King Saud University

College of Engineering

IE – 462: “Industrial Information Systems”

Fall – 2022 (1^{st} Sem. 1444H)

Chapter 2

Information System Development

Prepared by: Ahmed M. El-Sherbeeny, PhD

Lesson Overview

- System Development Life Cycle (SDLC)
- Programming Languages

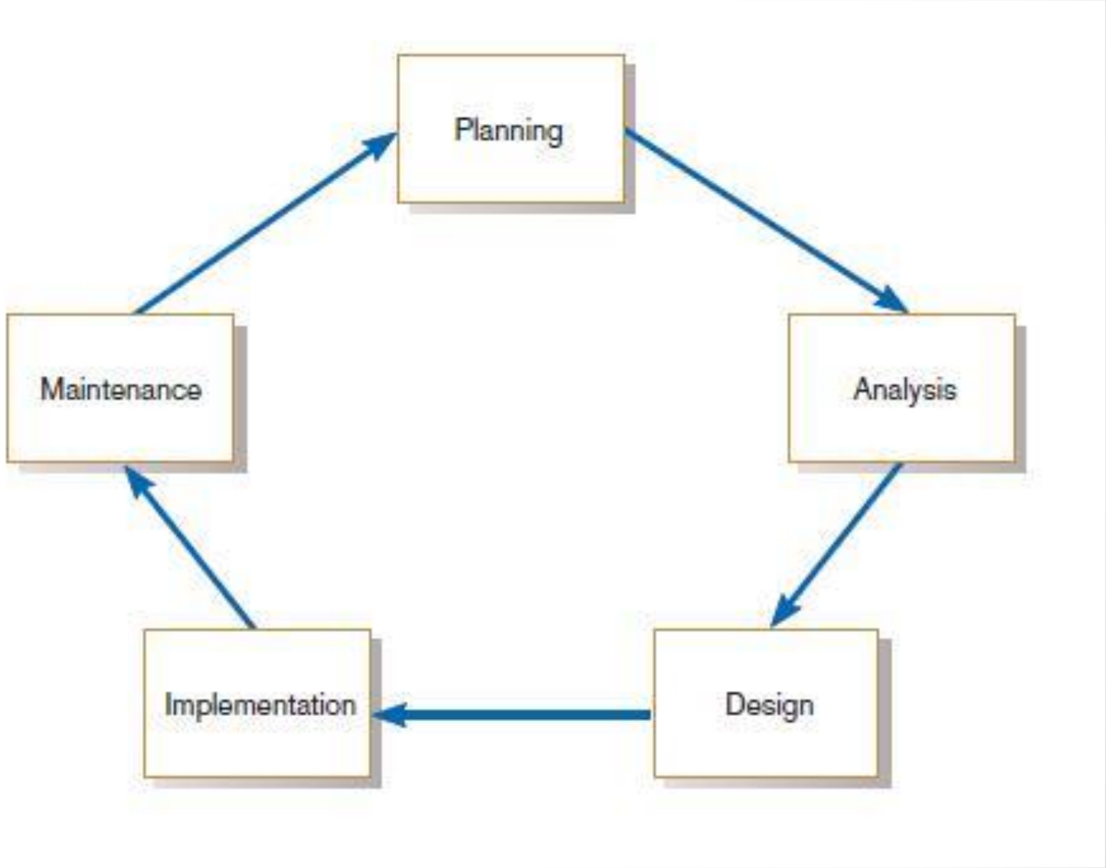
System Development Life Cycle (SDLC)

System Development Life Cycle (SDLC)

- **System Development Life Cycle** (SDLC):
  - traditional methodology/process followed in an organization
  - used to *plan*, *analyze*, *design*, *implement* and *maintain* information systems
  - **System analyst** is responsible for analyzing and designing an information system

SDLC- Cont.

- Phases in SDLC:
  - Planning
  - Analysis
  - Design
  - Implementation
  - Maintenance



SDLC- Cont.

- **Planning** – an organization's total information system objectives or purposes are identified, analyzed, prioritized, and arranged

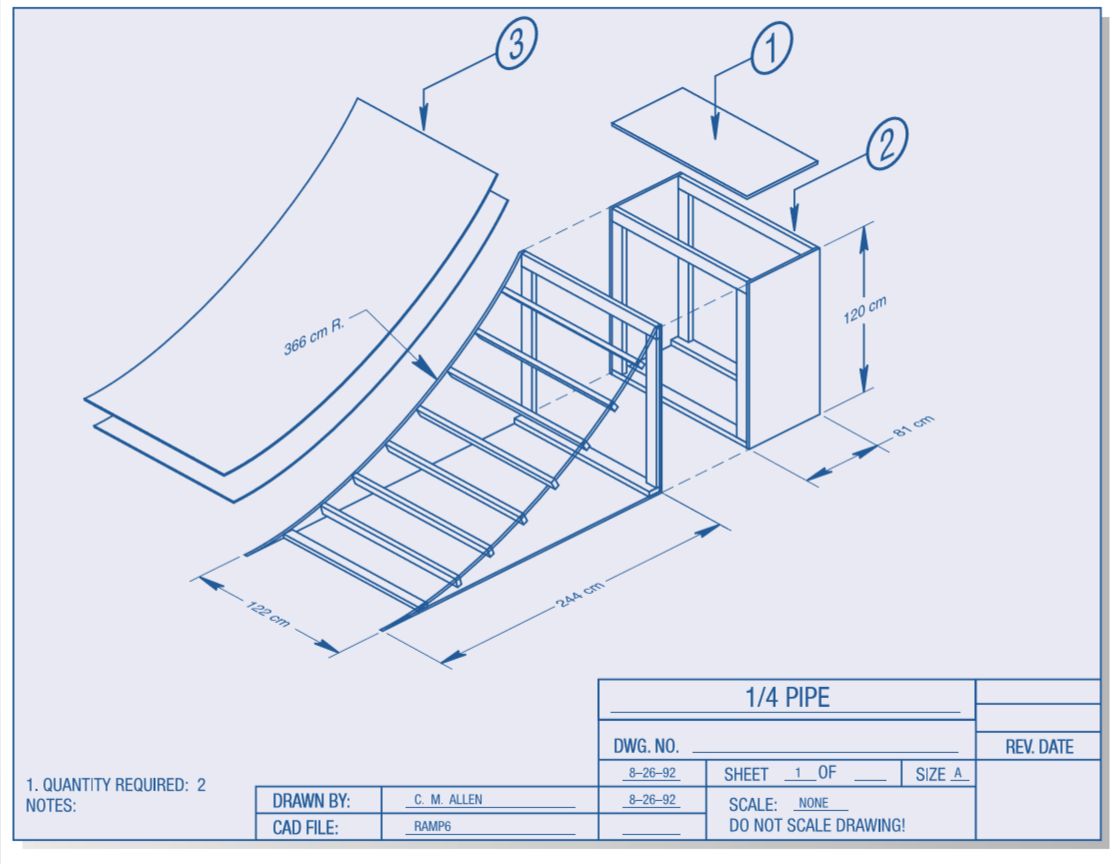
- **Analysis** – system requirements are studied and structured (this's called *system analysis*) Includes feasibility analysis:
  - technical feasibility
  - economic feasibility
  - legal feasibility

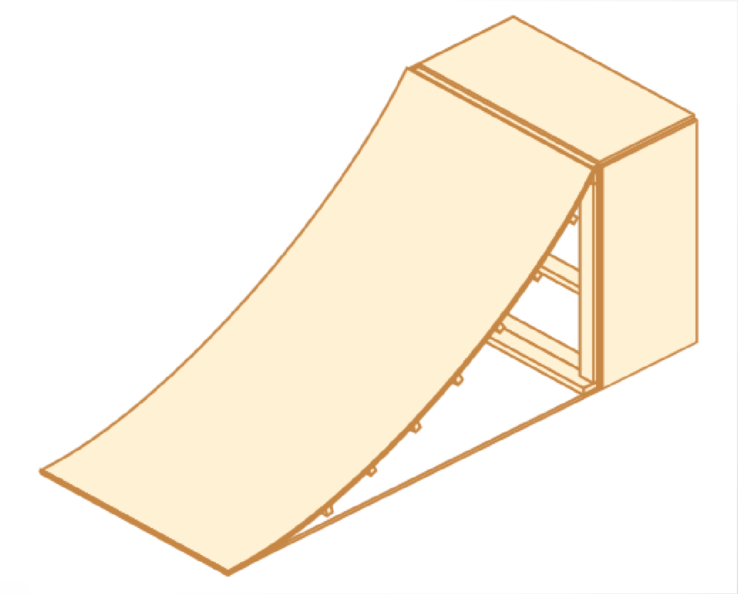
SDLC- Cont.

- **Design** – a description of the recommended solution is converted into *logical* and then *physical* system specifications
  - **Logical design**: all *functional features* of the system chosen for development in analysis are described *independently* of any computer platform
  - **Physical design**: transforming the logical specifications of the system into *technology-specific details*

SDLC- Cont.

- **Design** – cont.
  - See below: difference between physical and logical design





- Skateboard ramp blueprint (logical design) A skateboard ramp (physical design)

SDLC- Cont.

- **Implementation** – information system is:
  - coded (i.e. programmed)
  - tested (includes unit test, system test, user-acceptance test)
  - installed (training users, providing documentation, and conversion from previous system to new system)
- **Maintenance** – information system is systematically repaired and improved
  - structured support process: reported bugs are fixed, requests for new features are evaluated and implemented

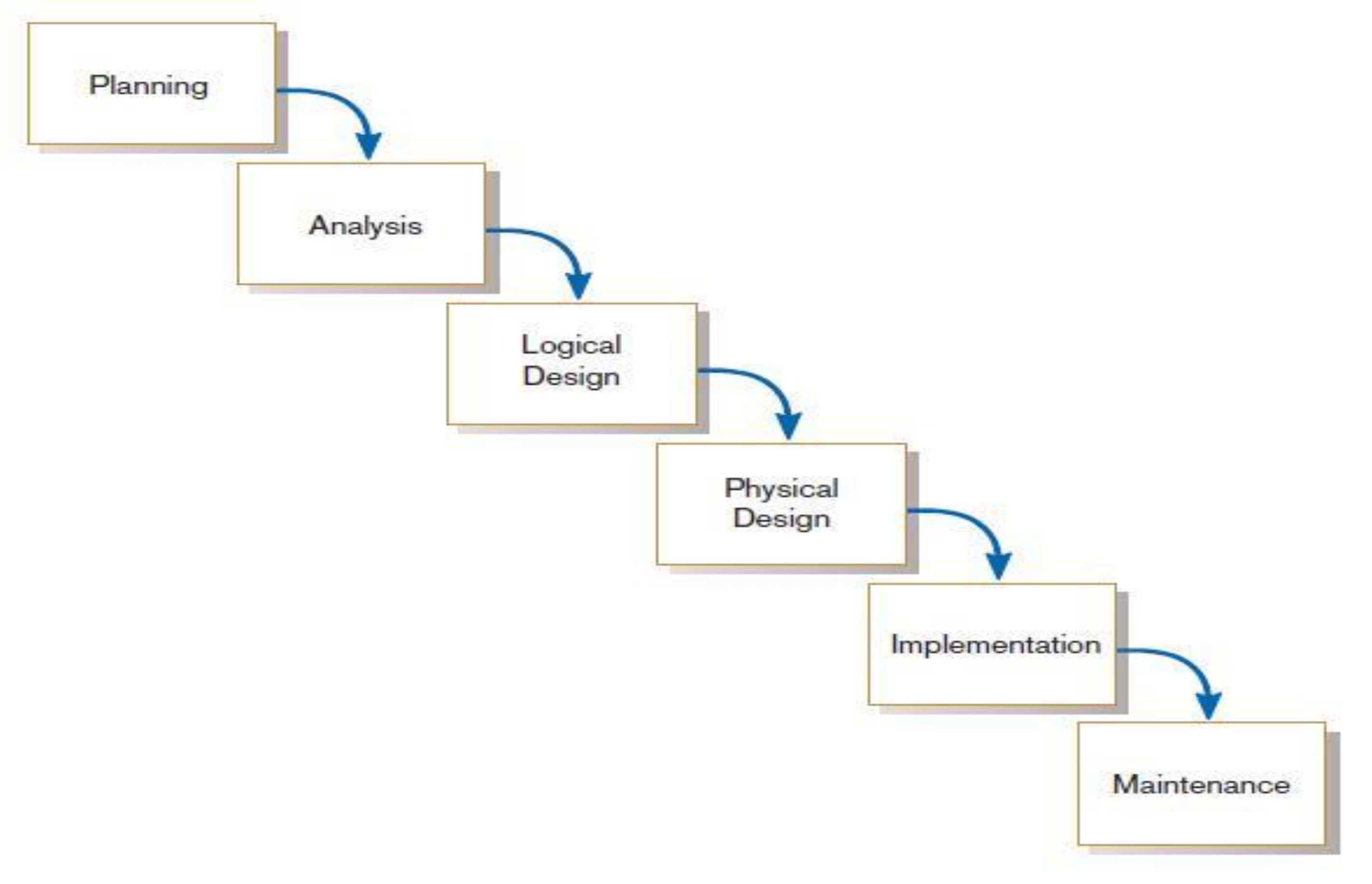
  - system updates/backups are performed on a regular basis

Types of SDLCs

- SDLC can be performed in several different ways:
  - **Traditional Waterfall SDLC**
  - **Iterative SDLC**
  - **Rapid Application Development (RAD)**
  - **Agile Methodologies**
  - **Lean Methodology**

SDLC Types: 1. Traditional Waterfall SDLC

- One phase begins when another completes, with little backtracking and looping

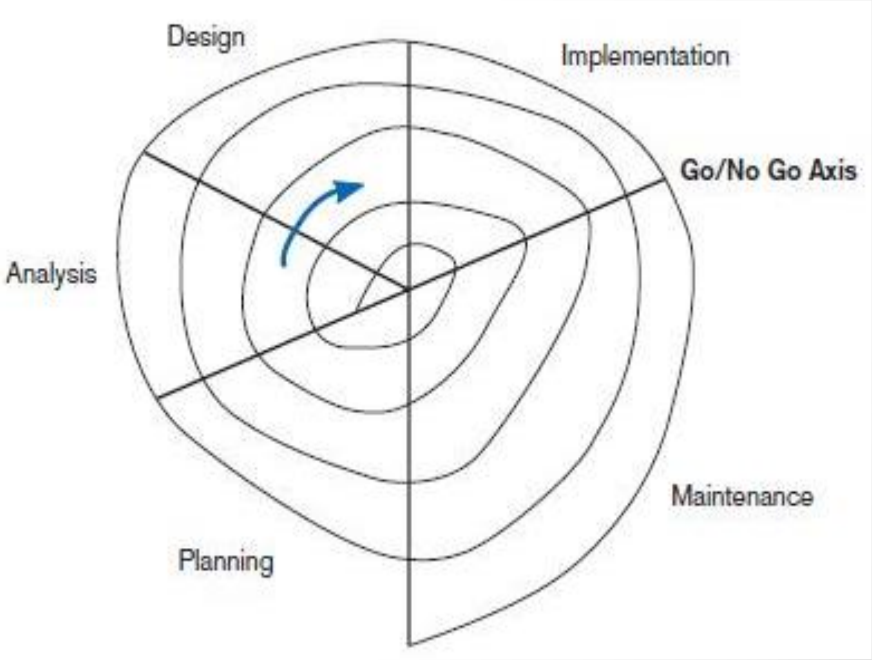


Problems with Waterfall Approach

- Quite rigid: system requirements can't change after being determined
- No software is available until after the programming phase
- Limited user cooperation (only in requirements phase)
- Projects can sometimes take months/years to complete

SDLC Types: 2. Iterative SDLC

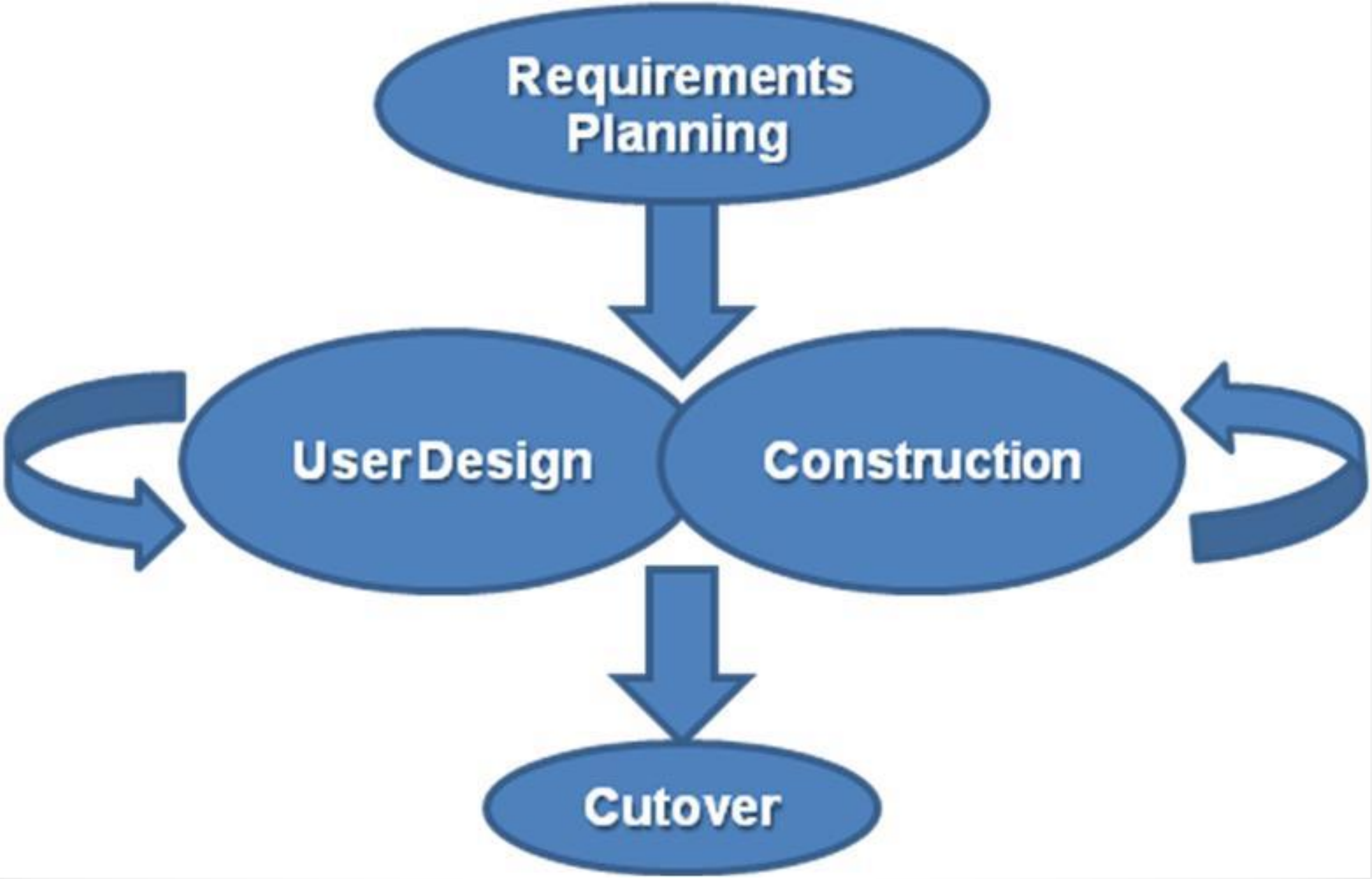
- Development phases are repeated as required until an acceptable system is found
- User participates
- Spiral (evolutionary) development SDLC in which we constantly cycle through phases at different levels of details



3. Rapid Application Development (RAD)

- Systems-development methodology that focuses on quickly:
  - building working model of software
  - getting feedback from users
  - using that feedback to update the working model
  - making several iterations of development
  - developing/implementing a final version
- This *greatly decreases* design / implementation time ⇒ shortened development (compressed process)
- Uses extensive user cooperation, prototyping, integrated CASE tools, and code generators

Rapid Application Development (RAD) – cont



Rapid Application Development (RAD) – cont

- **Requirements planning**:
  - overall requirements for system are defined
  - team is identified, and
  - feasibility is determined (similar to analysis/design phases in Waterfall Approach)
- **User design**:
  - prototyping the system with the user using CASE tools in creating interfaces/reports
  - e.g. JAD (**joint application design**) session: all stakeholders have a structured discussion about design of the system

Rapid Application Development (RAD) – cont

- **Construction**:
  - coding the system using CASE tools
  - it is an interactive, iterative process
  - and changes can be made as developers are working on the program
- **Cutover**:
  - delivery of developed system (i.e. implementation)

SDLC Types: 4. Agile Methodologies

- Group of methodologies that utilize incremental changes with a focus on quality, details (started: 2001)

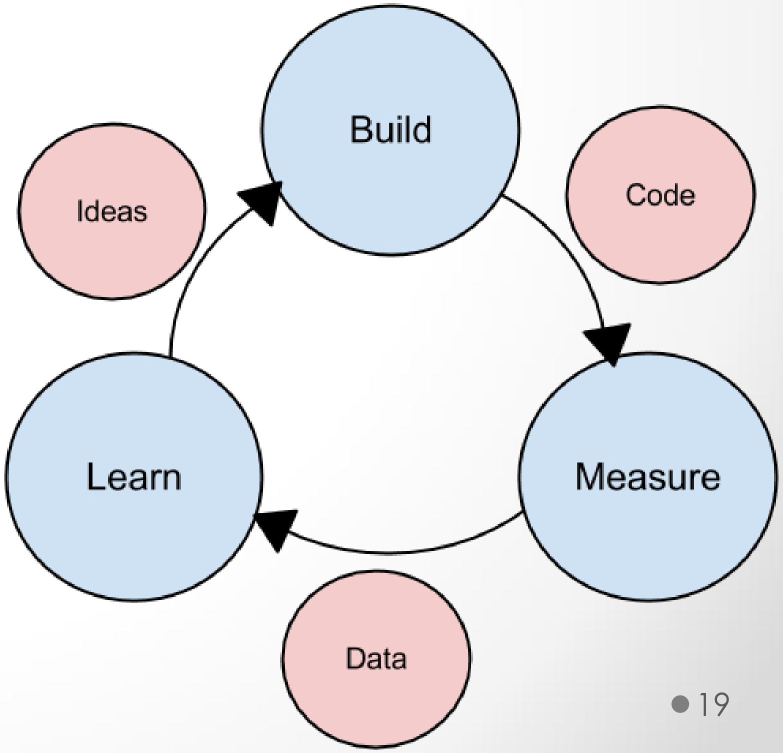
- Each increment is released in a specified time (called a "time box") ⇒ regular release schedule with very specific objectives

- Share some RAD principles:
  - iterative development
  - user interaction
  - ability to change

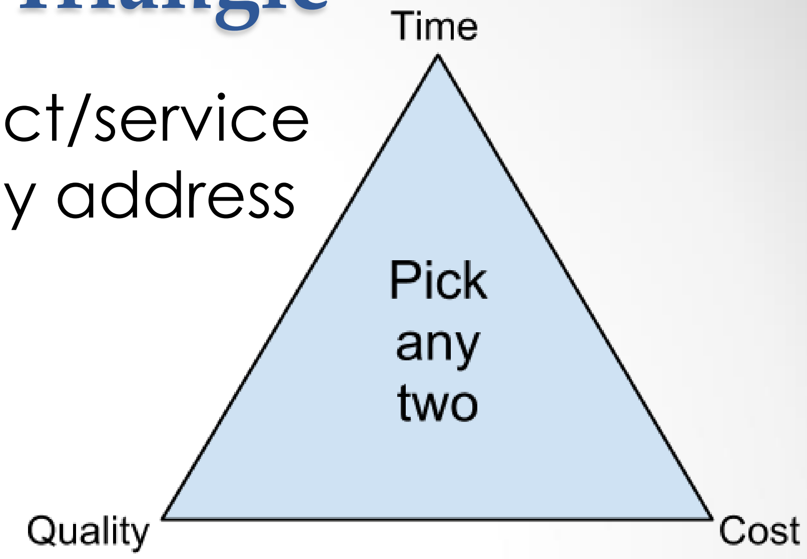
- Goal: provide flexibility of iterative approach, while ensuring a quality product

SDLC Types: 5. Lean Methodology

- Lean Methodology:
  - New concept
  - Focus is on taking initial idea and developing **minimum viable product** (MVP)
  - MVP: working software application with just enough functionality to demonstrate the idea behind the project
  - MVP is given to potential users for review; team then determines whether to continue in same direction or rethink idea behind project ⇒ new MVP
  - Iterative process: until final product is completed



Note: Quality Triangle



- Simple concept: for any product/service being developed, you can only address 2 of the following:
  - Time
  - Cost
  - Quality

- e.g. you cannot complete a *low-cost, high-quality* project in a *small amount of time*

- Also, if you can spend a *lot of money* ⇒ project can be completed *quickly* with *high-quality* results

- If *completion date* is not a priority, then it can be completed at a *lower cost* with *higher-quality* results

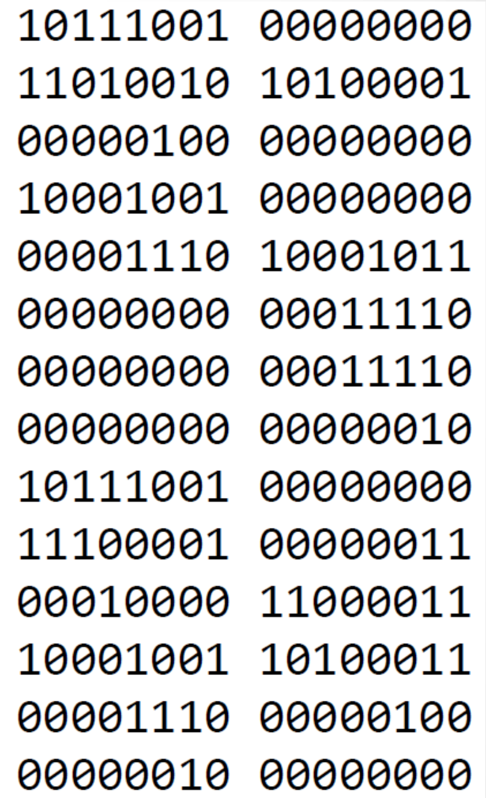
Programming Languages

Programming Languages

- One way to characterize programming languages is by their “generation”:
  - **First-generation languages**
  - **Second-generation languages**
  - **Third-generation languages**
  - **Fourth-generation languages**

Programming Languages (cont.)

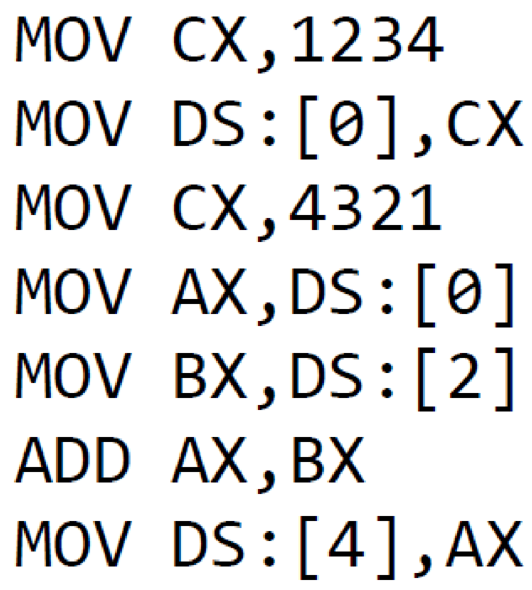
- First-generation languages
  - Called **machine code**: specific to the type of hardware to be programmed
  - Each type of computer hardware has a different **low-level programming language**
  - Uses actual ones and zeroes (bits) in the program, using binary code
  - Example here: adds ‘1234’ and ‘4321’ using machine language



```
10111001 00000000
11010010 10100001
00000100 00000000
10001001 00000000
00001110 10001011
00000000 00011110
00000000 00011110
00000000 00000010
10111001 00000000
11100001 00000011
00010000 11000011
10001001 10100011
00001110 00000100
00000010 00000000
```

Programming Languages (cont.)

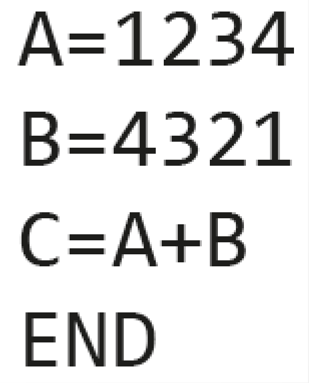
- Second-generation languages
  - Called Assembly language (also low-level language)
  - Gives English-like phrases to machine-code instructions, making it easier to program
  - Run through an assembler, which converts it into machine code
  - See here program that adds '1234' and '4321' using assembly language



```
MOV CX,1234
MOV DS:[0],CX
MOV CX,4321
MOV AX,DS:[0]
MOV BX,DS:[2]
ADD AX,BX
MOV DS:[4],AX
```

Programming Languages (cont.)

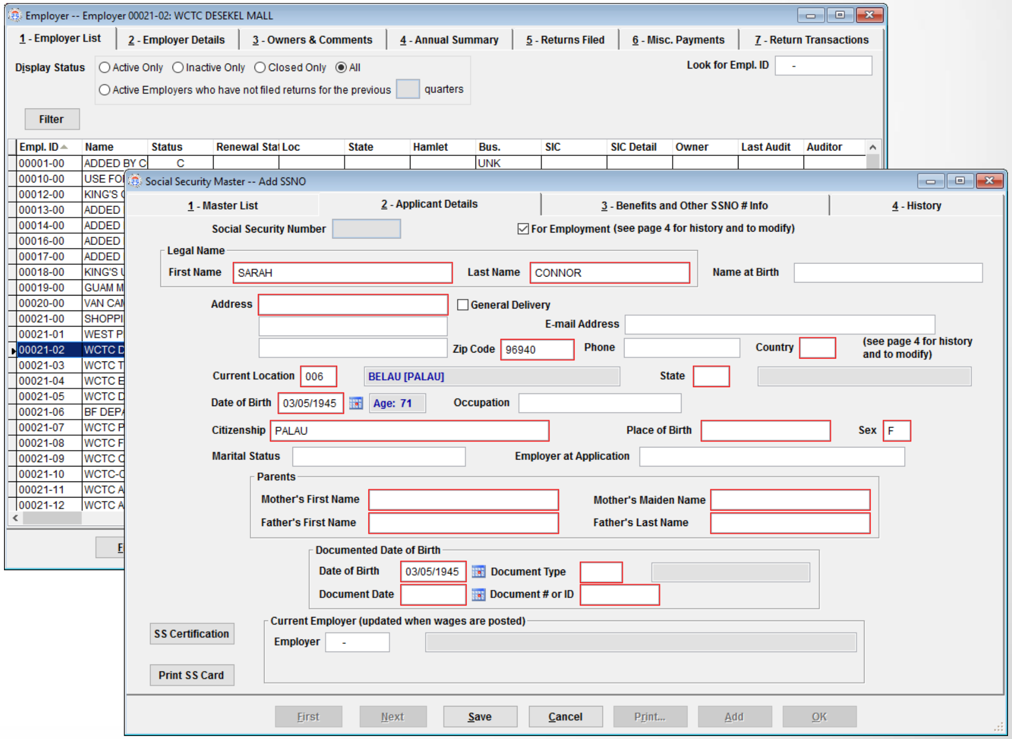
- Third-generation languages
  - *Not specific* to type of hardware on which they run
  - Much more like spoken languages
  - Most third-generation languages must be **compiled**, a process that converts them into machine code
  - Well-known third-generation languages: *BASIC*, *C*, *Pascal*, and *Java*
  - Here is a program (in *BASIC*) that adds '1234' and '4321'



```
A=1234
B=4321
C=A+B
END
```

Programming Languages (cont.)

- Fourth-generation languages
  - Class of *programming tools* that enable fast application development using *intuitive* interfaces and environments
  - Have very specific purpose, such as database interaction or report-writing
  - Can be used by those with very little training in programming; allow for *quick development* of applications and/or functionality
  - Examples: *Clipper*, *FOCUS*, *FoxPro*, *SQL*, and *SPSS*



Display Status ○ Active Only ○ Inactive Only ○ Closed Only ● All

○ Active Employers who have not filed returns for the previous quarters

Look for Empl. ID -

Empl. ID	Name	Status	Renewal Stat	Loc	State	Hamlet	Bus.	SIC	SIC Detail	Owner	Last Audit	Auditor
00001-00	ADDED BY C	C					UNK					
00010-00	USE FOI											
00012-00	KING'S (											
00013-00	ADDED											
00014-00	ADDED											
00016-00	ADDED											
00017-00	ADDED											
00018-00	KING'S U											
00019-00	GUAM M											
00020-00	VAN CAI											
00021-00	SHOPPI											
00021-01	WEST P											
00021-02	WCTC D											
00021-03	WCTC T											
00021-04	WCTC E											
00021-05	WCTC D											
00021-06	BF DEPA											
00021-07	WCTC P											
00021-08	WCTC F											
00021-09	WCTC C											
00021-10	WCTC-C											
00021-11	WCTC A											
00021-12	WCTC A											

Social Security Number

☑ For Employment (see page 4 for history and to modify)

Legal Name

First Name SARAH Last Name CONNOR

Name at Birth

Address

☐ General Delivery

E-mail Address

Zip Code 96940 Phone Country (see page 4 for history and to modify)

Current Location 006 BELAU [PALAU] State

Date of Birth 03/05/1945 Age: 71 Occupation

Citizenship PALAU Place of Birth Sex F

Marital Status Employer at Application

Parents

Mother's First Name Mother's Maiden Name

Father's First Name Father's Last Name

Documented Date of Birth

Date of Birth 03/05/1945 Document Type

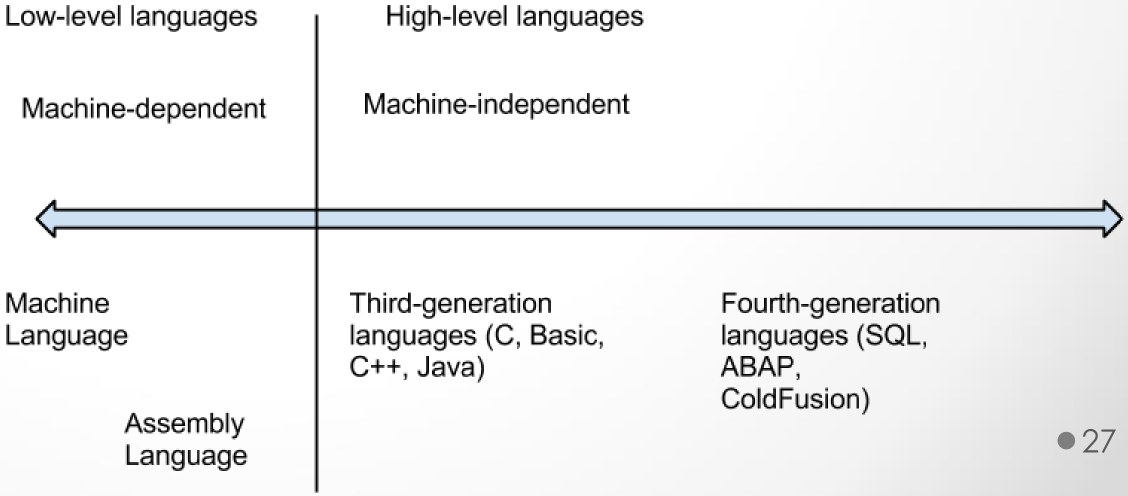
Document Date Document # or ID

Current Employer (updated when wages are posted)

Employer -

Programming Languages (cont.)

- Higher vs. Lower Level Languages
  - Lower-level languages (e.g. assembly language): much more efficient and execute much more quickly; you have finer control over the hardware as well
  - Sometimes, combination of higher- and lower-level languages are mixed ⇒ "best of both worlds": overall structure and interface using a higher-level language, but use lower-level languages for parts of program that are used many times or require more precision



Programming Languages (cont.)

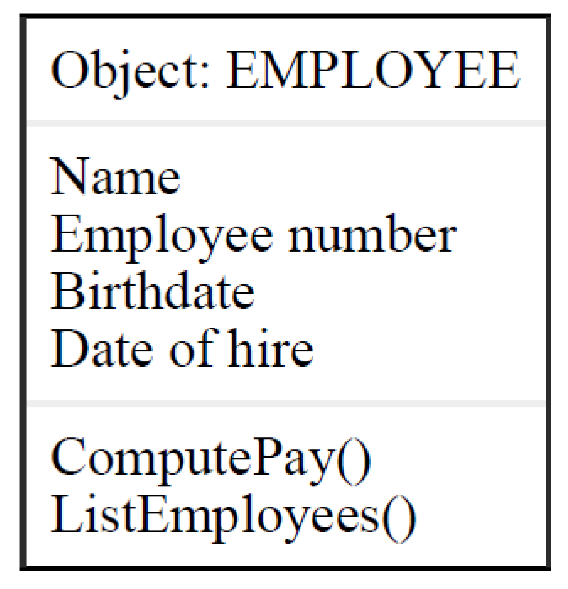
- Compiled vs. Interpreted
  - Another way to classify programming languages
  - **Compiled** language: code is translated into a machine-readable form called an "executable" that can be run on the hardware (e.g. *C*, *C++*, and *COBOL*)
  - **Interpreted** language: requires a "runtime program" to be installed in order to execute; this program then interprets the program code *line by line* and runs it; generally easier to work with but slower (e.g. *BASIC*, *PHP*, *PERL*, and *Python*)
  - Web languages (*HTML* and *Javascript*) also considered interpreted because they require a browser in order to run
  - Note, Java programming language: interesting exception to this classification (*hybrid* of the two)

Programming Languages (cont.)

- Procedural vs. Object-Oriented
  - **Procedural** programming language: designed to allow a programmer to define a specific starting point for the program and then execute *sequentially* (include all early programming languages)
  - **Object-oriented** programming language: uses *interactive* and *graphical user interfaces* (GUI) to allow the user to define the flow of the program
    - programmer defines "objects" that can take certain actions based on input from the user
  - Procedural program focuses on sequence of activities to be performed, while object-oriented program focuses on the different items being manipulated

Programming Languages (cont.)

- Procedural vs. Object-Oriented (cont.)
  - Example of object-oriented code (human resource system)
  - **object** ("EMPLOYEE") is created in program to retrieve or set data regarding an employee
  - Every object has **properties**: descriptive fields associated with the object ("Name", "Employee number", "Birthdate" and "Date of hire")
  - Object also has **methods** which can take actions related to the object:
    "ComputePay()": money owed to person
    "ListEmployees()": who works under that employee



Object: EMPLOYEE

Name
Employee number
Birthdate
Date of hire

ComputePay()
ListEmployees()

Programming Languages (cont.)

- Programming Tools
  - Traditional Tools: text editor, checking syntax, code compiler
  - Additional tools:
    - **Integrated Development Environment** (IDE)
    - **Computer-Aided Software-Engineering** (CASE) tools

Programming Languages (cont.)

- Programming Tools (cont.)

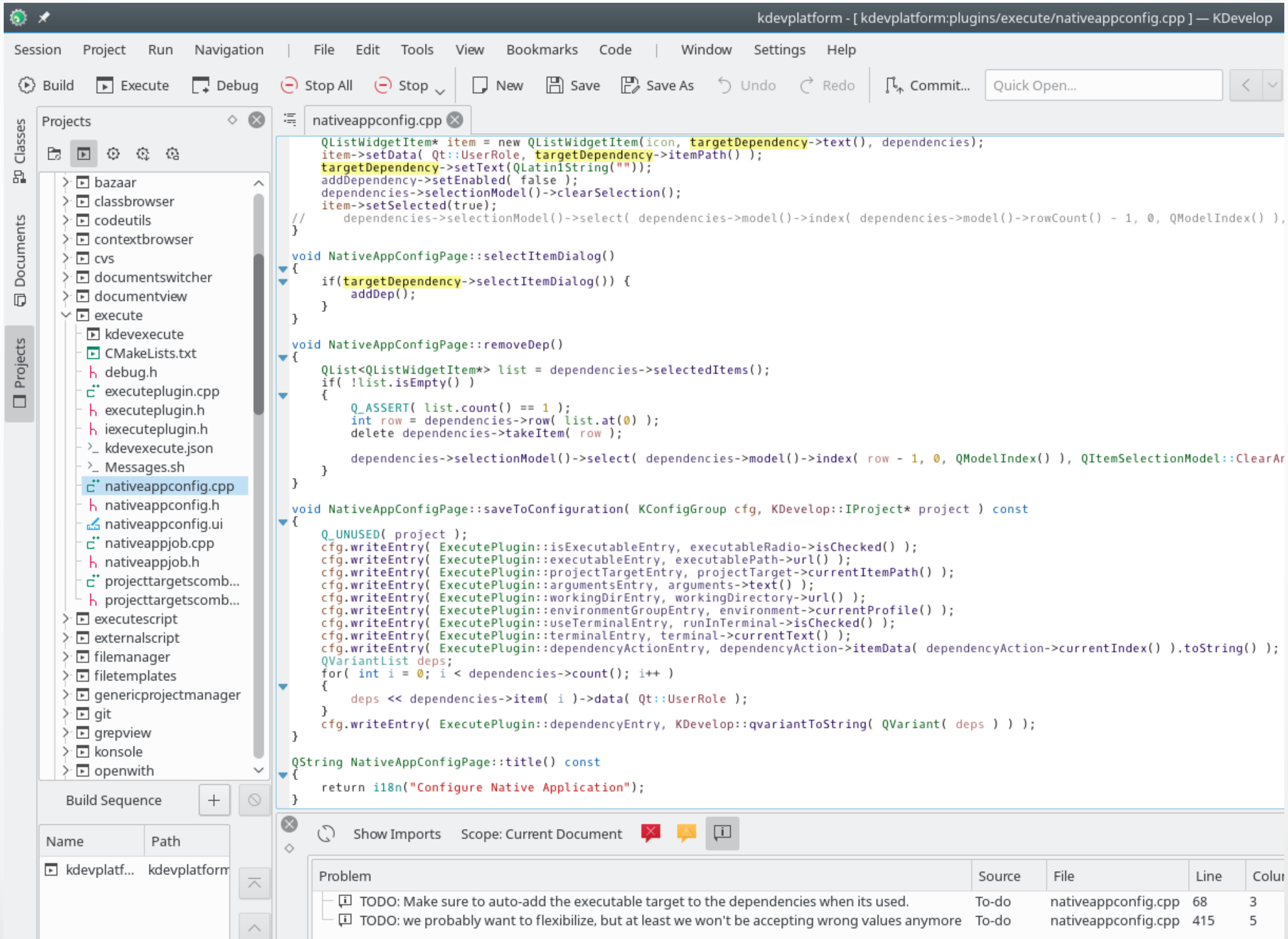
  Integrated Development Environment (IDE) provides:

  - an editor for writing the program that will *color-code* or *highlight* keywords from the programming language
  - help system
  - compiler/interpreter
  - *debugging* tool (to resolve problems)
  - *check-in/check-out* mechanism (so that more than one programmer can work on code)
  - e.g. Microsoft Visual Studio: IDE for Visual C++, Visual BASIC

Programming Languages (cont.)

- Programming Tools (cont.)

  Integrated Development Environment (IDE) example



```
    QListWidgetItem* item = new QListWidgetItem(icon, targetDependency->text(), dependencies);
    item->setData( Qt::UserRole, targetDependency->itemPath() );
    targetDependency->setText(QLatin1String(""));
    addDependency->setEnabled( false );
    dependencies->selectionModel()->clearSelection();
    item->setSelected(true);
//      dependencies->selectionModel()->select( dependencies->model()->index( dependencies->model()->rowCount() - 1, 0, QModelIndex() ),
}

void NativeAppConfigPage::selectItemDialog()
{
    if(targetDependency->selectItemDialog()) {
        addDep();
    }
}

void NativeAppConfigPage::removeDep()
{
    QList<QListWidgetItem*> list = dependencies->selectedItems();
    if( !list.isEmpty() )
    {
        Q_ASSERT( list.count() == 1 );
        int row = dependencies->row( list.at(0) );
        delete dependencies->takeItem( row );

        dependencies->selectionModel()->select( dependencies->model()->index( row - 1, 0, QModelIndex() ), QItemSelectionModel::ClearAr
    }
}

void NativeAppConfigPage::saveToConfiguration( KConfigGroup cfg, KDevelop::IProject* project ) const
{
    Q_UNUSED( project );
    cfg.writeEntry( ExecutePlugin::isExecutableEntry, executableRadio->isChecked() );
    cfg.writeEntry( ExecutePlugin::executableEntry, executablePath->url() );
    cfg.writeEntry( ExecutePlugin::projectTargetEntry, projectTarget->currentItemPath() );
    cfg.writeEntry( ExecutePlugin::argumentsEntry, arguments->text() );
    cfg.writeEntry( ExecutePlugin::workingDirEntry, workingDirectory->url() );
    cfg.writeEntry( ExecutePlugin::environmentGroupEntry, environment->currentProfile() );
    cfg.writeEntry( ExecutePlugin::useTerminalEntry, runInTerminal->isChecked() );
    cfg.writeEntry( ExecutePlugin::terminalEntry, terminal->currentText() );
    cfg.writeEntry( ExecutePlugin::dependencyActionEntry, dependencyAction->itemData( dependencyAction->currentIndex() ).toString() );
    QVariantList deps;
    for( int i = 0; i < dependencies->count(); i++ )
    {
        deps << dependencies->item( i )->data( Qt::UserRole );
    }
    cfg.writeEntry( ExecutePlugin::dependencyEntry, KDevelop::qvariantToString( QVariant( deps ) ) );
}

QString NativeAppConfigPage::title() const
{
    return i18n("Configure Native Application");
}
```

Name	Path
kdevplatf...	kdevplatform

Problem	Source	File	Line	Colu
TODO: Make sure to auto-add the executable target to the dependencies when its used.	To-do	nativeappconfig.cpp	68	3
TODO: we probably want to flexibilize, but at least we won't be accepting wrong values anymore	To-do	nativeappconfig.cpp	415	5

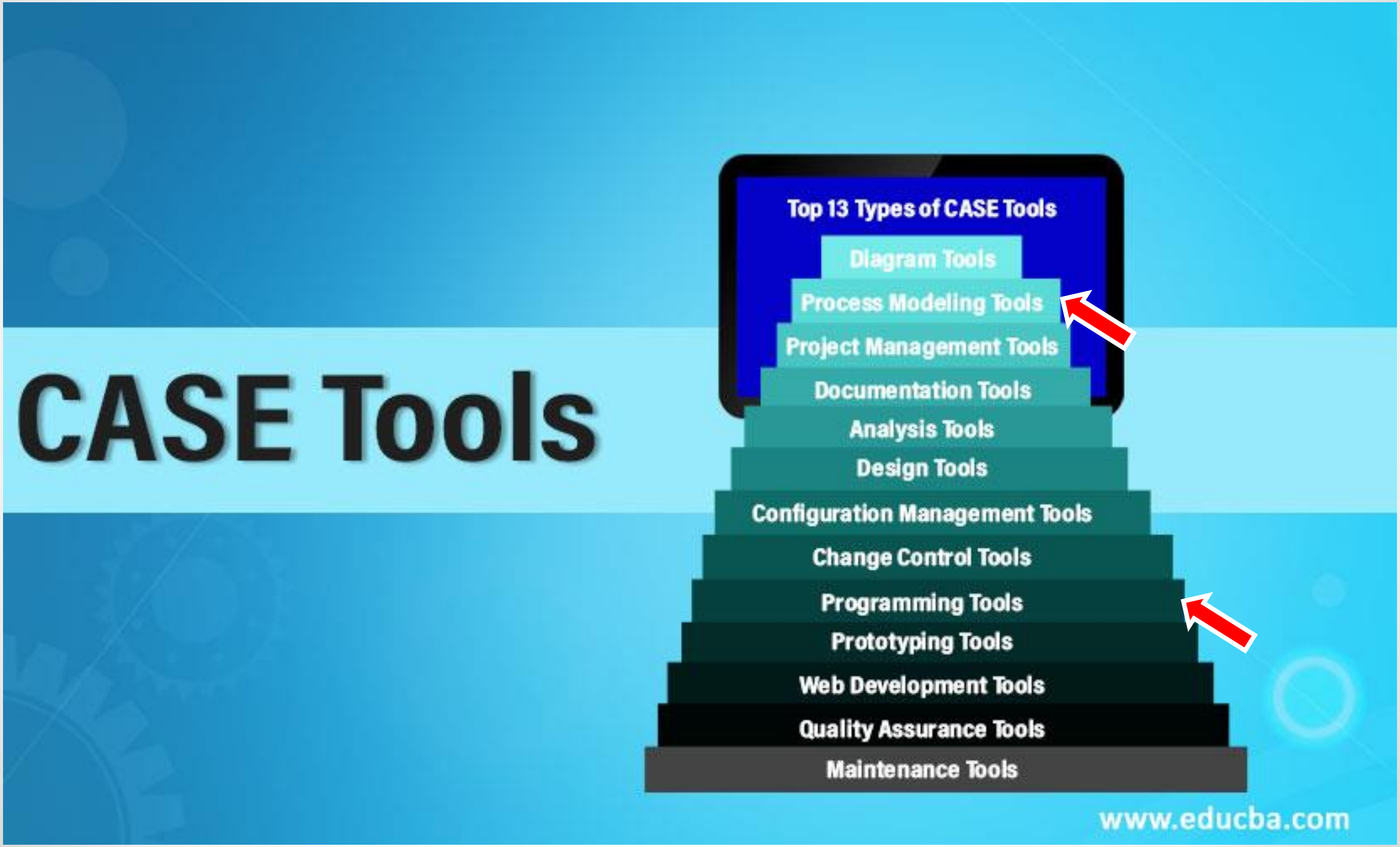
Programming Languages (cont.)

- Programming Tools (cont.)

  Computer-aided software-engineering (CASE) Tools:

  - Allows a designer to develop software with little or *no programming*
  - *Writes the code* for the designer
  - Goal is to generate quality code based on input created by the designer

Programming Languages (cont.)



CASE Tools

Top 13 Types of CASE Tools

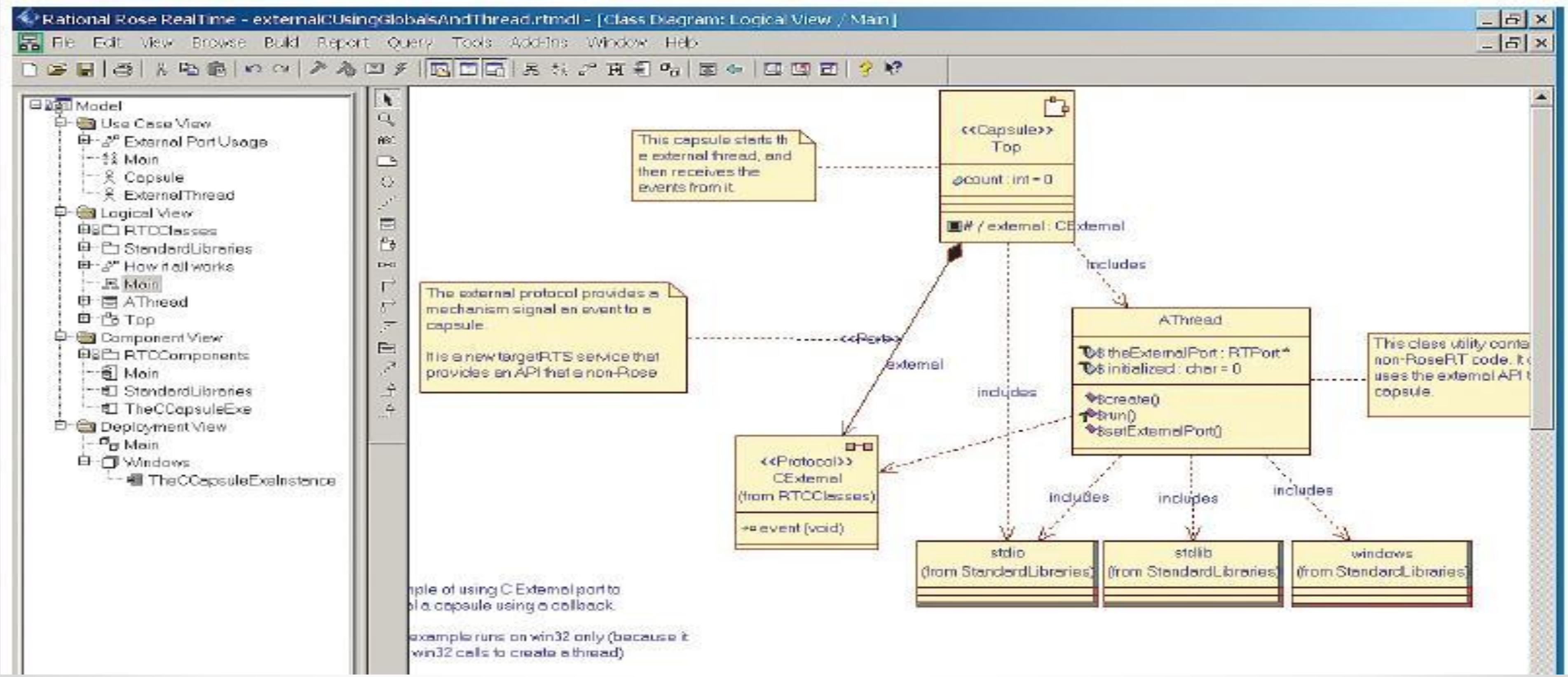
- Diagram Tools
- Process Modeling Tools
- Project Management Tools
- Documentation Tools
- Analysis Tools
- Design Tools
- Configuration Management Tools
- Change Control Tools
- Programming Tools
- Prototyping Tools
- Web Development Tools
- Quality Assurance Tools
- Maintenance Tools

www.educba.com

Programming Languages (cont.)

- Programming Tools (cont.)

  Computer-aided software-engineering (CASE) example:



This capsule starts th e external thread, and then receives the events from it.

The external protocol provides a mechanism signal an event to a capsule.

It is a new targetRTS service that provides an API that a non-Rose

This class utility conta non-RoseRT code. It uses the external API t capsule.

nple of using C External port to l a capsule using a callback.

example runs on win32 only (because it win32 calls to create a thread)

Programming Languages (cont.)

- Programming Tools (cont.)

  Computer-aided software-engineering (CASE) Tools (cont.):

  - Diagramming tools enable graphical representation
  - e.g. Unified Modeling Language (UML): general-purpose, developmental, modeling language used to *visualize the design of a system*
  - Computer displays and report generators help prototype how systems "look and feel"
  - Code generators enable automatic generation of programs and database code directly from design documents, diagrams, forms, and reports

Sources

- Modern Systems Analysis and Design. Joseph S. Valacich and Joey F. George. Pearson. Eighth Ed. 2017. Chapter 1: The Systems Development Environment.
- Information Systems for Business and Beyond. David T. Bourgeois. The Saylor Academy. 2014. Chapter 10: Information Systems Development.