# מסעותיו של בודק תוכנה מארץ המפל לארץ האג'יל והקנבן

מאת יובל ירט

במסגרת עבודתי אני נתקל בארגוני בדיקות תוכנה רבים. חלקם עובדים בשיטת המפל ונמצאים בשלבי חשיפה ראשונים לשיטות מבוססות אג'יל. חלקם נמצאים במהלך המסע למעבר בין השיטות, וחלקם עובדים כבר בשיטת אג'יל ומחפשים איך לעשות זאת יותר יעיל. במסגרת מאמר זה אנסה לתאר מה עובר על בודק תוכנה טיפוסי שהארגון שלו החליט לעבור לאג'יל.

אשית, בואו נבחן איך נראה היום יום של בודק התוכנה בשיטת המפל. פרויקט או גרסה טיפוסית מתוכננים לפי שלבים. במהלך השלבים הראשונים של הפרויקט בודקי התוכנה מעורבים מעט מאוד. מתכננים את שלבי הבדיקות, כותבים מסמכי בדיקות ומכינים את עצמם ליום בו תועבר הגרסה לבדיקות. בדרך כלל בשלבים הראשונים בודקי התוכנה עדיין עסוקים בפרויקט/גרסה קודמים...

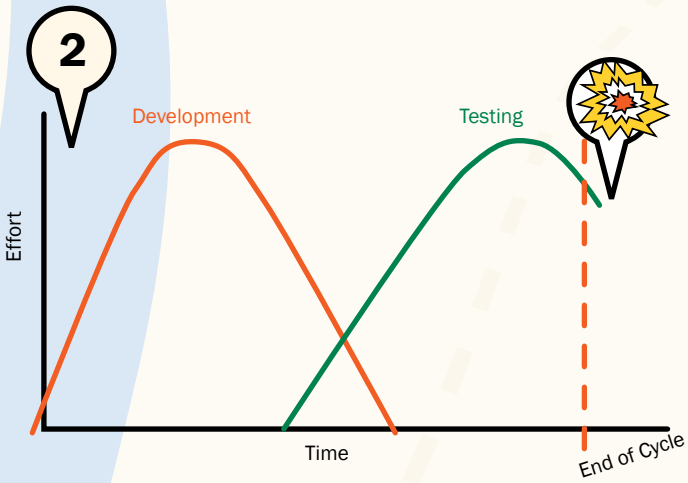
כשמתקרב יום ההעברה לבדיקות, המתח גובר. צוותי הפיתוח מתחילים לבדוק אינטגרציה של הגרסה, בשאיפה לראות שדברים עובדים ביחד. לרוב, בתאריך המתוכנן, הגרסה עדיין לא מוכנה להעברה לבדיקות. האינטגרציה עוד בצרות. עובר עוד זמן, ובודק התוכנה כבר בכוננות ספיגה. יהיה לו פחות זמן להשלים את הבדיקות, ומכיוון שהבדיקות תמיד בסוף, הוא כבר יודע מה יהיה הפתרון ואת מי יאשימו כשלמרות שעות נוספות

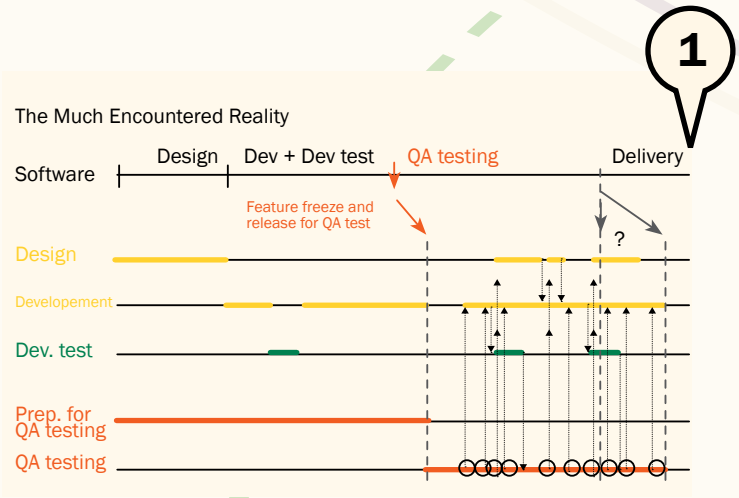
**יובל ירט**

יובל ירט הוא יועץ בכיר בחברת AgileSparks המתמקד בתחומי בדיקות אג'יליות, קנבן וניהול פרויקטים ותכניות גדולות. יובל מגיע מעולם ניהול הפיתוח, עם נסיון של מעל 17 שנה בתחומי פיתוח תוכנה ו-IT. יובל כותב בלוג בנושאי פיתוח תוכנה ואג'יל בכתובת www.yuvalyeret.com

AgileSparks מספקת פתרונות שלמים בתחום אג'יל/סקראם/קנבן ועוזרת לחברות לשפר את יכולות ניהול ניהול הפרויקטים שלהן, עם דגש על פיתוח תוכנה ואינטגרציות. שירותי AgileSparks כוללים הדרכה והטמעה בבית הלקוח וכן הדרכות ציבוריות ופרטיות בנושאי בדיקות אג'יליות, סקראם, קנבן, ניהול פרויקטים, ניהול מוצר בסביבה אג'ילית, ועוד. www.agilesparks.com

AgileSparks מארגנת את כנס Agile Israel השנתי, המתקיים ב11 באפריל השנה בפעם הרביעית. הכנס מיועד לעוסקים בתחום האג'יל והמתעניינים בו, ומתקיים עם נבחרת מרצים אורחים מחו"ל שלל מרצים מובילים מישראל. http://www.agilesparks.com/AgileIsrael2011Agenda



ולחץ מטורף הגרסה תידחה. "למה תמיד יש לנו צוואר בקבוק בבדיקות? תמיד הם מעכבים לנו את הגרסה!"

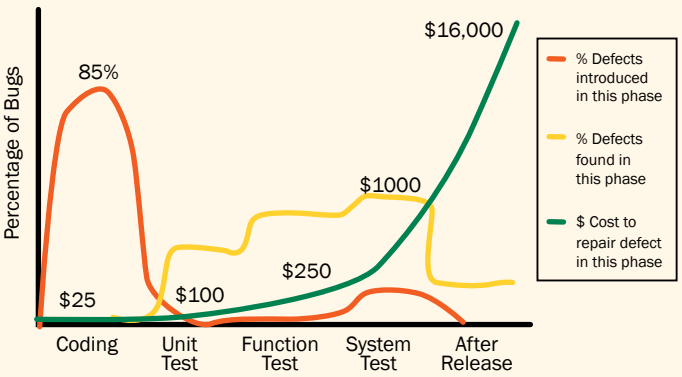
בהינתן הדינמיקה הזאת, שכמעט תמיד חוזרת על עצמה, אין פלא שבודק התוכנה רואה עצמו כקו ההגנה האחרון, וכשמגיע הזמן לבצע את הבדיקות, הוא נלחם בחירוף נפש על מציאת הבאגים, וידוא שהם מתוקנים, ומוצא עצמו מתוסכל כאשר הרבה מהבאגים שפתח לא יתוקנו בגרסה.

למה הבאגים לא יתוקנו? כי אנחנו בפיגור בלוח הזמנים, ואם נתקן, נצטרך לעשות עוד סבב רגרסיה. אין לנו אוטומציה מלאה כך שסבב רגרסיה הוא יקר מאוד. למה אין אוטומציה? כי קיבלנו את הגרסה ברגע האחרון, איך נספיק לעשות אוטומציה? ובכלל, מי שעושה אוטומציה הוא איש האוטומציה שיושב בצד ומנסה להשלים בגבורה פער של שנים של כתיבת קוד, כאשר אף אחד לא מנסה לעשות לו את החיים קלים יותר.

עוד עובדה שאיש הבדיקות שם לב אליה היא שככל שהבאג שהוא מוצא הוא בתכולה שנמצאת בגרסה זמן ארוך יותר, לוקח יותר זמן למצוא מה המקור לבעיה, ויש פחות סיכוי שיתקנו. למה?

3

Figure 1 Cost of Quality - Applied Software Measurement: Global Analysis of Productivity and Quality - Capers Jones




הלכו ובדקו במספר רב של פרויקטים וגילו שרוב הבאגים נכנסים לקוד בשלב הקידוד (מפתיע, נכון?). אם מסתכלים על מתי מגלים אותם, אנחנו רואים שרוב הבאגים מתגלים בצורה טיפוסית בשלבי בדיקות פונקציונליות ובדיקות מערכת. אם עובר זמן רב בין השלבים - כמו שקורה בשיטת המפל בה אנחנו מסיימים את כל הקידוד של כל התכולה ורק אז עוברים לשלבי הבדיקות – המחיר לתיקון הבאג עולה בצורה משמעותית ככל שמוצאים אותו בשלב מאוחר יותר. במלים אחרות, יש לנו פה בעיה של Late Feedback – משוב מאוחר.

עובדה זאת מובילה לשתי אפשרויות – או שאנחנו מקדשים את האיכות ואז פוגעים בתאריך היעד של הפרויקט, או שאנחנו מקדשים את תאריך היעד ואז מוצאים דרכים להתפשר באיכות. אנחנו "מנפנפים" יותר ויותר באגים או מחביאים אותם מתחת לשטיח. כל מי שעסק בכתיבת Release Notes וישב בפגישת Bug Waivers לקראת גרסה יודע על מה אני מדבר. אני מודה ומתוודה שגם אני עסקתי בפעילות נלוזה זאת ולא פעם הייתי צריך להסביר לבודקי התוכנה למה בעצם צריך לעשות את הפשרה הזאת, ולמה הם עדיין צריכים להמשיך ולבדוק למרות שלא את כל הבאגים מתקנים.

מי שהשתתף מספיק שנים ומספיק פעמים בפרויקטים כאלו, מתחיל לחפש דרך אחרת. העולם האג'ילי מספק אלטרנטיבה מעניינת. במקום להעביר את כל התכולה ביחד דרך שלבי העבודה (אפיון, פיתוח, בדיקות וכד') – אג'יל אומר שאנחנו בוחרים יחידות תכולה קטנות יותר וכל פעם לוקחים חלק קטן מהן ומעבירים אותן דרך כל שלבי העבודה. לדוגמא, אם יש לי בפרויקט 20 יכולות (Features), במקום לקודד את כולן ולהעביר לבדיקות, אני אבחר את היכולת הראשונה בתעדוף, אאפיין, אקודד ואבדוק אותה, ורק כשארגיש שהיא מוכנה לשחרור, אעבור ליכולת הבאה.

מעבר לכך, גישת אג'יל ממליצה לעבוד בצוותים בהם מתבצעות כל הפעילויות הנ"ל ביחד, המכילים נציגים של כל ההתמחויות הנדרשות על מנת לקחת תכולה ולהפוך אותה מ"דרישה" ל"מערכת עובדת ובדוקה".

איך נראית העבודה של בודק התוכנה בגרסה טיפוסית העובדת לפי אג'יל? הבודק נמצא בצוות יחד עם מפתחים מהתמחויות שונות. הצוות הזה עובד באיטרציות קצרות (לפעמים יקראו ספרינטים). כל שבועיים-שלושה הצוות יפגש לתכנן מה התכולה/סיפור שיתמקד בו באיטרציה הקרובה. הצוות מחליט מה יספיק בזמן הזה, איך תתחלק העבודה, ויוצא לדרך. במהלך האיטרציה הצוות נפגש כל יום כדי שכל חבר ישתף את שאר חברי הצוות במצבו ומה מעכב אותו, כך שחברי הצוות יוכלו לעזור אחד לשני לעמוד במשימה הקבוצתית שלהם. הם משתמשים בכלי Visibility כגון Burndown Burnup, Board, Task, הכל כדי שכולם יבינו איפה עומדים, האם אנחנו בכיוון הנכון ואיפה צריך למקד מאמץ כדי להצליח. המטרה היא שלצוות יהיה יעד מאתגר, אבל אפשרי, שהם יכולים לעמוד בו בלי להרוג את עצמם.

אם מסתכלים על העומס על בודק התוכנה, המטרה היא לעבור מעומס מטורף בסוף הגרסה וזמנים "מתים" יחסית במהלך הגרסה, לעומס עבודה סביר לאורך כל הגרסה.

עד כאן, התאוריה.

מה ההתנסות הטיפוסית של בודק תוכנה בארגון שבאמת עובר לאג'יל? זה באמת תלוי מהי נקודת ההתחלה. ארגונים שעברו לאג'יל מתוך איזון טוב בין בודקים למפתחים, עם כיסוי אוטומציה טוב ותשתית טובה לפיתוח אוטומציה, אכן יגיעו די מהר למנוחה ולנחלה.

אבל, רוב הארגונים מתחילים מנקודה אחרת. בדרך כלל אנחנו רואים חוסר-איזון כאשר אין מספיק בודקים ביחס למפתחים, וגם אין כיסוי אוטומציה טוב. במצב הזה, המעבר לגישה אג'ילית של התמקדות בתכולות אחת אחת, מציף די מהר מספר בעיות.

למשל, אפשר לראות שקצב הפיתוח יותר גבוה מקצב הבדיקות. הגישה האג'ילית אומרת שכולם צריכים לרוץ פחות או יותר ביחד – Whole Team Approach. הפערים שנוצרים הם בעיה שמובילה שוב למשוב מאוחר. זאת בעוד הגישה האג'ילית מדגישה מאוד את הצורך ואת היתרון דווקא במשוב מוקדם ככל האפשר – Early Feedback. בעצם אג'יל, ובצורה יותר רחבה, גם שיטת הניהול הרזה (Lean), מנחה אותנו לבחון כל החלטה תהליכית שאנחנו עושים ולראות האם היא מקדמת משוב מוקדם יותר ואת צמצום כמות העבודה שהתחלנו ולא סיימנו – במלים אחרות "המלאי" שיש לנו בתהליך.

בודק התוכנה הטיפוסי נתקל כבר בכל מיני שיטות להתמודד עם בעיית הפרש הקצבים הנ"ל:

הוצאת תכולה מהגרסה אחרי שהיא פותחה. המפתחים כבר פיתחו משהו, ועכשיו הם מחכים שהוא יבדק. בשלב זה מגיעים המנהלים ומבקשים מהם להוציא את התכולה הזאת מהגרסה כי אין זמן בדיקות. במקרה הטוב המפתחים יהיו מתוסכלים. במקרה היותר טיפוסי הם גם יפנו חלק מהתסכול כלפי ארגון הבדיקות שכל הזמן מעכב אותם וגורם להם לבזבוז.

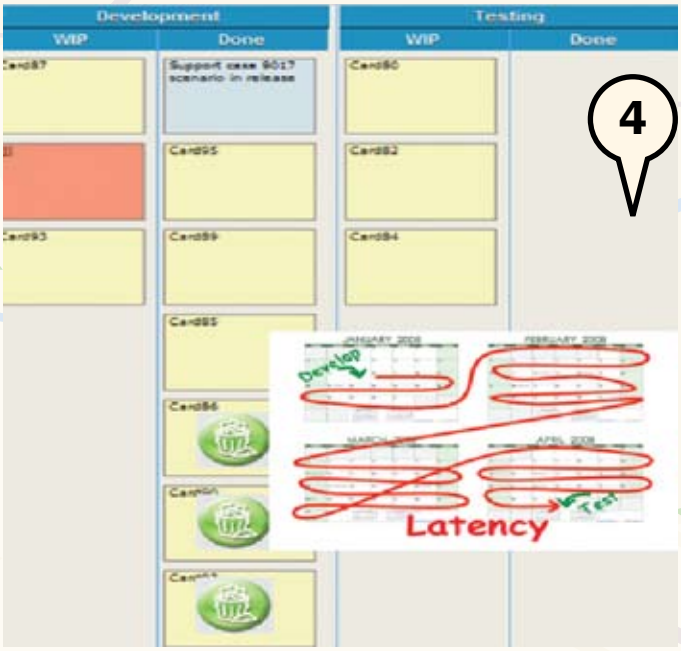
מוציאים תכולה באיכות נמוכה. מגיעים לבדוק אותה מאוחר בגלל התור הגדול שהצטבר בכניסה לבדיקות ואז כבר קשה לתקן את כל הבאגים ומחליטים להוציא בשביל "להגיד שיש לנו", אבל כולם יודעים שזה לא באמת עובד. בודק התוכנה מרגיש מתוסכל מכך שעכשיו בשטח יגידו שהגרסה לא איכותית ויאשימו אותו.

בסוף הגרסה, כאשר התור בכניסה לבדיקות כבר מאוד מאוד ארוך, נותנים למפתחים לבדוק חלק מהתכולות. המפתחים מתוסכלים כי זה לא תפקידם... בודקי התוכנה מוטרדים כי הם לא כל כך סומכים על המפתחים לבצע בדיקות.

לא יודע מה אתכם, אבל אף אחת מהאופציות הנ"ל לא מוצאת חן בעיני יותר מדי...

הגישה האג'ילית אומרת משהו אחר. שבהתחלה אולי יהיה קשה יותר וכואב יותר, אבל בהמשך הדבר ישתלם. אג'יל אומר שאסור לפתוח פער. בואו נעשה מה שצריך כל הזמן כדי לצמצם פערים ולנוע באותו הקצב. איך גורמים לזה לקרות?

בואו קודם כל נהפוך את שלב הבדיקות להכי יעיל שאפשר. אם זיהינו שהבדיקות הן צוואר הבקבוק, בואו נרכז את מאמצי השיפור שלנו והמשאבים שלנו כדי להשתפר בבדיקות, מתוך הבנה שזה שיפור שיתרום לנו בשורה התחתונה (בהשראת תורת האילוצים ע"פ אלי גולדרט). למשל, נשקיע בצורה רצינית באוטומציה. דוגמא אחרת – נמנע מצב בו בודקים מתחרים על משאבים במעבדת הבדיקות. מתחילים בכל ששואלים את בודק התוכנה איך אפשר לחסוך לו זמן או מה הכי מבזבז לו את הזמן.




אגב, אם הזכרנו כבר אוטומציה, אם נבנה על אנשי הבדיקות לבדם לכתוב את בדיקות האוטומציה מעל המערכת, אנחנו מסתבכים עוד יותר. לכתוב אוטומציה לוקח יותר מאשר להריץ בדיקות ידניות. החזר ההשקעה הוא בקיצור סבבי רגרסיה. הפתרון כאן הוא לחזור לגישת השלם – Whole Team Approach. האוטומציה והפרשי הקצבים הם אתגר של כל הצוות, או כל הקבוצה, וכנראה שנכון שהמפתחים יעזרו לכתוב אוטומציה כדי לאזן את הקצבים. אם המפתחים ישקיעו זמן בפיתוח אוטומציה, כנראה שהקצב שלהם יהיה נמוך יותר, ובהינתן אוטומציה המאפשרת קיצור סבבי רגרסיה המתבצעים בתדירות גבוהה יחסית באג'יל, כנראה שקצב הבדיקות יעלה. אם ניקח את האוטומציה ונחבר אותה לתהליך הנקרא Continuous Integration כלומר, ביצוע שוטף של אינטגרציה – קומפילציה, התקנה והרצת בדיקות כל יום או אפילו כל שעה או כל Checkin, בשלב שבו הבודק כבר לוקח גרסה, יש ודאות

אם נבנה על אנשי הבדיקות לבדם לכתוב את בדיקות האוטומציה מעל המערכת, אנחנו מסתבכים עוד יותר. לכתוב אוטומציה לוקח יותר מאשר להריץ בדיקות ידניות. החזר ההשקעה בקיצור סבבי רגרסיה. הפתרון כאן הוא לחזור לגישת השלם – Whole Team Approach. האוטומציה והפרשי הקצבים הם אתגר של כל הצוות, או כל הקבוצה, וכנראה שנכון שהמפתחים יעזרו לכתוב אוטומציה כדי לאזן את הקצבים

הרבה יותר גבוהה שהיא ראויה לבדיקות ולא "תישבר" כבר בהתחלה ותבזבז לו את הזמן.

שילוב המפתחים באוטומציה היא בעצם דוגמא להכפפה (Subordination) של שאר ההתמחויות/שלבים כדי לעזור לייעל את הבדיקות. עוד דוגמא היא ביצוע יותר בדיקות ברמת היחידה (Unit Testing) ע"י מפתחים כדי לצמצם כמות הבאגים והסבבים הנדרשים לערב את אנשי הבדיקות. מעבר לכך, כדאי בעת תכנון העבודה לחשוב על פירמידת האוטומציה. בסביבה אג'ילית אנחנו שואפים להסתמך כמה שיותר על בדיקות Unit, קצת פחות בדיקות ברמת ה- API (Acceptance Tests), ועוד הרבה פחות בדיקות ברמת ממשק המשתמש (GUI). בשאיפה, כ-5-10% בלבד מכיסוי הבדיקות יהיה דרך ה GUI. זאת בהנחה שאת הלוגיקה העסקית והפונקציונליות של המערכת ניתן לבדוק גם ללא מעבר דרך ממשק המשתמש. המעבר לפירמידת אוטומציה אג'ילית הוא אחד המפתחות להצלחה באוטומציה אשר מגיעה לכיסוי גבוה אבל גם ברת תחזוקה לאורך זמן. ככל שנצליח לדחוף יותר כיסוי לבסיס הפירמידה, יעלה לנו פחות לפתח ולתחזק את האוטומציה.

אחד החששות הכי נפוצים של ארגונים הנחשפים לצורך לאזן קצבים כחלק מה-Whole Team Approach הוא שאיזון הקצבים ימשוך את כל הארגון למטה, אל המכנה המשותף האיטי ביותר. חשש זה הוא לגיטימי. אם לצורך העניין נגדיר לצוות או לארגון שאסור לייצר פער ויש לצעוד לפי הקצב של צוואר הבקבוק, יש סיכוי טוב שגם ההתמחויות המהירות יותר "ישכחו" איך לרוץ מהר. הפתרון המומלץ הוא להגדיר בצורה מאוד ברורה את אוסף משימות ההכפפה ולצפות מהצוותים המהירים לנצל את היכולות העודפות שלהם לבצע כמה שיותר משימות הכפפה (לדוגמא אוטומציה), כך שבסופו של דבר הם לא מאיטים.

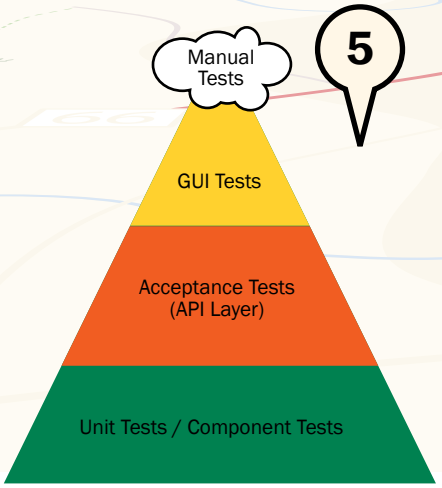
עוד אפשרות היא לשלב גישת ניהול סיכונים בבדיקות (Risk Based Testing). אפשר להגדיר שהתכולות/אזורים הפחות מסוכנים יבדקו ע"י מפתחים או יבדקו רק לקראת סוף הגרסה (כן, Late Feedback...) בעוד האזורים שבסיכון יבדקו מוקדם ככל האפשר ע"י בודקי התוכנה (Early Feedback). כך אפשר לייצר מסלול עוקף לבודקי התוכנה המתמודד עם בעיית הקצבים ומונע האטה של כל הארגון.

בשורה התחתונה, כבודקי תוכנה הנכנסים לעולם האג'יל, עלינו לזכור מספר דברים. קודם כל, מדובר במסע, בו לא כל הבעיות יפתרו בחודש הראשון ולא הכל יראה כמו בספרים. חשוב לזכור את העקרונות ולנסות לקבל כל הזמן החלטות מתוך העקרונות הללו. איך אנחנו יכולים להתקדם לקראת משוב מוקדם יותר (Early Feedback) על האזורים הכי מסוכנים? איך אנחנו יכולים לצמצם את כמות המלאי שיש לנו בתהליך – כמות הקוד שפותח ועדיין לא נבדק? בואו נצא מתוך נקודת הנחה שיש אמון ורצון

משותף להצליח, ונסכים לקבל עזרה המאפשרת לנו לעמוד באתגרים שלנו ולהתייעל. להבין ששווה לנו לפתוח את הקימונו?? ולהראות מה יכול לעזור לנו, לסמוך על זה שלא ינצלו את זה לרעתנו.

לבודקי תוכנה תפקיד מהותי מאוד בסביבה אג'ילית – הם עוזרים לייצג את הלקוח ואת ציפיותיו בצוות. גם אם חלק מהעבודה שנעשית היום ע"י הבודקים לא בהכרח נעשית על ידם בצוות אג'ילי – זה רק מאפשר להם לעבור מהגנה מגיבה (Reactive) אל ההתקפה הפרו-אקטיבית. מה זה מאפשר? השתתפות בהגדרת התכולה הפונקציונליות ומעורבות מוקדם יותר בתהליך – לדוגמא ע"י שיטה כגון Acceptance Test Driven Development. פינוי זמן לביצוע בדיקות Exploratory Testing מוקדם יותר, וזיהוי באגים מתקדמים יותר.

בודקי התוכנה שחברותיהם עברו לאג'יל מתחלקים לשתי קבוצות עיקריות: אלו שממנפים את המהלך הזה לקידום רמת הבדיקות בארגון והפכו בעצם לתומכים גדולים של המהלך, ואלו שעדיין מפחדים מהמהלך הזה והשלכותיו על תפקידם. התחלת השינוי היא בהכרת השיטה והכלים קצת יותר לעומק. מקווה שבמאמר זה סיפקתי הצצה לתוך עולם מרתק זה של בדיקות בסביבה אג'ילית.




**ביבליוגרפיה**


- Figure 1 Cost of Quality - Applied Software Measurement: Global Analysis of Productivity and Quality - Capers Jones 2
- http://www.slideshare.net/nashjain/role-of-qa-and-testing-in-agile-presentation 2
- http://yuvalyeret.com/2010/08/03/finding-the-right-dev-to-test-ratio-when-working-in-kanban/
- http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid
- http://www.slideshare.net/ehendrickson/agile-testing-u
- http://www.amazon.com/Agile-Testing-Practical-Guide-Testers/dp/0321534468