Syp-Zusammenfassung:

**1. Projektcontrolling**:

Aufgaben: Planungsabweichungen erkennen, Abweichungstendenzen erkennen.

Voraussetzungen: geeignetes Projektreporting, Methoden zur Darstellung und Bewertung des Projektfortschritts.

Kostenkontrolle: zeigt wie Finanzmittel im Projektverlauf eingesetzt werden.

Arten: Kostenhistogramm, Kostensummenkurve

Earned-Value-Analyse: dient durch Termine-Fertigstellungsgrad-Kosten integriertes Projektcontrolling aufzubauen.

Schritte dieser Analyse:

1. Projektsituation wird hinsichtlich der Ist-Kosten und Grad der Fertigstellung erhoben.

2. Geplante Kosten werden mit tatsächlichen Grad der Fertigstellung bewertet.

3. Berechnung der Indikatoren für Projektfortschritt. Kostenabweichung, Planabweichung, Performance-Indikatoren werden ermittelt.

4. Prognosen werden erstellt.

Formeln:

Earned Value: gibt Wert auf Basis der Plankosten an, der bereits erarbeitet wurde. Plankosten \* Fertigstellungsgrad

Kostenabweichung: (Earned Value – Ist-Kosten) / Earned Value

Planabweichungen:(EV – Plankosten)/Plankosten

**2. Docker:**

1. Was ist Docker:

Anwendungen werden in Containern ausgeführt.

Container bauen aufeinander auf, und kommunizieren.

Benötigt kein vollständiges Betriebssystem für jeden Container(Unterschied zu VM)

Ressourcenschonend.

Begriffe:

**Docker-Host:** System, auf dem der Docker-Daemon und

der Docker-Client läuft.

**Docker-Client:** Wird von Binary Docer repräsentiert.

**Dockerfile:** Textdokument mit Beschreibung, wie ein Image erstellt werden soll.

**Docker-Image**: Schreibgeschützte Vorlagen, die eine

Anwendung mitsamt allen Abhängigkeiten zusammenfasst.

**Docker-Container**: Sind lauffähige und nicht schreibgeschützte Varianten der Docker-Images.

**Docker-Registry**: Enthält Docker-Images

**Docker-Compose**: Tool, um mehrere Docker-Container mit

einem Befehl zu administrieren.

**Kommandos:**

**Docker ps**: Listet laufende Container auf.

**Docker stop**: Stoppt Container

**Docker start**: Startet Container

**Docker rm**: Löscht Container

**Images:**

Eingefrorener Zustand eines Containers.

Container basiert auf Image.

Images bauen einander auf.

**Image Befehle**:

**Docker Images**: Listet Images auf

**Docker pull Name**: Lädt das spezifizierte Image vom DockerHub

**Docker rmi Name**: Löscht Image

Docker Commit: Image aktualisieren/ändern

**Dockerfile:**

Beschreibt Aubau eines Docker-Images.

Ersetzt manuelles Absetzen

Befehle:

FROM: gibt das Basis-Image für unser neu zu erstellendes Image an.

LABEL: Fügt Meta-Daten zu einem Image hinzu.

RUN: Führt Befehl in der Shell aus.

**Docker Compose:**

Ein Werkzeug, welches erlaubt, eine Gruppe von Docker-Containern mit nur einem Befehl zu administrieren.

List die Dockerfiles der einzelnen Images, um diese zu erstellen und die Container zu starten.

Definitionen:

image: Definiert, auf welchem Image der jeweilige Container basieren soll.

Build: relativer Pfad zu einem Ordner wird mit Dockerfile angegeben und dann das Image erstellt.

Ports: Öffnet Ports des Containers und verbindet sie mit Ports auf dem Host.

Befehle:

docker-compose up: Erzeugt und startet alle Container

docker-compose build: Baut alle Container

docker-compose: Listet die Container

**GIT:**

Versionsverwaltung: System, welches die Änderungen an Dateien protokolliert.

Änderungshistorie entlang Zeitachse

Ermöglicht Zugriff auf bestimmte Versionen

Sinnvoll bei Textdokumenten

**Lokale Versionsverwaltung**:

Funktionsweise: Ordner kopieren

Vorteile: Einfaches System

Nachteile: Fehleranfällig, Zusammenarbeit schwer möglich, Risiko Harddisk-Crash

**Zentrale Versionsverwaltung:**

**Funktion:** Basiert auf zentralem Server -> verwaltet versionierende Dateien. Clients holen aktuelle Version vom Server (checkout) und teilen Änderungen dem Server mit (commit).

Bsp: CVS, Subversion

Vorteile: Mehrere Beteiligte können mit gleicher Codebasis arbeiten, Berechtigungsverwaltung, Administration von einer zentralen Basis

Nachteile: Risiko Ausfall, Risiko Plattencrash

**Verteilte Versionsverwaltung:**

**Funktionsweise:** Jeder Benutzer erhält eine vollständige Kopie vom Repository. Im Fehlerfall kann eine solche Kopie auf den Server zurückkopiert werden. Zusammenarbeit zw. Remote-Repositorys möglich

**Ziele von Git:**

Geschwindigkeit, Einfaches Design, Nicht-lineare Entwicklungen (tausende Parallelentw.), Dezentrale Struktur, Effizienz

**Zustände von Git:**

**Modified**: Datei wurde geändert, aber noch nicht in die lokale Datenbank eingecheckt

**Staged**: Geänderte Datei wurde für den nächsten commit vorgemerkt

**Commited**: Daten wurden in der lokalen DB gespeichert

**Befehle**:

Git Config: Konfiguration auf mehreren Ebenen:

Git Repository anlegen:

1. Git clone [url]: Klonen eines bestehenden Repos

2. Git init: Neues Repo erstellen

Dateien hinzufügen: Git add

Anzeigen des Status: git status [-s] [--short]

Dateien ausnehmen die nicht versioniert werden sollen: .gitignore

Offene Änderungen anzeigen(Änderungen die nicht zum Commit gemerkt wurden): git diff

Vorgemerkte Änderungen mit aktuellen Repo-Stand vergleichen: git diff –staged

Staging-Area einchecken: git commit

Kommentar ergänzen: git commit -m

Staging-Area überspringen: git commit -a

Dateien löschen: git rm [-f] [--cached]

Git remote: Erlaubt arbeiten mit Remote-Repos

zB.: -v: Remotes auflisten, add: hinzufügen