מגזין
עולם הבדיקות
רבעון רביעי 2024 גיליון מס' 39
www.testingworld.co.il

דבר העורך
ניצן גולדנברג

קוראים יקרים,

מונח לפניכם גליון מס 39 של מגזין "עולם הבדיקות".

בגיליון זה תוכלו למצוא מגוון רחב של מאמרים חדשים, בנוסף לטורים המעולים והקבועים שלנו:

"הנדסת כאוס, שיטה לבדיקת resilience של יישומים בענן" – ורד שיינמן

ירון יושינסקי חוזר אלינו עם עוד מאמר מעניין והפעם, "אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps"

"שיפור מצוינות ב-QA באמצעות זיהוי בעיות מוקדם, ניטור והתראות פרואקטיביות" – דניס קוזירה

יש לנו תשבץ וחידות חדשות ומאתגרות עבורכם בגליון זה

בנוסף, יש לנו את הטורים הקבועים: מחפש צרות, בחן את עצמך, ראיון עם מנהל/ת בדיקות, האנציקלופדיה לבדיקות, עושים QA לקריירה, משולחנו של שביט, סקירת כלים ונגיסה מ-TestIL.

אנו נשמח לקבל בקשות לנושאים חדשים ומעניינים למאמרים, צרו עימנו קשר במייל

magazine@testingworld.co.il

קריאה מהנה,
ניצן גולדנברג

ניצן גולדנברג

מזה 9 שנים בתחום בדיקות תוכנה. מהנדס בדיקות בחברת AppCard, מוביל את ערוץ הפודקאסט TestIL Podcast, מנהל את קבוצת המייעצים (AB) של עמותת ITCB®, יו"ר ומנהל תחרות הבדיקות הישראלית ISTC, חבר בקבוצת ה"מרקטינג" של ISTQB, המוביל של קבוצת המיטאפ הגדולה בישראל לבודקי תוכנה TestIL ומרצה בכיר בקורסים וכנסים לבודקי תוכנה.

עדכון מוסמכים בישראל

סה"כ מוסמכים בבורד הישראלי 10,629 – ITCB®

מתוכם 1104 "מצטיינים" אשר ענו מעל 90% תשובות נכונות.

97 מצטיינים ברבעון השלישי של שנת 2024

הודעות ועדכונים:

הסילבוס לבחינת ההסמכה הבסיסית המשודרגת CTFL 4.0 עכשיו גם בעברית!!!

סט בחינות חדשות בעברית לפי גרסת ההסמכה הבסיסית המשודרגת CTFL 4.0 הועלו לאתר ITCB®

מו"ל
Israeli Testing Certification Board
ITCB®

ניהול המגזין
יאן ברון, *iMDsoft*

ניהול התוכן
קובי הלפרין, *Red Hat*

עורך ראשי
ניצן גולדנברג, AppCard

עיצוב גרפי
בית נלי מדיה
סטניסלב קולנקו
www.beitnelly.com

יצירת קשר
אימייל:
magazine@testingworld.co.il

הרשמה
http://bit.ly/TW-Reg

פקס:03-6176605
כתובת: ברוך הירש 14 בני ברק 51202

www.testingworld.co.il

www.itcb.org.il

תוכן העניינים

עולם הבדיקות נכתב ע"י בודקים עבור בודקים

ITCB® מקדמים את קהילת הבודקים בישראל

הנדסת כאוס, שיטה לבדיקת resilience של יישומים בענן | ורד שיינמן

בתקופתנו הדיגיטלית זמן השבתה של אפליקציה או service בענן יכול להיות יקר מאוד.

זמן השבתה של דקה בודדת של AWS מוערכת בסביבות $154,333, לא כולל קנסות, השפעה על הלקוח, עלויות תפעול ופגיעה במוניטין. מכיוון שעסקים מסתמכים יותר ויותר על יישומים בענן, הבטחה שהיישומים הללו עמידים בפני כשלים שונים היא דבר חשוב ביותר. כאן נכנסת לתמונה הנדסת כאוס - chaos engineering, המציעה שיטה לבדוק ולשפר את העמידות של יישומים בענן[1]

הנדסת כאוס היא דיסציפלינה המתמקדת בניסויים מבוקרים במערכת אינטרנט מורכבת משירותים מורכבים כדי לעמוד בתנאים לא יציבים בסביבת בדיקות או פרודקשן. הרעיון החלוצי שהגתה נטפליקס היה החדרה מכוונת של כשלים למערכת כדי לזהות ממצאים במערכת. כשלים לדוגמא יכולים להיות האטה בזמן תגובה של המערכת, נפילה חלקית של הרשת ואפילו כשל מוחלט בשרת אחד או יותר, המדמים תרחישים בעולם האמיתי כמו הפסקות ענן (outages) ואסונות טבע.

כל ספקיות הענן הגדולות בעולם: AliCloud, GCP, Azure, AWS מיישמות הנדסת כאוס כדי להבטיח את החוסן, האמינות והעמידות של השירותים שלהן.

אז איזה חולשות במערכת ענן הנדסת כאוס עוזרת לזהות ולשפר?

1. בעיות scalability: בעיות בהוספת מופעים נוספים (אופקית) או בהגדלת המשאבים של מופעים קיימים (אנכית).
2. פעפוע תקלות: רוב היישומים תלויים בשירותים חיצוניים, שירותים בענן מקושרים אחד בשני. כשל בשירות אחד יכול לגרום לשרשרת כשלים מדורגת.
3. בעיות גיבוי: בזמן ששירות אחד לא מגיב, שירות אחר שתלוי בו מפעיל מערכות גיבוי ותמיכה ובכך נמנעת השבתה רחבה של כל הענן.
4. בעיות התאוששות (recovery): לאחר ששירות מסוים לא היה פעיל וחזר לפעול, הוא עשוי לעלות מחדש בצורה לא תקינה, או ששירותים שתלויים בו לא ידעו למצוא אותו או יחוו תקלות כתוצאה מתגובה מאוחרת או כפולה .
5. פגיעות אבטחה (security vulnerabilities): הגדרות שגויות שחושפות נתונים או גישה לא מורשית.
6. בעיות קונפיגורציה: הבדלים בין סביבות שונות (פיתוח, בדיקות, פרודקשן) המובילים להתנהגות בלתי צפויה.
7. בעיות פרישה (deployment): עדכונים חלקיים או שגויים.

הבנת הנדסת כאוס

מושגי הליבה של הנדסת כאוס כוללים ביסוס השערה (hypothesis) לגבי התנהגות מערכת, הפעלת ניסויים לבדיקת השערה זו (experiments) וניתוח תוצאות הניסוי כדי לשפר את חוסן המערכת.

היתרונות העיקריים בשימוש בהנדסת כאוס הם:

1. חוסן משופר: על ידי זיהוי וטיפול בכשלים פוטנציאליים הנדסת כאוס עוזרת ליצור מערכות עמידות יותר.
2. ביצועים משופרים: הבנה כיצד מערכת מתנהגת תחת עומס יכולה להוביל לשיפור ביצועים כללים.
3. מודעות תפעולית: ניסויי כאוס קבועים מגבירים את המודעות וההבנה של תלויות המערכת וסיבות לכישלונות ודרך התאוששות המערכת.
4. מערכת מודולרית יותר.
5. תכנון כשלים מודע (אנחנו בוחרים מה לעשות כאשר המערכת נכשלת באופנים מסויימים, ולא מקבלים את מה שבמקרה יצא).
6. הפיכת תקלות לאירוע שגרתי - אם משהו בכל זאת קורה, כל בעלי התפקידים יודעים מה לעשות, ויש סיכוי טוב יותר שחלקים גדולים יותר מהעבודה עברו אוטומציה.

> "הנדסת כאוס היא דיסציפלינה המתמקדת בניסויים מבוקרים במערכת אינטרנט מורכבת משירותים מורכבים כדי לעמוד בתנאים לא יציבים בסביבת בדיקות או פרודקשן"

שלבים מרכזיים של הנדסת כאוס:

1. הגדרת מצב יציב: הבנה ותיעוד כיצד נראות פעולות רגילות עבור המערכת במצב שימוש רגיל.
2. ניסוח השערה: הנחה כיצד המערכת תתנהג בתנאי כשל מסוימים.
3. הפעלת ניסויים: הכנסת כשלים בצורה מבוקרת ותצפית על התנהגות המערכת.
4. ניתוח תוצאות: השוואה בין ההתנהגות הנצפית להשערה וזיהוי אי התאמות.
5. שיפור המערכת: שימוש בתובנות שהושגו כדי לבצע שיפורים נחוצים והגברת את החוסן של המערכת.

ורד שיינמן

מהנדסת בדיקות תוכנה, בעלת ניסיון מעל 19 שנה בחברת SAP. מומחית בהגדרת אסטרטגיות בדיקות למוצרי SaaS ולפלטפורמת הענן העסקי של SAP . בעלת ניסיון עשיר בעבודה עם ספקי הענן המובילים בעולם: gcp, azure, aws. מחויבת לשיפור מתמיד של תהליכי הפיתוח וזמני המחזור, החל משלב ה-Commit ועד הגעת המוצר ל-Production. דת בשיתוף פעולה עם צוותי פיתוח ו-devops כדי להבטיח מערכות Production.

1 https://techwireasia.com/2023/06/what-happens-when-the-worlds-largest-cloud-service-provider-goes-offline/#:~:text=There%20has%20been%20no%20official,and%20financial%20services%20companies%20affected

הנדסת כאוס, שיטה לבדיקת resilience של יישומים בענן | ורד שיינמן

יישום הנדסת כאוס

יישום הנדסת כאוס דורש תכנון ושיתוף פעולה קפדניים בין צוותים. לצורך הצלחת הניסוי יש לתקשר את המטרה והיתרונות לבעלי העניין, להבטיח מערכות ניטור חזקות ולהתחיל בניסויים קטנים ומבוקרים. עם הזמן, ניתן להגדיל את הניסויים הללו כדי לבדוק תרחישי כשל מורכבים יותר.

יישום של הנדסת כאוס מתחילה בביסוס הנראות (visibility) של המערכת: כתיבה ללוגים, שימוש במערכות ניטור והתראות. עושים את זה גם בסביבת בדיקות ובטח שבמערכת פרודקשן.

כדי לוודא כי כלי וטכניקות הנדסת כאוס פועלים כראוי ללא השפעה על משתמשים אמיתיים מתחילים לעשות את הניסויים בסביבת בדיקות. לאחר שבארגון עשו למידה ופיתחו אסטרטגיות להתמודדות עם בעיות, מוסיפים ניסוי כאוס גם בסביבת פרודקשן, כדי לבדוק ולהבטיח כי ממנגנוני ניתור והתאוששות מתנהגים כמצופה בסביבה אמיתית.

יישום יעיל של הנדסת כאוס מצריך בדיקות מובנות ותקופתיות, לרוב באמצעות מה שמכונה "ימי הכאוס", Chaos Days. ימי כאוס הם תקופות מוגדרות שבהן ארגון מכניס באופן שיטתי תקלות לסביבות הייצור או סביבת בדיקות שלו כדי לבדוק את החוסן והעמידות של המערכות שלו.

תכנון ימים של משחקי הכאוס

1. **הגדרת יעדים**: קביעת יעדים ברורים ליום הכאוס, כגון תכנון בדיקות תרחישי כשל ספציפיים או אימות מנגנוני התאוששות. היעדים צריכים להתאים למטרות האמינות והחוסן הכוללות של הארגון.
2. **בחירת ניסויים**: בחירת מגוון ניסויים להזרקת כשל, לדוגמא: כיבוי מופעים, הדמיית זמן השהייה ברשת, מיצוי משאבים כמו מעבד או זיכרון, בדיקת מנגנוני כשל. הבחירה צריכה להיות מקיפה מספיק כדי לכסות היבטים שונים של המערכת.
3. **זיהוי בעלי עניין (stakeholders)**: וידוי שכל הצוותים הרלוונטיים, כולל פיתוח, operations ויחידות עסקיות, מודעים ליום הכאוס ולמטרותיו. תקשורת יעילה היא חיונית כדי להכין את כל בעלי העניין להשפעות אפשריות ולהבטיח מאמצים מתואמים במהלך הניסויים.

ביצוע ימים של משחקי הכאוס

1. ביצוע ניסויים בסביבה מבוקרת ומפוקחת. ניסויים יכול להיערך בייצור עם אמצעי בטיחות, כגון החזרות והתראות אוטומטיות, או בסביבת בדיקות המדמה סביבת ייצור. הבחירה תלויה בסיכון הפוטנציאלי ובבשלות שיטות הנדסת הכאוס בתוך הארגון.
2. שימוש בכלי ניטור כדי לבחון את ההשפעות של הכשלים שהוזרקו על ביצועי המערכת והזמינות. ניטור ורישום בזמן אמת חיוניים כדי ללכוד את ההשפעה המיידית ולספק נתונים לניתוח לאחר הניסוי.
3. תיעד תוצאות של כל ניסוי, תשומת לב להתנהגויות או חולשות בלתי צפויות שנחשפו. תיעוד זה צריך לכלול יומנים מפורטים, מדדי ביצועים וכל חריגה מההתנהגות הצפויה. חשוב לזכור - ההתנהגות האנושית היא חלק בלתי נפקד מהניסוי - בקשו מכל צוות מעורב לתעד את הנקודות שירצה לשפר לקראת הניסויים הבאים.

ניתוח ותיקון יום לאחר הכאוס

1. ערכו ניתוח מפורט של התוצאות כדי להבין מה השתבש ומדוע. ניתוח זה צריך לערב את כל בעלי העניין כדי לאסוף נקודות מבט ותובנות מגוונות. המטרה היא לזהות סיבות שורש ובעיות מערכתיות שיש לטפל בהן.
2. פתחו ויישמו תוכניות לטיפול בכל הממצאים שזוהו במהלך הניסויים. תוכניות אלה צריכות לכלול פעולות ספציפיות, לוחות זמנים ואחריות כדי להבטיח שהשיפורים יבוצעו בזמן.
3. שתפו את הממצאים והלקחים שנלמדו עם הארגון הרחב יותר כדי לשפר את המודעות והמוכנות הכוללת. ניתן להקל על שיתוף ידע באמצעות דוחות פנימיים, מצגות וסדנאות. זה עוזר לבנות תרבות של חוסן ושיפור מתמיד ברחבי הארגון.

על ידי תכנון, ביצוע וניתוח של ימי הכאוס באופן שיטתי, ארגונים יכולים לשפר באופן משמעותי את החוסן של היישומים שלהם בענן. גישה פרואקטיבית זו מבטיחה כי כשלים פוטנציאליים מזוהים וממתינים לפני שהם משפיעים על הלקוחות, ובכך שומרים על זמינות ואמינות גבוהות.

> "יישום של הנדסת כאוס מתחילה בביסוס הנראות (visibility) של המערכת: כתיבה ללוגים, שימוש במערכות ניטור והתראות"

שימוש בכלים להנדסת כאוס בארגונים גדולים

רוב הספקיות ענן הגדולות בעולם פיתחו כלים משלהן להנדסת כאוס על מנת לסייע למשתמשים שלהם לבנות ולתחזק אפליקציות עמידות המסוגלות לעמוד בפני שיבושים בלתי צפויים. לגוגל אין שירות הנדסת כאוס ייעודי, עם זאת, משתמשים שלהם יכולים לנצל את התכונות של Istio ו-Kubernetes לדמות תקלות רשת ולבדוק את חוסן המיקרו-שירותים בתוך Google Kubernetes Engine (GKE) ובנוסף להשתמש בכלי קוד פתוח להנדסת כאוס.

בטבלה הבאה ניתן לראות כמה מהכלים בהם ניתן להשתמש מעל כל פלטפורמת ענן:

ספקית ענן	כלים בהם משתמשים להנדסת כאוס
Amazon Web Services (AWS)	AWS Fault Injection Simulator (FIS) Chaos Monkey
Microsoft Azure	Azure Chaos Studio
Google Cloud Platform (GCP)	Google Kubernetes Engine (GKE) Istio Chaos Monkey LitmusChaos Gremlin
Alibaba cloud	Application High Availability Service (AHAS)

המלצות ומשמעויות כספיות:

הנדסת כאוס מתאימה אך ורק למוצרים מבוססי שירותים, במיוחד כאלו הנעזרים בתשתיות ענן. ביצוע ניסויי הנדסת כאוס מגיעים עם עלות שצריך לשלם מראש - בין אם במהלך ביצוע הניסויים עצמם (כוח עיבוד או נפח אחסון עודפים, רישיונות לכלים הדרושים) בין אם בבניית המומחיות הנדרשת על ידי גיוס עובדים מיומנים, יועצים, או שכירת שירותיה של חברה חיצונית ובין אם בתכנון והתאמה של ארכיטקטורת המערכת.

עם זאת, היתרונות לטווח הארוך, כגון מניעת השבתות יקרות ודמי השבתה יקרים, עולים ההוצאות הקצרות טווח הללו.

לסיכום

הנדסת כאוס היא תרגול חיוני להבטחת החוסן של יישומים בענן. על ידי בדיקה יזומה ושיפור יכולתן של מערכות להתמודד עם כשלים, עסקים יכולים לשנות את תרבות העבודה באופן שיסייע למזער זמן השבתה, לשמור על אמון הלקוחות ולהגן על פעילותם מפני אירועים בלתי צפויים. בין אם משתמשים בכלים כמו Chaos Toolkit או פיתוח פתרונות מותאמים אישית, הנדסת כאוס מספקת גישה מובנית לבניית שירותי ענן אמינים וחזקים יותר.

בחן את עצמך | ירון צוברי

בואו לבחון את עצמכם ולבדוק האם אתם מוכנים לענות על שאלה מתוכנית הלימודים הבסיסית המעודכנת והמשודרגת CTFL 4.0.

אם שאלתם את עצמכם "מה הכוונה למשודרגת?", כנסו לדפי המידע באתר ®ITCB וצפו בפרסומים השונים שלנו וכן בערוצי המדיה החברתית השונים של ®ITCB.

ההשקה הרשמית של תוכנית הלימודים הבסיסית המעודכנת והמשודרגת בשפה העברית הייתה לא מכבר בתחילת נובמבר, קרי, מעתה ואילך כל הבחינות לתוכנית הלימוד הבסיסית CTFL תהיינה לפי הסילבוס של התוכנית המשודרגת CTFL 4.0. אתם מוזמנים לגלוש לאתר ®ITCB ולהוריד את תוכנית הלימוד בעברית וכמו כן סטים של בחינות דוגמא לתרגול.

לא אחת נשאלתי – בעיקר בזמנים שהעמידו קשיים לחברות בהן עבדתי – מדוע צריך להשקיע בבדיקות? הלוא זה בזבוז של הזמן. היו כאלה שירדו לפרטים, משמע, עניין אותם להכיר את כל הפעילויות הנדרשות לביצוע בבדיקות, עלויות שלהן, וכמה הן מביאות תועלת להשקה של המוצר ו/או של הגרסה הקרובה. היו גם כאלו שזמנם דחק והיה חשוב להם לדעת שצוותי הבדיקות והפעילויות שבאחריותם ממוקדים במשימות החשובות ובאלו שתורמות להצלחה של המוצר והקבוצה.

למדתי עם הזמן שכדאי לתקשר זאת לפני שנשאלתי, ולהציג את המשימות, דרכי העבודה וכו', ואת ערכם, ובהמשך להציג עד כמה ההצהרות מגובות במעשים.

ועכשיו לשאלה:

איזו מהאפשרויות הבאות מציגה דוגמא לפעילויות בדיקות התורמות להצלחה?

א. מעורבות של בודקים במהלך פעילויות שונות במחזור חיי פיתוח תוכנה (SDLC) תעזור לזהות פגמים בתוצרי עבודה

ב. בודקים מנסים לא להפריע למפתחים בזמן הקידוד, כדי שהמפתחים יכתבו קוד טוב יותר

ג. בודקים המשתפים פעולה עם משתמשי הקצה, עוזרים לשפר את איכות דוחות הפגמים במהלך שילוב של רכיבים ובדיקות מערכת

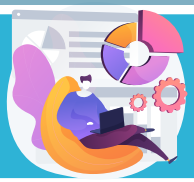
ד. בודקים מוסמכים יעצבו מקרי בדיקה טובים הרבה יותר מאשר בודקים לא מוסמכים

בחר אפשרות אחת

* לצפייה בתשובה המפורטת - דפדפו לעמוד 10

ירון צוברי

מעל 30 שנות ניסיון בפיתוח תוכנה, ניהול פרויקטים מורכבים, הנדסת מערכות ובדיקות. עם ניסיון בינלאומי במערכות מורכבות בתחומים: טלקום, פיננסים, אוטומוטיב ומערכות הגנה. מומחיות עיקרית: ניהול פרויקטים מורכבים, ייעוץ להנהלה בכירה בארגונים (Siemens Germany), אימון מנהלים. נשיא וסגן נשיא ארגון ®ISTQB לשעבר ונשיא ®ITCB מיום הקמתה.



משולחנו של שביט
"יש לי יום יום באג", האם אלו המילים הנכונות?

שביט ג'רסי

אבא לתאומים בני 4, מנהל בדיקות בחברת Wisetamp, בעל תואר ראשון בהנדסת תעשייה וניהול מהטכניון.

בעל 12 שנות נסיון בתחום הבדיקות, מתוכם מעל 9 שנים מדריך עצמאי ל-QA.

בעל סדרת הסרטונים השבועית "QA ללא הפסקה"

חברים, שיהיה לכם רבעון מוצלח ושבוע טוב לכולם.
הפעם אני רוצה לדבר איתכם על יחסי אהבה / שנאה עם הבאגים שאנו מצליחים לתפוס על גבי המערכת שלנו.

באיזשהו מקום...כשיש בעיות במערכת שלנו – זה מדאיג אותי ואמור להדאיג כל בודק אחר. על אחת כמה וכמה כשהבעיות האלו מתגלות ב-PRODUCTION.
משום שזה אומר שלא עלינו על הבעיות הללו ב-Sandbox.

לצד זאת, כשאתם מבינים שהאוטומציה שלכם חושפת בעיה מאוד עמוקה במערכת - אתם מרשים לעצמכם להיות שבעי רצון באיזשהו מקום, אבל די ברור לנו שמדובר ב-"שמחה מאופקת" משום שזה הדבר האחרון שאתם רוצים להראות למפתחים. שאתם שמחים על בעיה רצינית מאוד שהתגלתה במערכת.

זו הדינמיקה וזה הסיפור בקיצור של אנשי בדיקות אל מול אנשי פיתוח.

כאשר באג מתגלה בסביבת הטסט בזמן של בדיקות על גרסה חדשה, אם באופן ידני או עם בזמן הקידוד של הטסטים האוטומטיים על הגרסה החדשה.. כולם מקבלים את זה באהבה.
כולם מקבלים את זה בהבנה ואנחנו לא צריכים כלל להתבייש בשום דבר.. להפך.

אנחנו שמחים על כך כשהאוטומציה שלנו או הבדיקות הידניות שלנו מניבות פרי ומביאות ערך לחברה. חלק גדול ועיקרי מהעבודה שלנו הוא לעלות על התקלות האלו.

אבל מה קורה כשבאג מתגלה בפרודקשן? -
קודם כל הדבר החשוב ביותר הוא להבין – "ע"י מי?" , מי גילה את הבאג? מי עלה עליו?
האם זה ה-Support Team?, האם מנהל המוצר?, האם המפתחים או יותר גרוע מכך... מנהל פיתוח, מנהל בכיר אחר או גרוע מכך.. המנכ"ל??

האם קרה לכם שהמנכ"ל חווה באגים בזמן הצגת DEMO שלו ללקוחות? – רק תתארו לכם. גרוע ומביך יותר מזה כמעט ולא יכול להיות.

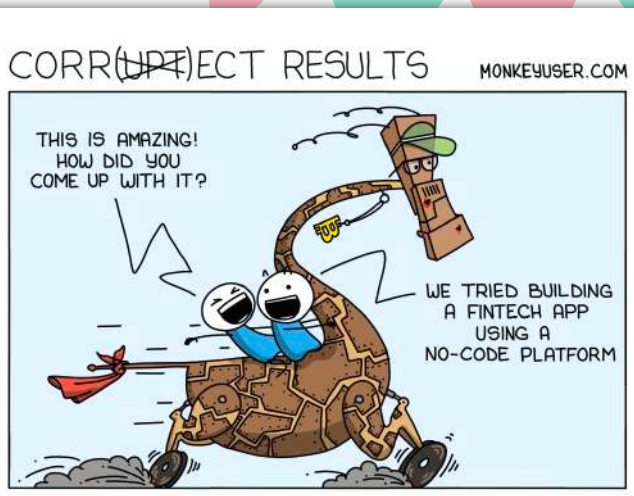
ומה זה אומר אם צוות ה – QA תפסו את הבאג הזה? – האם זה כבר יותר טוב? יותר נסלח?
אולי, ברמה הצוותית זה מעט נסלח יותר. פחות צורם. אבל האם אנחנו יכולים להרשות לעצמנו לשמוח על כך? , האם כל יום שבו אנחנו תופסים באג בכל סיטואציה אפשרית הוא "יום חג"?

אם כך, התשובה היא חד משמעית **לא**. התשובה היא שאנחנו נצטרך לתחקר איך החמצנו את הבאג הזה מלכתחילה? איך הוא עבר את ה – ש.ג?, איך הבאג הזה עבר אותנו שבדקנו את הגרסה הזאת ב-sandbox? ומאיזו גרסה בדיוק הוא הגיע?, האם הכיסוי לא היה מספיק טוב או שמא משהו אחר מנע מאיתנו / הפריע לנו לתפוס את הבאג הזה?

אם כך, לאו דווקא כל באג שנמצא ובכל סיטואציה תמיד הוא יום חג בשבילנו.
וגם אם מתפלק לנו איזה חיוך קטן לא יקרה דבר.
במקרים כאלו חשוב לנו לא לשכוח - לתחקר ולשאול את השאלות הקשות כדי להגיע למיצוי מלא של הסיטואציה הזו.

זכרו – השמחה בחייכם מסתמכת על איכות המחשבות שלכם.

אני מאחל לכם המשך רבעון נעים שקט ורגוע. מעט יותר אופטימיות גם לא תזיק לנו
וכמו תמיד..
שלעולם לא תצעדו לבד



עובד עצות

בסיפור "עצות מן המולדת לנוסע האמריקני" מתאר ויליאם סארויאן דוד זקן שנותן לאחיינו צרור עצות לפני שהלה יוצא לנסיעה ארוכה. האחיין מתעלם מכל העצות, לעיתים פועל ממש הפוך מהם, ונהנה מאוד מהמסע.

ובכל זאת, החודש פרשתי מאינטל אחרי קריירה של 35 שנים, ואיזו דרך מתאימה יותר לתמצת את חוכמת החיים שרכשתי, מאשר ברשימה של עצות? אם לא תשתמשו בהם, לפחות זה יתן לכם רעיונות מה לעשות ההפך. הרי לפניכם, אם כך, רשימה של תובנות שאספתי עם השנים. על חלקן כבר כתבתי; סימון של [*] מציין שיש מראה מקום בתחתית הטור למאמר רלוונטי בנושא. הרשימה לא ממוינת לפי שום סדר חוץ מקטגוריות כלליות. וכמובן שהחשיבות של כל רעיון תלויה בהקשר. אני מניח שזה לא כל מה שלמדתי עם השנים – אבל יש פה מספיק לכרגע...

1. כלים

א. אם צריך כלי לניהול אוטומציה; דרישות; באגים; וכו' – פעילות שכל קבוצת פיתוח עושה – כמעט בטוח שלא כדאי לפתח את הכלי בעצמכם. 99% שאתם לא עד כדי כך מיוחדים שאתם חייבים כלי ייחודי. בנוסף, מובטח לכם שבטווח הארוך זה יעלה הרבה יותר מאשר לאמץ כלי קיים שממומש את רוב הצרכים שלכם. אחרי שהחלטתם על כלי: **אל תנסו לאלץ את הכלי שבחרתם לעשות דברים שהוא לא תוכנן לעשות**. תשנו קצת את התהליכים שלכם כדי שיתאימו לכלי [*].

ב. כן הגיוני לפתח כלים "קטנים" , במיוחד אם הם קשורים באופן הדוק לטכנולוגיה והמוצר הספציפי שלכם. אבל **תשאירו אותם קטנים**, ותדאגו לרמה מינימלית של תיעוד ושמירה על הקוד [*].

ג. כלים ועזרי בדיקות שהמפתחים משתמשים בהם, יתוחזקו וישופרו לאורך זמן בלי שתצטרכו להתאמץ. כדאי לכן לבדוק במה המפתחים משתמשים בעבודת הדיבג שלהם, ולראות איך אפשר לנצל את הכלים האלה לצורך בדיקות. לפעמים זה פשוט אימוץ הכלים כמו שהם. במקרים אחרים די בתוספת קלה על מנת להפוך את הכלים האלה לעזרי בדיקה מעולים.

2. הצד האפל של אוטומציה

א. ברגע שאוטומציה הופכת למטרה העיקרית של קבוצת בדיקות, הפוקוס נע ממקצוענות בבדיקות למקצוענות בפיתוח קוד. אם לא מקפידים על שמירה, פיתוח ועידוד יכולת הבדיקות, תקבלו אולי מערכת אוטומציה ממש יוצאת מן הכלל, אבל מריצה טסטים פשוטים או לא יעילים, ומפספסת בעיות אמיתיות שיצוצו לאחר השחרור, בשטח.

> "כשהאוטומציה היא המטרה העיקרית, הפוקוס נע ממקצוענות בבדיקות למקצוענות בפיתוח קוד"

ב. יצירת הפרדה קבוצתית בין מי שכותבים אוטומציה למי שבודקים מייצרת היררכיה לא בריאה בקבוצת בדיקות: כל מי שבקבוצה של הבודקים שואף לעבור לקבוצת האוטומציה, שהיא יותר מעניינת (לא נשחקים ברגרסיות) ויותר דומה לפיתוח קוד (שבהגדרה, ברוב הארגונים, נחשב לקסטה גבוהה יותר מבדיקות). הדבר גורם לבעיות מורליות, עזיבה, וירידה באיכות הבדיקות. הפתרון צריך להיות סוג של ערבוב. למשל: חלוקה הוגנת של עבודת היום-יום (רגרסיות), בניית plug-ins לאוטומציה על ידי כל החברים בקבוצה, מערכת KDT (Keyword driven testing) מלאה המאפשרת כתיבת תסריטים לא טריוויאליים.

3. ניהול (סתם ניהול; ניהול איכות ובדיקות)

א. לא מנהלים בצעקות. זה לא מוסיף מוטיבציה לכפיפים ולא כבוד למנהל. אף אחד לא רוצה לעבוד באווירה של פחד. עם זאת, **מנהל צריך לדעת לדפוק על השולחן**. כשזה נעשה לעיתים רחוקות מאוד, דפיקה על השולחן היא כלי נהדר להעביר מסר של "הנה משהו שאני לא מוכן\ה לקבל יותר". מדהים כמה מהר המסר עובר כשמנהלים שבדרך כלל רגועים, דופקים על השולחן.

ב. מסמכי בדיקות נכתבים בעיקר עבור הבודקים. ממש **לא חשוב אם אף אחד לא יקרא את תוכנית הבדיקות שכתבתם**. התהליך המחשבתי שהשקעתם בכתיבת המסמך, וההחלטות שנאלצתם לקבל לגבי אסטרטגיית הבדיקות כתוצאה מהצורך לתאר איך ומה תבדקו, ישפיעו (מאוד) לטובה על פרויקט הבדיקות שלכם.

> "מנהל צריך לדעת לדפוק על השולחן"

ג. **בודקים הם לא חוטבי עצים ושואבי מים**. מנהלי הבדיקות צריכים להתנגד כשהמפתחים מזלזלים בזמן של הבודקים ומבקשים מהם לעשות עבודות מכניות שבעצם המפתחים יכולים לעשות בעצמם. אבל לפעמים זה בכל זאת הדבר הנכון [*].

מיכאל שטאל

ארכיטקט בדיקות תוכנה באינטל, ישראל
עוסק בעיקר בבדיקות מערכות משובצות מחשב.
במסגרת תפקידו, מיכאל מגדיר שיטות בדיקה ומתודולוגיות עבודה, עוסק בהדרכה ולפעמים אפילו מרשים לו לבדוק משהו (שזה הכי כייף).
מיכאל מציג תכופות בכנסים בארץ ובחו"ל ומלמד בדיקות תוכנה בפקולטה למדעי המחשב באוניברסיטה העברית.
ניתן לראות חלק מהמצגות והמאמרים שלו באתר
www.testprincipia.com

ד. בקבוצת בדיקות טובה, סביר שאחוז מסוים של באגים ידחו על ידי הפיתוח (מסיבות של: באג כפול; לא משתחזר; התנהגות נכונה שאינה באג).

אם אחוז הבאגים שנדחים הוא מאוד נמוך, יש להניח שהבודקים חטפו על הראש על כל באג שנדחה, והם (א) משקיעים יותר מדי זמן לוודא שמה שנראה כמו באג הוא אכן באג, במקום להריץ עוד בדיקות, או (ב) לא מדווחים על באגים אם יש להם חשש קל שאולי זה לא באג. כלומר: יש באגים שמתגלים אבל לא מדווחים.

מצד שני: אם אחוז הבאגים שנדחים הוא גבוה זה מצביע על חוסר ידע במוצר ובבדיקות.

> "בקבוצת בדיקות טובה, סביר שאחוז מסוים של באגים ידחו על ידי הפיתוח"

כמה זה "אחוז סביר"? חוק האצבע שלי זה 15%, מבוסס על התוצאה לאורך מספר פרויקטים בקבוצת בדיקות שנחשבה טובה. 2% זה ממש מעט. 25% זה ממש הרבה. אבל תחליטו בעצמכם – על פי ההיסטוריה של הארגון או המוצר שלכם.

ה. הטילו עליכם לקדם את האיכות בארגון? להגדיר תהליכים ולוודא את יישומם, לדאוג שהארכיטקטים יכתבו דרישות והמפתחים יכתבו מסמכי תכן (design)? מזל טוב. קיבלתם משימה של "לדחוף חוט". זה לא ילך בקלות, אם בכלל. (ניסיתם פעם לדחוף חוט? הצלחתם?...).

בגדול, מהנדסים (במיוחד מהנדסים ישראליים) לא אוהבים כתיבת מסמכים וביצוע תהליכים ככתבם וכלשונם. אפילו אני לא. אז: אם אין מחויבות מלאה של הנהלת הארגון להטמעת תהליכים – כולל דפיקה על השולחן – אפשר פחות או יותר להתייאש מזה כבר עכשיו. רק כשהמהנדסים יבינו שאין מרחב תמרון סביב הדרישות האלה ושאין טעם להתווכח, רק אז זה יקרה. ולא מספיקה דפיקה אחת על השולחן. ההנהלה הראשית צריכה להופיע לסקירות חשובות, לעודד ולהחמיא לביצועים ולדרוש הסברים כשאנשים לא עובדים לפי מה שהוגדר.

יצא לי לדחוף חוטים בכמה ארגונים. כללית אפשר להגיד שזו לא הייתה הצלחה מסחררת, ובמקרים שזה כן הצליח זה היה כשהמנהלים הודיעו לארגון באופן חד משמעי שנגמרו התירוצים ושמעכשיו עובדים לפי הנהלים. בפרויקט האחרון שלי באינטל עבדנו על מוצר רפואי שבלי תהליכי איכות מסודרים ומתועדים אי אפשר למכור. בפרויקט הזה הייתה תמיכת הנהלה מלאה, מוחשית, עם נראות גבוהה, וממש היה קל לדחוף את החוט לאורך כל הדרך.

ו. התחלתם לעבוד בפרויקט חדש. הקוד עדיין טרי ומשתנה בקצב גבוה. האינסטינקט אומר: "יאללה, בואו נתחיל להריץ בדיקות ונפתח באגים". אבל האם זה באמת הדבר הכי יעיל לעשות?

אל תנחשו.

תלכו לארגון הפיתוח ותשאלו: "איזה פעילויות בדיקה הכי יעזרו עכשיו"? יש להניח שנקודות מסוימות במוצר עדיין בשלבים של בדיקת היתכנות וניסיונות מה עובד יותר טוב. עוד עיניים וידיים שמנסות את האופציות השונות יאפשרו לפיתוח להחליט מה האלגוריתם \ הפרוטוקול \ הטכנולוגיה \ ה-whatever שמתאימים במיוחד לפרויקט שלכם.

לעומת זאת, מציאת באגים באזורים אחרים במוצר מייצרת רעש וממילא יתעלמו מהם - כי הפיתוח מרוכזים בלנסות להבין מה האלגוריתם \ הפרוטוקול \... וכו'. לא רק זה: אחרי שתהיה החלטה על הכיוון הטכנולוגי, יש סיכוי טוב שעוד כל מיני דברים ישתנו. למשל ה-UI, ה-API ושאר ירקות, וכל הבאגים שפתחתם יסגרו כלא רלוונטיים.

עוד ערך מוסף מפעילות תומכת פיתוח שכזו הוא שקבוצת הבדיקות תלמד הרבה פרטים על הטכנולוגיה שבבסיס המוצר, וההתרכזות במה שמעניין את הפיתוח תקנה לקבוצת הבדיקות מוניטין של קבוצה שעוזרת להתקדמות המוצר ולא עסוקה רק בביקורת.

4. כישורי הנדסה

א. מסמכי בדיקות, דוחות פגמים, הערות סקירה וסתם אי-מיילים – כל מהנדס מבלה חלק ניכר מזמנו בכתיבה. **חשוב ללמוד לכתוב טקסט שקל לקרוא ולהבין**. זה אומר להתחיל מההתחלה ולא מהאמצע. להסביר קצת יותר ממה שנראה לכם. לתת פרטים כשצריך, ולהשמיט אותם או להפריד אותם כשעודף פרטים יטביע את הקורא. שימוש נכון בכותרות, בשורות ריקות, בטבלאות, תרשימי זרימה וחלוקה לקטעים מקל על הקוראים שלכם.

צריך להתאמן על מנת להיות כותב טוב, ויש ערך מוסף למהנדסים שכותבים טוב [*].

ב. ובאותו עניין: אם קשה לכם לתקתק מהר (פשוטו כמשמעו: קצב הקלדה גבוה - לא במשמעות של יכולת לחשוב ולהתנסח מהר), בהגדרה אתם תכתבו מסמכים גרועים, כי יהיה לכם חבל על הזמן הארוך שלוקח לכתוב משהו בצורה בהירה ומפורטת. יהיה קשה להבין את דוחות הבאגים שלכם, הקוראים שלכם לא יבינו נכון את מה שהתכוונתם להגיד והתיקון שלהם יתעכב.

בקיצור: תשקיעו בלתקתק מהר.

ובלי שגיאות כתיב – שזה גם סתם פדיחה, במיוחד היום שיש spell checkers - וגם עלול להגמר בתוצאות מפתיעות שלא להם התכוונתם. אז **לפני שלוחצים על Send או Submit – תקראו את הטקסט עוד פעם אחת** לראות שלא פספסתם משהו. נכון גם לאימיילים, הודעות וואטסאפ וכו'.

ג. דוחות פגמים (bug reports) הם תעודת הזהות שלכם מול הארגון. זה תוצר העבודה המוחשי ביותר. אף אחד לא ממש מרגיש שהצלחתם להריץ מאה מקרי בדיקה ביום אחד, אם כולם עברו. אבל כל דוח פגמים שלכם נקרא על ידי מספר אנשים: מנהלי הפיתוח, מהנדסים, ולפעמים גם מנהלי הפרויקט. כמינימום הם קוראים את הכותרת של הבאג. מה זה אומר? שכדאי להשקיע בנושא. זה אומר לוודא שהכותרות והפרטים שכתבתם קריאים וברורים (ראה סעיף א' לעייל), ההצעה לחומרה (severity) אכן נכונה ומראה הבנה במוצר. לעשות חיפוש קצר על מנת להימנע מפתיחת דוח נוסף על באג שכבר דווח; לפרט את צעדי השחזור ככה שאפילו אתם תבינו מה התכוונתם כשתקראו אותם בעוד שבועיים; ושהתוצאה הצפויה לעומת התוצאה הנצפית יסבירו היטב למה אתם חושבים שזה באג.

> "דוחות פגמים הם תעודת הזהות שלכם מול הארגון"

ככה בונים מוניטין.

באג שנפתח על ידי בודקים עם מוניטין חיובי מועבר מיד לטיפול הפיתוח. באג שנפתח על ידי בודקים עם מוניטין שלילי יעבור קריאה מדוקדקת, קפדנית ואולי אפילו קנטרנית ויוחזר למדווחים עם בקשות לעוד פרטים והרצות נוספות.

מחפש צרות | מיכאל שטאל

ד. אתגר הבדיקות, ברמה של 10,000 רגל, די דומה בארגונים רבים. לכן גם שיטות וטכניקות בדיקה מוכרות הן רלוונטיות להרבה קבוצות פיתוח. זה אומר שחשוב **ללמוד** בדיקות - ולא רק לסמוך על האינטואיציה, הגיון בריא ו-On the Job training. הרצאות, סרטונים ובלוגים זה יפה, אבל למידה מסודרת ועמוקה של התיאוריה של בדיקות משיגים רק מספרים, כי המדיה הזאת מאפשרת לפתח נושא לרוחב ולעומק. כשזה לא נעשה, יתכן מאוד שתגיעו לפתרונות הנכונים בעצמכם, אבל לאט יותר ואחרי יותר התחלות כושלות. ואין דבר יותר מבאס מאשר להיות גאה ברעיון נהדר שעלה בדעתכם איך לשפר את הבדיקות, רעיון שנראה לכם כפריצת דרך מבחינה מקצועית, ואז לגלות שזו טכניקה שכבר מופיעה בספרות 30 שנה. **אם אתם רציניים לגבי התמקצעות בבדיקות – תקראו ספרים!** [*].

5. חוקי מרפי

ולסיום, התוספת האישית שלי לסדרת החוקים האלמותית של מר מרפי:

באגים שלא ניתנים לשחזור ישתחזרו בקלות אצל הלקוחות

צחוק צחוק, אבל מה שנובע מחוק זה הוא שכדאי לדווח על באגים שלא משתחזרים בקלות, על מנת שכל פעם שהבאג קורה (וזה יכול להיות בהפרש של חודשים), תהיה רשומה במערכת ניהול הבאגים שאפשר לצרף אליה מידע חדש. אם נראה שהבאג חוזר על עצמו מספר פעמים, זה מספק הצדקה להשקיע משאבים ולצוד אותו, למרות שקשה לשחזר אותו.

חומר קריאה נוסף

כלים [א]:

The Home-grown Tools Syndrome:
https://testprincipia.com/presentations-and-papers-on-software-testing/#heading5

Build VS Buy: https://www.informit.com/articles/article.aspx?p=21775 (by Allen Eskelin)

כלים [ב]:

כללי כלים:
https://testprincipia.com/looking-for-trouble/#heading7

ניהול [ג]:

תיקון ספונטני:
https://testprincipia.com/looking-for-trouble/#heading19

כישורי הנדסה [א]:

וזאת לתעודה:
https://testprincipia.com/looking-for-trouble/#heading4

כישורי הנדסה [ד]:

רשימת קריאה מומלצת:
https://testprincipia.com/other-testing-stuff/#stuff1

בחן את עצמך | ירון צוברי

הפתרון לשאלה

נתחיל עם יעד הלימוד (Learning Objective) ורמת הידע (Knowledge Level) הדרושים לנושא הזה. לידיעה כללית: את יעדי הלימוד ניתן למצוא בעמוד הפתיחה של כל פרק בתוכנית הלימוד (הסילבוס). לכל יעד לימוד מוגדרת רמת הידע הדרושה לו

(K-level – Knowledge Level). המודל המייצג לרמות הידע שארגון ISTQB משתמש בו הוא מודל הטקסונומיה של בלום (Bloom's taxonomy).

יעד הלימוד הרלוונטי לשאלה שלנו כאן הוא: FL-1.2.1 - "הסבר באמצעות דוגמאות מדוע הבדיקות נחוצות"; והשאלה שלנו היא ברמה של K2. רמה זו אומרת שעל הנבחן להצביע על התשובה הנכונה ממקום ויכולת של הבנה, הבחנה, השוואה וכו' של תוכן הקשור לנושא המסויים (או הנלמד במידה וניגש לבחינה לאחר הכשרה).

כאמור, בדיקות מספקות דרך יעילה כלכלית לגילוי פגמים. ניתן להסיר פגמים אלה (על ידי ניפוי פגמים (debugging) - פעילות שאינה בדיקה), ולכן בדיקות מסייעות בעקיפין להעלאת איכות אובייקט הבדיקה.

בדיקות מספקות דרך להערכת האיכות של אובייקט הבדיקה בשלבים שונים של מחזור חיי הפיתוח של התוכנה (SDLC). אמצעים אלה משמשים כחלק מפעילות ניהול רחבה יותר של הפרויקט, תורמים לדוגמא בהחלטות הנוגעות למעבר לשלב הבא בתהליך הפיתוח, כגון בהחלטה על שחרור המוצר.

בדיקות מספקות למשתמשים ייצוג עקיף של פרויקט הפיתוח. באמצעותן הבודקים מבטיחים שהבנתם את צרכי המשתמשים נלקחת בחשבון לאורך כל מחזור הפיתוח. האלטרנטיבה שעומדת מנגד היא לערב קבוצה מייצגת של משתמשים כחלק מפרויקט הפיתוח, אך הדבר לרוב אינו אפשרי עקב עלויות גבוהות וחוסר זמינות של משתמשים מתאימים.

בנוסף, בדיקות עשויות להידרש בכדי לעמוד בדרישות חוזיות (contractual) או חוקיות (legal), או לעמוד בתקנים רגולטוריים.

(למידע נוסף, מומלץ לקרוא את פרק 1 תת-פרק 1.2.1 שבתוכנית ההכשרה הבסיסית - CTFL - של הארגון הבינ"ל ISTQB).

הבה נבחן את השאלה והתשובות:

השאלה שלנו עוסקת בעיקר ביכולת להסביר על התרומה של הבדיקות להצלחה של המוצר, הקבוצה ו/או החברה, ושהדגש העיקרי על הפעילויות של הבדיקות שתורמות להצלחה.

נסתכל עתה על התשובות השונות וננתח האם הן נכונות או לא, ומדוע לא:

א. התשובה **נכונה**. חשוב שהבודקים יהיו מעורבים מתחילת מחזור החיים של פיתוח התוכנה (SDLC). מצב זה יגדיל את ההבנה של ההחלטות הקשורות לעיצוב ויסייע לזהות פגמים מוקדם יותר בתהליך הפיתוח.

ב. התשובה **אינה נכונה**. גם למפתחים וגם לבודקים תהיה הבנה רבה יותר של תוצרי העבודה האחד של השני וכיצד לבדוק את הקוד.

ג. התשובה **אינה נכונה**. משתמשי הקצה לא יעזרו לבודקים בהגברת האיכות של הדיווחים על הליקויים; כמו כן, משתמשים בדרך כלל אינם לוקחים חלק בפעילויות שמבוצעות ברמות הבדיקה (**test levels**) ולא כל שכן ברמת בדיקה נמוכה כמו בדיקות אינטגרציה.

ד. התשובה **אינה נכונה**. להיות מוסמך לא אומר אוטומטית שהבודק יהיה טוב יותר בתכנון הבדיקות.

אם תרצו שאציג שאלת דוגמה בנושא מסוים או אם יש לכם שאלות אנא פנו אלי באימייל: yaron.tsubery@itcb.org.il

אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps - לירן יושינסקי

במאמר הבא אשתף אתכם מדוע בודקים ובדיקות הם חלק חיוני בצוות DevOps ואיך לשלב בדיקות במהלך מחזור חיי פיתוח תוכנה (SDLC).

נבחן מספר נושאים מרכזיים שעלינו כבודקים בצוותים אלה להכיר.

- תפקיד הבדיקות;
- סוגי כלי בדיקה;
- תנאי קדם ל-Continuous Testing;
- היתרונות של Chaos Engineering;
- שימושים ב - ChatOps;

אז יאללה מוכנים? בואו נצלול פנימה...

תפקיד הבדיקות ב-DevOps או למה צריך בודקים בצוות שכזה??

ראשית מה זה DevOps?

DevOps, זו תרבות ארגונית המשלבת מתודולוגיות, פרקטיקות וכלים, במטרה לשפר את תהליכי הפיתוח והתפעול של המוצר. המונח "DevOps" משלב בין פיתוח (Development) לתפעול (Operations) ומטרתו היא לשפר את שיתוף הפעולה בין צוותי הפיתוח והתפעול על מנת לייעל תהליכי עבודה, לקצר את זמן הפיתוח ולהגיב מהר יותר לשינויים ותקלות.

צוות שכזה פועל יחד, משתף ידע ומשלב אוטומציה בתהליכי פיתוח, בדיקה, התקנה, ניהול תשתיות והפעלת התוכנה.

ה-CD (Continue Development) וה-CI (Continue Integration) וה-CT (Continue Testing), הם הבסיס לתהליכים מהירים, בטוחים ויעילים יותר, שמאפשרים גם הפצת פיצ'רים חדשים לסביבת הייצור בקלות ובמהירות.

בצוותי DevOps תפקיד הבודקים שונה מאשר בשיטות פיתוח מסורתיות (כמו אג'ייל ומפל מים) הבדיקות (CT) משולבות לכל אורך תהליך ה-CI/CD מה שיכול לגרום לשיטות בדיקה מסורתיות להיראות פחות מתאימות. לעיתים אנשי DevOps אינם בטוחים כיצד לשלב בדיקות בתהליך, או סבורים שהן אינן הכרחיות, משום שהגישה שלהם מתמקדת בתיקון מהיר ופריסה מחדש של התוכנה. עם זאת, התפיסה הזו השתנתה עם הזמן.

הפלטפורמה של CI/CD מאפשרת ביצוע בדיקות איכותיות לאורך כל הדרך. במילים אחרות, צוותי ה-DevOps חייבים לספק תוכנה יציבה אשר יכולה לעמוד בשימוש בסביבת הייצור, ולבודקים יש תפקיד קריטי בהבטחת האיכות הזו. הם אחראים לתכנון אסטרטגיית בדיקות שתכסה גם את תהליך ה-CI/CD וגם את התוכנה עצמה.

כיצד בודקים משתלבים בצוותי DevOps?

צוותי DevOps לא תמיד יקדישו מספיק מחשבה לבדיקות. הבודקים צריכים ליזום ולהכניס את עצמם לזרימות העבודה (workflows) זה אומר לאתר הזדמנויות ליישם בדיקות בכל שלב – לא רק בפיתוח ואינטגרציה, אלא גם בסביבת הייצור.

כדי להיות ספציפיים יותר, בודקים בצוותי DevOps עשויים למצוא את עצמם פועלים כ:

- יועצי QA;
- יוצרים תקני בדיקות ויעדים;
- משתפים פעולה עם המפתחים לקביעת מהות הבדיקות וטכניקות הבדיקה המתאימות;

לפיכך אנו מבינים שלאיכות הבדיקות יש השפעה רבה על הצלחת המוצר בפרט והארגון בכלל. להלן מספר טיפים לשיפור אסטרטגיית הבדיקות בצוות DevOps שהטמענו בקבוצה שלנו:

1. יש לזהות את כל מקרי הבדיקה (Test Cases) שיש לבצע עבור כל שלב בתהליך ה-CI/CD.
2. במקרים של אזורי קוד קריטיים שעלולים להשפיע על המערכת כולה, יש לבצע בדיקות מקיפות יותר.
3. הבדיקות צריכות להיות ממוקדות ואפקטיביות – כלומר, רק הבדיקות החיוניות שיבטיחו יציבות ואיכות המערכת צריכות להתבצע, מבלי להכביד על התהליך בבדיקות מיותרות.

4. אין צורך לבצע את כל ה-Regression Test Cases כדי לבצע את הבדיקות מסעיף 2.
5. יש להטמיע Coverage tools ו-specialized code analysis כדי להבטיח שכל הקוד מכוסה.
6. אסטרטגיית בדיקות לפיצ'ר חדש צריכה להיות סטנדרטית, תוך כדי כתיבת בדיקות אוטומטיות והפעלתן על ה-build החדש.
7. תהליך הבדיקות יימשך עד שהקוד יציב וניתן לפריסה בסביבת הייצור.
8. כל הפריסות צריכות להיות אוטומטיות, תוך הגדרת סביבות בדיקה מתאימות.
9. בדיקות אוטומטיות חוצות פלטפורמות שונות צריכות להתבצע על ידי הבודקים בצוות.
10. יש לבצע בדיקות במקביל (parallel execution) על מנת לחסוך זמן.

סוגי כלי בדיקה ב-DevOps

אז ראינו ש-DevOps מתבסס על CI/CD, ודורש אוטומציה בכל שלב בתהליך, ולכן יש צורך בכלים לניהול גרסאות, אינטגרציה, בדיקות, ניהול מהדורות, פריסה וניטור .ולכן משוב אוטומטי הוא קריטי - מה שהופך את ניטור התשתיות, ניטור האפליקציות והאגרגטורים לכלים חיוניים בשרשרת הבדיקות של DevOps

כלים אלו מבצעים אוטומציה פונקציונלית ולא פונקציונלית, תומכים בדרישות ה-Continuous Testing עליו נרחיב בהמשך ומאפשרים להקים סביבות, לחקות התנהגות של המוצר ולהריץ יישומים מרובים על מערכת הפעלה אחת או יותר. הם מאפשרים גם ביצוע בדיקות אינטגרציה ללא כל הרכיבים הזמינים.

הבא נבחן מספר כלים מרכזיים לבדיקות:

Unit-Tests

בדיקות יחידה הן הראשונות שנעשות ונכתבות כחלק מתהליך הפיתוח המונחה-מבחן (TDD - Test-Driven Development). בבדיקות אלו, כל חלק של הקוד נבדק באמצעות מבחנים שנכתבים עוד לפני שהקוד עצמו מפותח. גישת ה-TDD היא קריטית, כיוון שהיא עוזרת למפתחים לחשוב לעומק על התנהגות כל יחידה בתוכנה.

Apache JMeter

לבדיקות עומס וביצועים, המיועד לבדוק את ההתנהגות הפונקציונלית ולמדוד את ביצועי התוכנה. הוא תומך בבדיקות של יישומים דינמיים וסטטיים, כמו גם אפליקציות Web.

לירן יושינסקי

בעל ניסיון של כ-20 שנה בעולם ההיי טק ובפרט באוטומציה. בשש השנים האחרונות עובד בחברת הסייבר Tufin בתור Automation Tech Lead בקבוצת ה-DevOps ומוביל Automation Guild בחברה. חובב טכנולוגיות ובפרט מכור לכלי Open source בזמני הפנוי עוסק בספורט ימי בעיקר חתירת קיאקים בארץ ובעולם.

אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps - לירן יושינסקי

תכונות עיקריות הן כמפורט להלן:

- בדיקת עומס וביצועים על שרתים, יישומים ופרוטוקולי אינטרנט
- תמיכה בפרוטוקולים כמו LDAP, JDBC, FTP, ו-SOAP/REST Webservices
- משמש כ-IDE לבדיקות פונקציונליות המאפשר גם הקלטה, איתור באגים ובנייה של test plan
- יצירת דוחות HTML דינמיים
- עבודה במתכונת Multi-threading
- תמיכה ב-Continuous Integration על ידי Gradle, Maven ו Jenkins

הערה:
קיימים כלים חדשים ומעניינים המתחרים ב-Jmeter לדוגמא:

K6

כלי קוד פתוח המתמחה בבדיקות עומס וביצועים. הוא מאפשר כתיבת טסטים ב-JavaScript ומכיל מגוון רחב של פיצ'רים מובנים, ללא צורך בתוספים. הוא תומך בפרוטוקולים הבאים:

- Web - HTTP/1.1, HTTP/2 (Java, NodeJS, PHP, ASP.NET, …)
- WebSockets
- gRPC
- SOAP / REST Webservices

Mockito

Framework המיועד ליצירת אובייקטים מדומים (Mocking) ומיועד לפרויקטים שמשתמשים ב-TDD. הוא מסייע בפישוט הבדיקות על ידי הפחתת התלות במהלך הבדיקה, וניתן להשתמש בו כדי לדמות תלות חיצונית כמו DB או Web Services למשל בודקים ב-DevOps יכולים להשתמש ב-Mockito יחד עם Junit כדי לחזק את יכולות הבדיקה.

Junit

כלי פופולרי מבוסס Java גם הוא מעולמות ה TDD המיועד לכתיבת מקרי בדיקה חוזרים (repeatable cases) על גבי Java Virtual Machine (JVM) הכלי מספק תכונות שונות כמו test suites, fixtures, test runners, JUnit classes.

בנוסף מציע API של TestEngine כך שניתן לפתח עליו Testing frameworks. כמו כן, הוא תומך ב-Continuous Integration.

דוגמא קצרה היא בדיקת סטטוס סרוויסים במוצר תוך שימוש ב-Assertions של Junit:

```
fun checkServicesAreReady() {
    jsonResponse.dataStatus?.flatMap { it.readyStateStatus }?.forEach { status ->
        Assertions.assertTrue(status.isReady)
    }
    Gauge.writeMessage("TOS services are ready")
}
```

Pytest

Testing Framework חזק וקל לשימוש עבור חברינו הפייתונים. פופולרי במיוחד לבודקים ב-DevOps המשתמשים בו כדי לכתוב ולהתאים בקלות טסטים על בסיס Python CLI, כך שבעזרתו ניתן לכתוב Test cases פשוטים בבדיקות DB, UI ו-API.

Pytest מסוגל לגלות אוטומטית פונקציות ומודולים. בנוסף, מציע תמיכה בהרצת בדיקות יחידה מובנית.

ישנם יותר מ-1000 פרוייקטים PyPI או תוספים שיכולים לתרום הרבה בתהליך ה-TDD.

דוגמא לטסט קצר (קרדיט לאתר pytest)

```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.x.y, pytest-7.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item

test_sample.py F                                                        [100%]

================================= FAILURES =================================
_____________________________ test_answer _____________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================== short test summary info ==========================
FAILED test_sample.py::test_answer - assert 4 == 5
============================ 1 failed in 0.12s =============================
```

NUnit

אם אתם מפתחים את הטסטים שלכם ב-.NET. או שפות הנתמכות כמו F#, C#, Visual Basic תוכלו להשתמש ב-Framework הזה.

מבחינת פיצרים:

- Nunit 3 Test Adapter מאפשר לנו להריץ טסטים של NUnit 3 בתוך VS code
- ה-NUnit Engine מאפשר להריץ טסטים שפותחו על מספר Test Framework
- VS Test Generator עוזר לייצור IntelliTests ו-Unitests.

TestNG

Framework נוסף המיועד לביצוע בדיקות מגוונות מ-Unitests ועד בדיקות אינטגרציה.

הפיצ'רים העיקריים שמבדילים אותו מ-Junit ו-Nunit

- אנוטציה של Unitests cases
- בדיקת Multi-threading
- בדיקת מבוססת נתונים (Data-Driven)
- מגוון רחב של plug-ins וכלים עבורIDEA, Eclipse, Selenium Maven, Ant ועוד.

הרבה בודקים בצוותי DevOps משתמשים בפיתוח מונחה בדיקות קבלה ATDD (Acceptance Test-Driven Development) ופיתוח מונחה התנהגות BDD (Behavior Driven Development) שמאפשרות הרצה חוזרת ונשנית של בדיקות תוך כדי בניית רכיבי תוכנה מורכבים.

Robot Framework

כלי ATDD פופולרי המשתמש בגישת keyword-driven ומאפשר לכתוב תסריטי בדיקה בצורה קריאה ופשוטה תומך בטכנולוגיות שונות ומשתלב עם כלים וספריות שונות, מה שהופך אותו לגמיש וניתן להרחבה.

Gauge

כלי BDD אותו סקרתי במאמרי הקודם מעודד שיתוף פעולה בין בודקים, מפתחים וצוותים נוספים בארגון, על ידי כתיבת תסריטי בדיקה בשפה קריאה וברורה.

לדוגמא: בחברה שבה אני עובד, המוצר נמצא על תשתית קוברנטיקס- K3s שהיא גרסה קלה של K8s לסביבות עם משאבים מוגבלים.

אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps - לירן יושינסקי

בקובץ הקונספט הבא בוחנים תסריט עבור התקנה על תשתית זו

```
k3sValidation.cpt

This is a concept file with following syntax for each concept.

# As a user, I want to be able to validate k3s install finished successfully
* Run the kubectl version on "node1" command
* Verify cluster k3s Health on "node1"
* Verify the k3s secret "tufin-deploy-id" on "node1" is exist
* Print all nodes info
```

במקטע הקוד הבא אנו משתמשים בפניות API לקליינט קוברנטיקס של Fabric8 כדי לבחון את מצבו של הקלאסטר.

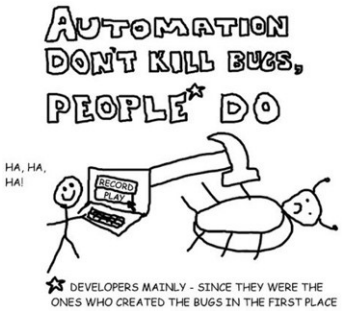
עם זאת אומר שבצוות DevOps אין כלי "הטוב ביותר" לבדיקה. לכל צוות יש את הכלים שיכולים לענות על הדרישות הספציפיות של המערכת והבדיקות הנדרשות. הכרת הכלים המתאימים והטמעתם בצורה נכונה מאפשרת אוטומציה של תהליכים ותורמת לשיפור איכות התוכנה.

```
KubernetesClient.kt

fun runGetKnodesIn() {
    val namespaces = client.namespaces().list().items
    val clusterMetricsList: MetricAPIGroupDSL? = client.top()
    val nodeMetricsList: NodeMetricsList? = clusterMetricsList?.nodes()?.metrics()
    val workerNodes = client.nodes().list().items.filter { it.metadata.labels["node-role.kubernetes.io/worker"] == "true" }

    try {
        val k3sStatus = sshClient.runSshCommand( command: "sudo systemctl is-active k3s")
        val k3sMessage = if (k3sStatus.trim() == "active") "K3s is running." else "K3s is not running."
        Gauge.writeMessage(k3sMessage)
        sshClient.runSshCommand( command: "sudo systemctl status k3s")

        for (node in workerNodes) {
            val nodeName = node.metadata.name
            val nodeConditionReady = node.status.conditions.find { it.type == "Ready" }
            val nodeReadyStatus = if (nodeConditionReady?.status == "True") "ready" else "not ready"
            Gauge.writeMessage("Worker node '$nodeName' is $nodeReadyStatus.")
        }

        Gauge.writeMessage(nodeMetricsList.toString())

        val nodeMetrics = nodeMetricsList?.items
        if (nodeMetrics != null) {
            if (nodeMetrics.isNotEmpty()) {
                nodeMetrics.forEach { nodeMetric ->
                    val nodeName = nodeMetric.metadata?.name
                    val nodeCpuUsage = nodeMetric.usage["cpu"]
                    val nodeMemoryUsage = nodeMetric.usage["memory"]
                    val nodeDiskUsage = nodeMetric.usage["disk"]
                    val nodeNetworkUsage = nodeMetric.usage["network"]

                    writeMessage("Node", nodeName)
                    writeMessage("CPU Usage", nodeCpuUsage?.amount)
                    writeMessage("Memory Usage", nodeMemoryUsage?.amount)
                    writeMessage("Disk Usage", nodeDiskUsage?.amount)
                    writeMessage("Network Usage", nodeNetworkUsage?.amount)
                    Gauge.writeMessage("Cluster is available. Found ${nodeMetrics.size} nodeMetrics.")
                }
            } else {
                Gauge.writeMessage("No node metrics found.")
            }
        }
    } catch (ex: KubernetesClientException) {
        logger.error("Error while checking cluster health: ${ex.message}", ex)
    }
}
```

תנאי קדם ל-continuous testing ב-DevOps

במהלך השנים האחרונות, התפתחו כלים לביצוע בדיקות אוטומטיות (Automated Testing) שהפכו לפופולריים מאוד בזכות יעילותם ומהירותם.

בדיקות אלה כוללות אינטגרציה, בדיקות מערכת, ביצועים, רגרסיה, ובדיקות קבלה.

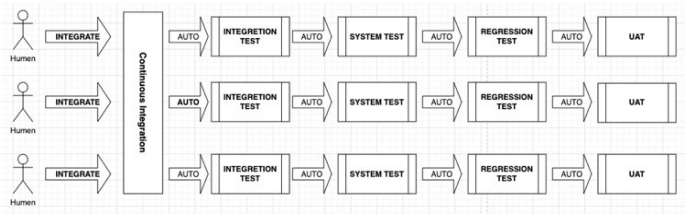


התהליך מוגדר כך שהקוד שנכתב עובר דרך כלי בדיקות, המבוססים על סקריפטים אשר מריצים את הבדיקות באופן אוטומטי. מלבד כתיבת הסקריפטים, אין מעורבות אנושית בתהליך.

Continuous Testing הוא תהליך אוטומטי גם הוא, אך הוא מהווה התקדמות נוספת לעומת הבדיקות האוטומטיות. בתהליך זה, הבדיקות מתחילות לפעול מיד לאחר שהקוד שולב (Continuous Integration) כאשר כל תסריטי הבדיקות כתובים מראש.

הבדיקות האוטומטיות רצות ברצף, ללא כל צורך במעורבות נוספת.

דוגמא לתהליך Continuous Testing

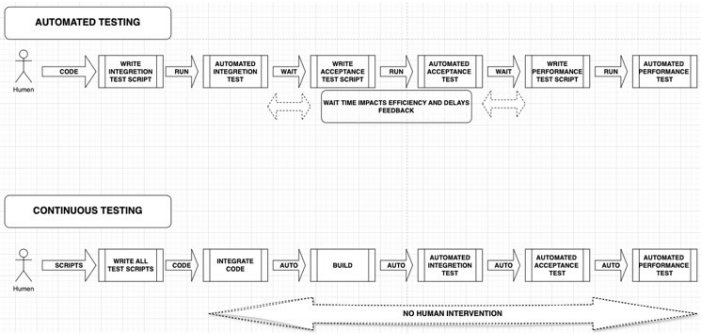


אז מהו ההבדל העיקרי בין Automated Testing ל-Continuous Testing?

ב-Automated Testing הבדיקות נכתבות לאחר שהקוד כבר שולב, והבדיקות מתבצעות מול הקוד באמצעות תסריטי בדיקה מותאמים.

לעומת זאת, ב-Continuous Testing, תסריטי הבדיקות נכתבים לפני שהקוד מפותח, כך שכאשר הקוד משולב, הבדיקות רצות אוטומטית אחת אחרי השנייה.
היתרון המרכזי של Continuous Testing הוא זמן תגובה מהיר יותר. ברגע שהקוד משולב, התהליך מתחיל באופן מיידי, והמשוב מתקבל כמעט בזמן אמת, ללא הפסקות או עיכובים הנובעים מהתערבות אנושית.

ניתן לראות בתרשים את ההבדלים



בסביבות עבודה מבוססות DevOps הבדיקות הללו חיוניות כדי לשמור על תהליך בדיקה רציף ומהיר. עם זאת, יש לזכור שהכנת סקריפטים ותחזוקתם דורש מאמץ ומשאבים. לכן, למרות היתרונות של בדיקות אוטומטיות בכלל, עדיין בדיקות ידניות נחוצות להשלמת התהליך ולהבטחת בדיקת המוצר באופן מקיף.

הבודקים הם שומרי האיכות לאורך כל תהליך ה-SDLC. ניהול איכות התוכנה כולל הרבה מעבר למשימות אוטומטיות. לכן, כישורי תקשורת הם קריטיים עבור בדיקות יעילות.

על הבודקים לשתף פעולה ולשתף ידע עם כלל חברי צוות ה-DevOps מפתחים, אנשי תפעול, מומחי אבטחה ואנשי מוצר.

חשיבות התרבות הארגונית ב-DevOps

התרבות הארגונית של ה-DevOps נשענת על תקשורת פתוחה לכל אורך ורוחב הארגון. הבודקים משתפים פעולה עם צוותים מגוונים כדי לעמוד ביעדים ארגוניים ובכך תופסים תפקיד מרכזי. תיאום בין היעדים הטכניים ליעדים העסקיים הוא חיוני, והבודקים צריכים להיות יצירתיים וחדשניים בפתרון תקלות ובהערכת איכות. המיומנויות הללו הן הכרחיות כדי להשיג Continuous Testing אמיתי, יש צורך בשינוי תרבותי רחב. כל חברי הצוות צריכים לאמץ גישה של אחריות לאיכות, אך הבודקים – המומחים לאיכות – הם אלה שצריכים להוביל את השינוי הזה.

כדי לקדם Continuous Testing יש להסתמך לא רק על כישורים טכניים, אלא גם על כישורי מנהיגות כמו תקשורת, שיתוף פעולה וחשיבה יצירתית.

למשל תנאים מוקדמים לטובת Continuous Testing שהנחלנו בצוות שלנו:

דברו איתנו - תקשורת היא לעיתים מאתגרת, במיוחד כשכל אדם מפרש רעיון או פתרון באופן שונה. הדרך הטובה ביותר לוודא שהמסר עובר בצורה ברורה היא לדבר באופן ישיר וגלוי. שימו בצד את הסחות הדעת, כמו טלפונים, במהלך פגישות, והקדישו זמן לשיחה משמעותית.

אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps - לירן יושינסקי

עבדו איתנו - שיתוף פעולה הוא מפתח להצלחה. כשלוקחים רעיון ומפתחים אותו יחד, הסקופ נהיה ברור ומוגדר יותר. כדי לעמוד בקצב האיטרציות המהירות של צוותי DevOps, חשוב שכולם יעבדו לפי אותו הסקופ. שקיפות במדדים עוזרת לבודקים להגביר את הנראות ולהבטיח שכל הצוות מבין את ההתקדמות והאתגרים.

צרו איתנו - שיתוף פעולה אינו רק מייעל תהליכים, אלא גם מטפח יצירתיות וחדשנות. לעיתים, צוותים נוטים לפנות לפתרונות מוכרים מהעבר, אך הדבר מגביל את היכולת לפתח גישות חדשות. גיוון בתחומי עניין ומומחיות מביא למגוון רחב של רעיונות ומעודד חדשנות אמיתית.

חשבו איתנו - הבנת החשיבה והמיומנויות של חברי הצוות – לצד מודעות עצמית – מאפשרת להוביל שינוי תרבותי שבו כל אחד בצוות אחראי לאיכות.

היתרונות של chaos engineering ב-DevOps

הנדסת כאוס (Chaos Engineering) הינו מושג פופולרי בעולם הפיתוח וההנדסה כיום. ארגונים ומהנדסים רבים מזהים את היתרונות שביצירת "כאוס" מבוקר, כדי לאתר כשלים בפיתוח ולדעת כיצד להתמודד איתם בסביבת הייצור.

לפי אחת ההגדרות המקצועיות, הנדסת כאוס כוללת ביצוע ניסויים במערכות מבוזרות במטרה ליצור כשלים מלאכותיים, מתוך כוונה לבחון את יכולת המערכת לעמוד בתנאים מאתגרים ולהגביר את האמון בעמידות שלה בסביבות ייצור בלתי צפויות.

המשמעות של Chaos Engineering הינה יותר מאשר בדיקת איכות המוצר, אלא בוחן לכל סביבת הייצור שמאפשר לארגון לבצע shift right שהוא מושג המרחיב את תהליך הבדיקה גם לסביבת הייצור, שזה תרגול חשוב של DevOps.

הטכניקה הזו מהווה דרך עבור בודקים בצוותי DevOps למצוא תחום אחריות ייחודי עבורם.

לפיכך אלה הן היתרונות:

1 שיפור יכולות הבדיקה של עמידות המערכת

בודקים משתמשים ב-Chaos Engineering כדי לבחון תרחישים של כשלי מערכת בזמן אמת. הדבר מאפשר להם לבדוק את עמידות המערכת בצורה מקיפה יותר, מעבר לבדיקות מסורתיות של פונקציונליות וביצועים. יחד עם זאת קיים חשש בקרב בודקים מנוסים. אחרי הכל, ניסויי כאוס מכוונים לגרום ליישומים להיכשל בסביבת הייצור ולא בסביבת staging כפי שהתרגלו במשך שנים.

2 חשיפה מוקדמת לכשלים פוטנציאליים

Chaos Engineering עוזר לבודקים לזהות בעיות פוטנציאליות באזורים שנוטים לכשל, כמו בעיות ניהול עומסים, תלות ברכיבים שונים, או תגובות המערכת לשגיאות לא צפויות.

3 בדיקות בסביבה קרובה לייצור

Chaos Engineering מאפשר לבודקים לבחון איך המערכת מגיבה למצבי קיצון וכשלים לא מתוכננים. כך מתקבלות תוצאות בדיקה אמינות יותר מאשר בתרחישים סימולטיביים בלבד בסביבות שהן לא ייצור.

4 שיפור באיכות התשתית

בעזרת Chaos Engineering בודקים יכולים לבחון לא רק את המערכת האפליקטיבית אלא גם את התשתית. זה כולל בדיקות תקלות ברכיבי רשת, בסיסי נתונים, או מערכות אחסון, ולוודא שכולם מתפקדים כראוי תחת עומס או כשל.

5 הכנה טובה יותר לתרחישי כשל

Chaos Engineering מסייע לבודקים לדמות תרחישי כשל אמיתיים, כך שצוותים יוכלו לתעד וללמוד איך המערכת מתנהגת במצבים כאלה, ולוודא שקיימות מערכות גיבוי והתאוששות יעילות.

6 העצמת שיתוף הפעולה עם צוותי DevOps

כשבודקים משתתפים בניסויי כאוס הם עובדים בצורה יותר קרובה עם אנשי DevOps הדבר מאפשר שיתוף ידע ותהליכי עבודה משותפים, מה שמוביל לשיפור בבדיקות ובאיכות הכללית של המערכת.

7 הגברת איכות החוויות של המשתמשים

על ידי זיהוי ושיפור המערכת תחת מצבי קיצון, בודקים עוזרים להבטיח שהמשתמשים הסופיים יתקלו בפחות תקלות ותקלות פחות חמורות. זה משפר את שביעות רצון הלקוחות ואת יציבות המוצר.

8 אוטומציה של בדיקות כשל

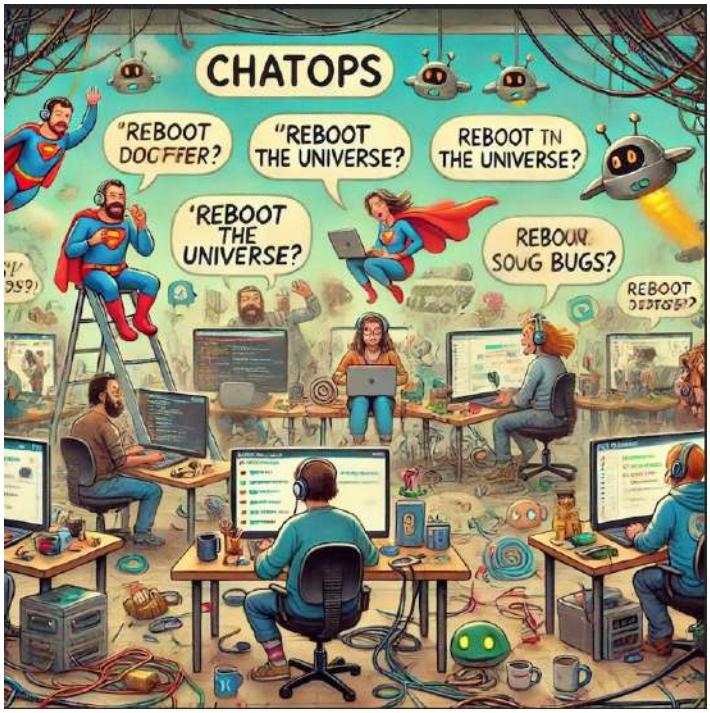
בודקים יכולים לשלב Chaos Engineering כחלק מתהליכי האוטומציה שלהם, וליצור מערך של בדיקות שמדמה כשלי מערכת כחלק רציף מתהליך הבדיקה האוטומטי. למשל להשבית מכונות וירטואליות, לדמות מתקפת מניעת שירות (DDoS) או ליצור בעיות רשת. כך יוודאו שמערכות שנבדקות עומדות בכשלי זמן אמת באופן קבוע.

באמצעות שימוש ב-Chaos Engineering בודקים יכולים לשפר את אמינות המערכות, להבטיח תגובות נכונות למקרי קצה, ולתמוך בתהליך ה-DevOps בצורה מקיפה יותר.

גם אצלנו בחברה התחלנו ליישם את הטכניקה הזו ועל כך אספר במאמר הבא שלי....

שימושים ב-ChatOps ב-DevOps

ChatOps הוא גישה המשתמשת בערוצי תקשורת שונים – כמו מערכות לניהול Tickets וכלים לשיתוף פעולה כמו Microsoft Teams, Slack ו-Twist כדי ליצור מסגרת לסביבת שיתוף פעולה מלוכדת. התיאור הזה אינו עושה חסד עם מערך הטכנולוגיות המיועד לפתרון מהיר של בעיות בסביבת הייצור באמצעות שיתוף פעולה.



אסטרטגיית בדיקות: שיטות עבודה מומלצות, יתרונות וכלים בצוות DevOps - לירן יושינסקי

נניח שיש תקלה בלתי צפויה בקוד או תקלת ביצועים תשתיתית. התראה נשלחת לצוות תמיכה, שמובילה לפתרון מהיר. אם התקלה מורכבת יותר, היא תצטרך לעבור אסקלציה, אך למי? ל-Devops או למפתח? כאן גישת "הצוות" עובדת בצורה הטובה ביותר.

ChatOps היא המפתח לגישת "הצוות" הזו. כפי שאמרנו התקלה מוסקלת – לא לאדם בודד, אלא לקבוצה מוגדרת. באופן אידיאלי, אחד מחברי הצוות יודע כיצד לאבחן ולפתור את התקלה, אך לרוב נדרש שיתוף פעולה בין מספר אנשים כדי להבין ולפתור אותה. באמצעות שיתוף הידע על היישום, התשתית והקוד, ניתן להגיע לאבחון מהיר ומדויק (בתקווה) ולפתרון.

בנוסף, ChatOps מאפשר שימוש בבוטים או בכלים אוטומטיים אחרים לעבודה עם שרתים או יישומים מרחוק. בוטים אלה למשל יכולים להפעיל מחדש את השרת, לשנות הגדרות תצורה או לאבחן את התקלה בצורה מעמיקה יותר.

ב-ChatOps השלם גדול מסך חלקיו. אנשים בעלי מיומנויות שונות יכולים לשתף פעולה בזמן אמת כדי לפתור תקלות במהירות.

כלי ChatOps

מוצרים כמו Slack ו-Microsoft Teams מאפשרים למשתמשים ליצור קבוצות ולפתוח ערוצי תקשורת, על מנת להקל על תקשורת קבוצתית מתוכננת.

כלים אלו מאפשרים לצוותים לתקשר במהירות ולהחליף מידע ,תמונות וקבצים.

לניהול תקלות ואסקלציות, יישומים כמו Splunk On-Call ו-PagerDuty מספקים מנגנון ניתוב לניהול תקלות. יישומים אלו קובעים את סדר העדיפויות של סוגי התראות ומוודאים שההתראה מגיעה לאנשים המוסמכים ביותר לפתרון התקלה. כלים אלו משתלבים עם מערכות הודעות מיידיות ומנועי התראות ליצירת גישה אוטומטית לקבלת התראות, ניתובן, תקשורת ופתרון תקלות.

תפקיד הבודקים בעולמות ה-ChatOps.

יופי הגענו לחלק המעניין.

במקרים רבים, לבודקים יש ידע רלוונטי. ייתכן שהתקלה נראתה בעבר במהלך הבדיקות, והם יכולים לחפש במהירות במאגר הבדיקות כדי לקבוע מה היה הפתרון.

בודקים הם תורמים חשובים בגישת "הצוות" לפתרון תקלות. הם ממוקדים והמיומנים ביותר בניתוח התנהגות המוצר ובהסקת הגורם השורשי להתנהגות זו. למרות שבודקים לא בהכרח יתקנו את התקלה, הם לרוב הראשונים שמזהים את הפתרון.

בודקים תורמים מניסיונם ביישומים ובפתרון תקלות. הם מצטרפים למפתחים, מומחי רשת, מנהלי מערכות, מומחי אבטחה ואחרים שאחראים להבטיח שהמוצר מפותח, נבדק, נמסר ומתופעל בהתאם לציפיות.

להלן מספר היבטים לשימוש בכלים אלה לטובת בודקים:

1. בדיקות אוטומטיות: בודקים יכולים להגדיר ולנהל אוטומציה של תהליכים שקשורים לבדיקות דרך כלי ChatOps. למשל הם יכולים להקים תסריטים שמבצעים בדיקות אוטומטיות על פי פקודות שנשלחות דרך כלים אלו.
2. תגובה מהירה לתקלות: כאשר תקלות מזוהות במהלך הבדיקות, הבודקים יכולים להשתמש בכלי ChatOps כדי ליידע את הצוותים הרלוונטיים במהירות ולנהל את תהליך תיקון התקלות.
3. שיתוף פעולה ותקשורת: בעזרת כלים אלה, הבודקים יכולים לשתף תוצאות בדיקות, לדון בבעיות ולשתף משוב עם חברי הצוות האחרים כמו מפתחים ומנהלי פרויקטים, מה שמסייע בתיאום מהיר ויעיל.
4. מעקב וניהול משימות: הבודקים יכולים לעקוב אחרי התקדמות הבדיקות, לנהל את המטלות השונות, ולוודא שהכל מתנהל לפי התוכנית. זה כולל קביעת סטטוסים, עדכון צוותים אחרים על התקדמות והקצאת משימות חדשות.
5. אינטגרציה עם כלים נוספים: הבודקים צריכים להכיר את האינטגרציות בין כלי ChatOps לבין כלים אחרים כמו Jira ,Jenkins, ו-GitLab הם יכולים לנהל ולפקח על תהליכים שונים, כולל השקת בדיקות אוטומטיות, בדיקת קוד, ותיאום עם כלים אחרים.
6. אופטימיזציה ושיפור תהליכים: באמצעות ChatOps הבודקים יכולים לנתח את האופן שבו התהליכים פועלים ולהציע שיפורים או אופטימיזציות שמבוססות על הנתונים שנאספו במהלך הבדיקות.

דוגמא בחברה אצלנו היא השימוש של Channel ייעודי ב-Slack לטובת תוצאות הרצה של Jobs ב-Jenkins שהם למעשה E2E tests

או למשל התרעה על פתיחת טיקט במערכת ה-Jira כתוצאה מתקלה שהתקבלה במהלך בדיקות אוטומטיות

את המאמר אני מקדיש לזכרו של אל"מ יצחק בן בשט (בנבה); ז"ל הי"ד שנהרג במלחמת חרבות. ברזל.

עושים QA לקריירה
כשהבאג מהג׳ירה זולג לקפיטריה | איילת מלמד כהן

איילת מלמד כהן

מנטורית, מאמנת בינ״ל בשיטת Elevate לפיתוח מנהיגות.

20 שנים בעולם האיכות, כמנהלת בכירה של קבוצות פיתוח, דאטה ו-QA. בנוסף לאימון, איילת מלווה טארטאפים בכל הקשור לניהול איכות, פיתוח צוותים ואסטרטגיות QA מותאמות לארגון.
מאמנת מוסמכת בדרגת PCC, בעלת תואר ראשון ושני במנהל עסקים וכלכלה מאוניברסיטת בר-אילן.
אמא לשלושה וזוקפת לזכותם חלק גדול ממיומנויות הניהול שלה

אתם פותחים את הסלאק, מקבלים הודעה מהמפתח שאתם עובדים אתו, ״סיימתי, הפיצ׳ר מוכן לבדיקות״, הסטטוס בגי׳רה עודכן, ועכשיו המשימה ב-To Do שלכם. הבדיקות שכתבתם מוכנות, אתם מתקינים גרסא, מריצים כמה בדיקות שפיות (Sanity) ומוצאים באג, שרק מי שהיה עושה כמה בדיקות בסיסיות לפני שמסר לכם גרסא, היה מוצא (וחוסך לכם זמן יקר).

זה לא מפתיע אתכם, זאת גם לא הפעם הראשונה שזה קורה מול מפתחים בצוות הזה, ואתם כבר למדתם איך מתנהלים. אז לא פתחתם מיד באג, אלא העדפתם לכתוב לו בחזרה בסלאק, או קמתם לשולחן שלו כדי לספר על הבאג ועל כך שאין מה להמשיך בדיקות עם באג כזה. בדיילי אתם נותנים סטטוס ומספרים שהגרסא חזרה לפיתוח, והסטטוס בגי׳רה - התעדכן בהתאמה. ואז, ממש לפני שאתם מתקדמים למשימה הבאה, אתם עוברים דרך הקפיטריה, ופוגשים שם את אותו מפתח. תוך כדי שאתם מכינים לכם משהו לשתות, אתם שמים לב שמשהו שונה בהתנהגות שלו כלפיכם, ולא לטובה. בפגישה אחרת שאתם יחד באותו יום, אתם שומעים אותו מעביר ביקורת על ה-QA וההתנהלות בצוות מול המנהל שלכם, מקבלים למייל זימון לפגישת ״QA Handover״ שהמנהל שלו יזם, כי בפיתוח חושבים ש״צריך לעשות סדר בבדיקות שה-QA עושים״.

מוכר? זאת רק דוגמא, אך כולנו מכירים את הדפוס. עניין מקצועי הופך לאישי, ומלהתעסק ב״אישיו״ אתם מוצאים את עצמכם מתעסקים בעיקר ב״איש״ (או האישה).

אני פגשתי את זה בכל ארגון שבו עבדתי. בייחוד כשהפכתי למנהלת. שמתי לב לעובדה שמסביבי קולגות, שההתנהלות המקצועית שלהם, היא קודם כל אישית. בכל ארגון, היו מפתחים או מנהלים שהתגובה שלהם לדוח הסטטוס של ה-QA, הייתה כזאת שגרמה לי להשקיע שעות בלבחור מילים בכל מייל שהם או המנהלים שלהם מכותבים אליו.

באחד הארגונים בהם עבדתי, היה מנהל פיתוח שטען שהוא ״לא צריך את ה-QA״, ושהאיכות מטופלת InHouse בצוותים שלו. באגים מלקוחות, זמן שנחסך בזכות בדיקות ה-QA וההנחיות שקיבל מהמנכ״ל לגבי איכות לא הצליחו לשנות את מה שמבחינתו, הוא יודע, צודק ומבין הכי טוב מה נכון. הפער נשמר וכך עבדנו כל אחד בגזרתו התקדם ועבד מעל פני השטח. לכאורה יחסים מקצועיים תקינים.

ומה מתחת לפני השטח - ?

בכל ארגון תמצאו אותם: קונפליקטים גלויים וסמויים, על תהליכים או סדרי עדיפויות, על משאבים, תשומת הלב מההנהלה, או על המקום בשולחן מקבלי ההחלטות. כל ארגון בנוי מתפקידים שיחד אמורים לתפקד כמו מנגנון משוכלל. מעין מכונת דליברי שצריכה תמיד לעבוד. וכמו כל מנגנון, By Design, יהיו לפעמים חריקות ותקיעויות.

> ״כולנו מכירים את הדפוס. עניין מקצועי הופך לאישי, ומלהתעסק ב״אישיו״ את מוצאים את עצמכם מתעסקים בעיקר ב״איש״ (או האישה)״

אז בואו נכיר בזה, שבין אם זה על בסיס אישי או מקצועי, גלוי או סמוי, בגלל התפקיד או התפקוד, שלכם או של הצד השני - קונפליקטים הם חלק מהותי, ונורמלי בבניית כל מערכת יחסים, מקצועית או אישית, ולכן תמצאו אותם בכל ארגון ותפקיד. בזכות הקונפליקטים יחסים מתפתחים: כשיש פער, אנחנו בוחנים גבולות. בזכות ההתנהלות בזמן קונפליקט אנחנו מכירים טוב יותר את עצמינו, את מי שמולנו ומבינים מה הכי חשוב לנו. פתרון הקונפליקט וההתאוששות ממנו, מניחים את היסודות להמשך היחסים, שיתוף פעולה אפקטיבי ומזינים את חווית העבודה ורמת האנרגיה שלנו.

אז אחרי שהבנו שזה נורמלי שזה קורה גם לנו, ניגש להבין את הקונפליקט ואת עצמנו בו. כמו שאתם יודעים, אין מתודולוגית QA לפתרון קונפליקטים מקצועיים או אישיים בעבודה מול פיתוח או פרודקט. במצבים כאלו, מתקיים מעין מערך ניסוי בו שני הצדדים לומדים איך להתנהל זה לצד זה, אך אם כן הייתה מתודולוגיה היא הייתה מתחילה בהגברת מודעות והבנה מעמיקה יותר של מקור הקונפליקט והתרומה שלנו לקיומו.

קחו קונפליקט שיש לכם, אחד שפוגע בתפקוד שלכם ו/או של הצוות שלכם, ואפיינו אותו:

האם הקונפליקט גלוי או סמוי?

קונפליקט גלוי מקצועי אומר שהוא כולל מחלוקות סביב תהליכי עבודה, תפקידים, חלוקת אחריות, או החלטות ניהוליות שלא מקובלות על הצדדים. הוא נוגע ישירות לעבודה (אין בו משהו אישי) ולרוב נובע מהבדלים בדעות לגבי מה צריך להיעשות ואיך.

דוגמאות: פינג פונג בין ה-QA לפיתוח על אותו עניין, שוב ושוב. בדיקות שפיות נכשלות ברוב הגרסאות. אותו באג חוזר מהשטח שוב ושוב. אתם בקוד ריוויו, ומי שנותן לכם פידבק חושב שהדיזיין היה שגוי מהיסוד ואתם מחליפים דעות.

כיוון שהוא גלוי, אין מניעה משני הצדדים לנהל עליו שיח, ויש מוטיבציה הדדית, להתקדם ממנו.

פתרון קונפליקטים מהסוג הזה הוא לרוב בתוך הצוות, או בין צוותים ללא צורך לערב את הדרג הניהולי הבא, ויכלול חידוד נהלים וציפיות לגבי תחומי אחריות הצדדים - כמו הסכמה על בדיקות שעושים לפני העברת גרסא, איך נקבעת רמת חומרת הבאגים, מה עושים כשאין הסכמה וכו׳.

בארגונים בהם יש רובד סמוי עמוק - בתרבות הארגונית ובסגנון הניהולי - גם קונפליקטים פשוטים יצריכו מאמץ גדול ומעורבות הנהלה.

קונפליקט סמוי יכול להיות על בסיס אישי או מקצועי. אם הקונפליקט הוא ברמה האישית, אתם מרגישים את זה היטב, התחושות הולכות איתכם לישון ומופיעות שוב ושוב מול הצד השני. קונפליקט כזה נוגע

עושים QA לקריירה
כשהבאג מהג'ירה זולג לקפיטריה | איילת מלמד כהן

לרגשות, תחושות חבויות או חששות שאינם מובעים בצורה ישירה, מייצרים מתח ומשפיעים על התקשורת שלכם עם הצד השני וחווית העבודה.

קונפליקט סמוי ברמה המקצועית מתרחש כאשר יש חוסר הסכמה לגבי נושאים מקצועיים, אך הוא לא נאמר באופן ישיר ונשאר בתת-המודע הארגוני. ההשפעה שלו על שביעות הרצון בעבודה היא עצומה ותוצרי הלוואי שלה הם ירידה בביצועים, אנרגיה ניהולית גבוהה שנדרשת כדי "שהכל ינגן" וגם - יותר ימי חופש וימי מחלה, כי מי רוצה להרגיש את אלו כל יום ברצף.

דוגמאות: שוב מצפים מכם לקחת אחריות על משהו שאתם לא אחראים עליו, אתם לוקחים את זה עליכם וממשיכים עם זה בבטן. יש לכם קולגה שאתם חושבים שתפקידו מיותר או שחושבים שתפקידכם מיותר כדוגמת "לא צריך את ה-QA" מתחילת המאמר. במקום לפתור את זה אתם נמנעים משיתוף פעולה, דוחים פגישות אתו או חווים את זה מהצד השני.

"אין מתודולוגית QA לפתרון קונפליקטים מקצועיים או אישיים בעבודה מול פיתוח או פרודקט. במצבים כאלו, מתקיים מעין מערך ניסוי בו שני הצדדים לומדים איך להתנהל זה לצד זה, אך אם כן הייתה מתודולוגיה היא הייתה מתחילה בהגברת מודעות והבנה מעמיקה יותר של מקור הקונפליקט והתרומה שלנו לקיומו"

פתרון לקונפליקטים סמויים הם מורכב, מערב את המנהלים שלכם, ולעיתים HR או מנטור/מאמן ארגוני. אך, אם יש משהו אחד שאתם כן יכולים לעשות הוא לענות - **מה התרומה שלי לקיומו?** מה עשיתי או לא עשיתי כדי שהקונפליקט הזה יתקיים? מה אני עושה היום כדי שהוא ימשיך?

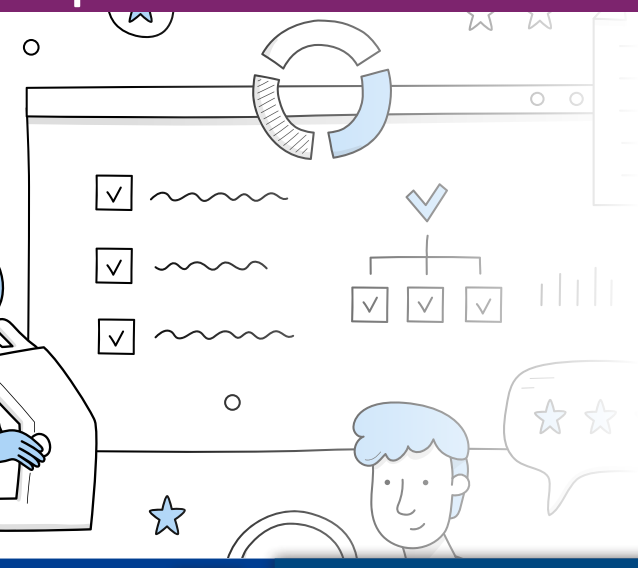
פרד קופמן, פרופ' מ-MIT וסמנכ"ל המנהיגות של גוגל, אומר "אתם לא יכולים להיות חלק מהפתרון אם אתם לא מסכימים שאתם חלק מהבעיה" אז איך אתם חלק מהבעיה?

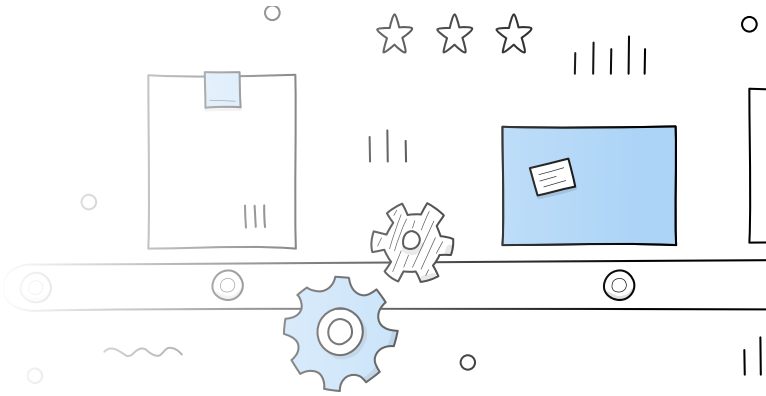
זאת השאלה הכי מאתגרת, אך בה חלק גדול מאסטרטגית הפתרון. כיוון שרק המעשים שלנו בשליטתנו, ואנו רק יכולים לנסות להשפיע על הצד השני, אך לא לשלוט בתוצאות. הדרך הקצרה ביותר להתקדם תעבור בהכרח, דרך אחריות אישית ופרואקטיביות שלנו בדרך אל פתרון.

אתם מכירים את האמרה "עובדים לא עוזבים ארגון, הם עוזבים מנהלים"? מניסיוני עובדים לא עוזבים תפקיד, הם עוזבים סביבת עבודה שלא נמצאת בהלימה עם הערכים והציפיות שלהם. מנהלים הם חלק מזה, אך יש רובד ארגוני שמכתיב את החוויה לא פחות. אם אתם חשים שהקונפליקט שלכם הוא סמוי, אל תישארו אתו לבד. בדקו מה

ההשפעה של הקונפליקט הזה עליכם ועל הצוות שלכם, ואם המחיר גבוה ממה שאתם מוכנים להמשיך לשאת, תשתפו את מי שנראה לכם שהכי יוכל לעזור לכם.

אתם אולי חלק מהבעיה, והקונפליקט, אך אתם יכולים להיות אלו שמנהיגים את הפתרון ואת עצמכם בכך.





בדיקות קופסא לבנה
White box Testing

בדיקות קופסא לבנה הן סוג של בדיקות תוכנה שמתמקדות במבנה הפנימי של רכיב או מערכת. בניגוד לבדיקות קופסא שחורה, שמטרתן לאמת את עמידת הרכיב בדרישותיו, בדיקות קופסא לבנה נועדו לבדוק את תקינות החלקים הפנימיים של הרכיב. בדיקות אלו מתבססות על קוד המקור ונכתבות בעיקר על ידי מתכנתים, ולא על ידי בודקי תוכנה.

בסילבוס של ISTQB הנושא של בדיקות קופסא לבנה מקבל מקום בפרק 4 העוסק בטכניקות בדיקה. פרק זה כולל בדיקות קופסא שחורה, בדיקות קופסא לבנה וטכניקות בדיקה נוספות כמו בדיקות מבוססות ניסיון עליהן כתבתי בעבר.

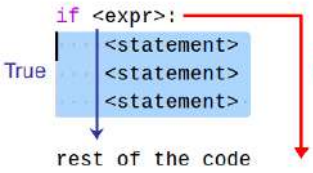
בבדיקות קופסא לבנה נוכל להשתמש בכל רמות הבדיקה, אולם הטכניקות הידועות ביותר שאותן אציג כעת נפוצות בעיקר ברמת בדיקות הרכיבים (Integration Testing). טכניקות אלו מתבססות על שתי צורות בדיקה חשובות:

בדיקת הצהרות קוד וכיסויין (Statement Testing And Coverage)

שיטת בדיקות הקובעת את מקרי הבדיקה כך שכל הצהרת קוד נבדקת לפחות פעם אחת. הצהרת קוד מבחינתנו היא שורת קוד אחת כאשר המטרה היא להגיע ל 100% כיסוי הצהרות קוד. במילים אחרות עלינו לבדוק כל שורת קוד ולוודא שהיא פועלת כראוי.

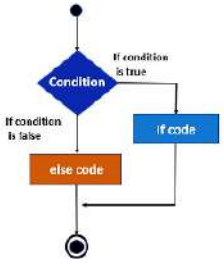
דוגמא לקוד:

בדוגמא זאת עלינו לדאוג שהביטוי בשורה הראשונה יהיה חיובי כך שנוכל להיכנס לכל אחת מ-3 ההצהרות ולוודא שהן עובדות.



בדיקת החלטות וכיסויין (Decision Testing And Coverage)

שיטת בדיקות שבה נריץ את מקרי הבדיקה שגורמים לכל נקודת החלטה בקוד (לדוגמא משפט תנאי IF) לקבל בהרצה אחת ערך חיובי (True) ופעם אחת נוספת ערך שלילי (False).



הערך של בדיקות משפטים ובדיקות ההחלטה:

100% כיסוי משפטים יוכל להבטיח שכל משפטי הקוד נבדקו והורצו לפחות פעם אחת, אבל אין בכך להבטיח שכל לוגיקת ההחלטה נבדקה.

מכאן ניתן למדוד שבדיקות משפטים מספקות כיסוי נמוך יותר מבדיקות החלטות ובמילים אחרות:

100% כיסוי החלטות מבטיח 100% כיסוי משפטים אך לא להיפך!

דוגמא לניתוח קוד:

```
A (Function 1)
IF A=0
than
print“1”
print“2”
```

אם נרצה לממש בדיקת משפטים נבחר **בבדיקה אחת** שכוללת הצבת הערך אפס עבור המשתנה A (A=0) כך נוכל לעבור על כל שורות הקוד ולוודא שקיבלנו את התוצאה המבוקשת של: הדפס "1" והדפס "2".

לעומת זאת,

בבדיקת משפטים נצטרך לממש **שתי בדיקות** אשר כוללות הצבת הערך אפס, כדי לבדוק את התוצאות של הפונקציה כאשר התנאי מתממש.

אבל כדי לסיים את הבדיקה נצטרך גם להציב כל ערך ששונה מאפס, כדי לבדוק את התנהגות הקוד כאשר התנאי לא מתממש.

בדיקות קופסא לבנה כוללות טכניקות נוספות שאינן מופיעות בפרק 4 של ISTQB כמו:

בדיקות לולאות

לולאות בקוד יכולות לגרום לבעיות כמו לולאות אינסופיות אם לא נבדקות כראוי. בדיקות לולאות בודקות את תנאי היציאה מהלולאה ואת התנהגות הלולאה במצבים שונים. חשוב לוודא שהלולאות מתנהגות כנדרש ושלא מתרחשות בעיות בביצועים.

בדיקות מסלולים

בדיקות מסלולים עוסקות במעקב אחרי המסלולים השונים שהקוד עובר. כל מסלול בקוד נבחן על פי התנאים וההגבלות שלו, במטרה לזהות בעיות המתרחשות רק במסלולים ספציפיים. טכניקה זו מאפשרת לוודא שהמערכת יכולה להתמודד עם כל האפשרויות.

בדיקות מחסומים (נקרא גם: בדיקות גבולות קצה)

בדיקות מחסומים נועדו לבדוק את יכולת הקוד להתמודד עם קלטי קצה, כלומר קלטים שאינם סטנדרטיים. בדיקות אלו עוזרות לוודא שהמערכת לא קורסת כאשר היא נתקלת בקלטים בלתי צפויים, נושא חשוב במיוחד במערכות קריטיות (mission or safety critical) או בסביבות שמצריכות אמינות גבוהה (high integrity environment).

בדיקות משולבות (בדיקות אינטגרציה)

בדיקות משולבות בוחנות את הקוד בהקשר של שילובו עם רכיבים אחרים במערכת. טכניקה זו מאפשרת לזהות בעיות הנובעות מאינטראקציה בין רכיבים שונים, מה שיכול לסייע בזיהוי בעיות מורכבות יותר.

בדיקות חיזוי

בדיקות חיזוי מתמקדות בניתוח התוצאות הצפויות על בסיס קוד המקור. המטרה היא לוודא שהפונקציות השונות מייצרות את התוצאות הצפויות במצבים שונים. באמצעות טכניקה זו ניתן לבדוק האם המערכת מתפקדת כראוי בכל התרחישים האפשריים.

בדיקות קופסא לבנה מציעות יתרונות משמעותיים בתהליך פיתוח התוכנה, שכן הן מתמקדות במבנה הפנימי של הקוד ובבדיקת

"בדיקות קופסא לבנה הן סוג של בדיקות תוכנה שמתמקדות במבנה הפנימי של רכיב או מערכת"

קובי יונסי

בעל תואר ראשון בלוגיסטיקה וכלכלה מאונ' בר אילן.
מייסד המרכז המוביל לבדיקות תוכנה שהוא גוף הכשרה הרשמי של חב' Google וספק ההדרכה של חברות הבדיקה הגדולות בישראל.
באקדמיה, ראש החוג לבדיקות תוכנה בטכניון.
מעביר קורסי QA באקדמיה ר"ג, במכללה להנדסה עזריאלי י-ם, במרכז האקדמי פרס רחובות ובמרכז האקדמי רופין. חבר מייעץ בעמותת ITCB® ויו"ר מרכז מצויינות QAבלשכה לטכנולוגיות המידע. מלמד אנשים וארגונים כיצד לחשוב יצירתי ומסייע לאנשים להיכנס לתעשיית ההיי-טק דרך עולם הבדיקות.

הלוגיקה של המערכת. טכניקות אלו מאפשרות גילוי מוקדם של בעיות, מה שמפחית עלויות וזמן תיקון מאוחר יותר. יתרה מכך, הן מספקות כיסוי קוד מעמיק, מה שמבטיח שהרכיבים הפנימיים פועלים בצורה תקינה ועומדים בדרישות המהותיות של המערכת. בזכות הגישה המעמיקה, ניתן גם לייעל את ביצועי הקוד ולוודא שהמערכת מתפקדת בצורה אופטימלית.

יתרונות וחסרונות בבדיקות קופסה לבנה

יתרונות

גישה למבנה הפנימי:

- מאפשרת להבין את הלוגיקה והמבנה הפנימי של הקוד, מה שמוביל לגילוי בעיות שלא תמיד ניתן לראות מבחוץ. מקרי קצה ושגיאות שחבויות בקוד יתגלו באופן מהיר ויעיל.

זיהוי בעיות מוקדם:

- בדיקות קופסה לבנה מאפשרות גילוי בעיות בתהליך הפיתוח, דבר שמפחית עלויות וזמן תיקון בהמשך. כאשר בדיקה נכשלת, יותר קל לאתר את שורות הקוד שאחראיות לבעיה.

כיסוי קוד גבוה:

- ניתן להשיג כיסוי קוד גבוה יותר, מה שמסייע להבטיח שהרבה מהקוד נבחן ונבדק.

יכולת לנתח ביצועים:

- מאפשרת לנתח את ביצועי הקוד בצורה מעמיקה, כולל זיהוי צווארי בקבוק וביצועים לא אופטימליים. (בעיות Performance)

בדיקות מותאמות אישית:

- ניתן להתאים את הבדיקות לצרכים הספציפיים של הפיתוח והדרישות הטכניות.

בדיקות קופסה לבנה, אף שהן מציעות יתרונות רבים, נושאות עימן גם חסרונות משמעותיים. אחת הבעיות המרכזיות היא הצורך בידע טכני מעמיק, מה שעשוי להגביל את יכולת הבודקים שאינם מתכנתים לבצע את הבדיקות בצורה אפקטיבית. בנוסף, בדיקות אלו עשויות לדרוש יותר זמן ומאמץ, במיוחד כאשר מדובר במערכות מורכבות. כמו כן, ההתמקדות במבנה הפנימי של הקוד עלול להוביל להזנחת דרישות המשתמש והציפיות שלו, דבר שיכול לפגוע בחוויית השימוש הכוללת.

חסרונות

דרישות טכניות:

- נדרשת הבנה מעמיקה של הקוד והמבנה הפנימי, מה שעלול להקשות על אנשי QA שאינם מתכנתים.

זמן ומאמץ:

- עלולות לדרוש יותר זמן ומאמץ לפיתוח בדיקות מקיפות, במיוחד עבור מערכות גדולות ומורכבות.

מוגבלות בהבנת דרישות משתמש:

- מתמקדות בקוד ולא תמיד בודקות את עמידת המערכת בדרישות המשתמש או בציפיותיו.

סיכון של "ראיית הקוד":

- הבודקים עשויים לפתח "הטיית קוד", שבהם הם מתמקדים באספקטים הטכניים ולא בבעיות חוויית המשתמש.

להלן כמה דוגמאות מהתעשייה:

גוגל:

משתמשת בבדיקות קופסה לבנה במגוון המוצרים שלה, כולל מנוע החיפוש ו-Google Maps. הם מבצעים בדיקות פונקציות כדי לוודא שכל אלגוריתם מקבל את הקלטים הנכונים ומחזיר את התוצאות הצפויות. בנוסף, גוגל משתמשת בבדיקות כיסוי קוד כדי לוודא שכמה שיותר מהקוד נבדק ומעובד, מה שמפחית סיכוני תקלות במוצרי החברה.

טסלה:

משתמשת בבדיקות קופסה לבנה במערכות התוכנה של רכבי החשמל שלה. הם מבצעים בדיקות כיסוי קוד על מנת להבטיח שכל רכיב במערכת הבקרה של הנהיגה האוטונומית נבדק ומאומת בתנאי סביבה שונים. (עומס תנועה, תנאי מזג אויר קשים, מסלולי כביש ירודים וכו').

Adobe:

מבצעת בדיקות קופסה לבנה בתוכנות כמו Photoshop ו-Illustrator. הם משתמשים בבדיקות תנאים ובדיקות מסלולים בכדי לוודא שכל תכונה במוצר פועלת כראוי, והמשתמשים חווים ביצועים חלקים.

דוגמא למקרה שלא בוצעו בו בדיקות קופסא לבנה:

חברה: טויוטה

תקלת הקוד: מערכת בקרת הדלק ברכב

פרטי התקלה:

בשנת 2019, טויוטה הודיעה על תקלה במערכת בקרת הדלק בכמה דגמים שלה. התקלה נגרמה כתוצאה משגיאה בקוד שבדק את מצב החיישנים במערכת. כאשר החיישנים לא פעלו כראוי, המערכת לא הצליחה לזהות את כמות הדלק הנכונה שיש להזריק למנוע.

תוצאה:

במהלך נהיגה, רכבים מסוימים חוו הפסקות פתאומיות של כוח המנוע, דבר שגרם לסכנת תאונה. כתגובה לתקלות, החברה נאלצה לזמן יותר מ-1.3 מיליון רכבים ברחבי העולם כדי לתקן את הבעיה.

השלכות:

1. פגיעה בבטיחות: התקלה יצרה סכנה ממשית לנהגים ולהולכי רגל, דבר שהשפיע על בטיחות הרכבים.
2. נזק תדמיתי: האירוע פגע במוניטין של טויוטה, שנחשבת לחברה שמקפידה על איכות ובטיחות המוצרים שלה.
3. הוצאות תיקון: החברה נדרשה להשקיע כספים רבים בתהליך התיקון, כולל עלויות של זימון רכבים למוסכים.

לקחים:

המקרה הזה מדגיש את החשיבות של בדיקות קוד מקיפות ומדויקות, במיוחד במערכות קריטיות לבטיחות. בדיקות קופסה לבנה שיכולות לזהות בעיות לוגיות בתהליך הפיתוח עשויות היו למנוע את התקלה ולחסוך נזקים משמעותיים.

לסיכום

בדיקות קופסה לבנה הן כלי חיוני בתהליך פיתוח התוכנה, ומספקות תובנות מעמיקות על איכות הקוד ותפקוד המערכת. באמצעות הטכניקות השונות, כגון בדיקות פונקציות, מסלולים, תנאים ולולאות, ניתן לזהות בעיות בשלב מוקדם ולשפר את ביצועי התוכנה. השגת כיסוי קוד גבוה ובדיקת אינטראקציות בין רכיבים מאפשרת להבטיח שהמערכת פועלת בצורה תקינה ומספקת חוויית משתמש איכותית. השילוב של טכניקות בדיקה קופסא לבנה בתהליך הפיתוח הוא חיוני להצלחת המערכת ומסייע להימנע מתקלות בעת השקת המוצר.

שיפור מצוינות ב-QA באמצעות זיהוי בעיות מוקדם, ניטור והתראות פרואקטיביות | דניס קוזירה

חשיבות הניטור וההתראות ב-QA

בעולם הפיתוח המהיר של תוכנה, בדיקת האיכות היא קריטית. כמומחי בדיקות איכות (QA), המטרה שלנו היא לספק חוויית משתמש חלקה תוך הפחתת שיבושים. כדי להשיג זאת, עלינו להשתמש בכוחם של זיהוי בעיות מוקדם, ניטור משתמשים בזמן אמת והתראות פרואקטיביות. כלים אלה לא רק מייעלים את תהליכי העבודה שלנו אלא גם חוסכים זמן וכסף לחברות על ידי מניעת בעיות לפני שהן מתפתחות.

חלק 1 - ההשפעה הכללית של זיהוי בעיות מוקדם

ניצול זיהוי בעיות מוקדם לחיסכון בעלויות

ב-QA, ניטור פרואקטיבי הוא הרבה יותר מסתם פרקטיקה טובה - הוא אסטרטגיה קריטית לחיסכון בעלויות. זיהוי מוקדם של בעיות חיוני להגנה על חוויית המשתמש ועל הארגון עצמו. מניסיוני, שילוב של פרקטיקות ניטור יעילות תוכניות בדיקות תוכנה (STPs) ובתיאורי בדיקות תוכנה (STDs) הוכח כיעיל.

יתרונות מרכזיים:

- מניעת רגרסיות: זיהוי רגרסיות מוקדם מונע מהן להפוך לבעיות גדולות ומורכבות יותר. הדבר שומר על שלמות המוצר ומבטיח את שביעות רצון המשתמשים.
- ניטור תהליכים (**FLOW**)**:** ניטור תהליכים קיימים, במיוחד אלו שעליהן נבנות תכונות חדשות, הוא חיוני. זיהוי מוקדם של כשלי תהליכים/פרוססים חוסך זמן יקר למפתחים ומבטיח שהבעיות יפתרו לפני שהן משבשות את המחזור הפיתוח.

> "ניטור פרואקטיבי הוא הרבה יותר מסתם פרקטיקה טובה - הוא אסטרטגיה קריטית לחיסכון בעלויות"

השפעה בעולם האמיתי:

ניטור יעיל לא רק מפחית סיכונים—הוא מתורגם לחיסכון כלכלי מוחשי:

- שביעות רצון לקוחות: זיהוי ותקשורת בעיות באופן פרואקטיבי ממזערים שיבושים ומובילים ללקוחות מרוצים יותר.
- יעילות בזמן: פתרון בעיות מהיר, המושג על ידי ניטור חזק, חוסך זמן למפתחים שמתורגם לחיסכון כלכלי.

אימוץ פרקטיקות הטובות ביותר בזיהוי בעיות מוקדם יכול לשפר משמעותית את תהליכי ה-QA ולהשפיע לחיוב על הבריאות הפיננסית של הארגון.

פתרון בעיות פרואקטיבי:

פתרון בעיות פרואקטיבי הוא המקום שבו RUM באמת מצטיין. על ידי ניתוח מגמות התנהגות המשתמשים וזיהוי דפוסים, אנו יכולים לצפות בעיות פוטנציאליות לפני שהן משפיעות על המשתמשים. גישה פרואקטיבית זו לא רק משפרת את חוויית המשתמש אלא גם מצמצמת את מספר הבעיות המגיעות לצוות הפיתוח, וחוסכת זמן ומשאבים.

נקודת מבט של QA:

כמומחי QA, RUM מעניק לנו יכולת לראות מעבר לשיטות בדיקה מסורתיות. הוא עוזר לנו לזהות מקרים קיצוניים ובאגים פוטנציאליים שעלולים לחמוק מתחת לרדאר. על ידי מתן תמונה ברורה יותר של האופן שבו משתמשים מתקשרים עם התוכנה שלנו, RUM מאפשר לנו לספק חוויית משתמש חלקה ולתפוס בעיות לפני שהן משפיעות על המשתמשים.

חלק 2: ניצול ניטור משתמשים בזמן אמת (Real User Monitoring) של Datadog לשיפור ה-QA

מהפכת הפתרון בעיות עם ניטור משתמשים בזמן אמת

בהמשך לחשיבות זיהוי בעיות מוקדם, נתמקד בכלי עוצמתי המעצים את יכולתנו לנטר ולפתור בעיות משתמשים: ניטור משתמשים בזמן אמת (RUM) של Datadog. מנקודת מבט של QA, RUM הוא מהפכני, ומספק תובנות עמוקות לגבי האופן שבו משתמשים מתקשרים עם המערכות שלנו.

פתרון בעיות ריאקטיבי:

פתרון בעיות ריאקטיבי הוא חלק הכרחי מ-QA, אך הוא לא חייב להיות גוזל זמן. עם RUM, כל הסשנים של המשתמשים נאספים, ומספקים לנו תמונה כוללת של מסע המשתמש. נתונים אלה הם בעלי ערך רב כאשר אנו חוקרים בעיות, שכן אנו יכולים לזהות שגיאות ישירות בדפדפן ולעקוב אחריהן עד לשורות המדויקות של הקוד השגוי. זה מבטל את הניחושים, ומאפשר למפתחים לטפל בבעיות בצורה יעילה יותר.

דניס קוזירה
בן 34, איש מקצוע בתחום הבטחת איכות (QA) עם תשוקה לשיפור ביצועי תוכנה ולמידה מתמדת. כיום מהנדס אוטומציה בחברת Skai.
ה"אני מאמין" שלי זה שהבטחת איכות אינה רק איתור תקלות, אלא יצירת חוויית משתמש טובה, הבטחת אמינות המוצר ושיפור תהליכי הפיתוח. מטרתי היא לגשר בין פיתוח למסירה על ידי שיתוף פעולה, אוטומציה של תהליכים שגרתיים ומתן משוב אפקטיבי בכל שלב בתהליך הפיתוח.

שיפור מצוינות ב-QA באמצעות זיהוי בעיות מוקדם, ניטור והתראות פרואקטיביות | דניס קוזירה

חלק 3: התראות פרואקטיביות עם Datadog

הגדרת התראות יעילות להבטחת בריאות המערכת

החלק האחרון בארגז הכלים שלנו ב-QA הוא התראות פרואקטיביות. בעוד שזיהוי מוקדם וניטור הם קריטיים, יש חשיבות עליונה בהתקנת מנגנוני התראות נכונים שידווחו על בעיות פוטנציאליות לפני שהן מתפתחות. תכונות ההתראות של Datadog, בשימוש נכון, יכולות להיות כלי מרכזי בשמירה על בריאות המערכת.

שיקולים מרכזיים:

- כתיבת פקודות יעילות: לדעת לכתוב פקודות מדויקות הוא קריטי לניטור חלקים מסוימים במערכת ולזיהוי בעיות מוקדם. שליטה בסינטקס עבור שאילתות - בין אם לשירותים(SERVICES) או לסביבות מסוימות - היא חיונית לניטור מדויק.
- הבנת רמות שגיאות: זיהוי הרמות הנכונות של שגיאות (ERROR, WARN, INFO) בתוך שאילתות מספק מבט מפורט על בריאות המערכת. זה עוזר בבידוד וטיפול מהיר בבעיות.
- מקור ההתראות: התראות יכולות להגיע ממקורות שונים כמו לוגים, מטריקות, ועוד. על ידי ניטור כל המקורות הללו, נוכל לתפוס ולתקן בעיות לפני שהן מתפתחות לבעיות גדולות יותר.
- מרכאות לעומת כוכביות: הבחירה בין שימוש במרכאות או כוכביות בשאילתות יכולה להשפיע משמעותית על התוצאות. מרכאות משמשות להתאמות מדויקות וספציפיות, בעוד כוכביות מאפשרות חיפושים רחבים וגמישים יותר. הבנה מתי להשתמש בכל אחת יכולה לשפר את יעילות הניטור.

> "אימוץ פרקטיקות הטובות ביותר בזיהוי בעיות מוקדם יכול לשפר משמעותית את תהליכי ה-QA ולהשפיע לחיוב על הבריאות הפיננסית של הארגון"

יישום מעשי:

על ידי שליטה בטכניקות ההתראה הללו, נוכל להישאר צעד אחד לפני בעיות פוטנציאליות ולהבטיח שהמערכות שלנו פועלות בצורה חלקה. התקנת התראות אינה רק קבלת התראות; היא לוודא שאנו מנטרים את הדברים הנכונים בדרך הנכונה.

לסיכום

השילוב של זיהוי בעיות מוקדם, ניטור משתמשים בזמן אמת, והתראות פרואקטיביות יוצר משולש עוצמתי שמשפר את איכות התוכנה ואת יעילות תהליכי ה-QA. על ידי יישום אסטרטגיות אלו, מומחי QA יכולים לא רק למנוע בעיות לפני שהן קורות אלא גם לחסוך זמן ומשאבים יקרי ערך לארגונים שלהם.

בין אם מדובר בתפיסה של רגרסיות מוקדם, בניצול RUM של Datadog לתובנות עמוקות יותר, או בהתקנת התראות מדויקות, פרקטיקות אלו מבטיחות שהתוכנה שלנו עומדת בסטנדרטים הגבוהים ביותר של איכות ומספקת חוויית משתמש חלקה.

כשאנו ממשיכים לאמץ כלים וטכניקות אלו, אנו סוללים את הדרך למצוינות ב-QA ותורמים משמעותית להצלחת הארגונים שלנו.

ראיון עם מנהל בדיקות - אורן עמית AppCard

אורן עמית

אני אורן, בן 48, נמצא בזוגיות מזה 5 שנים, עוסק בשעות הפנאי בעולמות של בישול ושוק ההון. אין לי הסמכות פורמליות למעט בגרות מלאה, לאורך השנים עבדתי קשה על שריר האוטודידקטיות. עוסק בעולמות של QA למעלה מ-20 שנה, הובלת בדיקות תוכנה עבורי היא הרבה מעבר להבטחת איכות – זה ליצור ערך עסקי אמיתי דרך תשומת לב לפרטים הקטנים, אסטרטגיה חכמה ושאיפה בלתי פוסקת לאיכות. הגעתי לחברה הנוכחית בצורה הכי בנאלית שיש, מיציתי את דרכי בתפקידי בחברה הקודמת והחלטתי להמשיך הלאה. ממש במקרה פנתה אלי head hunter ומשם דברים התגלגלו במהרה.

במה עוסקת הקבוצה וכיצד היא בנויה?

הקבוצה הנוכחית מורכבת מ-5 אנשים, כולם אנשים עם ניסיון רב בתחום, אני מאמין בתפיסה שבה כולם מתמקצעים בכל התחומים ויישמתי את הפילוסופיה הזו גם כאן.

מבחינת תחומי אחריות, כולם עובדים בצורה רוחבית על כל המוצרים, החל מעולמות ה-UI ועד עולמות ה-DATA, AI ומערכות BE מורכבות.

איך נראה יום העבודה שלך כמנהל?

כמו כל סטארפ אפ, קשה לנבא את מהלך היום, אנחנו משלבים את שגרת הבדיקות ביחד עם תקלות שמגיעות אלינו מהשטח ועזרה לצוות הפיתוח והמוצר, לרוב אטרף.

אנחנו מאוד משתדלים לא להעמיס את האנשים במיילים מיותרים ובפגישות ארוכות ולהיות מפוקסים על העבודה.

היום מתחיל בבוקר מוקדם במעבר על תקלות (לשמחתי מעטות) שהגיעו במהלך הלילה ומשם לשגרת היום של מעבר על המשימות היומיות של הצוות.

מה הן הדרישות הניהוליות והעסקיות מאנשי הבדיקות וממך כמנהל נכון להיום ובעתיד?

הדרישה הניהולית היא לספק איכות ללא פשרות, הבנה מקצועית רחבה ככל הניתן בעסקי הליבה של החברה.

הצוות עובד בצורה צמודה עם קבוצת ה-Account Managers וצוות ה-Sales ודואג לעדכן אותם בכל שלבי המוצר כולל הדגמות.

מה האתגרים שלכם בתחום הבדיקות וכיצד אתם מתגברים עליהם, האם נראה כי אלו ישתנו בעתיד?

כרגע עומדים בפנינו שני אתגרים, הראשון הכנסה של אוטומציה שהייתה חסרה במשך תקופה ארוכה והשנייה מעבר מתודולוגיה אל עולמות ה-Agile, מה שיצריך שינוי תפיסתי בכלל הארגון.

האתגרים הנוספים קשורים למורכבות המערכת, חיבור של רכיבי תוכנה וחומרה ועבודה עם מספר DB שנמצאים בתהליכי קונסולידציה.

באילו אתגרים נתקלים הבודקים שלכם?

האתגרים די דומים לשאר החברות, עבודה אינטנסיבית שמלווה בהמון context switch, התמודדות יומית עם צוות המוצר וצוותי הפיתוח ואתגרים בעולמות ה-CI/CD.

באילו אתגרים ניהוליים הנך נתקל?

האתגרים שלי קשורים להבנה של סדרי עדיפויות ומאמץ יומיומי לשמור על האיכות תוך מיצוי מקסימלי של כוח האדם ועמידה במשימות.

מהם הדברים שהכי מתסכלים אותך מבחינה מקצועית?

קשה לי עם העובדה שאנשים לא לוקחים ברצינות את מקום העבודה שלהם ולא משקיעים מאמץ עליון בלשים את הצלחת החברה במקום הראשון על חשבון האגו הפרטי שלהם. (אחד בשביל כולם וכולם בשביל אחד)

כיצד אתה מניע (Motivate) את הבודקים ומה עוד היית רוצה לעשות?

אני מאמין בתקשורת אישית, בהמון הקשבה ותמיכה ובחלוקה נכונה של גזרים ומקלות.

הייתי שמח שיהיה יותר תקציב שיאפשר פעילויות מגבשות עבור חברי הצוות ולכלל הארגון.

כיצד הנכם פועלים להעשרת הידע של הבודקים? (הן מבחינה מתודולוגית והן מבחינה טכנית)

נקודת תורפה שלנו, אנחנו לא עושים מספיק בתחום הזה ומבחינתי זה יעד חשוב לשיפור.

אנא שתף אותנו בכמה מההישגים העיקריים של קבוצת הבדיקות

קבוצת הבדיקות היום שולטת ביד רמה מבחינה מקצועית ברזי המוצר ומהווה מקור ידע קריטי בארגון שהושג ברובו על ידי למידה עצמית.

הקבוצה עובדת בצורה מסודרת ושותפה למסע של כל מוצר משלבי האפיון של המוצר ועד מסירתו ללקוח הקצה.

מה התובנות שלך לגבי תחום הבדיקות?

תחום מרתק שבכל כיום ניתן ללמוד בו דברים ולהעשיר את עצמך ואת הסובבים

אני מבין היום את חשיבות כלי ה-AI וכמה בקלות אני כותב כלים המסייעים לכלל הצוות.

מה היית ממליץ לבודקי תוכנה הנמצאים בתחילת הקריירה שלהם?

לעבוד במקומות טובים, כאלו שישקיעו בך ויספקו לך את אבני הדרך הראשונות, מעבר לזה חשוב תמיד לזכור שאיכות היא לא רק QA אלא הדרך שהמוצר עבר וקריטי שאתה בתור בודק תוכנה, צעיר או ותיק תהיה חלק מאותה תפיסה של איכות.

ראיון עם מנהל בדיקות - אורן עמית AppCard

מה הן ההמלצות שלך לבודקים ומנהלים בתחום?

תמשיכו להעשיר את הידע ותדעו לעמוד על שלכם אל מול צוות ההנהלה.

תעבדו עם כלי AI לבצע את המשימות שלכם בצורה יעילה.

ממה היית ממליץ להימנע?

סמים? 😊

המראיין: ניצן גולדנברג

מזה 9 שנים בתחום בדיקות תוכנה. מהנדס בדיקות בחברת AppCard, מוביל את ערוץ הפודקאסט TestIL Podcast, מנהל את קבוצת המייעצים (AB) של עמותת ®ITCB, יו"ר ומנהל תחרות הבדיקות הישראלית ISTC, חבר בקבוצת ה"מרקטינג" של ISTQB, המוביל של קבוצת המיטאפ הגדולה בישראל לבודקי תוכנה TestIL ומרצה בכיר בקורסים וכנסים לבודקי תוכנה.

פספורט קבוצתי

APP CARD Buy. Earn. Redeem.

סוג המוצר הנבדק: המוצר הינו מוצר מעולם ה-Retail. המוצר מורכב ממערכות BE בעולמות של Java ו-Python, כולל מוצרי קצה Web, Mobile apps (iOS & Android) ועבודה עם מסדי נתונים מגוונים ומערכות לינוקס.

גודל קבוצת הבדיקות: 5 בודקים (כולל מנהל בדיקות), כרגע כולם בודקים ידניים.

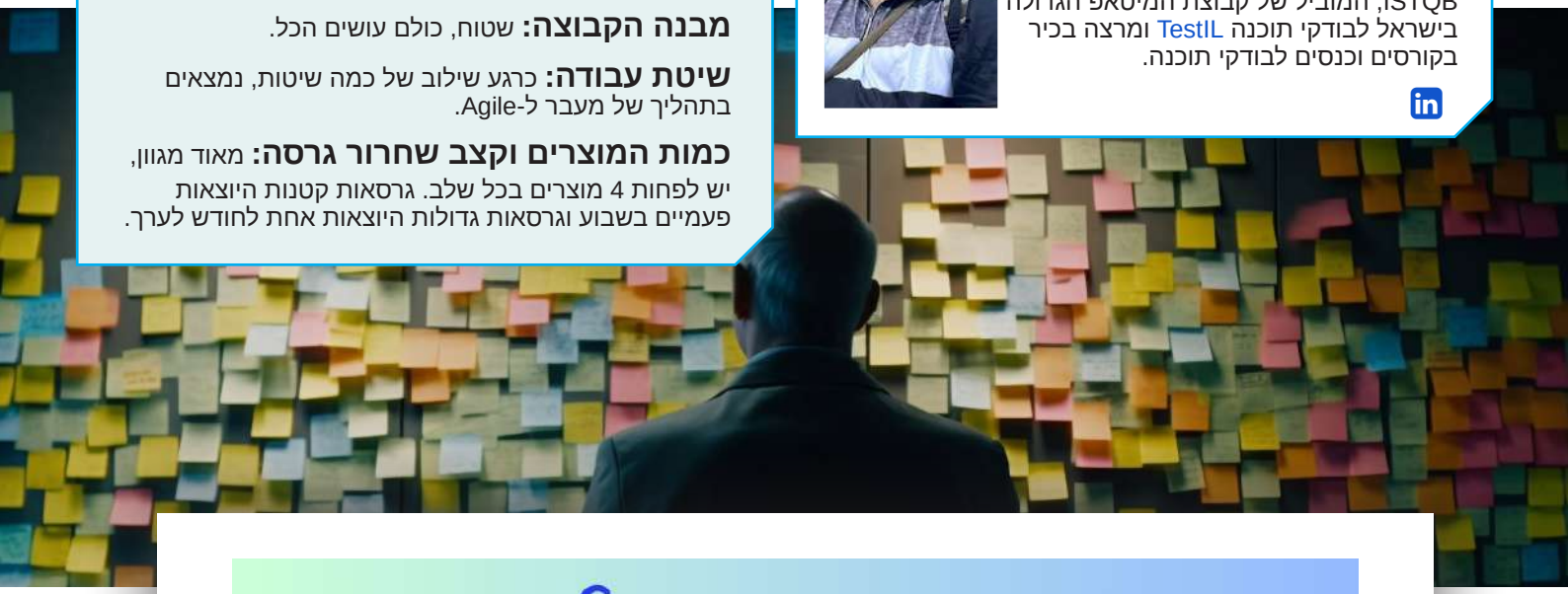
וותק הבודקים: כולם בעלי וותק של שנים רבות – מעל 15 שנות נסיון.

סוגי בדיקות: בדיקות Web & Mobile, בדיקות לוגיות, דאטה וכו'.

מבנה הקבוצה: שטוח, כולם עושים הכל.

שיטת עבודה: כרגע שילוב של כמה שיטות, נמצאים בתהליך של מעבר ל-Agile.

כמות המוצרים וקצב שחרור גרסה: מאוד מגוון, יש לפחות 4 מוצרים בכל שלב. גרסאות קטנות היוצאות פעמיים בשבוע וגרסאות גדולות היוצאות אחת לחודש לערך.



סקירת כלים Testmo | ניצן גולדנברג

במהלך ביקורי בפורום TestIL נתקלתי בפוסט המדבר על כלי לניהול בדיקות בשם Testmo.

בתור חובב כלים לניהול בדיקות ותקלות, הרגשתי צורך לנסות את הכלי וכמובן להביא לכם, קוראים יקרים, את הסקירה האישית שלי עליו.

Testmo הוא כלי לניהול והרצת בדיקות המגיע אלינו מברלין, גרמניה. הכלי קיים בשוק מזה כ-5 שנים ומציע יכולות נרחבות של ניהול בדיקות והרצתן, אך ללא מודול לפתיחה וניהול תקלות. לכלי יש תכונות ויכולות התממשקות מרובות, אותן אסקור במאמר זה.

התכונות המרכזיות של Testmo

- מערכת ניהול בדיקות
- ניהול ותיעוד בדיקות חקרניות
- התממשקות עם כלים צד שלישי
- ממשק מנהל מערכת (Admin)

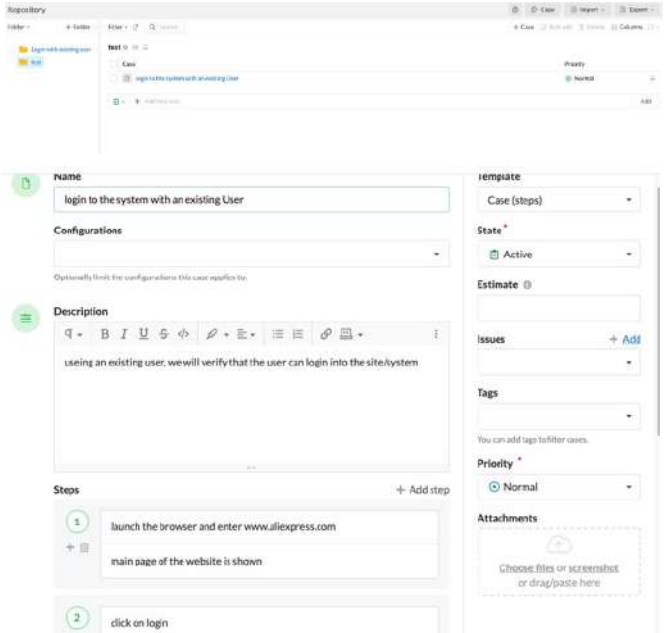
מערכת ניהול בדיקות

התכונה המרכזית בכל מערכת לניהול בדיקות היא בנק מקרי הבדיקה (Repository). ב-Testmo, כמו בכלים אחרים לניהול בדיקות, ניתן לפתוח תיקיות ולהכניס לתוכן את מקרי הבדיקה. עם זאת, בניגוד לכלים דומים, לא ניתן לייצר מקרי בדיקה ללא יצירת ספריות מראש. במקרה של יצירת מקרי בדיקה לפני יצירת הספרייה, הם משויכים אוטומטית לספרייה ללא שם, ניהול זה אינו ידידותי למשתמש.

בנוסף, אם רוצים לצאת מחלון יצירת מקרה בדיקה או ספרייה ללא שמירת הנתונים, באמצעות לחיצה על אחת מהאפשרויות בסרגל הצדדי, לא ניתן לעשות זאת. יש ללחוץ על "ביטול" או "אישור".

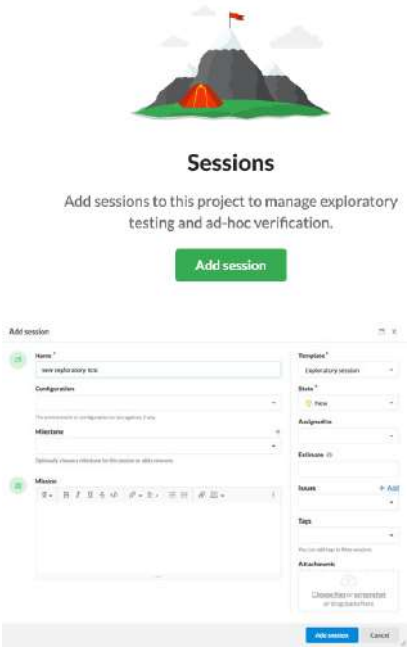
כמו בכלים אחרים, יש אפשרות לבחור איזה סוג מקרה בדיקה מעוניינים לייצר (טקסט בלבד או פירוט צעדי הבדיקה). האפשרות היחידה להוספת צעדים היא באמצעות הוספת צעד חדש, מה שטוב בפני עצמו. אך מה קורה כאשר יש מקרי בדיקה שבהם הצעדים הראשוניים חוזרים על עצמם? הייתי מצפה שתהיה אפשרות להשתמש בצעדים ממקרי בדיקה אחרים או להשתמש במקרה בדיקה הכולל צעדים כצעדים ראשוניים (Call test). בנוסף, חסר שדה של "ערכים לשימוש" בצעדים. הייתי מצפה שיהיו שלוש שדות לכל צעד: פעולה, ערך לשימוש ותוצאה צפויה.

ישנן כל אפשרויות עורך הטקסט המצופות, יכולת הוספת קובץ למקרה הבדיקה, סטטוס מקרה הבדיקה וקישור תקלה (דורש חיבור לכלי לניהול תקלות). קיימת אפשרות לייבא או לייצא את מקרי הבדיקה או לשכפל אותם.



ניהול ותיעוד בדיקות חקרניות

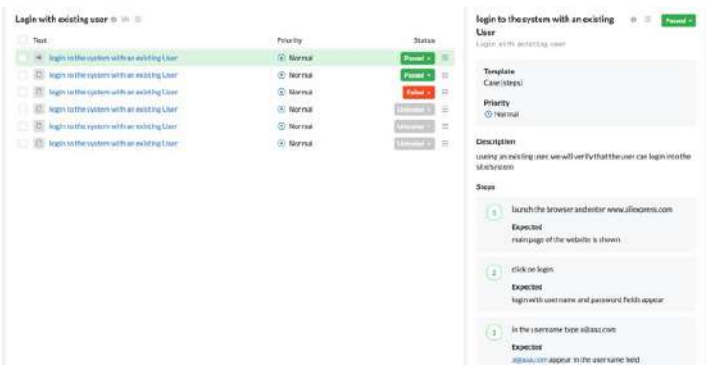
לכלי יש יכולת לנהל מקרי בדיקה שנוצרו באמצעות בדיקות חקרניות (Exploratory Testing). אופן העבודה הוא לפתוח "Session" חדש ולתעד את כל מה שאנו עושים באותו הרגע במערכת. עם זאת, גיליתי שהתכונה הזו לא עובדת כפי שהייתי מצפה.



אין אפשרות להקליט את הסשן, לעשות צילומי מסך או לסמן על גבי צילומי המסך הערות.

הרצת הבדיקות

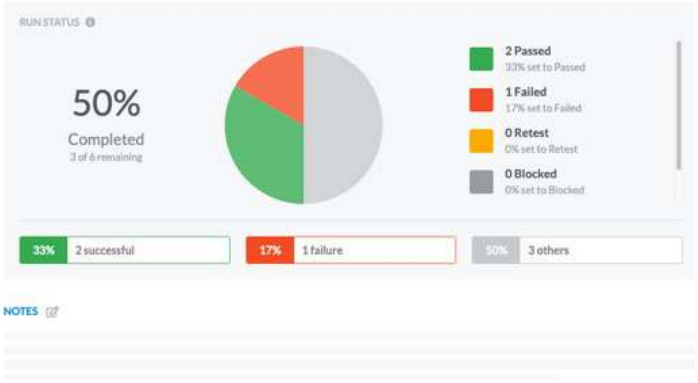
מודול הרצת הבדיקות בכלי הוא בעל נראות נעימה ונקייה. יש אפשרות להוסיף מקרי בדיקה מתוך רשימה קיימת או לייצר חדשים. יש לשונית של סטטוס הרצת הבדיקות, המעניקה מידע לגבי מצב ההרצה. לעומת זאת, המודול לא מאפשר לסמן את הסטטוס של כל צעד בנפרד (האם עבר או נכשל), אלא רק את הסטטוס של מקרה הבדיקה כולו. אין כפתור "הרצה", מה שגרם לי להתעכב בהבנה כיצד "מריצים" את הצעדים של מקרי הבדיקה.



ניצן גולדנברג

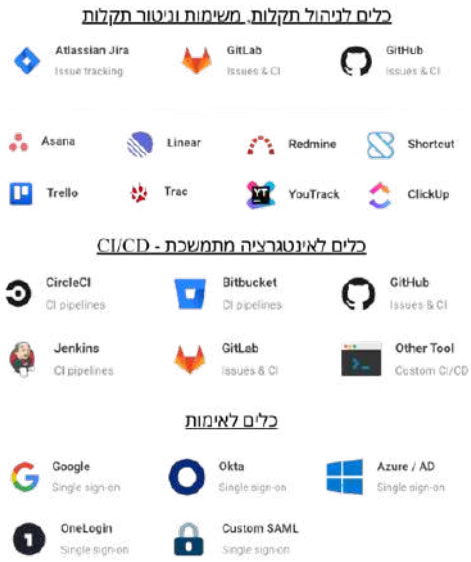
מזה 9 שנים בתחום בדיקות תוכנה. מהנדס בדיקות בחברת AppCard, מוביל את ערוץ הפודקאסט TestIL Podcast, מנהל את קבוצת המייעצים (AB) של עמותת ISTQB®, יו"ר ומנהל תחרות הבדיקות הישראלית ISTC, חבר בקבוצת ה"מרקטינג" של ISTQB, המוביל של קבוצת המיטאפ הגדולה בישראל לבודקי תוכנה TestIL ומרצה בכיר בקורסים וכנסים לבודקי תוכנה.

סקירת כלים Testmo | ניצן גולדנברג



התממשקות עם כלים צד שלישי

לכלי יש רשימה מכובדת של יכולות התממשקות עם כלים צד שלישי.



ממשק מנהל מערכת

לכלי יש מנהל מערכת עשיר, המאפשר למנהל לאפיין את הפרויקט באמצעות ניהול משתמשים בצורה קלה ונוחה, הוספת שדות חדשים לכל המסכים, בניית תהליכי שימוש, הוספה ושינוי סטטוסים של מקרי בדיקה והרצות, ניהול אבני דרך (Milestones) וקינפוג התממשקות עם כלים צד שלישי.

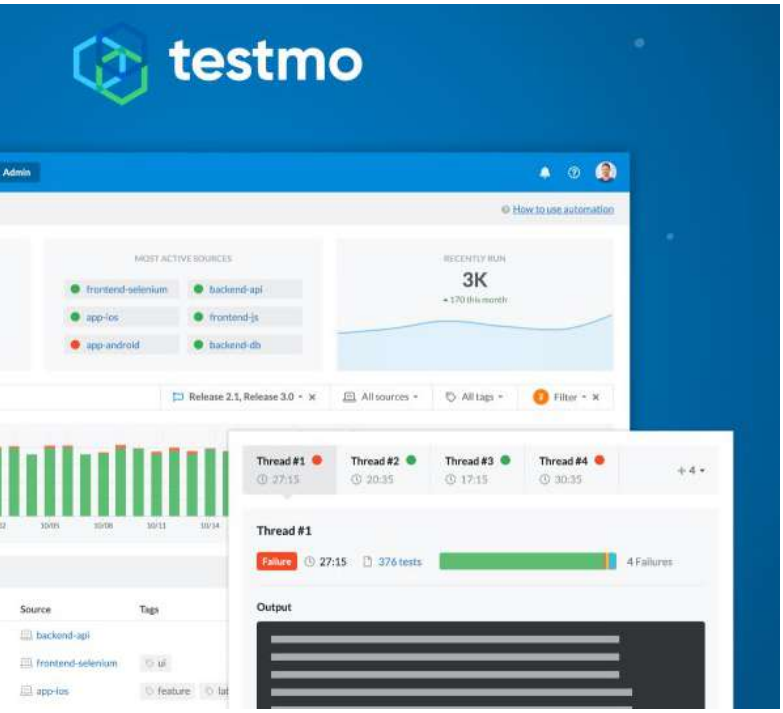
יתרונות

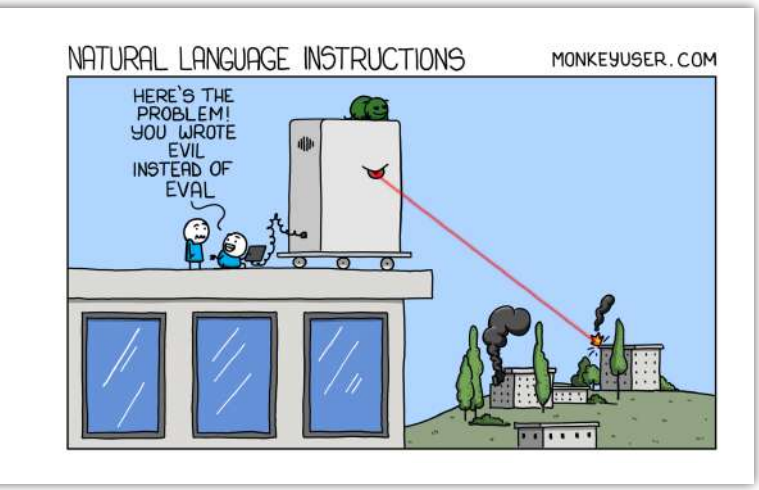
- עיצוב ממשק משתמש פשוט ונעים לשימוש
- קל להבנה ולשימוש
- מגוון רחב של אפשרויות התממשקות
- ממשק מנהל מערכת עשיר
- 20 ימי שימוש חינם
- התממשקות ל-CI/CD ויצירת מנות הרצה לבדיקות אוטומטיות ישירות לכלי

חסרונות

- הכלי אינו מגיע עם מודול ניהול תקלות
- חוויית המשתמש של הכלי לוקה בחסר באזורים רבים
- באתר החברה מצוין שיש דוחות, אך לא מצאתי אפשרות לייצר דוחות
- אכזבה במודול בדיקות חקרניות
- מודול האוטומציה אינו מאפשר יצירת מקרי בדיקה וכתיבת קוד

מענה לצרכים של החברה 5/10
נותן למשתמש חווית שימוש 2/10
תמיכה וקהילה 2/10
סה"כ 3.3





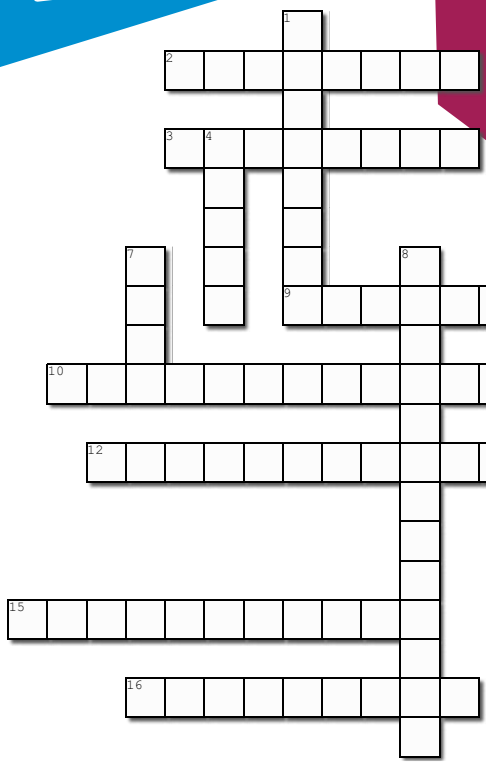
בוקים FUN בכיף

ISTQB - Expert Test Management

התבלבלו לי המילים

1. cecpcaneat _____
2. niesgtt _____
3. geail _____
4. tilaibvaiayl _____
5. tfaitcnroeici _____
6. revnngceeco _____
7. itrmec _____
8. fieseseftvecn _____
9. oacniflunt _____
10. eihutrics _____
11. ailtayamiinnibt _____
12. armetsnumee _____
13. ocnaprfmeer _____
14. rtyioirp _____
15. yiqault _____
16. isrgseenro _____
17. yllebtraiii _____
18. etirmerquen _____
19. ecuisryt _____
20. rpniegrot _____

מיצאו את המושגים

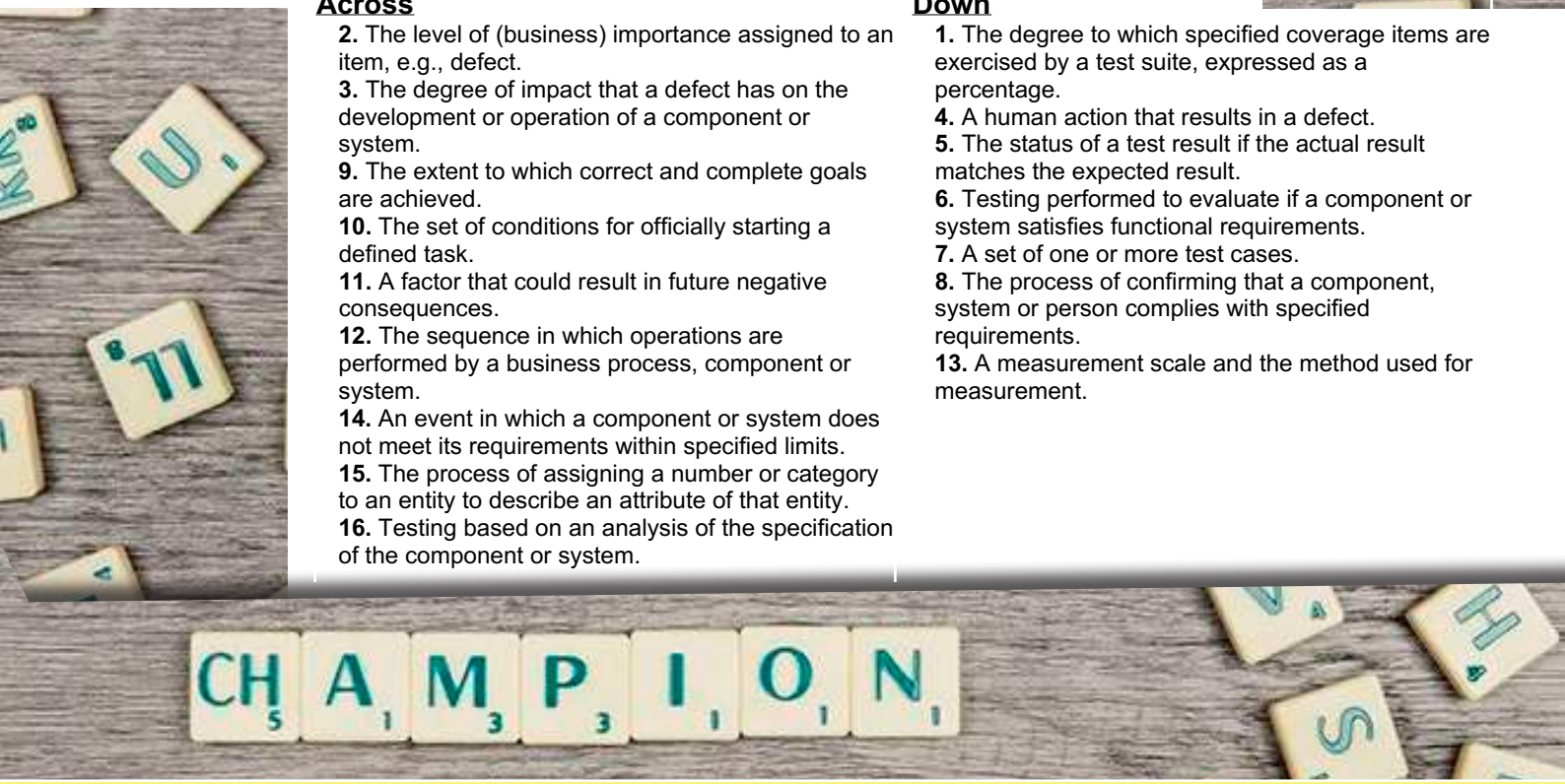


Across

2. The level of (business) importance assigned to an item, e.g., defect.
3. The degree of impact that a defect has on the development or operation of a component or system.
9. The extent to which correct and complete goals are achieved.
10. The set of conditions for officially starting a defined task.
11. A factor that could result in future negative consequences.
12. The sequence in which operations are performed by a business process, component or system.
14. An event in which a component or system does not meet its requirements within specified limits.
15. The process of assigning a number or category to an entity to describe an attribute of that entity.
16. Testing based on an analysis of the specification of the component or system.

Down

1. The degree to which specified coverage items are exercised by a test suite, expressed as a percentage.
4. A human action that results in a defect.
5. The status of a test result if the actual result matches the expected result.
6. Testing performed to evaluate if a component or system satisfies functional requirements.
7. A set of one or more test cases.
8. The process of confirming that a component, system or person complies with specified requirements.
13. A measurement scale and the method used for measurement.



נגיסה מ-TestIL
מפגשים לבודקי תוכנה

עדכונים מערוץ הפודקסט הרישמי של TestIL Podcast מבית ITCB®

את הערוץ מנהלים נתנאל הרוש וקובי יונסי חברים
בקבוצת המייעצים (AB) של ITCB®

מוזמנים להאזין לכל הפרקים שיצאו ברבעון האחרון להנאתכם

פרק #33 – עתיד בדיקות תוכנה – חזון של בינה מלאכותית עם אסף האוזר

בפרק זה של פודקאסט "TESTIL" מבית ITCB, המנחה נתנאל הרוש מארח את אסף האוזר, מנכ"ל ומייסד חברת automatico לשיחה מרתקת על עתיד בדיקות התוכנה. אסף משתף מניסיונו העשיר של מעל 25שנה בתחום, ומדבר על האתגרים המרכזיים שבודקי תוכנה מתמודדים עמם כיום, כמו הצורך בשחרורים מהירים ושילוב בדיקות אוטומטיות בצורה יעילה.

אסף מסביר כיצד הבינה המלאכותית צפויה לחולל מהפכה בתחום הבדיקות, ומספק תובנות מעמיקות על הדרך שבהTestOps משנה את האופן שבו בדיקות משתלבות במחזור חיי הפיתוח. בפרק נידונה גם גישת ה-"Shift-Left" וה-"Shift-Right" של בדיקות תוכנה, ושילובן יחד על מנת להבטיח איכות ואמינות גבוהה לאורך כל מחזור חיי התוכנה.

לבסוף, אסף מסביר כיצד הוא רואה את עתיד התפקידים של מהנדסי האוטומציה והבודקים הידניים בעידן החדש של AI ו-TestOps ונותן עצות חשובות לבודקי תוכנה ולמנהלי הבדיקות של ימינו.

הפרק מציע למאזינים הצצה לעתיד בדיקות התוכנה והאופן שבו בינה מלאכותית ו-TestOps עשויים לשנות את התחום בשנים הקרובות.

פרק #34 – מהם היתרונות בהעסקת בודקים על הרצף?

בראיון, מספרים אברהם ואוריה על הפודקאסט עוסק ביתרונות של העסקת בודקים על הרצף האוטיסטי. הרצל מישל מדבר על חשיבות הגיוון בעובדים בתחום בדיקות התוכנה, ומדגיש את הכישורים והיכולות הייחודיות שמביאים בודקים על הרצף לתחום, כמו דיוק, ריכוז בפרטים וחשיבה לוגית. הוא מתאר את היתרונות שמעסיקים יכולים להפיק משילובם בצוותי בדיקה, לצד האתגרים השונים שיכולים לעלות בתהליך ההעסקה וההכשרה שלהם.

פרק #35 – שורטקאסט עם קובי יונסי – "טכניקת בדיקה "קופסא לבנה"

בפרק הפעם נשוחח על טכניקת הבדיקה "קופסא לבנה" אשר נמצאת בפרק מס 4 בסילבוס שלISTQB .
הפרק הוא חלק מסדרת שורטקאסטים ומתמקד בטכניקות בדיקה חשובות למפתחי תוכנה ובודקים, כאשר טכניקת הקופסא הלבנה מאפשרת הבנה עמוקה של מבנה הקוד הנבדק והיכולת לבדוק את כל המנגנונים הפנימיים שלו. יונסי מסביר את עקרונות הטכניקה, יתרונותיה ויישומים מעשיים בעבודה היומיומית של אנשי בדיקות תוכנה. הפרק מתאים גם לאלו המעוניינים להעמיק ביסודות הבדיקות וגם לאנשי מקצוע מנוסים.

פרק #36 – שילוב "פריימוורק" אוטומציה לבינה מלאכותית עם ניר גלנר

הפרק עוסק בשילוב של "פריימוורק" אוטומציה עם בינה מלאכותית בתהליכי בדיקות תוכנה. המנחים דנים על היתרונות שבשימוש בבינה מלאכותית כדי לשפר את איכות ומהירות הבדיקות, עם דגש על יישומים מעשיים שמפשטים את העבודה ומאפשרים אוטומציה חכמה ויעילה יותר. הפרק מציג כלים וטכנולוגיות חדשות, לצד אתגרים שעולים משילוב אוטומציה מבוססת AI בפרויקטי תוכנה, כולל טיפים לשילוב מוצלח בפרויקטים קיימים.

פרק #37 – שימוש במודלי שפה מקומיים ולא מקומיים כדי להאיץ את פיתוח האוטומציה לדפדפנים (Scrapping ל-dom) עם סעיד ג'אבר

הפרק עוסק בשימוש במודלי שפה מקומיים ולא מקומיים לצורך האצת פיתוח אוטומציה לדפדפנים .סעיד ג'אבר דן באתגרים והיתרונות בשילוב מודלים מסוג זה בתחום האוטומציה, ומתאר כיצד מודלים מקומיים יכולים לשפר את מהירות ועומק הבדיקות עבור יישומים שונים. בנוסף, הוא משתף בכלים שמקלים על הטמעת המודלים, ומדגיש את חשיבות ההתאמה של המודלים לצרכים ייחודיים של כל פרויקט.

אם גם אתם מעוניינים להשתתף בפודקסטים, אנא צרו עימנו קשר במייל: testilpodcast@gmail.com,
קישור לערוץ הפודקסט שלנו.

עורכי הגיליון

תמרה מוסונובה

בודקת תוכנה ואינטגרציה בחברת Varonis. בעלת תואר בתקשורת וקולנוע והסמכות פיתוח אפליקציות Webשל מיקרוסופט, כך משלבת חשיבה יצירתית ואנליסטית בעבודה. בונה אתרים ועורכת סרטים כתחביב. שחקנית כדורשת בליגה ארצית, תופסת כדורים ובאגים מקצועית.

אפרת וינברג

עוסקת בבדיקות תוכנה קרוב ל-20 שנה. עבדה במספר ארגונים בתפקידי בדיקות וניהול בדיקות. בשנים האחרונות עוסקת בפיתוח והוראת קורסים בבדיקות תוכנה ונושאים נוספים

אסתר צבר

מהנדסת (M.Sc.) בעלת 23 שנות ניסיון בפיתוח ובדיקות תוכנה, מתוכן 11 שנים בניהול QA בחברות ECI ו BMC - ובנוסף חברה ב-Advisory Board של ITCB® הארגון הישראלי להסמכת בודקי תוכנה. בתשע השנים האחרונות – יזמית ומנהלת של AQA המכשירה ומשלבת אנשים עם תסמונת אספרגר בעבודה בהייטק כבודקי תוכנה.

משה מאמיה

בעל 17 שנות ניסיון כמהנדס, מתוכן מעל עשר שנות ניסיון ניהולי, מתמחה בפיתוח אוטומציה ובבדיקות ביצועים. עובד מעל 5 שנים ב-HP כמנהל קבוצות QA ו-DevOps. חבר מייעץ למועצת מנהלים של ISTQB® ומרצה בפקולטה להנדסת תוכנה במכללת SCE.

עמית ורטהיימר

בודק תוכנה ב-Deep Instinct.

רחל ברוך

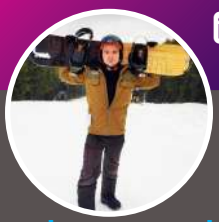
עובדת כבודקת תוכנה בכלל ביטוח. לאחר הפסקה של עשור מעולם התוכנה חזרה למקצוע הכי אהוב אליה. כשעובדים בעבודה שנהנים בה הזמן טס.

דור רוזנברג

לפני כמה שנים החלטתי לעשות שינוי במקצוע שלי. לאחר מספר תחומי לימוד הכי הרבה התלהבתי מלימודי בדיקות תוכנה. כרגע עובד בשרות לקוחות בישראכרט.

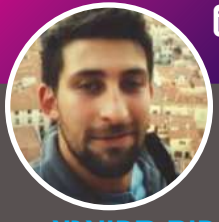
שירה נוסבוים

הייטקיסטית ואמא במשרה מלאה, בדקות הבודדות שנשארות ביום בלוגרית אפייה.
בעלת תואר ראשון במדעי המחשב ובכימיה, מעל 10 שנות ניסיון כמפתחת תשתיות אוטומציה וכלים אוטומטיים בחברות גדולות, בסטארטאפים שונים בתעשייה במגוון תחומים. מובילת תחום, מרצה ומפתחת קורסי אוטומציה. בשבילה החיים זה לא מספיק.



אלכסנדר זבולוקו

מפתח תשתיות אוטומציה בחברת SeatGeek. מתמחה ב-DevOps וטכנולוגיית ענן לעומק. מנצל את הידע והמיומנות לפיתרון בעיות ולהביא לחדירה מירבית של טכנולוגיות חדשות בסביבת התשתיות.
מחבר את המקצועיות עם התשוקה לטייל ברחבי העולם, מתרגש לחוות חוויות ותרבויות חדשות בזמן עבודה על מייזמים פרטיים.

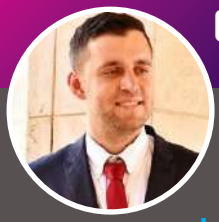


רוביק סביאנץ

בודק תוכנה, נמצא בתחום מעל 4 שנים את דרכו התחיל בחברת CARAMBOLA נכון להיום מחזיק את מערך הבדיקות בחברת OOLO בזמן הפנוי - ספורט, טיולים ומחשבים

שי ביטון

על 12 שנות ניסיון בפיתוח אוטומציות ובדיקות.
עובד כיום ב- Qwilt.



אלכס קומנוב

בעל מספר שנים בתחום הבדיקות האוטומטיות. עובד כמפתח בדיקות ותשתיות אוטומציה בכיר בחברת SeatGeek נשוי+1 חובב נגינה על גיטרה וטיולים.

טל פאר

בעל ניסיון של יותר מ-25 שנים כבודק ומנהל בדיקות במגוון חברות וטכנולוגיות. כיום טל הוא יועץ בכיר ב-Grove Software Testing, חברת הדרכה מובילה בבריטניה וספקית של חומרי הדרכה.
טל פעיל ב-ISTQB® והיה חבר בצוות המנהל של הארגון במשך 6 שנים. טל חבר בצוות המנהל של ITCB®.

מגזין עולם הבדיקות