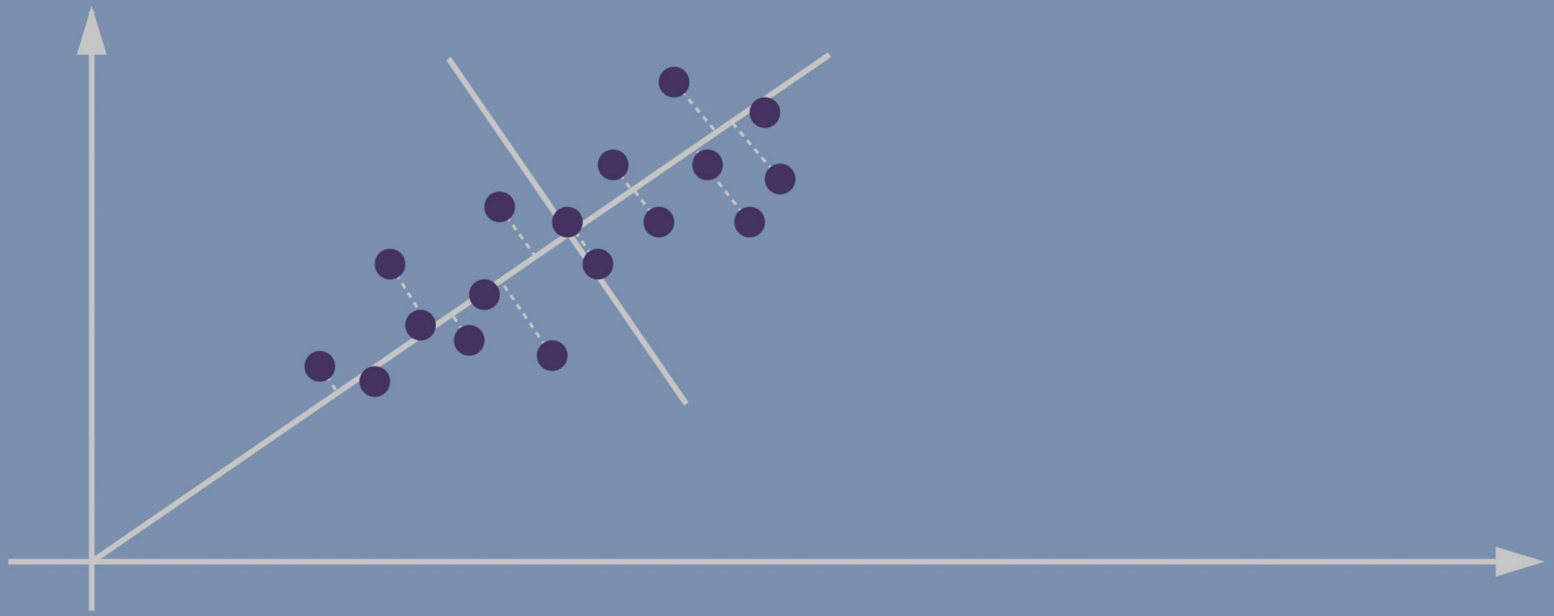


PCA-06-scaled.jpg (2560×1051) (perfectial.com)

Lecture 5: Scalable PCA for Dimensionality Reduction

COM6012: Scalable ML by Haiping Lu

YouTube Playlist: https://www.youtube.com/c/HaipingLu/

Week 5 Contents / Objectives

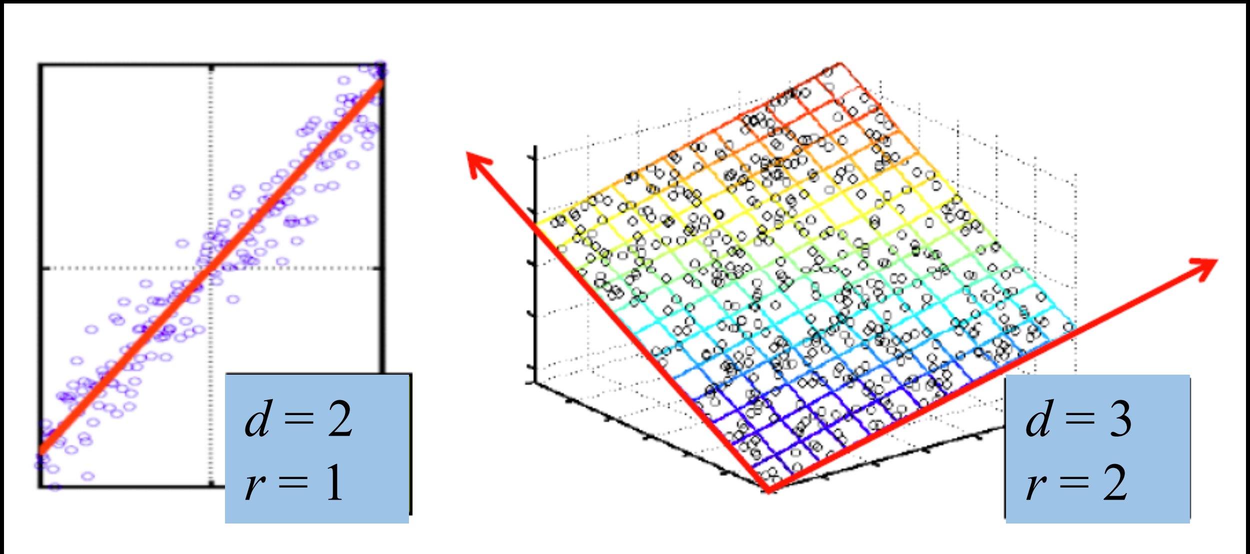
- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Week 5 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Motivation of Dimensionality Reduction

- Raw data: complex and high-dimensional
- **Assumption**: they lie on a low-dimensional subspace
  - Axes of this subspace → representation of the data
  - Simpler, more compact, showing interesting patterns

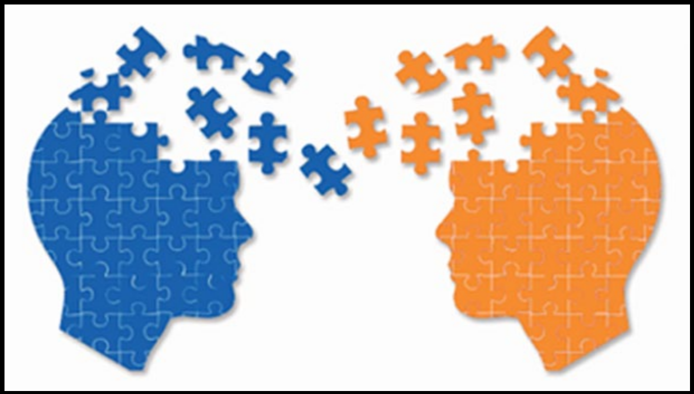


Utilities of Dimensionality Reduction

- Discover hidden correlations/topics
- Remove redundant/noisy features
- Interpretation and visualisation
- Easier storage and processing of the data

owners-icebergs-blog-image-300x300.jpg (resettogrow.com)

1*KvKIx9OnlxdoTfNxWKAY_g.jpeg (480×320) (medium.com)



Interpreting and Translation Blog: Image (wordpress.com)

PCA → Variance Maximisation

- Input: n d-dimensional data points
- PCA algorithm
  - $\mathrm{X}_0 \leftarrow n \times d$ data matrix, data point → row vector x_i
  - X: subtract mean x from each row vector x_i in X_0
  - $\Sigma \leftarrow \mathrm{X}^\mathrm{T}\mathrm{X}$: Gramian/scatter matrix for X
  - Find eigenvectors and eigenvalues of Σ
  - U ($d \times r$) ← the top r eigenvectors (PCs)
- PCA features for y: $\mathrm{U}^\mathrm{T}\mathrm{y}$ (dimension: $d \rightarrow r$)
  - Zero correlations, ordered by variance

Scalability Problem of PCA

- Input dimensionality → scatter matrix
  - Images: 100 x 100 → 10^4; 1000 x 1000 → 10^6
  - Scatter matrix Σ is of size d^2
    - $d = 10^4$ → Σ size 10^8
    - $d = 10^6$ → Σ size $= 10^{12}$

- Alternative: Singular Value Decomposition (SVD)
  - Efficient algorithms available
  - Often need just top r eigenvectors

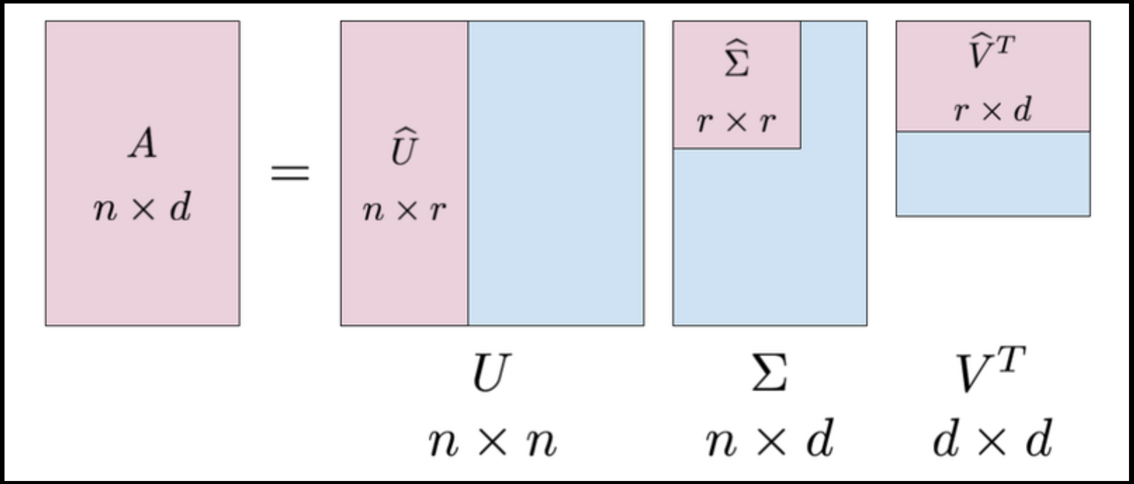
Week 5 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Singular Value Decomposition (SVD)

$$A_{[n \times d]} = U_{[n \times r]} \Sigma_{[r \times r]} (V_{[d \times r]})^T$$

- A: $n \times d$ matrix
- r: the rank of the matrix
- U: $n \times r$ matrix, column orthonormal, $U^T U = I$
- Σ : $r \times r$ diagonal matrix, strength of each factor
- V: $d \times r$ matrix, column orthonormal, $V^T V = I$



svd-matrices.png (800×339) (intoli.com)

SVD $\leftarrow\rightarrow$ Eigen-decomposition

- SVD gives
  - $X = U \Sigma V^T$
- Eigen-decomposition gives
  - $B = X^TX = W \Lambda W^T$
- U, V, W: orthonormal $\rightarrow$ $U^TU = I$, $V^TV = I$, $W^TW = I$
- Σ, Λ: diagonal
- Relationship:
  - $XX^T = U \Sigma V^T (U \Sigma V^T)^T = U \Sigma V^T (V \Sigma^T U^T) = U \Sigma^2 U^T$
  - $X^TX = V \Sigma^T U^T (U \Sigma V^T) = V \Sigma \Sigma^T V^T = V \Sigma^2 V^T$
  - $B = X^TX = W \Lambda W^T = V \Sigma^2 V^T$

PCA via SVD

- $X_0 \leftarrow n \times d$ data matrix, data point $\rightarrow$ row vector x_i
- X: subtract mean x from each row vector x_i in X_0
- $U \Sigma V^T \leftarrow$ SVD of X
- The top r right singular vectors V of X $\rightarrow$ the PCs
- The singular values in Σ = the square roots of the eigenvalues of $X^T X$

Example on a Document x Term

Term Document	data	information	retrieval	brain	lung
CS-TR1	1	1	1	0	0
CS-TR2	2	2	2	0	0
CS-TR3	1	1	1	0	0
CS-TR4	5	5	5	0	0
MED-TR1	0	0	0	2	2
MED-TR1	0	0	0	3	3
MED-TR1	0	0	0	1	1

- d = 5 but r=2 → two bases [1 1 1 0 0] & [0 0 0 1 1]
- U: document-to-concept similarity matrix
- V: term-to-concept similarity matrix
- Σ: its diagonal elements → strength of each concept

Interpretation

Term Document	data	information	retrieval	brain	lung
CS-TR1	1	1	1	0	0
CS-TR2	2	2	2	0	0
CS-TR3	1	1	1	0	0
CS-TR4	5	5	5	0	0
MED-TR1	0	0	0	2	2
MED-TR1	0	0	0	3	3
MED-TR1	0	0	0	1	1

doc-to-concept similarity matrix

CS-concept

MD-concept

strength of CS-concept

CS-concept

term-to-concept similarity matrix

$$
\begin{array}{c c}
 & \begin{array}{ccccc} \text{data} & \text{inf.} & \text{retrieval} & \text{brain} & \text{lung} \end{array} \\
\begin{array}{c} \text{CS} \\ \\ \\ \\ \text{MD} \\ \\ \\ \end{array} &
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 \\
2 & 2 & 2 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 \\
5 & 5 & 5 & 0 & 0 \\
0 & 0 & 0 & 2 & 2 \\
0 & 0 & 0 & 3 & 3 \\
0 & 0 & 0 & 1 & 1
\end{bmatrix}
\end{array}
=
\begin{bmatrix}
0.18 & 0 \\
0.36 & 0 \\
0.18 & 0 \\
0.90 & 0 \\
0 & 0.53 \\
0 & 0.80 \\
0 & 0.27
\end{bmatrix}
\times
\begin{bmatrix}
9.64 & 0 \\
0 & 5.29
\end{bmatrix}
\times
\begin{bmatrix}
0.58 & 0.58 & 0.58 & 0 & 0 \\
0 & 0 & 0 & 0.71 & 0.71
\end{bmatrix}
$$

SVD – Dimensionality Reduction

- To reduce the dimensionality further (3 zero singular values have already been removed)
  - Best rank-1 approximation →

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.18 & \cancel{0} \\ 0.36 & \cancel{0} \\ 0.18 & \cancel{0} \\ 0.90 & \cancel{0} \\ 0 & \cancel{0.53} \\ 0 & \cancel{0.80} \\ 0 & \cancel{0.27} \end{bmatrix} \times \begin{bmatrix} 9.64 & 0 \\ 0 & \cancel{5.29} \end{bmatrix} \times \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0.71} & \cancel{0.71} \end{bmatrix}$$

Week 5 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Three PCA APIs in Spark MLlib

- DataFrame-based API – PCA (source code, Scala doc)
  1 • `pyspark.ml.feature.PCA(k=None, inputCol=None, outputCol=None)`
- RDD-based API – RowMatrix (source code, Scala doc)
  2 • `computePrincipalComponents(k)`
  3 • **Scalable**: `computeSVD(k, computeU=False, rCond=1e-09)`

```scala
  @Since("1.6.0")
  def computePrincipalComponentsAndExplainedVariance(k: Int): (Matrix, Vector) = {
    val n = numCols().toInt
    require(k > 0 && k <= n, s"k = $k out of range (0, n = $n]")

    if (n > 65535) {
      val svd = computeSVD(k)
      val s = svd.s.toArray.map(eigValue => eigValue * eigValue / (n - 1))
      val eigenSum = s.sum
      val explainedVariance = s.map(_ / eigenSum)
```

SVD in Spark MLlib (RDD)

- U: $m \times k$; Σ : $k \times k$; V: $n \times k$
- Assumption: n (dimensionality) < m (# samples)
- Different methods based on computational cost
  - If n is small (n<100) or k is large compared with n (k>n/2), compute A^TA first and then compute its top eigenvalues and eigenvectors locally on the driver
  - Otherwise, compute $(A^TA)v$ in a distributive way and send it to ARPACK to compute (A^TA)'s top eigenvalues/eigenvectors on the driver node

Selection of SVD Computation

```
        if (n < 100 || (k > n / 2 && n <= 15000)) {
          // If n is small or k is large compared with n, we better compute the Gramian matrix first
          // and then compute its eigenvalues locally, instead of making multiple passes.
          if (k < n / 3) {
            SVDMode.LocalARPACK
          } else {
            SVDMode.LocalLAPACK
          }
        } else {
          // If k is small compared with n, we use ARPACK with distributed multiplication.
          SVDMode.DistARPACK
        }
      case "local-svd" => SVDMode.LocalLAPACK
      case "local-eigs" => SVDMode.LocalARPACK
      case "dist-eigs" => SVDMode.DistARPACK
      case _ => throw new IllegalArgumentException(s"Do not support mode $mode.")
```

Week 5 Contents / Objectives

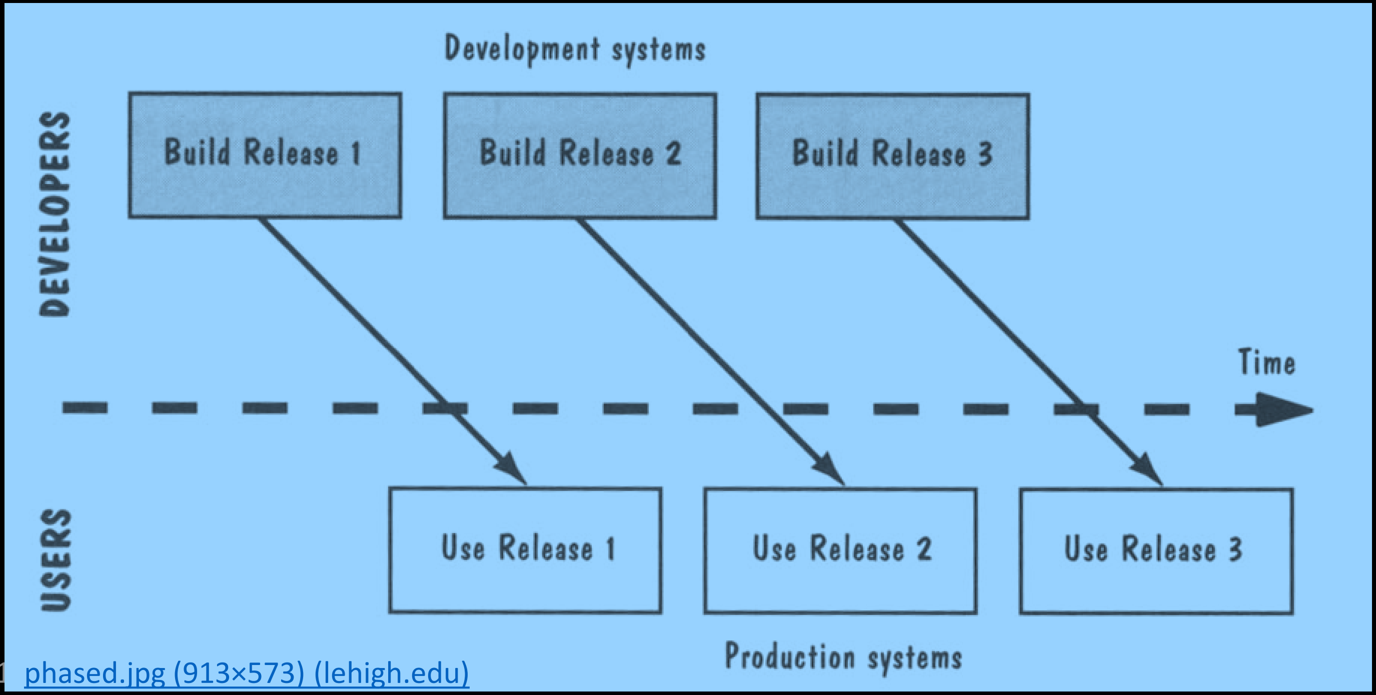
- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Software and Changes

- A virtue of software: relatively easy to change
  - Otherwise, it might as well be hardware
- Planning for change
  - Good *comments* describe meaning of code to facilitate and reduce the cost of software maintenance
  - *Modularity* help manage change because modules help to isolate and localize change

Phased Development

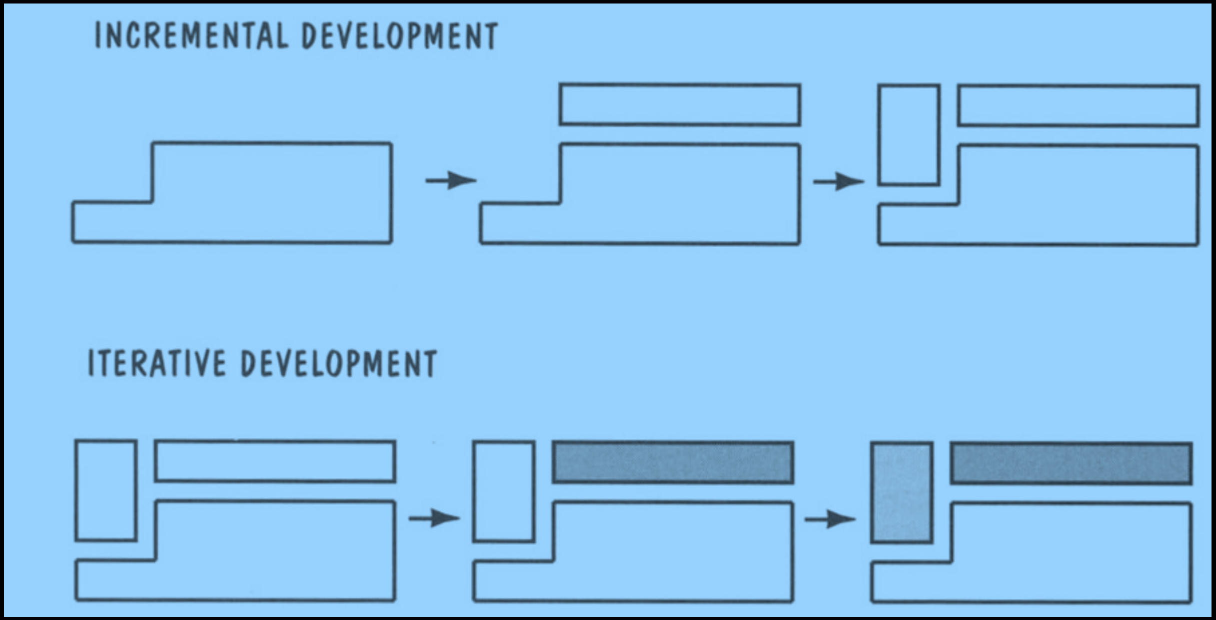
- Reduce cycle time, deliver in pieces, let users have some functionality while developing the rest
- Two or more systems in parallel
  - The operational/production system in use by customers
  - The development system to replace the current release



[phased.jpg (913×573) (lehigh.edu)](phased.jpg)

Iterative/Incremental Development

- **Incremental**: partition a system by functionality
  - Early release: small, functional subsystem
  - Later releases: add functionality
- **Iterative**: improve overall system in each release

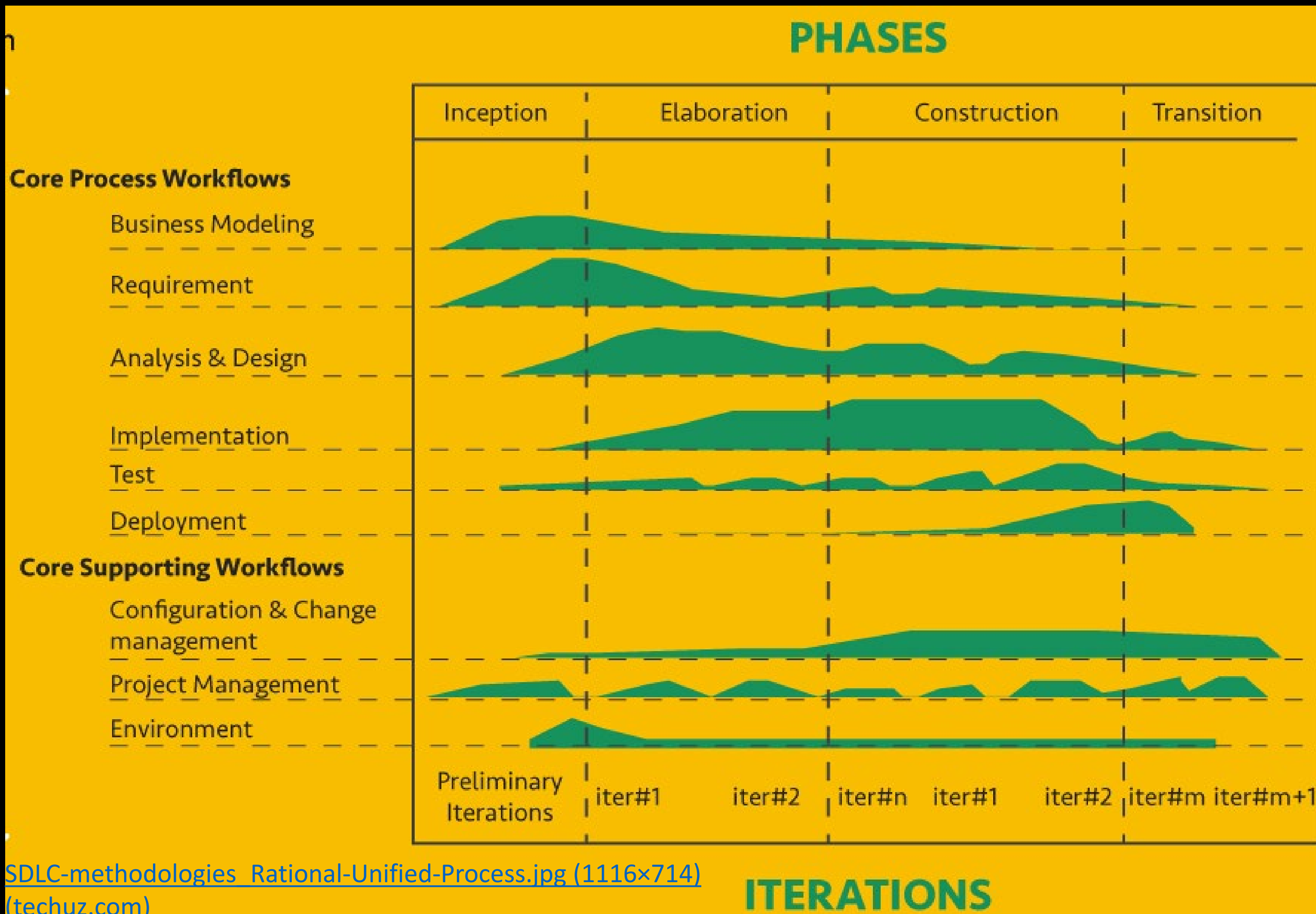


[iterative.jpg (902×539) (lehigh.edu)]

Lifecycle Phases

- Inception: rationale, scope, and vision
- Elaboration: “Design/Details”
  - detailed requirements and high-level analysis/design
- Construction – “Do it”
  - build software in increments, tested and integrated, each satisfying a subset of the requirements
- Transition – “Deploy it”
  - beta testing, performance tuning, and user training
- Phases: NOT the classical requirements/ design/coding/implementation processes
- Phases iterate over many cycles

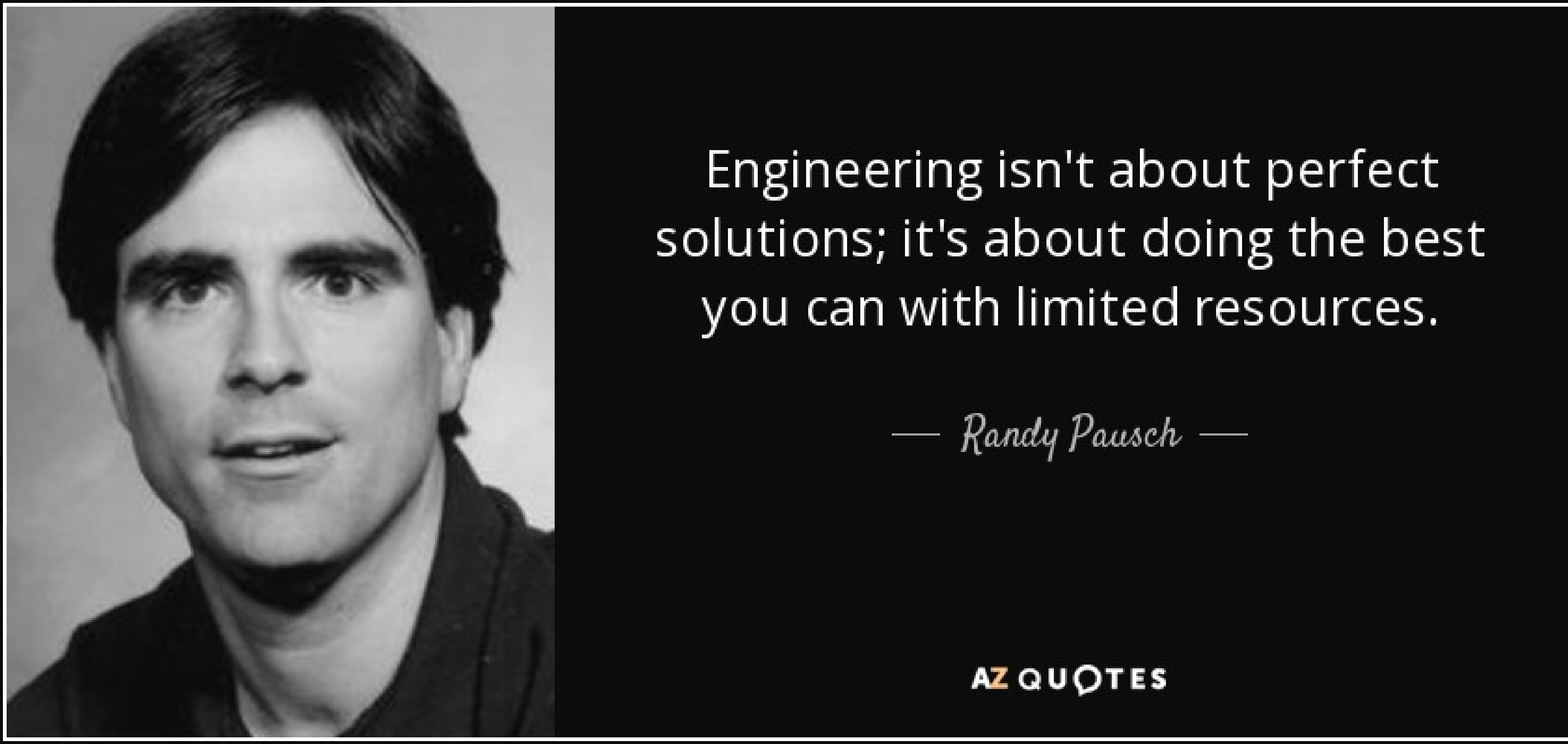
Phases: Iterative & Incremental



[SDLC-methodologies_Rational-Unified-Process.jpg (1116×714) (techuz.com)]

Work on Your Project

- Get a small subset or a reduced version to study, develop, debug, and test
- Break down big/difficult problem into smaller/easier sub-steps (avoid black-box debugging)
- Be structured, organised, and logical
- Keep good documentation
- Get help online (e.g. search) and keep the references



[Randy Pausch quote: Engineering isn't about perfect solutions; it's about doing the best... (azquotes.com)]

Acknowledgement & References

- Acknowledgement
  - Some slides are adapted from the MMDS book slides and the slides on software process life cycles by Glenn Blank
- References
  - Chapter 11 of the MMDS book