UPES

Xebia

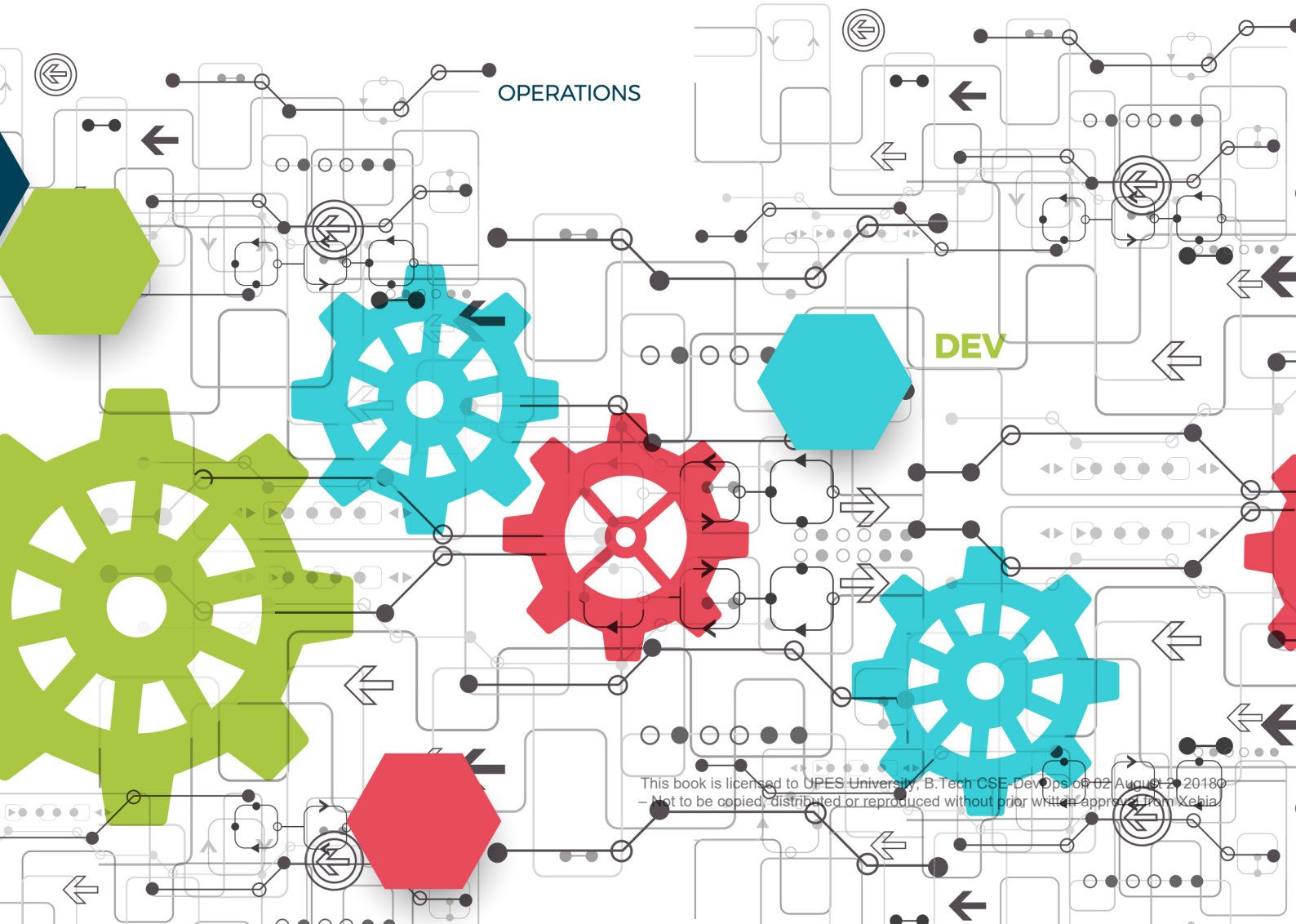
2 YEAR

B.Tech Computer Science and Engineering in DevOps

DEVOPS AUTOMATION

Semester 03 | Facilitator Handbook

Release 1.0.0



Copyright & Disclaimer

B. TECH CSE with Specialization in DevOps

Version 1.0.0

Acknowledgements

We would like to sincerely thank the experts who have contributed to and shaped B. TECH CSE with Specialization in DevOps. Version 1.0.0

SME

Rajagopalan Varadan
A tech enthusiast who loves learning and working with cutting-edge technologies like DevOps, Big Data, Data science, Machine Learning, AWS & Open stack

Course Reviewers.

Aditya Kalia | Xebia
Maneet Kaur | Xebia
Sandeep Singh Rawat | Xebia
Abhishek Srivastava | Xebia
Rohit Sharma | Xebia

Review Board Members.

Anand Sahay | Xebia

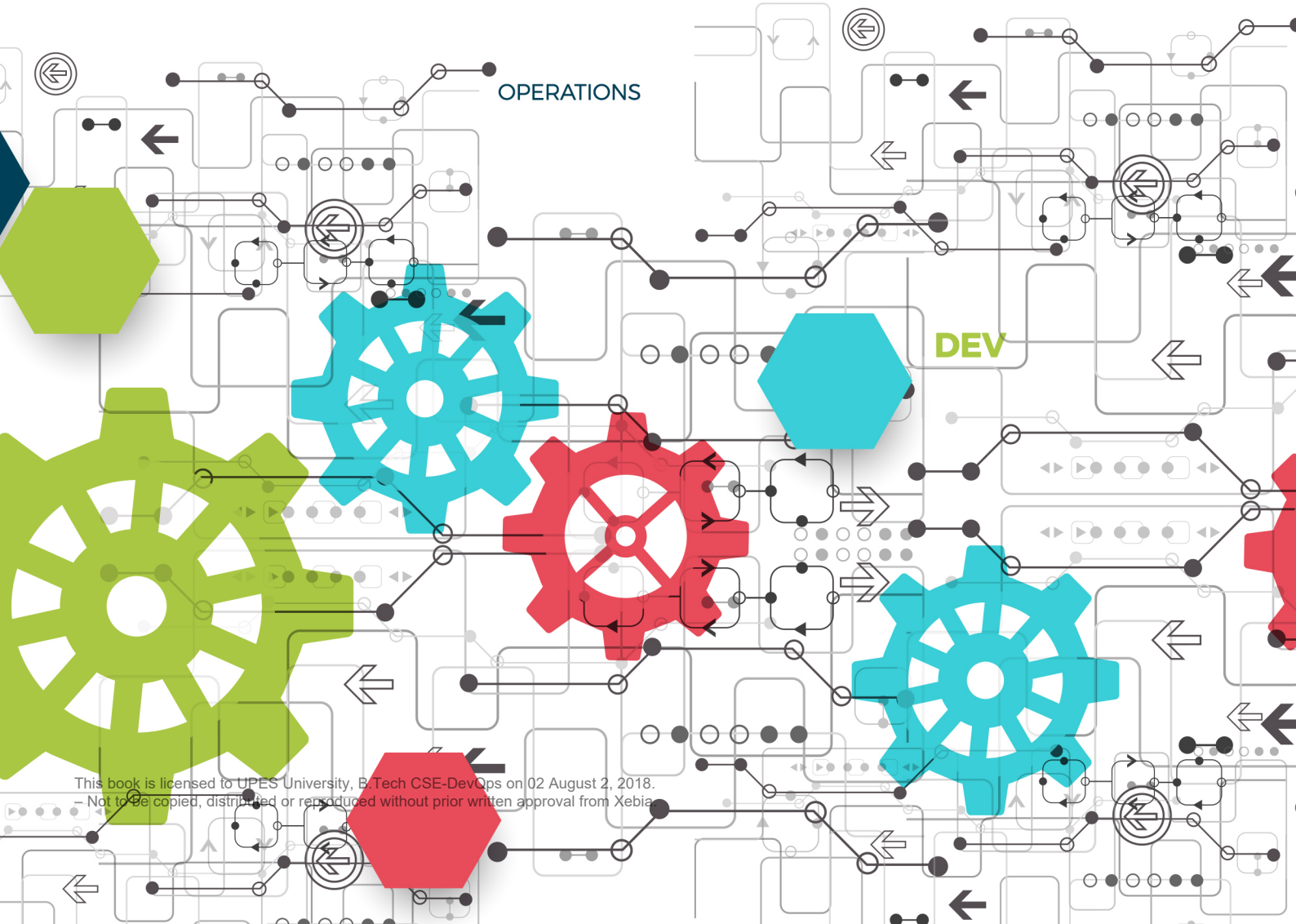
Xebia Group consists of seven specialized, interlinked companies: Xebia, Xebia Academy, XebiaLabs, StackState, GoDataDriven, Xpirit and Binx.io. With offices in Amsterdam and Hilversum (Netherlands), Paris, Delhi, Bangalore and Boston, we employ over 700 people worldwide. Our solutions address digital strategy; agile transformations; DevOps and continuous delivery; big data and data science; cloud infrastructures; agile software development; quality and test automation; and agile software security.

ODW is dedicated to provide innovative and creative solutions that contribute in growth of emerging technologies. As a learning experience provider, ODW strengths include providing unique, up to date content by combining industry best practices with leading edge technology. ODW delivers high quality solutions and services which focus on digital learning transformation.

B.Tech Computer Science and Engineering in DevOps

DEVOPS AUTOMATION

MODULE 1
Introduction to Automation



Contents

Module Learning Objectives	**1**
Module Topics	**1**
1. Introduction to Automation	**2**
2. The Software Delivery Pipeline	**4**
3. The Build Process	**8**
4. Overview of Rapid Application Development (RAD)	**20**
5. Code Generation - An Introduction	**24**
6. Model-Driven Architecture (MDA) - An Introduction	**26**
In a nutshell, we learnt:	**32**

MODULE 1

Introduction to Automation

You will learn about the 'Introduction to Automation' in this module.

Module Learning Objectives

At the end of the Module you would be able to learn the following

- Introduction to Automation
- The phases involved in software development - delivery pipeline.
- Fully automated software delivery process that includes:
  - Automated Build
  - Automated Test
  - Automated Deployment
  - Automated Provisioning
- The concept of Rapid Application Development, its advantages and disadvantages.
- Modern code generators and the way they work.
- The Model-driven architecture, its concepts, models and tools.

Module Topics

Let us take a quick look at the topics we will cover in this module:

1. Introduction to Automation
2. Development Delivery Pipeline Overview.
3. Automation of Software Delivery Process.
   - Automated Build

- Automated Test
- Automated Deployment
- Automated Provisioning

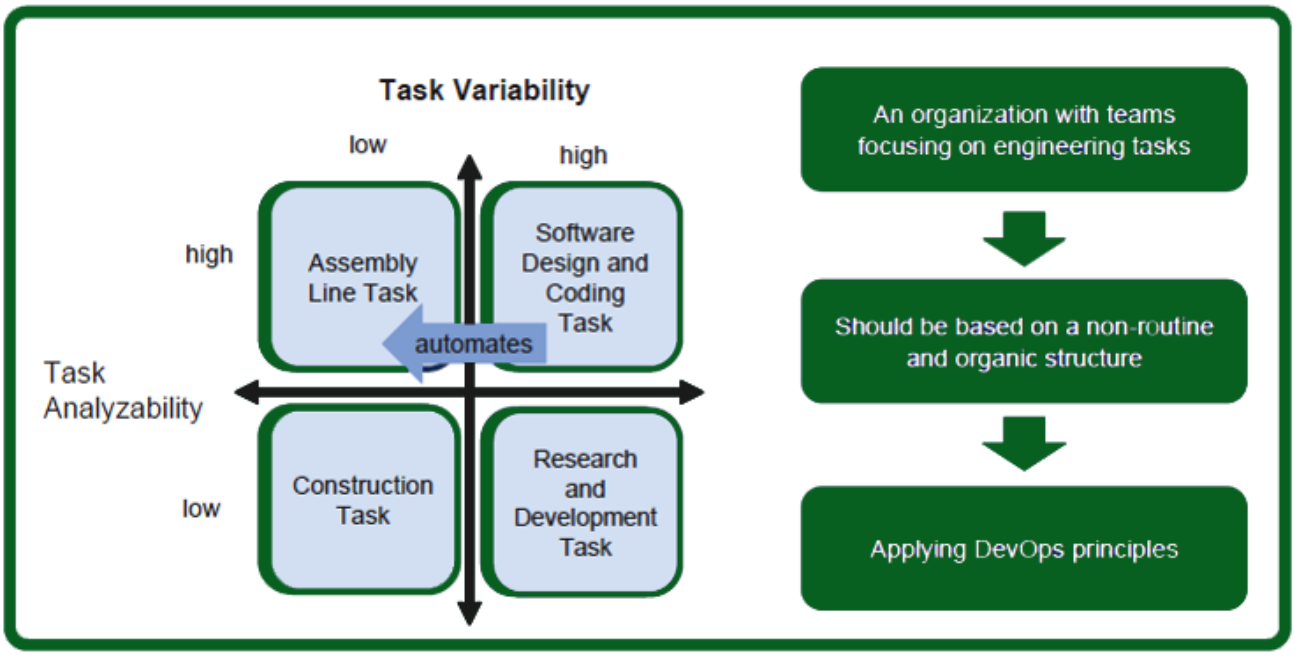
4. RAD (Rapid Application Development).
5. Code Generation.
6. MDA/MDD (Model-Driven Architecture/Development).

1. Introduction to Automation

Both cognitive and manual routine jobs can be automated due to technology advancements. IT software delivery also involves many manual jobs that can be automated.

Optimization of software delivery processes can be achieved by continuous delivery practices that rely on automation.

Automation, in combination with other DevOps principles help in delivering value efficiently.



Source: Perrow (1967)

To explain automation, we can take for example two routine types of jobs:

- Routine cognitive, include sales and office occupations performing administrative tasks.
- Routine manual, include construction, transportation, production and repair occupations.

With the advent of technology advancements, more and more routine jobs can be automated. For example, ATMs greatly reduce the job load of a teller in a bank. In a few seconds, the transaction gets over and the entries are automatically done in the user's account. This has also eliminated the time spent in standing in queues in bank and the time taken for manual transaction and entries.

Similarly in IT, the distinction between manual versus cognitive and routine versus non-routine tasks can be made. Also in IT, routine tasks can be very well automated.

Traditionally, IT software delivery started with project plan, with task or work broken down. From the perspective of a single project, many tasks appear to be non-routine at the start of the project, because only a few task iterations are planned. Retrospective, many project routine activities are executed throughout the project (planned or not planned). Furthermore, after the production handover to operations, many manual routine tasks are required throughout the lifecycle of the software delivered by each project.

Continuous Delivery and Data Center/Cloud Automation have a profound impact on the automation of routine tasks in IT. With Continuous Delivery and Data Center/Cloud Automation, many manual tasks, such as installation and deployment activities are being automated. Combined with Agile and

Lean principles, these initiatives, identify the routine tasks within the software delivery which can be automated.

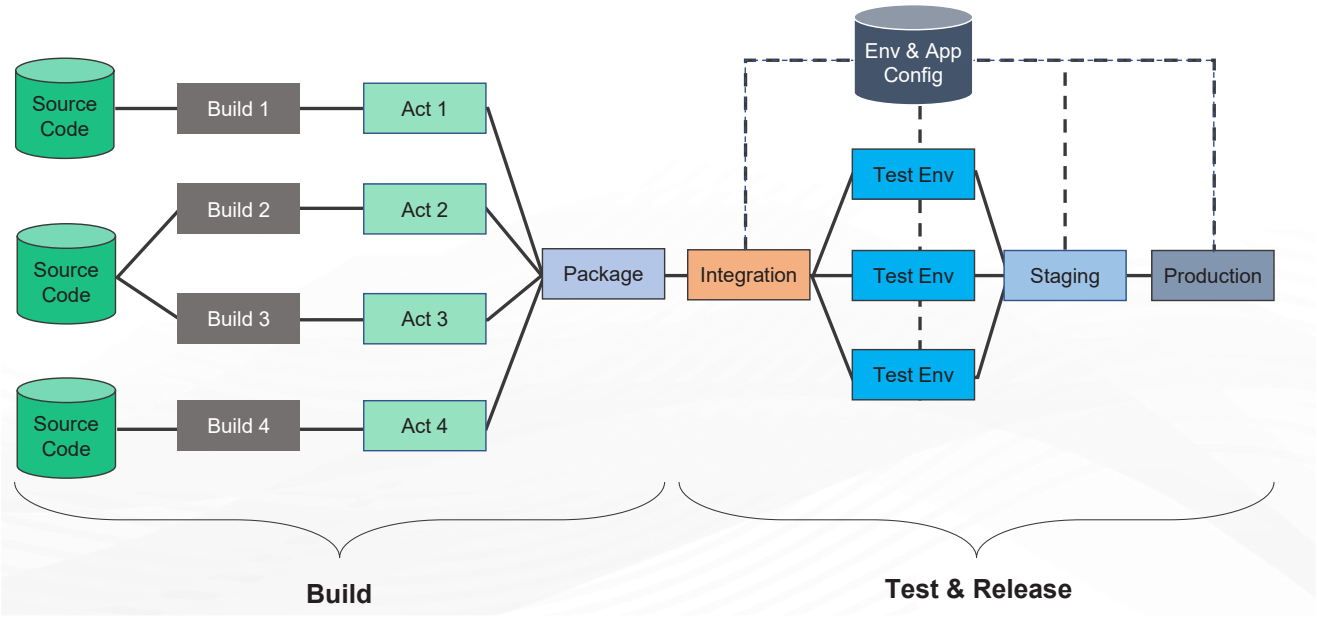
The figure shows the task classification quadrant by Charles Perrow. The quadrant is based on two dimensions: Tasks Analyzability and Task Variability. Their distinction:

- Task Analyzability is defined as the extent to which, when an exception is encountered, there are known analytical methods for dealing with it.
- Task Variability is defined by the number of exceptions to standard procedures encouraged in the application of a given technology.

Considering software delivery processes, many tasks to deliver software has a relatively low task variability and a high task analyzability or the software delivery process can be transformed using tasks with low variability. Once, these manual routine tasks are identified, these tasks can be automated with engineering tasks. Scripted (manual) test execution, (manual) deployment execution, (manual) code compilation are examples of routine tasks within a software delivery process.

We'll now see about the technologies and processes involved in software development and delivery.

2. The Software Delivery Pipeline



Depending on the software being developed the delivery pipeline will differ. Hence, there's no strict rule for the stages in the pipeline. When we look at different pipelines, the common stages involved in a software delivery pipeline are:

Build

During the build stage, application/software is built, i.e, the actual code is written and compiled and archived. Units tests are also run at this stage. A source-code repository is similar to a database where the source code is committed or uploaded by developers and maintained. Multiple versions of a commit are maintained in this repository. E.g BitBucket and GitHub. This source-code repository serves as the input for a build. The output is a working piece, that is a small component of the software being built. This output, an artefact, is stored in an artefact repository. It is in this stage that multiple builds are packaged into one single unit, to enable for testing.

Test

During this stage, the production environment is cloned and a staging environment is created. The build artiefacts are integrated and installed or deployed in this staging environment. In this stage, automated tests are run to check the new version of the software. Functional tests or integration tests are done to verify the new functionalities. Regression tests are done to ensure that new version doesn't impact any functionality. Performance tests are done finally.

Release

Here, the software is deployed in the production environment. In this stage, additional tests are also done to ensure the software works as expected.

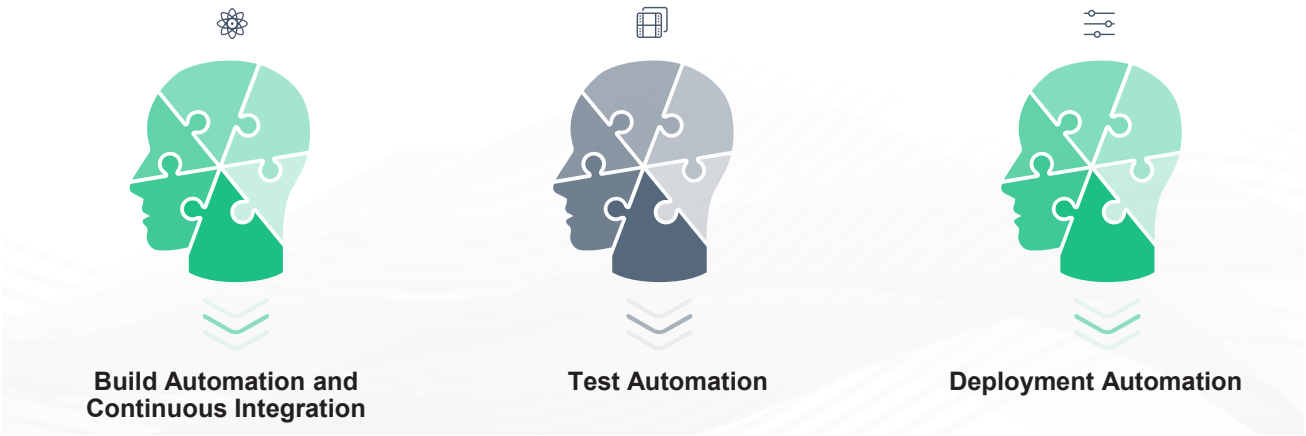
The stages happen in a sequential manner in simple pipelines. This means, any error in one stage has an impact on the next stage, there is a possibility that the overall process gets delayed.

In a complex pipeline, multiple instances of some stages can exist. For example, there could be production stages for each of the data centres. These production stages are run in parallel. These production runs can have manual approval steps to avoid unintended deployments.

To provide rapid feedbacks, early stages should be kept simple and run fast. Later stages run progressively more complex tests in a production-like environment. These stages also take a longer time to run and they even require more dependencies in a production-like environment.

2.1 Overview of the Continuous Delivery Pipeline

A typical continuous delivery pipeline includes the following stages.



Continuous delivery is a strategy, by which companies deliver new features of the software to the users, in a fast and efficient manner. Continuous delivery believes in creating a repeatable, reliable and incrementally improving process, for taking software from an idea to the product. Continuous delivery imparts automated software production line to enable a constant flow of changes into production.

Software delivery process is broken down into multiple stages, where at each stage, quality of new features undergoes a verification process from a different angle. This ensures that functionalities are tested properly and the product is error free when it reaches the user.

Though there is no such thing as a standard pipeline, a typical Continuous Delivery pipeline has the following stages.

Build Automation and Continuous Integration:

During the first step, binaries are first built to create deliverables to be passed on to the subsequent phases. It is during this phase that the new features developed by individual developers are integrated into the central code base on a continuous basis. As soon as the code is integrated, it is built and unit tested. Continuous feedbacks keep the development team aware of the health of their application.

Test Automation:

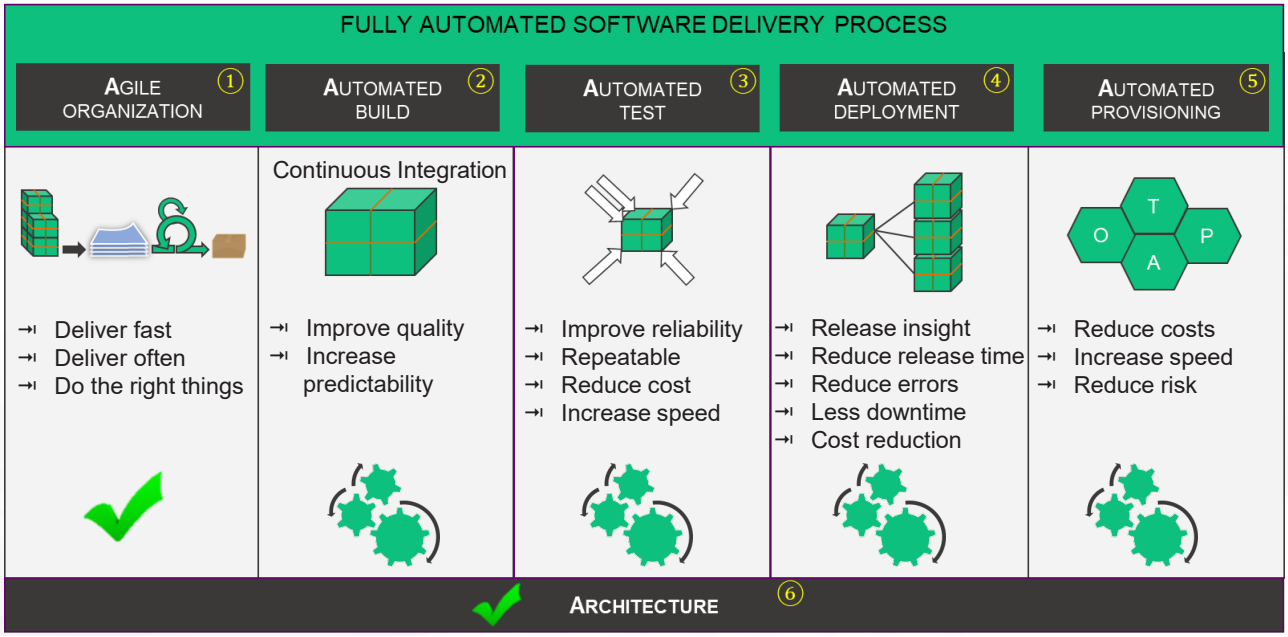
Testing of the application is done rigorously throughout this stage to make sure that the newer version of the application meets the necessary standards. It is at this stage that functionality, security,

performance and compliance are thoroughly verified by the pipeline. This stage involves different types of automated or manual activities.

Deployment Automation:

Each and every time, an application is installed for testing, it requires a deployment. During the testing process, the overall quality of the application is verified and so, the rollout time is important. Because of the quality checks done during the previous stages, this step involves relatively lower risk. The newer version of the deployment is initially rolled out to a subset of the production environment and monitored for performance and feedbacks before it is completely rolled out. With automated deployment, new functionalities can be delivered to users within minutes.

2.2 Fully Automated Software Delivery Process



We learnt about continuous delivery in the first semester. Continuous delivery can be adopted by using three basic principles.

- **Rigorous automation** - Automation of software delivery activities, such as test execution and deployments results, make the software delivery process faster, cheaper, and better. Automated task execution requires no manual (human) machine interaction time, it eliminates wait times as a result of dependencies between manual tasks, and it eliminates the need for manual task execution validation activities. Automation results in a highly reliable, repeatable, standardized process to transform business ideas into working software in production.
- **Extreme feedback -** Extreme feedback helps the teams to analyze: the effectiveness of their software delivery process, the effectiveness (real business value) of features they have delivered, and it enables teams to experiment with (the implementation of) software features using statistical evaluation methods and real customer feedback.

- **Continuous change** - The first two principles:, Rigorous Automation and Extreme Feedback, enable continuous change. The team applies collected feedback, ideas and observations of team members, and best practices from their organization or external best practices to improve their software delivery process and their (software) product.

As given in the image above, continuous delivery can be adopted using six focus topics.

1. **Agile Organization:** An Agile organization adopts Agile and Lean principles combined with autonomous, multidisciplinary teams. A DevOps organization with its culture and principles, as already covered in previous modules, represents such an Agile organization and more.
2. **Automated Build:** Automated build is defined as the automated process to transform code changes (committed by team members), automatically to published deployment artifacts, ready for deployment, and validation in (test) environments in a consistent manner.
3. **Automated Test:** Automated testing involves the automated test execution of test specifications/ test scripts. Example of tests are: static code quality analysis, unit tests, functional tests, and load tests.
4. **Automated Deployment:** Automated deployment is defined as the process to automatically deploy published deployment artifacts to application environments. This includes steps to move deployment artifacts to target (virtual) machines and steps to configure these (virtual machines) and other servers/components used by the software.
5. **Automated Provisioning:** Automated provisioning ensures the components, such as network components, server components, and runtime software stacks, of (test) environments can be created on demand using a fully automated (system provisioning) process.
6. **Architecture:** The architecture(s), defined at all levels (for example, enterprise, business, application, and technical) either amplifies or dampens the ability to optimize the software delivery process of teams. For example, strong coupling between teams results in low autonomy of teams due to cross team dependencies. Another example, the software architecture determines the complexity of adopting software code changes.

Of these, we will focus on automated build, test, deployment and provisioning.

What did you Grasp?

1. In which of the following stages, is the Functionality test done?

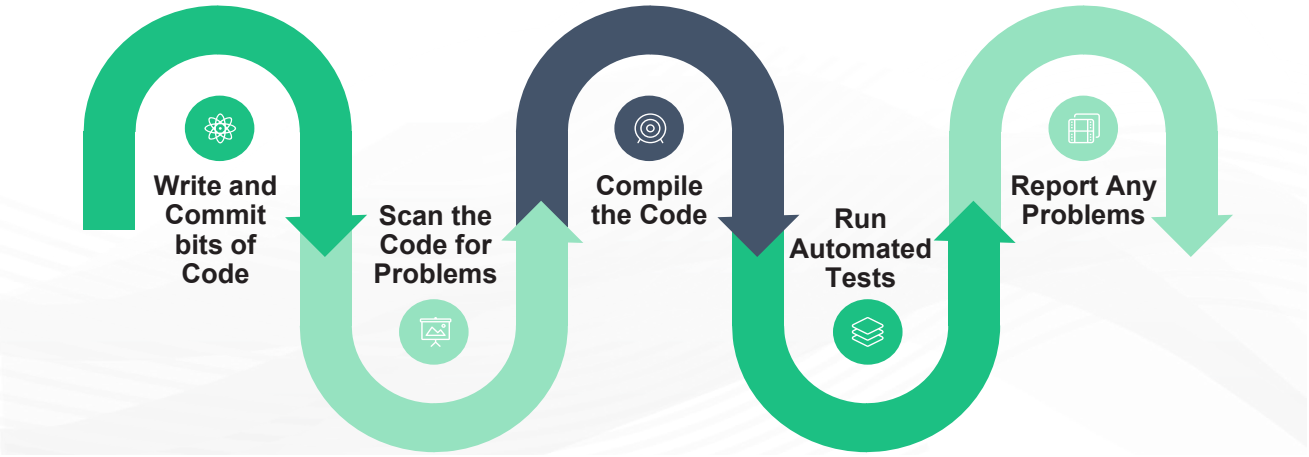
   A) Build
   B) Staging
   C) Production

2.3 Group Activity

Imagine you are developing a social network application, which would be used internally by the students and teachers in your university. Discuss among yourselves, create a storyboard for the app and list down the various steps involved the development of the application till rolling it out for production.

3. The Build Process

The build process includes the following steps:



You will learn about the Build Process on this topic.

The main phases of the Build Process are:

1. **Write and commit bits of code:** As a measure of safe-keeping the Build, the code is generally committed or uploaded to a central source code repository. Multiple versions of the source code will be available in this repository for reference at any point in time. To prevent any Data loss, due to system crashes or other unforeseen situations, the code is not kept in the individual's system for longer time periods. The pieces of code are committed to the repository, as soon as it is complete.

2. **Scan the code for problems:** After the code is committed to the repository, then it is scanned for common bugs. Based on the programming language used, multiple tools are available for scanning the code. These scanners automatically monitor the code before it is deployed.

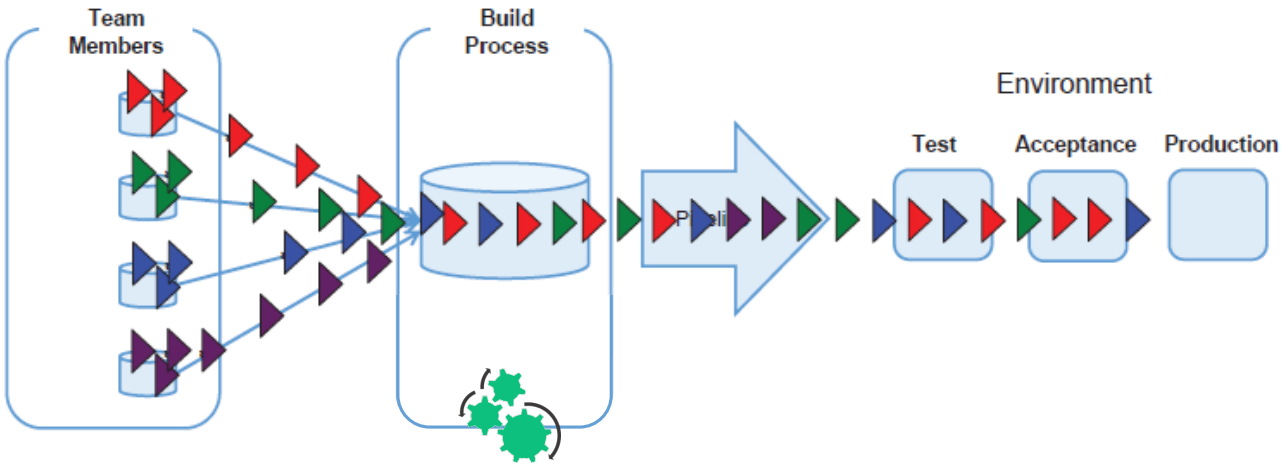
3. **Compile the code:** During the compilation process, the code is compiled to create working components of the application for various devices. The scripts can also be used to create Database Tables, Test Data and stored procedures.

4. **Run automated tests:** Here, scripts are used to load the browser, land on a web page, and test the usability of features on that page. No manual intervention is needed during this phase. Automated tests run daily, ensure high-quality applications. There are hundreds of automated tests available for a given application.

5. **Report any problems:** Whenever any problem is detected by automated tests, the development team is notified immediately of action. This helps to create bug-free code throughout the development process.

3.1 Automated Build

Automated build enables automated software package delivery flow.



Build automation transforms code changes, committed by team members, automatically to published deployment artifacts, ready for deployment, and validation in (test) environments.

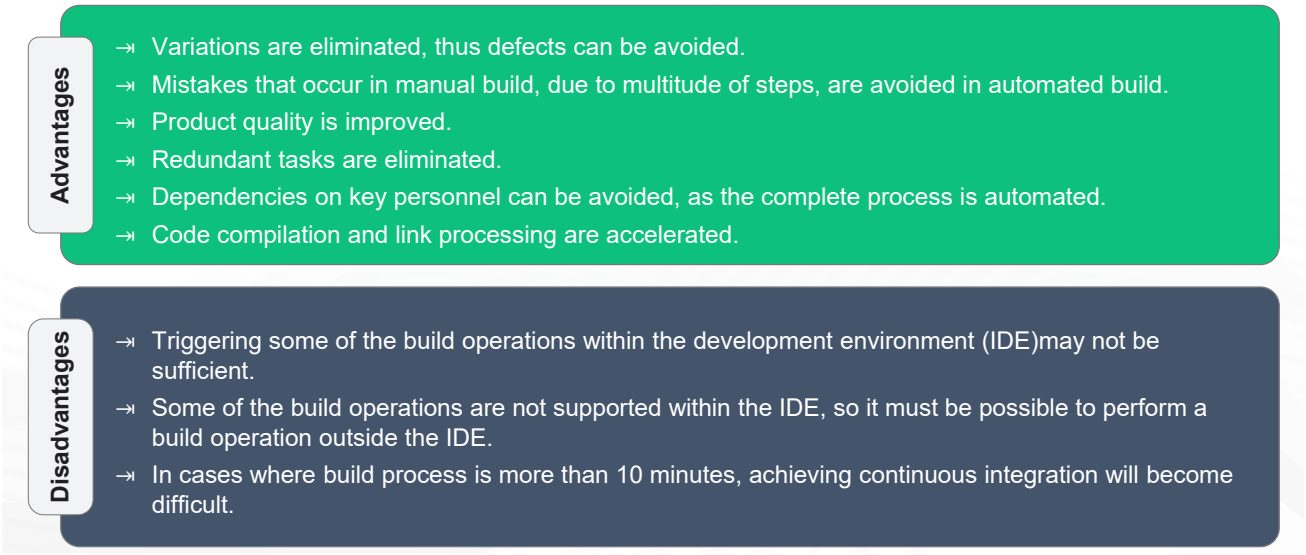
Once a team member commits a number of code changes, the code changes can be merged automatically, analyzed, compiled, unit tested, and assembled automatically. The automated build process can create a new deployment package and publish it to an artifact repository. In this manner, a continuous flow from code commit to validated deployment package can be implemented.

The automated build process is important for a fail-fast strategy. It is the component which provides the first (central) feedback on the quality of committed code changes. A central build system can adopt a fail-fast strategy. For example, if there are code merge conflicts, the build should fail, so team members can fix the code merge conflict. Traditionally build automation was done using Make files. We'll learn about this in the fifth module.

An automated build system typically includes:

- **Source Version Control System:** For example, Git, Gitlab, GitHub, Atlassian Stash, Subversion
- **Continuous Integration Server:** For example, Atlassian Bamboo, Jenkins
- **Code Quality Analysis and Reporting:** For example, Sonarcube
- **Artifact Repository:** For example, Nexus or Artifactory
- **Build Tools:** For example, Maven, Gradle, Grunt, NPM, Bower, Ant, Gulp
- **Static Analysis:** For example, FindBugs, CheckStyle, PMD, PHPMD.
- **Unit Test and Test Runner Frameworks:** For example, Junit and Karma

3.1.1 Advantages and Disadvantages of Build Automation



Automating the build process is the preliminary step to introduce Continuous delivery and DevOps concepts. With build automation tools, automation of simple and repeatable tasks can be achieved. Build automation saves a lot of time and money, which is not the case with manual builds as the cost of 'bad builds' is very high.

The advantages and disadvantages of build automation are listed above.

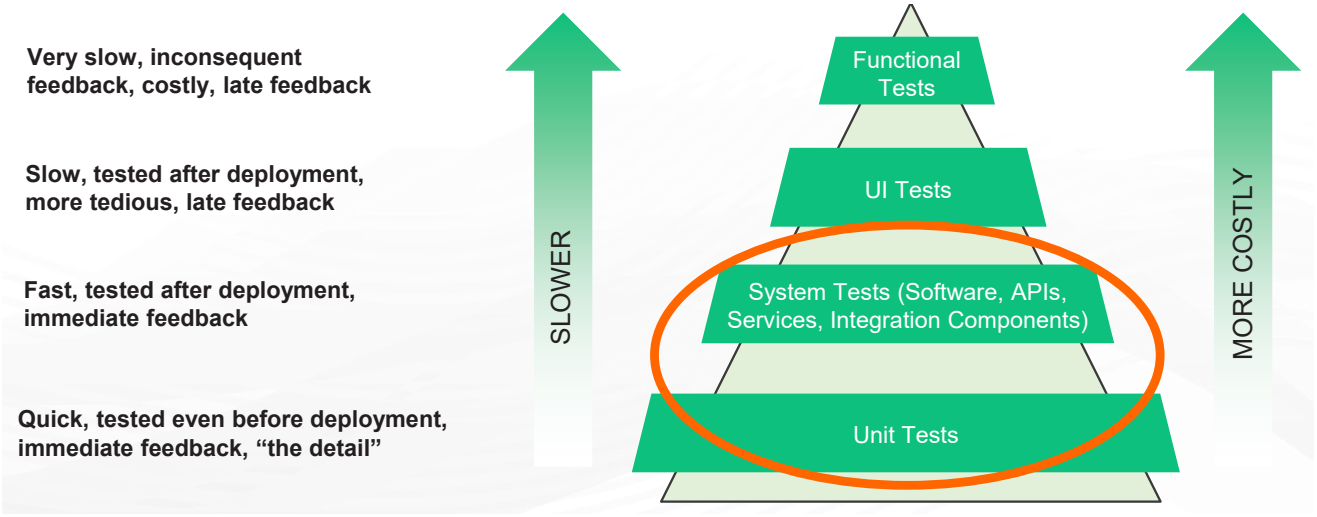
What did you Grasp?

1. Which of the following steps ensures the production of high-quality, bug-free software?

   A) Code compilation
   B) Automated tests
   C) Code building
   D) Code scanning

3.2 Automated Test

Optimization of software validation is done using automated tests.



Distinction between end-to-end tests and unit tests:

- End-to-End Tests: Tests the flow of the process and makes sure that functional requirements are met resulting in happy scenarios. End-to-end tests inform you when the code is failing.
- Unit Tests: Tests the functionality, the edge cases, boundaries, and exceptions. Unit tests inform you where the code is failing.

A reverse pyramid (as depicted in the figure) is an anti-pattern, also when you automate the higher level tests, it results in a very costly anti-pattern!

Example:

An application has 3 layers and each layer has 5 flows.

Test: To test all paths through the entire application and this would require:

- End-to-End Tests: 5^3 = 125
- Unit Tests: 5 x 3 = 15

Siloed testing is slow, very expensive, and (frankly) no longer a viable option.

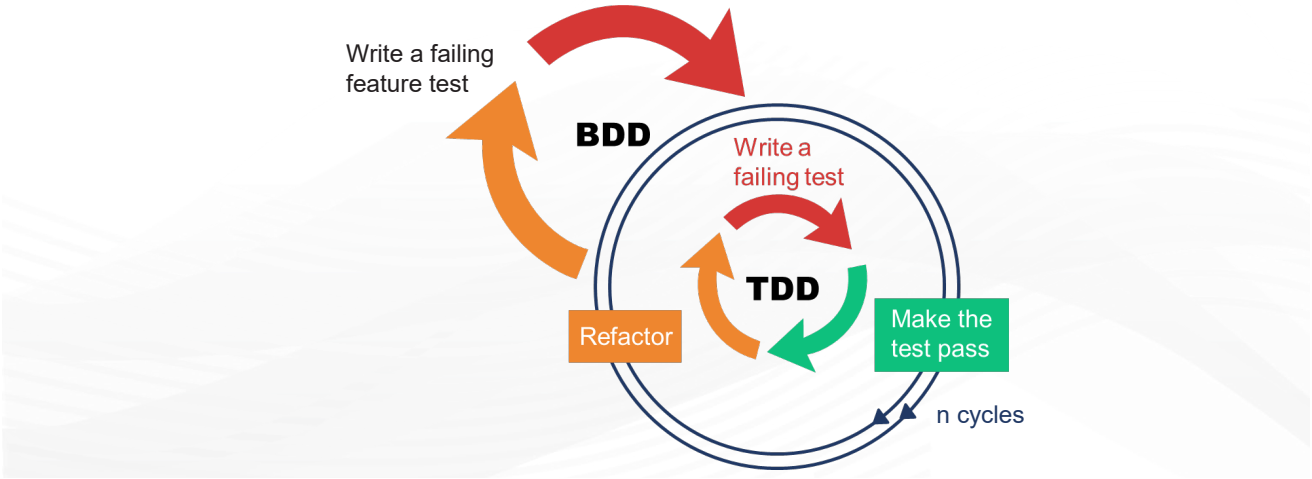
Testers and developers should work together to write, run, automate, and maintain tests which implies:

- Whole team testing
- Test code is production code

Tests should be written, before code is written. Once written, tests can be used over and over again and help keeping the system stable.

3.2.1 Automated Test - Specification and Verification Merged

Combination of Behavior Driven Development (BDD) and Test Driven Development (TDD) principles



Test automation can be adopted in the software delivery process. Combining Behavior Driven Development (BDD) and Test Driven Development (TDD) principles, testing becomes a first class citizen in the process and techniques replace traditional Waterfall process artifacts.

The process:

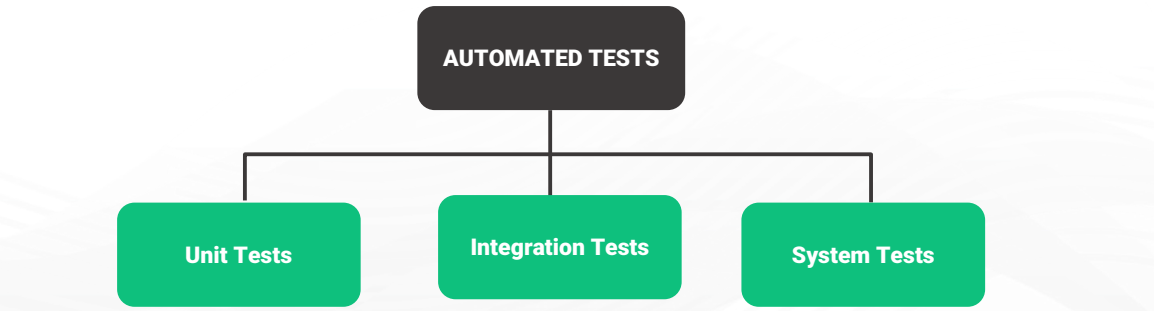
Given an Agile development process, the development cycle starts with a backlog of user stories. These user stories are refined with clear acceptance criteria.

- Now, a feature test can be specified using the BDD “Specification By Example” technique. This technique defines functionality using a strict syntax (Gherkin specification). The specified functionality (for example, the feature test scenarios) proves the acceptance criteria of the user story. The feature test specification is also a functional specification of the system.

- Now, when the feature test is executed, it will fail, because at this point, no code has been written to implement the user story.
- Next, the low level tests, like unit tests can be specified. Specifying these types of test is both a test specification, as well as a (detailed) design activity. Writing these type of tests upfront forces you to think about the (planned) software design.
- When these low level tests are executed immediately, they will fail, because at this point, no code has been written to implement the user story.
- Now, the software for (part of) the user story can be written.
- The implemented software can be tested with the specified low level tests. The goal is to ensure that all low level tests run successfully. This includes fixing bugs and refactoring the software code and/or test code.
- Once, all low level tests are successful, the feature tests can be executed. If these tests fail, the process of writing low level tests, adjusting the software code, fixing bugs, and refactoring is repeated until all feature tests and low level tests run successfully.

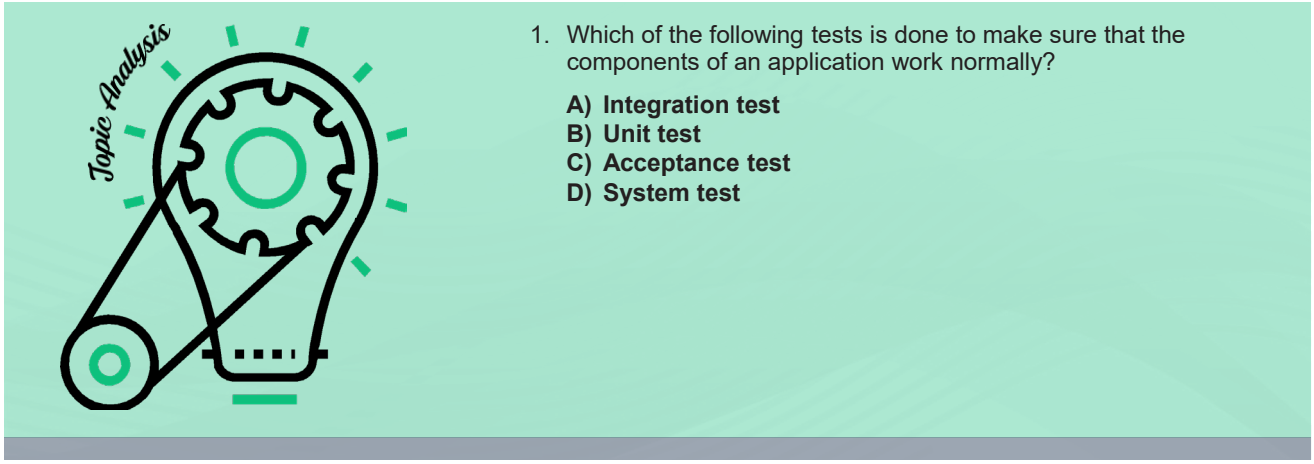
3.3 Types of Automated Tests

Automated tests are generally divided into multiple 'suites'. Different suites of automated tests are depicted in the illustration below.



- **Unit tests:** Unit tests are run first by developers, in some cases even before they add their changes to the repository. Unit tests normally test individual classes or functions. When those classes or functions need access to external resources, those resources are often faked as "mocks" or "stubs".
- **Integration tests:** Integration tests are a next level up from unit tests. Integration tests make sure that the modules that comprise an application work properly with each other. Ideally integration tests are run in environments that are similar to the production environment. For example, you'd want to make sure that if your application uses a database that the same database is available in your integration environment.
- **System tests:** Should test the entire system in an environment as close as possible to the real production environment.

What did you Grasp?

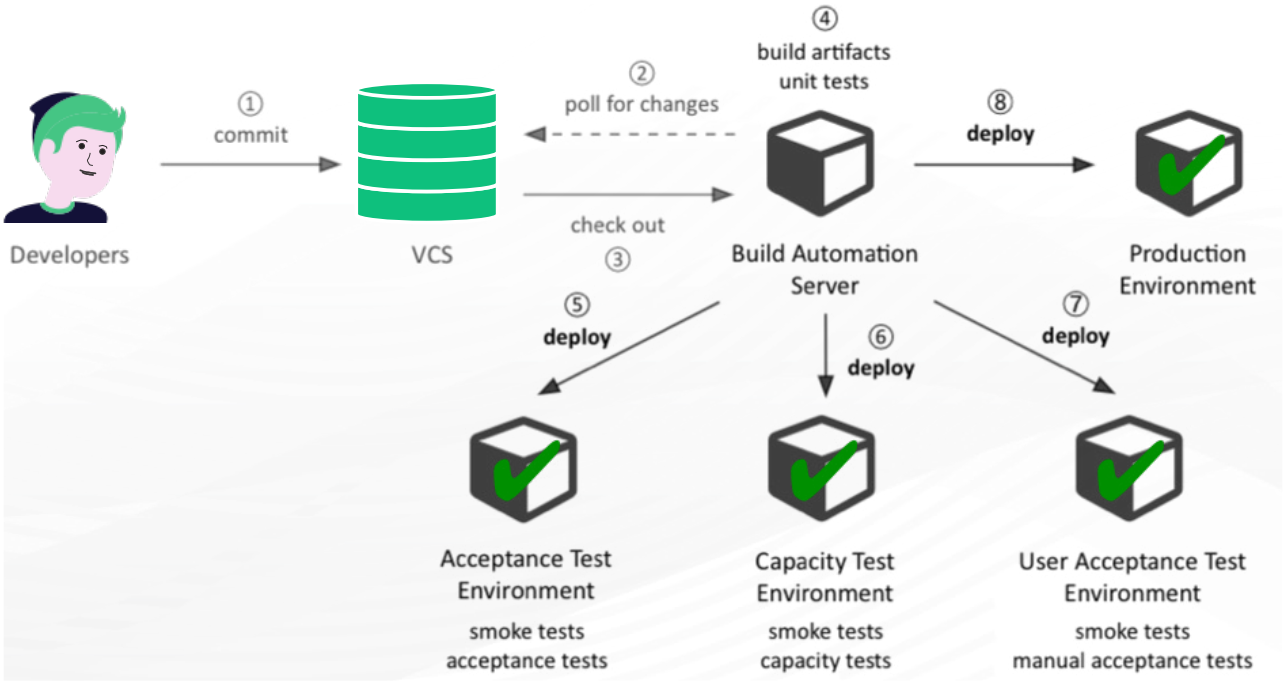


1. Which of the following tests is done to make sure that the components of an application work normally?

   A) Integration test
   B) Unit test
   C) Acceptance test
   D) System test

3.4 Automated Deployment

While build automation systems produce executable code, deployment automation systems deploy that executable code to an environment.



Source: Dynatrace

Application deployment implies:

- Installing applications
- Updating applications
- Configuring resources

- Configuring middleware components
- Starting/stopping components
- Configuring the installed application
- Configuration systems like load balancers, routers
- Verification of components
- Scaled to the enterprise

Providing a new functionality is always a traditional clashing point between development and operations as operations is accountable for continuity. Business units would like to have new functionality live in production as soon as possible, while Service Management is responsible for maintaining applications on a technical perspective and operations is responsible for hosting the application. Application deployment is where the two worlds meet! There is lots of confusion at the boundary of these worlds.

Developers are typically concerned with:

- Understanding requested features
- Designing components
- Writing code
- Writing test cases
- Compiling code
- Testing code

An Operator is typically concerned with:

- Installing server hardware and Operating Software (OS)
- Configuring servers, network, and storage
- Monitoring servers
- Responding to outages
- Appling security measures
- Maintaining disaster recovery protocols

The common issues associated with manual application deployment are:

- Inconsistency
- Slow deployment
- Not repeatable or reliable

- Outdated and extensive documentation is required
- Affects collaboration between teams

These reasons make the application deployment process an ideal candidate for automation.

According to Jez humble and Dave Farley, “A deployment pipeline is, in essence, an automated implementation of your application’s build, deploy, test and release process.”

There are a set of requirements that a deployment automation system needs to satisfy.

- The system must support packages, environments and bindings, which are important for any deployment.
- The system should support middleware systems.
- The system should have the ability to customize common deployment scenarios.
- Irrespective of the role, developers, administrators or deployment personnel should be able to use the tool with ease.
- The system must have the ability to scale, to accommodate many applications that have many users.
- There must be cross-platform support.

3.4.1 Benefits of Automated Deployment

Benefits of Automated Deployment

Cost effective	Activities reusable over and over again
Fast	Full deployments with a press of a button
Reliable	Tested and proven deployments
Maintainable	One standardized central point of access
Transparent, informative	Deployments with full transparency and informative
Auditable	Deployments can be audited easily
Secure	Automation ensures that deployments are fully secure
Accessible	Automated deployments are highly accessible
Repeatable	Reuse of deployment patterns
Consistent	Subsequent deployments identical over time
Scalable	Deployment patterns can be applied to many environment over time

A fully automated deployment process brings its own set of benefits, which are otherwise not possible with manual deployment. With automated deployment tools, engineers can focus more on developing

the software, than spending their valuable time in performing and validating a manual deployment. The time taken to initiate a fully automated deployment is almost negligible.

Automated deployments are not only repeatable but also configurable. Thus, if the software is to be deployed in an entirely new test environment or a new client installation needs to be done, the overhead of deployment can be avoided.

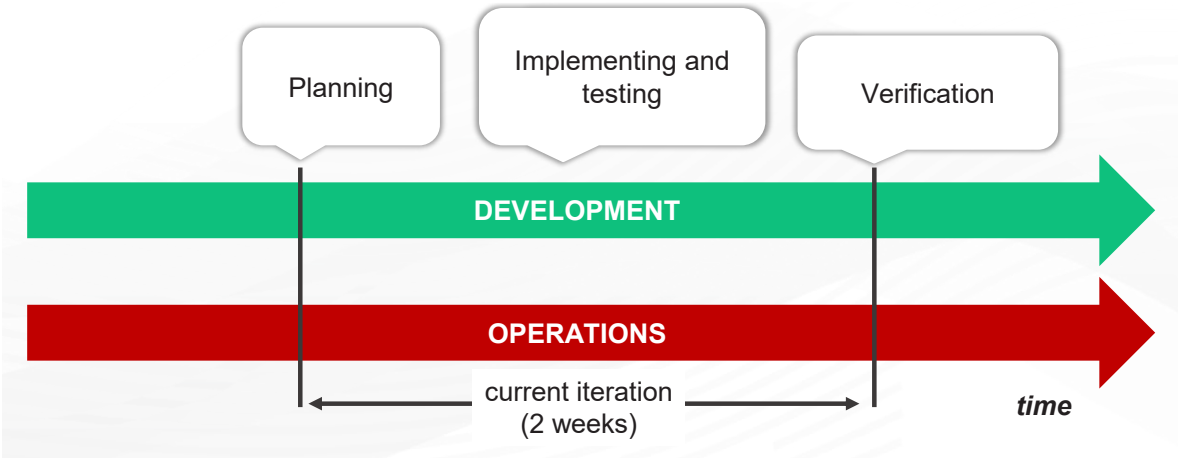
Most importantly, automated deployment does not require any technical knowhow, since anyone with basic knowledge can initiate the process at the click of a button. This eliminates dependencies on a subset of expert personnel.

The important benefits of automated deployment are summarized above.

3.4.2 Automated Deployment and DevOps Adoption

Automated deployment ensures that the team always has deployable code and working software at the end of each iteration.

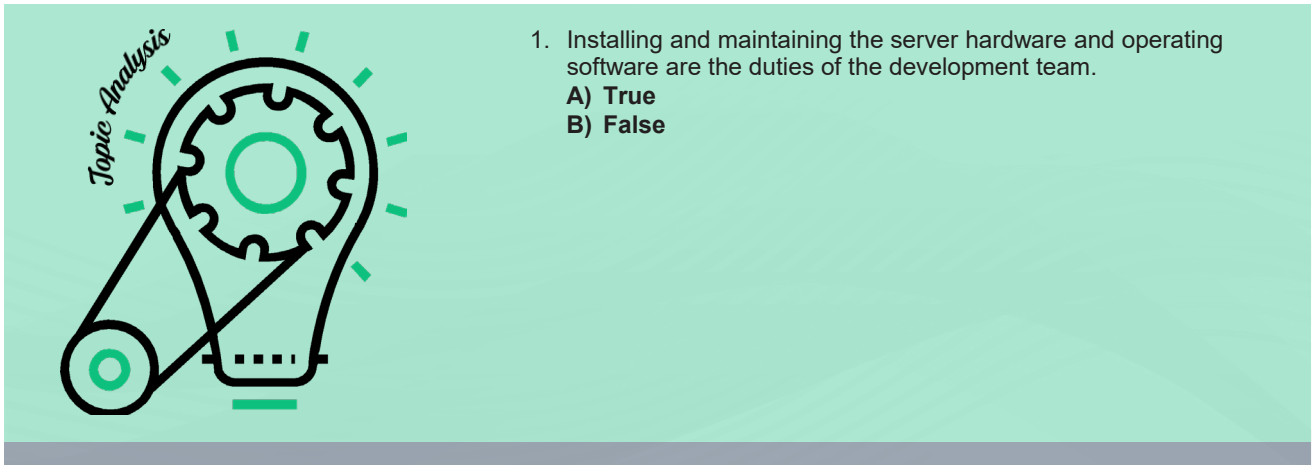
Deployment automation helps in bringing in changes as quickly as possible, both planned and unplanned, that the business demands.



Deployment automation allows us to think infrastructure as code, thus the entire deployment process becomes an engineering task, that the development teams can focus on. This brings the development and operation teams together to define and plan the tasks, even before any iteration begins. Once an iteration commences, the respective teams carry out the tasks and demonstrate to the customer. Then the teams sit together and brainstorm on the feedback and ways for improvement. By this, the teams always have the deployable code and working software at the end of each iteration.

Businesses continuously demand changes, and deployment automation is one of the keys that enable the teams to bring in changes. User feedback is received at frequent intervals and in early stages. Deployment automation also helps in developing and maintaining software in a lean way, with the features that the users want.

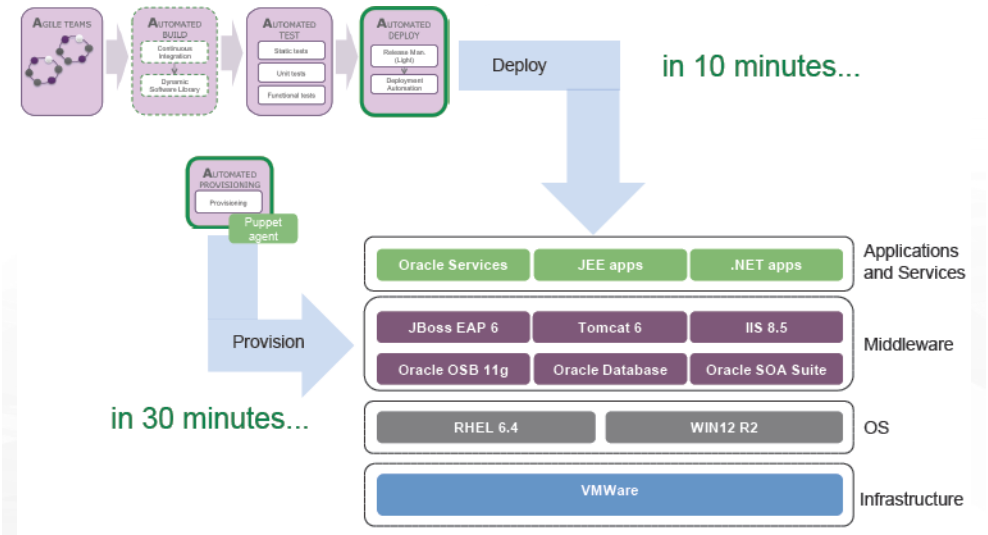
What did you Grasp?



1. Installing and maintaining the server hardware and operating software are the duties of the development team.
   A) True
   B) False

3.5 Automated Provisioning

An overview of automated provisioning:



Rather than provisioning and managing environment components, like an Application server manually, DevOps teams must be able to acquire new environment components on demand, fully automated. No manual setup, installation, configuration, and maintenance should be required.

Automated provisioning is defined as the fully automated delivery and maintenance of application environment components. Application environment components are the deployment target containers of the application. For example, a Database server or Application Runtime server. In a DevOps organization, automated provisioning can be the responsibility of DevOps Platform teams. Ideally, these teams deliver platform products, which are used autonomously by DevOps Business System teams. In other words, the Business System DevOps teams can use the platform products via fully automated self services. There is a clear separation of responsibilities.

Goals of automated provisioning:

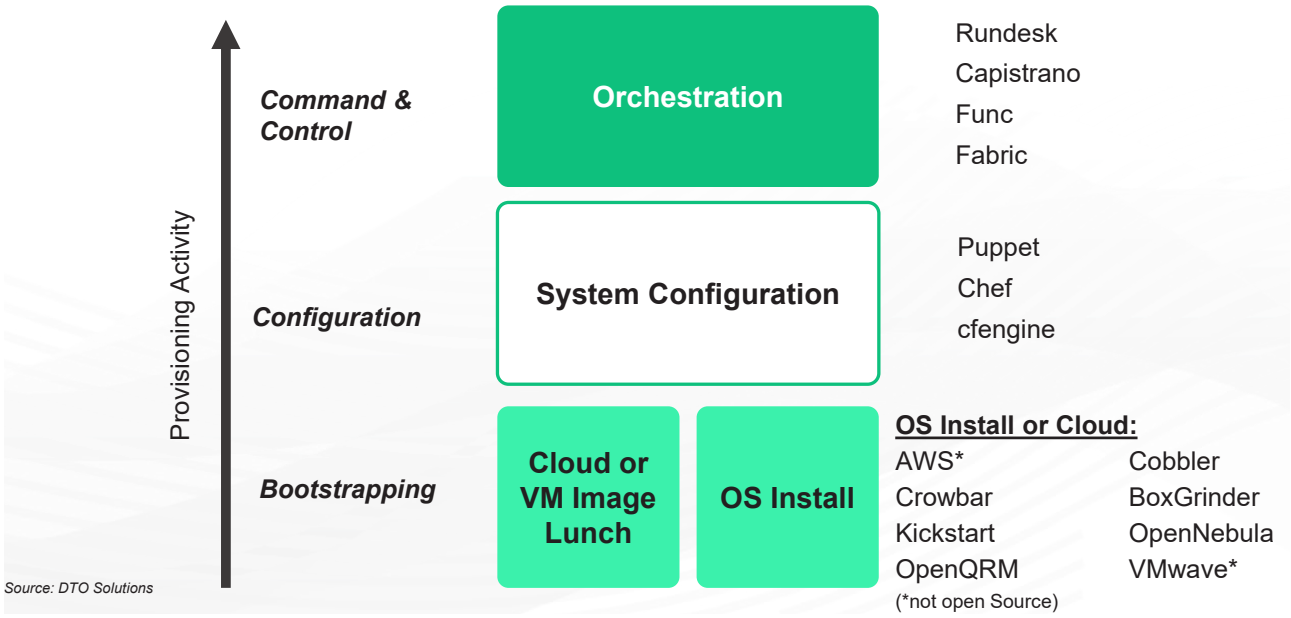
- Consistency across application environments
- Standardization of the platforms and reduction of the number of variations
- Control, traceability of system changes, no change is applied manually, and there is a single, central source of truth
- New environments are delivered within minutes/hours instead of weeks/months, hence, great speed
- Defect reduction

Continuous Delivery for Automated Provisioning

Automated provisioning requires a development rather than an operations perspective, because changes to infrastructure must be provisioned in a controlled manner, including extensive verification of these infrastructure changes. Infrastructure changes must be viewed as code (Infrastructure as Code) rather than change instructions bound to a change ticket. This new perspective implies that software delivery process based on the Continuous Delivery can be applied to manage infrastructure changes (for example, the platform products). This approach requires profound standardization of the platform product characteristics.

3.5.1 Tools for Fully Automated Provisioning

The image below represents tool options (some are open source) for a fully automated provisioning.

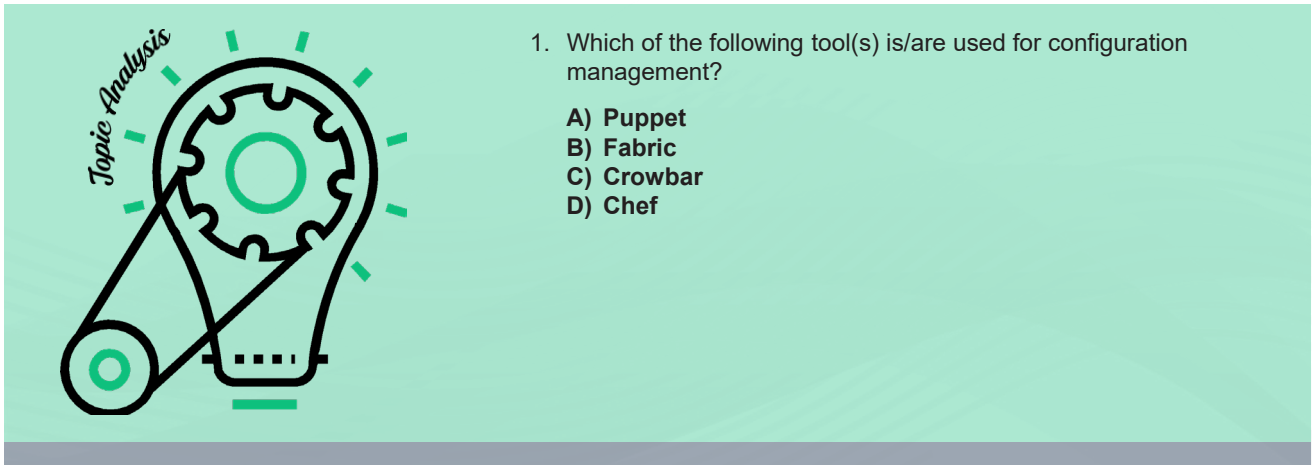


Fully automated provisioning doesn't involve a single technology. It is a set of tools that enable the achievement of end-to-end automation. The image above represents the toolchain approach.

The different layers of automated provisioning are as follows:

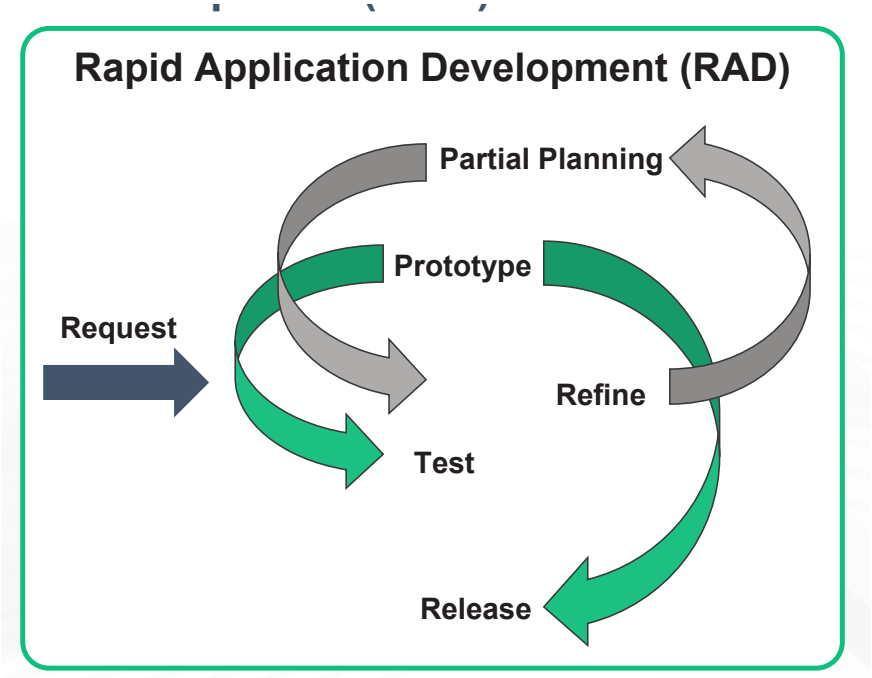
1. Automated OS install or Image Launch: Any specific version of the operating system has to be automatically installed across multiple machines, which involve physical or virtual machines (local or cloud-based). Cloud infrastructure allows to configurable virtual machine images that can be copied and these machines can be spun up as and when needed.
2. System Configuration: Once the OS is installed, we need to make sure that all the OS, network and security settings are in place. The correct version of the libraries and packages also have to be configured. This layer of automation enables us to automatically identify and correct and configuration drift. The common tools of configuration management are given above.
3. Orchestration: Orchestration refers to the coordination of actions that needs to be there across multiple tiers, such as web application, database, etc. and multiple servers have to effectively start, update and manage the integrated system. Similar to the configuration layer, the orchestration layer also needs to be specification-driven. The facilities needed to speed up ad hoc management and emergency troubleshooting are also provided by the orchestration layer.

What did you Grasp?



4. Overview of Rapid Application Development (RAD)

- Rapid Application Development (RAD) model:
- A type of incremental software development methodology that applies principles of iterative development and prototyping.
- Involves minimal planning and rapid prototyping. A prototype is a functionally equivalent component of the final product.
- Phases, such as analysis, design, build and test are divided into series of short, iterative development cycles..



Tell the participants that Rapid Application Development (RAD) model is a type of incremental software development methodology that applies principles of iterative development and prototyping.

Reiterate that in this model, analysis, design, build and test phases are divided into series of short, iterative development cycles.

4.1 Phases in RAD

The RAD process goes through the following phases:

Business Modeling	The information flow is identified between various business functions.
Data Modeling	Information gathered from business modeling is used to define data objects that are needed for the business.
Process Modeling	Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.
Application Generation	Automated tools are used to convert process models into code and the actual system.
Testing and Turnover	Test new components and all the interfaces.

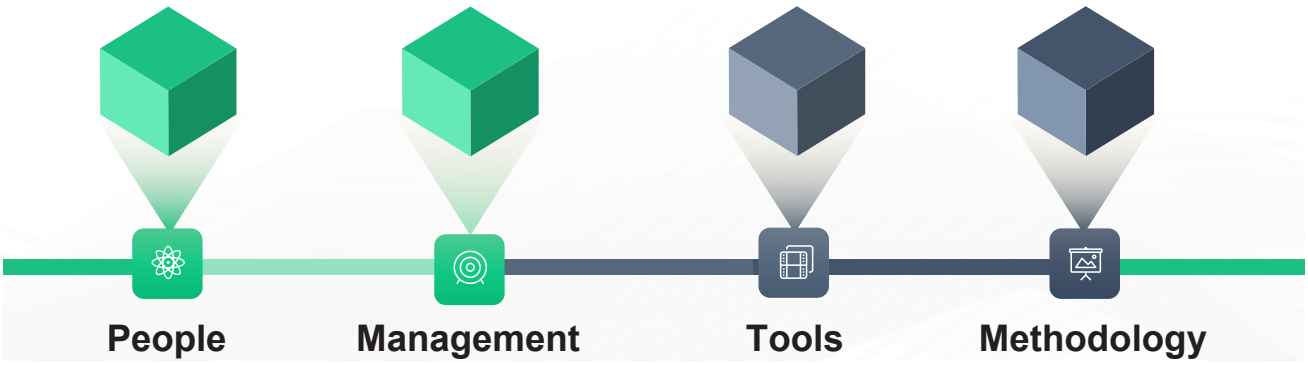
Let us learn more about the phases in RAD:

1. **Business modeling:** A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

2. **Data modeling:** The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model. Data objects that are required for the business are defined in this phase. Information collected during business modeling is used in this phase.
3. **Process modeling:** The flow needed for the business objective is generated in this phase.The Process model for change or enhancement to the data object is also defined during this phase.
4. **Application generation:** This is the phase of prototype development. It is in this phase that the actual system is created and coding is done by using automation tools. This converts the overall concept, process and related information into actual desired output. This output is called a prototype as it's still half-baked.
5. **Testing and turnover:** As the model is based on iterative development, prototype testing happens during every iteration and testing time is actually reduced. Data flow and the interfaces between all the components need to be thoroughly tested.

4.2 Essential Aspects of RAD

The essential aspects of RAD are:



It is critical that all the aspects mentioned above are adequate to ensure high-speed development. Development life cycles, which make up these aspects together as effectively as possible, are of the utmost importance. Various aspects of RAD are:

Methodology:

Important features of RAD methodology include:

- Combining the best available techniques and specifying the sequence of tasks that will make those techniques most effective.
- Using evolutionary prototypes that are eventually transformed into the final product.
- Using workshops, instead of interviews, to gather requirements and review design.
- Selecting a set of tools to support modelling, prototyping and code reusability, as well as automating many of the combinations of techniques.

- Implementing time-boxed development that allows development teams to quickly build the core of the system and implement refinements in subsequent releases.
- Providing guidelines for success and describing the pitfalls to avoid.

People:

In RAD there are 5 typical user roles:

- RAD Facilitator who plans and manage the project
- The Information specialists who translate the end users' needs.
- The End users
- The scribe who documents the sessions
- The developers.

Management:

It is important that the management is completely committed to RAD to enable high-speed development. Managers must motivate the IT team and end users for faster system building. Managers should also focus on 'Early Adapters', who see value in new methodologies and lead the way in making it practical to use. The best way for managers to introduce RAD is to start small. Original Construction Teams of two to four people should be established and their members should be thoroughly trained in the use of the tools and techniques. As these teams gain experience, they will be able to fine-tune the development lifecycle to improve its effectiveness in their environment.

Tools:

Examples of tools that can be used in RAD projects are Computer-Assisted Software Engineering (CASE) tools. These tools play a pivotal role in eradicating some problems that exist in other models of software development. For example, CASE tools can be used to develop models (using e.g UML diagrams) and directly generate code based on those models instead of hard coding.

What did you Grasp?

1. Which of the following statements about RAD is true?

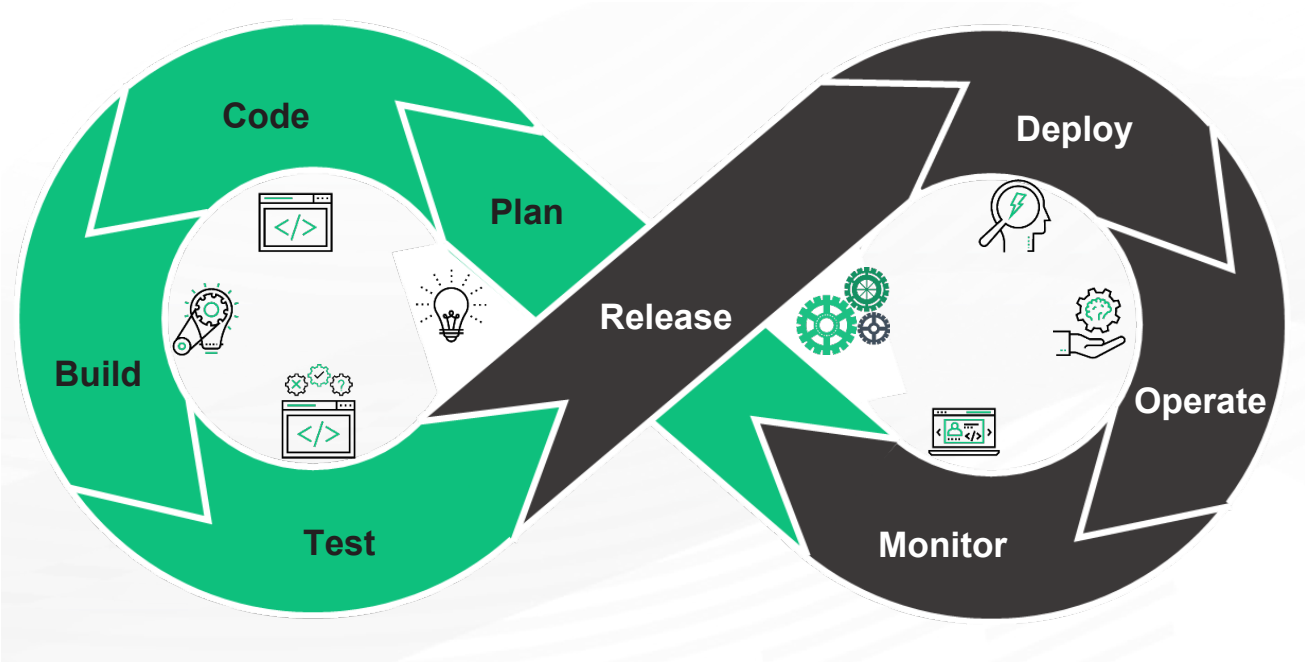
   A) RAD follows a sequential model and whole application is built and delivered at the end of project.
   B) RAD involves minimal planning and rapid prototyping.
   C) With RAD software is built over a long time.
   D) RAD follows an incremental approach, where multiple phases are divided into short iterations

2. When is the prototype generated in RAD?

   A) Business Modeling phase
   B) Application generation
   C) Testing and turnover
   D) Data modeling

5. Code Generation - An Introduction

- Code generation tools help in creating applications from scratch, that can run on a specific platform.
- Have the capability to generate bug-free, reliable basic set of code.
- There exists some code generators that can deliver code in accordance with predefined functionalities..



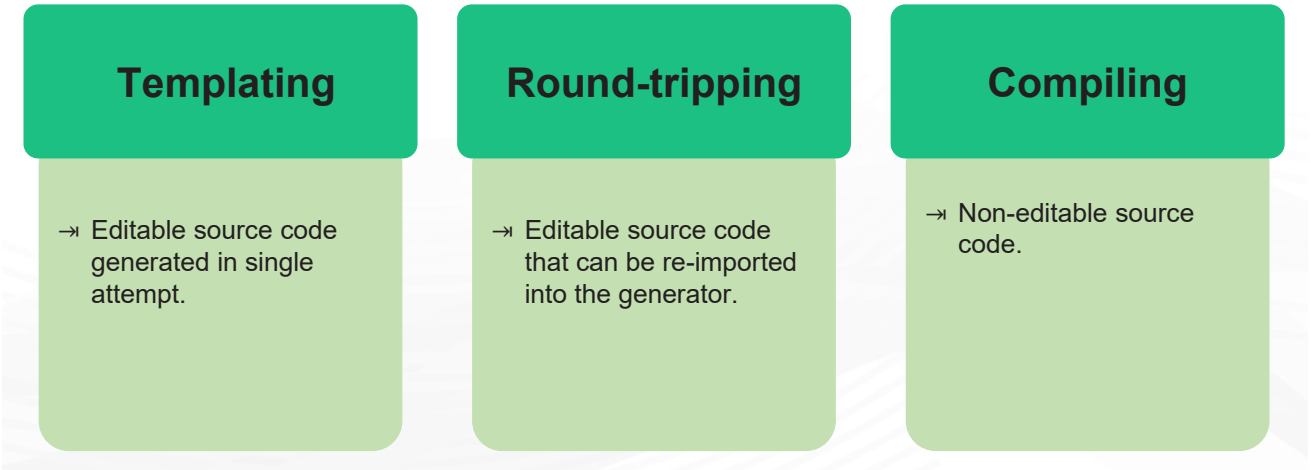
Code generators, also called application generators, have significant advantages. By taking a set of inputs provided by the developers, they start the development process. These generators can automate the development process, by creating bug-free and reliable basic code, as compared to manual code written by developers. Testing and bug-fixing thus become easier.

Some sophisticated code generators have the ability to generate complex web pages including reports, forms, and other complex features. Code quality is one of the major advantages offered by these generators, as the code will be clean and bug-free similar to the ones written by veterans.

Active code generators generate code and keep track throughout its lifecycle. Passive code generators' job gets over once the code is generated.

5.1 Categories of Code Generators

Three categories of Code Generators:

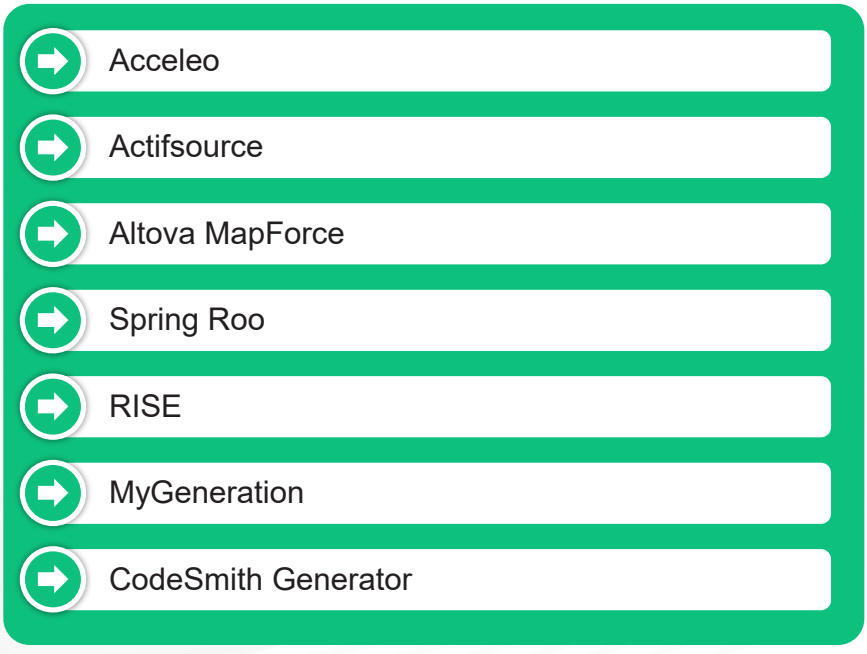


There are three categories of Code Generators. Look at the table and observe the characteristics of the categories.

Code generators can also be categorized as follows:

- **Database script generators:** Relational database models are created and updated by means of incremental scripts.
- **Application source code generators:** Code is generated in any programming language.

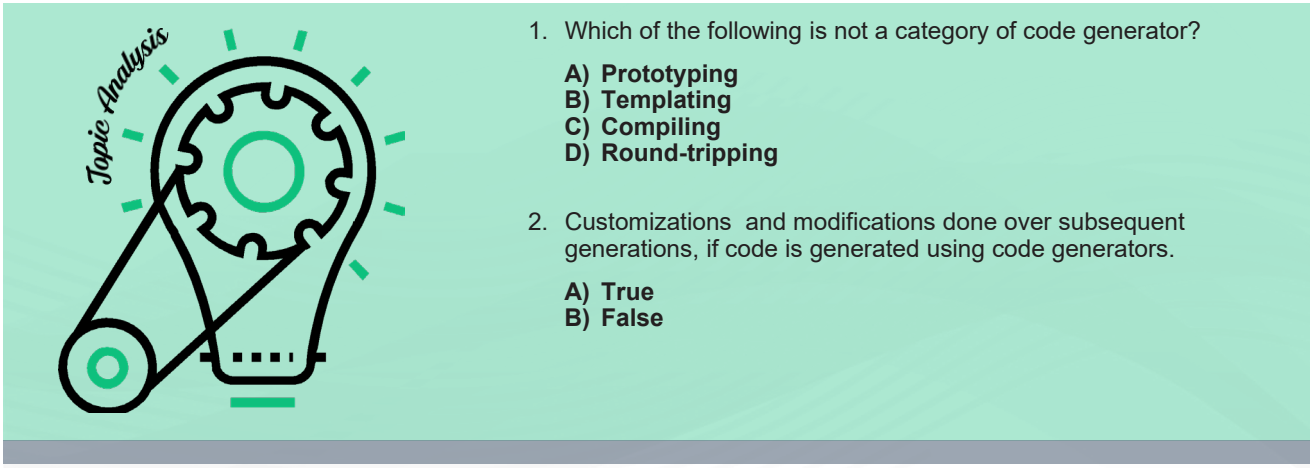
5.2 Common Code Generation Tools



Commonly used code generation tools are:

- **Acceleo:** An open-source code-generator, where the model-driven approach is used for generating applications. It helps in the incremental generation of code, the code can be generated, modified and regenerated without losing previous modifications.
- **Actifsource:** It is used in creating any graphical editor for the domain models. A comprehensive design and code generation tool, that follows a model-driven approach. Based on Eclipse IDE, this also offers multi-model support.
- **Altova MapForce:** The Graphical data mapping, conversion and integration, instantly transforms data or auto-generates data integration code.
- **Spring Roo:** The Offers more features than a code generator. Any change made to the entities, changes the code automatically, with just a set of commands.
- **RISE:** Based on the RISE-model, that contains the entire life-cycle of the data model, and code generator translates this into an incremental database specific scripts.
- **MyGeneration:** Microsoft .NET based tool.
- **CodeSmith Generator:** The template-driven source code generator that automates the creation of source code for any programming language.

What did you Grasp?



6. Model-Driven Architecture (MDA) - An Introduction

- A software design approach launched in 2001 by Object Management Group (OMG).
- System functionality is first defined as Platform Independent Model (PIM) through domain-specific language like CORBA, .Net, the Web etc.

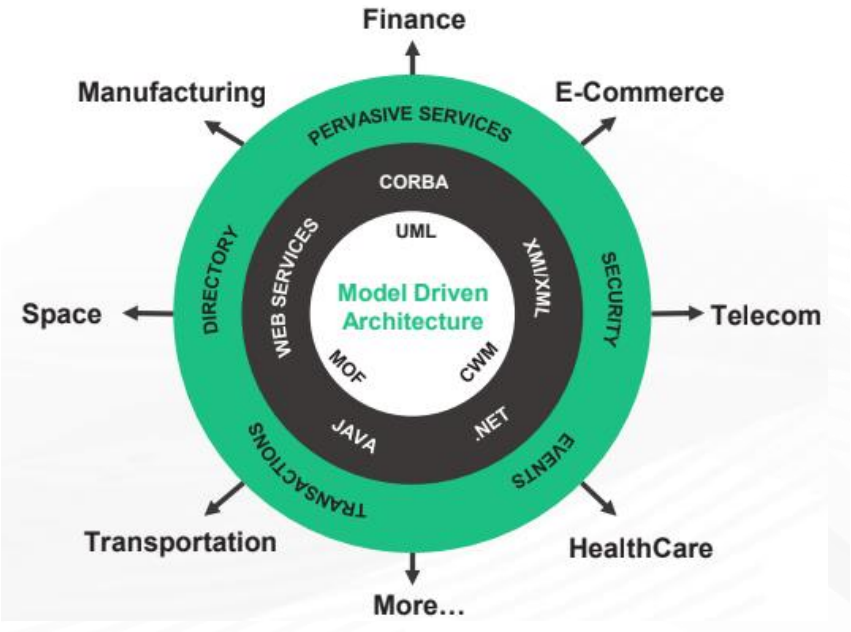
- PIM then translated to platform-specific models (PSM) through domain-specific or general purpose language like Java, C#, Python, etc.
- Translations between PIM and PSMs done using automated tools, like model transformation tools.
- MDA model is related to multiple standards, such as Unified Modeling Language (UML).

MDA is intended to support the model-driven engineering of software systems. Developed by the Object Management Group (OMG) in 2001. The MDA is a specification that provides a set of guidelines for structuring specifications expressed as models. MDA is an open and vendor-neutral architectural framework that leverages associated OMG standards (and models specifically) within the systems development lifecycle across various domains and technologies.

The overall process of MDA is documented in a document produced and regularly maintained by the OMG and called the ***MDA Guide***. The principles of MDA can also be applied to other areas like business process modelling where the architecture and technology neutral PIM is mapped onto either system or manual processes.

Note that the term "***architecture***" in MDA does not refer to the architecture of the system being modelled but rather to the architecture of the various standards and model forms that serve as the technology basis for MDA.

- A software design approach launched in 2001 by Object Management Group (OMG).
- System functionality is first defined as Platform Independent Model (PIM) through domain-specific language like CORBA, .Net, the Web etc.
- PIM then translated to platform-specific models (PSM) through domain-specific or general purpose language like Java, C#, Python, etc.
- Translations between PIM and PSMs done using automated tools, like model transformation tools.
- MDA model is related to multiple standards, such as Unified Modeling Language (UML).



MDA is intended to support the model-driven engineering of software systems. Developed by the Object Management Group (OMG) in 2001. The MDA is a specification that provides a set of guidelines for structuring specifications expressed as models. MDA is an open and vendor-neutral architectural framework that leverages associated OMG standards (and models specifically) within the systems development lifecycle across various domains and technologies.

The overall process of MDA is documented in a document produced and regularly maintained by the OMG and called the ***MDA Guide***. The principles of MDA can also be applied to other areas like business process modelling where the architecture and technology neutral PIM is mapped onto either system or manual processes.

Note that the term "***architecture***" in MDA does not refer to the architecture of the system being modelled but rather to the architecture of the various standards and model forms that serve as the technology basis for MDA.

One fundamental aspect of MDA is its ability to address the complete development lifecycle, covering analysis and design, programming, testing, component assembly as well as deployment and maintenance.

MDA itself is not a new OMG specification but rather an approach to software development which is enabled by existing OMG specifications such as the Unified Modeling Language (UML), the Meta Object Facility (MOF) and the Common Warehouse Metamodel (CWM). Other adopted technologies which are of interest are the UML Profile for Enterprise Distributed Object Computing (EDOC), including its mapping to EJB, and the CORBA Component Model (CCM). Since MDA is the topic of interest here, we will not be seeing the other OMG specifications.

The MDA approach is aimed at separating design from architecture.

Design: Addresses the functional requirements (use cases) of the application.

Architecture: Infrastructure for non-functional requirements like scalability, reliability and performance.

The advantages of MDA are as follows.

- **Portability:** The migration of existing functionalities to new environments and platforms, as per the business requirements, can be rapidly done.
- **Productivity and Time-to-Market:** Many of the tasks are automated and the development team can concentrate on the core logic. Thus productivity is increased and time to reach market becomes shorter.
- **Quality:** The consistency and reliability of the components built, contributes to the overall quality of the system.
- **Integration:** The MDA greatly facilitates the production of integration bridges with legacy and/or external system.
- **Maintenance:** Since the design is available in a machine-readable form, analysts, developers and testers can directly access the specifications of the system and thus maintenance process is simplified.

- **Testing and Simulation:** The models can be validated against requirements and tested against various infrastructures.
- **Return on Investment:** The investments done on tools generates greater value.

6.1 MDA - Concepts

The important concepts associated with MDA are:

1. System
2. Model
3. Model-driven
4. Architecture
5. Viewpoint
6. Platform
7. Platform Independence
8. Platform Model
9. Model Transformation
10. Implementation

The following are the important concepts related to MDA.

1. **System:** The context of MDA is the software system, either pre-existing or the one being built.
2. **Model:** A formal specification of the function, structure and behavior of a system in a given context. A model is usually represented by a combination of pictures and text, the formal notation being UML. In order for the specification to be formal, a model is expressed in XML in accordance with a well-defined schema (XMI). This is because, XML has a well-defined semantics associated with each of its constructs, which makes it more meaningful than a diagram with shapes and lines.
3. **Model-driven:** An approach to software development, where models are the primary sources of documentation, analysis, design, construction, deployment and maintenance of a system.
4. **Architecture:** The specification of the parts and connectors of a system, and the rules that define how the parts interact using the connectors. In MDA, inter-related models are used to represent the parts, connectors and rules.

5. **Viewpoint:** An abstraction technique for focusing on a particular set of concerns within a system. A viewpoint can also be represented by one or more models. Three default MDA viewpoints: computation independent, platform independent and platform specific.

MDA viewpoints

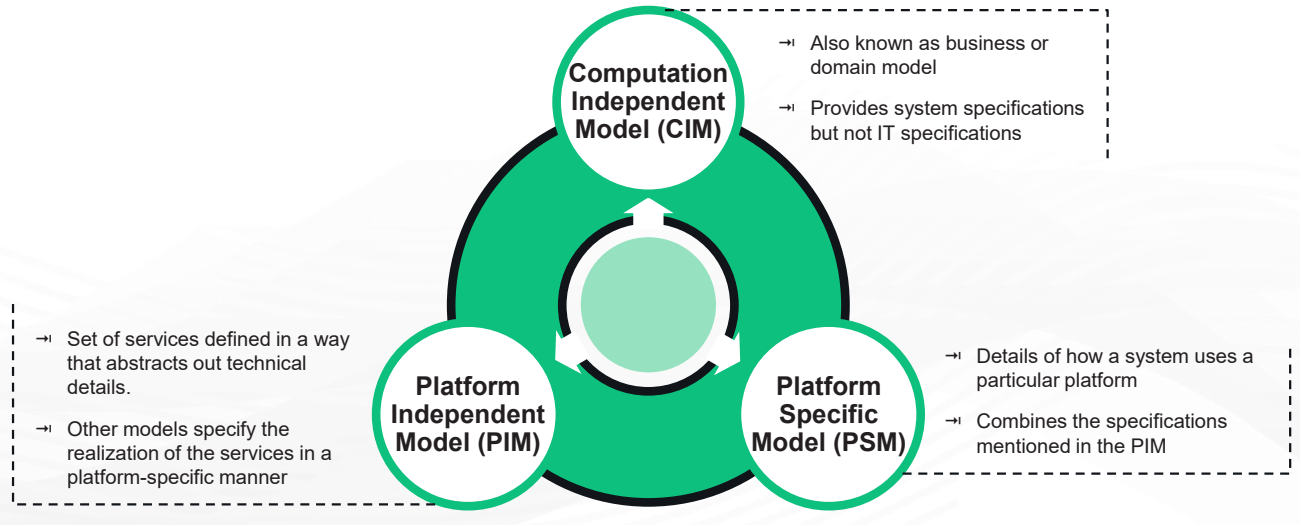
MDA specifies three default viewpoints on a system: computation independent, platform independent and a platform specific.

- The computation independent viewpoint focuses on the context and requirements of the system without consideration for its structure or processing.
- The platform independent viewpoint focuses on the operational capabilities of a system outside the context of a specific platform (or set of platforms) by showing only those parts of a complete specification that can be abstracted out of that platform.
- A platform specific viewpoint augments a platform independent viewpoint with details relating to the use of a specific platform.

6. **Platform:** A set of subsystems and technologies that provide a coherent set of functionality through interfaces and usage patterns. Common examples of platforms are operating systems, programming languages, databases, user interfaces, middleware solutions, etc.
7. **Platform Independence:** A characteristic exhibited by a model when it is expressed independently of the features of another platform. Independence is a measure of abstraction, that separates one platform from another.
8. **Platform Model:** A platform model provides the set of technical concepts representing the constituent elements and the services it provides. Platform model also specifies the constraints on the use of the elements and services by other parts of the system.
9. **Model Transformation:** The process by which one model is converted to another, within the same system is called Model Transformation. The Transformation combines the concepts of platform, independent model with additional details to produce a platform specific model.
10. **Implementation:** A specification that provides the necessary details for constructing a system and rolling it out for operation.

6.2 MDA Models

The three major models of MDA based on three MDA-Viewpoints are:



There are three major MDA models specified by MDA based on the three viewpoints stated in the previous slide. These models can perhaps more accurately be described as layers of abstraction since within each of these three layers a set of models can be constructed, each one corresponding to a more focused view of the system (user interface, information, engineering, architecture, etc.).

1. **Computation Independent Model (CIM):** It is termed business or domain model because it uses a vocabulary familiar to the subject matter experts. Provides exact specifications as to what the system must do but hides all the IT specifications. This is to ensure the model remains independent of how the system will be implemented. The role of CIM is significant in bridging the gap that exists between the subject matter experts and IT people who implement the system. In an MDA specification, CIM requirements should be traceable to the PIM and PSM constructs that implement them.

2. **Platform Independent Model (PIM):** This model exhibits its degree of independence so that it could be mapped to one or more platforms. To make this happen, a set of services is first defined in a way, that abstracts out technical details. Other models then specify a realization of these services in a platform-specific manner.

3. **Platform Specific Model (PSM):** A combination of specifications mentioned in PIM and additional details, which elaborates how a system uses a particular platform. A PSM is considered abstract, if it does not include details of implementing the platform. In these cases, it relies on other explicit or implicit models that contain these details.

What did you Grasp?

1. Which of the following options explain the function, structure and behaviour of the system being built?

   A) Viewpoint
   B) Model
   C) Platform
   D) Model transformation

2. The term 'Architecture' with reference to MDA, refers to the architecture of the system being built.

   A) True
   B) False

In a nutshell, we learnt:

1. Introduction to automation
2. The stages in a software delivery pipeline.
3. How a continuous delivery pipeline is structured.
4. The various steps involved in automating the build process.
5. The Rapid application development, phases involved, important aspects, advantages and disadvantages.
6. Introduction to code generation and types of code generators.
7. The Model-driven architecture, its concepts, models and the tools for MDA.

In this module, you have learned:

- The stages in a software delivery pipeline.
- How a continuous delivery pipeline is structured.
- Various steps involved in automating the build process.
- Rapid application development, phases involved, important aspects, advantages and disadvantages.
- Introduction to code generation and types of code generators.
- Model-driven architecture, its concepts, models and the tools for MDA.

Release Notes

B. TECH CSE with Specialization in DevOps

Semester three -Year 02

Release Components.
Facilitator Guide, Facilitator Course Presentations, Student Guide, Mock exams and relevant lab guide.

Current Release Version.
1.0.0

Current Release Date.
2 July 2018

Course Description.
Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications.

Bugs reported	Not applicable for version 1.0.0
Next planned release	Version 2.0.0 Feb 2019