ניהול פרויקטי תוכנה

14.03.2021

<u>הרצאה 3</u>:

הגישה הרציפה

בשיעור האחרון דיברנו על היבטים של ניהול: עלויות, זמנים

דיברנו גם על הגישה האג'ילית, על KanBan ו-Scrum

המעבר בין מפל המים לאג'יל היה בגישות של אב-טיפוס

בניגוד לסדרתיות של מפל המים, אם אנחנו מסתכלים על הגישה האיטרטיבית של יצירת גרסאות קטנות שהולכות ומתפתחות עד לגישה הסופית במודל הספירלה, אחד הגורמים המרכזים זה ניהול סיכונים , בוחנים את ההישגים שלנו באותה איטרציה ומנהלים תוך כדי את הסיכונים, במידה וסיכון עולה ניתן לזה מענה, מקבלים משוב מהלקוח, וכל ההתנהלות הזו מובילה אותנו בעצם מגרסה ראשונית לגרסה סופית, בהתאם למשובים אנחנו מקבלים ניתן להגדיר את הדרישות כמו שצריך.

<u>גם בגישה האג'ילית יש משהו ש</u><u>חסר</u> –

איזה מגבלות יש לגישה של הספרינטים?

בכל פעם יכול להיות שהאטרציה הבאה תגרום לשינויים / התנגשויות / חוסר התאמה בהשוואה לגרסאות הקודמות, גם כאן צריך לחשוב על איזשהי דרך שמצד אחד אנחנו מצמצמים את המשלוח, ולכן אנו מנסים לחשוב על כיוונים נוספים שהם מעבר לשיטות המקובלות ברמה הרגילה של אג'יל, ואנחנו רוצים לעשות תהליך של speeding up, לגרום לכל הסיפור הזה לעלות רמה, ולייצר לנו תוכנה בתהליכים רציפים, שבכל רגע נתון מפתחים ומדלברים את זה לסביבת הפרודקשיין גם אם זה פחות מ-3-6 שבועות, גם אם זה לא נכנס בדיוק לסלוט כזה / טיים בוקס של ספרינט.

בדיוק כמו האיילה שיודעת לטפס על סלעים ולרוץ בתנאי שטח שונים, גם אנחנו צריכים להתאים את עצמנו תוך כדי ריצה, להתאים את עצמנו לסביבה ולשינויים ולמה שנדרש.

האתגר פה – להבין איך אנחנו מצד אחד מריצים הכל יותר מהר, ומצד שני אנחנו רוצים גם לשלוח תוכנה, לדלבר מוצר איכותי שהוא נבדק. אז לכאורה יש פה trade off – אם נרצה לבדוק את זה באופן איכותי צריך יותר זמן / יותר אנשים / יותר עלויות או גם וגם וגם.

האתגר פה הוא לעמוד באותה מסגרת זמן, עם אותו כוח אדם ובאותן עלויות.

אנחנו מדברים פה על סביבה של ריבוי משתמשים שאנשים עובדים פה על קבצים שונים / אותם קבצים

אנחנו רוצים לייצר במקביל כמה פיצ'רים ומצד שני שיהיה תיאום / סנכרון. באופן כזה הקוד יגדל באופן מסודר.

מה עוד יכול לקרות כשעובדים במקביל על דברים?

נרצה שהתהליכים יהיו אוטומטיים ככה שכל שינוי שנעשה נדחף למאגר עם כל שאר הרכיבים ורק אז ניתן לומר שסיימנו את הפיתוח, כל הבדיקות יהיו כחלק אינטגרלי של תהליך הפיתוח, כך שניתן לומר של קוד שנמצא בריפוסיטורי נבדק.

אנחנו צריכים מערכת שתיזום את ההרצה של הסקריפטים שיוצרים את השרשרת הזו, כל הפייפ ליין, הצינור של התהליכים שרצים אחד אחרי השני בכל מקום כשאנחנו עושים בילד נניח.

אנחנו רוצים שהסביבה הזו תהיה סביבה שמשקפת לנו את המצב הקיים בזמן אמת באופן שוויוני,

כלומר כל המפתחים יודעים בזמן אמת מה קורה בקוד – גם המנהלים ולרוב גם הלקוח, המערכות האלו מאפשרות לנו לנטר את המערכת על כל ההיבטים, החל מלוחות זמנים, קצב התקדמות,

מקבלים תמונה מלאה על ממוצע זמן הפיתוח לסיפור משתמש מסוים נניח, נוכל לדעת כל כמה זמן נוצר באג, נוכל לדעת כל כמה זמן נוצר קונפליקט, וזה בגלל שיש לנו את הכלים שמאפשרים לנו לעשות את זה.

המעבר מאותן סביבות אג'יליות שמיישמות תהליכים רציפים הולכות לכיוון CI/CD.

בקורס שלנו לא נעשה CI/CD.

Continues - תהליכים רציפים –

CI/ CD – אחד המאמרים שמתעסקים בזה וסורקים את המעבר מהאג'ייל ל-CI/CD זה המאמר "Stairs to Heaven" – התהליך הזה, המעבר הזה הוא מעבר משולב, מעורב. זה תהליך כבד כי הוא דורש שינוי מבנה עבודה, תקשורת, עבודה מול לקוח, כל ההתנהלות של הפרויקט וההגדרות עצמן משתנות, "השאיפה להגיע לגן עדן" אנחנו לא מבזבזים זמן בלחכות לספרינט, אנחנו עושים פייפ ליין ומעוניינים שהכל יעבור בזמן אמת לסביבת לקוח.

בדיקת אינטגרציה – לבדוק איך שני רכיבים עובדים יחד, לא מספיק לבדוק איך כל אחד מהם עובד בנפרד אלא צריך לבדוק ששניהם עובדים טוב גם יחד. בין היתר חיבור בין פרונט אנד, לבק אנד.

! אם אנחנו רוצים לעבור לפיתוח רציף, אנחנו חייבים לבצע בדיקות.

לאחר פיתוח ושליחת המערכת, השלב הבא זה תחזוקה של המערכת.

השלבים הם כאלו: פיתוח -> בדיקות -> שליחה של המערכת לסביבת פרודקשיין -> משתמשים מתחילים להשתמש בתוכנה -> מנטרים בעיות ומתחזקים את המערכת.

נשים לב למעגלים בכמה רמות. המעגל הפנימי, מתחילים בתכנון - > מימוש -> אינטגרציה ובדיקות - > מעבירים לסביבת פרודקשיין -> נתחזק בסביבת פרודקשיין -> זה המחזור חיים של פיצ'ר.

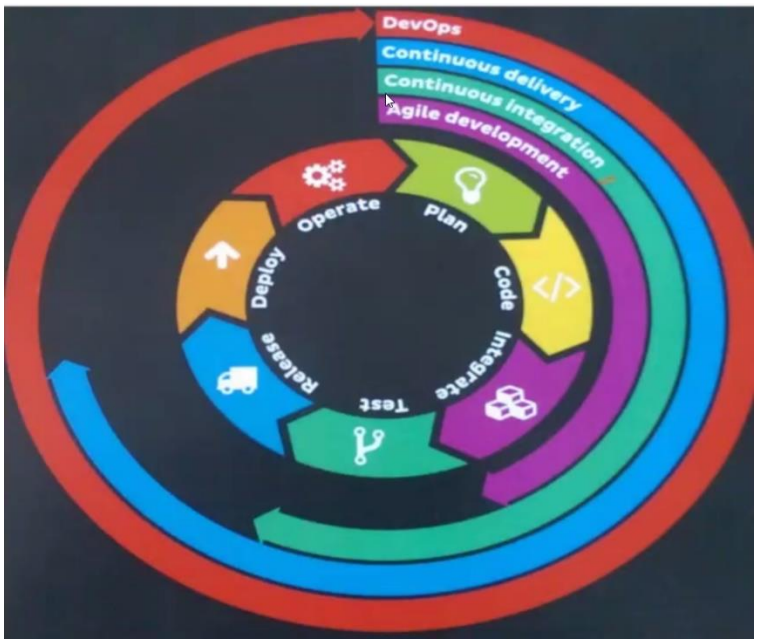
המערכת שמעל המעגל הזה בתרשים למטה, אומרת –

נעבוד בסביבה אג'ילית, עם תהליכים רציפים, שילוח מתמשך, DevOps

DevOps – בשביל שלנו תהיה את התשתית לפתח את הקוד על איזשהי סביבה שהיא אוסף של סביבות בעצם, פיתוח ותשתיות. המעבר ל-CI/CD הוביל לכך שגם המפתחים הם גם אמורים לדעת תשתיות, ניהול סביבות כחלק מתהליך הפיתוח. אחד הכלים שקיימים זה ה-Jenkins.

אוסף של יחידות שעוברות מתחנה אחת לתחנה שניה באופן אוטומטי כשניתן לבצע בקרה על כל שלב ושלב.

נסתכל על החיבור בין פיתוח לתשתיות בתרשים שזה בשאיפה לאינסוף.



השוק מוצף בכלים שניתן להשתמש בהם בפרויקט שהוא עובד בשיטת CI/CD.

יש לנו אוסף שלם של תהליכים, לא ניתן להגיד שאנחנו עושים תהליכים רציפים מבלי שנעשה תכנון רציף או בדיקות רציפות, צריך להתייחס על הכל, שכל תהליכים נעשים תחת בקרה רציפה.

נניח אם אני מגדירה סיפור משתמש, אני ארצה לקבל משוב מבעלי העניין, ואז להמשיך בפיתוח.

גם את הסיפורי משתמש נמשיך לבדוק בפרודקשיין. ונבדוק האם הפונקציונליות טובה. אז אפילו לסיפורי משתמש יש את ההיבט של הרציפות.

נמשיך לבדוק האם המשתמשים נוטשים את המערכת, או חלקים ממנה. מערכת שהמשתמשים לא מאמינים בה, אנחנו חוזרים חזרה לתכנון / מימוש ועושים חושבים מחדש.

זוהי גישה הוליסטית, רציפה לכל אורך תהליך הפיתוח.

אחד הפערים בין המפתחים ללקוח , זה ממשקים חיצוניים. כשאנחנו מדמים סביבת פיתוח לסביבת פרודקשיין אנחנו צריכים לחשוב איזה ממשקים קיימים –

<u>איזה ממשקים של סביבות קיימים</u>?

מערכות הפעלה שונות, כל נושא השרתים – הודעות, אנשי קשר, תמונות נניח ..

כל השרתים שקיימים בסביבה אמיתית אבל לא קיימים בסביבת פיתוח, אז ננסה לדמות את זה בסביבת הפיתוח, כדי שאחר כך נעביר לסביבת פרודקשיין המערכת תתפקד באותו אופן.

כשאנחנו עובדים בcontinues אנחנו צריכים לקחת בחשבון שבאופן מהיר זה ייכנס למשתמשי הקצה השונים.

במעבר ל-CI/CD אנחנו נדרשים לרמת אוטומציה מאוד גבוהה,

כל הדברים שנעשו באופן ידני עוברים למצב אוטומטי באמצעות קוד.

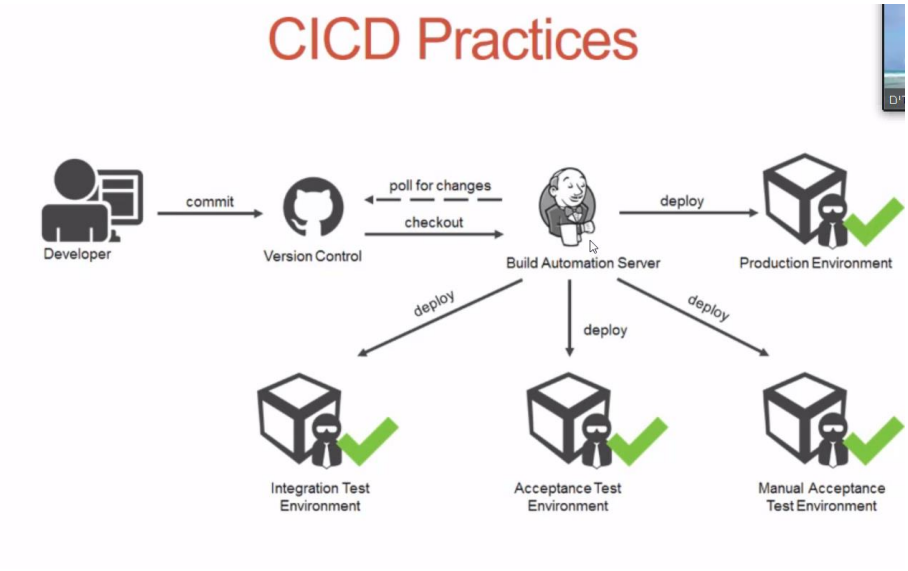
התהליכים הללו חייבים לרוץ כמה פעמים ביום, כמובן בהתאם לשינויים בקוד,

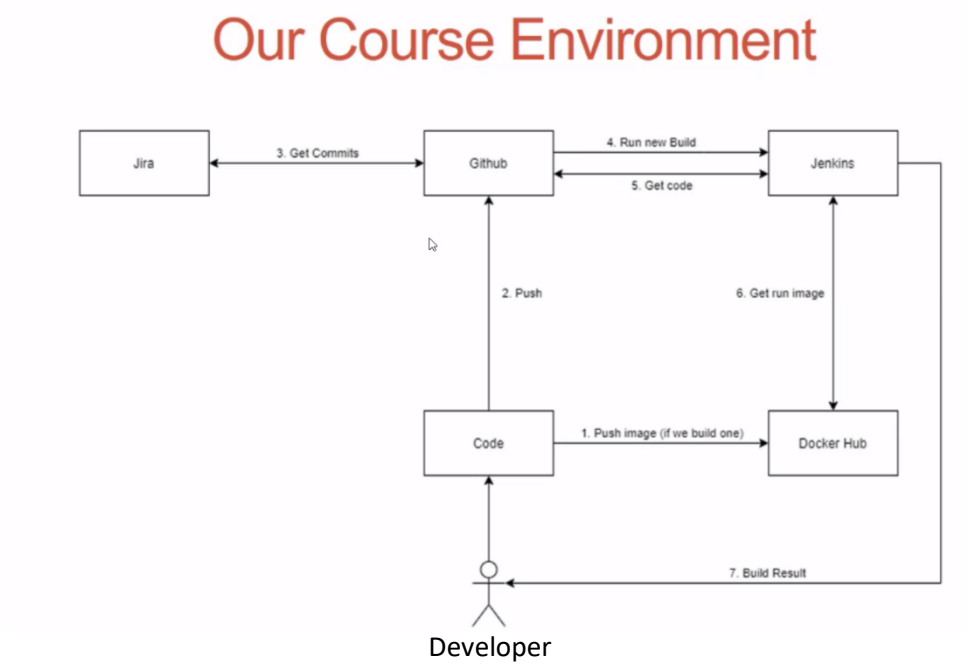
כשאנחנו גוררים שינוי, אנחנו מבצעים בילד, קומיט, פוש, ותוך כדי להריץ את הבדיקות,

ולכן אנחנו רוצים מנגנונים שיבצעו את זה,

<u>מה האתגרים במעבר ל-CI/CD במבט על הגורם האנושי ?</u>

- להכיר כלים חדשים שלא הכיר לפני כן
- האחריות האישית של כל מפתח הופכת למרכזית – אם אני מפתחת אני רוצה שמה שפיתחתי יהיה קוד נקי, כדי שאוכל לבדוק אותו. המפתח "חובש כובע" של הבודק.
- נדרשים לשינוי ארגוני / מבנה ארגוני שמשתנה.
- למידה והשתפרות באופן רציפה
- גישה / שקיפות כלפיי לקוח, כלומר הלקוח רואה מה שכל מפתח עושה, אם עושים deployment לסביבת פרודקשיין אז הלקוח מקבל הכל באופן מהיר, לכן הקוד שמפתח מפתח צריך להיות עם מינימום בעיות ובמינימום זמן למעבר ללקוח.





כלים סטנדרטיים בשיטת CI/CD:

בשלב הקוד – יש את השימוש בגיט, גיטאהב, הג'ירה משותף גם לקוד וגם לתכנון
BUILD – מייבן
טסטים – Junit
הרצה של התהליכים / ריליס – ג'נקינס
Deployment – שף, פאפט
Monitor על סביבת פרודקשיין – ספלאק, סנסו.

היתרונות ב-DevOps –
העברת התוצרים בצורה מהירה וחלקה,
שביעות רצון מכל התהליך של המשלוח היא גבוהה, אפשר לעשות פחות rollBacks, יודעים לתקן מיידי בתוך הסביבה עצמה,
אפשר לנטר ולהגביל את רמת הביצועים,
אפשר לבחון את כל נושא המשאבים החל מתקציב, דרך סיפורי משתמש שיודעים מתי פותח, כמה נבדק ואיך, האם יש בעיות ואיך מטופלות במידה וכן, וכו'..

הרבה פעמים סביבות שבעבר נפלו והיה צריך להרים אותן ולהקים מחדש, היום באופן אוטומטי הן קמות באופן קצר, השרתים נמצאים הרבה פחות למטה,
עדיין שומעים פה ושם על כל מיני תקלות , אבל הזמן תגובה והתיקון הוא כמעט בזמן אמת. השירות שאנחנו מקבלים הוא כמעט בזמן אמת בזכות המוניטורינג.
עדיין יש קשיים לא מעטים בסביבות האלה, כשעובדים בסביבות מתקדמות זה לא שחור לבן, יש חברות מאוד מאוד מתקדמות ועובדות בתהליכים רציפים, יש כאלה יעבדו בהייברידס.

פייסבוק – אחת החברות שעובדת ב-continues הרבה שנים. החברה הזאת יודעת לעשות את הפיתוח ואת ה-deployment תוך שעות עד ימים. אם יש פיצ'רים חדשים, לפני שעושים ניסיונות על לקוחות קצה יתנו קודם כל לעובדי פייסבוק, פידבק ראשוני על השימוש באותם יכולות ורק אחר כך זה עובר למשתמשי הקצה וגם זה רק לפלח מסוים מהמשתמשים.

- רוב התיקונים נעשים בסביבת פרודקשיין.

Dog Fodding – זה הרעיון שהעובדים קודם מתנסים
הגישה השניה זה הפצה לחלקים מאוד ספטציפיים של המשתמשים – TESTING AB.
Crowd Testing – גרסא עובדת שאנחנו עושים את הבדיקות על פילוח מסויים של השוק, הרבה פעמים ממש מעסיקים בודקים שהם משתמשי קצה בכל העולם במדינות שונות כדי שנוכל לבוא ולהגיד שחוכמת ההמון הראתה ש-אותו מימוש / אותה