רבעון ראשון 2021 גיליון מס' 24

מגזין עולם הבדיקות

www.testingworld.co.il

דבר העורך
ניצן גולדנברג

ניצן גולדנברג

מזה 6 שנים בתחום בדיקות התוכנה, תפקיד נוכחי מהנדס בדיקות בכיר בחברת SeatGeek, המוביל הראשי של קבוצת המיטאפ TestIL, מרצה בקורסים לבודקי תוכנה וחבר בצוות המייעץ AB של ISTQB®.

קוראים יקרים,

"You can't depend on your eyes when your imagination is out of focus"
Mark Twain

מצאתי לנכון להביא לכם את המשפט הידוע של מארק טוויין שאומר "אין ביכולתך לסמוך על עיניך כאשר הדמיון שלך לא בפוקוס". ה"אני מאמין" שלי הוא שאנו הבודקים חייבים שיהיה לנו דמיון פורה ואפילו "משוגע" כי עלינו מונחת האחריות לבדוק את המערכות שאנו בודקים ברמה גבוהה ולמצוא בעיות שאחרים לא חשבו שיכולות להיות וכל זה, בשביל שהלקוח או המשתמשים לא יתקלו בתקלות כאשר הם משתמשים במערכת, גם אם זה אומר שהתסריטים שלנו יהיו לא הגיוניים.

יצא לי להתראיין בכמה חברות וגם כיום בתור מראיין אני מבקש לתת לי תסריטי בדיקה ייחודיים שלא חשבו עליהם עוד לשואב אבק רובוטי. בשאלה הזו מטרת המראיין היא לא לבדוק האם אתם משתמשים בשואב אבק באופן יום יומי או את היכולת שלכם לחשוב על תסריטי בדיקה שיקריסו את שואב האבק הכי חדשני בשוק..., מטרת המראיין היא לבחון איך אתם "חושבים", איזה ראש יצירתי יש לכם.

אז לכל הקוראים היקרים אני אומר, אל תפחדו לדמיין, אל תפחדו לחשוב מחוץ לקופסא, אל תפחדו "להשתגע" עם הדמיון שלכם כי בזכות הדמיון שלנו, השמיים הם כבר לא הגבול.

מונח לפניכם גיליון מספר 24 של מגזין "עולם הבדיקות".

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

חשיבות ה-Traceability מאת ניקיטה יריומין אשר יספר לנו על הערך המוסף בעקיבות.

רפי ברי מביא לנו מהניסיון שלו דוגמאות לכמה נקודות מרכזיות / מדדים לבחינה במאמר שלו "בקרה על תוצרי תהליכי העבודה של אנשי הבדיקות".

ליזה ויסמן תענה לנו על השאלה שרבים שואלים במאמר שלה על "הדרך לבחירת כלי לניהול בדיקות".

יאיר נסימוב כתב לנו את "המדריך לאוטמציוניסט המתלבט: ראש בראש "Java, C# & Python" אשר יעשה לנו סדר בשלושת שפות התכנות המובילות כיום בעולם פיתוח אוטומציה.

בצער רב אני מעדכן בגיליון זה אלון פרידמן כותב לנו בפעם האחרונה את הטור שלו "עולם הנגישות" אך במקום זאת יש לנו 2 טורים חדשים בשבילכם:

נטליה רחמני מביאה לנו בטור החדש שלה "מתודולוגיה ותהליכי בדיקות" את הידע והניסיון שלה בכל הקשור למתודולוגיה בעולם הבדיקות.

איסי חזן בטור החדש שלו "הקופסא - תבניות חשיבה שימושיות לבדיקות" יביא לנו תבניות חשיבה שימושיות מהניסיון הרב שלו.

כמו כן תוכלו להנות מהטורים הקבועים שלנו: ראיון עם מנהל/ת בדיקות, האנציקלופדיה לבדיקות, מחפש צרות, בחן את עצמך וסקירת כלים.

אם יש לכם רעיונות ובקשות למאמרים או מעוניינים לפרסם מאמר במגזין, מוזמנים ליצור עימנו קשר במייל: Info.testingworld@gmail.com

קריאה מהנה,
ניצן גולדנברג

תודה

- **ימי העיון אשר תוכננו להתקיים לפני משבר הקורונה חוזרים בזום, לפרטים והרשמה יש להירשם: http://bit.ly/TestIL**
- **החלו ההכנות לתחרות הבדיקות הישראלית ISTC 2021 אשר מתקיימת זו השנה החמישית, ניתן להתעדכן בפורום ובאתר התחרות.**

תוכן העניינים

מו"ל
Israeli Testing Certification Board
ITCB®

ניהול המגזין
יאן ברון, *iMDsoft*

ניהול התוכן
קובי הלפרין, *Nokia*

עורך
ניצן גולדנברג, SeatGeek

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

יצירת קשר
אימייל:
info.testingworld@gmail.com

הרשמה
http://bit.ly/TW-Reg
פקס:03-6176605
כתובת: ברוך הירש 14 בני ברק 51202

www.testingworld.co.il

www.itcb.org.il

מתודולוגיה ותהליכי בדיקות | נטליה רחמני

מה מודל הפיתוח הנכון עבור הארגון שלך?

השוק כיום רווי במספר רב של גישות פיתוח, החל מהמפל והספירלי המסורתיים ועד האג'ייל הפופולארי.

בטור הנוכחי אסקור ואבצע השוואה מעמיקה בין מודלי הפיתוח הנפוצים ביותר בשוק כרגע: Agile, V-Model, Waterfall, Spiral, תוך מתן דגש על התאמת המודל לסוג הפרויקט.

מטרת המודלים היא לייצר תבנית ברורה לתהליכי הפיתוח עד שלב שחרור המוצר.
המודלים הללו מגדירים דרך שיטתית לבצע עבודה מסודרת במהלך הפרויקט. בדיקות תוכנה מופיעות בכל אחד ממודלי מחזור החיים הללו, אך המודלים נבדלים ביניהם בגישתם לתכנון, כתיבת ויישום הבדיקות. לכל מודל יש יתרונות וחסרונות מובנים. בחירת המודל הנכון תהיה תלויה בסוג הפרויקט או המוצר. מרבית הפרויקטים עובדים על פי מודל פיתוח תוכנה ספציפי שנבחר על ידי גוף הפיתוח, אך במקרים מסוימים ישנם מספר מודלים שונים הפועלים במקביל.

לדוגמא: שילוב בין מודלי ה-V או ה-Waterfall לבין מודל ה-Agile:
במהלך תהליך הפיתוח פרויקט המשתמש ב- V-Model עודכנו חלק מהדרישות על ידי הלקוח וכתוצאה מכך גרמו לשינוי במימוש הקוד על ידי הפיתוח.

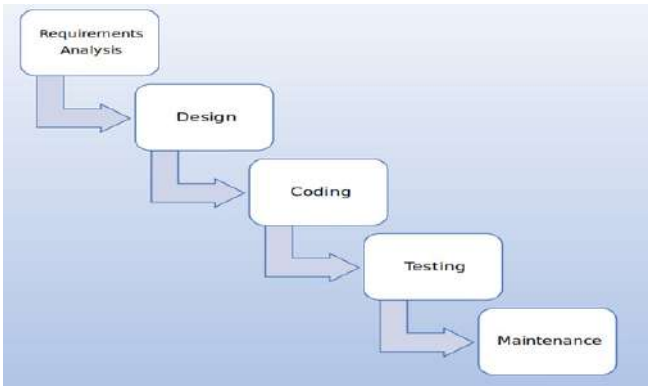
בדוגמה זו, אנו יכולים לראות שמודל ה-V משתלב עם אלמנטים ממודל האג'ייל (שינוי הדרישות במהלך הפיתוח).

להלן מדריך מעמיק ליתרונות ולחסרונות של דגמי מחזור חיים שונים בפיתוח תוכנה: אג'ייל, מודל V, מפל וספירלה.

דגם מפל המים – Waterfall Model

דגם ה-SDLC המקורי היה **דגם המפל**

מודל זה פשוט מאוד להבנה וידוע מאוד בתהליך הפיתוח.



הרעיון העיקרי של מודל זה הוא שרק כאשר תושלם רמת פיתוח אחת, תתחיל רמת פיתוח הבאה.

בסוף כל שלב מתקיימת סקירה כדי לקבוע אם הפרויקט בדרך הנכונה והאם להמשיך את הפרויקט.

יתרונות	חסרונות
שיטת המפל ידועה גם בקרב מפתחי התוכנה, ולכן היא קלה לשימוש	החיסרון המכריע במודל זה הוא שהבדיקה מובנת כפעולה "חד פעמית" או חזרה על כמה מחזורים של בדיקות לקראת שחרור הגרסה ולא מקביל לפיתוח
עובדת היטב בפרויקטים קטנים יותר שבהם הדרישות מובנות היטב	השקעה גבוהה בתיעוד הבדיקות שעשויה להשתנות במהלך הפיתוח
עלות-תועלת: זמן הבדיקות המושקע בתכנון הגדרה מקיפה של יעדי הפיתוח התוכנה יכול להוביל לחסכון רב יותר בשלבים מאוחרים יותר	הבדיקה נתפסת כ"בדיקה סופית", אנלוגיה לבדיקת ייצור לפני מסירת המוצר ללקוח

נטליה רחמני

חיה בעולם בדיקות התוכנה מעל 16 שנים ואני נושמת כל רגע את עולם האיכות.

בעלת ניסיון רב בניהול צוותים בינלאומיים, הקמת מערכי איכות, בניית מתודולוגיות בדיקות במגוון תחומים: ביטחוני, רפואי ופיננסי.

כיום Head of QA בחברת ThetaRay.

productivehut.com

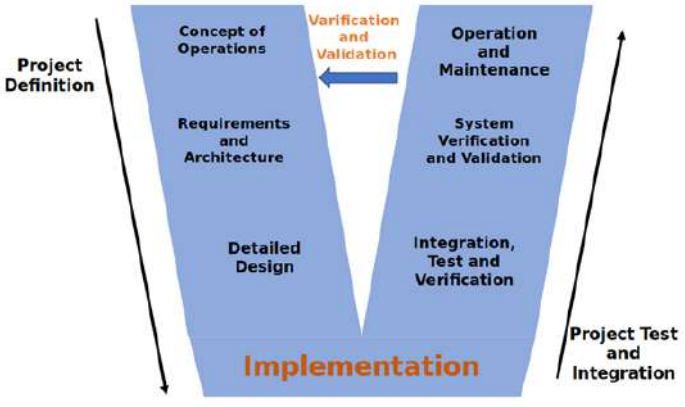
	כמויות גבוהות של סיכון ואי וודאות
	לא גמיש
	מודל לא מומלץ לפרויקטים מורכבים

לסיכום, **מודל המפל** ניתן להשתמש בפרויקטים עם הבנה ברורה ומובנת של דרישות הפרויקט, התכנון, הכלים הטכניים והתשתיות.

מודל V – V Model

אחד המודלים המוכרים ביותר בתהליך הפיתוח, שמו של המודל נגזר מצורת סכמת הפרויקט.

הרעיון המרכזי במודל ה-V הוא שמשימות פיתוח ומשימות בדיקה הן פעילויות תואמות בעלות חשיבות שווה, המסומלת על ידי שני הצדדים של ה-"V".



הליך הפיתוח מתקדם מהנקודה השמאלית העליונה של ה-V לכיוון ימין, ומסתיים בנקודה הימנית העליונה.

בענף ה-V השמאלי והמשופע כלפי מטה, אנשי פיתוח ומוצר מתחילים בעיצוב המטרות הפרויקט, משם ממשיכים בהגדרת דרישות העסקיות והגדרות התוכנה המפורטות, ומימוש הקוד.

בנקודת הבסיס של ה-V, צוות הפיתוח סיים לכתוב את הקוד.

בענף ה-V הימני והמשופע כלפי מעלה, נעשים הבדיקות וניקוי הבאגים: תחילה בדיקת היחידות, ולאחר מכן בדיקות האינטגרציה (שילוב מודולים), בדיקות המערכת, בדיקות קבלה מלמטה למעלה.

הנקודה הימנית העליונה של ה-V מייצגת שחרור מוצרים ותמיכה שוטפת.

כמו במודל המפל, כל שלב חייב להסתיים לפני תחילת השלב הבא, מודל V הוא למעשה גרסה משודרגת של מודל המפל.

מודל ה-V מסדיר את תהליך הבדיקות, כך שתיעוד בדיקות הקוד נכתב בד בבד עם שלבי הפיתוח. המשמעות היא שיש לתעד את מבחני האינטגרציה כאשר הסתיים התכנון ברמה הגבוהה. בדיקות היחידה צריכות להיות מוכנות כאשר המפרט המפורט נקבע.

במודל זה, על הבוחנים להיות מעורבים במימוש הפיתוח בהקדם האפשרי.

כמו תהליך הבדיקה של מודל המפל, תיקון תקלות ניתן לעשות בכל שלב במחזור החיים, אך עלות איתור ותיקון תקלות עולה באופן דרמטי עם התקדמות הפיתוח.

יתרונות	חסרונות
בשל העובדה כי במודל ה-V מתוקנים פגמים זמן קצר לאחר שאותרו, זול יותר לתקן אותם	מודל ה-V הוא נוקשה מאוד והכי פחות גמיש, המשמעות היא שאם אחת הדרישות משתנה, עיצוב ומימוש הקוד עשוי להשתנות ועל הבוחן לעדכן את תיעוד הבדיקה בכללותו
למודל יש מוניטין של בסיס טוב מאוד לחלוקת הבדיקה	המודל מצריך משאבים רבים – ולכן מתאים בעיקר לחברות גדולות
על כל המשתתפים בפיתוח מערכת אחריות על אבטחת איכות ובדיקה	
פעילויות כמו דרישות, עיצוב המבחנים מתקיימות הרבה לפני הקידוד ** עובדה זו חוסכת זמן רב וגם מסייעת בפיתוח, תורם להבנה טובה מאוד של הפרויקט בשלב הראשוני	
יעדי הבדיקה משתנים וספציפיים לכל רמת בדיקה	

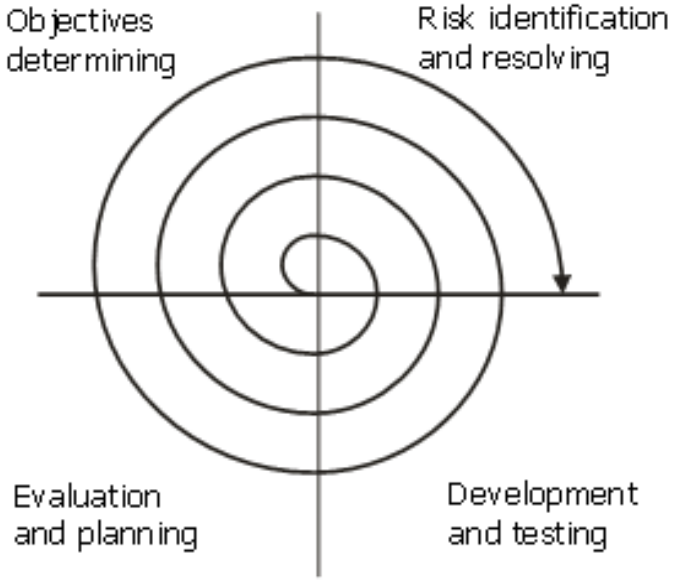
לסיכום: במודל ה-V מקום הבדיקה בתהליך הפיתוח הוא קריטי. איתור התקלות בשלב התכנון והעיצוב מתרחש בשלב המוקדם של תהליך הפיתוח המספק את החלופה הזולה יותר לתיקונה.

מודל הספירלה – Spiral Model

מודל פיתוח זה משלב את התכונות של דגם המפל.

המודל הספירלי הוצג לראשונה על ידי **בארי בוהם** במאמרו **"מודל ספירלי לפיתוח תוכנה ושיפור" (1985)**.

מודל זה לא היה המודל הראשון שדן בפיתוח איטרטיבי, אך הוא היה המודל הראשון שהסביר מדוע יש חשיבות לאיטרציה.



פיתוח מצטבר פירושו שהפרויקט לא נעשה על ידי חלקים (אולי גדולים), אלא כסדרה של פיתוחים ומשלוחים קטנים יותר.

מודלים מצטברים מנסים להפחית את הסיכון לפתח מערכת שגויה על ידי אספקה מוקדמת של חלקים שימושיים במערכת וקבלת משוב לקוחות.

עלות איתור ותיקון תקלות עולה באופן דרמטי עם התקדמות הפיתוח

דרישות פונקציונליות מערכת ואמינות יגדלו עם הזמן, החל מגרסה מוקדמת רק עבור הפיתוח או לשימוש ראשוני עבור המשתמשים, ועד לגרסה שתשוחרר ללקוחות הסופיים מאוחר יותר.

מה לגבי בדיקות במודל הספירלה?

הבדיקות צריכות להתבסס על מקרה בדיקה רב פעמי לכל רכיב, יש לעשות בו שימוש חוזר ולעדכן בכל תוספת נוספת.

אם בדיקות לא מתעדכנות עם התוספות, אמינות התוכנה נוטה לרדת עם הזמן במקום לגדול.

יתרונות	חסרונות
מודל מחזור חיים ספירלי הוא מודל גמיש מאוד. את שלבי הפיתוח ניתן לקבוע על ידי מנהל הפרויקט, בהתאם למורכבות הפרויקט	לא עובד טוב בפרויקטים קטנים

אומדנים (כלומר תקציב, לוח זמנים וכו') נהיים מציאותיים יותר ככל שמתקדמת העבודה מכיוון שנושאים חשובים מתגלים מוקדם יותר	גילוי מאוחר של הסיכונים הכרוכים בפרויקט עשויה להעלות את העלות והיא עשויה להיות גבוהה מעלות בניית המערכת
המודל מומלץ לפרויקטים גדולים	ניתוח סיכונים דורש מומחיות מאוד ספציפית

לסיכום: המודל משמש כאפשרות הטובה ביותר לעסקים עם יעדים עסקיים תנודתיים, אך כאשר יש צורך באב-טיפוס לטיפול במורכבות ההליכים העסקיים. **מודל הספירלה** משמש בעיקר בפרויקטים גדולים.

עבור פרויקטים קטנים יותר, סביר כי מודל האג'ייל יתאים יותר.

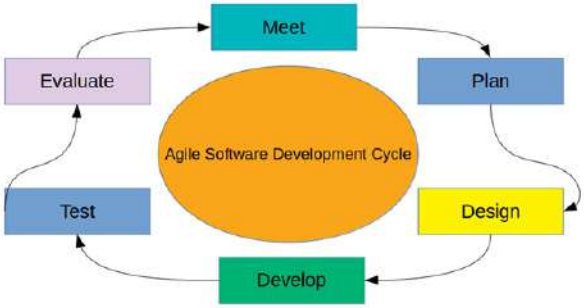
אג'ייל – Agile

מודל Agile זה של פיתוח תוכנה הוא מסגרת רעיונית להנדסת תוכנה המקדמת איטרציות פיתוח לאורך מחזור החיים של הפרויקט.

הגישת אג'ייל שמשמעותה כשמה כן היא מרמזת על מה לעשות במהירות רבה.

שיטת אג'ייל שואפת לעבוד באפקטיביות רבה, אך יחד עם זאת, שיפור איכות המוצר המפותח הוא מטרה מרכזית המושחלת בגישה.

ישנן שיטות פיתוח זריזות רבות; ברובן ממזערים את הסיכון על ידי פיתוח תוכנה בפרקי זמן קצרים.



תוכנה שפותחה במהלך יחידת זמן אחת מכונה איטרציה, שעשויה להימשך בין שבוע לארבעה שבועות.

כל איטרציה היא פרויקט תוכנה שלם: כולל תכנון, ניתוח דרישות, קידוד, בדיקות ותיעוד.

בבדיקות זריזות בדיקות מתבצעות במקביל לפיתוח הפיצ'ר. השאיפה היא לעבוד אינקרמנטלי עם צוות פיתוח, ברגע שמומש חלק מהקוד הקוד מועבר לבדיקות על מנת להתחיל לבדוק כמה שיותר מוקדם.

ישנם מאפיינים מקובלים לפיתוח אג'ייל :

- מחזורי שחרור גרסה קצרים.
- משוב מוקדם של הלקוח עוד במהלך הפיתוח ולא רק בסיומו.
- מענה לשינוי - התפתחות זריזה מתמקדת בתגובות מהירות לשינוי והתפתחות מתמשכת.
- התמקדות באנשים ואינטראקציות על פני תהליכים וכלים.

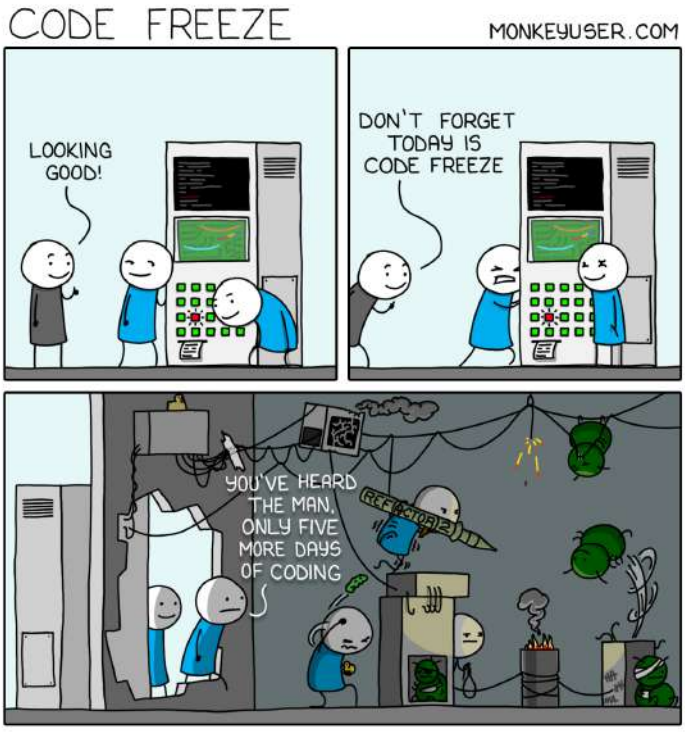
יתרונות	חסרונות
חוסך זמן וכסף	הפרויקט יכול לרדת בקלות מהמסלול אם ללקוחות לא ברור איזו תוצאה סופית הם רוצים
זמן Delivery מהיר	חסר דגש על תכנון ותיעוד הכרחי
התמקדות יותר ביישום במקום בתיעוד דברים	מעורבות משתמשים היא לרוב בעיה פוטנציאלית במיוחד בפרויקטים גדולים ומורכבים
הדרישות משתנות גם בשלב המאוחר של הפיתוח	פערים פוטנציאלים בתיעוד
התוצאה הסופית היא תוכנה איכותית לאורך הזמן ומותאמת לצרכי הלקוח	

לסיכום:

הגעתי למסקנה שעם פיתוח מהיר יותר, בדיקות ומשוב מתמיד מהמשתמש, מתודולוגיית אג'ייל הופכת לגישה המתאימה לפרויקטים שמדולוורים בפרק זמן קצר.

שיטות זריזות מדגישות גם תוכנית עבודה כמדד ההתקדמות העיקרי.

בשילוב עם העדפה לתקשורת פנים אל פנים, שיטות זריזות מייצרות מעט מאוד תיעוד כתוב ביחס לשיטות אחרות.



בדיקות אבטחה

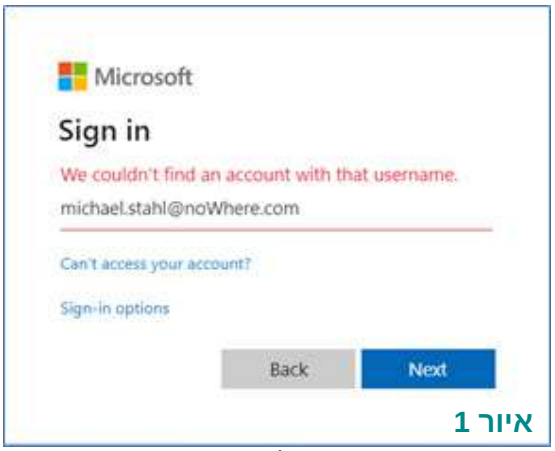
כשמדברים על בדיקות אבטחת תוכנה, האסוציאציה היא מיד ל - white-hat hacking, penetration testing וכדומה. אבל מסתבר שגם בודקי התוכנה ה"רגילים" – בני תמותה כמונו - יכולים לתרום הרבה לשיפור הקשיחות של התוכנה והעמידות שלה לפריצה. בדיקות אבטחה מסוג זה הן במידה רבה הרחבה של בדיקות פונקציונאליות שאנו ממילא מבצעים, ולא תחום חדש לגמרי. לעיתים זה פשוט עניין של שימת לב נוספת להתנהגות התוכנה תוך כדי הרצת בדיקות שממילא אנו מתכננים.

אחת הפונקציות שהקשר שלהן לאבטחת תוכנה ברור ומובן אינטואיטיבית היא ה-log in. כל אתר שמספק שירות וחלק מאתרי התוכן כוללים פונקציה זו, וכל סט בדיקות פונקציונאליות של האתר מכיל מן הסתם גם בדיקות של מסך הכניסה למערכת. למשל, בדיקה שמשתמש רשום יכול להכנס רק אם סיפק סיסמה נכונה; שמשתמש לא רשום לא נכנס, ושיש יכולת שיחזור סיסמה ("שכחתי סיסמה").

כיוון ששדות הקלט (שם משתמש, סיסמה) מקבלות מחרוזות, הרי שגם נבדוק מקרי קצה: מחרוזת ריקה, מחרוזת באורך 1 ומחרוזת באורך מקסימלי. עוד סט בדיקות שעדיין "יושב טוב" בתוך התחום הפונקציונאלי הם בדיקות שהקוד מבטיח איכות מינימלית של הסיסמה (לפחות 8 תווים; לפחות ספרה אחת, אות גדולה ואות קטנה, וכו'). כללים אלה נועדו להקשות על ניחוש הסיסמה על ידי פורצים. עד כאן, הכל פונקציונאלי: עובד נכון או לא עובד. מכאן אציין עוד מספר בדיקות ש"מתרחקות" מבדיקת היכולת הבסיסית של אימות המשתמש ומטרתן לוודא שמסך הכניסה לא סובל מחולשות שיקלו על האקרים לחדור לאתר. בדיקות אלו מתייחסות גם **לדרך המימוש** של אימות זהות המשתמש, ולא רק ל"עובד\לא עובד".

זליגת מידע

על מנת לפרוץ לחשבון של מישהו, צריך לפחות שני פריטי מידע: שם המשתמש וסיסמה. אם סתם אנסה המון צירופים של שמות משתמש וסיסמה, צפויה לי הרפתקה ארוכה ומתסכלת עד שבמקרה אפול על צירוף נכון. על כל שם משתמש שאנחש, עלי לנסות את כל הסיסמאות שאני מנחש. כמה נחמד היה אם יכולתי לקבל רמז ששם המשתמש שנתתי אינו תואם אף אחת מהשמות שרשומים באתר ולכן אין טעם לנסות שם זה עם המון סיסמאות! הנה דוגמה לא טובה: אני מנסה לעשות log-in לשרת של חברה קטנה מרדמונד, ומכניס שם משתמש שאינו קיים במערכת. בתגובה, המערכת טורחת להודיע לי ששם המשתמש אינו נכון! (ראו איור 1).

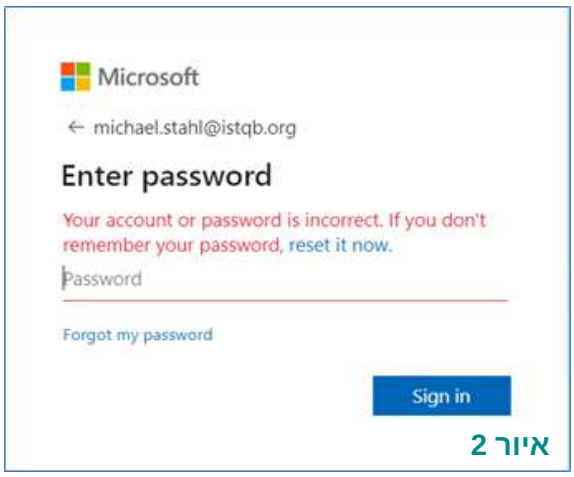


איור 1

התנהגות כזאת נקראת "זליגת אינפורמציה". המערכת מספקת מידע שהוא בעל ערך למי שמנסה לנחש שמות משתמשים. הודעה זו מדווחת שהשם michael.stahl@noWHere.com אינו קיים - ולכן הפורץ יכול לדלג על שם זה ולא לבזבז זמן בנסיונות ניחוש של הסיסמה שלו. לעיתים עצם העובדה שיש לי חשבון באתר מסויים יכולה לגרום לי מבוכה או צרות (ראו בקישור)

> עצם העובדה שיש לי חשבון באתר מסוים יכולה לגרום לי מבוכה או צרות

דרך אגב: בדקתי, וגם הכניסה לגוגל מתנהגת ככה... אני אתן לחברות מכובדות אלה להנות מהספק ולומר שכיוון שהאימייל שלי הוא לא בדיוק סוד, אין בעייה עם זה. או שהחבר'ה האלה יודעים מה הם עושים, ויש להם הגנות מספקות אחרות מעבר לשם המשתמש. אבל בעצם עדיף שמסך הכניסה ידרוש גם שם משתמש וגם סיסמה, וכשאחד מאלה אינו נכון, יתן הודעה כללית, כמו שאני מקבל בהכנסת סיסמא שגויה אחרי שם משתמש נכון (ראו איור 2)



איור 2

שימו לב שההודעה לא מספרת לי מה לא נכון – שם משתמש, הסיסמה, או שניהם – וזו התנהגות נכונה. בדוגמה כאן זה קצת מצחיק כי אי אפשר להגיע למסך הסיסמה בלי להכניס שם משתמש נכון, אבל העקרון עדיין תופס – וצריך לבדוק שזו ההתנהגות של מסך הכניסה למערכת.

העקרון של מניעת זליגת מידע תופס גם לגבי התנהגויות אחרות במערכת: האם קריאה שגויה ל-API; מייצרת הודעת שגיאה שמספקת לי מידע מיותר? האם נסיון גישה לדף שאינו קיים במערכת מספרת לי משהו שאוכל להשתמש בו למציאת פירצה (כגון: כתובת IP, הוורסיה של תוכנת השרת; מיקום מדוייק של קבצים על השרת, וכו')? כל אינפורמציה כזו מרחיבה את מה שפורץ פוטנציאלי יודע על האתר או התוכנה שלי, ואולי תספק משהו שניתן לנצל לצורך חדירה לאתר. בקיצור, כלל טוב (גם לחיים): עדיף לדבר מעט.

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.

גלישת חוצץ (Buffer Overflow)

"גלישת חוצץ" היא שגיאת תכנות המתבטאת בכך שתוכנית מחשב כותבת לאזור בזיכרון המחשב יותר מידע מאשר אותו אזור מסוגל להכיל. כתוצאה מכך "גולש" חלק מהמידע אל מחוץ לגבולות החוצץ, ומשנה נתונים שלא היו אמורים להשתנות. גלישת חוצץ עלולה, בין השאר, לאפשר הרצה של "קוד זדוני" הגורם לתוכנית לפעול באופן שלא תוכנן מראש. מבחינת אבטחת תוכנה, לא רק כתיבה אלא גם קריאה מאיזור שהוא מעבר למה שהוקצה לנתון מסוים, יכולה להוות פרצה הניתנת לניצול.

מה שמעניין הוא שהרבה פעמים הקוד – גם כשיש בו טעות שגורמת לגלישה – ירוץ בסדר גמור. יתכן שרק קומבינציה מאוד ספציפית של נתוני הרצה יגרמו לשגיאה שניתן לחוש בה באופן חיצוני. המשמעות של מצב זה היא שבמקרים רבים בדיקות פונקציונאליות לא יזהו שיש בעיה של גלישה.

אחת השיטות לבדוק אם יש גלישה, היא שימוש בטכניקה של fuzzing בשילוב עם וורסיה של הקוד שעבר מיכשור (אינסטרומנטציה) מיוחד.

(א) Fuzzing היא טכניקת בדיקות שבהם אלגוריתם מייצר מספר רב של קלטים שונים עבור התוכנה הנבדקת, שולח אותם לתוכנה, ובודק שהתוכנה לא נתקעה או התרסקה. אין כאן בדיקה פונקציונאלית האם התנהגות התוכנה מתאימה לקלט שנשלח – בעיקר כי הרוב המוחלט של הקלטים שהאלגוריתם מייצר הם קשקוש מוחלט ולא קלט הגיוני שאפשר לעשות איתו משהו. כל מה שאנו מנסים לעשות זה לייצר קלט שהתוכנה לא יודעת לזרוק או להתעלם ממנו. מצב כזה, שבו התוכנה "מתבלבלת" מהווה סיכון בחינת אבטחת תוכנה, כי יתכן שתגובת התוכנה תהיה לקרוס תוך זליגה של מידע, או להתקע לפני שהספיקה למחוק סודות (למשל, מפתחות הצפנה) מהזכרון.

(ב) מיכשור: שתי הפונקציות של הקצאת זכרון ושחרור זכרון (malloc; free) מוחלפות בקוד שמאפשר לבדוק אם כתיבות או קריאות נעשות רק מאיזורים בזכרון שמותר לתוכנה לגשת אליהן. ברגע שהקוד מזהה גישה לאיזור אסור בזכרון (שזה מה שקורה בגלישת חוצץ), הוא מקריס את התוכנה. שיטה זו נקראית AddressSanitizer או בקיצור: Asan.

> לעיתים, כדי למצוא בעיות אבטחת מידע, כל מה שצריך זה מודעות

הרצה ארוכה של fuzzing (שעות או ימים!) מעלה את הסיכוי ליצירת שילוב של קלטים שיגרום לגלישת חוצץ. שילוב של הרצת fuzzing על קוד שעבר מיכשור של AddressSanitizer יגרום לקוד לקרוס ברגע שגלישה כזו קרתה, וכך נוכל לזהות שיש בקוד בעיית גלישה.

פעילות בדיקה מסוג זה נופלת בין אחריות של קבוצת הבדיקות "הרגילה" לבין אחריות של מומחים באבטחת תוכנה. גישה פרקטית היא שפיתוח ה-fuzzer והמכשור של הקוד נעשה על ידי מומחי אבטחת התוכנה, והאחריות להרצת בדיקות אלה מידי פעם (נגיד, על כל וורסיה חדשה של התוכנה) מוטלת על קבוצת הבדיקות. אציין גם שהתמחות בטכניקת fuzzing מספקת מסלול קידום לאנשי בדיקות, שמאפשר להם לקחת אחריות גם על הצד המתקדם יותר של פיתוח ה-fuzzer.

מנסיון, גם הרצה של הבדיקות הרגילות (בלי fuzzer) על קוד שעבר מיכשור של AddressSanitizer מוצאת בעיות של גלישות זכרון. חלק מהגלישות לא מחייבות קלט מאוד ספציפי, ויקרו גם על הקלט שכבר קבעתם לבדיקה מסויימת. זה הופך את השיטה לכלי יעיל מאוד במציאת חולשות בקוד.

> יתכן מאוד שקוד – גם כשיש בו גלישת חוצץ – ירוץ בסדר גמור

יצירת הקוד המיוחד אינה קשה; העלות העיקרית של פעילות זו היא הצורך בהרצה של סבב בדיקות שאי אפשר להתחשב בתוצאותיו כיוון שהסבב מבוצע על קוד שעבר מיכשור והוא שונה מהקוד של המוצר הסופי (זו בדיוק אותה בעיה שיש כשמריצים בדיקות לצורך הערכת כיסוי הקוד – code coverage). שימו לב ששפת התכנות שבה מפתחים את המוצר משפיעה על הסיכון שהקוד סובל מגלישות. #C למשל כמעט ואינה רגישה לגלישה (צריך להתאמץ כדי לייצר את זה). C לעומת זאת, לא מוגנת כלל, אלא אם משתמשים בדגלים מסוימים בזמן הקומפילציה, כמו למשל fstack-protector-. מומלץ לחפש חומר בנושא ולראות מה הקומפיילר שלכם יכול לעשות בתחום זה (חפשו: "security hardening compilation flags").

מתחת לפנס

לפעמים לא צריך להתאמץ יותר מידי כדי למצוא בעיות אבטחת מידע בקוד – כל מה שצריך זה מודעות ופתיחת עיניים. הנה כמה דוגמאות:

- הריצו את הקוד להחלפת סיסמה, תוך שאתם מקליטים (עם sniffer) את התעבורה בין המחשב שבו אתם משתמשים והשרת. וודאו שהסיסמה החדשה מוצפנת לפני שהיא נשלחת דרך הרשת אל השרת. לעיתים מה שעובר זה לא ממש הצפנה אלא digest (מעין "חתימה") של הסיסמא – שזה ממש לא הצפנה. אפשר להקליט את ה-digest, ושידור שלו ישירות לשרת (ולא דרך דף ה-log in) יעבוד יפה מאוד - השרת יחשוב שקיבל סיסמה נכונה.
- אם המוצר שלכם כותב קבצים זמניים או קבועים לדיסק, כדאי לפתוח אותם ולבדוק אם הם מכילים נתונים חסויים (נגיד, מספר כרטיס אשראי... מספר חשבון בנק...) ללא הצפנה.
- האם השרת שבניתם עובד עם cookies? איזה מידע נכתב בהם? האם יש שם מידע חסוי, שלא הוצפן ויכול להיקרא על ידי כל אדם עם גישה למחשב?

ישנן עוד הרבה בדיקות שחורגות מעבר לשאלה של "האם זה פועל". אפשר למצוא הרבה חומר ברשת שמלמד מה עוד כדאי לבדוק על מנת להבטיח שהתוכנה שלכם לא רגישה לפריצה או לשיתוף במידע חסוי.

לסיכום

שלושת הנושאים שהבאתי הם רק דוגמה לדברים שבעבר נחשבו כאחריות של מומחי אבטחת תוכנה (יש עוד!). החשיבות שיש למחשב ולחיבוריות בחיינו מצד אחד, והתגברות הסכנות לפריצה מצד שני, משמעותם שאבטחת מידע ותוכנה הם תחומים שכל בודק תוכנה צריך להבין. כמו כן סביר לדרוש שבדיקות בסיסיות של אבטחת תוכנה תהיינה כבר חלק מהבדיקות הרגילות של המוצר. לא לדאוג: עדיין תשאר עבודה למומחי הפריצות לחפש חורים במוצר – אבל זה יהיה משמעותית יותר יעיל אם הטעויות הפשוטות, שניתן לזהות בבדיקות ישירות ומתוך מודעות, מראש לא יכנסו למוצר שמומחים אלה יתקפו.

תודה לעמיתי יאיר נצר על ההערות לטור, ועל הפניות למקורות ברשת.

חשיבות ה-Traceability | ניקיטה יריומין

אתם עוקבים?

כנראה שכולנו שמענו לא אחת על המושג עקיבות (Traceability). על פי ההגדרות הפורמליות, עקיבות מוגדרת כ**יכולת לזהות קשר בין פריטים בתיעוד ובתוכנה**, כגון בין דרישות לבין בדיקות (תרגום חופשי מתוך מילון המונחים לתחום בדיקות התוכנה ISTQB, גרסה 2.2). עקיבות טובה אמורה לאפשר:

- ניתוח השפעות של שינויים.
- סקירה ובקרת הבדיקות.
- הבנה טובה יותר של היבטים טכניים של תהליכי הבדיקות לבעלי עניין ולשקף את מידת ההתקדמות של תהליכים אלה.
- לשקף תמונת מצב אודות איכות המוצר והתקדמות הפיתוח אל מול היעדים העסקיים.

אבל בינינו - כנראה שעבור רובנו עקיבות היא מילה נרדפת לעבודה סיזיפית ונטל פרויקטאלי.

הטענה שלי היא לא רק שעקיבות יכולה לתת לנו ערך מוסף משמעותי אף יותר מההגדרות שפורטו קודם, אלא יכולה להיות גם פשוטה יותר ופחות מאמללת לביצוע. כדי לנסות להוכיח זאת, אציג את הניסיון שלי בשני פרויקטים שונים שישקפו זאת. על מנת לא לחשוף את פרטי הפרויקטים, אתייחס אליהם לפי הסדר הכרונולוגי: *הפרויקט הראשון*, היה פרויקט ארוך טווח של פיתוח גרסת תוכנה חדשה (במינימום התאמות חומרה) למערכת מבוזרת הכוללת המון רכיבי לוגיקה פנימיים וייחודיים, עם תלויות מורכבות ביניהם. היכולת המרכזית של הגרסה הייתה מעבר לארכיטקטורת שרת-לקוח והחלפת ממשק המשתמש ממודול לינוקס ליישום חלונות וכדי לא לסבך את העניינים יותר מדי, הוחלט בשלב הראשון שהאפליקציה החדשה תשמור על פונקציונליות זהה לקודמת. גוף הפיתוח מנה כ-60 איש, שחולקו לצוותים על פי מרכיבי המערכת המרכזיים (GUI, בסיס נתונים, תשתיות ומערכת הפעלה, וכו').

תפקידי היה להוביל את הבדיקות, כאשר אתי בצוות עוד מספר בודקים. *הפרויקט השני*, היה פרויקט מו"פ של מערכת גילוי בטכנולוגיה חדישה. הפרויקט שילב מספר רב של דיסציפלינות הנדסיות כגון מכאניקה, בקרה, אופטיקה וכמובן תוכנה מסוגים שונים (כ-12 אפליקציות נפרדות שעובדות במקביל). צוות הפרויקט בשיאו מנה 8 אנשי פיתוח סה"כ, ותפקידי היה להוביל את הבדיקות ולנהל את הדרישות (לא להגדיר או לכתוב אותן, אלא רק לוודא שהן קיימות, מעודכנות ומאורגנות היטב). שני הפרויקטים התנהלו בשיטת מפל המים הקלאסית.

אציין שאחד מהפרויקטים הסתיים בכישלון, והשני נחל הצלחה רבה שאף עלתה על הציפיות של כל המעורבים בו. בואו נתעמק קצת:

בפרויקט הראשון

היו קיימים מסמכי דרישות אך ורק לממשק המשתמש החדש. הפער נסגר על ידי שילוב מפעילים מנוסים של המערכת בצוות הבדיקות, כמו גם בהוספת דרישת העל שבטח רובכם מכירים - "זה צריך לעבוד כמו בגרסה הקודמת". במעט המסמכים הקיימים נוהלה עקיבות בשיטה לא מאוד נוחה – סקריפט היה מופעל על מסמך דרישות ומוסיף מספר מזהה לכל דרישה פרטנית. בהמשך, כאשר הבדיקות נכתבו, מספרי הדרישות הללו צוינו בתיאור הבדיקה ובנוסף, לכל מפרט דרישות נבנתה טבלת עקיבות שפירטה את כל מקרי הבדיקה המקושרים לכל דרישה פרטנית. השיטה הזו החזיקה בדיוק עד העדכון הראשון למסמך הדרישות שכלל הוספה של דרישה חדשה, שכן עכשיו היה צורך למצוא דרך להוסיף מספר מזהה חדש לדרישה זו (הסקריפט לא ידע להתמודד עם עדכון, כמובן) ולעדכן את הטבלה ואת הבדיקות הרלוונטיות – דבר שהסתבר כלא טריוויאלי בכלל. הדבר שהכי הקשה על כל התהליך, היה העובדה שמסמכי הדרישות (המעטים שהיו קיימים) ומערכת ניהול הבדיקות שכנו ברשתות מחשבים מופרדות לחלוטין עקב אילוצי בטחון מידע. על אף הקשיים, כאשר גרסאות ראשונות התחילו להימסר לצוות הבדיקות והחלו ניסויים בסביבה המיועדת, הבאגים זרמו בהמוניהם – הבדיקות עשו את העבודה. לאורך הפרויקט כולו נמצאו מעל 6000 באגים, מתוכם כ-1200 ברמות חומרה קריטית וגבוהה.

בפרויקט השני

הוגדרה מראש טבלת אימות ותיקוף (Validation and Verification) – טבלה הממפה בין סעיפי דרישות המערכת הכלליים לבין האופן בו יש להוכיח את איכותם (על ידי קיום ניסוי פרטני, ביצוע בדיקה והצגת דו"ח, הוכחת שימוש ברכיב שכבר נבדק, וכו'). בפרויקטים בטחוניים, נהוג שטבלה זו מוגדרת על ידי הלקוח של הפרויקט. מכיוון שבמקרה זה לא היה לקוח – צוות הפרויקט הגדיר את הדרישות בעצמו. עיון קצר בטבלה זו מאפשר להבין שבפרויקט צריכים להתבצע מספר ניסויי ביניים מרכזיים שלכל אחד מהם מטרה אחרת, ניסוי מסכם של הוכחת היכולת הכללית, ומספר סבבי בדיקות תוכנה למודולים שונים.

עקיבות מוגדרת כיכולת לזהות קשר בין פריטים בתיעוד ובתוכנה

אירועים אלה נפרסו על תכנית הפרויקט, ומתוכם נגזר ה-STP (Software Test Plan). מכיוון שבכל אירוע כזה הוגדרו סעיפי הדרישה הפרטניים שיש לבדוק וסעיפי דרישה אלה היו מקושרים למסמכי אפיון ודרישות פרטניים לרכיבי המערכת – העקיבות הייתה מובנית, צוות הפרויקט לא היה צריך להתאמץ כדי לייצר אותה.

תרחישי הבדיקה נכתבו בהתאם ובמהלך כתיבתם עלתה מספר פעמים ההבנה שנכתבות בדיקות העתידות להיכשל, שכן לא הושקע בפיתוח של רכיבים כאלה ואחרים. הבנה זו הובילה לשינוי מידי במיקוד הפרויקט והשלמת פיתוח היכולת החסרה, שכמובן כבר לקח בחשבון את הבדיקות שנכתבו.

ניקיטה יריומין

בתחום הבדיקות משנת 2010.

הובלתי בדיקות במספר פרויקטים בטחוניים והקמתי את תחום הבדיקות והאיכות בחברת Puzzle Projects.

מאז 2016 משמש כמהנדס מערכת ומזה שנה Product Owner עבור לקוח בטחוני, במקביל לניהול התחום בחברה

חשיבות ה-Traceability | ניקיטה יריומין

ניחשתם מי הפרויקט הכושל?

הפרויקט הראשון נכשל מכיוון שכמות הבאגים שנמצאה עלתה על כל דמיון, רובם היו מהותיים ולא התרכזו בממשק המשתמש החדש שהיה מתועד היטב, אלא דווקא ביכולות הליבה של המערכת שכאמור לא היו קיימים מסמכי דרישות עבורן. המערכת נדחתה על ידי הלקוח והזמן שהוערך לפתרון כלל הבאגים המהותיים היה שקול לפיתוח גרסה נוספת בגודל דומה.

הגרסה נגנזה ומיד החל פרויקט ההמשך, שרובו היה תיקון הליקויים של הגרסה הכושלת.

הפרויקט השני הצליח מכיוון שכמות הבאגים הייתה נמוכה מאוד, עקב המיקוד הפרטני בדרישות המערכת הכלליות, ויעדי הניסויים הצפויים. באותם הניסויים, המערכת הפתיעה בביצועים שלה גם את צוות הפרויקט וגם את בעלי העניין השותפים למו"פ, שמיהרו להתניע תהליכי רכש למערכת. ניחשתם נכון?

כמובן שהעקיבות לבדה לא הייתה מה שגרם לפרויקט הראשון להיכשל וגם לא לפרויקט השני להצליח. העקיבות הייתה רק סימפטום. כמו בכל פרויקט (ובכלל, כל דבר בחיים) – אם אנחנו לא יודעים למה אנחנו עושים את מה שאנחנו עושים – רוב הסיכויים שלא נצליח. רישום עקיבות רק לצורך תיעוד, החלפת תשתית ממשק המשתמש של המערכת מבלי להתייחס ל-Backend שלה, ומציאת באגים ללא כוונה או תכנית לתקן אותם – כל אלה מעידים על כך שהייתה כאן תכנית ללא כוונה ברורה. מנגד, בפרויקט השני המטרה הייתה ברורה והוגדרה היטב. מתוך הגדרה זו היה טבעי לבצע עקיבות לאופן היישום והבקרה על התוצר. אבל מה שיפה באמת, זה שבפרויקט השני, עצם הביצוע של העקיבות היה הדבר שאילץ את הפרויקט להתמקד בדברים הנכונים. בתחילת הפרויקט, לאחר שה-STP הוצג לצוות לראשונה, הפרויקט הבין שמתוכנן להתקיים ניסוי שנכון לאותה נקודת זמן לא תוכנן הפיתוח של הרכיב שהיה צריך להיבדק שם, כאשר פורטו הבדיקות המתוכננות בניסוי זה, מהנדס המערכת סיכם ש"הבדיקות האלה יכשלו כי אנחנו לא מפתחים את זה כך וזו טעות". בשיח של רבע שעה סביב הנקודה, הוחלט להסיט את מאמצי הפרויקט לפיתוח אותה היכולת בשלב מוקדם יותר.

השיטה האג'ילית מאלצת אותנו לבצע עקיבות כל הזמן

בדיעבד, החלטה זו הייתה אחד הדברים שהובילו להצלחה בניסוי הצגת התכלית הסופי. בנוסף למקרה זה, היו עוד מספר מקרים בהם ביצוע העקיבות הוביל להגדרה של דרישות מפורטות יותר ולפיתוח של כלים מיוחדים שנדרשו לטובת קיום הבדיקות בפרויקט. **העקיבות גרמה לנו להבין במה צריך להתמקד**, בהתאם למטרה הסופית של הפרויקט.

כמובן, שגם היתרונות שהוזכרו בהגדרת העקיבות באו לידי ביטוי – הודות לעקיבות, בכל שלב בפרויקט הייתה לנו תמונה טובה של כיסוי הדרישות והבדיקות, והתמונה הזו בשילוב עם סטאטוס הביצוע של כל הבדיקות נתנה להנהלה תמונת מצב ברורה שאפשרה לקבל את ההחלטות הנכונות.

אם אנחנו לא יודעים למה אנחנו עושים את מה שאנחנו עושים – רוב הסיכויים שלא נצליח

צייניתי קודם שאני מאמין שעקיבות יכולה להתבצע בקלות, כאן המקום לדבר על האתגרים הנפוצים ביישום עקיבות:

ככל הנראה הקושי הנפוץ ביותר הוא הצורך בתיעוד לאחור, כאשר מחליטים ליישם עקיבות כבר בתהליך העבודה – אף אחד לא רוצה להיות זה שעובר על כל הבדיקות שכבר נכתבו ומקשר אותן לדרישות.

קושי נפוץ נוסף הוא כמובן ביצוע עקיבות כאשר אין סדר במסמכי הדרישות ו/או הבדיקות. בהקשר לשני הקשיים הללו, אני טוען שמה שציינתי קודם נכון גם במקרה הזה – העקיבות מאלצת את הפרויקט לשים לפניו תמונת מראה שתשקף לו את מצבו.

נכון, קשה ליצור תיעוד בדיעבד, אבל זה משתלם בטווח הארוך. נכון, זה לא נעים לגלות שחסרים מסמכים בפרויקט – אבל זה עדיף מאשר להתנהל באי וודאות לא מבוקרת.

כאמור – **העקיבות מאלצת אותנו לעשות את מה שצריך**. כיצד להימנע מהקשיים הללו? ללא ספק, הטיפ הכי טוב שיש לי בהיבט הזה הוא להתחיל לבצע עקיבות מוקדם ככל הניתן. אם זה לא אפשרי – זכרו את היתרונות של העקיבות, קחו נשימה עמוקה, ועשו את הדבר הנכון.

לפני שנסכם, חשוב להתייחס לעקיבות בעידן האג'ילי. כאמור, שני הפרויקטים שהזכרתי נוהלו לפי מודל מפל המים – אז מה יכול ללמוד מזה ארגון "זמיש" (זריז וגמיש)? אז תתפלאו – **השיטה האג'ילית מאלצת אותנו לבצע עקיבות כל הזמן**, מבלי לקרוא לזה כך. מה היא הגדרת Definition of Done ו-Acceptance Criteria אם לא עקיבות בין הדרישה לתוצאה הצפויה של הבדיקה? מה הוא מימוש איטרטיבי ומדורג של יכולות?, נכון מאוד – דרך להוכיח שאנחנו בודקים את מה שרצינו שיפותח – או במילים אחרות – עקיבות.

כמובן, שבעבודה באג'יל יש אתגר אחר בהיבט העקיבות והוא ניהול המעקב אחר השינויים התכופים שקורים במוצר.

מה שאמור לתת מענה לכך הוא עבודה צמודה בין צוות הפיתוח לאנשי המוצר שצריכים להגדיר מה נכון ומה כבר לא, בהתאם לכך להתאים את בדיקות הנסיגה (Regression) (שבשאיפה הן אוטומטיות) לרכיבי המוצר ה"וותיקים", שפיתוחם הסתיים כבר מזמן.

לסיכום

אני רוצה להעלות שתי נקודות למחשבה:

1. כפי שראינו בדוגמאות, תקלות יהיו תמיד, ועקיבות טובה ככל שתהיה לא תעזור לפתור אותן. אבל זכרו – **עקיבות לא פותרת באגים, היא עוזרת לנו להתמקד**. במערכת שפותחה עם מיקוד נכון, יהיו פחות באגים, והם יהיו פחות מהותיים.

2. אני מציע הסתכלות אחרת על עקביות: **תחשבו עליה לא כעל אחת מהמשימות של אנשי הבטחת האיכות, אלא כעל כלי ניהולי**.

כל היתרונות של עקיבות שהוזכרו, משמשים את מנהל הפרויקט או המוצר הרבה יותר מאשר רק את הבודק. אם ביצוע עקיבות יהיה משהו שהמנהלים יהיו מעוניינים בו ויתעקשו עליו, כל הפרויקט יוכל לקצור את הפירות של יישום העקיבות. תפקיד הבודקים הוא לשכנע את המנהלים בחשיבות של העקיבות.

עקבתם? עכשיו תורכם לחשוב איך זה עוזר לכם בעבודה שלכם. בהצלחה!

בחן את עצמך | טל פאר

הסילבוס של ISTQB® לרמת הבסיס (CTFL) מקדיש פרק קטן לכלים התומכים בבדיקות. לא מדובר רק על כלים לאוטומציה, אלא על כלל הכלים שעשויים לשמש אותנו במהלך הבדיקות.

על אוטומציה תוכלו ללמוד עם הסילבוס CTAL-TAE Test Automation Engineer של ISTQB®.

בדרך כלל, כשמוזכר הנושא של כלי בדיקות, אנשים מתכוונים לכלי הרצה אוטומטית של בדיקות. אלה כלים שמשלבים הקלטות של פעולות הבודק או המשתמש עם קוד, אותו מתחזק מהנדס בדיקות אוטומטיות. כלים אלה משמשים אותנו בבדיקות בצורה ישירה.

אבל ישנם עוד כלים בהם אנחנו משתמשים. קיימים כלים המסייעים לבודקים ולמנהלי הבדיקות לנהל את הבדיקות, את מקרי הבדיקה (test cases), את דו"חות הפגמים (defect reports), הדרישות וכל מה שמשמש בבדיקות. כלים אחרים משמשים לניתוח והערכת תוצאות הבדיקה או של המערכת (לדוגמה, כלים המשמשים לניתוח ביצועי המערכת).

כל כלי שנשתמש בו, עלינו להבין את היתרונות והחסרונות שלו, להגדיר את דרכי העבודה עם הכלי וכמובן, לבחור את הכלי המתאים לנו ביותר.

טל פאר

בעל ניסיון של יותר מ-20 שנים כבודק ומנהל בדיקות במגוון חברות וטכנולוגיות במודלי פיתוח שונים. כיום טל יועץ ומדריך בדיקות עצמאי.

טל חבר ב-ITCB® וגזבר הארגון העולמי ISTQB®.

השאלה של היום מדברת על יתרונות של כלי לביצוע בדיקות.

איזה מהבאים הוא בעל הסבירות הגבוהה ביותר להיות יתרון של כלי ביצוע בדיקות (test execution tools)?

א. קל ליצור בדיקות נסיגה (regression tests) בעזרת הכלי

ב. קל לתחזק בקרת גרסאות של תוצרי הבדיקות בעזרת הכלי

ג. קל לעצב בדיקות אבטחה (security tests) בעזרת הכלי

ד. קל להריץ בדיקות נסיגה (regression tests) בעזרת הכלי

*לצפייה בתשובה המפורטת - דפדפו לעמוד 26

המדריך לאוטמציוניסט המתלבט, ראש בראש: Java, C# & Python

יאיר נסימוב

אז המנהלת שלחה אתכם למשימת התאבדות. נדרשתם להרים תשתית אוטומציה שעובדת 100% לכל 358 הבדיקות הידניות שלכם בזמן שהיא מסיימת את כוס הקפה של הבוקר. אך, כשפתחתם את הדפדפן בחיפושים אחר אותה תשתית נכספת נתקלת במספר שפות תכנות שיתאימו למשימה אם גם אתם מרגישים אבודים, הגעתם למקום הנכון. מאמר זה סוקר את שלושת שפות התכנות המובילות כיום בעולם פיתוח האוטומציה ואת הכלים הנלווים אליהם מתוך רצון לעזור לכם בבחירה הלא פשוטה.

נתחיל בקצת רקע על כל אחת מהן, נעבור על כמה פופלארית השפה, מהן יתרונותיה וחסרונותיה גם כשפת תכנות וגם כשפה לכתיבת בדיקות אוטומטיות. נמשיך בבדיקה של אילו מערכות להפקת דו"חות לבדיקות אוטומטיות תומכות בכל שפה ונסכם בהמלצה של כיצד להתאים את השפה הנכונה לפרויקט שלכם.

*חשוב לציין שהמאמר מתמקד בכתיבת בדיקות אוטומטיות בסלניום (Selenium), אך כל האמור רלוונטי לחלוטין גם אם אתם עובדים על פרויקט סלולר ומשתמשים באפיום (Appium).

יאיר נסימוב
בעל ניסיון של כ-10 שנים בתחום הבדיקות האוטומטיות.
מייסד חברת Rain the Dog המתמחה בייעוץ ומתן פתרונות אוטומציה מותאמים אישית לחברות.
מרצה ומפתח קורסי אוטומציה. יוטיובר מתחיל
אספן תקליטים, חובב טיולים, ריצה ואגרוף תאילנדי
נשוי + 2 + כלב - Rain.

ג'אווה – Java

זוהי שפת תכנות מונחית עצמים (OOP) אשר נועדה לשימוש נרחב ככל הניתן וייחודה מתבטא בשימוש שלה במכונה הוירטואלית (JVM). באמצעות שימוש במכונה הוירטואלית אנחנו יכולים לכתוב קוד במכונה אחת ולהריץ אותו על כל מכשיר או מחשב שמותקנת עליו המכונה הוירטואלית. השפה רצה על מגוון רחב של מכשירים, החל ממחשבים ביתיים ותעשייתיים, רכיבי תוכנה, מכשירים סלולריים ועוד. זוהי אחת משפות התכנות הפופולריות ביותר, עם יותר מ-9 מיליון משתמשים.

מוסדות לימודיים רבים, בהן מוסדות אקדמיים משתמשים בג'אווה כשפת ההוראה כשלומדים נושאים הקשורים לתכנות.

מהסיבות שהוזכרו לעיל ומסיבות נוספות, אין ספק שהתמקדות בלימוד השפה יכולה להיטיב עם הקריירה שלכם. מבחינת מפתחים רבים, ג'אווה היא הימור בטוח. שיטוט מהיר לאתר אולג'ובס וסינון תחום התוכנה לפי מילת החיפוש "Java" מביא ל-229 משרות פנויות כרגע[1], יותר מכל שפה אחרת. אם נוסיף לכך את המשרות הפנויות למפתחי אנדרואיד - 43, שם מרבית הפיתוח מתבצע בג'אווה, אנחנו מקבלים קרוב ל-300 משרות פנויות ברגע נתון, כל זאת לפני שבכלל בדקנו כמה מפתחי אוטומציה שיודעים את השפה נדרשים.

סביבות הפיתוח הפופולאריות ביותר לשפה הן Eclipse ו-IntelliJ. שתי סביבות מוכרות אלו נמצאות בשימוש נרחב במגוון חברות, מוסדות לימודיים ומדיניים. כמו כן, כפי שהוזכר קודם לכן, ג'אווה היא שפת התכנות העיקרית המשמשת לפיתוח באנדרואיד, כך שגם Android Studio, המבוססת IntelliJ, היא סביבת פיתוח פופולארית לג'אווה.

בעולם האוטומציה לשפה יש כלים רבים שקל לשלב אותם כחלק מפרויקט האוטומציה שלכם ובראשם TestNG, שהוא אחד מה"פריימוורקים" המפותחים ביותר לבדיקות וכולל מגוון אפשרויות של שימוש ב-Testing annotations, listeners, data providers ועוד. TestNG נתמך ככלי "Built in" ב-Eclipse & intelliJ, כך שתוכלו להריץ את הבדיקות שלכם ישירות מסביבת הפיתוח ללא התקנה של Plugins.

כמו כן, מאגר התיעוד של פתרונות ובעיות למרבית האזורים שאותם תרצו לכסות בבדיקות שלכם הוא אדיר, כך שכמעט אף פעם לא תצטרכו להמציא את הגלגל מחדש. בדיקה זריזה באתר stackoverflow של צירוף המילים של Java + Selenium מוביל ל-19,167 אשכולים שונים בהם נשאלו שאלות בנושא[2].

דבר חשוב נוסף הוא שכ-77% מכלל כותבי האוטמציה בסלניום משתמשים ב - Java[3].

אם כן, ראינו שמספר לא מבוטל של מפתחי Java קיימים בשוק ולכן למנהלי הפיתוח והבדיקות יהיה קל יותר לשכור מפתח אוטומציה מנוסה שיש לו ידע קודם ב - Java מאשר שפות תכנות דומות.

Java כוללת גם תמיכה בממשק ניהול התלויות הפופולארי - Apache Maven, שנותן גישה באופן מיידי למאות אלפי ספריות, גם כאלו הקשורות לבדיקות אוטומטיות. באמצעות שימוש ב-Maven תוכלו לשתף את הפרויקט שלכם עם חברי צוות בקלות, ללא צורך בהתקנה של ספריות חיצוניות באופן ידני.

כל הנתונים האלו מחזקים את הבחירה ב-Java כשפה הנכונה עבור פרויקט האוטומציה שלכם, אך מהן החסרונות העיקריים של שימוש ב-Java?

ג'אווה זוהי שפת תכנות מונחית עצמים (OOP) אשר נועדה לשימוש נרחב במגוון מערכות

בין החסרונות הבולטים ניתן למצוא את העובדה ש-Java מתרגמת כל קוד בזמן ריצה לקוד שאמור לרוץ ב-JVM, דבר המאט את הפעלת התוכנית שלכם. חסרון נוסף הוא ש-Java היא שפה קשיחה. היא לא סלחנית לגבי כתיבה של מתודה בשם לא נכון, שימוש בפרמטרים לא נכונים או בשמות. כך שאפילו טעות אחת של אות קטנה במקום אות גדולה תוביל לכך שכל הפרויקט שלכם לא יוכל לעבור קומפילציה ולרוץ (בניגוד לPython למשל). כמו כן, לעיתים, כדי להשיג מטרה פשוטה ב-Java כמו קריאה של קובץ, נדרשות שורות קוד רבות (שוב, בניגוד לPython שם הדבר נפתר בשורה אחת בלבד).

ל-Java חסרונות נוספים שהם פחות רלוונטיים עבורנו ככותבי בדיקות אוטומטיות כגון: Java לא מאפשרת למפתחים שליטה ברמה נמוכה יותר של החומרה, כמו הקצאות זיכרון וכו'.

פייתון – Python

מה לגבי פייתון? מה לא נאמר על השפה האהודה הזו. על פי סקרים אחרונים שנערכו בשנת 2020, פייתון עוקפת את ג'אווה בפופולאריות שלה ומתייצבת במקום השני בין שפות התכנות הפופולאריות ביותר ברשת (אחרי JavaScript[4]). כמובן שהדבר לא אומר שרוב המערכות בעולם כתובות בפייתון אלא שכמות השאלות הכוללות שימוש בשפה שנשאלות באתרים דוגמת stackoverflow הייתה גדולה יותר משל שפות אחרות, ושכמות החיפושים של השפה במנועי החיפוש הייתה גדולה יותר.

האינטואיטיביות של השפה, הקלילות שלה, יחד עם היכולת להריץ אותה במגוון צורות הופכת אותה לפופולרית בקרב מפתחים רבים בכלל ומפתחי אוטומציה בפרט. פייתון יכולה לשמש גם כשפת סקריפט וגם כשפה מונחית עצמים (OOP) לחלוטין. כך שהיא מאפשרת גמישות רבה יותר בנוגע לאופן השימוש שלנו בה.

השפה תומכת בירושה מרובה, דבר המאפשר ייצור של פרויקטים

1 נכון לתאריך כתיבת חלק זה במאמר 5.10.20
2 https://stackoverflow.com/questions/tagged/java+selenium
3 https://www.gcreddy.com/2016/05/advantages-and-disadvantages-of-selenium.htm
4 https://redmonk.com/sogrady/2020/07/27/language-rankings-6-20/

המדריך לאוטמציוניסט המתלבט, ראש בראש: Java, C# & Python

יאיר נסימוב |

גמישים ומעניינים הרבה יותר. כמו כן, ניתן להציב ערכי ברירת מחדל בפרמטרים המגיעים למתודות שנקראות בקוד, דבר היכול להוביל לקוד קריא ופשוט יותר.

בדיקה מהירה באתר אולג'ובס וסינון תחום התוכנה לפי מילת החיפוש "Python" מביא ל-144 משרות פנויות כרגע[1], כחצי ממה שראינו עבור ג'אווה, אך היא חולקת את המקום השני יחד עם C#. כמובן שאם נציץ בכמות המשרות של מפתחי אוטומציה הדורשים ידע בפייתון, נגיע למספר גבוה בהרבה.

סביבות הפיתוח הפופולאריות לשפה הן Pycharm ו- Visual studio code. שתי הסביבות מהוות יתרון, בהתאם לחברה בה אתם עובדים או לנסיון הקודם שלכם. Pycharm היא סביבת פיתוח אחות ל-IntelliJ, כך שאם יש לכם ניסיון קודם בשימוש בכלי, מיד תרגישו בבית. הסביבה השניה נועדה לפרויקטים מבוססי שירותים של מיקרוסופט כך שאם אתם עובדים בצוות, או שאתם בעלי רקע של Net. או C# כמובן שהכל יראה לכם מוכר. כמובן שאפשר לכתוב פייתון בעוד עורכים רבים, כולל עורכי קוד שאפשר להתקין על האייפד שלכם, או שימוש ב-Vi במערכת הפעלה מבוססת Unix. פייתון גם ניתן להרצה בקלות משורת הפקודה, דבר שיכול להפוך אותה לשפה האולטימטיבית המותקנת בשרתים נטולי ממשק משתמש (UI).

מלבד השימוש בפייתון וסלניום ביצירת בדיקות אוטומטיות, השפה גם פופולרית בשימוש של web scraping ו-data scientists, שם הקלילות שלה והיכולת להריץ אותה על מגוון מכשירים ללא צורך בהתקנות מסובכות הופכת אותה לבחירה ברורה עבור רבים.

פייתון עוקפת את ג'אווה בפופולאריות שלה ומתייצבת במקום השני בין שפות התכנות הפופולאריות ביותר ברשת (אחרי JavaScript)

מבחינת הבחירה בפייתון כשפה לפרויקט אוטומציה, כמובן שבשימוש בסלניום אין הבדל פונקציונלי לעומת השימוש בג'אווה. לפייתון יש את Py Test פריימורק לבדיקות וכמות האפשרויות בה מרשימה. Py Test מאפשרת כתיבה של טסטים רבים עם תשתית מאחורי הקלעים שיכולה לקשר בין רכיבים שונים של הבדיקות לקוד יעיל וקריא יותר. בבדיקה באתר stackoverflow של הצירופים python + Selenium מתקבלות 22,423[2] תוצאות של אשכולות בהם נשאלו שאלות בנושא. נתון זה לבדו מעניק לשפה את הבכורה כשפת התכנות הפופולארית ביותר בקרב כותבי אוטומציה. אין ספק שהמסקנה בנידון מורכבת יותר שכן ראינו בחלק שסקר את ג'אווה שכמעט 80 אחוזים מכלל המפתחים העושים שימוש בסלניום עושים זאת עם ג'אווה, אך התחושה שמתקבלת היא הפופולאריות של פייתון נוסקת במהירות בתחום הבדיקות האוטומטיות כשם שהיא נוסקת בתחומי תכנות אחרים.

אחד החסרונות הבולטים של השפה בפרויקט אוטומציה רחב היקף הוא הגמישות הרבה **מידי** של השפה. אין כלל כפיה של סוגי המשתנים, או הערכים המוחזרים ממתודות מסוימות, דבר היכול להפוך את הקוד לבלתי קריא ככל שהפרויקט הופך למורכב יותר ויותר. על כן, מכיוון שפייתון לא זורקת שגיאות בעת הקומפילציה, לעיתים לא נדע שיש לנו קטע קוד שלא עובד עד שלא נריץ קטעים מסוימים מהתשתית, דבר שיכול להקשות על עבודת צוות מקבילית. כמו כן, אם כבר במקבילות עסקנו, היכולת להריץ טסטים מקבילים בפייתון פחותה באופן משמעותי לעומת ג'אווה בגלל המחסור ב-Threads. חסרון בולט נוסף של פייתון לעומת ג'אווה בעולם האוטומציה הוא העובדה שלג'אווה יש את TestNG שזו מסגרת הבדיקות השלמה והמפותחת ביותר שיש כיום בשוק. לכן, עם כל הכבוד ל-Py Test, היא לא יכולה לספק את מה ש TestNG יכולה.

C#

זוהי שפת תכנות מונחית עצמים שפותחה על ידי מיקרוסופט במטרה להוות את התחליף העיקרי לג'אווה ב"אקו סיסטם" של מיקרוסופט - Net. ככזו, C# היא השפה הטובה המתאימה ביותר לפיתוח אפליקציות ללמערכת הפעלה "חלונות". אם אתם מגיעים עם רקע של ג'אווה, מיד תרגישו תחושה מוכרת, שכן הסינטקס של השפות דומה באופן מחשיד מעט. C# דומה מאוד לג'אווה גם בתכונות מונחות העצמים שלה ואף היא לא מאפשרת ירושה מרובה, אם כי היא מאפשרת מימוש מרובה של ממשקים (Interfaces).

בדיקה באתר אולג'ובס וסינון תחום התוכנה לפי C# מביא ל-143 משרות פנויות כרגע, כחצי ממשרות הג'אווה, אך כאמור היא חולקת את המקום השני יחד עם פייתון. כמובן שאם נציץ בכמות המשרות של מפתחי האוטומציה הדורשים ידע בשפה, נגלה שמשרות רבות מתווספות למספר לעיל.[3]

סביבת הפיתוח ל- C# היא Visual studio. סביבה זו מהווה יתרון, אם רכשתם ידע בפיתוח מוצר אחר של מיקרוסופט על גבי .NET. מפתחים רבים רואים ב-VS את סביבת הפיתוח האולטימטיבית עקב התמיכה וההשקעה הרבה של מיקרוסופט בה. הסביבה תומכת בהתקנה של תוספים שונים ומאפשרת אינטגרציה של פריימוורקים של בדיקות בה. הדיבגר של VS מאפשר חזרה אחורה במהלך ההרצה, פיצ'ר שימושי עבור מפתחים רבים.

מבחינת הבחירה ב-C# כשפה לפרויקט אוטומציה עולה השאלה מדוע לבחור ב-C# על פני שפות פופלאריות יותר ומגבילות פחות כג'אווה ופייתון. התשובה לכך אינה פשוטה. ראשית, יתכן ואתם עובדים בחברה שמרבית מוצריה מבוססים על הפריימוורק של מיקרוסופט ובמקרה זה יש הגיון רב ביישור קו עם שאר המפתחים בחברה כדי לקבל את התמיכה שלהם וכמובן את הגרסאות המלאות של התוכנות עליהם הם עובדים. פרויקטים רבים של אוטומציה יכולים להרוויח מגישה ישירה למערכת כגון שימוש באותם APIs שהמפתחים משתמשים בהם. כך שבמקרים רבים הבחירה בשפה תהיה הדבר ההגיוני ביותר לעשות.

השפה פחות חשובה כמו הפריימורק של הבדיקות בו אנחנו משתמשים

מבחינת פריימורק לבדיקות ב-C# קיימים NUnit ו-xUnit. השימוש בהם רחב בהקשר של פרויקטים אוטומטיים, למרות ששמם מרמז על עיקר תפקידם: בדיקות יחידה (unit testing). למרות זאת, NUnit תומכת ב- fixtures בדומה ל-pytest, דבר שעוזר בהקמה של תשתיות לפרויקט אוטומציה רחב היקף. בדיקה באתר stackoverflow של הצירופים C# + Selenium מתקבלות 6026 תוצאות של אשכולות בהם נשאלו שאלות בנושא. כמובן שנתון[4]

1 נכון לתאריך כתיבת חלק זה במאמר 5.10.20

2 נסו בעצמכם, ככל הנראה עד שתקראו את המאמר המספר יאמיר: https://stackoverflow.com/questions/tagged/python+selenium

3 נכון לתאריך כתיבת חלק זה במאמר 6.10.20

4 נסו בעצמכם, ככל הנראה עד שתקראו את המאמר המספר יאמיר: https://stackoverflow.com/questions/tagged/python+selenium

המדריך לאוטמציוניסט המתלבט, ראש בראש: Java, C# & Python

יאיר נסימוב |

זה מראה שהשאלות הנשאלות בנושא הן כרבע מהשאלות בפייתון וכשליש מהשאלות בג'אווה. עם זאת, זהו מספר לא מבוטל של שאלות, כך שאם אתם משתמשים בשפה כשפת האוטומציה שלכם התמיכה של קהילת המפתחים הבינלאומית בהחלט קיימת.

החסרונות העיקריים של שימוש ב-C# בולטים מאליהם: זוהי שפת תכנות שפותחה על ידי מיקרוסופט עבור מערכות ותוכנות המבוססות ווינדוס וככזו היא מגבילה אתכם לעולם הזה. התמיכה והשימוש בשפה קשורים בקשר הדוק ל-Net. עם היתרונות והמגרעות. כמו כן, הפריימוורק לבדיקות שמציעה השפה הוא לטעמי הכי פחות מפותח מבין שלושת האפשרויות. ניתן לעבוד איתו, אך הוא לא עשיר כ-TestNG ו-Py Test. הרצה של השפה בשרת בדיקות תדרוש שהוא יהיה מבוסס "חלונות[1]".

חסרון נוסף שמצאתי בשפה היא ה"דיבגר" שיש בשימוש ב-VS, שהוא לטעמי נופל משמעותית ביכולותיו מזה הניתן למצוא ב-IntelliJ ו- Pycharm.

דוחות – Reports

כמובן שמערכת דו"חות טובה, נעימה לעין ויציבה היא הדובדבן שבקצפת בכל פרויקט אוטומציה. כל מנהלת ומנהל ירצו לראות את הריפורט בסוף הריצה כדי לקבל תמונת מצב עדכנית על מצב הבדיקות והמערכת. ככל שהריפורט יהיה קריא ומפורט יותר כך תקבל האוטומציה שלנו ניקוד גבוה יותר אצל המנהלים שלנו.

אחד הדוחות הידועים והמפותחים מבחינה גרפית הוא Extent report. דוח זה מציג את הטסטים שלנו לפי קבוצות, סטטיסטיקות על זמן ריצה, גרפים ועוד. השימוש ב-Extent מוגבל לג'אווה ו-C# בלבד. מפתחי הפייתון יצטרכו לחפש אחר פתרון אחר.

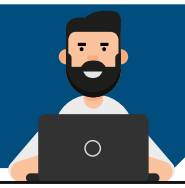
מערכת דוחות פופלארית אחרת היא Allure, שלמרבה המזל תומכת בכל שלוש השפות והיא מציגה דו"חות דומים במראם ל-Extent, אם כי מעט פחות אטרקטיביים לטעמי. ל-Allure מגוון איסוף צעדים אוטומטי ויכולת מובנית של היסטורית ריצות לכל טסט.

מערכות נוספות ראויות לציון: דוח TestNG - מערכת ברירת המחדל שאנחנו מקבלים עם שימוש בפריימוורק. מערכת זו מוגבלת ביכולותיה הגרפיות והפירוט של תוצאות הבדיקה מועט לעומת מערכות אחרות. מיותר לציין ששימוש ב-testNG דוח מוגבל לג'אווה. יש גם את המערכת של OpenSDK שמציגה ריפורט מעניין ומושך לעין ותומך בכל השפות. אחד מחסרונותיו הבולטים הוא הצורך בשימוש בשירותי הענן של OpenSDK בכדי לגשת לדוח. גם הדוחות של Azure DevOps של מייקרוסופט תומכים בכל שלושת השפות באופן מפתיע, אך כדי לייצר את הריפורט הזה יש צורך באינטגרציה מלאה של הפרויקט שלכם עם Visual Studio ומערכת ה-CI/CD של מיקרוסופט.

לסיכום

במאמר זה סקרנו את השימוש של כל אחת משפות התכנות המובילות להקמת פרויקט אוטומציה: Python & Java, C#. ראינו שלכל שפה יתרונות וחסרונות משלה. אחת המסקנות הבולטות מסקירה זו היא שברמה של הפעולות עצמן על המערכת, בין אם אתם משתמשים בסלניום או באפיום (Appium) השפה פחות חשובה כמו הפריימוורק של הבדיקות בו אנחנו משתמשים. ללא ספק TestNG הוא הפריימוורק הבשל ביותר מבין כולם והוא מציב את ג'אווה כדאית ביותר לשימוש בפרויקטי אוטומציה רחבי היקף, למרות המגבלות שלה כשפה. עם זאת, הבחירה בפייתון, הכוכבת העולה בתחום האוטומציה היא ללא ספק בחירה טובה שכן ראינו שיש תיעוד ותמיכה רחבה של הקהילה בשפה בהקשר של סלניום. השימוש בפייתון גובר מיום ליום וכך גם התמיכה ב-Py Test שכבר כיום יכולה לעשות דברים רבים שהיו שמורים ל-TestNG בלבד. לטעמי, ככל שהפרויקט שלכם רחב היקף יותר, כך כדאי לוותר על השימוש בפייתון לעומת ג'אווה, אך אם הצוות בחברה שלכם מורכב ממפתחי פייתון ותוכלו לקבל מהם תמיכה, תוכלו לייצר כל פרויקט גם באמצעות השפה הזו, גם אם הדבר יהיה כרוך בעבודה קשה יותר. ההמלצה לבחירה ב-C# תהיה הפחותה ביותר ממספר סיבות: התלות במוצרי מיקרוסופט, הקהילה התומכת קטנה יותר וכמובן הפריימוורק הפחות מפותח שיש לשפה להציע. אני סבור שאם אין לכם דרישה חד משמעית לעבוד ב-C# בפרויקט האוטומציה שלכם, כלומר אם אתם ממש מתממשקים עם רכיבי מערכת שכתבו המפתחים של התוכנה, כדאי לכם לחפש שפה ופתרון שונים. שכן, לא לשכוח שכאשר אנחנו מבצעים בדיקות אוטומטיות על השכבה הוויזואלית (UI) של התוכנה או האתר, שפת התכנות שבה כתובה שכבה זו פחות מהווה פקטור בשאלה באיזו שפה כדאי להשתמש בפרויקט אוטומציה ולכן ישנן אופציות אחרות טובות יותר.

1 ניתן להריץ C# על מק ולינוקס, אך הדבר לא יעבוד טוב וחלק כמו הרצה על מחשבי ווינדוס



סיכום תקופה

מבוא

לאחר ארבעה טורים (בגיליונות 20, 21, 22, 23), בהם כיסינו חלק נכבד מתחום בדיקות הנגישות, בחרתי לקחת הפסקה מכתיבת הטור. ברמה האישית זו גם הזדמנות טובה מבחינתי לקחת הפסקה מכתיבת הטור. הטור הנוכחי יתמקד בצעדים הבאים שתוכלו לקחת בכדי להרחיב את ידיעותיכם בתחום באופן עצמאי:

- נזכיר בכותרות את הנושאים שנידונו בטורים הקודמים
- נפרט שוב את הכלים שהוזכרו
- נתן קישורים ללמידה עצמאית להרחבת הידע בתחום

בדיקות נגישות

בטורים הקודמים דיברנו על:

- ניגודיות צבעים
- מבנה כותרות
- טקסט חלופי לתמונות
- בדיקות מקלדת
- בדיקות עם קורא מסך

כלים

הזכרנו בטורים הקודמים את הכלים הבאים.

כלי בדיקות ניגודיות צבעים:

- WebAIM's Contrast Checker
- Colour Contrast Analyser
- Button Contrast Checker
- Chrome Dev Tools
- aXe Developer Tools (aXe for Firefox, aXe for Chrome)

כלי בדיקת מבנה כותרות:

- headingsMap (headingsMap for Chrome, headingsMap for Firefox)
- Accessibility Insights for Web for Chrome
- ARC ToolKit for Chrome

טקסט חלופי לתמונות:

- ARC ToolKit for Chrome
- aXe Developer Tools (aXe for Firefox, aXe for Chrome)

מה הלאה?

הטורים היו מבוא ראשוני, אבל עיקר הלמידה נעשית מתוך סקרנות ורצון אמיתי להביא את המוצרים שאנחנו עובדים עליהם לרמת נגישות גבוהה. על מנת להעמיק את הידע, אני ממליץ על שלוש דרכים:

1. קריאה עצמאית במסמכי הנחיות WCAG ושיטות כתיבת ARIA.
2. רישום לקורסים מקוונים.
3. התעדכנות שוטפת בקהילה: במיילים, בטוויטר ובסלאק.

הנחיות WCAG ושיטות כתיבת ARIA

אמנם במבט ראשון זה לא נראה מסמך ידידותי במיוחד, אבל כשמקדישים לו זמן, וקוראים ביסודיות קריטריון אחר קריטריון, מגלים שהוא די בהיר. כיוון שזהו הבסיס לשיח אודות נגישות באינטרנט בכלל, ולדרישות החוקיות בפרט, מומלץ מאוד להכיר אותו לעומק. הגרסה המומלצת רשמית של W3C היא WCAG2.1 (על אף שהחוק הישראלי עדיין מתבסס על WCAG2.0). מומלץ במיוחד להכיר את הקריטריונים ברמות A ו AA.

מסמך נוסף מומלץ, שגם אותו הזכרנו כבר בטורים קודמים, הוא שיטות הכתיבה של WAI-ARIA גרסה 1.1 המפרט את דרכי השימוש בכללי ARIA להעשרת הנגישות של אפליקציות רשת. שימו לב במיוחד לאזהרות: אין להשתמש ב ARIA כשלא צריך ובכל מקרה "No ARIA is better than Bad ARIA".

WCAG גרסה 2.1:

https://www.w3.org/TR/WCAG21/

WCAG גרסה 2.2 (טיוטא):

https://www.w3.org/TR/WCAG22/

וסיכום קצר לגבי הסעיפים החדשים ב2.2:

https://www.deque.com/blog/what-to-expect-from-wcag-2-2/

שיטות כתיבה של WAI-ARIA גרסה 1.1

https://www.w3.org/TR/wai-aria-practices-1.1

קורסים מקוונים

ישנם מגוון של קורסים מקוונים שאפשר לעשות, אני רוצה להמליץ על שניים מהם. הראשון הוא של W3C ואני אוהב את העומק שלו, ואת המקום הנרחב שהוא מקדיש להיכרות עם אנשים עם מוגבלויות שונות. בסופו של דבר נגישות זה לא רק למלא צ'קליסט של דרישות. נגישות היא הבנת צרכי המשתמשים, ומתוך הבנה זו לזהות את המקומות שעלולים ליצור קשיים עבורם.

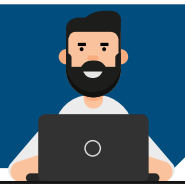
קורס מומלץ שני הוא של רוב דודסון ואליס בוקסהול מגוגל. ולמי שרוצה, יש לו גם מקבילה טקסטואלית. זהו גם קורס מומלץ מאוד, ולמי שקצרים בזמן, אני מצרף גם סדרת סרטונים מקבילה שרוב דודסון הכין, שמכסה תוכן דומה, רק במנות קטנות וממוקדות.

אלון פרידמן ויסברד

אחראי בדיקות בצוות הנגישות של Wix.

בעל רקע של 15 שנים בבדיקות תוכנה.

נשוי + ילד + חתולה.



קורס מקיף של W3C:
https://www.edx.org/course/web-accessibility-introduction

קורס מבוא של חבר'ה מגוגל
https://www.udacity.com/course/web-accessibility--ud891

גרסה טקסטואלית של תכני אותו הקורס:
https://developers.google.com/web/fundamentals/accessibility/

סדרת הסרטונים של רוב דודסון (a11y casts)
https://www.youtube.com/playlist?list=PL7Bjl0Cb4SboBHNihVBRd-AdctfXcmClc

קהילה

מעבר למשאבים הנ"ל, הדרך הטובה ביותר להישאר בעניינים, לשמוע על עדכונים ולהשתלב בדיונים היא פשוט להצטרף לקהילת אנשי הנגישות ברשת. מספר דרכים מומלצות הן: להצטרף לסלאק, לעקוב חשבונות מובילים בתחום בטוויטר, ולהירשם לניוזלטרים.

סלאק:

https://web-a11y.slack.com/

רשימת מעקב בטוויטר:
https://twitter.com/i/lists/1325583302615556096

ניוזלטר שבועי a11y weekly:

https://a11yweekly.com/

ניוזלטר חודשי של WebAIM:

https://webaim.org/newsletter/

לסיכום

קשה מאוד לכסות את כל עולם הנגישות בכמה טורים, ולא הייתה לי יומרה כזו מעולם. הטורים פתחו פתח כניסה לעולם הזה, דרך כמה מהנושאים המרכזיים. מצד אחד זה היה רק קצה הקרחון, ומצד השני זהו בסיס איתן על מנת להמשיך ממנו לבד. כולי תקווה שההפניות בטור זה ישמשו אתכם ושתיקחו את הנושא רחוק ככל שתחפצו.

לתגובות, שאלות או סתם שיתוף בהתקדמות, מוזמנים לכתוב לי בטוויטר, שם משתמש:

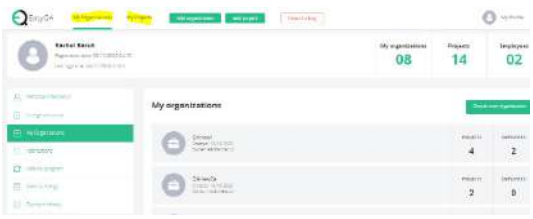
@awaisbard

בברכה, אלון.

כלי לבדיקות EasyQA | רחל ברוך

בחיפושי אחר כלי לניהול בדיקות חינמי לחברות סטארט-אפ מצאתי את EasyQA אשר נותן מענה לדרישות רוב החברות, בין אם זו חברה גדולה או עבור חברות סטארט-אפ קטנות, EasyQA הינה בגרסת בטא ניתנת ללא תשלום.

הכרות



הוא כלי ניהול בדיקות נוח ופונקציונלי לשימוש עבור כל תהליך הבדיקות וקל להבנה עבור צוות הפרויקטים – ממנתח הנתונים (Data analyst), מפתח ועד מעצב האתר (וגם כתב טכני). הכלי מאפשר להריץ עד חמישים פרויקטים בו זמנית. ניתן להגדיר חמישים ארגונים ובתוכם מספר רב של פרויקטים.

תכונה חשובה של הכלי הוא שהכלי חוסך זמן, מכיוון שקיימים בו כל הכלים הנחוצים לפיתוח המוצר כגון: מעקב אחר באגים, מערכת בקרת גרסאות, מאגר קוד, כלי מסמך תכנון בדיקות, כלי זיהוי בעיות ושיתוף אובייקטים לבדיקה בין חברי הצוות. הכלי גם מכיל את כל התכונות הנדרשות על מנת לספק תוצאות טובות לדיווח. ניתן להזמין חברי צוות ולהגדיר להם תפקידים, לנהל פרויקטים בפלטפורמת המוצר, לייצר תכנון בדיקות ומקרי בדיקות שניתן להגדיר להם מבנה עם פרמטרים שימושיים, ביצוע ריצות בדיקה, מעקב אחרי באגים, לוח אג'ילי פשוט המכיל מעט עמודות ופרמטרים, SDK המאפשר זיהוי קריסות ושליחתם ללוח מחוונים עם אפשרות לצילומי מסך, הקלטת וידאו ודיווח באגים ממכשיר סלולר עם אפשרות לשיתוף אובייקט בדיקה ללא צורך בשירותים חיצוניים. הכלי מאפשר למשתמש לעבוד בין פלטפורמות שונות.

התכונות המרכזיות של המערכת הם:

1. המערכת - יצירת מספר רב של פרויקטים במספר רב ארגונים למשתמש
2. אובייקט הבדיקה
3. תכנון בדיקות
4. מקרה בדיקות
5. EasyQA SDK ליישומוני אנדרואיד ואייפון
6. התממשקות עם מערכת ניהול תקלות
7. לוח אג'ילי (Agile) ידידותי למשתמש
8. מעקב ודיווח תקלות
9. תהליך זיהוי קריסה ליישומוני אנדרואיד ואייפון
10. ביצוע ריצות בדיקה
11. דוחות

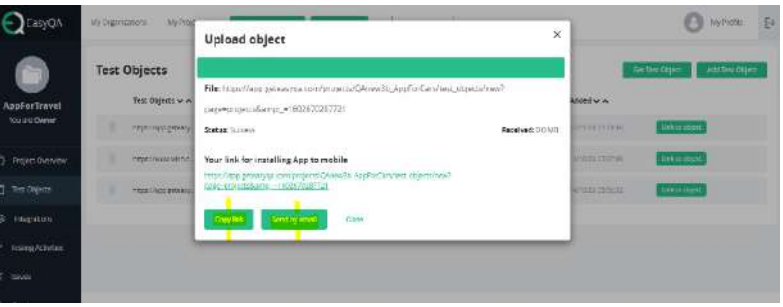
תכונה חשובה של הכלי הוא שהכלי חוסך זמן, מכיוון שקיימים בו כל הכלים הנחוצים לפיתוח המוצר

אובייקט הבדיקה

ניתן לתאר את אובייקט הבדיקה עם קבצי התקנה של אנדרואיד או אייפון (קובץ ipa או apk) וקישור לאתר האינטרנט.

ניתן להעלות את הקבצים מ-GitHub או GitLab. ניתן להתקין ולהריץ את קבצי ipa או apk במכשיר לאחר קבלת קישור מאובייקט הבדיקה.

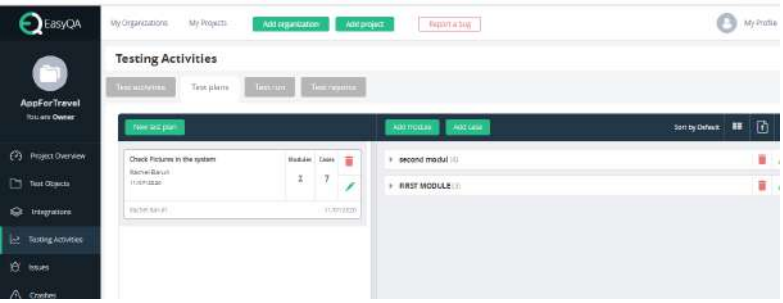
לאחר העלאת קישור/קובץ, הכלי מאפשר לבחור אותו בעת יצירת תקלה ולשלב את ה-SDK כדי לתפוס את התרסקויות (קריסות).



תכנון הבדיקה (Test Plan)

הכלי מאפשר לבצע תכנון בדיקות של כל התהליכים כגון: יצירת תכנון בדיקות, פיצול לתבניות, יצירת מקרי בדיקות, והכנת רשימות צ'ק-ליסט. כל חברי הפרויקט, למעט מפתחים או משתמשי קצה יכולים ליצור תכנון בדיקה.

משתמשים יכולים ליצור ולהריץ מספר תכנוני בדיקה בו זמנית. גם אם קיימים תכנוני בדיקה נפרדים לכל אחת: יישומי אנדרואיד, אייפון ודפדפן, או תכנון בדיקה משותפת אחת לכולם. לאחר יצירת תכנון בדיקה, ניתן לתכנן את המבנה על ידי יצירת מה שנקרא מודולים או תבניות. ניתן להציג מבנה של תכנון הבדיקה בקלות.



המערכת – פרויקטים וארגונים

ניתן להגדיר בכלי פרויקטים לאחר הגדרת שם ארגון לאותו משתמש. מי שמגדיר את הארגון הופך להיות הבעלים של הארגון והוא יכול להזמין חברים צוות מארגונים אחרים. ניתן לעבוד במספר ארגונים בו זמנית וגם במספר פרויקטים. הכלי מאפשר עבודה משותפת בין חברים, להקצות תפקידים ולהגדיר להם הרשאות על ידי בעלי ארגון או מנהלים. הפורמט של הוספת חברים מאפשר הוספת משתמשים רבים בו זמנית. בארגון יכולים להיות 3 תפקידים: בעלי ארגון, מנהל ומשתמש. המשתמש יכול להיות בודק תוכנה, מנהל פרויקט, מפתח או משתמש קצה.

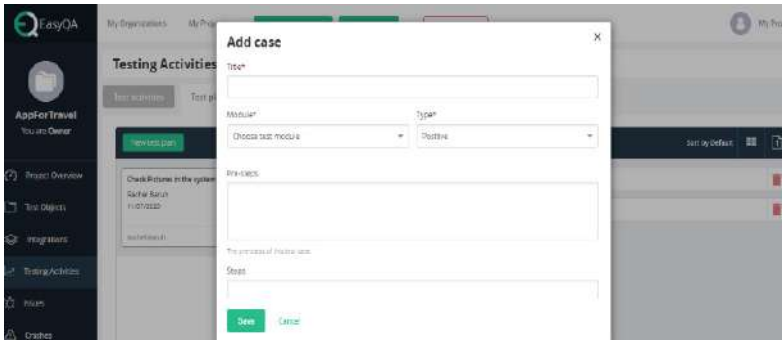
רחל ברוך

הנדסאית תוכנה, לפני 3 שנים לאחר הפסקה של שנים מעולם ההייטק חזרה לעולם התוכנה בכל הכוח. נהנית להיות בצד הבודק עם חשיבה של מפתחת. אין יום שהיא לא לומדת משהו חדש בעולם התוכנה.

כלי לבדיקות EasyQA | רחל ברוך

מקרי בדיקות (Test Cases)

על ידי הזנת מקרה הבדיקות, ניתן לבחור תבנית של תכנון הבדיקה, סוג של מקרי הבדיקה ותיאור צעדים מוקדמים, ותוצאות צפויות. ניתן לגרור ולשחרר או להעתיק מקרי בדיקה מתבנית אחד למשנהו אחד אחד או בבת אחת.



EasyQA SDK יישומוני אנדרואיד ואייפון

ה-EasyQA SDK הוא כלי שפותח על ידי ThinkMobiles, תואם ליישומוני אנדרואיד או אייפון.

ניתן להקליט כל קריסה שעלולה להתרחש בתהליך בדיקת היישומון ולצלם צילום מסך, להקליט וידאו, לתאר את השלבים ולשלוח הכל לדף הבעיות (issues עם SDK).

אם אובייקט הבדיקה הוא אתר אינטרנט או יישום אינטרנטי כלשהו, ניתן להשתמש ב- EasyQA Chrome Extension כדי לדווח על באגים עם צילומי מסך וסרטונים ישירות מהאתר שנבדק לדף הבעיות בתוך הפרויקט.

התממשקות עם מערכת לניהול תקלות

כל תהליך ההתממשקות של הכלי מתחיל ביצירת תוספים. לשם כך ניתן לפתוח פרויקט, למצוא את הדף וללחוץ על כרטיסיית התוספים. תהליך ההתממשקות קל ואינטואיטיבי.

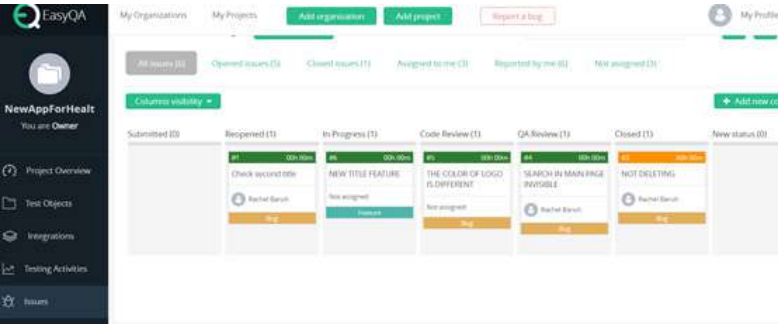
YouTrack		Trello	
Jira		GitHub	
RedMine		GitLab	
Pivotal Tracker		Jenkins	

לוח אג'ילי (Agile) ידידותי למשתמש

לוח זה מכיל מספר עמודות ופרמטרים ליצירת כל יישום.

ניתן לשנות מטלות בקלות על ידי גרירה ושחרור בעמודות. כל עמודה מראה את מצב הפרויקט כגון משימות שהושלמו בהצלחה ועוברות לעמודה סגורה. שימושי וקל!

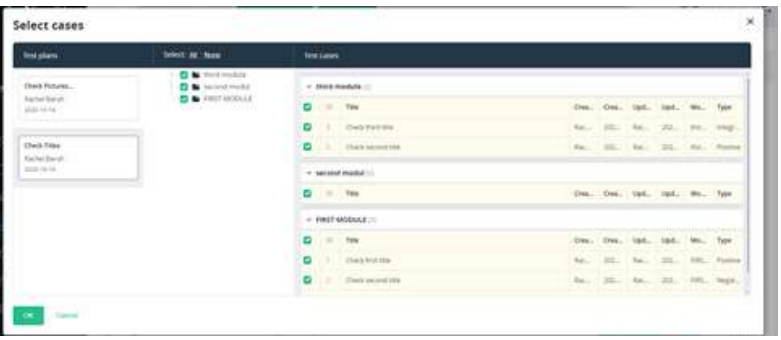
כמו כן, המשתמש יכול להתאים אישית את הלוחות והעמודות.

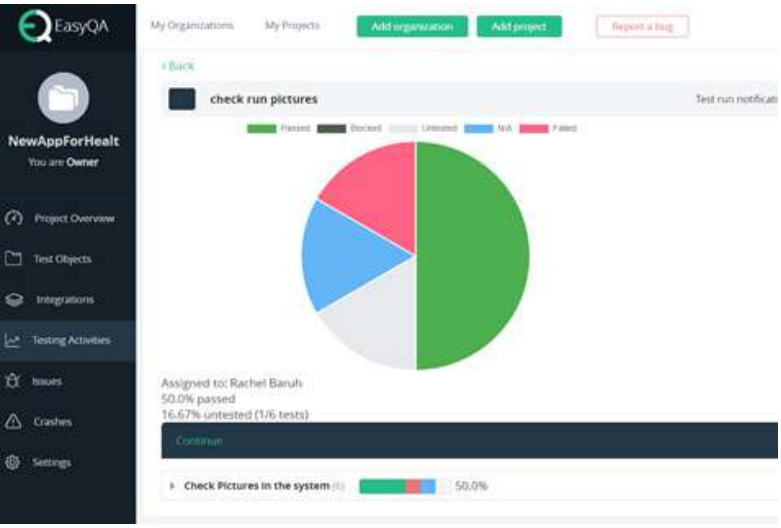


דיווח תקלות ביישמוני סלולר (אנדרואיד/ אייפון/בתוסף כרום) על ידי ניעור הטלפון

ביצוע ריצות בדיקה

ניתן לבצע ריצות על ידי בחירת תכנון בדיקה והוספת מקרה בדיקה מרשימה. חשוב לציין שרק תכנוני בדיקות שהוגדרו עם מקרי בדיקות יופיעו ברשימה. ניתן לבחור מרשימה נוספת את מקרי הבדיקות הספציפיות עבור אותו תכנון בדיקה. המערכת תיידע בודקים על הפעולה בעזרת מייל. תוצאות הריצות יהיו עבר, נכשל או חסום. לאחר סיום הריצות ניתן לקבל את דיאגרמת הסטטיסטיקה.

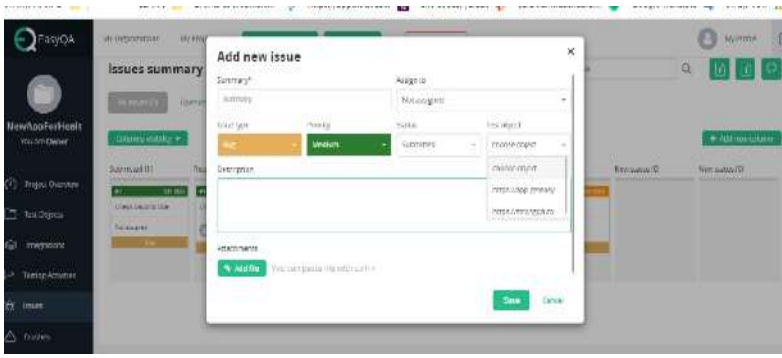




כלי לבדיקות EasyQA | רחל ברוך

מעקב ודיווח תקלות

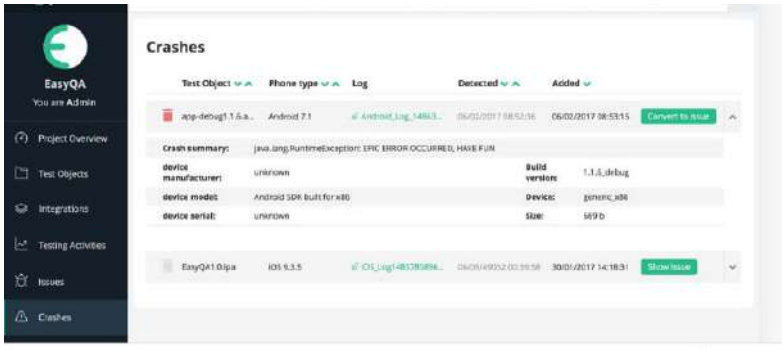
תכונה חשובה נוספת היא שיטת מעקב ודיווח אחר תקלות. בדף תקלות (issues) ניתן להגדיר ולעקוב אחר כל המתרחש במקום אחת לפי אובייקט הבדיקה. ניתן לבחור את סוגי הכרטיסים כגון: תיעוד, משימה, תכונה, באג או קריסה, ולקבוע את חומרתו, תיעדופו, שלבים וצירוף קבצים. ניתן להשתמש בקובץ CSV לדיווח על באגים, ניתן לייבא או לייצא את כל הכרטיסים מדף הבעיות.

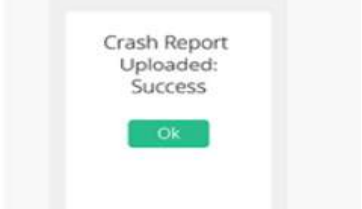


תהליך זיהוי קריסה ביישומי אנדרואיד ואייפון

הכלי מאפשר לתפוס קריסות ביישומוני אנדרואיד ואייפון. התהליך מתבצע על ידי שליחת קריסות לעמוד הקריסה שבשרת כך שאין צורך לשחזר את הקריסות. המידע מוצג בעמוד קריסות שבשרת עם סיבת ההתרסקות וכל הנתונים נמצאים על המכשיר עליו אירעה ההתרסקות ובקובץ הלוג.

כדי לתפוס קריסות ביישומי האנדרואיד והאייפון צריכים להתממשק בין ה-EasyQA SDK לקוד היישום על ידי חיבור לדף ההתממשקות בכלי. כדי לחבר בין ה-SDK לפרויקט צריך להשתמש עם הטוקן שהוגדר.

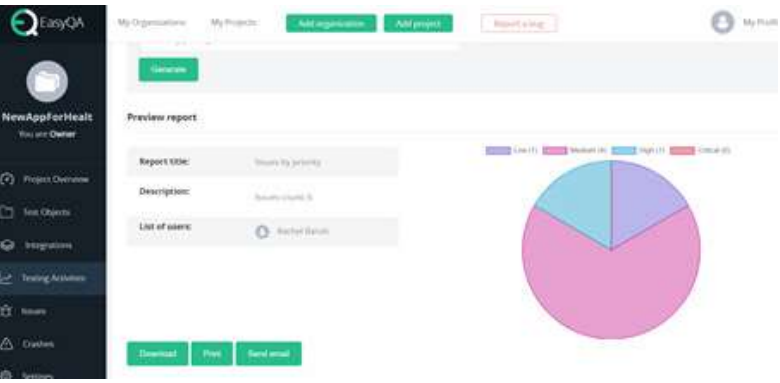




דוחות

החברה מקדישה תשומת לב מיוחדת לנושא הדוחות. ניתן ליצור דוחות למקרי בדיקות, ריצות בדיקות, נושאים לפי סטטוס, עדיפות או סוג, קריסות לפי גרסה. ניתן להציג מיד את התרשים, להוריד ולהדפיס. הכלי מאפשר לצוות לעקוב אחר תוצאות עבודתו ולדווח עליהם ללקוחות בצורה תמציתית וקריאה. כאשר כל הדוחות נאספים במקום אחד ניתן לגשת אליהם על ידי לחיצה על כפתור אחד בלבד. קיימים מספר סוגי דיאגרמות העוזרות לעקוב אחר העבודה של חברי הצוות.

גישה נוחה לדוחות על ידי לחיצה על כפתור אחד



חסרונות

- המחיר הוא לפי מספר משתמשים בגרסה רגילה.
- ניתן להקליט וידאו עם SDK משולב רק באנדרואיד 5
- התאמה אישית אינה זמינה עדיין, כלומר משתמשים אינם יכולים ליצור סקריפטים משלהם. (החברה אוספת את כל המשוב מהמשתמשים על מנת להשתפר).

יתרונות

- הכלי בגרסת בטא אינו כרוך בתשלום, כולל השירות מהחברה וזה בכדי לתמוך בחברות סטארט-אפ קטנות.
- הכלי מאפשר לצוות לעקוב אחר תוצאות עבודתו ולדווח עליהם ללקוחות בצורה נוחה ונגישה.
- קיימות מספר דיאגרמות העוזרות לעקוב אחר העבודה של חברי הצוות.
- גישה נוחה לדוחות על ידי לחיצה על כפתור אחד.
- התקנת יישומון בקלות על ידי שליחת קישור של אובייקט בדיקה לחברי צוות הפרויקט.
- דיווח תקלות ביישמוני סלולר (אנדרואיד/אייפון/בתוסף כרום) על ידי ניעור הטלפון.

לסיכום

התנסתי בכלי במשך 30 יום ללא עלות והייתי מאוד מרוצה לקבל 9 סוגי דוחות. ממשק משתמש קל לשימוש (הכל מוגדר מראש). המחיר יקר לדעתי. עם זאת, אם אתם עובדים בחברת סטארט-אפ קטנה אז ישתלם לכם להשתמש גרסת הבטא ללא עלות.

ציון:

מענה לצרכים של החברה: 8.5/10

נותן למשתמש חווית שימוש 8/10

תמיכה וקהילה : 4/10

סה"כ: 6.8

בקרה על תוצרי תהליכי העבודה של אנשי הבדיקות | רפי ברי

כמנהלים בכלל, וכמנהלי בדיקות בפרט, חשוב שנבחן מדי פעם את תהליכי העבודה הקיימים ואת התוצרים שלהם. האם התהליכים שהחלטנו עליהם, בין אם מדובר בתהליכי בדיקות בלבד, או בתהליכים שאנו לוקחים חלק בהם, כמו תהליכים ברמת הפרויקט, באמת משפרים את התפוקה שלנו ואת איכות התוצרים, או לחלופין, המצב נשאר סטטי, או אפילו אולי ירד?

האם אנו מוצאים את עצמנו לא פעם, בפיגור בתוכניות העבודה, עקב אין סוף "הפתעות" ועיכובים, שלא נלקחו בחשבון מלכתחילה, ושאולי יכולנו למנוע עם היערכות אחרת?

כיום רוב כלי הבדיקות הקיימים מאפשרים למשתמשים ולמנהלים להוציא דוחות, גרפים ונתונים סטטיסטיים כמעט על כל פריט מידע שקיים במערכת.

מעבר להוצאת הנתונים עצמם, חשוב לנסות להבין את המשמעויות שלהם – האם נתון כזה או אחר מצביע על בעיה מסוימת בתהליך העבודה? ואם כן, מה עלינו לעשות כמנהלים כדי לשפר אותו? או אולי, נקבל אינדיקציה על נקודת חוזק שכדאי לשמר.

במאמר זה, אציע מספר דוגמאות לכמה נקודות מרכזיות / מדדים לבחינה, בהיבט של תהליכי בדיקות התוכנה, שלדעתי יתקיימו במרבית הפרויקטים, אפרט סיבות וגורמים אפשריים לכל נקודה, והמלצות לטיפול בכל מדד כזה.

כמובן שעל מנת להשיג את הנתונים הרצויים, יש לדאוג תחילה ליצירה, בהתאמה אישית, של השדות והערכים הרלוונטים במערכת, ולא פחות חשוב, לדאוג שאנשי הצוות ימלאו אותם באופן שוטף, כחלק אינטגרלי מעבודתם.

מדד סך התקלות שנמצאו אצל לקוחות

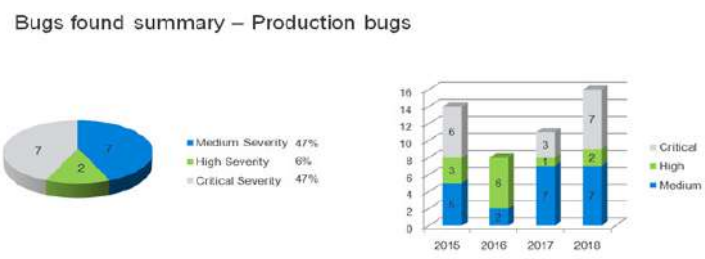
ידוע שבמסגרת האחריות של צוות הבדיקות, אחד מתפקידיו העיקריים זה להוריד למינימום את האפשרות שתקלות יופיעו אצל הלקוחות. זה לא באמת מעניין אף אחד אם במהלך הבדיקות התגלו 500 תקלות, או 2000 תקלות, יציבות המערכת שחווים הלקוחות היא מטרת הארגון כשהוא מעסיק צוות בדיקות, ומספר התקלות שדווחו על ידי לקוחות הוא מדד שמאפשר להעריך את השגת המטרה הזו.

> **כמנהלים בכלל, וכמנהלי בדיקות בפרט, חשוב שנבחן מדי פעם את תהליכי העבודה הקיימים ואת התוצרים שלהם**

מדד זה לא אפקטיבי כשהוא מציג נקודת מידע על תקופה אחת ספציפית. הוא חייב להיבחן לאורך זמן, כדי שנבין מה היא המגמה ואם מספר התלונות עולה.

נניח שעם הכניסה לתפקידכם כמנהלי בדיקות, היו בשנה X דיווחי תקלות מלקוחות, כאשר מחציתן תקלות בחומרה מאוד גבוהה שחייבו טיפול מיידי. ולאחר שנה או שנתיים, כמות התקלות החמורות שדווחו, הכפילה את עצמה. אז יש סיכוי סביר שמשהו בתהליכי העבודה לא נעשה כמו שצריך, ומחייב בחינה וניתוח הגורמים לכך.

או בדוגמא אחרת, כמות התקלות אצל הלקוחות הוכפלה, אבל 90% מהן כעת תקלות בחומרה נמוכה. האם זה טוב? התשובה כמובן סובייקטיבית, בהתאם ליעדים שאליהם אתם שואפים.



איור 1

עליכם להגדיר יעדים (כמובן סבירים לביצוע), ולכוון אליהם. לא הגעתם, כנראה שהתהליכים הקיימים עדיין לא עובדים מספיק טוב, ויש לבחון דרך אחרת. הצלחתם להגיע ליעד? כל הכבוד, עכשיו תחליטו אם טוב לכם "לשמר" את המצב, או לחשוב על עוד אפשרויות לשיפור התהליכים.

המלצה שלי – עבור כל תקלה רצינית שהתגלתה אצל לקוח, מעבר לבדיקת התיקון עצמו, נסו להבין מדוע היא התרחשה מלכתחילה, מדוע לא הצליחו לגלות אותה בזמן הבדיקות, וכיצד ניתן למנוע את התרחשותה (או דומות לה) שנית בעתיד.

כאשר לאורך הזמן, אתם רואים בסופו של דבר מגמת שיפור, ומצליחים להתכנס ליעדים שקבעתם, אז כנראה שתהליכי העבודה שהגדרתם אכן משיגים את מטרותיהם.

ניתן לראות בדוגמא (איור 1) מצב המצביע על מגמה שלילית, ועל עלייה בדיווחי תקלות מהשטח, כאשר בשנה האחרונה, יותר ממחצית התקלות שדווחו משם, היו בחומרה גבוהה, מצב שמחייב בחינה מחדש של תהליכי העבודה הקיימים, וזיהוי הנקודות הבעייתיות הגורמות למצב זה.

מדד תקלות שנמצאו ונסגרו במסגרת הבדיקות, לפי חומרתן

כפי שציינתי במדד הקודם, פחות רלוונטי המידע של כמה תקלות נפתחו במהלך הבדיקות, אבל כן יש משמעות לדעתי להתפלגות של חומרת התקלות שנמצאו, ולאחוז התקלות שטופלו ונסגרו בפועל לאחר וידוא תיקון.

כאשר בוחנים את התפלגות חומרת התקלות ומגלים שרוב התקלות שנמצאו בבדיקות הם תקלות בחומרה גבוהה, ניתן להסיק מכך שאיכות הגרסאות הנמסרות לבדיקה אינה גבוהה. זה יכול להצביע על כך שהמפתחים לא מבצעים מספיק בדיקות יחידה לקוד שלהם, או לא מבצעים בדיקות אינטגרציה בסיסיות עם קוד של מפתחים אחרים, או אולי אף לא הבינו מספיק לעומק את הדרישה העיסקית של אותו מוצר אותו הם מפתחים..

אם זה הכיוון, אז כמובן מומלץ לחקור ביחד עם מנהל הפיתוח את הנושא. אולי יש מקום להגדיר סט בדיקות שפיות שיורץ ע"י המפתחים או הבודקים, ושיהיה חייב לעבור, טרם הגשת הבילד או הגירסה לבדיקות.

מצד שני, אם רוב התקלות המתגלות הן בחומרה בינונית ונמוכה (וכמובן הפונקציונליות העיקרית עובדת), הייתי לוקח זאת למקום חיובי, המצביע על כך שמצד אחד הקוד העיקרי שפותח משיג את מטרתו, ומצד שני הבודקים הציפו מגוון רחב של תקלות, בהיבטים שלא נלקחו בחשבון מצד המפתחים, ובכך יחדיו השיגו את מטרתם.

גם לאחוז התקלות שנסגרות יש משמעות – האינטרס שלנו כבודקים שמקסימום תקלות שנפתחו במהלך בדיקות, יתוקנו ויסגרו בסופו של דבר, גם אם יקח עוד גירסה אחת או שתיים בעתיד. ומדוע?

מהסיבה השכיחה ביותר שרובנו נתקלנו בה – מה שזמני הופך לקבוע. כשאומרים לנו נטפל בזה בהמשך, בסבירות גבוהה אנו יודעים שזה לא יקרה, כי תמיד נכנסות תכולות חדשות לבדיקה, ויש דברים דחופים יותר. אבל מצד שני – בשביל זה נקבע ערך התעדוף לכל תקלה, ויש תקלות "שניתן לחיות איתן", ויש כאלה שפחות.

לגבי תקלות בתעדוף נמוך יותר, גם אותן עלינו לנסות לדחוף לסגירתן בשלב כזה או אחר, וזאת על מנת למנוע מצב של:

כפילויות של פתיחת תקלות – אם תקלה נפתחה לפני זמן רב, וטרם תוקנה, בסבירות גבוהה, בודק אחר, בגירסה אחרת, יעלה על אותה תקלה, לא ימצא תיעוד שלה (או לא יחפש), ואז יפתח תקלה כפולה,

רפי ברי

בעל נסיון של יותר מ-16 שנים כבודק ומנהל בדיקות במגוון חברות וטכנולוגיות ובמודולי פיתוח שונים.

ויבזבז זמן לכולם.

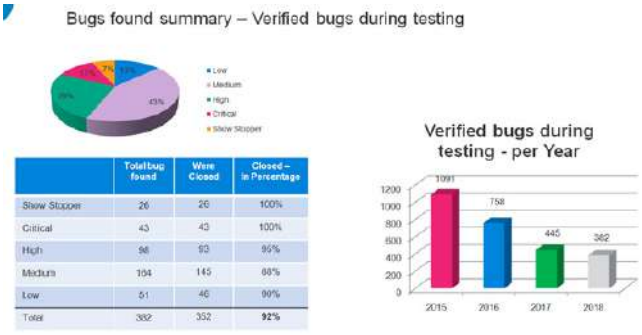
שנית, גם תקלות בחומרה נמוכה יותר, לעיתים יכולות לגרום לכישלון תסריטים אוטומטיים שכבר כתובים (ויש הבנה שהוחלט לא לתקן את התקלה בשלב זה, או בזמן הקרוב), מצריך בלא מעט פעמים לביצוע "מעקפים זמניים" בקוד, ע"מ להתעלם מהתקלה הידועה, דבר שכאמור מצריך עוד משאבים.

כמובן שככל שחומרת התקלה עולה, אחוז התקלות שצריכות להיסגר לאותה חומרה הוא בהתאם, וחייב גם לעלות. כמובן, תקלות שהוגדרו כ-Show Stopper הן כאלה שמחייבות 100% סגירה.

המלצתי – עקבו אחר אחוזי הסגירה של התקלות, תקבעו מה הוא אותו אחוז שמקובל עליכם, ובמידה ואתם לא מתכנסים אליו, דברו עם הפיתוח / מוצר, ונסו לראות איך בכל זאת ניתן לתת להם "יחס", מבחינת תיעדוף.

אני הייתי ממליץ לכוון לסגירה ממוצעת של כ-90% מסך כל התקלות שנמצאו. מניסיוני, זאת כמות ריאלית שמיטבה עם התנהלות הפרויקט.

להלן דוגמא לכמות תקלות (מאומתות) שנמצאו בשנה מסוימת לפי חומרתן ואחוז הסגירה שלהם באותה תקופה:



	Total bug found	Were Closed	Closed – in Percentage
Show Stopper	26	26	100%
Critical	43	43	100%
High	98	93	95%
Medium	104	145	88%
Low	51	46	90%
Total	382	352	92%

מדד יחס תקלות הנסיגה (Regression) מכלל התקלות שנמצאו בבדיקות

גם פה אני חוזר ומדגיש - פחות רלוונטי המידע של כמה תקלות נפתחו במהלך הבדיקות, אבל כן יש משמעות לדעתי לאחוז התקלות שנמצאו במסגרת הרצת הרגרסיות, מסך כל התקלות שנמצאו.

כולנו יודעים שמטרת בדיקות הרגרסיה היא לוודא שתכולות ופונקציונליות קיימת, מגירסאות קודמות, לא נהרסו עקב כניסתם של פיתוחים חדשים, או תיקונים של תקלות ספציפיות בגירסה הנבדקת.

אז נשאלת השאלה – מה נחשב אחוז טוב של תקלות שהתגלו ברגרסיה? האם אחוז גבוה או נמוך? על פניו, מטרתינו העליונה כבודקים, היא למצוא מקסימום תקלות לפני עליה לאוויר. אז כמובן אחוז גבוה של תקלות שהתגלו ברגרסיה, יכול להצביע שכנראה עשינו את עבודתינו נאמנה בבדיקות. אך חשוב מזה, זה גם יכול להצביע שמשהו לא טוב קרה בצד הפיתוח, ולכניסתו של הקוד החדש היו השלכות לא טובות.

יתכן שהמפתחים "חיפפו", ולא בדקו את עצמם כפי שנדרש מהם, או לא הבינו לעומק את האימפקט של הקוד שלהם במקומות אחרים במערכת. כמובן במצב כזה, הדבר מחייב בחינה מעמיקה יותר עם הפיתוח, היות וידוע שככל שיש יותר תקלות רגרסיה, זה מתורגם ישירות ליותר זמן הדרוש לפיתוח, יותר זמן הדרוש לבדיקות, ועוד פוטנציאל לעיכובים בלוחות הזמנים.

בחלק מהחברות ניתן גם להבחין בין תקלות רגרסיה שהתגלו במסגרת בדיקות ידניות, ותקלות שהתגלו במסגרת בדיקות אוטומטיות.

לתפיסתי, כאשר התקלות רגרסיה המתגלות דווקא ע"י הבדיקות האוטומטיות, זה אף יותר חמור, מכיוון שבד"כ יותר משאבים מושקעים בתסריטים אוטומטיים ביחס לבדיקות ידניות, ונהוג להכניס תחת אוטומציה פיתוח או פונקציונליות שנתפסים כיציבים, ללא צפי

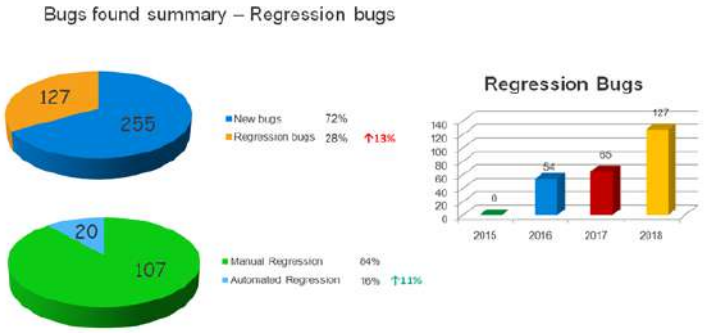
גם תקלות בחומרה נמוכה יותר, לעיתים יכולות לגרום לכישלון תסריטים אוטומטיים שכבר כתובים

לשינויים מהותיים, שמורצים על מנת "לסגור פינה", ולחסוך "עבודה שחורה" ומונוטונית לבודק הידני. אז היה ואם יש לא מעט תקלות רגרסיה מהרצות אוטומטיות, כאמור, לתפיסתי משהו לא טוב קרה עם תהליכי הפיתוח, עד כדי השפעה ממשית אפילו על תכולות שנתפסות 'יציבות'.

אני נוטה להעריך שכלל תקלות הרגרסיה שנמצאו במהלך הבדיקות, ידני או אוטומטי, מכלל התקלות שנמצאו בגירסה / תקופה, לא צריך לעלות על 15%.

להלן דוגמא המפרטת את אחוז התקלות שנמצאו – תקלות על תכולות חדשות, ותקלות רגרסיה על תכולות קיימות, ומתוך הרגרסיה, את החלוקה שנמצאו בין ידני לאוטומטי.

ניתן לראות כי בשנה האחרונה, הייתה עליה חדה באחוז תקלות הרגרסיה (באדום) ביחס לשנה קודמת, דבר שכמובן מחייב בחינה, מדוע זה קרה.



מדד סך תקלות השווא שנפתחו בתקופה מסוימת

כפי שכולכם חווים זאת, כמעט אף איש צוות פיתוח לא מקבל בהבנה יתרה את העובדה שבודק "מבזבז" לו זמן בלחקור תקלה שנפתחה, רק כדי להבין בסוף, שאין באמת תקלה על הפרק. ובצדק. מן הסתם, לא חסר לו/ לה משימות דחופות אחרות, שעליו/ה לעסוק בהם כעת. אותו דבר אנו מרגישים כבודקים, כאשר אנו נאלצים לבדוק לדוגמא את אותו תיקון שוב ושוב (אתייחס לכך בדוגמא הבאה).

אני הייתי ממליץ לכוון לסגירה ממוצעת של כ-90% מסך כל התקלות שנמצאו

בקרה על תוצרי תהליכי העבודה של אנשי הבדיקות | רפי ברי

כאשר אחוז לא מבוטל מהתקלות שנפתחות במערכת, הם תקלות שווא (לרוב מוגדר כ- **Rejected**), אפשר להבין שבמצטבר, באמת נוצר מצב של לא מעט זמן עבודה שהיה ניתן לחסוך לטובת משימות אחרות. והשאלה שכמובן עולה, מה יכולות להיות הסיבות למצב.

מניסיוני ניתן לחלק את הגורמים לתקלות השווא ל-4 נקודות עיקריות:

- **תקלת Cannot reproduce**

הבודק נתקל בבעיה פוטנציאלית, פתח עליה תקלה במערכת, המפתח ניסה לשחזר זאת, ולא הצליח. אולי אפילו הבודק עצמו ניסה לאחר מכן לשחזר בעצמו את התקלה שפתח, וגם לא הצליח, ולכן הוחלט לסגור / לדחות את התקלה.

מצב זה יקרה בד"כ כאשר:

הבודק "עלה" על תרחיש מאוד ספציפי, שאפילו הוא לא היה, או עדיין לא מודע אליו, ולא תיעד את הצעדים שעשה בפועל על מנת להגיע לסיטואציה הספציפית הזאת. הסיכון כמובן במקרה הזה, שהפעם הבאה שהתקלה תופיע שוב, תהיה בסביבת הפרודקשיין. ואם התקלה במהותה חמורה, אז הנזק ברור.

או לחלופין, אולי בכלל היתה בעיה נקודתית בסביבה הנבדקת, שבכלל קשורה לתשתית הסביבה (נניח היה אובדן תקשורת רשת לכמה דקות, שהשפיע על הפונקציונליות במערכת).

לכן, במידה ונתונים הסטטיסטיים מצביעים שלא מעט מהתקלות הדחויות הן מהסיבה שלא ניתן לשחזור, הייתי ממליץ לבחון תחילה את תהליך העבודה של הבודקים, בכל מה שקשור להיבט של תיעודי נתוני בדיקה בעת פתיחת התקלה (הצירוף לוגים, נתונים בשימוש, צעדים מדויקים שנעשו לשחזור, תמונת מסך לתיעוד המקרה, וכו').

שנית, הייתי ממליץ לבחון את סביבת הבדיקות עצמה מבחינה תשתיתית – האם כל הגדרות הקונפיגורציה נעשו כפי שנדרש? האם הוקצו משאבים מתאימים לסביבה המאפשרים את יציבותה?

תקלת Deferred

הבודק נתקל בבעיה פוטנציאלית, פתח עליה תקלה במערכת, המפתח בחן את התקלה, והגיע למסקנה שאין באמת תקלה, כי הבודק לא בדק בצורה נכונה את המערכת, והשתמש כנראה בנתוני בדיקה לא מתאימים, או אולי מצבים שהמערכת לא בהכרח תומכת, או שאולי בכלל בדק גירסה לא נכונה.

לכן, במידה והנתונים הסטטיסטיים מצביעים שלא מעט מהתקלות הדחויות הן מהסיבה Deferred, הייתי בוחן יותר לעומק תחילה את אופן תכנון תסריטי ומיקרי הבדיקות, והגדרות הנתונים לבדיקה. אולי תוכננו תסריטים לא רלוונטים. מומלץ בכלל, לשתף ולסקור יחד את תכנון כיסוי הבדיקות עם גורמים נוספים, טרם ביצוע הבדיקות, כמו עם אנשי פיתוח, מנהלי מוצר ופרויקט, ע"מ לקבל חיווים גם מצידם, שמה שתוכנן בהכרח תואם את דרישות המערכת.

תקלת As Designed

הבודק נתקל בבעיה פוטנציאלית, פתח עליה תקלה במערכת, המפתח בחן את התקלה, והגיע למסקנה שאין באמת תקלה, כי המערכת עובדת "בתכלס" כפי שאופיין.

להבדיל מתקלות Deferred, ששם לתפיסתי הבודק מבין מה צריך לבדוק מבחינה עיסקית, אך שגא באופן הבדיקה, לדעתי האישית, לתקלות מסוג as Designed, יש את המשקל הכי גבוה – כי כאשר יש לא מעט תקלות מסוג זה, הדבר בהכרח מצביע על כך שהבודקים אינם מבינים מה הם בפועל בודקים. הם אינם מבינים כיצד המערכת אמורה לפעול, ולמעשה אינם מבצעים את עבודתם בצורה מקצועית.

יתכן בהחלט שהסיבה לכך היא שהמערכת מאוד מורכבת, ונדרשים לימודים והסברים חוזרים ע"י גורמים שמכירים היטב את המערכת, ויש מקום לדאוג לכך שהפער יושלם. אופציה אחרת היא שיתכן שהגורם הוא בכלל מסמכי אפיון ברמה ירודה, או כתובים באופן מאוד כללי (או אפילו כלל לא קיימים לפעמים), מה שמאלץ את הבודקים לנחש לפעמים את "פערי המידע" ולהעריך בעצמם כיצד המערכת אמורה לפעול, ואולי בכלל הבינו אחרת מכפי שהמפתח הבין. עלינו כמנהלים לדרוש ולדאוג ולקדם במקרה כזה את הכשרת הבודקים, ו/או את קבלת המסמכים הרלוונטים ברמה טובה, להשלמת פערי המידע, כדי להוציא תוצרי בדיקות מדויקים יותר. אינטרס של כולם בסופו של דבר.

תקלת Duplicated

הבודק נתקל בבעיה פוטנציאלית, פתח עליה תקלה במערכת, ולאחר מכן התקבלה אינדקציה מהמפתח או המנהל הישיר, שתקלה זהה כבר קיימת (בין אם נפתחה ע"י מישהו אחר לאחרונה, או הייתה קיימת כבר מגירסה אחרת שנפתחה בעבר וטרם תוקנה, ופשוט "שכחו ממנה").

במידה ונתונים הסטטיסטיים מצביעים שלא מעט מהתקלות הדחויות הן מהסיבה של כפילויות, הייתי ממליץ לבחון מספר תהליכי עבודה -

תחילה אופן חלוקת העבודה בין אנשי הצוות. האם אולי יותר מאדם אחד בודק פונקציונליות מסוימת, בין אם במכוון, או בין אחד מהם "גלש" לתחומו של אחר. לתפיסתי, צריך להיות גורם אחד אחראי לכל תכולה נבדקת, ועליו לרכז ולפתוח את התקלות הקשורות לאותה תכולה נבדקת.

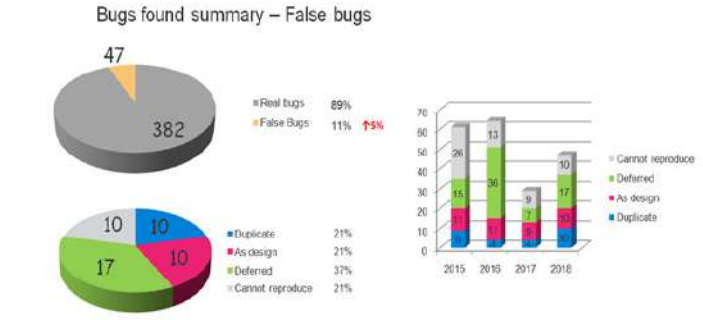
אופציה אחרת – וודאו כי הבודקים מקפידים על קישור במערכת בין תקלה לדרישה / תכולה נבדקת. מצב זה יאפשר למי שמעוניין לפתוח תקלה בנושא מסוים, לוודא תחילה האם התקלה כבר קיימת, וזאת ע"י קישורים שבוצעו קודם לכך, ובלחיצה אחת במערכת, יוכל לוודא זאת.

אופציה שלישית – כמנהלים, ודאו שיש לכם שאילתות מובנות במערכת, לשליפה מהירה של תקלות פתוחות גם מגירסאות קודמות / ישנות. בקשו מהבודקים להשתמש בתגיות או מילות מפתח, בעת פתיחת תקלה במידה ואפשר, ואז כאשר אחד מאנשי הצוות שלכם, או כל גורם אחר, יצרה לדעת האם יש צורך לפתוח תקלה, או שהיא כבר קיימת, תוכלו לתת תשובה יחסית די מהיר, ולהמנע מכפילות.

מומלץ להגדיר מהו אותו אחוז (מהו היעד), של תקלות דחויות, שלהבנתכם המקצועית שאם הגעתם אליו, מבחינתכם יש בעיה שחייבת התייחסות. יכולים להיות חברות, שאפילו אחוז אחד מסך התקלות שנפתחו שהינם תקלות שווא, זה יהיה קריטי בהיבט הארגוני.

אני מעריך שעל אחוז התקלות הדחויות, צריך להיות בסביבות ה-5% לכל היותר.

ניתן לראות בדוגמא שלפנינו, נתונים סטטיסטים על כמות תקלות שנפתחו בתקופה שנתית, עם חלוקה של כמה תקלות הם תקלות אמיתיות, וכמה מהן, הן תקלות שווא. ניתן לראות גם באדום בדוגמא זו, את השינוי באחוזים ביחס לתקופה קודמת שנבחנה, ולהסיק כי משהו בתהליכי העבודה בשנה האחרונה עבד פחות טוב, ולכן אחוזי תקלות השווא עלה. מתוך התקלות שווא, יש את החלוקה הפנימית, ע"ב הגורמים שפורטו לעיל:



מדד כמות התקלות שנפתחו יותר

מפעם אחת

אז בהמשך למדד הקודם, שיותר מודד את "ביזבוז" זמן עבודה של המפתח, ניתן למדוד גם את "ביזבוז" זמן העבודה של הבודק, ע"י בחינה כמה אחוז מהתקלות שלכאורה תוקנו, נבדקות, נמצאות כלא באמת מתוקנות, ונפתחות מחדש. למעשה, ניתן לבחון איזה אחוז מהעבודה (בצורה גסה), הצריך זמן עבודה כפול (או אפילו יותר בחלק מהמיקרים), גם של הפיתוח וגם של הבודקים.

בחלק מכלי הבדיקה בשוק, יש שדה מובנה שסופר את מספר הפעמים שתקלה בסטטוס Open. בחלק מהכלים, ניתן לחלץ את המידע ומההיסטוריה של התקלה.

כמנהלי בדיקות, כמו בכל תחום ניהולי אחר, מומלץ וכדאי פעם בכמה זמן לנסות "לעצור" לרגע ולראות את התמונה הגדולה

כאשר אחוז התקלות שנדרש עבורם ליותר מתיקון אחד, הינו גבוה, הדבר בהכרח מצביע על משהו לא תקין בתהליך העבודה של המפתחים, דבר שבאופן ישיר משפיע על לוחות הזמנים וקצב העבודה של הבודקים, ושל הפרויקט בכלל.

כאשר בודק מבצע Re-open לתקלה שעברה תיקון (תקלה שהייתה בסטטוס Resolved / RFT), כנראה קרה אחד משני הדברים הבאים:

או שהמפתח לא תיקן כמו שצריך את התקלה. סביבת העבודה שלו לא איפשרה לו לתת כיסוי לתיקון באופן מלא, כפי שהיה אולי מתקבל בסביבת הבדיקות, או שפשוט הוא לא הבין נכון את מהות התקלה, ולכן גם לא תיקן כמו שצריך.

אופציה שכיחה נוספת, שהמפתח עושה Check-in לקוד הרלוונטי, ב"ענף" הלא הנכון, והגירסה הבאה שמגיעה לבדיקות, אינה מכילה את התיקון שנעשה.

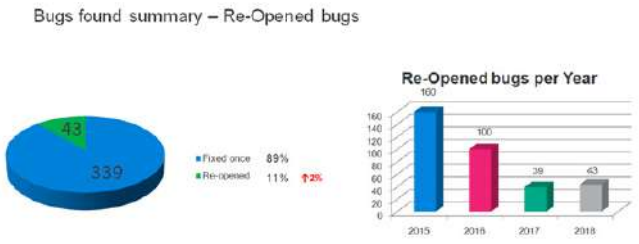
וכאמור, כאשר יש לא מעט מיקרים כאלו, במצטבר "נשרף" המון זמן שלא נלקח בחשבון במסגרת הערכת לוחות הזמנים, ואז קיים חשש ממשי לעיכוב בהגשת הגירסה, או שפשוט מעביר את כולם למצב של עבודה בלחץ, דבר שבאופן טבעי פוגע באיכות העבודה של כולם.

בשורה תחתונה - אחוז גבוה של תקלות Re-opened מחייב בחינה משותפת עם מנהל הפיתוח, למציאת הגורמים לכך. גם פה, הייתי מכוון לאחוז התקלות שנפתחות מחדש, שיהיה בסביבות ה- 5% לכל היותר.

להלן דוגמא המפרטת את אחוז התקלות שנדרשו ליותר מתיקון אחד, מכלל התקלות שנפתחו באותה תקופה.

ניתן לראות כי בשנה האחרונה הייתה עלייה קלה באחוזים של תקלות שנפתחו יותר מפעם אחת (באדום).

שינוי מגמה ביחס לשנים קודמות. מה שמצביע שמשהו בתהליכי עבודה שנעשו עד לאחרונה, לא מיושמים יותר במלואם, או מחייב התאמות. יש שיראו זאת כשינוי זניח, והמצב למעשה "נשמר".

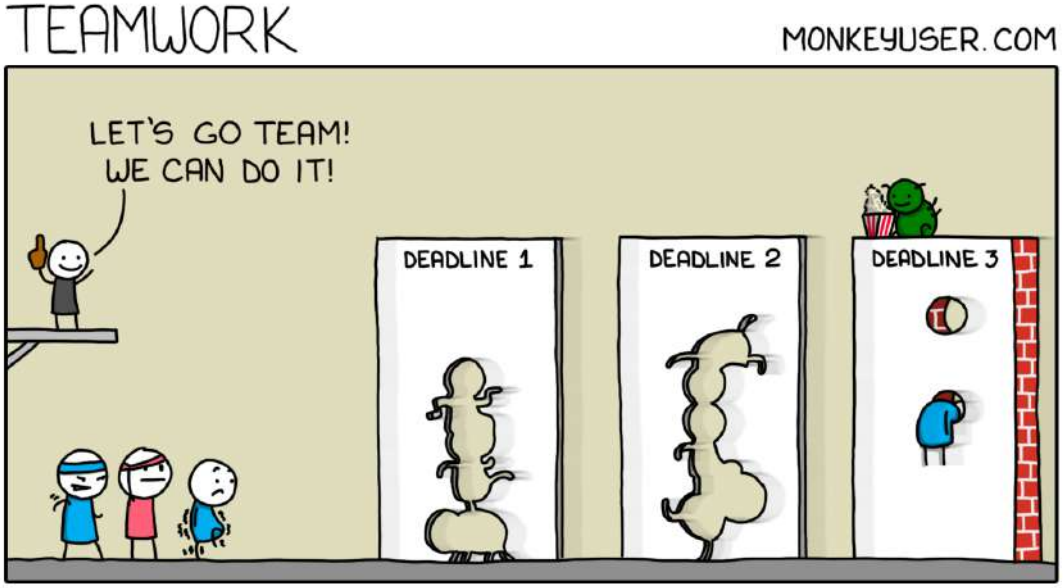


לסיכום

כמנהלי בדיקות, כמו בכל תחום ניהולי אחר, מומלץ וכדאי פעם בכמה זמן (חצי שנה, שנה.. כל אחד יבחר מה שטוב לו), לנסות "לעצור" לרגע ולראות את התמונה הגדולה. לראות מה עובד טוב ומה לא. איפה ניתן להשתפר, ומה כדאי לשמר.

הדוגמאות שעלו פה, הינם חלק ממגוון האפשרויות שכלי הבדיקה מסוגלים לספק לנו כיום מבחינת מידע. השאלה היא כיצד אנו מוציאים את המקסימום מכך, ולוקחים את "המערכת" צעד קדימה. לכל ארגון יש את תהליכי העבודה שלו, המותאמים לאופי הארגון, ולכן יתכנו גורמים שונים לאותם בעיות, בין ארגון לארגון. ניסיתי להביא כמה דוגמאות שהינם גנריות, שיכולות לשמש את מרבית המנהלים. כמובן שניתן להסיק מכל מדד כזה, עוד מסקנות ותובנות, שאני בטוח שתדעו להגיע גם אליהם, עם קצת בחינה וחקירה.

עבודה פוריה!



ראיון עם מנהל בדיקות אלמוג כהן, NESS

אלמוג כהן

נעים להכיר - שמי אלמוג, בן 36, אב לשלושה ילדים. לעולם הבדיקות הגעתי בשנת 2006 היישר מהשירות הצבאי. שירתתי כמוקד ימי בחיל הים. אחד הקצינים ביחידה השתחרר, פתח חברת הייטק ולקח מהיחידה אנשים שהתמקצעו בהגנת חופים. החברה - "הרטק טכנולוגיות" סיפקה שירותים דומים לצבאות אחרים בחו"ל וכך, אנשים כמוני, שלא היה להם ידע בבדיקות אבל היה להם את הידע הנדרש בתחום, הגיעו לחברה והתחילו לבצע בדיקות ועם הזמן גם אני למדתי להכיר את עולם הבדיקות.
משנת 2010 ועד היום אני מספק שירותי בדיקות למערכות והאתרים השונים למוסד פיננסי מוביל. ב-5 השנים האחרונות משתייך לחטיבת הבדיקות של חברת נס טכנולוגיות. למעשה אנחנו מספקים מעטפת מלאה לבדיקות השונות הן מבחינת בדיקות ידניות והן מבחינת בדיקות אוטומטיות. זו עבודה מאתגרת ומעניינת. עבודה מסביב לשעון.

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.

ממה מורכב הצוות שלך?

אנחנו מספקים שירותים בתחום הדיגיטל ובתחום מערכות הליבה לכלל המערכות והאתרים. אני מנהל את תחום הדיגיטל כאשר תחתיי ישנם שישה מנהלים שהם ראשי הצוותים אשר לכל אחד צוות משלו. הצוותים מתמחים בעולמות שוק ההון, עולמות פיננסים, מערכות דפדפנים, צוות ייעודי לעולם היישומונים, צוות נגישות וצוות האחראי על האתרים השיווקים של המוסד הפיננסי.

יש צוותים גדולים של 12 בודקים ויש צוותים יותר קטנים של 2-3 אנשים. הייחודיות של הצוותים היא שהם צוותים דינמים. אנחנו עובדים גם במודל אג'ילי וגם במודל היותר מסורתי של מפל מים. כל פרויקט מתנהל בהתאם לדרישות המגיעות מהשטח.

במה עוסקת הקבוצה וכיצד היא בנויה?

אנו מספקים את שירותי הבדיקות עבור המערכות והאתרים השונים למוסד פיננסי. יש לנו מנהל אתר שהוא המנהל הישיר שלי והוא למעשה מנהל את כל הפעילות. מתחתיו יש שני מנהלי קבוצות, אני מנהל קבוצה אחת ואני אחראי על תחום הדיגיטל וקולגה שלי מנהל את הקבוצה השניה שאחראית על תחום מערכות הליבה, שהן בעיקר המערכות הנמצאות מאחורי הקלעים כמו עולמות כרטיסי האשראי. יחד אנחנו מספקים מעטפת מלאה של בדיקות לכל המערכות השונות של המוסד.

כיצד מתבצעת חלוקת הצוותים מבחינת בודקים ידניים ובודקי אוטומציה?

צוות האוטומציה כולל שבעה אנשים אשר נותנים שירותי אוטומציה לשאר הצוותים.

אנחנו דוגלים בכך שגם הצוותים הידניים ייקחו חלק בתחום האוטומציה ולכן פיתחנו ממשק משתמש (UI) פנימי שיעזור בתהליך וכך כל בודק ידני שלא מגיע מעולם התכנות יוכל להריץ את הבדיקות שלו וגם לדעת איך לנתח את התוצאות ללא צורך בידע של קידוד. כך הבודק הידני נעשה שותף בהרצות בפועל וגם מקבל את הדוח הסופי המכיל את כל המידע על מה שקרה במהלך הריצה וכמובן את התוצאה הסופית. במקרה שטסט מסוים נכשל, הבודק הידני יכול לחקור ולשחזר את הפעולות שהביאו לכישלון, להבין ממה הוא נובע ולתקן לפי הצורך בהתאם או לפתוח באג מתאים לקבוצות השונות.

איך אתם מפתחים את הבודקים שלכם?

אנחנו מאמינים בתהליך שיפור מתמיד. כל עובד צריך לשאול את עצמו איפה הוא רוצה להיות בנקודת זמן כזאת או אחרת ולכל עובד אנחנו מגדירים את התהליך המתאים עבורו עם יעדים ברורים. אנחנו מוציאים לקורסים והכשרות אנשים עם אוריינטציה טכנולוגית בתחום מסוים, נותנים להם להתעסק באותו התחום שבו הם חזקים, ומנסים לבדוק איך ניתן לחבר אותם לתחום. לדוגמה, יש לנו בודקים שמתעסקים עם בדיקות API או בדיקות מיקרוסרויסים, שהן לא רק בדיקות אוטומציה או רק בדיקות ידניות והבודקים יכולים לנוע מתחום אחד לאחר. אנשים מאוד צמאים לזה כי הם מבינים שהם רוצים להיות חלק מהפרויקט, להיות שותפים לתהליך ולא להישאר באותם מקומות לאורך זמן.

האם יש אנשים שלא מחפשים דווקא את ההתפתחות לאוטומציה?

בוודאי. יש לא מעט אנשים טובים מאוד במה שהם עושים ואוהבים את הבדיקות הידניות. כאן בא לידי ביטוי הכוח שלנו. אנחנו למעשה עובדים רוחבית ואנשים יכולים לעבור מתפקיד לתפקיד, מצוות דסקטופ לצוות סלולר למשל וכך הבודק יכול

אנו דוגלים בכך שגם הצוותים הידניים ייקחו חלק בתחום האוטומציה

להתמקצע בהרבה תחומים. אנחנו כל הזמן עושים הצלבות פנימיות על מנת לרענן ולאתגר את האנשים. תמיד נוכל למצוא לעובד תפקיד חדש ההולם את כישוריו ומתאים לציפיות שלו. מי שרוצה, ישאר בחטיבה וימצא את מקומו תוך כדי הדדיות בין שני הצדדים. בעיניי זה הכוח בעבודה עם חברה גדולה המספקת מגוון רחב של שירותים רוחביים.

מהן הדרישות הניהוליות והעסקיות גם ממך כמנהל הקבוצה וגם מאנשי הבדיקות?

אנחנו כפופים לדרישות הלקוח וכאשר אנחנו מקבלים משימה שיש לה דד-ליין מאוד ברור, אנחנו כצוותי הבדיקות אמורים לספק את המעטפת המלאה על מנת להגיע בזמן ליעד שהוגדר לנו. זה כולל הקצאת כוח אדם מתאים מבחינת כמות וידע, לעשות את השינויים המתאימים לפרויקט מבחינת אופן העבודה בין אם זה פרוייקט אג'ילי או פרוייקט שמתבצע בתצורת מפל המים ודורש אפיון וכתיבת מסמכי בדיקות מתאימים. אנחנו עושים את ההתאמות לפי פרויקט מתוך מטרה לעמוד ביעדים שהלקוח הגדיר לנו.

הציפייה שלי בתור מנהל היא ציפייה מאוד פשוטה. רוב האנשים מתמקצעים בתחום הספציפי שלהם אבל זה לא המצב מבחינת הבודק. הבודק מכיר את התהליך מתחילתו ועד סופו. אני מצפה שהבודק יכיר את הארכיטקטורה, יכיר את מאחורי הקלעים של המוצר וגם שיהיה מודע לצד העסקי ויתחבר למשמעות העסקית של כל חלק וחלק במוצר כדי שלא יבדוק כמו רובוט את הכפתור או הלחצן אלא יבין ממש מה המשמעות העסקית של אותו כפתור או לחצן ובכך הוא יהיה בודק טוב יותר ואיכותי יותר.

המנהל האישי שלי הוא המנטור הכי טוב שהיה לי בחיים, הוא דורש ממני עבודה קשה אבל מצד שני מיום ליום הוא מוסיף לי כלי ניהול שיגרמו לי להיות מנהל טוב יותר שיפתחו אותי בעולם הניהול.

עם השנים ועם הניסיון, השאיפה שלי היא לכבוד הדדי ביני ובין העובדים. אני פועל לכך שכל עובד ידע מה מצופה ממנו בהיבט המקצועי והחברתי ויעשה את תפקידו בצורה הטובה ביותר.

ראיון עם מנהל בדיקות אלמוג כהן, NESS

המסר שלי גם למנהלים וגם לבודקים - תאהבו את מה שאתם עושים, תתחברו למה שאתם עושים וככה תהיו עובדים טובים לאורך זמן.

מהם האתגרים הייחודיים של קבוצת הבדיקות שלכם וכיצד אתם מתמודדים אתם?

אחד מהאתגרים הכי גדולים שלנו הוא מתודולוגיות העבודה. כמו רוב המקומות גם אנחנו התחלנו את שיטת העבודה שלנו בצורה מפל המים אבל העולם משתנה כל הזמן והיום יותר ויותר עוברים לעבודה בצורת אג'יל ויש כאן צורך בשינוי רציני במתודולוגיות הבדיקות. יש שינוי בתפיסה, שינוי בארגון, שינוי בכלים.

לדוגמה, אנשים שהיו רגילים לעבוד עם כתיבת מסמכי אפיון לתהליך כולו צריכים עכשיו לשנות את דרך העבודה שלהם ולכתוב מסמכי אפיון לכל איטרציה ואיטרציה ולפעמים גם לכתוב מסמכי אפיון לאיטרציה הבאה תוך כדי הרצה של האיטרציה הנוכחית.

בתצורת מפל המים כל חלק עומד בפני עצמו. כאשר עוברים לתצורת אג'יל יש סקראם מאסטר שלרוב הוא מהפיתוח ואז עולות שאלות: למי אני מדווח, מה התפקיד של המנהל שלי ויש הרבה אנשים שקשה להם עם השינוי. מצד שני יש אנשים שדווקא מחפשים את השינוי. האתגר שלי הוא למצוא את האיזון ולדאוג לזה שכל עובד ימצא את מה שעובד בשבילו.

אתגר נוסף שלנו הוא לספק פתרון 24/7 ללקוח שלנו שנמצא בתחרות עסקית עם מוסדות מהעולם שלו ואנחנו לא מעט פעמים נדרשים לספק פתרונות בזמנים שונים, בחגים, בסופי שבוע, בלילות. אנחנו בארגון מאמינים שאי אפשר לשמור את העובדים דרוכים לאורך זמן וחשוב לדעת מתי לשחרר ולהרגיע את הלחץ. חשוב לנו לשלב את הדרישות הגבוהות של הלקוח תוך כדי שמירה על איזון בית - עבודה של העובד. זה משהו שאנחנו מאוד מאמינים בו ועובדים קשה כדי שזה יתקיים.

איך אתם מניעים את הבודקים?

יש לנו מערכת "גיימיפיקציה" בצורת משחוק (משחק - בעזרת חברה חיצונית) אליה אנחנו מזינים מאמרים ותכניים מקצועיים בתחומים רחבים ויש לנו משהו שנקרא "חמש דקות ביום". כל בודק נכנס 5 דקות ביום למערכת ומבצע את הלומדות, קורא תכנים מקצועיים, משחק משחקים חברתיים ועל כל זה הוא מקבל ניקוד. עם הניקוד הוא יכול להזמין שוברים שונים לשעות הפנאי.

ככה אנחנו מרוויחים שיפור מתמיד בו העובד כל הזמן מתפתח ולומד עוד תחומים ועוד כלים ומצד שני חווית משתמש שבה העובד נהנה מהלמידה ומתחבר לעובדים אחרים שהוא לא נחשף אליהם ביום יום.

בנוסף אנחנו מאוד מאמינים בהכשרת עובדים. אחת לשנה מתקיימים קורסים מקצועיים בנושאים טכניים שונים על מנת להכשיר את העובד, לפתח אותו ברמה המקצועית ולחשוף אותו לתחומים נוספים וכמובן שהעובד מעריך את זה בהתאם.

עם השנים ועם הנסיון, השאיפה שלי היא לכבוד הדדי ביני לבין העובדים

מהן ההמלצות שלך לבודקים ומנהלים בתחום?

לצערי, אנחנו מבלים יותר שעות בעבודה מאשר בבית ולכן חשוב מאוד לבחור את התפקיד ואת האתגר שיגרום לך לקום בבוקר עם חיוך ולעשות אותו בצורה הטובה ביותר. אני מאמין שאם מישהו עושה משהו שהוא אוהב, מאמין בו ומתחבר אליו הוא עובד טוב ויותר ובכך הוא תורם גם לעצמו וגם לארגון.

הרבה אנשים עושים הסבה להייטק מסיבות כספיות בלבד ואז הם מוצאים את עצמם מתוסכלים מהעבודה וחבל.

המסר שלי גם למנהלים וגם לבודקים - תאהבו את מה שאתם עושים, תתחברו למה שאתם עושים וככה תהיו עובדים טובים לאורך זמן.

לגבי מנהלים, מבחינתי מנהל טוב צריך לדעת איך לפתח עובדים, לדעת להאציל סמכויות וליצור מתחתיו שכבת ניהול יציבה. האינדיקציה הכי טובה למדוד מנהל היא לראות את הצוות שלו מתנהל כאשר הוא בחופשה של שבוע שבועיים. אם יש אי סדר אחד גדול, הבעייה העיקרית היא אצל המנהל. זה מראה שלמעשה זה הוא שלא יודע ליצור סדר, לא נותן לעובדים שלו להתמודד עם אתגרים, לא סומך על אנשים. אם המצב ממשיך להתנהל כרגיל זה מוכיח שהמנהל עושה את העבודה שלו כראוי ושהוא חלק מאוד חשוב לחברה.

דבר נוסף שהייתי ממליץ לכל התפקידים באשר הם הוא לא לתת ביקורת אישית, לשאול את השאלות בצורה מכובדת ולהבין שאנשים לא תמיד רואים את התמונה המלאה. לדעת לנהל שיח מקצועי ולזכור תמיד שאין פה אנחנו והם. כולנו פה ביחד במטרה אחת.

פספורט קבוצתי – Ness

סוג המוצר:

דפדפן, שולחן עבודה, סלולר, צד שרת/לקוח

התמחות: שוק ההון, SAP, פיננסי, כ.אשראי, MF

סוגי בדיקות:

ידניות, אוטומטיות, Nft, נגישות

שיטות עבודה:

אג'יל, מפל מים, קודד ותקן (שיטת עבודה זו מותאמת לפעילויות בהן הבודק עובד צמוד למפתח. שיטה זו מתאימה לפרויקטים שלא עובדים במתכונת של דרישות עסקיות, אפיונים וכו, אלא בעיקר מתאימה לבדיקות על מערכות סגורות, בהן המפתח מקמפל תוכנית, הבודק בודק ישירות אחריו ומעביר את התוצרים למפתח וכך ממשיכים עד אשר הבדיקה מאושרת מבחינת תקינות.)

כמות מוצרים:

סדר גודל של 40 מערכות ואתרים שונים,

אחת לחודש גרסה עולה לאוויר.

הקופסא - תבניות חשיבה שימושיות לבדיקות

איסי חזן-פוקס

כל המודלים שגויים, אך חלקם מועילים[1]. אף מודל אינו מדויק לחלוטין והוא בסך הכל הפשטה של מציאות מסוימת, אך לעיתים בחינת המודל מעלה תובנות מעניינות וחשובות[2]. בטור זה אציג בכל פעם תבנית חשיבה שמנסיוני היא שימושית בעבודה שלנו כבודקים.

ההגדרה של איכות

כשהייתי בודק צעיר שמתחיל להעמיק וללמוד, אספתי לעצמי כמה שאלות מרכזיות שמציאת התשובות להן תביא אותי להבנה עמוקה של מקצוע הבדיקות. באופן טבעי, אחת השאלות הראשונות היתה "מה זו איכות". כבודק, אני מבלה את רוב זמני בפעולות שהתכלית שלהן היא לתת חוות דעת שתגרום לשיפור איכות המוצר (אחרת, בשביל מה באתי לעבודה?). אם אגדיר לעצמי למה אני מתכוון כשאני משתמש במילה איכות, אוכל למקד גם את הבדיקות שלי וגם את התקשורת של התוצאות שלהן.

בשלב הזה אני מציע לכם, קוראים וקוראות נאמנים, להפסיק לקרוא, לעצור ולחשוב על ההגדרה **שֶׁלָּכֶם** לאיכות ואיך העבודה שאתם עושים מתקשרת לזה.

ההגדרה שמצאתי מתאימה, היתה זו של ג'רי ויינברג, "Quality is value to some person". מצאתי אותה מספיק רחבה ולא מצומצמת לקונטקסט מאוד מסויים. כבודק אני מְחַוֶּה את דעתי: האם המוצר נותן את הערך שהוא מתיימר לתת למישהו? האם ישנן בעיות (באגים) הפוגעות בערך למישהו? כיוון שלא כל "מישהו" באמת מעניין אותנו, ג'יימס בך צמצם את ההגדרה ל "Value to some person who matters". אם למשל יש פרצת אבטחה במוצר יש כאן ערך להאקרים אבל לא הם אלו שאכפת לנו מהם. אכפת לנו ממי שאנחנו בודקים את המוצר בשמו:

> **ההגדרה של "איכות" יכולה לשמש לנו בתור מצפן כדי לחשוב באופן ביקורתי על הגדרות אחרות שמשמשות אותנו ביום-יום**

העסק שלנו, הלקוחות המשלמים ומשתמשי הקצה.

אז מה עושים עם זה? ההגדרה של "איכות" יכולה לשמש לנו בתור מצפן כדי לחשוב באופן ביקורתי על הגדרות אחרות שמשמשות אותנו ביום-יום. לדוגמא, אם ישנה הגדרת דרישות שנראית לנו שונה ממה שיתן ערך בסופו של דבר למשתמש, נבין שיש כאן בעיה ונציף אותה.

אם אנחנו נמצאים בדיון על מידת חומרה של באגים, נחשוב האם יש כאן בעיה בערך שהמשתמש מקבל מהמוצר וכמה היא חמורה וכך נגדיר לעצמנו את הבעיה בצורה מדויקת יותר.

מה ההגדרה שלכם לאיכות? ואיך אתם משתמשים בה ביום-יום?

לתגובות והצעות issihf@gmail.com

איסי חזן-פוקס

המייסד הגאה של קבוצת המיטאפ TestIL, בודק תוכנה, קושחה וחומרה כבר יותר מ-20 שנה במרכז הפיתוח של אינטל בירושלים.

כיום משמש כמהנדס בדיקות מערכת בקבוצת ה Wi-Fi

---

1. Box, G. E., Hunter, J. S., & Hunter, W. G., Statistics for experimenters, Wiley Hoboken, NJ, USA, 2005
2. ויקיפדיה "מודל מתמטי"

בחן את עצמך | טל פאר

השאלה שלנו מתייחסת לפרק 6.1.2 בסילבוס שמדבר על יתרונות וסיכונים באוטומציה של בדיקות.

עבדתי פעם כמנהל בדיקות בחברה כשהגיעה אלי דרישה להטמיע אוטומציה של הבדיקות. הדרישה לא הגיעה מהפיתוח אלא ממנהל לקוחות ששמע מחבר שלו שאוטומציה פתרה להם הרבה בעיות. אבל האם באמת רכישה של כלי בדיקות תבטיח הצלחה ותפתור את הבעיות שאולי קיימות בבדיקות?

תהליך הכנסת כלי חדש לארגון דורש מאמץ לשם השגת יתרונות שיתרמו להצלחת הכלי והצלחת הקבוצה אליה מיועד הכלי. הטענה "זה הצליח לחברה ההיא" אינה מספקת כשבאים לבקש לרכוש כלי חדש. לשם כך עלינו להבין מה אנחנו מצפים לקבל מהכלי, מה היתרונות ומה החסרונות של הכלי.

הטמעת כלים לביצוע בדיקות עשויים לתת מספר יתרונות:

- הפחתה בעבודה ידנית חוזרת. באמצעות כלי לביצוע בדיקות ניתן להריץ בדיקות נסיגה מספר רב של פעמים. ניתן גם להכניס נתוני בדיקה שוב ושוב.
- כלים כאלה יגדילו את העקביות בבדיקות, מאחר והם יבצעו את הבדיקות באותה צורה שוב ושוב.
- כלי בדיקות יאפשרו גישה קלה יותר למידע על הבדיקות וכך ניתן יהיה לקבל בקלות את סטטוס הבדיקות וניתוחים סטטיסטיים אחרים.

במקביל יש לא מעט סיכונים בשימוש בכלים לביצוע בדיקות. הציפיות מהכלי עשויות להיות לא ריאליות ופעמים רבות הכלי נתפש כפתרון האולטימטיבי לקיצור זמן הבדיקות, עד שמתברר שזה לא כך. גם המאמץ להשגת היתרונות מהכלי עשוי להיות מוערך בחסר וכך לא נקבל את הזמן והתקציב הדרושים, דבר שיביא לאכזבה ואף כישלון של הכלי.

ישנם עוד יתרונות וסיכונים ואני מזמין אתכם לקרוא עליהם בסילבוס, אבל הדבר החשוב ביותר הוא שלפני שבוחרים כלי לביצוע בדיקות יש לוודא שאנחנו מבינים את היתרונות והסיכונים בשימוש בכלי ולתכנן נכון את הבחירה והרכישה של הכלי המתאים לצרכים שלנו.

הבה נבחן את התשובות:

א. היתרונות של כלי לביצוע בדיקות הן לא ביצירת בדיקות נסיגה, אלא יותר בביצוע שלהם. בדיקות הנסיגה הן נגזרת של הבדיקות הרגילות שאנחנו מבצעים (ראה על בדיקות נסיגה בפרק 2.3 בסילבוס).

ב. תוצרי הבדיקות הם מקרי הבדיקה, תסריטי הבדיקה (ידניים ואוטומטיים). אלה יהיו מתוחזקים באמצעות כלים לניהול תצורה (configuration management) ולא באמצעות כלי לביצוע בדיקות.

ג. לצורך עיצוב של בדיקות אבטחה יש להשתמש בכלים ייעודיים לכך.

ד. אחד היתרונות הגדולים של כלי לביצוע בדיקות הוא הפחתה בחזרה על עבודה ידנית, לדוגמה הרצה חוזרת של בדיקות ידניות או הכנסה חוזרת של נתוני הבדיקות למערכת.

לפי ההסברים הנ"ל התשובה הנכונה היא תשובה ד'.

השאלה תורגמה משאלון דוגמה A של ISTQB®.

הדרך לבחירת כלי לניהול בדיקות

ליזה (אליזבטה) ויסמן

לאחרונה נתקלתי בפורום של בודקי תוכנה בשאלה: "על איזו תוכנה לניהול בדיקות חינמית או זולה מאוד אתם ממליצים?" ואז זה היכה בי. עד כמה מאפשרים לנו, לבודקי התוכנה, לבחור כלי לניהול בדיקות? ואם כבר מאפשרים, מבקשים שנתבסס על קריטריון אחד - זול?

אם כן, לא כל הארגונים מאפשרים לבודקים שלהם לבחור לבד את התוכנה לניהול הבדיקות מאילוצים שונים. בארגון שבו אני עובדת זה נעשה, אך המשימה לא הייתה קלה כלל.

יחד עם הניסיון בעבודה בארגונים שונים באה ההתנסות בכלי לניהול בדיקות מסוים. על כן, מדוע שפשוט לא נתקין בדיוק את הכלי בו צברנו את הניסיון הרב ביותר?

בעולם התוכנה יש אינספור כלי בדיקה. לכן, חיפוש אחר תשובות בפורומים שיענו על השאלה, איזו מהן עדיפה, לא בהכרח תיתן לכם אינדיקציה למה שאתם באמת צריכים. לכל חברה הדרישות שלה ההתאמות הנדרשות עבורה. אפשר לראות את הדבר כהתאמה לזוגיות... בן הזוג המושלם יהיה שונה בעיני כל אחד ואחת.

אם כך, המסקנה היא שיש צורך בתהליך מיון...

להלן הקריטריונים החשובים ביותר שיש לקחת בחשבון בבחירת כלי בדיקה מתאים לארגון שלכם:

01 הגדרה של קריטריונים לסינון ראשוני

זהו הקריטריון הקריטי ביותר והקשה ביותר לשינוי עקב מגבלות שכבר יש לכם מראש בארגון בו אתם עובדים. למשל, לא תמיד ניתן "לייבא" את הכלי ממקום העבודה הקודם כי בארגון הנוכחי חלק מהמשתמשים בארגון הנוכחי עובדים עם Mac והכלי המדובר נתמך אך ורק במערכת הפעלה "חלונות".

קריטריונים לדוגמא:

- כמה משתמשים הולכים להשתמש במערכת הזו ומה הם הצרכים שלהם
- האם חובה שהעלות תהיה עד $X שנתי או שיש דרישה רק לכלי חינמי לחלוטין
- האם יש צורך בתמיכה גם בבדיקות ידניות וגם בבדיקות אוטומטיות או רק בדיקות ידניות
- האם יש צורך בהתממשקות עם תוכנות הקיימות בחברה כבר, כגון JIRA, GIT וכו'.
- האם ישנה מגבלת זמן לבחון את כל כלי ניהול בדיקות. במידה ויש נעבור רק על X כלים ומתוכם נבחר אחד

02 לערוך רשימת כלים

נבחן ונוודא שהם מתאימים לסינון שהגדרנו בסעיף הקודם.

איפה מאתרים כלים כאלו?

ניתן לחפש בפורומים ובקבוצות מדיות חברתיות, במגזינים של בודקים (כגון עולם הבדיקות), בחיפוש בגוגל, בהתעניינות אצל חבריכם הבודקים - במה משתמשים בארגון שלהם. רעיון נוסף הוא להציץ בפרסומי משרות פנויות עבור בודקים כדי להבין מה הן מרבית הדרישות במרבית המשרות בשוק כה דינאמי וכמובן מהניסיון התעסוקתי שלנו, דבר שככל הנראה לקחנו בחשבון כבר בסעיף הקודם.

אתרים לדוגמא בהם נוכל לחפש אחר רעיונות:

https://www.guru99.com/top-20-test-management-tools.html

https://www.softwaretestingmaterial.com/test-management-tools/

03 יש להגדיר קריטריונים המתייחסים לעומק השימוש

הכוונה בעומק השימוש היא שעליכם להגדיר מה אתם באמת צריכים מתוך פיצ'רים רבים ושונים המוצעים בכלים השונים. כדאי לייצר מטריצת בדיקות לכלים שונים. כלי מסוים יכול להיות עם כיוון שיכול להתאים לחברה עם מוצר ותיק שרצות עליו רגרסיות אוטומטיות ויש מעט פיצ'רים חדשים ובחברה כזו אף אחד לא מתעניין בסטטיסטיקות ביצועים של בודקים. כלי כזה למשל לא מתאים לחברה עם מוצר חדש ללא בדיקות רגרסיה אוטומטית הדורש התמקדות בדו"חות עבור ההנהלה על התקדמות הפרויקט. **זהו השלב בו עליכם להגדיר מה הן דרישות החברה.**

כדאי לתת לכל קריטריון משקל (ניקוד).

בשלב ראשוני ניתן להתחיל בבניית קטגוריות יותר כלליות ואז לפרק אותן לתתי נושאים מדויקים יותר (עם ניקוד מדויק יותר)

קריטריונים לדוגמא:

- דו"חות – כדאי לפרק לאילו בדיוק דו"חות צריך להפיק (מה יהיה מוצג בדו"ח?)
- הרצת בדיקות אוטומטיות – מה חשוב לנו שיהיה \ שלא יהיה ?
- בדיקות – למשל איך תתבצע תחזוקה \ מיון \ חיפוש \ שכפול \ העתקה \ עדכון וכיו"ב.
- עץ בדיקות – האם יהיה מסוג תיקיות או מיון?

טבלה לדוגמא:

ליזה (אליזבטה) ויסמן

בודקת תוכנה עם ניסיון מעל 7 שנים.

בעלת תואר בהוראת מתמטיקה מהטכניון והסמכת בודקת תוכנה גם מהטכניון.

בודקת תוכנה נלהבת, אוהבת ללמוד, לחקור ולגלות

הדרך לבחירת כלי לניהול בדיקות
ליזה (אליזבטה) ויסמן

04 מילוי נתונים

בשלב זה יש להיכנס לאתר הבית של כל כלי מסעיף 2 ולמלא את כל המידע בטבלה שבניתם בסעיף 3. כך תהיה לכם השוואת תפוחים מול תפוחים.

לדוגמא:

Criteria	Decision Weight		tool 1 - name		tool 2 - name	
Type Cloud	Cloud No	3	Cloud	100	Cloud	100
Type Server linux only install	Server No	1	Server	100	No	0
JIRA Integration	JIRA Integration No	2	JIRA Integration	100	JIRA Integration	100
OS Support:WIN + MAC & Browsers Support	WIN + MAC & Browsers Support WIN+MAC Browsers Support Only WIN Only MAC Only	6	WIN + MAC & Browsers Support	100	WIN + MAC & Browsers Support	100
Jenkins Integration	Jenkins Integration No	1	Jenkins Integration	100	Jenkins Integration	100
Selenium Integation	Selenium Integration No	0	Selenium Integration	100	Selenium Integration	100
Requirements	Requirements No	2	Requirements	100	Requirements	100
New Test Plan Creation Add attachment to the step\test	Add attachment to the step\test No	2	Add attachment to the step\test	100	Add attachment to the step\test	100
New Test Plan Creation Add link to the step\test	Add link to the step\test No	3	No	0	Add link to the step\test	100
Test Runs Opening bug from the test run	Opening bug from the test run No	3	Opening bug from the test run	100	Opening bug from the test run	100
Search option	Search option(0-100)	3	Search option	100	Search option	100
Add link to actual result of test run	Link test results to issues No	1	Link test results to issues	100	Link test results to issues	100
Report Report inside the tool	Report inside the tool No	0	Report inside the tool	100	Report inside the tool	100
Dashboards	Dashboards No	3	Dashboards	100	Dashboards	100
Tree Functionality Import Excels	Import Excels No	2	Import Excels	100	Import Excels	100
Summry	points	90		77.63888889		80.41666667

05 התנסות עם שלושת הכלים שקיבלו את הניקוד הגבוה ביותר

חלק מהכלים יכולים להיות חינמיים, בעוד כלים אחרים מאפשרים לכם להתקין גרסאות ניסיון (trial). זו יכולה להיות ההזדמנות שלכם לבחור מבין שלושה כלים, שכולם, על הנייר, אמורים להתאים לכם כך תוכלו לבחון האם אחד משלושת הכלים יהיה ידידותי יותר לשימוש בגלל קריטריונים שלא חשבתם עליהם כלל, טרם השתמשתם בכלי הזה בפועל. פעולות אלו, מובילות אותנו לסעיף הבא.

06 יש להגדיר קריטריונים עוד יותר ספציפיים בעקבות שימוש בגרסת הניסיון עבור הכלים מהסעיף הקודם.

הקריטריונים הפעם מתייחסים יותר לביצועים וליכולות של פיצ'רים לזרז או לבצע את עבודת הבודק בקלות יתרה ועם פחות מקום לאי נוחות, תמיכה ושירות. קלות ההתקנה והתחזוקה יכולים להוות גם קריטריונים חשובים.

אגב, בשלב זה כדאי לתת למספר בודקים להתנסות בכלים אלו על מנת לקבל פידבק מהמשתמשים בפועל. אלו יכולים להיות בודקי אוטומציה, בודקים ידניים, בודקי E2E ואלו שבודקים CORE. אלו יכולים, עקב נקודת המבט שלהם, לשים לב לדברים מעט שונים בכלי שאותו בוחנים.

Criteria	Decision Weight		tested 1 - name		tested 2 - name	
Reports To see report without entrance to the Test Run	To see report without entrance to the Test Run No	1				
Reports Report to see the difference between build versions runs	Report to see the difference between build versions runs No	3				
Reports Report per Test (to see the version when started failure)	Report per Test (to see the version when started failure) No	2				
Dashboards	Dashboards No	3				
Tree Functionality Import Excels	Import Excels No	2				
Tree Functionality Export all tests for using in another tool	Export all tests for using in another tool No	3				
Tree Functionality Add several test from Test Plan to Test Run altogether	Add several test from Test Plan to Test Run altogether No	3				
Tree Functionality Several times duplication of the same Test Plan to different Test Runs	Several times duplication of the same Test Plan to different Test Runs No	3				
Tree Functionality Reuse tests with a little change	Reuse tests with a little change No	3				
Tree Functionality Drag&Drop, re organizing the tree	Drag&Drop No	3				
Tree Functionality yes\no		5				
Pricing	10-20 users per user: Free Price	0				
Performance	High Performance Medium Performance Bad Performance	3				
Support	Support (add via email, cell, forum in comment)					

07 מילוי נתונים

כעת יש למלא נתונים בטבלה של סעיף 6 וכמובן לתת ניקוד יחסי עבור כל קריטריון ולסכם כל כלי.

08 מזל טוב – בחרתם כנראה את הכלי עם הניקוד הגבוה ביותר

ייתכן שתצטרכו להכין מצגת מסכמת עבור ההנהלה על מנת לקבל אישור כספי, זמני הטמעת המערכת וכוח אדם שיבצע זאת.

המסקנה שלי:

אין אף כלי לניהול בדיקות מושלם, תמיד יהיה איזה פיצ'ר שיחסר לכם. יש ריבוי תוכנות כאלו בשוק, כדי שכל ארגון יוכל להתאים לעצמו: לסגנון שונה של עבודה, לצרכים שונים של כל ארגון, לשלב המוצר שהארגון מייצר וכמובן לתקציב שהארגון מוכן להשקיע.

אם ממש אין לכם זמן לכל תהליך ההשוואה הזה, יש כבר אתרים שישמחו לעשות את ההשוואה הזו בשבילכם, אך כמובן שלא יהיה כיסוי מלא לכל הצרכים שתגדירו.

לדוגמא:

https://www.getapp.com/it-management-software/testing/

בהצלחה!

עורכי הגיליון

טל פאר

בעל ניסיון של יותר מ-20 שנים כבודק ומנהל בדיקות במגוון חברות וטכנולוגיות במודלי פיתוח שונים. כיום טל יועץ ומדריך בדיקות עצמאי.

טל חבר ב-ITCB® וגזבר הארגון העולמי ISTQB®.

רוביק סביאנץ

בודק תוכנה, נמצא בתחום כ-3 שנים את דרכו התחיל בחברת CARAMBOLA בה הוא נמצא גם היום.בשנה וחצי האחרונות מחזיק את כל מערך הבדיקות של החברה כבודק יחיד. בזמן הפנוי - ספורט, טיולים ומחשבים.

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.

שי ביטון

בעל 12 שנות ניסיון בפיתוח אוטומציות ובדיקות.

עובד כיום ב-Qwilt.

עמית ורטהיימר

בודק תוכנה ב-Deep Instinct.

רחל ברוך

הנדסאית תוכנה, לפני 3 שנים לאחר הפסקה של שנים מעולם ההייטק חזרה לעולם התוכנה בכל הכוח. נהנית להיות בצד הבודק עם חשיבה של מפתחת. אין יום שהיא לא לומדת משהו חדש בעולם התוכנה.

יאיר נסימוב

בעל ניסיון של כ-10 שנים בתחום בדיקות אוטומטיות. מייסד חברת Rain the Dog המתמחה בייעוץ ומתן פתרונות אוטומציה מותאמים אישית לחברות.

מרצה ומפתח קורסי אוטומציה.

יוטיובר מתחיל אספן תקליטים, חובב טיולים, ריצה ואגרוף תאילנדי נשוי + 2 + כלב - Rain.

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.

תמרה מוסונובה

בודקת תוכנה ואינטגרציה בחברת Varonis.

בעלת תואר בתקשורת וקולנוע והסמכות פיתוח אפליקציות Web של 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.