1．オブジェクト指向について（a. 特徴3つ＋その説明　b. 具体例）

　オブジェクト指向とは概念的なもので ”オブジェクト（モノ）” と ”操作” に分けてプログラムを組み立てていくことが出来るシステム構成の考え方の事である。開発するプロジェクトが持つサービスを役割毎にモノとして分類・作成し、出来上がったモノとモノを組み合わせることでプログラムを作成していくスタイルの事だ。

　例えば、従業員の勤怠管理システムもオブジェクト指向プログラミングで作ることが可能である。まず、作成する上で必要なモノの要素として、”従業員・勤務時間”などがモノとしてあげられる。
ここで指すモノは”人”のように形あるモノとは限らず、目に見えない概念的なモノも”モノとして定義”することが可能で、その場合”時間”もモノとして扱う事が可能である。何をモノとして扱うか扱わないかを決めるのは開発者だ。
　モノには1つ1つ、どんな要素で成り立つか・どんな動きをするかの”設計書（クラス）”がある。
先ほどの”勤務時間”でも、構成要素(データ)として〖誰が・いつ・何時間働いたか〗、ふるまい(処理)として〖入力・編集・削除〗で作成出来る。
出来上がったモノとモノを全て組み合わせる事で、1つの勤怠管理システムのプログラムを作成する事が可能なのである。

　オブジェクト指向を使用する特徴として、
〖効率よくプログラムを設計・開発できる〗
　プログラムを記述する際、同じコードをいくつか別の場所で必要になる場合に、オブジェクト指向開発でなければ、同じプログラムを繰り返し記述する事になる。だが、オブジェクト指向で１つモノとして定義・作成すると、内容を少し変えるだけで同じモノを用意する事が可能である。
〖不具合・エラーの原因特定のしやすさ〗
　不具合・エラーが起きた際、オブジェクト指向開発でない場合の原因特定は、容易ではない。だが、オブジェクト指向の場合は、モノ毎にふるまいが分かれているので ”どこのモノでエラーが出たのか” ”どんな操作・ふるまいをした時に起きたのか” が 、特定しやすいのである。
〖プログラムの仕様変更にも対応が容易〗
　例えばWebサイト上の、全てのボタンサイズなどの仕様変更があった際、オブジェクト指向でない場合はそれぞれを変更する必要がある。だが、オブジェクト指向開発だと、定義している内容を変更するだけで良いため、すべて仕様変更が1度の修正で対応することが可能である。

　これらの特徴があるため、オブジェクト指向で行う開発は設計・開発が効率的に進められ、開発後も修正が容易なのである。

引用、参考：https://youtu.be/75EHCXuR5BI

2．GitHub flowについて（i. リポジトリ　ii. main　iii. リモート　iv. ブランチ）

GitHub flowとは、〖GitHub〗の開発に使用されているワークフローのことである。
まず〖Git〗とは、バージョン管理システムの１つで、誰が・いつ・どこを編集したか、細かい履歴まで保存管理が出来る仕組みである。過去のファイルに簡単に戻れたりするので、1つのファイルを複数人で編集する場合でも、管理がしやすい事が特徴である。
Gitではデータを保管しておく貯蔵庫のようなものがあり、これを ”リポジトリ” という。
このリポジトリには、私たちのPC上の貯蔵庫としてあるローカルリポジトリと、開発者それぞれが作ったファイルをまとめて綺麗に貯めておく場所としてリモートリポジトリの2種類がある。
”ローカルリポジトリ” は、Gitを使うと開発者それぞれのPC上に自動で作られ修正・保存した履歴を貯めて置ける場所である。この修正して保存できるようにする行為を ”add”、編集内容をローカルリポジトリに貯める行為を ”commit”、ローカルリポジトリからリモートリポジトリに反映させる行為を ”push” という。
”リモートリポジトリ” は、サーバー上に一つだけ存在し、開発者がpushしたファイルを綺麗にまとめる事が可能である。この行為を ”meage” という。
〖GitHub〗 とは、このGitを使った開発を支援する仕組みを提供するサービスで、アカウント登録をすることによりリモートリポジトリ等のサービスも使用することが出来るのである。
先に述べたadd・commit・push等は、GitHubが開発したアプリケーション/GitHubDesktopで簡単に操作が可能なのである。
GitHubには ”mainブランチ” という軸のようなものがあり、そこから作りたい部分(ブランチ)を取り出してコードを記述し開発していく。このブランチを ”トピックブランチ”、それらブランチを取り出す行為を ”ブランチを切る” という。
このトピックブランチに記述したコードが良い書き方か・プロジェクトに沿った書き方なのかを開発者メンバーが ”Pull Request” で、確認・判断・アドバイスをしてくれる。ここで問題がなければ承認してもらい、mainブランチにmergeする事が可能ということだ。
また、他の開発者が同じようにmainブランチにmergeしているので、mainブランチの最新コードを自分のPC上に反映させるようにする。これを ”Pull” といい、新たにブランチを切る前に行うことで最新コードが書かれている状態で開発を進めることが出来るのである。

・自分が開発するトピックブランチを切り、コードを記述しcommit・pushする
⇩
・開発者メンバーにPull Requestを送り、確認・承認をしてもらう
⇩
・トピックブランチをmainブランにmargeする
⇩
・mainブランから最新コードをPullする
⇩
・mainブランから新しくトピックブランチを切る　　これらを繰り返す

つまり、この流れがGitHub Flowなのである。

引用、参考：https://atmarkit.itmedia.co.jp/ait/spv/1708/01/news015.html
https://qiita.com/tatane616/items/aec00cdc1b659761cf88

3．サーバーサイドエンジニア・フロントエンジニアの違いについて

一言で表すと、ブラウザ上でユーザーから見える・触れる部分の開発を行うのがフロントエンドエンジニア、サーバー側のプログラムの実装・データ処理などの目に見えない部分の開発を行うのがサーバーサイドエンジニアなのである。

”フロントエンジニア” はWeb開発において、ユーザーが見たり操作したりするフロントサイド側の開発を行う人材のことである。フロントサイド側とは、Webデザイナーが作ったデザインをもとにコードを記述しサイトを作成したり、必要な情報をサーバーに送ったり、くださいとリクエストが出来る様にするのもフロントサイドの役割である。
クライアントの要望を反映させたWebサイト開発を行うため、HTML・CSS・JavaScript (JQuery) 、WordPressなどを使用した実装スキル・Webサイト構築スキルが求められるのである。

”サーバーサイドエンジニア” は、ユーザーの操作やリクエストに対して情報・データの提供が出来るようにするサーバーサイド側の開発を行う人材のことである。
サーバーサイド側では、目に見えないシステムの裏側で動作するプログラムの開発や、データの検索・管理・保存・変更などの処理に使用するデータベースのシステム構築を行い、フロントサイド側からのリクエストに適切な情報を渡すなどの役割がある。
ユーザーからのリクエストにレスポンスを返す処理に必要な開発・保守を実行するため、JavaScript・PHP・Ruby・Python・Java・Cなどの言語を駆使して開発するスキルが求められるのである。

他にも収入などの違いはあるが、主にこういった違いがあるのである。

引用、参考：
https://engineer-shukatu.jp/column/archives/47255
https://engineer-shukatu.jp/column/archives/47227

4．AWSについて（その特徴）

AWSとは、Amazonが提供するクラウドコンピューティングサービスのことである。
クラウドコンピューティングサービスとは、インターネット環境（クラウド環境）で
〖サーバー・ストレージ・ネットワーク・データベース・ソフトウェア〗 などの様々な ”サービス” を利用する事である。
この技術が登場する以前は、サーバー機器の準備に莫大な初期費用が発生し、設置や管理でシステムを利用できるまでに時間がかかるなどの課題があった。だが、クラウドコンピューティングサービスの登場により、従来よりもスピーディかつ気軽に利用できるようになり、企業は様々なメリットを受けることができるようになったのである。
その特徴として、“サービス・コスト・セキュリティ・パフォーマンス力” などが挙げられる。
サービスでは、仮想サーバーを作成・利用出来る “Amazon EC2” や、データベースを簡単に作成出来る “Amazon RDS”、静的コンテンツを配信出来る “Amazon S3” などの100以上あるサービスを組み合わせて使用する事で様々なサービスを柔軟に利用する事が可能である。
コスト面では、利用時間や通信料に応じてサービスを利用した分だけ支払うので、無駄な出費を抑える事が出来る。
セキュリティでは、Amazonが自社のサーバーとして利用していたものを提供しているため、高いセキュリティレベルがそのまま活かされているのである。加えて常にセキュリティレベルの向上・更新を行っており、政府・金融機関・小売業者・医療機関などのセキュリティやコンプライアンス基準もクリアしているのである。
パフォーマンス力では、世界各国の地域にサーバーを設置しているためユーザーに最も近い設備からサービスが提供され、高いパフォーマンス力を保ったままサービスを利用することが可能である。ハードウェアの更新も定期的に行われているため、サーバーの質も常に高く保たれているのである。

これらのクラウドコンピューティングサービスをうまく活用することで、自社の業務効率化や生産性の向上を実現することが可能となり、今やAWSは企業にとって必要不可欠な存在になっていると言えるのであろう。

引用、参考：https://hnavi.co.jp/knowledge/blog/aws/

５．Dockerとは具体的に何ができる技術か　またDocker導入のメリットについて

Dockerとは、Docker社が開発するコンテナ型の仮想環境を構築・管理ができるOSS（オープンソースソフトウェア＝誰でも自由にソースコードの改変や再配布が認められている無償のソフトウェア）のことだ。
１つのハードウェアの中にDockerで別の環境・仮想環境を作り出し、プログラミング言語やデータベース、アプリケーション等を ”いつでも・どこでも・同じように” 使用できる技術のことである。
たとえば、Mac端末のMac OS(ホストOS)を使用しているユーザーがWindows専用ソフトを使用したい時に、仮想環境上にWindows OS(ゲストOS)を構築することで使用できるようになる。この仮想環境に含まれるアプリケーションをパッケージ化してまとめたものを、”コンテナ”と表す。
このコンテナを作成・構築するための設計書が ”Dockerイメージ” と呼ばれるものであり、異なるサーバーでも1つのDockerイメージを共有することで、誰でも、簡単に、同じ構成の環境を複数構築することが出来るのである。そのため、自身のPC・ローカルで開発していたものを本番のサーバーへ上げる時も、”移動がスムーズ” となり簡単に環境構築が出来るのである。
仮想環境構築では、コンテナ型の他にもハイパーバイザ型などの仮想マシンもあるが、コンテナ型はホストOSのカーネル(OSの中核)を利用するので、少ないリソースでのアプリケーション実行が可能となり、他と比べて実行に必要なメモリ・CPUリソースも余分に使用する事がなく、”軽量で処理が速い” という特徴もある。

Dockerには、〖自分の作りたい環境が作れる・簡単に同じ環境が作れる・他のサーバーへの移動の容易さ・軽くて処理が早い〗 といったこれらのメリットがある。
モダンな開発現場であれば、必ず取り入れている技術と言っても過言ではないので、Dockerは覚えておくべきスキルの1つなのである。

引用、参考：
https://hnavi.co.jp/knowledge/blog/oss/
https://youtu.be/lCMFOEpgRaU