


Student Referral Sheet – Faculty of Computing (Final Examination Jan-June 2025)

Lec01: 1.What is programming? Programming simply means writing Code to create an application. What is Software Engineering? Software engineering is the end-to-end process of applying good engineering principles and software development practices to produce quality software.		Lec 03 : 1.Git branching: the basics→ Default branch in git – master/main branch Why is branching needed ? +Branching lets developers work independently inside a single code repository. Problem without branching : merge conflicts Solution : let each other work on different branches. +Allows a developer to switch among different tasks easily. If there is a problem : Stop working on the current feature → Swith to main branch →make a hotfix branch → push the hot fix to remote repo → switch back to feature and continue + Git branches makes development more organized and manageable + Essential for Team Collaboration + try out new features or ideas without affecting the main codebase. + Useful for Solo Projects Too + Lightweight and Fast		Student ID : Module Code :	
2.practices come under software engineering? Analyzing user requirements and designing the software Communicating with all the stakeholders Choosing the software architectures, patterns, algorithms, coding techniques Quality Assurance (QA) Deploy→release→maintain. 3. Generic functionalities of software development <u>Common features :</u> User AUTH, database connectivity, scheduled jobs. <u>Issues :</u> donot need to write all the generic functionalities all over again when starting a new program Solution → FRAMEWORKS		Lec 04 : Web Application Architecture 1.Web site vs Web Application → Website provides static content which can be consumed by the end users → Web Application is designed for interaction with the end users.		2. Three tier architecture 1. Presentation tier/ layer 2.Application tier/ layer 3.Database tier/ layer > good choice for the development of web applications > Each tier/ layer runs on its own infrastructure > Each tier/ layer can be developed independently, in parallel by different teams > scaling up/down each independently without impacting the other tiers > Better security when compared with two tier architecture	
4. A framework is an integrated set of software artifacts that collaborate to provide a reusable architecture for a family of related applications. Objective → deliver faster development of an application. Developers do this by adding code to a framework, then personalizing it for their specific purpose. A software framework is an abstraction in which software, providing generic functionality, can be selectively changed (extended) by additional user-written code, thus providing application-specific software		2. How do we create a branch on git? git branch <branch_name> - just creates the branch based on your current branch git checkout <branch_name> - checks out the branch. git checkout -b <branch_name> - creates the branch, and switches to the new branch (checks it out) from your current branch immediately.		3. PRESENTATION LAYER – client side/ frontend Technologies – html ,css, js. angular, Vue.js, legacy technologies (jQuery,AJAX) Architecture - Model-view-controller (M-datalogic,V-userinterface,C-business logic) / Model-view-viewmodel (MVVM)	
5.How do Frameworks help with the Software Engineering Approach? Architectural Patterns →Design Patterns→Standards/ Best Practices(naming conventions, coding standards)→Tooling Support(package managers)→Testing Support		3. WORKFLOWS (BRANCHING STRATEGIES) Git is very flexible in how branching is done. While there is no single "correct" method. but some well accepted how-to branch 'recipes' are there. They are called workflows or branching strategies . popular workflows → GitFlow GitHub Flow GitLab Flow Trunk Based Development (goes with CI/ CD)		4. APPLICATION LAYER – server side / backend Technologies - Programming languages and frameworks (java,node.js,python,php), virtual machine, serverless computing. Backend and frontend communicates through API	
6.Framework VS Libraries (framework different from other library forms) : Default Behavior - framework behaves in a manner specific to user action before customization Inversion of Control -library, you call the functions. -framework, the framework calls your code. Extensibility - extend or customize a framework's behavior. Non-modifiable Framework Code - not allowed to change the internal source code		4. Gitflow Workflow →GitFlow is very comprehensive →Can become unnecessarily complex for small projects. →It is more suited to Enterprise grade software development efforts. → Requires enforcement of the practices for the proper functioning of the workflow.		5.API A software interface which facilitates communication between two or more applications. Abstracts away the complexities of application integrations Benefits – Developers don't need to build everything from scratch. Reduces development time and effort APIs improve connectivity between different systems, services, and teams. APIs allow integration with external platforms(eco systems) APIs enable automated workflows Types of API : + Representational State Transfer API (REST API) + Simple Object Access Protocol API (SOAP API) + Remote Procedure Calls (RPC) + WebSockets + GraphQL APIs Architecting the application layer: +Monolithic Architecture +Microservices Architecture +Service Oriented Architecture (SOA)	
7.WHY FRAMEWORKS? Focus on Requirements, Not Low-Level Details(database connections, routings) Frameworks provide a standard structure and ready-made components. Saves Time and Effort		5. Github Workflow : Workflow is GitHub Flow which can be used by anybody. GitHub Flow  <pre>graph LR Master[Master Branch] -- "create branch" --> Branch[Feature Branch] Branch -- "commit changes" --> Commit[Commit] Commit -- "Pull Request" --> PR[Pull Request] PR -- "get feedback" --> Review[Review] Review -- "test changes" --> Test[Test] Test -- "merge branch" --> Master</pre>		6.Monolithic Architecture Slow development speed high local complexity(service implementation). You cannot scale individual components separately. A failure in one module can potentially bring down the entire application. All modules are tightly coupled, switching to a new language, framework, or database becomes risky, expensive, and time-consuming. Harder to update	
8.Advantages → Improved Coding, Reusability & Easier Debugging Most popular frameworks have large communities .(can take support). Faster Development Built-in Caching(for faster loading) & Optimization. Less Code with Faster Methods Better Security		1.Create a new feature branch from the main (master) branch. 2. Make the necessary changes to the feature branch – continue until you think your feature is ready. 3. Once you think the feature is done, ask for feedback from the other developers – create a Pull Request (PR) for this. 4. Address their review comments received on the Pull Request (PR). 5. Merge your Pull Request (PR) to the main branch. 6. Delete your feature branch.		7.Microservice Architecture Microservices architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services. Benefits : Microservices are small, independent, and loosely coupled single team can write/ maintain a single service Services can be deployed independently Can scale each service independently High reliability Supports polyglot programming Problem : High global complexity (interaction between services) High infrastructure costs Debugging challenges	
9.Limitations → Learning the Framework Instead of the Language Restricted by Framework's Design(lost freedom of coding) Limited Customization- difficult to make changes even if you want your project loading unnecessary features Choosing the Wrong Framework Constant Updates and Changes Lack of Flexibility: Not all problems are suited to the same framework. erformance Issues: A mismatched framework can add unnecessary		6. Pull Requests (PR) Steps : Create a Feature Branch → Open a Pull Request → Code Review by Collaborators(They may approve it or request changes and improvements) → Update the PR with the requested changes. → Merge the PR Advantages: → Maintains code quality and consistency . → Encourages collaboration and peer review . → Helps track discussions and decisions around changes. → Ensures tested code enters the main branch.		7. Git branching: best practices → Create a branch for each feature. → only one person should work on one feature branch. → Feature branches should be short-lived.-delete after completion → Delete local and remote feature branch after merging to the master branch → Merge early and often to avoid merge conflicts. → Only production ready code should be in the master branch → Use a good naming convention to name your feature branches. Everyone should stick to the same naming convention. Ex : feature//task-name-in-lowercase-spaces-replaced-with-dashes feature/vishan.j/awesome-feature-x-to-our-app	
10. examples → Spring/ Spring Boot (Java), Express.js (with Node.js) , Laravel (PHP) ,.NET/ .NET Core (C# etc.), Django (Python) , Angular (Typescript) ,Vue.js (JavaScript)		8. Git general best practices 1. Make small, incremental changes. 2. Keep commits atomic.(forcus one part of the feature). 3. Commit often 4. Pull the changes in origin/main branch before you create a new branch from your local/main 5. Push local feature branch commits to Remote often 6. Use branches. 7. Write good commit messages. 8. Select one branching strategy (Git Workflow) and stick with it.		Dependency Injection (DI) is when a framework (like Spring Boot) automatically provides the objects your classes need, instead of you creating them manually. Modularity means breaking a big system into smaller, separate parts (modules) so each part can do one specific task. A force push (git push –force) is a Git command that overwrites the remote branch with your local changes, even if the remote has commits that are not in your local copy. It replaces history, which can be dangerous in a shared repository. Statelessness means the server does not remember anything about previous client requests. Access Token: short life (e.g., 15 minutes) o Refresh Token: longer life (e.g., 7 days) • When the access token expires, the client can send the refresh token to a secure endpoint to get a new access token. • Store the refresh token safely, like in an HttpOnly cookie, so attackers can't steal it using JavaScript.	
Lec 02 : 1. What is version controlling? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.		2.Working version control system : →Revert, compare, push, pull, merge. Two or more people : →maintain a log file. Should be an automated system. Ex : GIT, SVN, Mercurial		git checkout -b feature-branch # create and switch to new branch git add . (filename.txt) # stage changes git commit -m "Feature done" # commit changes git push -u origin feature-branch # . Push the feature branch to remote git checkout main # switch to main git pull origin main # get latest updates git merge feature-branch # merge branch into main # if conflict arises git status # see conflict files # Edit conflicting files manually git add <resolved-files> git commit -m "Resolved conflict" # Commit the merge with a message git push origin main # Push updated main branch to remote repository	
3. History on VCS→ 1 st Version Control System-Source Code Control System(SCCS) bell labs. 1st gen Local Version Control Systems → track changes of individual files can be edited locally by one user at a time. Problem : collaboration among large teams. Solution : new type of VCS were developed. 2nd gen Centralized Version Control Systems. Store files on a centralized repository in a network, multiple users can use the files concurrently (Follows a Client-Server approach). Problem : All should save their work in a centralized repository which require a central authority . Solution : Distributed Version Control Systems (3rd gen). No centralized authority needed. Have peer to peer approach(P2P) De-facto standard in Distributed VCS → Git		4. GIT: the de-facto standard today in vcs By Linus Torvalds(linux) Git supports non-linear development→ parallel branches.			
5.GIT BASIC CONCEPTS 2 repositories →local(each user),remote(all users). Git works on Commits -a snapshot of all changes in the local repository Each Commit has a unique Commit ID.(unique SHA-1). The most recent commit is called the Head →add – new file to the local repo, and do changes →commit file to local repo →pull changes form remote repo →push local changes to the remote repo					

<div>Lec 05 – Rest APIs</div> <div>1.How do you consume web resources?</div> <div>Web browser,web application, mobile application.</div> <div>What is an API?</div> <div>Examples:</div> <div>CORBA Implementations (Common Object Request Broker Architecture)</div> <div>SOAP (Simple Object Access Protocol)</div> <div>RPC (Remote Procedure Calls)</div> <div>REST (Representational State Transfer)</div> <div>What is REST? - an architectural style for creating web services by Roy Fielding in 2000.</div> <div>Why is REST Popular?(representational state transfer)</div> <div>→ Just an architectural style →6 constarints then rest api →No strict rules, very flexible →Simple to use and learn →Runs on HTTP Protocol</div>	<div>Lec 06 - REST APIS - AUTHENTICATION AND AUTHORIZATION</div> <div>Authentication is when an entity proves an identity</div> <div>Authorization is when an entity proves a right to access.</div> <table><tr><td>Authentication</td><td>Authorization</td></tr><tr><td>Verifies who the user is</td><td>Verifies what the user can access</td></tr><tr><td>Requires credentials (e.g., password, OTP, biometrics)</td><td>Checks access rights based on policies/rules</td></tr><tr><td>Happens before authorization</td><td>Happens after successful authentication</td></tr><tr><td>Typically uses an ID Token</td><td>Typically uses an Access Token</td></tr><tr><td>Follows OpenID Connect (OIDC)</td><td>Follows OAuth 2.0</td></tr><tr><td>Employee logs into email system</td><td>Employee is granted access to specific files/resources</td></tr></table> <div>Methods for auth in rest – http basic authentication, API Keys, JSON Web Tokens (JWT), OAuth 2.0 → OpenID Connect (OIDC)</div> <div>http auth- basic form of auth in API (username, password)</div> <div>-does not require cookies, session identifiers, Lightweight, not recommended</div> <div>API KEYS - unique generated key value for each first time user and send request with the key by user.(only a method of authentication), no expiration.not recommended.</div> <div>JSON WEB TOKENS (JWT) – transmit information between parties and JSON object. Digitally signed, therefore can be verified and trusted. Used for user authorization after authentication. Very popular!</div>	Authentication	Authorization	Verifies who the user is	Verifies what the user can access	Requires credentials (e.g., password, OTP, biometrics)	Checks access rights based on policies/rules	Happens before authorization	Happens after successful authentication	Typically uses an ID Token	Typically uses an Access Token	Follows OpenID Connect (OIDC)	Follows OAuth 2.0	Employee logs into email system	Employee is granted access to specific files/resources	<div>Infrastructure Automation Overview</div> <div>Definition: Using software/tools to automate the setup, configuration, and management of IT infrastructure.</div> <div>Purpose: Reduce manual tasks, minimize human error, and improve efficiency and reliability.</div> <div>SOFTWARE TESTING : (2 types) →functional test → non functional test</div> <div>FUNCTIONAL TEST TYPES →unit, interface, smoke, integration, regression, sanity, system, user acceptance.</div> <div>NON-FUNCTIONAL TEST TYPES → stress, volume, maintainability, security, scalability, compatibility, usability, performance, load</div> <div>Test Automation in CI/CD</div> <div>Performed as part of the CI/CD pipeline.</div> <div>Helps ensure code quality, reliability, and early bug detection.</div> <div>Types of Tests:</div> <div>1. Unit Testing</div> <div>Tests individual pieces of code (functions, methods).</div> <div>Ensures each unit works in isolation.</div> <div>2. Integration Testing</div> <div>Tests interactions between modules/services.</div> <div>Ensures components work together correctly.</div> <div>3. Regression Testing</div> <div>Run after bug fixes or new changes.</div> <div>Ensures previously working features still behave as expected.</div>
Authentication	Authorization															
Verifies who the user is	Verifies what the user can access															
Requires credentials (e.g., password, OTP, biometrics)	Checks access rights based on policies/rules															
Happens before authorization	Happens after successful authentication															
Typically uses an ID Token	Typically uses an Access Token															
Follows OpenID Connect (OIDC)	Follows OAuth 2.0															
Employee logs into email system	Employee is granted access to specific files/resources															
<div>2. Design principles</div> <div>→REST is designed to facilitate strong decoupling between the client and the server to facilitate internet-scale usage.</div> <div>→Request and response communication</div> <div>Client,server,resource.</div> <div>When a client requests for a resource, the server will respond with:</div> <div>1. Representation of the resource in the current state(JSON/XML).</div> <div>2. Hypermedia links() with relevant action</div> <div>6 CONSTRAINTS TO BE A REST</div> <div>Client-server architecture → Stateless → Cacheable →Uniform Interface →Layered system →Code on demand (Optional)</div>	<div>OAuth 2.0- It is an authorization framework/ protocol.</div> <div>Resource owner gives third party limited access to third party to their resources.</div> <div>Sol: OAuth allows this access to be scoped, temporary, and revocable.</div> <div>OAuth 2.0 handles authorization.</div> <div>OpenID Connect (OIDC) is built on top of OAuth 2.0 and adds authentication.</div> <div>OAuth Roles</div> <div>Resource Owner: user who authorizes an application to access their account.</div> <div>Client: The client is the application that wants to access the user's account</div> <div>Authorization Server: verifies the identity of the user , issues access tokens to the client application.</div> <div>Resource Server: The resource server hosts the protected resources.</div> <div>What is an OAuth 2.0 "Flow" (Grant Type)?</div> <div>specific way to get an Access Token depending on: user, kind of app, how secure you need the process</div> <div>why OAuth 2.0? →enables an application to obtain limited access (scopes) to a user's data with no password.</div> <div>Why use OpenID Connect →Accelerate Sign Up Process at Your Favorite Websites</div> <div>No need to maintain multiple usernames/ passwords.</div> <div>Minimize password security risks.</div> <div>Lec 09 - DEVOPS AND CI/ CD - AN INTRODUCTION</div> <div>Application Deployment, also known as Software Deployment, process of installing, configuring, updating, and enabling one application or suite of applications that make a software system available for use.</div> <div>Early ages : SDLC</div> <div>Fixes will be carried out for some time after the release to stabilize the product.</div> <div>The product is actually usable only after few days/ weeks after the release.</div> <div>DEVOPS(development operations): Bring the Developers, QA, IT Operations (IT Ops) teams together to work collaboratively on planning, building, testing and releasing software with shared responsibility for successful business outcomes.</div> <div>→Based on Agile approach to software development!</div> <div>It is defined as a software engineering methodology which aims to integrate the flow of development teams and operations teams by facilitating a culture of collaboration and shared responsibility</div>	<div>Lec10 - SOFTWARE QUALITY ASSURANCE (SQA)</div> <div>What is Software Quality? how well a software product conforms to the given requirements and meets the needs of its users. It involves both the software product itself as well as the processes used to develop it.</div> <div>Why is software quality assurance (sqa) important?</div> <div>Helps Create Better Software</div> <div>Improves User Satisfaction (recommend the product to others)</div> <div>Saves Money, Time, and Reputation (cheaper and easier to fix.)</div> <div>Follows Rules and Standards (industry rules and standards)</div> <div>Keeps Good Documentation</div> <div>THE SQA PROCESS :</div> <div>1. Identify Requirements</div> <div>2. Establish Quality Standards</div> <div>3. Plan SQA Activities (Make a plan for how quality will be checked)</div> <div>→ How problems will be reported and fixed</div> <div>→ How software files (artefacts) will be managed</div> <div>→ How you will review the process</div> <div>→ What testing methods will be used</div> <div>4. Execute the SQA Plan</div> <div>Now, follow the plan you created:</div> <div>→Review the software documents (designs, code, etc.)</div> <div>→Do testing many times (manual or automated)</div> <div>→Monitor product quality throughout development</div> <div>→Track defects (bugs)</div> <div>→Measure quality (e.g., code coverage, defect density)</div> <div>→Do root cause analysis (find out why problems happen)</div> <div>5. Continuously Improve the Process</div> <div>Learn from mistakes and reviews.</div> <div>Use data to make your quality process better.</div> <div>Follow the PDCA Cycle:</div> <div>→Plan – Make a plan to improve</div> <div>→Do – Try the new plan</div> <div>→Check – See if it worked</div> <div>→Act – Keep improving</div> <div>BUILDING QUALITY INTO THE SOFTWARE DEVELOPMENT PROCESS</div> <div>1. Define Clear Requirements – understand what need to be built</div> <div>2. Adopt Agile Methodologies - Agile methodologies like Scrum or Kanban allows for early detection and resolution of issues, leading to higher-quality software</div> <div>3. Design Reviews -Identify and solve defects before coding begins. Helps to prevent error when coding and saves time and effort.</div> <div>4. Implement Code Reviews - Regular code reviews help catch issues early in the development process. Facilitate knowledge sharing among team members.</div> <div>5. Automate Testing - including unit tests, integration tests, and end-to-end tests, helps identify defects quickly and consistently. Continuous integration and continuous deployment (CI/CD) pipelines can automate the testing process, enabling faster feedback loops.</div> <div>6. Use Version Control - Version control systems like Git enable teams to track changes to the codebase, collaborate effectively, and revert to previous versions if needed.</div> <div>7. Prioritize Security - threat modelling, code scanning, penetration testing, and security reviews</div> <div>8. Focus on User Experience (UX) – make the software hoe the user want it by collecting feedback and doing research.</div> <div>9. Document everything - Besides standard documents like SRS and test plans, record every decision.</div> <div>10. Continuous Monitoring and Feedback - Implement monitoring and logging mechanisms to track system performance, errors, and user behaviour in real time.</div> <div>11. Invest in Training and Education- training, practicing ,updated technologies makes a knowledgeable and skilled team.</div> <div>12. Promote a Culture of Quality -Foster a culture where quality is everyone's responsibility.</div>														
<pre>{ "id": 10, "name": "Jane Doe", "links": { "self": { "href": "/employees/10" }, "update": { "href": "/employees/10", "method": "PUT" }, "delete": { "href": "/employees/10", "method": "DELETE" }, "manager": { "href": "/employees/5" } } } rest GET ("href"/department/1/schedule) POST ("href"/devices/employees)</pre>	<div>4.TRUE REST VS. RPC (REMOTE PROCEDURE CALLS)</div> <div>1.</div> <div>GET /employees/10</div> <div>{</div> <div>"id": 10,</div> <div>"name": "Jane Doe"</div> <div>}</div> <div>//this is RPC</div>	<div>Dev – plan, create, verify, package</div> <div>Ops – Release, Configure, Monitor</div> <div>CORE DEVOPS PRINCIPLES</div> <div>1. Automation of the software development lifecycle</div> <div>→ Continuous Integration (CI)</div> <div>→ Continuous Delivery and Continuous Deployment (CD)</div> <div>→ Infrastructure Automation</div> <div>→ Test Automation - Shift Left Testing</div> <div>2. Collaboration and communication</div> <div>3. Continuous improvement - focusing on experimentation, minimizing waste for speed, cost, and ease of delivery.</div> <div>4. Customer-centric action- short feedback loops with customers.</div>														
<div>5.RICHARDSON MATURITY MODEL</div> <div>Level 0 -Swamp of POX (Plain Old XML/JSON)</div> <div>→ One single URI for everything</div> <div>- Always uses POST</div> <div>- No real concept of resources</div> <div>Level 1 - Resources Introduced</div> <div>- Multiple URIs for different resources (e.g., /employees, /orders)</div> <div>- But still only uses POST for everything</div> <div>Level 2 -HTTP Methods Used Properly</div> <div>- Uses GET, POST, PUT, DELETE, etc. appropriately</div> <div>- Now follows REST conventions for methods + resources</div> <div>Level 3 - HATEOAS Added</div> <div>- Server responses include hypermedia links guiding the client on possible next actions</div> <div>- Client doesn't need to be hardcoded with URI structure</div> <div>HTTP methods →GET,POST,PUT,DELETE</div> <div>HTTP status code →2xx – success response, 4xx- client error,5x- server error</div> <div>SOME FACILITATORS OF REST</div> <div>JSON (JavaScript Object Notation) – A lightweight human readable format for storing and transporting data as key-value pairs. Y` HAL (Hypertext Application Language) – An open specification that provides a structure to represent RESTful resources</div> <div>Best practices:</div> <div>→Use correct HTTP methods, headers and status codes</div> <div>→Accept and respond with JSON.</div> <div>→Always use proper authentication and authorization mechanisms</div> <div>→Give meaningful names to the URIs</div> <div>Ex: https://abcd.com/users/</div> <div>→Keep the API consistent</div>	<div>1.CONTINUOUS INTEGRATION (CI) :</div> <div>→Frequently push (integrate) code into the main branch</div> <div>→Each push automatically triggers a build of the application</div> <div>→And then it runs tests automatically to check if everything still works.</div> <div>Advantages: Work faster</div> <div>Catch bugs earlier</div> <div>Reduce stress at the end of a project</div> <div>Improve product quality from day one</div> <div>2. CONTINUOUS DELIVERY (CD):</div> <div>after code is built and tested automatically in CI. Controlled by the user by a trigger(button).</div> <div>Feature flags are used to disable/ hide incomplete features from affecting customers in production.</div> <div>3. CONTINUOUS DEPLOYMENT (CD) :</div> <div>every change that passes all stages of the production pipeline is released to the customers. There is no human intervention. Feature flags are an inherent part in Continuous Deployments</div> <div>BENEFITS OF CONTINUOUS INTEGRATION:</div> <div>Fewer bugs in production – Automated tests catch issues early.</div> <div>Easy release process – Integration issues are solved early.</div> <div>Less context switching – Developers fix issues immediately.</div> <div>Lower testing costs – CI servers run tests quickly and efficiently.</div> <div>Better use of QA – QA team focuses on improving quality, not just finding bugs.</div> <div>BENEFITS OF CONTINUOUS DELIVERY:</div> <div>Streamlined Deployment – No more spending days on release prep.</div> <div>Faster Feedback Loop – Frequent releases help get customer feedback quickly.</div> <div>BENEFITS OF CONTINUOUS DEPLOYMENT:</div> <div>Automatic Triggers – Every code change triggers deployment.</div> <div>Low-Risk Releases – Small, frequent changes are easier to fix.</div> <div>Continuous Improvement – Customers regularly receive new features and better quality.</div>	<div>Traditional approach (Waterfall SDLC) – each phase is completed before going to the next phase and SQA is done at the end of each phase.</div> <div>Requirement analysis, design, coding, testing, and maintenance to ensure that the software product is developed with minimal errors.</div> <div>Agile approach - The Agile approach to SQA is an iterative, incremental, and flexible approach that focuses on delivering software products in small increments.</div> <div>+Collaboration between the development team and the stakeholders for a seamless and quick development process.</div> <div>+Agile SQA is quite popular and focuses on self-organizing teams, continuous integration and testing, continuous delivery, and continuous feedback to ensure a high-quality software product.</div> <div>DevOps approach - This approach promotes collaboration between development and IT operations to ensure that the software product meets the requirements of the customers.</div> <div>DevOps is focused heavily on automation, such as continuous integration, continuous testing, and continuous deployment to deliver a high-quality product.</div> <div>This approach is great for projects that require frequent updates.</div>														