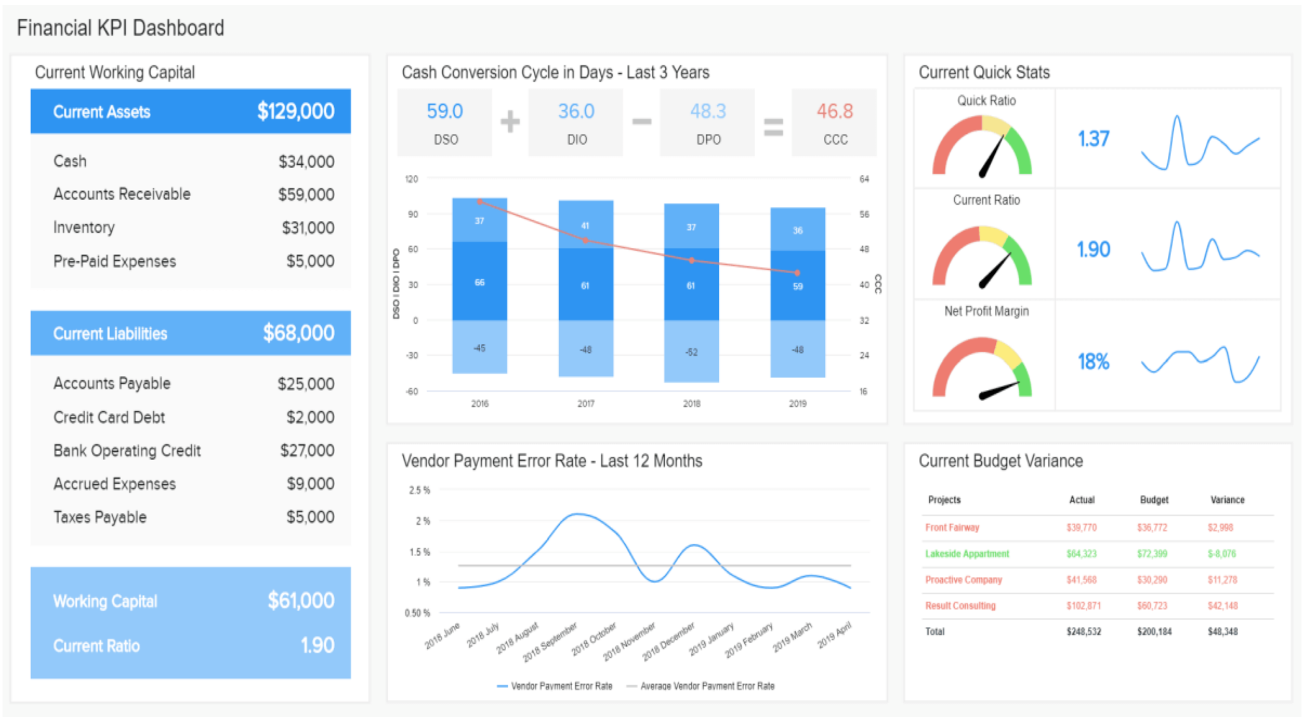
1.1. Defining the usage and tasks of dashboards

1.1.1. What is a data dashboard?

A data dashboard is a tool that provides a centralized, interactive means of monitoring, measuring, Analysing, and extracting relevant business insights from different datasets in key areas while displaying information in an interactive, intuitive, and visual way.

They offer users a comprehensive overview of their company's various internal departments, goals, initiatives, processes, or projects. These are measured through key performance indicators (KPIs), which provide insights that help to foster growth and improvement.

Online dashboards provide immediate navigable access to actionable analytics that has the power to boost your bottom line through continual commercial evolution.


Financial KPI Dashboard

Current Working Capital

Current Assets	$129,000
Cash	$34,000
Accounts Receivable	$59,000
Inventory	$31,000
Pre-Paid Expenses	$5,000

Current Liabilities	$68,000
Accounts Payable	$25,000
Credit Card Debt	$2,000
Bank Operating Credit	$27,000
Accrued Expenses	$9,000
Taxes Payable	$5,000

Working Capital	$61,000
Current Ratio	1.90

Cash Conversion Cycle in Days - Last 3 Years

59.0 DSO + 36.0 DIO − 48.3 DPO = 46.8 CCC

Current Quick Stats

Quick Ratio	1.37
Current Ratio	1.90
Net Profit Margin	18%

Vendor Payment Error Rate - Last 12 Months

Current Budget Variance

Projects	Actual	Budget	Variance
Front Fairway	$39,770	$36,772	$2,998
Lakeside Appartment	$64,323	$72,399	$-8,076
Proactive Company	$41,568	$30,290	$11,278
Result Consulting	$102,871	$60,723	$42,148
Total	$248,532	$200,184	$48,348

Figure 185: Compare the similarity between Reports and Dashboards

To properly define dashboards, you need to consider the fact that, without the existence of dashboards and dashboard reporting practices, businesses would need to sift through colossal stacks of unstructured data, which is both inefficient and time-consuming. Alternatively, a business would have to 'shoot in the dark' concerning its most critical processes, projects, and internal insights, which is far from ideal in today's world.

1.1.2. What Is The Purpose Of A Data Dashboard?

As mentioned earlier, a data dashboard has the ability to answer a host of business-related questions based on your specific goals, aims, and strategies. By taking raw data from a number of sources and consolidating it before presenting it in a tailored, customized visual way, data dashboards can help make sense of your company's most valuable data and empower you to find actionable answers to your most burning business questions.

Through linking with specific KPIs that align with your business goals, you can drill down into specific pockets of information, creating benchmarks, and measuring your success on a continual basis.

In doing so, your business will be data-driven, and as a direct result – more successful. To find out more about dashboards and key performance indicators, explore our ever-expanding collection of various business-boosting KPI examples and templates.

1.2. Theoretical foundations for the creation of web applications

In order to be able to make empirical observations, one needs theoretical concepts that can be applied. We are utilizing a concept of information based on different subprocesses of information that take place in social life and are technically supported by ICTs. These are cognitive, communicative, and co-operative processes.

- Cognitive processes (including emotional ones) are individual, or ,in case of any supraindividual social agency named a subject, intra-subjective processes of generating information.Human-Computer Interaction as discipline deals with how cognition is being supported and influenced by using ICTs.
- Communicative processes are interactive, that is, among individuals or other social subjects.Due to the coupling of cognitive subjects, communicative processes can be understood as information generation processes. Computer-mediated communication deals with these processes supported by ICTs
- Cooperative processes are integrative, concern the supra-individual level and let information emerge from synergetic effects of communicating subjects. Originally, Computer-Supported Cooperative Work researched this topic from the perspective of the involvement of ICTs.

Nowadays, this approach takes advantage from research in collective intelligence, wisdom of the crowds and so on.

1.3. Comparison of different creation methods

1.3.1. What is a report?

Reports can be a presentation of corresponding charts and other visualizations, or they can be a large set of charts and visualizations that may or may not directly relate. A report is meant to be used to gather detailed intelligence on the operations within an organization, thus a report can be either very broadly covering a wide scope of related information, or narrowly focusing on details of a single item, purpose, or event. All of this information, while presented in a report, is meant to be a snapshot in time.

In a data visualization platform, like Chartio, a report can even be built in the same environment as a dashboard leading to even more confusion about the difference between the report and dashboard. This report might even have the look and feel of a dashboard, in this case you are likely creating a “dashport” of sorts. Reports can also be series of dashboards that may be interrelated to each

other and as a whole show more of the information needed to understand the status of things. These groups of dashboards can either be linked directly or grouped in the software's library, categorization, or organization technology.
Depending on the number of I/O modules that the PLC owns, they may be in the same enclosure as the PLC or in a separate enclosure. Some small PLCs called nano/micro PLCs usually have all their parts including power, processor, etc. in the same enclosure.

1.3.2 How do Reports and Dashboards differ?

First, a report contains much more detailed information. Where a dashboard might only provide a CEO with information on how the entire company's sales are progressing, a corresponding report will give the CFO or VP of Sales the ability to see how each sales region, or even sales person is performing and make leadership decisions. Just like responsibility, data will get more granular and more into the minutiae as the organizational hierarchy goes down The C Suite might be interested in the detailed data, but for seeing a snapshot of high level information, the dashboard is the desired mode.

Second, a Report is much longer than a dashboard. Not only in the amount of detail, but also visually. Tables and charts that live within a report can take up many pages of a printed medium, and can even be books or many volumes of books. In the electronic mediums, a report will likely require the reader to scroll through many screens or click from page to page.

This is important because of the first of noted Data Visualization Professor Stephen Few's common mistakes of Information Dashboard Design, "Exceeding the boundaries of a single screen."

Few describes it in this way. “My insistence that a dashboard should confine its display to a single screen with no need for scrolling or switching among multiple screens, might seem arbitrary and a bit finicky, but it is based on solid and practical rationale. After studying data visualization and visual perception for a while, we discover that something powerful happens when we see things together, all within eye span. Likewise, something critical is compromised when we lose sight of some data by scrolling or switching to another screen to see other data.”

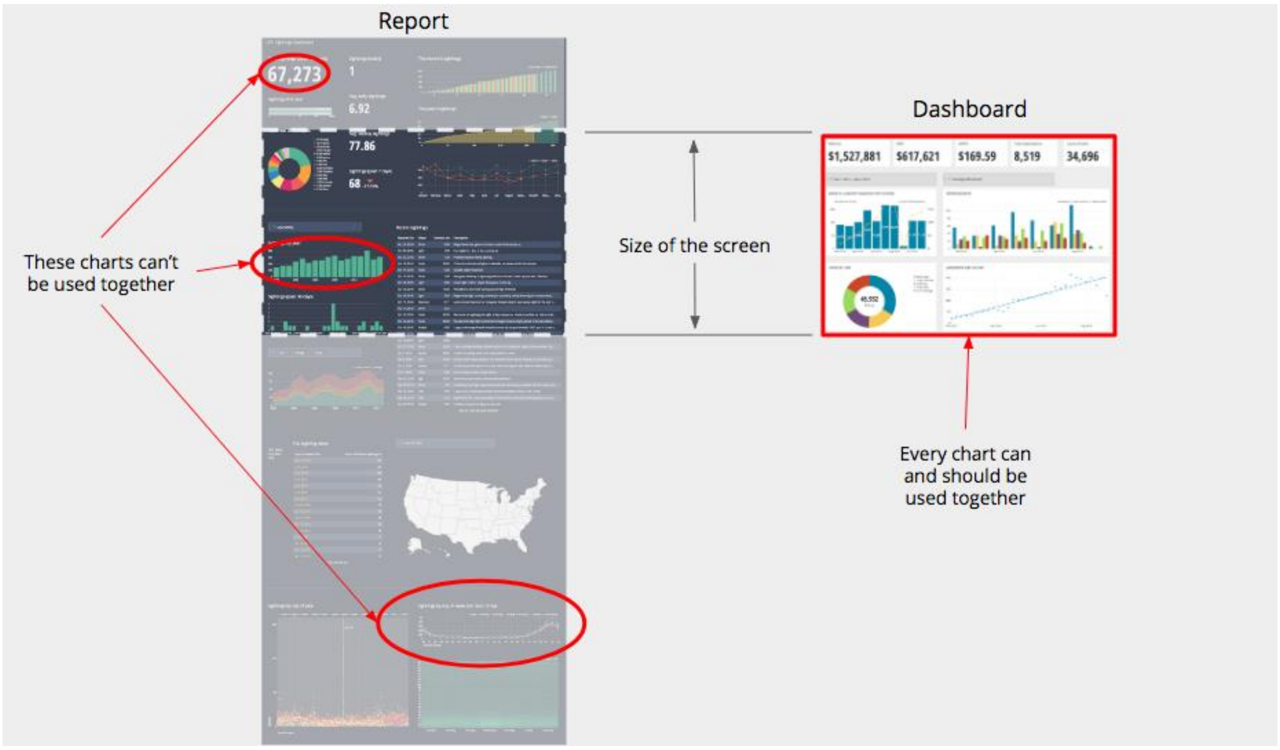

Report
67,273
These charts can't be used together
Size of the screen
Dashboard
$1,527,881
$617,621
$169.59
8,519
34,696
Every chart can and should be used together

Figure 186: Show only the most important content

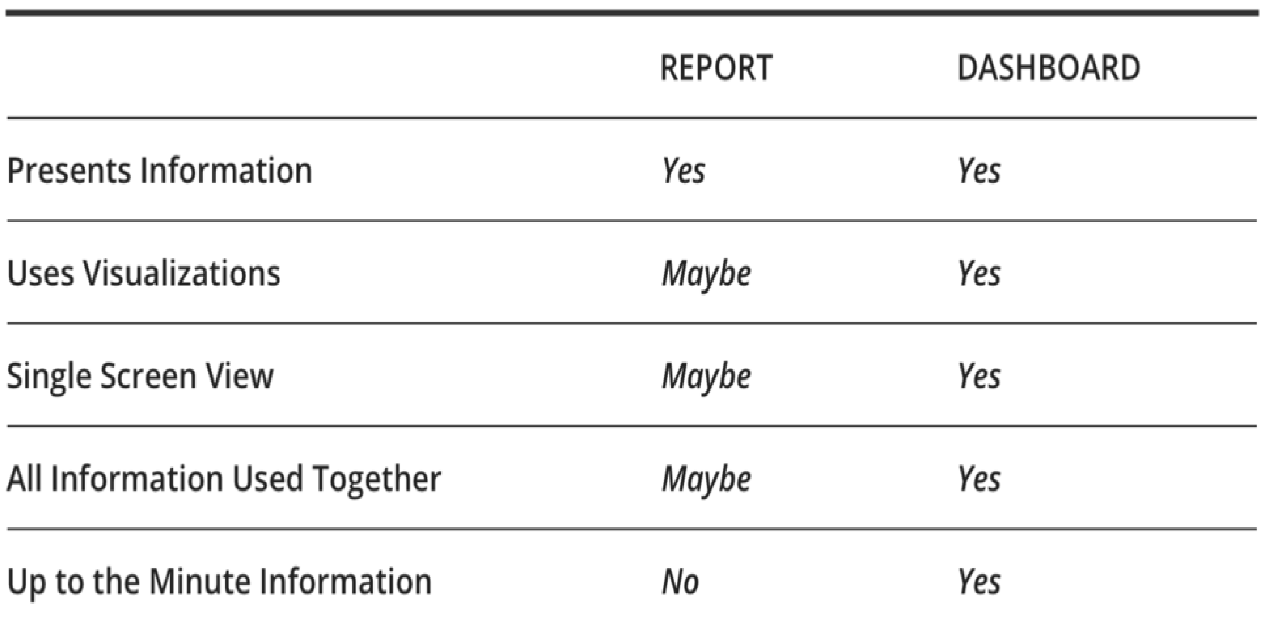
When an individual dashboard has so much information on it that scrolling is required, the power of the dashboard is diminished because the information that lives there is intended to be viewed together. Each piece of information on the dashboard is meant to give the reader the ability to answer part of the central question of the dashboard. These charts combine to answer the question, so if the reader can’t see them together, making them work together is much more difficult.

Lastly, a report will more than likely include written explanations of the data presented. It can also be accompanied by summaries and even recommendations for the future of the business. A dashboard will presume a higher level of understanding of the subject matter by the reader and will not include much explanation, if any at all.

1.3.3. How they are the same?

Similarities are relatively few in number. The first, both a report and a dashboard present information. The second, mostly revolves around the content that they have. While charts and tables can appear on both, tables are less likely to appear on dashboards. This isn't to say that tabular information isn't important, this is only to stress the idea that the dashboard must be consumable in short order and must contain only pieces of information that work together to answer the central question.

As you can see in the table provided below, certain key aspects of a dashboard might also be involved in the creation of a report but they do not have to. These key aspects must be answered "Yes" in order for it to be a dashboard.



	REPORT	DASHBOARD
Presents Information	*Yes*	*Yes*
Uses Visualizations	*Maybe*	*Yes*
Single Screen View	*Maybe*	*Yes*
All Information Used Together	*Maybe*	*Yes*
Up to the Minute Information	*No*	*Yes*

Figure 187: Use the right size and position

1.4. Design guidelines for dashboards

1.4.1. Be clear about what you’re trying to achieve

The first step to designing any dashboard is to clearly define what you’re trying to achieve. What‘s the purpose of your dashboard? Who’s it for? What do you want them to do differently because of it?

Perhaps you’re trying to focus your team on a specific goal, or show them how they contribute to the bigger picture. Or maybe you want to make sure a particular type of problem gets noticed quicker. These are all good purposes to keep in mind.

1.4.2. Include only the most important content

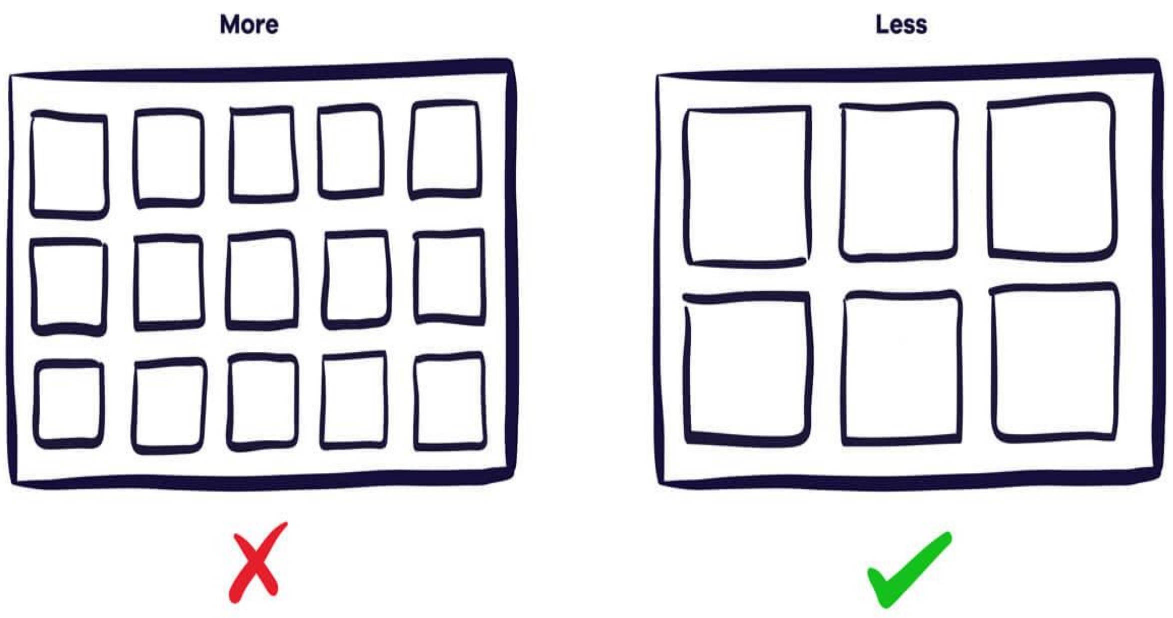


Figure 188: Add alerts as needed

Content is key when it comes to dashboarding. If you’re not showing useful metrics then it doesn’t matter how you arrange them.

Often you’ll already have some goals and KPIs defined, and adding those is a great starting point. Just remember, everything should tie back to the purpose of your board.

Every inch on your TV dashboard is valuable real-estate. Adding too much information can detract from what’s important and make everything harder to find. If you’re really struggling to fit everything in then you may need more than one dashboard.

When putting any metric on your dashboard you should make sure they:

- Match the purpose of your board
- Can be influenced by your team
- Can be easily understood
- Change reasonably often (you don't want to be staring at numbers that never change)
- Don't vary so much that you can't easily detect trends

1.4.3. Use size and position to show hierarchy

Dashboards need hierarchy to be easy to scan. Use size and position to give emphasize the most important information and to downplay metrics that need to be looked at less frequently. Consistent sizes and clear relationships between elements will help create patterns and visual flow.

In terms of positioning, the top left corner of your dashboard is the best location as that's where your eyes are naturally drawn to first.

Don't be afraid of empty space. It's better to leave a gap than to make something bigger just to fill it.

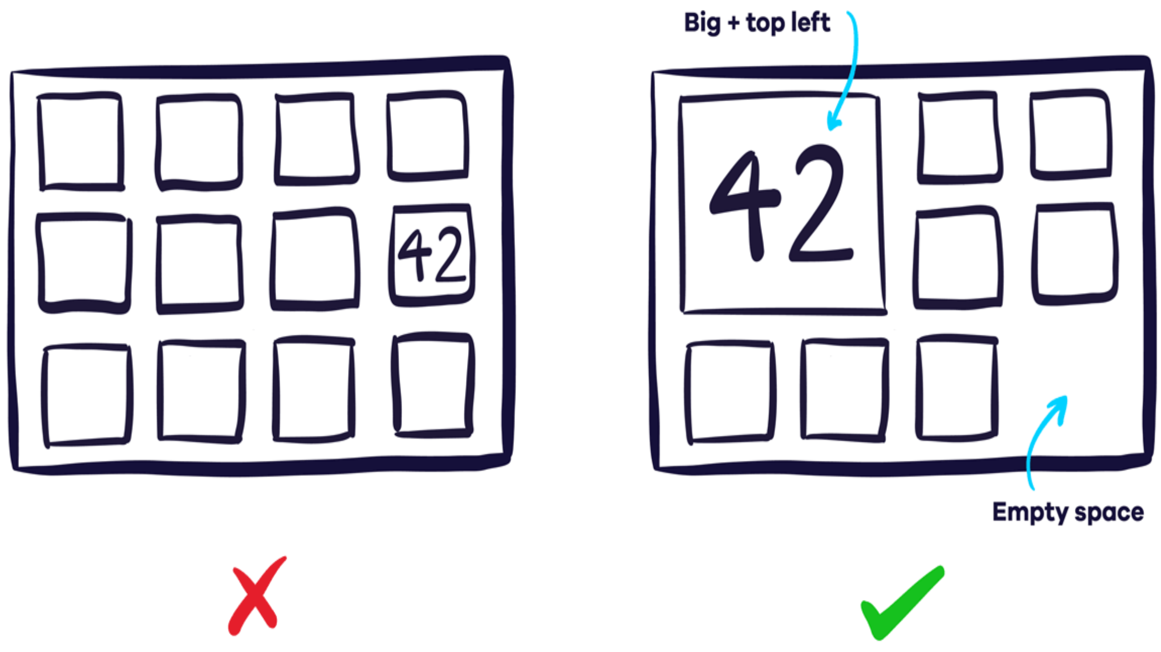


Figure 189: Group related metrics together

1.4.4.Give your number context

To know if a number’s good or bad your viewers need context. Would they know, for instance, that 42 new leads today is out of the ordinary?

One of the easiest ways to do this is to include past data. You could include the same metric for the previous day, or even a line or column chart showing how the metric tracks over a longer period of time. Another technique is to include the average or previous highs and lows.

If you’re working towards a goal, include the target as well as your current progress.

You can also add warnings for when a metric is above or below a certain threshold to make it easier to spot problems.

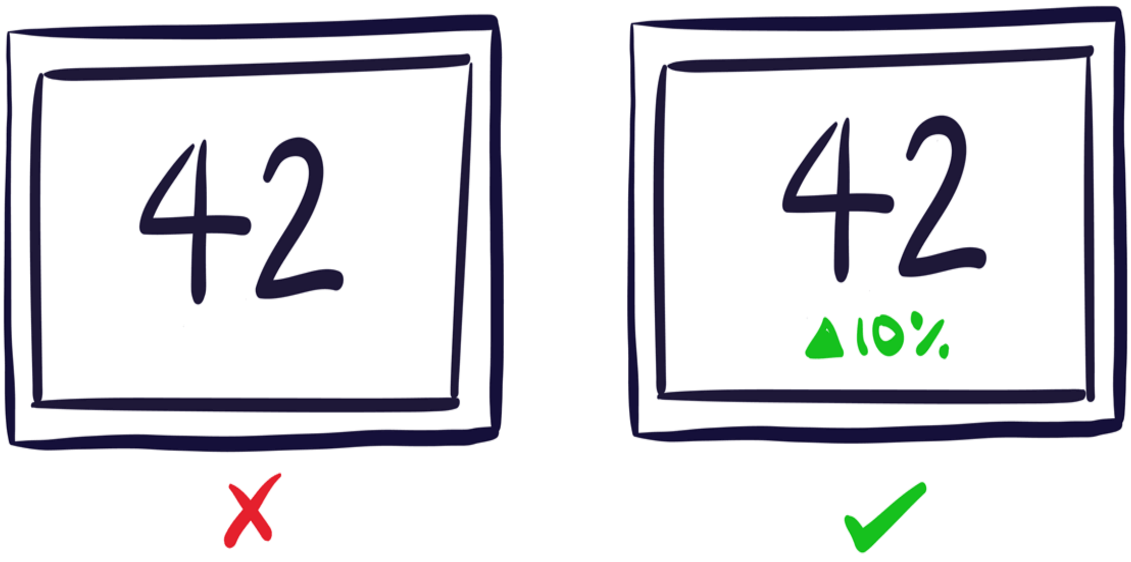


Figure 190: Dashboard - easier to read

1.4.5. Group your related metric

Positioning the information on your dashboard logically is essential. Grouping related metrics next to each other makes them easy to find — and makes your dashboard's design more attractive.

There are many different ways to group e.g. by metric, product, brand, campaign, region, team or even time period. You may need to experiment with which is most appropriate for your board.

Giving groups a title makes them easier to spot.

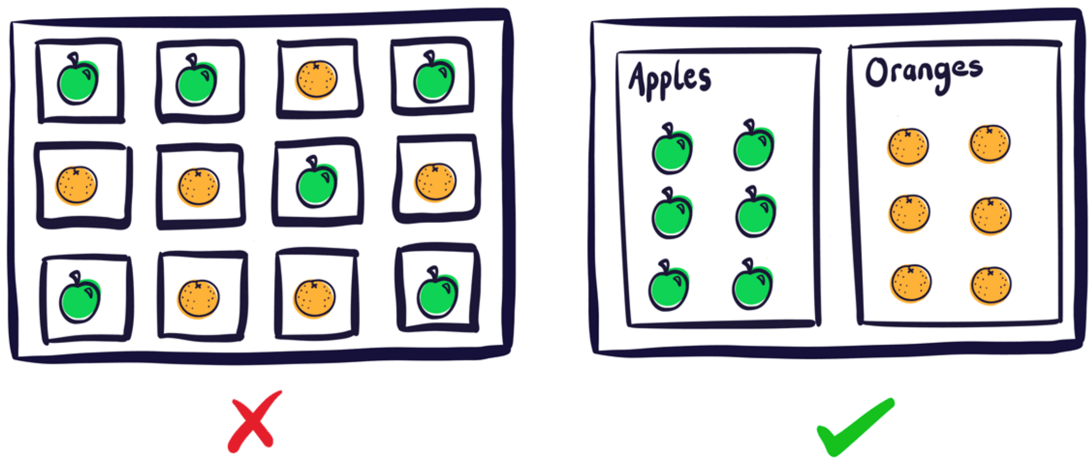


Figure 191: *Dashboard brief*

1.4.6. Be consistent

With many dashboards you'll find there's an element of repetition, for example you might be showing the same set of metrics for multiple things. Your dashboard will be far easier to read if you use the same visualizations and layouts between groups. It will also look far more pleasing, so avoid the temptation to use a line chart instead of a column just to spice things up.

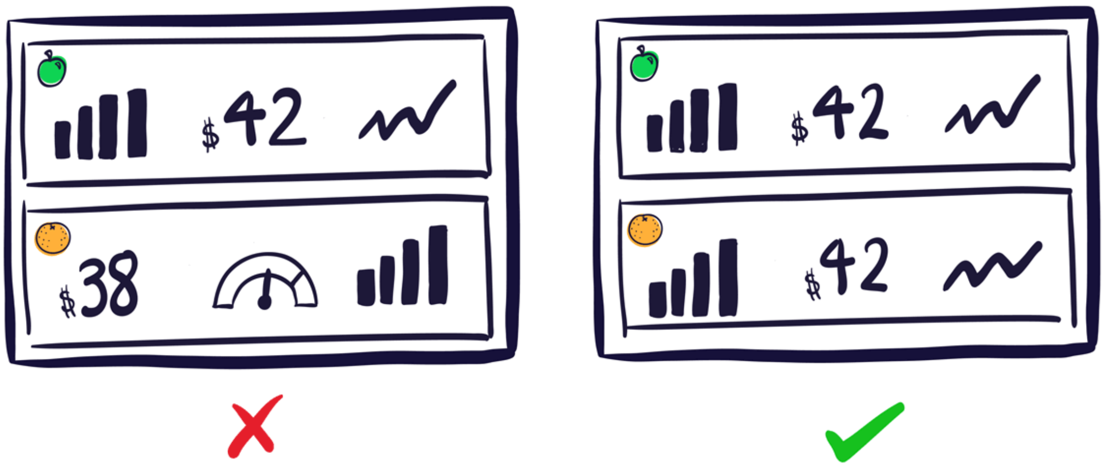


Figure 192: Operational Dashboard Example

1.4.7. Use clear labels your audience will understand

A key part of your dashboard are the labels that describe each metric or chart. They should be self explanatory, and unambiguous for your viewers At the same time, you should try and keep them as short as possible to avoid cluttering up your board and getting in the way of the data.

Abbreviations can be helpful too (as long as your audience understand them) e.g. “7d” instead of “7 days”. Symbols like ‘%’ can replace the word. You may also get away with a shorter definition for a metric if people are already familiar with it.

Headings can also be used to reduce repetition. Imagine you have the same metric for different time frames e.g. signups today, signups this month etc. If they’re all grouped under a heading called “Signups” you don’t need to repeat it each time.

1.4.8. Round your number

When displaying numbers, don’t include more precision than you need. Showing your conversion rate to 3 decimal places or your revenue to the nearest cent when you only care about much bigger changes just distracts from what’s important. Plus, including too much detail can make a mountain out of a molehill.

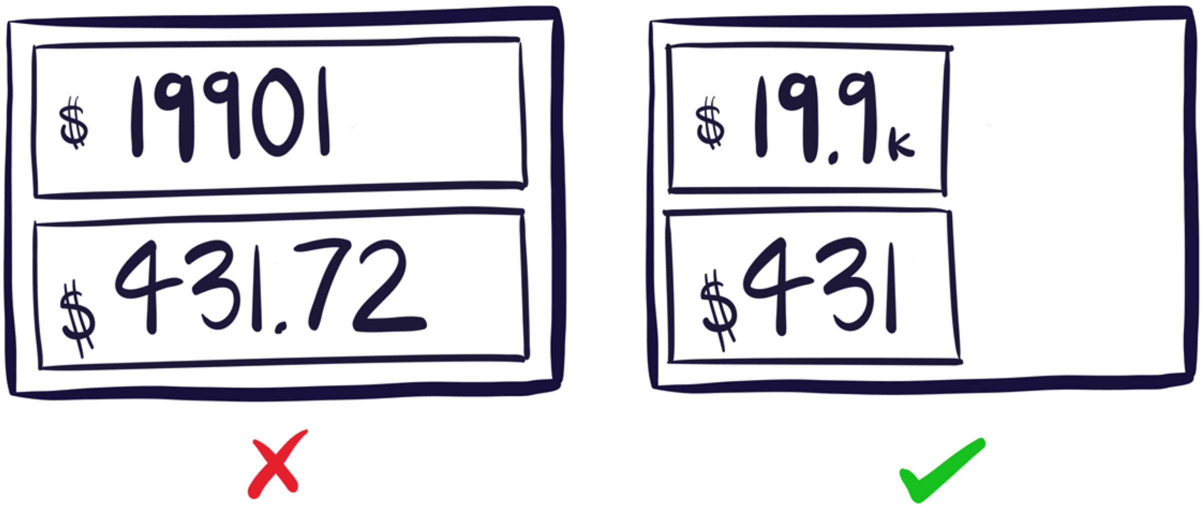


Figure 193: Analytical Dashboard Example

1.4.9. Keep evolving your dashboards

Our final piece of dashboard design advice is the most important. Once you've built your dashboard don't just leave it. Ask you team for feedback.

- What do they look at most often or find most useful, and why?
- What do they never look at or find least useful, and why?
- Is there anything missing that they'd find useful?
- Has it changed anything about the way they work?

Use this feedback to iterate your dashboard. Check your dashboard is driving the behaviour you intended. Step back from your board every now and then and look at how all the elements work together. Remind yourself what information you're primarily trying to get across and how effectively those important elements stand out.

As your goals and priorities change, make sure you update your board so it acts as the heartbeat for whatever you're doing.

Illustration 7: Data analysis with dashboard

1.5. Dashboard limitations

1.5.1. Lack of real-time anomaly detection prevents proactive incident management

Most BI dashboards do not show data in real-time, and when they do, there are so many metrics cluttering screens that users can easily miss the most critical information. Timely intervention is crucial to modern businesses, which often run tightly integrated ecosystems of applications and infrastructure that stretch across multiple departments and process enormous amounts of data.

For example, leading adtech platform, Rubicon Project, fields trillions of bid requests per month and needs to Analyse data points from millions of potential sources. In an environment like that, every minute can have significant impact. They found that traditional dashboards failed to deliver the real-time detection and response capability necessary to intervene before anomalies impacted their bottom line.

1.5.2. Over-reliance on historical data

Most companies configure and use traditional dashboards to track KPIs and other critical business metrics to understand how their business and systems perform. One factor often missed by decision-makers is that the data they view in traditional dashboards describes what has already happened and might not be a reliable indicator of what will happen in the future.

Moving from descriptive to predictive modes of thinking requires a deep understanding of the business context and critical thinking, which can be challenging for any person, or even a dedicated team, given the diversity of the data set, new trends, and fluctuating behaviors.

1.5.3. Missing small incidents that have a negative impact

Some incidents are hard to spot, but that doesn't mean they won't significantly impact the business. When undetected, hard-to-spot incidents can accumulate and can end up having the same impact as more prominent issues.
A typical scenario involves incidents affecting only one business component.

These isolated issues can easily get lost in KPIs based on a calculated average of multiple metrics. For example, a server cluster might be displaying a 99.99% average uptime. If one server in that cluster is experiencing an anomalously high amount of downtime, it could remain invisible to the dashboard. A single server is a small data point in a data center with thousands of servers, but it could be vital depending on what that server is running.

1.5.4. Cluttered dashboards and false positives

Sometimes, even with all the necessary information, BI dashboards struggle to present a coherent picture. With CEO dashboards, in particular, there's some guesswork in determining ahead of time what information is important enough to display in the limited real estate available on the screen. When alerts start to pop up, it can be difficult to tell which data is necessary or worth ignoring. The sheer volume and increasing complexity of data can quickly overwhelm the dashboard interface, making it much harder for business leaders to consume in a timely, accurate manner.

1.5.5. Lack of intelligent prioritization

Collecting thousands of events or alerts every minute from your applications and infrastructure, and presenting that data in a dashboard isn't analytics. Users apply filters on this data, performing their own analysis and work.
Deriving intelligence from data shouldn't require an end user to define what to look for, or where, or what are the most critical KPIs, or what normal or abnormal is. This is not intelligence because a user is telling the dashboard exactly what data to show.

1.5.6. Leverage the Power of AI Analytics

Business strategy is only effective if empowered with enough intelligence and agility to outmaneuver the competition. Traditional dashboards don't provide insight fast enough in today's data-driven world, and when a business can lose hundreds of thousands of dollars in a single hour due to a pricing glitch on an e-Commerce site, the stakes are too high.

Companies need real-time, actionable insights across all data metrics relevant to performance. The best-performing businesses leverage BI solutions empowered by AI and machine learning to eliminate the need for human correlation across the millions of critical metrics needed to understand business and system performance.

Grouping and correlating multiple anomalies by design, Anodot's AI-powered analytics elevates the essential insights first. By learning the normal behavior of millions of metric's, Anodot detects only the most impactful incidents and alerts relevant teams at the start.

1.6. Explanation of the procedure for creating a dashboard

Step 1: Straight to the point

A dashboard is one single screen made up with the most important indicators. Therefore, a dashboard should not show all the data points. It must only cover essential information. It should be a selection of the **most relevant indicators** of your business.

When looking at your dashboard, the first thing you should see is a quick presentation of the most important data related to your activity.

Excel tabs or other secondary menus created with other data visualization tools will allow you to study your data more in-depth.

Step 2: Addressing a problem related to your activity

You must start by identifying your operational needs, then understanding your audience. You have to ask yourself: which data best addresses the specific questions of interest? As organizations and professional fields are very diverse, **the way the dashboard is configured** will have to accommodate their different needs and backgrounds.

Create a screen or a tab for each data query. Too often dashboards are full of irrelevant data. The problem with these dashboards is that they do not portray a clear message. Your user doesn't need to see all of the data which are available. They only need to see the information that they can use for their tasks. Therefore it is your job to organize the data for them.

In order to do this, you must know what the board and your colleagues expect to see.

Useful tip: a huge amount of public data is available on Office for National Statistics' website and on other public databases. With this data, you can give your audience the big picture.

Step 3: Creating action points

Building a dashboard is a process that evolves constantly. Your projects will evolve, and so will your company. An effective dashboard is one that you can revise later if necessary. Meet with colleagues regularly and review the objectives to make sure the selection of indicators still meets their needs.

Each indicator must represent an action for its users. To give an example, a declining number of visitors for a website can alert the marketing department to take immediate action. The data must generate an action; the data can signal an anomaly. This is one important criterion that needs to be built into a dashboard for it to be effective.

Step 4: Defining performance indicators
The wrong choice of indicators is a curse for the organization. Each indicator must serve the company's global strategy. Furthermore, it should be clear what actions are required if there is a sudden change in the data.
If you choose the wrong indicator for your dashboard, you risk that your team loses focus and takes the wrong decisions.

For an indicator to be on your dashboard, **it needs to be comprehensible and adapted** to all the people that will have access to it. In addition, it must be reliable and comparable contextually.
For example, it wouldn't make sense to give the company's annual revenue without first comparing it to the previous years' revenues or to another company's revenues.
It is necessary to keep this logic with respect to the audience and for all indicators.

Step 5: Creating a mobile dashboard
How does your audience use information? Most users like to use their computers to look at reports. However, some users might have a more mobile profile, like managers and salesmen. These users are always on the go. Therefore it is important for them to be able **to access the dashboard freely on a mobile device.**

Since dashboards are interactive tools, mobile compatibility allows the user to interact with the data at his fingertips. You can touch your creations to show more detail, navigate through the main menu into different subcategories. Most importantly, you can give a live demo when making a presentation to clients.

Step 6: Making your data more familiar
If your audience is knowledgeable about the data that you are presenting, they will be more receptive to the message you are trying to relay.

It is your job to find the best examples that relate to your audience. Try new things, show not only your data but also why you are showing it. Give context and illustrate your data with stories and experiences.

Step 7: Making it easy to read
Are you tired of not understanding someone else’s dashboard? Don’t make the same mistakes.

The easier the data is to read, the quicker it will be understood. Decision making is also affected by the clearness of your dashboard. If it is clear to the reader, he can take instantaneous decisions on the outcome. Something to keep in mind is that the user isn’t always familiar with the data format. Therefore it is very important **to give accessible information to the users.**

A graph must convey a single message. Often, dashboards present dozens of indicators and graphs which result in the loss of the users’ interest. Do not overwhelm the audience by presenting graph after graph of information. Present only one idea or visualization per screen. You can use many slides if you have a lot of things to the present.

Step 8: Plotting the data
To captivate your audience, narrate a story.
Data is used to add truth to what you are saying. It shouldn’t be the center of your arguments. To tell a story, you **must contextualize your data**. Write a story that evolves around your data.

Work on making your dashboard more colourful. Your comments and titles should be easy to read. This allows your audience to follow and anticipate the data that is being presented.

Step 9: Understanding your audience
When you are preparing your data analysis report, it is essential to answer the right questions. To do this you must put yourself in the shoes of your audience.

You have to ask yourself, ‘**Do the reporting tools that I am using correspond to the message I want to convey?** Is the graph I am using intuitive enough for my audience? Is this indicator comprehensive enough?

Step 10: Presenting it smoothly
Fear of data and statistics is often an obstacle for the audience in understanding a report. You can overcome this by visualizing the data. At times, the audience needs you to guide them. The situation is to demonstrate and use data storytelling.

1.7. Security aspects for dashboards

A good security dashboard needs to include the following for a specified/measured time period: An indication of current threat level to the organization; an indication of events and incidents that have occurred; a record of authentication errors; an indication of scans, probes and unauthorized access, and an indicator if those key measures are up, down or unchanged; brute force attacks against the system and non-compliant devices; policy violations; malware events; and phishing events.

Security dashboard checklist:

- Current threat level to the organization.
- Events and incidents that have occurred.
- Authentication errors.
- Scans, probes and unauthorized access.
- Brute force attacks against the system and non-compliant devices.
- Policy violations.
- Malware events.
- Phishing events.
- Detailed technical metrics specific to the controls that are employed to manage risks to existing and emergent threat vectors.
- Number of covered assets.
- Newly discovered assets.
- Decommissioned assets.
- Number of threats detected and their risk levels.
- Clear visibility into the risk landscape.
- Areas of training/awareness.
- Vulnerability management.
- Third-party risk management.
- Incident management.
- Overall risk management.
- Mean time to patch.
- Mean time to detect and respond to potential incidents.
- Average window of exposure.
- Number of exceptions/types of exceptions.
- Phish fail percentage.
- Impact of remedial training.

1.8. Installation of necessary libraries

We will start by installing an explainer dashboard using pip. The command given below will do that.

```
pip install explainerdashboard
```

- **Importing required libraries**

In this step, we will import the required libraries and functions to create a machine learning model and dashboard.

```
from sklearn.ensemble import RandomForestClassifier
from explainerdashboard import ClassifierExplainer, ExplainerDashboard
from explainerdashboard.datasets import titanic_survive, titanic_names
```

- **Creating the Model & Dashboard**

This is the final step in which we will create the machine learning model and then interpret that model by creating a dashboard.

+ Creating the Model:

```
feature_descriptions = {
   "Sex": "Gender of passenger",
   "Gender": "Gender of passenger",
   "Deck": "The deck the passenger had their cabin on",
   "PassengerClass": "The class of the ticket: 1st, 2nd or 3rd class",
   "Fare": "The amount of money people paid",
   "Embarked": "the port where the passenger boarded the Titanic. Either
Southampton, Cherbourg or Queenstown",
   "Age": "Age of the passenger",
   "No_of_siblings_plus_spouses_on_board": "The sum of the number of siblings
plus the number of spouses on board",
   "No_of_parents_plus_children_on_board" : "The sum of the number of parents
plus the number of children on board",
}X_train, y_train, X_test, y_test = titanic_survive()
train_names, test_names = titanic_names()
model = RandomForestClassifier(n_estimators=50, max_depth=5)
model.fit(X_train, y_train).
```

- **Creating the Dashboard:**

```
from sklearn.ensemble import RandomForestClassifier
from explainerdashboard import ClassifierExplainer, ExplainerDashboard
from explainerdashboard.datasets import titanic_survive,
titanic_namesfeature_descriptions = {
    "Sex": "Gender of passenger",
    "Gender": "Gender of passenger",
    "Deck": "The deck the passenger had their cabin on",
    "PassengerClass": "The class of the ticket: 1st, 2nd or 3rd class",
    "Fare": "The amount of money people paid",
    "Embarked": "the port where the passenger boarded the Titanic. Either
Southampton, Cherbourg or Queenstown",
    "Age": "Age of the passenger",
    "No_of_siblings_plus_spouses_on_board": "The sum of the number of siblings
plus the number of spouses on board",
    "No_of_parents_plus_children_on_board" : "The sum of the number of parents
plus the number of children on board",
}X_train, y_train, X_test, y_test = titanic_survive()
train_names, test_names = titanic_names()
model = RandomForestClassifier(n_estimators=50, max_depth=5)
model.fit(X_train, y_train)explainer = ClassifierExplainer(model, X_test, y_test,
                                cats=['Deck', 'Embarked',
                                    {'Gender': ['Sex_male', 'Sex_female', 'Sex_nan']}],
                                cats_notencoded={'Embarked': 'Stowaway'},
                                descriptions=feature_descriptions,
                                labels=['Not survived', 'Survived'],
                                idxs = test_names,
                                index_name = "Passenger",
                                target = "Survival",
                                )db = ExplainerDashboard(explainer,
                        title="Titanic Explainer",
                        shap_interaction=False,
                        )
db.run(port=8050)
```

1.9. Explanation of the necessary programming concepts, functions and classes to display dashboards

Object-oriented programming is an approach to problem solving where all computations are carried out using objects. An object is a component of a programme that knows how to perform certain actions and how to interact with other elements of the programme. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You could also expect a person to be able to do something, such as walking or driving. This would be considered a method of the person.

Code in object-oriented programming is organized around objects. Once you have your objects, they can interact with each other to make something happen. Let's say you want to have a programme where a person gets into a car and drives it from A to B. You would start by describing the objects, such as a person and car. That includes methods: a person knows how to drive a car, and a car knows what it is like to be driven. Once you have your objects, you bring them together so the person can get into the car and drive.

A **class** is a blueprint of an object. You can think of a class as a concept, and the object is the embodiment of that concept. You need to have a class before you can create an object. So, let's say you want to use a person in your programme. You want to be able to describe the person and have the person do something. A class called 'person' would provide a blueprint for what a person looks like and what a person can do. To actually use a person in your programme, you need to create an object. You use the person class to create an object of the type 'person.' Now you can describe this person and have it do something.

Classes are very useful in programming. Consider the example of where you don't want to use just one person but 100 people. Rather than describing each one in detail from scratch, you can use the same person class to create 100 objects of the type 'person.' You still have to give each one a name and other properties, but the basic structure of what a person looks like is the same.

A function is a combination of instructions that are combined to achieve some result. A function typically requires some input (called arguments) and returns some results. For example, consider the example of driving a car. To determine the mileage, you need to perform a calculation using the distance driven and the amount of fuel used. You could write a function to do this calculation. The arguments going into the function would be distance and fuel consumption, and the result would be mileage. Anytime you want to determine the mileage, you simply call the function to perform the calculation.

Illustration 8: Object-Oriented Programming

1.10. Clarification of the necessary web-based programming components for dashboard programming

A typical dashboard contains three elements:

- Heading explaining the content of the dashboard and its purpose
- Diagram visualizing the metrics
- Short explanation of the status and information in the diagram

In designing the pages of the dashboard the principles of cognitive perception abilities should be taken into account, such as:

1. Elements of the dashboard should be logically and conceptually related to each other
2. The number of elements in the dashboard (diagrams, text fields, explanations, buttons) should be no more than 7 (+2 if necessary) as this is the number of elements an average person can keep in the short term memory.
3. The use of colours should be limited to the minimum and the colours should extrapolate the diagrams and the important information in the dashboard.

A number of technologies and framework exists which can support the development of a dashboard, for example:

- Dashing.io (open source): http://dashing.io/ - a ready-to-use dashboard software based on XML file links to the web server. The framework is simple to set up, but limited in its graphical abilities. It also requires a backbone processor of data as it cannot process the data itself.
- The dash (free): https://www.thedash.com/ - an alternative to dashing.io, with similar requirements on backbone processor scripts, but more flexible in terms of available visualizations (e.g. diagrams). Software Center metrics project 8
- Google dashboard (free):
- https://developers.google.com/appsscript/articles/charts_dashboard - a set of simple-to-set-up javascript and SVG based charts which can be customized very easily. The main advantage is that it is simple and easy to use but it also requires backbone processing of the data.
- D3 (Data Driven Documents, open source): http://d3js.org/ - a more flexible (powerful and expressive) alternative to Google charts/dashboard.

- Tibco Spotfire:
- http://spotfire.tibco.com/products/spotfiredesktop?gclid=CjwKEAjwkK6wBRCcoK_tiOTzFASJAC7RArijfNQV5JgnHYXKOVyhwDlfgKdTj0b3ei4xyJBqn6VqhoCLO3w_wcB – a business intelligence tool which allows to easily create drill-down reports and dashboards. The main advantage is that once the data is in a database the tool has a graphical way of creating the charts (no programming needed as in the previous techniques); the main disadvantage is that it is commercial and that setting up the database and importing the data requires programming and more effort than in the case of the scripts for the previous techniques.
- Tableu: http://www.tableau.com/ and http://www.tableau.com/learn/whitepapers/5- best-practices-for-effective-dashboards - an alternative to Spotfire.
- Qlikview: http://www.qlik.com – another alternative to Spotfire

Illustration 9: Future artificial intelligence robot and cyborg

2. Possiblities for data visualization

2.1. Fundamentals of data visualization

2.1.1. Display of different diagram types with application areas

Operational Dashboards: Picture a "traditional" dashboard. Are you seeing metrics, updating in real-time, showing performance data related to the operations of the day? If so, you're envisioning an **operational dashboard**, arguably the most common dashboard type. These are the dashboards well-suited to a wall display on a manufacturing plant floor, or a command suite of global operations.

An operational dashboard is designed to provide, at a glance, a comprehensive snapshot of the performance of the day. Much like the dashboard on a car, operational dashboards give the viewer information related to the immediate performance of the organization. They shouldn't require drill downs to be useful, because often the viewer won't have the option to manipulate the dashboard past the initial view.

This means that the operational dashboard has to have a fairly detailed view. In turn, it's important to make sure, when planning an operational performance dashboard, that the scope does not become too wide. If you try to accomplish too much with one dashboard, it can become unclear and ultimately unused. Keeping the end user in mind will assist in this process. Are you communicating metrics to an assembly line, or an executive? Make sure to focus group your end users so you know exactly what they need to see to perform their job functions effectively.

Analytical Dashboards : If you are using data from the past to identify trends that can help you make decisions about the future, you are on your way to creating an analytical dashboard. These dashboards are tools that the user should be able to interact with, inquire of, and explore. As such, features like pivot tables and drilldowns are well-suited to analytical dashboards.

Comparing and contrasting data across multiple variables is a crucial aspect of data analytics. A user must be able to compare data across time, to see if performance differences correlate with corporate action (or if outside forces, such as seasonality, have measurable effects on metrics). Slicing and dicing the data in a structured way allows the user to determine what efforts have worked.

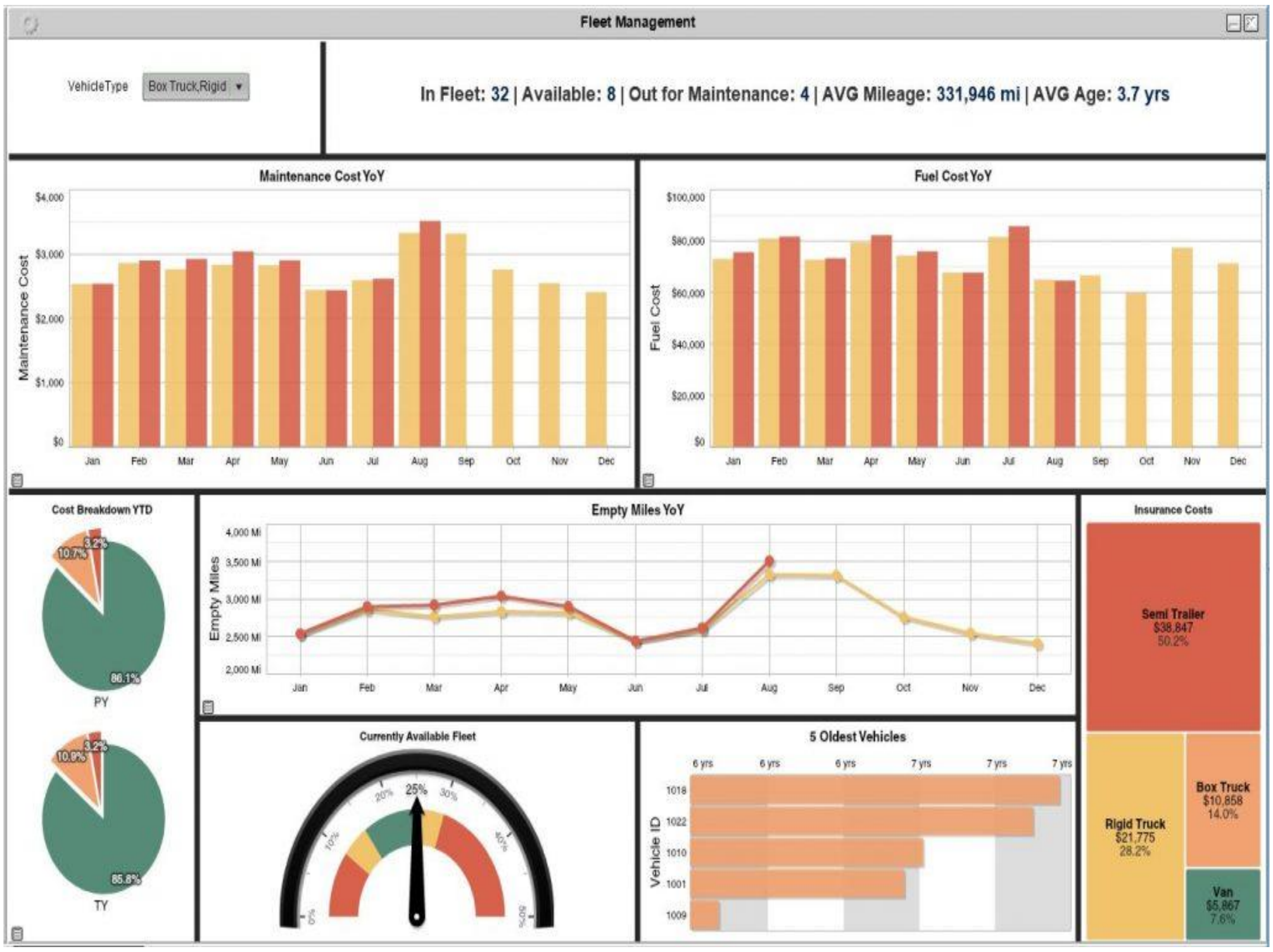


Figure 194: Strategic Dashboard Example

An analytical dashboard can be a valuable tool in the right hands, but they require a level of understanding that the average business user may not possess. The data in an analytical dashboard is typically complex, as are the analytical exercises the dashboard is suited towards. As such, analytical dashboards are best left to your database analysts as opposed to the whole company. Defining user permissions is a simple way to ensure that your analytical dashboards are being served to the right group.

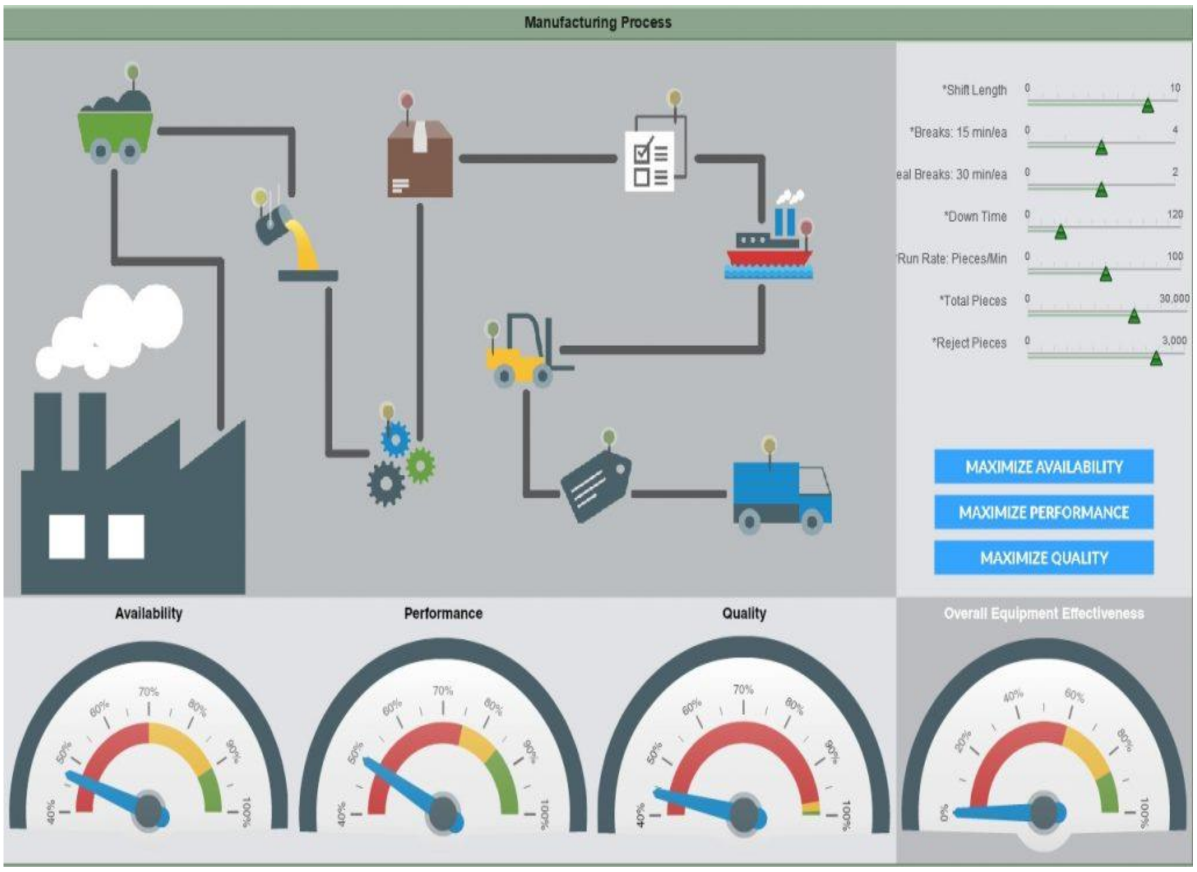


Figure 195: Analytical Dashboard Example

Strategic Dashboards: We spend a lot of time talking about KPIs, because the evidence is clear that setting goal and aiming towards them is the surest path to success. If you've defined key performance indicators and are tracking performance in relation to those KPIs, odds are you've got yourself a **strategic dashboard**. These dashboards are often used to align departmental performance with overall corporate strategy.

Typically, strategic dashboards have a retrospective flair. They look at benchmark performance data from, say, last quarter, and compare it to the current period. Have things improved, stayed the same, or worsened? They're also often composed of data from multiple sources, as company-wide goals are affected by multiple systems and actions.

Strategic dashboards often share metrics that are important to the whole organization, so consider having them available to the whole organization. It's obvious that management and the executive team would want a birds-eye-view of strategic KPIs, but when this sort of performance data is transparently shared with lower level employees, there can be unexpected benefits. You never know where the next great idea in your organization will come from, and when you empower the whole team with knowledge, that becomes especially true.

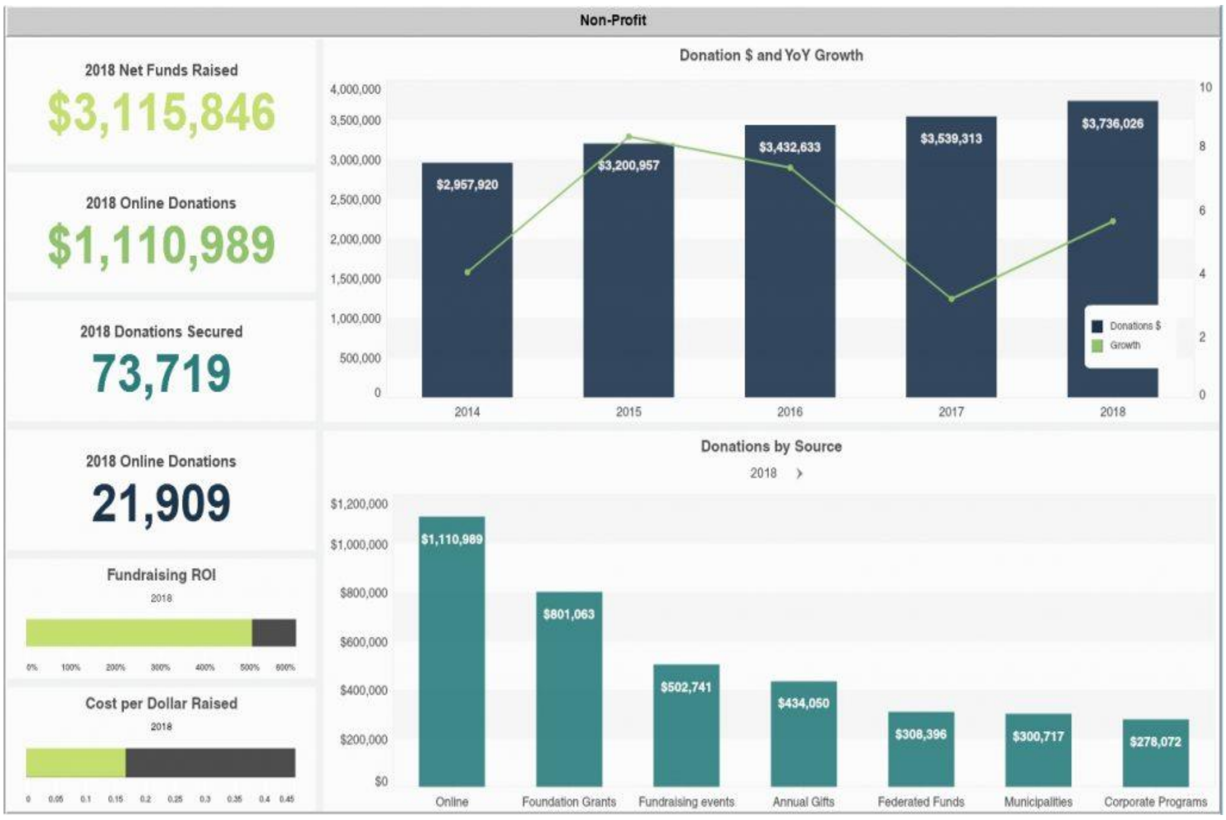


Figure 196: Strategic Dashboard Example

2.1.2. Definition of the necessary data structures to use diagram type:
We can't stress enough the importance of choosing the right data visualization types. You can destroy all of your efforts with a missing or incorrect chart type. It's important to understand what type of information you want to convey and choose a data visualization that is suited to the task.

Dashboard-centric charts and visualizations fall into four primary categories that are related to the aim of the visualization: relationship, distribution, composition, and comparison. It is important to understand the aim of the metric before picking the chart type that you want. Here we will talk about a few of the most common types and their aims:

Line charts are great when it comes to displaying patterns of change across a continuum. They are compact, clear, and precise. Line charts format is common and familiar to most people so they can easily be Analysed at a glance.

Choose bar charts if you want to quickly compare items in the same category, for example, page views by country. Again such charts are easy to understand, clear, and compact.

Pie charts aren’t the perfect choice. They rank low in precision because users find it difficult to accurately compare the sizes of the pie slices. Although such charts can be instantly scanned and users will notice the biggest slice immediately, there can be a problem in terms of scale resulting in the smallest slices being so small that they even cannot be displayed. A good practice when using pie charts is to only do it with a couple of slices, this way, you make sure that the information is easy to understand and will bring value to your dashboard.

Sparklines usually don’t have a scale which means that users will not be able to notice individual values. However, they work well when you have a lot of metrics and you want to show only the trends. They are rapidly scannable and very compact.

It’s also not that easy to decipher scatterplots as they are an advanced type of visualization for more knowledgeable users. They aim to find the correlation between two variables. When the data is distributed on the chart, the results show the correlation to be positive, negative, or nonexistent.

Gauge charts are valuable visualizations to provide context. The advantage of these charts lays in the fact that they are easy to interpret as they use various colours to represent different values of the same metric. They are usually used in situations where the expected value is already known, this way the different stakeholders that use the dashboard can understand where they stand just by looking at the gauge chart. For example, to monitor the sales target or sales growth.
Most experts agree that bubble charts are not fit for dashboards. They require too much mental effort from their users even when it comes to reading simple information in a context. Due to their lack of precision and clarity, they are not very common and users are not familiar with them. in order)

-There are three types of dashboards: operational, strategic, and analytical.
+ Metrics you can track on an operational dashboard
- Website performance metrics like new users or bounce rate
- Follower count or comments across your social media channels

+ Metrics you can track on an analytical dashboard
•Annual contract value to track the dollar amount an average customer contract is worth
•Measure your company spending habits with the Bessemer Efficiency Score

•Understand the increase in daily active users over a period time
- Return on ad spend to track the effectiveness of your digital advertising dollars

+ Metrics you can track on a strategic dashboard
- Monthly, quarterly, or yearly fiscal performance
- Account and MRR growth rate
- Earnings before interest, tax, depreciation, and amortization (otherwise known as EBITDA)

2.2. Placement of diagrams in dashboards

Dashboard best practices in design concern more than just good metrics and well-thought-out charts. The next step is the placement of charts on a dashboard. If your dashboard is visually organized, users will easily find the information they need. Poor layout forces users to think more before they grasp the point, and nobody likes to look for data in a jungle of charts and numbers. The general rule is that the key information should be displayed first – at the top of the screen, upper left-hand corner. There is some scientific wisdom behind this placement – most cultures read their written language from left to right and top to bottom, which means that people intuitively look at the upper-left part of a page first, no matter if you're developing an enterprise dashboard design or a smaller-scaled within the department - the rule is the same.

Another useful dashboard layout principle is to start with the big picture. The major trend should be visible at a glance. After this revealing first overview, you can proceed with more detailed charts. Remember to group the charts by theme with the comparable metrics placed next to each other. This way, users don't have to change their mental gears while looking at the dashboard by, for example, jumping from sales data to marketing data, and then again to sales data. This analytics dashboard best practice will enable you to present your data in the most meaningful way and clear to the end-user.

2.3. Conversion of data from data source to data sink in the diagram

2.3.1. Data preparation, pre-filtering, source selection

- Data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data.

Data preparation is often a lengthy undertaking for data professionals or business users, but it is essential as a prerequisite to put data in context in order to turn it into insights and eliminate bias resulting from poor data quality.

- 76% of data scientists say that data preparation is the worst part of their job, but the efficient, accurate business decisions can only be made with clean data.

Data preparation helps:

- **Fix errors quickly** — Data preparation helps catch errors before processing. After data has been removed from its original source, these errors become more difficult to understand and correct.
- **Produce top-quality data** — Cleaning and reformatting datasets ensures that all data used in analysis will be high quality.
- **Make better business decisions** — Higher quality data that can be processed and Analysed more quickly and efficiently leads to more timely, efficient and high-quality business decisions.

Additionally, as data and data processes move to the cloud, data preparation moves with it for even greater benefits, such as:

- **Superior scalability** — Cloud data preparation can grow at the pace of the business. Enterprise don't have to worry about the underlying infrastructure or try to anticipate their evolutions.
- **Future proof** — Cloud data preparation upgrades automatically so that new capabilities or problem fixes can be turned on as soon as they are released. This allows organizations to stay ahead of the innovation curve without delays and added costs.
- **Accelerated data usage and collaboration** — Doing data prep in the cloud means it is always on, doesn't require any technical installation, and lets teams collaborate on the work for faster results.

Additionally, a good, cloud-native data preparation tool will offer other benefits (like an intuitive and simple to use GUI) for easier and more efficient preparation.

- Data Preparation Steps

The specifics of the data preparation process vary by industry, organization and need, but the framework remains largely the same.

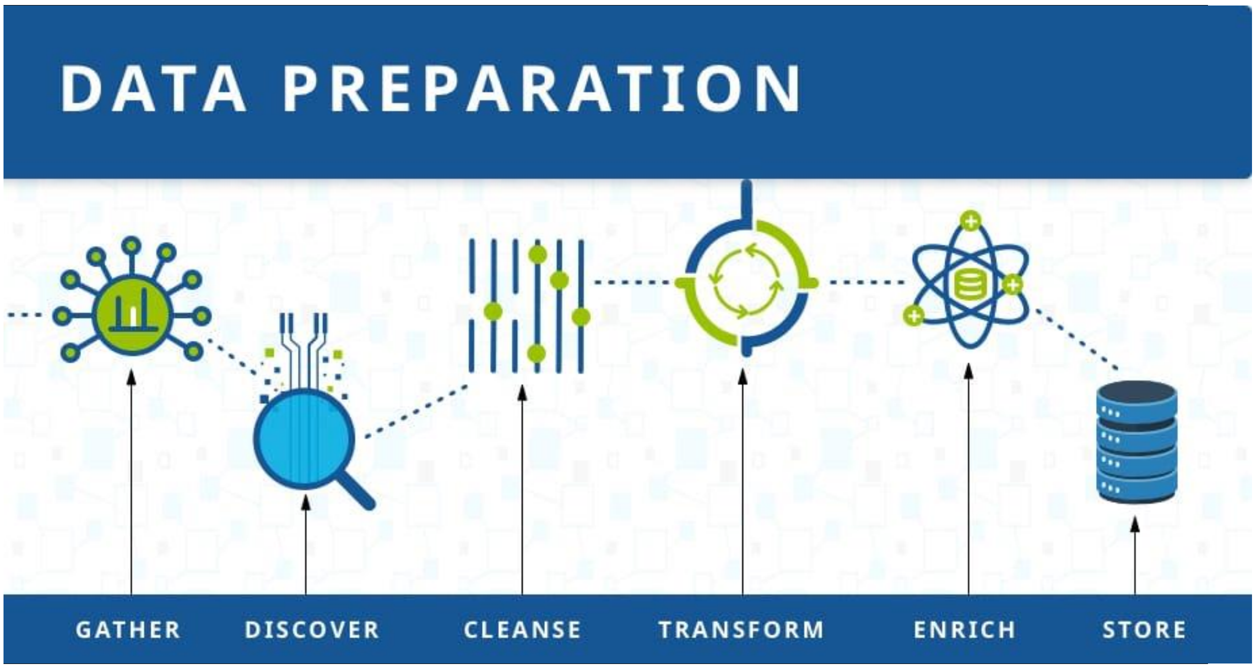


Figure 197: Illustration of Data preparation

i. Gather data

The data preparation process begins with finding the right data. This can come from an existing data catalog or can be added ad-hoc.

ii. Discover and assess data

After collecting the data, it is important to discover each dataset. This step is about getting to know the data and understanding what has to be done before the data becomes useful in a particular context.

Discovery is a big task, but Talend's data preparation platform offers visualization tools which help users profile and browse their data.

iii. Cleanse and validate data

Cleaning up the data is traditionally the most time consuming part of the data preparation process, but it's crucial for removing faulty data and filling in gaps. Important tasks here include:

Removing extraneous data and outliers.

Filling in missing values.

Conforming data to a standardized pattern.

Masking private or sensitive data entries.

Once data has been cleansed, it must be validated by testing for errors in the data preparation process up to this point. Often times, an error in the system will become apparent during this step and will need to be resolved before moving forward.

iv. Transform and enrich data

Transforming data is the process of updating the format or value entries in order to reach a well-defined outcome, or to make the data more easily understood by a wider audience. *Enriching* data refers to adding and connecting data with other related information to provide deeper insights.

v. Store data

Once prepared, the data can be stored or channeled into a third party application—such as a business intelligence tool—clearing the way for processing and analysis to take place.

3. Create and deploy dashboards

3.1. Implementation of a suitable programme logic to create a dashboard

3.1.1. Implementation of static dashboards without data update

Static dashboards are reporting tools used to summarize information into digestible graphical forms that offer at-a-glance visibility into business performance. The value resides in their capability to illustrate progressive performance improvements via fitting visual features to users. Depending upon the purpose and context, data updates can occur once a month, week, day, or even in real-time . Static dashboards can fulfill both the urgency of fast-paced environments, offering real-time data support, and tracking of performance metrics against enterprise-wide strategic objectives based on historical data. However, such dashboards do not involve the user in the data visualization process and have issues with handling complex and multidimensional data . Interactive dashboards can be considered a step toward directly involving the user in the analysis process. Interactive dashboards consider not only visual features, but also introduce functional features such as point and click interactivity . These capabilities allow operational decision-makers to enable more elaborate analyses . Addressing the impediments of static dashboards, they are used to establish a better understanding of the complex nature of data, which can also foster decision-making. However, the benefits of interactivity could increase the users' cognitive effort and required manual analysis time, increasing the probability of delayed decisions or (even) errors.

3.1.2. Implementation of dynamic dashboards with update interval

- A dynamic dashboard is a type of data dashboard that updates automatically in real-time. You might also hear them referred to as interactive dashboards, since the reports can be changed, reorganized, and manipulated. That's different from a static dashboard, which only displays a fixed set of data.

Most of the time when we use the word ‘dashboards’, what we’re really talking about is dynamic dashboards. Below, let’s take a look at a few ways you might use dynamic dashboards to explore your EHS data.

i. Analyse data in real-time

As we said before, dynamic dashboards update automatically to show your data as it is collected. Information captured across your company on mobile devices and through automated systems integrations shows up in your dashboards almost immediately.

If you use a continuous emissions monitoring system, for example, you can view emissions data as it is collected. Using interactive dashboards, you can Analyse trends, drill down on details by location, measure actuals versus permit limits, and monitor emissions performance related to production throughput. That means you can get insights as soon as information is entered into your system and react without delay.

ii. Drill down for deeper insights

Unlike static charts and graphs, dynamic dashboards allow users to interact with their data. You can drill down — or see more specific data — on a particular element or KPI until you find the level of detail you require.

Let’s say you want to see more detailed information about your CO2 emissions. By clicking on that particular report, you can access additional layers of data. In that way, dynamic dashboards enable you to dive deeper into your data without cluttering the main data view.

iii. Drag-and-drop to build custom data views

Another advantage of dynamic dashboards is that you can control the information that is displayed. You can drag-and-drop to add or remove reports from your dashboard. You can also change the date range, chart type, size, and placement of reports to draw people’s attention to the most important metrics. In that way, dynamic dashboards allow you to visualize your data exactly how you want to.

iv. Tailor dashboards to specific users

With dynamic dashboards, you can publish different dashboards for different groups of users. So you could build an executive dashboard for senior leaders. Or, you could create a dashboard specifically for plant managers. This solves the problem of having to create a different dashboard for each individual, or forcing hundreds of employees to share a single static dashboard view.

Dynamic dashboards also allow you to control access to sensitive information. A plant manager should only see data for the facility they oversee, while an executive should see data for the entire company. With dynamic dashboards, you can set permissions so that users will only see data that is relevant to their role and that they are allowed to access.

3.2. Consideration of user authentication options for security aspects

3.2.1. User name, password:

The dashboard user interface is graphically divided into several *blocks*: the *utility menu*, the *main menu*, and the main *content pane*.

The top right part of the screen contains a utility menu. It includes general tasks related more to the dashboard than to the Ceph cluster. By clicking its items, you can access the following topics:

- Change the language of the dashboard's user interface. You can choose from Czech, English, German, Spanish, French, Portuguese, Chinese, or Indonesian.
- Display a list of Ceph related tasks that are running in the background.
- View and erase recent dashboard notifications.
- Display a list of links that refer to the information about the dashboard, its complete documentation, and an overview of its REST API.
- Manage the dashboard's users and user roles. Refer to Chapter 14, *Managing Users and Roles on the Command Line* for more detailed command line descriptions.
- Log out of the dashboard.

3.3. Creating a visualization programme

3.3.1. *Creating a measurement programme to store sensor data in a database*

- In this project both environmental data as well as the amount of produced goods shall be measured. Object awareness can be measured via different types of sensors. One possibility is the usage of a ultrasonic sensor. These types of sensors are used to measure distances. Their benefit is that they can be used for detection on most surfaces. Depending on the area of application it can be a disadvantage that their detection radius is cone shaped, which can lead to crosstalk phenomena. Further a infrared (IR) sensor will be used. Their benefit is that they have a point shaped detection mechanism. Their disadvantage is that certain surfaces and materials do not reflect the IR signals which causes the sensors to not be able to detect the object. For environmental data different types of sensors are available. You have too choose two types of sensors. Write the sensor type according to the datasheet in the table below.

Sensor1	Sensor2	Sensor3	Sensor4

The bosh XDK is a robust microcontroller platform with a wide sensor array. It is commonly used in industrial applications especially for data tracking. The bosh XDK delivers a wide sensor array, a integrated wifi module, a sd card reader to store data on a memory card, freely configurable buttons and a usb connector to connect the XDK to a PC for flashing new programmes and directly outputting values in the console of its own IDE.

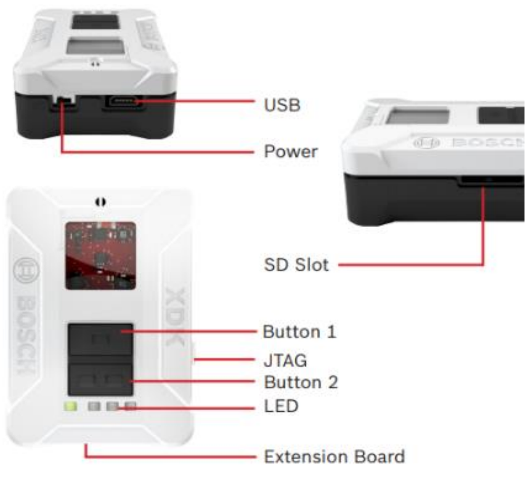


Figure 198: XDK sensor structure

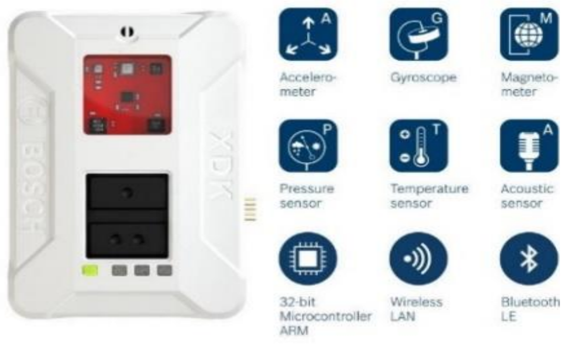


Figure 199: XDK sensor function

The bosh XDK further permits its user to access many types of sensors. Users can measure acceleration, temperature, humidity, air pressure and many more. Further the XDK uses “FreeRTOS” as realtime operating system. This ensures that precicly timed measurements with defined intervals are possible.

Further bosh uses a own eclipse based IDE called XDK-Workbench. This IDE gives the user the choice to either programme the XDK directly with C, or to use MITA programming language. MITA programming language was developed to lessen the burden of creating I4.0 applications without knowledge of embedded programming. The website: https://developer.bosch.com/web/xdk/getting-started#1 is provided by bosh and has sample code for reading in sensor values. Part of this task is to read through the descriptions and sample codes and reuse them to create a working programme.
The task of this section is to create a measurement programme that reads in the sensor values. Atleast two sensors of the XDK have to be used in this exercise. You can choose which ever sensors you want to.
The read in values shall be outputted on the console of the XDK – Workbench. The project shall be programmed in MITA programming language. Measurements shall be done every 5 seconds. The desired console output shall be as follows:
“TYPE OF SENSOR”: XXX”
“TYPE OF SENSOR”: XXX”
Enter a screenshot of the console output in the table below. Upload your code and a screenshot of the console output.

Console output:

After the measurement and communication systems got set up the relational database system can be started. As a first step a ERM diagram shall be created. For the ERM diagram it is necessary to model all parts of the system. The Database shall include the stations name, the sensor names, sensor types, the sensor values, the microcontroller type and the time stamp when the measurement was taken. All names have to be in english. Draw the ERM diagram below:

The ERM diagram is the basis of every database. After it was created Create the database itself. The database name shall be either “Bottling Station” or “CSPi4.0” depending on which station you work. All used names shall match the ERM diagram. All tables have to be in 3rd normal form, if necessary normalize the tables. Upload and include a picture of all created tables and rows. Also upload the picture.

Table

3.3.2. Implementing a dashboard to continuously read and graphically display measured values from the database

After the database is set up and the measurement programme works it is possible to store the measured data inside the database.

Writing to database

Whenever the programme receives a value over MQTT all received sensor values plus the current time stamp of reception shall be saved into the database. Ensure that the right tables and values are send to the database. Further ensure that the timestamp has the right format to fit with your used database. Below a small list of necessary SQL commands is presented. It is also necessary to provide a library to send SQL commands to your database.

Test your setup by creating a programme that reads in values for 5 minutes and save the measured values to the database. Before starting your measurement, programme ensure the stations are running. Upload the code.

SQL commands:

Inserting values into a table

INSERT INTO *tableName*

VALUES (*value1*, *value2*, *value3*, ...);

Reading row values from table

SELECT colName FROM tableName

Reading from database

After completion load all stored sensor values from the database. Create a visualization of the sensor values. Put every sensor value into a own diagram. As tile choose the sensor name. Insert the diagrams below. Upload the code and a screenshot of the diagrams.

3.3.3. Generation of a limit value mechanism to detect exceeding of a limit value

- After the test programmes were successfully completed the sensors can be placed a desirable location. For the bottling station the sensors have to be placed in a way that the ultrasonic sensor measures if a bottle is present from the side. The IR sensor shall be used to measure if a cap was put on the bottle.

Find a valid location for the sensors to fullfill this requirement. On the CPSi 4.0 station the same procedure shall be conducted. Instead of bottles the position of the blocks shall be figured out. Make sure to adjust the IR sensor so it can detect if the block has the hole on the upside or not. Find a valid location for the sensor placement and document your decision with a picture. Further describe in your own words why you choose this location in the table below:

Picture	Documentation

3.3.4. Generation of an Alerting Mechanism to give warnings when limit values are exceeded

The XDK offers the possibility to transmit data via publisher and subscriber method using the MQTT protocol. Since the XDK offers a great variety of sensors multiple sensors shall be transmitted to the computer. In this example all measured data shall be converted to a JSON object. This object shall be passed over MQTT. Therefore, there will only be one topic to send the data. The sensor data shall be named as follows:

- For acceleration
  - AccelX
  - AccelY
  - AccelZ
- For humidity
  - Humidity
- For temperature
  - Temperature
- For Pressure
  - Pressure

Steps:

1. Setup a broker on the computer and run the broker
2. Use your measurement programme for the microcontroller to read in the sensor values. Let the microcontroller read in with a interval of 3 seconds
3. Include the necessary libraries on the microcontroller and create the topic to be published to.
   a. Topic name
      i. MeasurementXDK/ States
4. Include the necessary libraries on the computer and subscribe to the topic where the sensor values are published at
5. Print the published values on the computers console
6. Console output

After the deployment of the web app was successfully implemented the dashboard application shall be created. The dashboard shall visualize the measured sensor values in the database.

Static dashboard

In the first attempt a static dashboard shall be created. The following steps shall be implemented:

1. Connect to the database
2. Load the stored values of all sensors and their corresponding
3. Create a dashboard
    a. For every sensor create one diagram
    b. Set the title of every diagram to the sensor name
    c. Set the y axis to be the sensor value
    d. Set the x axis to be the timestamp of the corresponding sensor value
4. Set the background to be in gray colour
5. For one of the sensors also add a gauge
6. Before testing let the measurement run for 2 minutes

Include a picture of your dashboard below. Upload your code and your dashboard picture.

Dynamic dashboard

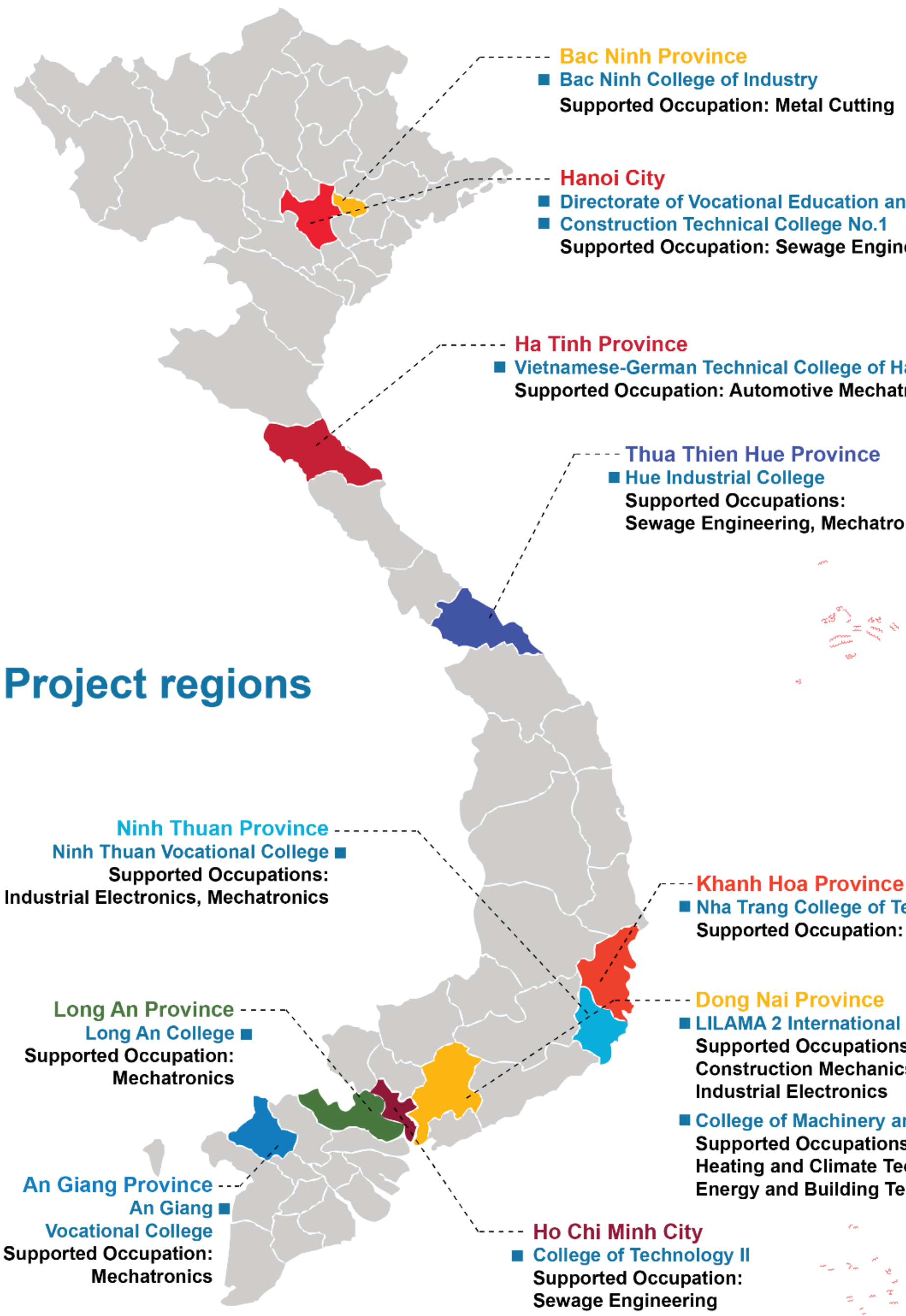
In this extension the dashboard shall be updated every 10 seconds. To achieve this task use a interval object with 10 seconds interval. Create a callback function which is called with the defined interval. In the callback function read in the updated database values and update your visualization automatically.

Steps:

1. Connect to database
2. Define interval object
3. Create a dashboard
    a. Create the base structure of the dashboard
    b. Set the background to be in gray colour
    c. Create callback function
        i. Load the stored values of all sensors and their corresponding
        ii. For every sensor create one diagram
        iii. Set the title of every diagram to the sensor name
        iv. Set the y axis to be the sensor value
        v. Set the x axis to be the timestamp of the corresponding sensor value
        vi. Update the plot
4. Start the measurement
5. Deploy the visualization

APPENDIX

1. PROJECT REGIONS AND PARTNERING TVET INSTITUTES



Project regions

Bac Ninh Province
- Bac Ninh College of Industry

Supported Occupation: Metal Cutting

Hanoi City
- Directorate of Vocational Education and Training (DVET)
- Construction Technical College No.1

Supported Occupation: Sewage Engineering

Ha Tinh Province
- Vietnamese-German Technical College of Ha Tinh

Supported Occupation: Automotive Mechatronics

Thua Thien Hue Province
- Hue Industrial College

Supported Occupations: Sewage Engineering, Mechatronics

Ninh Thuan Province
- Ninh Thuan Vocational College

Supported Occupations: Industrial Electronics, Mechatronics

Khanh Hoa Province
- Nha Trang College of Technology

Supported Occupation: Automotive Mechatronics

Long An Province
- Long An College

Supported Occupation: Mechatronics

Dong Nai Province
- LILAMA 2 International Technology College

Supported Occupations: Mechatronics, Construction Mechanics, Metal Cutting CNC, Industrial Electronics

- College of Machinery and Irrigation

Supported Occupations: Mechanics for Sanitary, Heating and Climate Technology, Electronics for Energy and Building Technology

An Giang Province
- An Giang Vocational College

Supported Occupation: Mechatronics

Ho Chi Minh City
- College of Technology II

Supported Occupation: Sewage Engineering

APPENDIX

2. LIST OF CHARTS

Chart 1: Procedural programming 29
Chart 2: Condition function 31
Chart 3: Condition programming path 31
Chart 4: Scan cycle of PLC 35
Chart 5: Cycle and real time 36
Chart 6: Class Possibilities 50
Chart 7: Example of inheriting properties of the parent class 60
Chart 8: Process of software testing 76
Chart 9: Types of software testing 78
Chart 10: Operating cycle on PLC 105
Chart 11: I/O structure of the microcontroller 118
Chart 12: Function block diagram of microcontroller 119
Chart 13: Memory distribution in microcontrollers 120

3. LIST OF TABLES

Table 1: Comparison of procedural programming vs. object-oriented programming 47
Table 2: Comparison of Class and Object 48
Table 3: Comparison of Black Box, White Box and Grey Box Testing 91
Table 4: Control signal board 106
Table 5: Database 190
Table 6: Join function 194
Table 7: Artwork database template 197
Table 8: Explanation for Figure 171 204

4. LIST OF ILLUSTRATION

Illustration 1: Realistic microchip processor 8
Illustration 2: Electronic board components 19
Illustration 3: Data visualizer graphic 21
Illustration 4: Electronic control board element 71
Illustration 5: CPU processor lighting circuits 158
Illustration 6: Cloud data storage and secure data storage cloud server 203
Illustration 7: Data analysis with dashboard 236
Illustration 8: Object-Oriented Programming 246
Illustration 9: Future artificial intelligence robot and cyborg 248

APPENDIX

5. LIST OF FIGURES

Figure 1: Windows system command interface 39
Figure 2: Microsoft windows command window screen 40
Figure 3: Demonstration between “Class” and “Object” 51
Figure 4: OOP Example 51
(Source: https://www.clipartfree.de)
Figure 5: Inheritance Python 56
Figure 6: Inheritance – Overriding 56
Figure 7: Logo of pandas 57
(Source:https://www.pandas.pydata.org/pandasdocs/stable/getting_started/intro_tutorials/01_table_oriented.html)
Figure 8: Sample of Pandas Dataframe 57
Figure 9: Sample of Python Dataframe 57
Figure 10: Explanation of parts of figure 59
Figure 11: Graphical user interfaces 60
Figure 12: Inheritance Overview via Tree; node width = NIV, node height = NOM and colour = WLOC 65
Figure 13: ATM machine system illustration 72
Figure 14: Illustration of Verification and Validation Model 75
Figure 15: Different kinds of manual testing...81
Figure 16: Illustration of high cost of software defects 83
(Source: Jones C, 2008, Applied software measurement: global analysis of productivity and quality, New York : McGraw-Hill)
Figure 17: Objectives of software test plan 84
(Source: 5 steps to Set Smart Objectives | GHCC, Atlanta, Georgia, https://ghcc.org/en/5-steps-to-set-smart-objectives-examples/)
Figure 18: Software testing plan...86
(Source: Why Testing is Important in the Software Development Life Cycle - UTOR (u-tor.com), https://u-tor.com/topic/importance-of-testing-in-sdlc)
Figure 19: Test case for software..88

(Source: How to Write Test Cases for Software: Examples & Tutorial (parasoft.com), https://www.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/)

Figure 20: Resource planning in project management 90

(Source: Stafiz - Activity management, resource planning, time tracking and billing, https://stafiz.com/en/)

Figure 21: Life Cycle of automation testing 91

(Source: All You Need To Know About Automation Testing Life Cycle (lambdatest.com), https://www.lambdatest.com/blog/all-you-need-to-know-about-automation-testing-life-cycle/)

Figure 22: Illustration of Microcontroller 94

Figure 23: Basic Structure of a Microcontroller 97

Figure 24: Microcontroller and PLC 101

Figure 25: PLCs Architecture 102

Figure 26: PLC hardware structure 103

Figure 27: Microcontroller's Architecture 104

Figure 28: Ladder logic/diagram...106

Figure 29: Instruction on software download 109

Figure 30: Instruction on balenaEtcher Setup 1 109

Figure 31: Instruction on balenaEtcher Setup 2 110

Figure 32: Instruction on balenaEtcher Setup 3 110

Figure 33: Instruction on balenaEtcher Setup 4 111

Figure 34: Instruction on balenaEtcher Setup 5 111

Figure 35: Instruction on balenaEtcher Setup 6 112

Figure 36: Instruction on Rasperry Pi setup 1 112

Figure 37: Illustration of event driven 113

Figure 38: Comparison of Delay driven vs Timer driven 115

Figure 39: Raspberry Pi4 118

Figure 40: XDK sensor 119

Figure 41: Illustration of Hard vs Soft RTOS 119

Figure 42: Pulse width modulation 126

(Source:https://upload.wikimedia.org/wikipedia/commons/0/03/RGB_farbwuerfel.jpg)

Figure 43: Sensor Set 2.0 für Raspberry Pi 4 Modell B 129

(Source: Book: German-Sensor Set 2.0 für Raspberry Pi 4 Modell B 2020.07.08 - SunFounder)

Figure 44: Fundamentals of state machines 133
Figure 45: Example system state diagram 134
Figure 46: Hardware used to implement the example 137
Figure 47: Webbased communication 143
Figure 48: Client Server Communication 147
Figure 49: Client Server model 148
Figure 50: Working principle of Client Server model 149
Figure 51: Publish/subscribe pattern 156
Figure 52: CPS i4.0 training system (front view) 161
Figure 53: Cubes at start under RFID antenna 162
Figure 54: Magazine cylinder with two reed contacts 163
Figure 55: CPSi4.0 workpiece positions Stations (to fig. 54) 164
Figure 56: Cube detected by capacitive sensor 165
Figure 57: Contour check cylinder with position sensor for base part (above) and cover part (below) 165
Figure 158: Inductive sensor with aluminum cube (above) and light barrier with plastic cube (below) 166
Figure 159: Light conductor stops cube under RFID antenna (above and wrong cube in ejection position (below) 167
Figure 160: Bottling station full setup, cleaning(left), filling (middle) and capping (right) 168
Figure 161: Cleaning station 169
Figure 162: Filling station 170
Figure 163: Capping station 170
Figure 164: Visualise curve 172
Figure 165: Linear interpolate 173
Figure 166: XDK sensor structure 175
Figure 167: XDK sensor function 175
Figure 168: Database security 186
Figure 169: ERM model of a book management system 199
Figure 170: ERM model show all the relationship 201
Figure 171: CPS i4.0 training system (front view) 205
Figure 172: Cubes at start under RFID antenna 206
Figure 173: Magazine cylinder with two reed contacts 207

Figure 174: CPSi4.0 workpiece positions 208
Figure 175: Cube detected by capacitive sensor 209
Figure 176: Contour check cylinder with position sensor for
base part (above) and cover part (below) 210
Figure 177: Inductive sensor with aluminum cube (above) and
light barrier with plastic cube (below) 211
Figure 178: Light conductor stops cube under RFID antenna (above)
and wrong cube in ejection position (below) 212
Figure 179: Bottling station full setup, cleaning(left), filling (middle)
and capping (right) 213
Figure 180: Cleaning station 214
Figure 181: Filling station 215
Figure 182: Capping station 216
Figure 183: Example of Databoads...218
Figure 184: Compare the differences between Reports and Dashboards ...219
Figure 185: Compare the similarity between Reports and Dashboards 227
Figure 186: Show only the most important content 230
Figure 187: Use the right size and position 231
Figure 188: Add alerts as needed 232
Figure 189: Group related metrics together 233
Figure 190: Dashboard - easier to read 234
Figure 191: Dashboard brief...235
Figure 192: Operational Dashboard Example...235
Figure 193: Analytical Dashboard Example 236
Figure 194: Strategic Dashboard Example 251
Figure 195: Analytical Dashboard Example 252
Figure 196: Strategic Dashboard Example 253
Figure 197: Illustration of Data preparation..257
Figure 198: XDK sensor structure 261
Figure 199: XDK sensor function 262

REFERENCES

Data banks Handbook
Informatics Handbook
Mechatronics Handbook and textbook
Mechatronics Textbook
Hồ Viết Bình, Tự động hóa quá trình sản xuất (Automatisierung von Produktionsprozessen), University of Technology and Education, Ho Chi Minh City, Viet Nam
Graphic
https://www.anodot.com
https://www.datapine.com
https://www.electronicshub.org/microcontrollers-basics-structure-applications/
https://www.geeksforgeeks.org
https://www.geckoboard.com
https://www.gupea.ub.gu.se
https://www.heptapod.com
https://www.toucantoco.com
https://www.w3schools.in
Stefan Paschek, Database
Stefan Paschek, Database System
Stefan Paschek, Exercises Database
Stefan Paschek, Microcontroller
Stefan Paschek, Microcontroller Exercises
Stefan Paschek, Microcontroller Project: Collecting Data in an I4.0 Setup

Photo/graphic pages (cover photos & illustrations): Freepik/cong.inwent

The training module was developed in the frame of the Development Partnership with the Private Sector (DPP) Integrating Requirements of Industry 4.0 in TVET. The DPP is implemented jointly by the cooperation partners Bosch Rexroth AG, LILAMA 2 International Technology College and Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH, supported by the develoPPP programme of the German Federal Ministry for Economic Cooperation and Development (BMZ).

Viet Nam, 12/2021