



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 כיף).
מיכאל מציג תכופות בכנסים בארץ ובח"ל ומלמד בדיקות תוכנה בפקולטה למדעי המחשב באוניברסיטה העברית. ניתן לראות חלק מהמצגות והמאמרים שלו באתר www.testprincipia.com



התיאוריה אומרת שבאגים הם יצורים חברתיים, ואוהבים לחיות אחד ליד השני, האמנם?

2. טכניקות בדיקה

לפני זמן לא רב נמצאה פירצה בלינוקס שמאפשרת להריץ קוד עם הרשאות של root גם למשתמש שאינו מורשה לעשות זאת (ראו בקישור) מעבר להנאה מבאג קריטי, יש (לי לפחות) הנאה גם מזה שהבאג מוכיח את התיאוריה שמאחורי בדיקות ערכי גבול (boundary value testing). על מנת לנצל את הפירצה צריך להריץ את הפקודה עם קביעת הערך של user ID כמינוס אחד (-1) או 4294967295 (שזה הערך הגבוה ביותר שניתן לייצג על ידי משתנה של 32 ביט). בקיצור - הבאג מופיע כשמתשתמים בערכי גבול.

התיאוריה של בדיקות תוכנה אומרת שיש לבדוק מקרי גבול כי שם יש סיכוי גבוה יותר למצוא באגים. שהרי ידוע שיש שתי סיבות עיקריות לבאגים בתוכנה: בעיות בדרישות, עיצוב

מסובך וטעויות של פספוס ב-1. עד כמה זה נכון? בכלל, איזה אחוז מהבאגים שגילינו נמצאו על ידי מקרי בדיקה שתוכננו לפי טכניקות הבדיקה המקובלות שאותם מלמדים בכל הקורסים?

עקרונית נראה שלא יהיה נורא מסובך לעשות את האנליזה הזאת. צריך להוציא את רשימת הבאגים ממערכת ניהול הבאגים, ולעבור עליהם אחד אחד...

אופס... אחד אחד, אמרת? יש כמה אלפי באגים בבסיס הנתונים. זה הולך לקחת הרבה מאוד זמן. אם באג נמצא על ידי תסריט בדיקה שנכתב מראש, והעיצוב שלו נעשה על ידי שימוש מכוון באחת מטכניקות הבדיקה, אז אולי יהיה קל לקבוע שהבאג התגלה על ידי טכניקת בדיקות ידועה. אני אומר אולי כי הרי לפעמים תופסים באג

שהבדיקה בכלל לא כיוונה אליו. למשל: הרצנו בדיקה לוודא ששדה מסוים מסוגל לקבל ערך טקסט באורך המקסימלי המותר של 512 תווים (כלומר, בדיקת מקרה גבול). הבדיקה נפלה, אבל בבחינה מדוקדקת יותר התברר שמה שגרם לנפילה זה בכלל לא אורך המחרוזת, אלא שלוקח זמן ארוך להכניס אותה פנימה (512 תווים!) וזה גורם ל-timeout. כל הרצה של בדיקה שבה היינו מתעכבים על השדה הזה מעל ל-5 שניות - ולא חשוב עם איזה אורך מחרוזת - היה גורם לבאג להופיע.

לגבי באגים שנמצאו בבדיקות חוקרות (exploratory testing) או "תוך כדי" בדיקת משהו אחר, נצטרך לעשות ממש אנליזה לעומק כדי לדעת איך היה צריך לתכנן מקרה בדיקה שהיה תופס את הבאג הזה ורק אז לסווג אותו לאחת מטכניקות הבדיקה הקיימות, אם הוא אכן מתאים למי מהן.



פרויקטים לפנסיה

כמו כל אחד (אני מניח), גם אני מתחזק כבר הרבה שנים רשימה של דברים שאני מתכוון לעשות פעם, כשיהיה לי זמן. אפשר למצוא ברשימה שמות של ספרים שאני רוצה לקרוא, נושאים שאני מתכוון ללמוד, מטלות ותיקונים בבית, סרטים שחובה לראות ובין השאר גם רשימה של רעיונות למחקר בתחום הבדיקות.

כיוון שבינתיים הרשימה הזאת רק מתארכת ואני לא מגיע לבצע ולו גם רעיון אחד, החלטתי לפרסם כאן חלק מהרעיונות, בתקווה שאולי מישהו אחר ירים את הכפפה.

1. התקבצות פגמים - Bug Clustering

התיאוריה אומרת שבאגים הם יצורים חברתיים ואוהבים לחיות אחד ליד השני. לכן, אם באיזור מסויים (יכולת או מודול) בקוד התגלו הרבה באגים, יש שם עוד הרבה באגים שלא התגלו, ושווה להשקיע שם מאמץ בדיקות נוסף. על פניו, זה דבר שממש קל לבדוק: בודקים את התפלגות הבאגים בכל מודול ורואים איפה יש יותר באגים. האמנם? אם המון באגים נמצאו בסבב בדיקות אחד ואחר כך היה שקט תעשייתי במודול הזה, אז להפך - הטענה הופרכה. כלומר, צריך לבדוק את ההתפלגות כל סבב בדיקות - ולראות האם לאורך זמן, מודול שיש בו הרבה באגים ממשיך לספק את הסחורה. אבל גם זה לא לגמרי פשוט. אם יש סבבים בהם מוצאים במודול הרבה באגים ובסבבים אחרים פחות, צריך לוודא שאנחנו משווים דברים זהים. יתכן שלפני סבב א' נעשו שינויים ותוספות למודול ואילו בין סבב א' ל-ב' נעשו רק תיקונים. אם כך, הרי צפוי שבסבב א' יתגלו יותר באגים. דבר נוסף שצריך לוודא הוא שבכל סבבי הבדיקות שאנו בוחנים הרצנו את אותם הבדיקות פחות או יותר (כלומר, שלא היו סבבים שבהם מסיבה של זמן, כוח אדם או אילוצים אחרים הרצנו רק חלק מהבדיקות). רצוי גם שאותם הבודקים בדיוק ישתתפו בכל סבב, כי הרי יש בודקים עם חוש שישי למצוא באגים ויש כאלה שפחות.

מפה לשם, מסתבר שלא כל כך פשוט להוכיח את העקרון.

אפשר אולי לעשות ניתוח סטטיסטי על כמות רבה של נתונים ולאורך זמן, מתוך כוונה לראות תופעות שחוזרות על עצמן באופן עקבי. למשל: לפני שנים עבדתי על מערכת משובצת מחשב (embedded system) שעיקר הקוד בה היה קושחה (firmware). במוצר הזה, לאורך ורסיות רבות, תמיד התקבלו הרבה דיווחים על באגים באותם שלושה מודולים - גם במעבדה וגם מהלקוחות. אהה! הוכחה ניצחת שבאגים מתקבצים! אז זהו - שלא בטוח. המודולים הבעייתיים היו תוכנת ההתקנה, ועוד שני מודולים שקשורים לאינטראקציה של המשתמש עם המערכת.

האם הסיבה לריבוי הבאגים היא שמערכת embedded די לא נגישה למשתמש (וגם לבודק) ולכן קל יותר להגיע למקרי קצה משונים באותם חלקים מעטים שבהם האינטראקציה קלה?

נראה שעל מנת לייצר ניסוי מדויק של עקרון התקבצות הפגמים צריך לארגן מצב לגמרי לא נורמלי: פרויקט תוכנה בלי לחצים, שיאפשר לתכנן את כל סבבי הבדיקות כך שיהיו פחות או יותר דומים מבחינת זמן, תוכן והמעורבים בפרויקט. פרויקט שבו כל הקוד נכתב לפני שמתחילים בדיקות, ואחר כך לא מוסיפים פונקציונאליות אלא רק מתקנים באגים, ותמיד על ידי אותם מפתחים. בקיצור: אין דבר כזה.

אם יש למישהו רעיון יעיל איך להוכיח את העיקרון, אשמח לשמוע!



האם יש לקבוצת פיתוח אופי, מבחינת הבאגים שהיא מייצרת?

בקיצור - גם כאן צריך להשקיע משאבי מחקר רבים. ומצד שני - זה די חשוב! כל קורס בדיקות וכל ספר בדיקות מלמד את טכניקות הבדיקה הבסיסיות האלה. עד כמה החיים מתאימים לתיאוריה? איזה טכניקה היא היעילה ביותר? איזה טכניקה נותנת פירות רק במקרים מסויימים? אני מכיר מחקר רחב היקף אחד שנעשה בתחום, על ידי ג'ימס וויטאקר. התוצאה שלו מסוכמת בספר המצוין "איך לשבור תוכנה" (How to Break Software, James Whittaker). רוב הטכניקות שספר זה מלמד הן פחות טריוויאליות מהטכניקות הבסיסיות והמקובלות שכולנו למדנו. חומר למחשבה...

עד כמה החיים מתאימים לתיאוריה?

3. פרופיל באגים

האם יש לקבוצת פיתוח "אופי" מבחינת הבאגים שהיא מייצרת? האם יש לפרויקטים מאפיין כזה?

נניח שאוכל להראות שקבוצת פיתוח מסויימת מתאפיינת בפרופיל הבאג: בפעם הראשונה שמקבלים מהם קוד לבדוק, הוא עובד ממש טוב וכמעט אין באגים. בהמשך, בכל פרויקט שמפותח על ידי קבוצה זו, מיד אחרי אלפא, כמות הבאגים מרקיעה שחקים ומגיעה לשיא עם שחרור ביתא. משם זה הולך ויורד עד ליום השחרור. אפשר כמובן לדון למה זה קורה ואיך לשפר - אבל לא זו הפואנטה כרגע. אם אפשר להראות שזה המצב פרויקט אחרי פרויקט, הרי שאם

נקבל פרויקט חדש שבו יהיו המון באגים לפני אלפא ומעט אחרי אלפא, נוכל מיד להבין שמשהו שונה קורה. משהו לא מתנהג כרגיל. אולי זה טוב ואולי לא - אבל כדאי להבין מה השתנה ולא להמתין לעוד הפתעות.

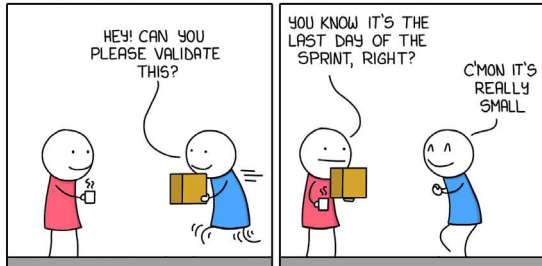
באותה מידה אפשר להכין פרופיל של "איך פרויקט מסוג X נראה" מבחינת הכמות הכוללת של באגים, איך הבאגים מתפלגים לרמות החומרה, ואיזה סוג פגמים מתגלה ומתי לאורך חיי הפרויקט. יתכן שנגלה שבאופן עקבי בפרויקטים של אפליקציות רשת, ללא תלות מי מפתח אותן, רוב הבאגים בהתחלה הם ברמת חומרה בינונית ומתקבצים באיזור ממשק המשתמש. בהמשך פרויקטים כאלה יש יותר באגים הקשורים לביצועים והם בדרך כלל חמורים יותר. לעומת זאת, בפרויקטים של מערכות משובצות מחשב המצב אחר - קודם מתגלים באגים של פונקציונאליות, ואחר כך באגים של אינטגרציה. בהינתן פרופיל כזה, הרי גם כאן אפשר לקבל התראה מוקדמת שמשהו לא עובד כרגיל אם פרופיל הבאגים (כמות, סוג, חומרה) לא נראה כמו בפרויקטים דומים. מעבר לזה - אם פרופילים כאלה יתפרסמו, חברות שונות יוכלו להשוות את הפרויקטים שלהם למה שאחרות חוות, ולנסות ללמוד משהו מההבדלים.

פירטתי כאן רק חלק מרשימת רעיונות המחקר שלי ואני מעודד אתכם לחשוב על עוד רעיונות. יש כמה דברים שאפשר לעשות עם הרעיונות האלה.

- א) לבצע אותם ולפרסם את התוצאות
- ב) לנסות לעניין סטודנטים להנדסת מערכות או תעשייה וניהול לקחת מחקר כזה כפרויקט ההנדסי שלהם
- ג) (ברירת המחדל...): להוסיף אותם לרשימה של הדברים שתעשו פעם, כשיהיה לכם זמן.

קדיחות

A QA WALKS INTO THE OFFICE



MONKEYUSER.COM

