בודק מוסמך
תוכנית לימודים לרמת הבסיס

מהדורה עברית 2018.01

מבוססת על מהדורה 2018 באנגלית

הארגון הישראלי להסמכת בודקי תוכנה

הארגון הבינלאומי להסמכת בודקי תוכנה

Copyright Notice *זכויות יוצרים*

עורכי המהדורה העברית מעבירים בזאת את הזכויות לארגון הישראלי להסמכת בודקי תוכנה (ITCB). העורכים ו-ITCB הסכימו על תנאי השימוש במהדורה העברית כדלהלן:

1. כל אדם או חברת הדרכה יכולים להשתמש במהדורה עברית זו כבסיס לקורס הדרכה אם עורכי המהדורה העברית, ITCB ו-ISTQB מוזכרים בתור המקור ובעלי זכויות היוצרים של הסילבוס העברי. פרסום לקורס כזה יכול להזכיר את הסילבוס ואת ארגוני ITCB ו-ISTQB רק לאחר שחומרי ההדרכה נמסרו ל-ITCB לשם הסמכה רשמית
2. כל אדם או קבוצה יכולים להשתמש במהדורה עברית זו כבסיס למאמרים, ספרים או דרכי חיבור אחרות, אם עורכי המהדורה העברית, ITCB ו-ISTQB מוזכרים כמקור וכבעלי זכויות היוצרים של המהדורה העברית.

היסטוריית גרסאות

גרסה	תאריך	הערות
2018.01	1 פברואר 2019	גרסה ראשונה (2018) לתרגום העברי המלא של סילבוס הרמה הבסיסית 2018

Revision History

Version	**Date**	**Remarks**
ISTQB 2018	Effective 4-Jun-2018	Certified Tester Foundation Level syllabus
ISTQB 2011	Effective 1-Apr-2011	Certified Tester Foundation Level Syllabus Maintenance Release – see Appendix E – Release Notes
ISTQB 2010	Effective 30-Mar-2010	Certified Tester Foundation Level Syllabus Maintenance Release – see Appendix E – Release Notes
ISTQB 2007	01-May-2007	Certified Tester Foundation Level Syllabus Maintenance Release
ISTQB 2005	01-July-2005	Certified Tester Foundation Level Syllabus
ASQF V2.2	July-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25-Feb-1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999

תוכן העניינים

תודות

אלישבע הרשלר, שביצעה את התרגום הבסיסי של מהדורת 2011 במסירות רבה; אבי עופר על תרגום המונחים.

טל פאר, שביצע את התרגום למהדורת 2018 ולחברי הוועד המייעץ של ITCB על הגהת התרגום: מיכאל שטאל, יונית אלבז, אייל זילברמן, נועם כפיר, עומר פיליפ, עופר פלדמן, מור שאול, דקר שלום וגיל שקל

The document was produced by a team from the International Software Testing Qualifications Board: Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, and Eshraka Zakaria,

The team thanks Rex Black and Dorothy Graham for their technical editing, and the review team, the cross-review team, and the Member Boards for their suggestions and input.

The following persons participated in the reviewing, commenting and balloting of this syllabus: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliveira, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, and Karolina Zmitrowicz.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, and Stevan Zivanovic. The core team thanks the review team and all Member Boards for their suggestions.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2011): Thomas Müller (chair), Debra Friedenberg. The core team thanks the review team (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) and all National Boards for the suggestions for the current version of the syllabus.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2010): Thomas Müller (chair), Rahul Verma, Martin Klonk and Armin Beer. The core team thanks the review team (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) and all National Boards for their suggestions.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2007): Thomas Müller (chair), Dorothy Graham, Debra Friedenberg, and Erik van Veenendaal. The core team thanks the review team (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, and Wonil Kwon) and all the National Boards for their suggestions.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2005): Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veenendaal and the review team and all National Boards for their suggestions

0 הקדמה לתוכנית לימודים זו

0.1 מטרת המסמך

תוכנית לימודים זו מהווה את הבסיס להסמכה הבינלאומית לבדיקת תוכנה ברמת הבסיס. התוכנית נמסרת על ידי הארגון הבינלאומי להסמכת בודקי תוכנה (®ISTQB)

1. לוועדים המקומיים, לשם תרגום התוכנית לשפה המקומית והסמכת ספקי הדרכה. הוועדים המקומיים רשאים להתאים את תוכנית הלימודים לצרכי השפה המקומית ולשנות את ההפניות לשם התאמה לפרסומים מקומיים.
2. לגופי הסמכה, לשם הפקת שאלות בחינה בשפה המקומית, מותאמות ליעדים של תוכנית הלימוד.
3. לספקי הדרכה, לשם הפקת מערכי הדרכה ובחירת שיטות הוראה מתאימות.
4. למועמדים לבחינת ההסמכה, לשם הכנה לבחינה ההסמכה (כחלק מקורס הכנה או בלימוד עצמאי).
5. לקהילת מהנדסי התוכנה והמערכת הבינלאומית, לשם קידום מקצוע בדיקות התוכנה והמערכת, וכבסיס לספרים ומאמרים.

®ISTQB רשאי לאפשר לגופים אחרים להשתמש בתוכנית הלימודים לצרכים אחרים, לאחר שקיבלו אישור מראש בכתב מ-®ISTQB.

מידע אודות הרקע למסמך ניתן למצוא בנספח א'.

הערה א: בכל מקום בו יש אי-התאמה בין התרגום העברי למקור האנגלי, המקור האנגלי הוא הקובע.

הערה ב: כל הניסוחים במסמך הם בלשון זכר למניעת סרבול, אבל הם מופנים במידה שווה לנשים ולגברים, לבודקות ולבודקים.

0.2 בודק מוסמך לבדיקות תוכנה ברמת הבסיס

ההסמכה ברמת הבסיס מיועדת לכל מי שמעורב באופן כלשהו בבדיקות תוכנה. קהל היעד כולל בודקים, מנתחי בדיקות, מהנדסי בדיקות, יועצי בדיקות, מנהלי בדיקות, בודקי קבלת המוצר ומפתחי תוכנה. הסמכה בסיסית זו מתאימה גם לכל המעוניינים לרכוש הבנה בסיסית של תחום בדיקות התוכנה, כמו למשל מנהלי פרויקטים, מנהלי איכות, מנהלי פיתוח, מנתחים עסקיים, מנהלי IT ויועצי ניהול. בעלי תעודת הסמכה ברמת הבסיס יוכלו להמשיך לדרגת הסמכה מתקדמת יותר.

ISTQB Foundation Level Overview 2018 הוא מסמך נפרד שמכיל את המידע הבא:

- תחומי הידע[1] הנלמדים על פי הסילבוס
- מטריצה המציגה נֶעֱקָבוּת בין תחומי הידע ויעדי הלימוד
- תקציר של הסילבוס

(מסמך זה לא תורגם לעברית)

[1] המקור האנגלי משתמש כאן במושג business outcomes. לא מצאנו תרגום הולם המשלב את המילה עסק/עסקי מבלי שיתפרש כמטרה כלכלית. בחרנו על כן לתרגם מושג זה כ"תחומי הידע"

0.3 יעדי לימוד / רמת ידע קוגניטיבית

יעדי הלימוד משמשים להפקת בחינות לצרכי הסמכת הרמת הבסיסית.

ככלל, כל התוכן בתוכנית הלימודים יכול להיכלל בבחינה ברמת K1, מלבד ההקדמה והנספחים. כלומר, הנבחן יתבקש לזהות ולזכור מילות מפתח או מושגים שמוזכרים בכל אחד מששת הפרקים. יעדי הלימוד מוצגים בתחילת כל פרק ומסווגים כדלהלן:

- K1: לזכור
- K2: להבין
- K3: ליישם

פרטים נוספים ודוגמאות ליעדי לימוד ניתן למצוא בנספח ב'.

התלמידים נדרשים לזכור (K1) את כל המושגים המופיעים תחת הכותרת "מושגים" בכל פרק, אפילו אם המושג לא מצוין במפורש ביעדי הלימוד.

0.4 בחינת ההסמכה לרמת הבסיס

בחינת ההסכמה לרמת הבסיס תהיה מבוססת על תוכנית לימודים זו. יתכן שהתשובות לשאלות הבחינה ידרשו שימוש בחומר לימוד המופיע ביותר מיחידה אחת בתוכנית זו. כל יחידות התוכנית נכללות בבחינה מלבד ההקדמה והנספחים. תוכנית הלימודים כוללת התייחסות והפנייה לתקנים וספרים, אך התוכן של אלה אינו נכלל בבחינה, מעבר למה שנכלל בתוכנית הלימודים.

הבחינה בנויה במתכונת של שאלון רב-ברירתי (בחינה אמריקאית), וכוללת 40 שאלות. כדי לעבור את הבחינה, יש לענות נכון של לפחות 65% מהשאלות (כלומר, 26 שאלות).

ניתן לגשת לבחינה ההסכמה במסגרת קורס הכשרה מוכר או באופן עצמאי (למשל במרכז בחינות או במבחן ציבורי). השלמת קורס הכשרה מוכר אינה תנאי מוקדם לצורך זכאות לגשת לבחינה.

0.5 הסמכת ספקי הדרכה

הוועד המקומי של ISTQB® (בישראל, ארגון ITCB®) רשאי להסמיך ספקי הדרכה אשר תוכן הקורס שלהם מבוסס על תוכנית הלימודים הזו. על ספקי ההדרכה לקבל הנחיות זכאות להסמכה מ-ITCB®. קורס מוסמך הוא כזה שתואם את תוכנית הלימודים הזו, ורשאי להגיש את הלומדים בו לבחינת ההסמכה של ISTQB® כחלק מהקורס.

0.6 רמת הפירוט

רמת הפירוט של תוכנית הלימודים הזו מאפשרת אחידות בינלאומי בהוראת הקורסים ובבחינה. כדי להשיג מטרה זו, תוכנית הלימודים כוללת:

- יעדי הוראה כלליים המתארים את מטרות רמת הבסיס
- רשימת מושגים שעל התלמידים לזכור
- יעדי לימוד לכל תחום ידע, המתארים את תוצאת הלימוד הקוגניטיבי המבוקש
- תיאור של מושג יסוד, כולל מקורות מהספרות המקצועית המקובלת או מתקנים בינלאומיים.

תוכן תוכנית הלימודים אינו תיאור של כל גוף הידע בבדיקות תוכנה אלא משקף רמת פירוט שיש להקיף בקורסי הדרכה לרמת הבסיס. התוכנית מתמקדת במושגים וטכניקות של בדיקות שניתן ליישם בכל פרויקט תוכנה, כולל פרויקטים במתודולוגית אג'ייל. תוכנית הלימודים לא כוללת יעדי לימוד ייחודיים למחזור חיי תוכנה או מתודולוגית פיתוח כלשהם, אך כן דנה ביישום העקרונות האלה בפרויקטים

במתודולוגית אג'ייל, במודלים של פיתוח בסבבים (iterative), מודלים מצטברים (incremental) ומודלים של פיתוח סדרתי (sequential).

0.7 מבנה תוכנית הלימודים

בתוכנית שישה פרקים עיקריים. הכותרת הראשית של כל פרק מציגה את הזמן שיש להקדיש להוראת הפרק. תוכנית הלימוד דורשת מקורסים מוסמכים להעביר 16.75 שעות של הדרכה, מחולקים בין ששת הפרקים כדלהלן:

- פרק 1: 175 דקות, יסודות בדיקות תוכנה
- פרק 2: 100 דקות, בדיקות לאורך מחזור חיי התוכנה
- פרק 3: 135 דקות, בדיקות סטטיות
- פרק 4: 330 דקות, טכניקות לעיצוב בדיקות
- פרק 5: 225 דקות, ניהול בדיקות
- פרק 6: 40 דקות, כלים תומכי בדיקות

1 יסודות בדיקות התוכנה — 175 דקות

מושגים

כיסוי (coverage), ניפוי באגים (debugging), פגם (defect), שגיאה (error), כשל (failure), איכות (quality), אבטחת איכות (quality assurance), שורש הבעיה (root cause), ניתוח בדיקות (test analysis), בסיס בדיקות (test basis), מקרה בדיקה (test case), השלמת הבדיקות (test completion), מצב בדיקה (test condition), בקרת בדיקות (test control), נתוני בדיקות (test data), עיצוב בדיקות (test design), ביצוע בדיקות (test execution), לוח זמנים לביצוע הבדיקות (test execution schedule), יישום מערך בדיקות (test implementation), ניטור בדיקות (test monitoring), מושא הבדיקות (test object), מטרת הבדיקות (test objective), אורקל הבדיקות (test oracle), תכנון בדיקות (test planning), הליך בדיקה (test procedure), סדרת בדיקות (test suite), בדיקות (testing), מכלול מרכיבי הבדיקות (בוֹדְקָה-testware), נֶעֱקָבוּת (traceability), תיקוף (validation), אימות (verification)

יעדי הלימוד של פרק יסודות בדיקות התוכנה

1.1 בדיקות, מהן?

FL-1.1.1 (K1) פרט את מטרות הבדיקה הנפוצות

FL-1.1.2 (K2) הבחן בין בדיקות לניפוי באגים

1.2 מדוע הבדיקות נחוצות?

FL-1.2.1 (K2) הסבר באמצעות דוגמאות מדוע הבדיקות נחוצות

FL-1.2.2 (K2) הסבר את היחסים בין בדיקות ואבטחת איכות והבא דוגמאות לאופן בו בדיקות תורמות לרמת איכות גבוהה יותר

FL-1.2.3 (K2) הבחן בין שגיאה, פגם וכשל

FL-1.2.4 (K2) הבחן בין שורש הבעיה הגורמת לפגם לבין השפעותיו של הפגם

1.3 שבעת עקרונות הבדיקה

FL-1.3.1 (K2) הסבר את שבעת עקרונות הבדיקה

1.4 תהליך הבדיקות

FL-1.4.1 (K2) הסבר את השפעת ההקשר של התוכנה על תהליך הבדיקות

FL-1.4.2 (K2) תאר את פעילויות הבדיקה ואת המטלות הקשורות אליהן במסגרת תהליך הבדיקות

FL-1.4.3 (K2) הבחן בין התוצרים השונים שתומכים בתהליך הבדיקות

FL-1.4.4 (K2) הסבר את החשיבות בשמירה על נֶעֱקָבוּת בין בסיס הבדיקות ותוצרי הבדיקות

1.5 הפסיכולוגיה של הבדיקות

FL-1.5.1 (K1) זהה את הגורמים הפסיכולוגיים המשפיעים על הצלחת הבדיקות

FL-1.5.2 (K2) הסבר את ההבדל בין דפוס החשיבה הנדרש לפעילויות בדיקה לבין דפוס החשיבה הנדרש לפעילויות פיתוח

1.1 בדיקות, מהן?

מערכות תוכנה הן חלק בלתי נפרד מחיינו, החל מיישומים עסקיים (כגון בנקאות) וכלה במוצרי צריכה (כגון מכוניות). מרבית האנשים התנסו בתוכנה שלא פעלה כמצופה. תוכנה שלא פועלת כראוי עשויה לגרום לבעיות רבות, כולל הפסד של כסף, זמן או מוניטין, ואף פציעות וגרימת מוות. בדיקות תוכנה הן אמצעי להערכת יכולת התוכנה ולהפחתת הסיכון לכישלון התוכנה בזמן פעולתה.

תפיסה שגויה ונפוצה של בדיקות אומרת שהן כוללות רק הרצה של בדיקות, כלומר הפעלת התוכנה ובדיקת התוצאות. זוהי רק אחת מבין שלל פעילויות הבדיקות. כפי שיתואר בפרק 1.4, בדיקות תוכנה הן תהליך שכולל מגוון פעילויות שונות, כשביצוע בדיקות (כולל בדיקה של התוצאות) היא רק אחת מפעילויות אלה. תהליך הבדיקה כולל פעילויות כגון תכנון בדיקות, ניתוח, עיצוב ויישום מערך הבדיקות, דיווח על התקדמות הבדיקות ועל תוצאות הבדיקה והערכת איכות מושא הבדיקות.

ישנן בדיקות שמפעילות את המערכת הנבדקת או חלק ממרכיביה; בדיקות אלה נקראות בדיקות דינמיות (dynamic testing). פעילויות אחרות אינן מפעילות את המערכת הנבדקת או רכיבים ממנה. למשל: סקירות של תוצרי עבודה כמו דרישות, סיפורי משתמש וקוד מקור. פעילויות אלה נקראות בדיקות סטטיות (static testing) וגם הן נכללות במושג "בדיקות".

תפיסה שגויה נוספת אומרת שבדיקות מתמקדות אך ורק באימות של הדרישות, סיפורי המשתמש ומאפיינים אחרים. בדיקות אכן בודקות שהמערכת עונה על דרישות מסוימות, אך הבדיקות כוללות גם תיקוף, כלומר בדיקה שהמערכת עונה על צרכי המשתמש ובעלי עניין נוספים בסביבה התפעולית בה היא פועלת.

פעילויות בדיקה מאורגנות ונערכות בצורה שונה במחזורי חיים שונים (ראו פרק 2.1).

1.1.1 *מטרות אופייניות של בדיקות*

לכל פרויקט נתון, מטרות הבדיקות יכולות לכלול:

- הערכת תוצרי עבודה כמו דרישות, סיפורי משתמש, עיצוב (design) וקוד
- אימות שכל הדרישות ממומשות
- תיקוף שלמות מושא הבדיקות ווידוא שהמוצר עובד בדרך בה המשתמשים ובעלי עניין אחרים מצפים שיעבוד
- בניית אמון ברמת האיכות של מושא הבדיקות
- מניעת פגמים
- מציאת כשלים ופגמים
- אספקת מידע מספק לבעלי העניין שיאפשר להם לקבל החלטות מושכלות, בייחוד בנוגע לרמת האיכות של מושא הבדיקות
- הקטנת רמת הסיכון לאיכות התוכנה (כגון, הפחתת הסיכוי להופעת כשלים בזמן תפעול המערכת)
- וידוא שמושא הבדיקות עומד בדרישות חוזיות, בדרישות חוקיות או דרישות המופיעות בתקנים של התעשייה

מטרות הבדיקות עשויות להשתנות בהתאם להקשר של התוכנה או המערכת הנבדקת, לרמת הבדיקות, ולמודל מחזור חיי התוכנה (SDLC – Software Development Lifecycle). הבדלים אלה עשויים לכלול, למשל:

- במהלך בדיקות רכיבים (Component Testing) מטרה מרכזית אחת תהיה למצוא כמה שיותר כשלים על מנת שפגמים בסיסיים יזוהו ויתוקנו מוקדם. מטרה נוספת עשויה להיות העלאת כיסוי הקוד של בדיקות הרכיבים.
- במהלך בדיקות קבלה (Acceptance Testing) מטרה מרכזית אחת תהיה לוודא שהמערכת עובדת כמצופה ועונה על הדרישות. מטרה נוספת של בדיקות אלה עשויה להיות אספקת מידע לבעלי העניין בנוגע לרמת הסיכון של שחרור המערכת בזמן נתון.

1.1.2 *בדיקות וניפוי באגים*

בדיקות וניפוי באגים (debugging) הן שתי פעולות שונות זו מזו. הרצת בדיקות עשויה למצוא כשלים אשר נגרמים על ידי פגמים בתוכנה. ניפוי באגים הינה פעילות פיתוח שמוצאת, מנתחת ומתקנת פגמים אלה. בדיקות אימות (confirmation testing) בודקות אם התיקונים אכן פתרו את הכשלים. במקרים מסוימים הבודקים אחראים לבדיקות הראשוניות ולבדיקות האימות הסופיות, בעוד שהמפתחים עושים את ניפוי הבאגים ואת בדיקות הרכיבים. אולם בפיתוח במתודולוגית אג'ייל ובמודלים אחרים, בודקים עשויים להיות מעורבים בניפוי הבאגים ובבדיקות הרכיבים.

תקן ISO (ISO/IEC/IEEE 29119-1) מספק מידע נוסף על מושגים בבדיקות תוכנה.

1.2 מדוע הבדיקות נחוצות?

בדיקות קפדניות של תוכנה, מערכות והמסמכים הקשורים אליהן, עשויות לסייע בהפחתת הסיכון להופעת כשלים בזמן הפעלת המערכת. איכות התוכנה או המערכת עולה כשפגמים נמצאים ומתוקנים. בנוסף, בדיקות תוכנה עשויות להידרש על מנת לענות על דרישות חוזיות או חוקיות, או לעמוד בתקנים ייחודיים לתעשייה.

1.2.1 *תרומת הבדיקות להצלחה*

לאורך כל ההיסטוריה של המיחשוב, תוכנה ומערכות נמסרות לשימוש כשהן מכילות פגמים, ועקב כך מתגלים כשלים במערכות או שהמערכות לא עומדות בדרישות בעלי העניין. שימוש בטכניקות בדיקה מתאימות עשוי להפחית את תדירות המקרים שבהן מערכות עם כשלים משוחררות למשתמשים, במידה וטכניקות אלה מיושמות במומחיות הנכונה, ברמות הבדיקה המתאימות, ובשלבים הנכונים במחזור חיי התוכנה. דוגמאות לכך יכולות להיות:

- בודקים המשתתפים בסקירה של דרישות או סיפורי משתמש עוזרים בזיהוי פגמים הקיימים בהם. זיהוי והסרה של פגמים בדרישות יפחית את הסיכון שהתוכנה תממש יכולות שלא נדרשו, יכולות שגויות, או כאלה שלא ניתנות לבדיקה.
- שילוב בודקים בעבודה עם מתכנני המערכת בשלב העיצוב יגדיל את ההבנה של שני הצדדים בעיצוב המערכת ואיך לבדוק אותה. הבנה זו תפחית את הסיכון לפגמים בסיסיים בעיצוב המערכת ותאפשר לזהות את הבדיקות הנדרשות בשלב מוקדם.
- שילוב בודקים בעבודה עם המפתחים יגדיל את ההבנה של שני הצדדים בקוד ואיך לבדוק אותו. הבנה זו תפחית את הסיכון לפגמים בקוד ובבדיקות.
- אימות ותיקוף שיעשה על ידי הבודקים לפני שהמערכת נמסרת, עשוי לגלות כשלים שלא התגלו בדרכים אחרות, ובכך לסייע בהסרת הפגמים שגרמו את הכשלים הללו (כלומר ניפוי באגים). בדיקות אלה יגדילו את הסבירות שהתוכנה תענה על צרכי בעלי העניין ותעמוד בדרישות.

נוסף על דוגמאות אלה, השגת מטרות הבדיקות המוגדרות (ראו פרק 1.1.1) תורם להצלחה הכללית של פיתוח ותחזוקת המערכת.

1.2.2 *אבטחת איכות ובדיקות*

אנשים נוטים להשתמש במושג *אבטחת איכות* (או QA) כשהם מדברים על בדיקות, אבל אבטחת איכות ובדיקות אינן אותו הדבר, למרות שהמושגים קשורים זה לזה. מה שמקשר ביניהם הוא המושג הגדול ניהול איכות (quality management), שכולל בתוכו את כל הפעילויות שמכוונות ומנהלות ארגונים בכל מה שנוגע לאיכות. בין השאר, ניהול איכות כולל גם אבטחת איכות (quality assurance) וגם בקרת איכות (quality control). אבטחת איכות בדרך כלל מתמקדת בעבודה על פי תהליכים מתאימים, שמטרתם להעלות את הסיכוי שרמת האיכות המתאימה תושג. כאשר עובדים נכון לפי התהליכים, מתקבלים בדרך כלל תוצרים באיכות גבוהה יותר, דבר התורם למניעת היווצרות פגמים. בנוסף לכך, ניתוח שורש הבעיה על מנת לזהות ולתקן את הגורמים לפגמים, יחד עם שיפור תהליכים באמצעות יישום הממצאים שעלו בישיבות הפקת לקחים (retrospective meetings), תורמים לאבטחת איכות יעילה.

בקרת איכות כוללת פעילויות שונות, ובכללן פעילויות בדיקה, שתומכות בהשגת רמה מתאימה של איכות. פעילויות הבדיקה הן חלק מתהליך כולל של פיתוח תוכנה ותחזוקת תוכנה. מאחר ואבטחת איכות עוסקת ביישום נכון של כל התהליך, הרי שאבטחת איכות כוללת ביצוע של בדיקות במידה הנדרשת. כמתואר בפרקים 1.1.1 ו-1.2.1, הבדיקות תורמות לאיכות במגוון של דרכים.

1.2.3 *שגיאות, פגמים וכשלים*

אדם יכול לשגות (לעשות טעות), ובכך לגרום לפגם (ליקוי או באג) בקוד התוכנה או בתוצר קשור אחר, כמו מסמך. שגיאה שתגרום לפגם בתוצר אחד עשויה לגרום לשגיאה שתגרום לפגם בתוצר קשור. למשל, שגיאה בזיהוי דרישה עשויה לגרום לפגם בדרישה, שבהמשך תגרום לשגיאה בכתיבת הקוד, דבר שיגרום לפגם בקוד.

אם הפגם בקוד יבוצע, הדבר עשוי לגרום לכשל, אם כי לא בכל המקרים. לדוגמה, ישנם פגמים שדורשים קלט ייחודי או תנאים מקדימים ייחודיים על מנת שיגרמו לכשל, כך שהכשל עשוי לא להופיע או להופיע במקרים נדירים.

שגיאות עשויות לקרות מהרבה סיבות, לדוגמה:

- לחץ של זמן
- טעות אנוש
- עובד לא מיומן או לא מנוסה
- בעיות תקשורת בין אנשים בפרויקט, כולל בעיות תקשורת הנוגעות לדרישות ולעיצוב
- סיבוכיות של הקוד, העיצוב, הארכיטקטורה, הבעיה היסודית אותה באים לפתור ו/או הטכנולוגיה בה משתמשים
- חוסר הבנה בנוגע לממשקים התוך-מערכתיים ובין-מערכתיים, בייחוד כאשר יש מספר גבוה של ממשקים כאלה
- טכנולוגיה חדשה ולא מוכרת

בנוסף לכשלים שנגרמים עקב פגמים בקוד, כשלים עשויים להיגרם גם על ידי תנאי הסביבה. למשל: קרינה, שדות אלקטרו-מגנטיים וזיהום יכולים לגרום לפגמים בקושחה (firmware) או להשפיע על ביצועי התוכנה על ידי שינוי בתנאים בהם החומרה פועלת.

לא כל תוצאות הבדיקה הלא צפויות הן כשלים. לעיתים ניתקל בתוצאה חיובית כוזבת (false positive) שנגרמת עקב שגיאות בדרך בהן הבדיקות מבוצעות, או פגמים בנתוני הבדיקות, בסביבת הבדיקות או בבודקה (testware), או מסיבות אחרות. המצב ההפוך גם יכול לקרות, בו שגיאות דומות או פגמים

יובילו לתוצאה שלילית כוזבת (false negative). תוצאה שלילית כוזבת מתקבלת כאשר בדיקות לא מזהות פגמים שהבדיקות היו אמורות לזהות; תוצאה חיובית כוזבת מתקבלת כאשר מדווחים פגמים שאינם באמת פגמים.

1.2.4 *פגמים, שורש הבעיה וההשפעות שלהם*

שורש הבעיה של פגמים הוא הגורם המוקדם ביותר שהביא ליצירת הפגם. ניתוח הפגמים לזיהוי שורש הבעיה יביא להפחתה בפגמים דומים בעתיד ויוביל לשיפור תהליכים שימנעו יצירת פגמים.

לדוגמה, נניח שלקוח מתלונן על חישוב ריבית שגוי, שנגרם עקב שורת קוד שגויה אחת. הקוד השגוי נכתב עבור סיפור משתמש שהיה לא ברור, דבר שנגרם עקב הבנה שגויה של מנהל המוצר מהי הדרך הנכונה לחישוב ריבית. אם קיים מספר רב של פגמים בחישובי ריבית, ושורש הבעיה של הפגמים האלה הוא הבנה שגויה, ניתן ללמד את מנהלי המוצר את נושא חישוב הריביות ובכך לצמצם את מספר הפגמים העתידיים.

בדוגמה הזו, תלונות הלקוח הן ההשפעה. תשלומי הריבית השגויים הם הכשלים. החישוב הלא נכון בקוד הוא הפגם, שנגרם מהפגם המקורי שהיה סיפור המשתמש הלא ברור. שורש הבעיה של הפגם המקורי היה חוסר ידע מצדו של מנהל המוצר, שגרם לו לשגות בכתיבת סיפור המשתמש. תהליך ניתוח שורש הבעיה נידון בהרחבה בתוכנית הלימודים לרמת מומחה – מנהל בדיקות (ISTQB-ETM, Expert Level Test Manager) ובתוכנית הלימודים לרמת מומחה – שיפור תהליך הבדיקות (ISTQB-EITP, Expert Level Improving the Test Process).

1.3 שבעת עקרונות הבדיקה

במהלך 50 השנים האחרונות הוצעו מספר עקרונות בדיקה שכללו עקרונות מנחים משותפים לכל סוגי הבדיקות.

1. הבדיקות מצביעות על נוכחות של פגמים, לא על היעדרם (presence of defects)

בדיקות יכולות להצביע על נוכחותם של פגמים, אך אינן יכולות להוכיח את העדרם של פגמים. בדיקות מפחיתות את הסיכוי שיישארו פגמים חבויים בתוכנה, אך גם אם לא נמצאו פגמים, בדיקות אינן הוכחת תקינות.

2. בלתי אפשרי לבצע בדיקות ממצות (exhaustive testing is impossible)

בדיקה של הכל (כל צירופי הקלט והתנאים המוקדמים) אינה ברת-ביצוע, למעט במקרים טריוויאליים. במקום לנסות לבצע בדיקות ממצות, יש לעשות שימוש בניתוח סיכונים (risk analysis) ובטכניקות לעיצוב בדיקות, ולהעמיד סדרי עדיפויות על מנת למקד את מאמץ הבדיקה.

3. בדיקות מוקדמות חוסכות זמן וכסף (early testing)

על מנת להקדים ולמצוא פגמים, יש להתחיל פעילות בדיקות דינמיות וסטטיות מוקדם ככל האפשר במחזור חיי התוכנה. בדיקות מוקדמות נקראות לעיתים הזזה לשמאל (shift left). שילוב הבדיקות מוקדם במחזור חיי התוכנה עוזר להפחית ואף למנוע שינויים יקרים (ראו פרק 3.1).

4. התקבצות פגמים (defect clustering)

בדרך כלל מספר קטן של מודולים מכיל את רוב הפגמים המתגלים במהלך בדיקות טרום שחרור, או אחראי למרבית הכשלים התפעוליים. הצפי להתקבצות פגמים, כמו גם הצפיפות הנצפית בפועל בבדיקות או בתפעול המערכת, הינם קלט חשוב לניתוח סיכונים שנעשה על מנת למקד את מאמץ הבדיקות (כמצוין בעקרון 2).

5. היזהר מעקרון ההדברה (pesticide paradox)

אם חוזרים על אותן בדיקות שוב ושוב, בסופו של דבר בדיקות אלה לא ימצאו שום פגמים חדשים. כדי למצוא פגמים חדשים, יש לשנות את הבדיקות הקיימות ואת נתוני הבדיקה, ואולי אף לכתוב בדיקות חדשות. במקרים מסוימים, למשל במקרה של בדיקות נסיגה (regression tests) אוטומטיות, עקרון ההדברה הוא חיובי, כיון שאנו מצפים שכל הבדיקות יעברו. כשל בבדיקות אלה מצביע על כך שקרתה רגרסיה בתוכנה.

6. בדיקות הן תלויות הקשר (context dependent)

בדיקות נעשות באופן שונה, על פי ההקשר. למשל, תוכנת בקרה תעשייתית למערכת חיונית (safety critical) תיבדק באופן שונה מיישומון המיועד למסחר אלקטרוני. כדוגמה נוספת, בדיקות בפרויקט אג'ייל נעשות באופן שונה מאשר בדיקות בפרויקט עם מחזור חיים סדרתי (sequential) (ראו פרק 2.1).

7. העדר שגיאות הוא סברה מוטעית (absence-of-errors is a fallacy)

ישנם ארגונים המצפים שהבודקים יריצו את כל הבדיקות האפשרויות וימצאו את כל הפגמים האפשריים, אבל עקרונות 2 ו-1, בהתאמה, אומרים לנו שזה בלתי אפשרי. מעבר לכך, זו סברה מוטעית לצפות שרק זיהוי ותיקון מספר רב של פגמים יבטיח את הצלחת המערכת. לדוגמה, למרות שבוצעו בדיקות יסודיות של כל הדרישות שנכתבו ותוקנו כל הפגמים שנמצאו, עדיין נוצרה מערכת קשה לתפעול, שאינה עונה לצרכים ולציפיות של המשתמשים, או שהיא נחותה בהשוואה למערכות מתחרות.

ראו דוגמאות לעקרונות הללו ולעקרונות בדיקה נוספים אצל Myers 2011, Kaner 2002 ו- Weinberg 2008.

1.4 תהליך הבדיקות

אין בנמצא תהליך בדיקות תוכנה אוניברסלי, אבל ישנן מספר פעילויות בדיקה מקובלות בלעדיהן הסיכוי של הבדיקות להשיג את מטרותיהן יהיה קטן יותר. פעילויות אלה מהוות תהליך בדיקות. תהליך הבדיקות הנכון, הייחודי לכל מצב נתון, תלוי בהרבה גורמים. אילו פעילויות בדיקה מעורבות בתהליך הבדיקה הזה, איך פעילויות אלה מיושמות, ומתי פעילויות אלה יתרחשו – כל זה נידון באסטרטגיית הבדיקות (test strategy) של הארגון.

1.4.1 תהליך הבדיקות בהקשר

גורמים המשפיעים על תהליך הבדיקות עבור ארגון, כוללים, למשל, את:

- מודל מחזור חיי התוכנה ומתודולוגיות ניהול הפרויקט בהם משתמשים
- רמות הבדיקה (test levels) וסוגי הבדיקה (test types) האפשריים
- סיכוני המוצר וסיכוני הפרויקט
- התחום העסקי בה פועלת החברה (business domain)
- מגבלות תפעוליות, כולל אך לא מוגבל ל:
  - תקציב ומשאבים
  - לוחות זמנים
  - סיבוכיות
  - דרישות חוזיות ודרישות תקינה
- מדיניות ונהלי הארגון

- תקנים פנימיים וחיצוניים

רשימה זו אינה ממצה ויתכנו גורמים משפיעים נוספים.

הסעיפים הבאים מתארים היבטים כלליים של תהליכי הבדיקה הארגוניים מנקודות המבט הבאות:

- פעילויות בדיקה ומטלות בדיקה
- תוצרי עבודת הבדיקות
- נָעֲקְבוּת בין בסיס הבדיקות ותוצרי הבדיקות

הגדרת קריטריון כיסוי[2] בר-מדידה לבסיס הבדיקות (לכל רמה או סוג של בדיקות בשימוש) תהיה שימושית מאוד. קריטריון הכיסוי יכול לשמש כמדד ביצוע מרכזי (KPI – Key Performance Indicator) יעיל שיראה אם בדיקות התוכנה משיגות את מטרותיהן (ראו פרק 1.1.1).

למשל, עבור יישומון לטלפון נייד, בסיס הבדיקות עשוי לכלול רשימה של דרישות ורשימה של מכשירים ניידים נתמכים. כל דרישה הינה פריט בבסיס הבדיקות. כל מכשיר נתמך הינו גם פריט בבסיס הבדיקות. קריטריון הכיסוי עשוי לדרוש לפחות מקרה בדיקה אחד לכל פריט בבסיס הבדיקות. לאחר שבוצעו הבדיקות, תוצאות הבדיקות יאמרו לבעלי העניין האם הדרישות המוגדרות ממומשות והאם התגלו כשלים במכשירים נתמכים.

תקן ISO (ISO/IEC/IEEE 29119-1) מספק מידע נוסף על תהליכי בדיקות.

1.4.2 *פעילויות בדיקה ומטלות בדיקה*

תהליך בדיקה מורכב מקבוצות הפעילויות המרכזיות הבאות:

- תכנון בדיקות
- ניטור ובקרה של בדיקות
- ניתוח בדיקות
- עיצוב בדיקות
- יישום מערך בדיקות
- ביצוע בדיקות
- השלמת הבדיקות

כל קבוצת פעילויות מורכבת ממספר פעילויות שיתוארו בתת הפרקים הבאים. כל פעילות בתוך קבוצת פעילות עשויה להיות מורכבת ממספר מטלות ייחודיות, שעשויות להשתנות מפרויקט אחד למשנהו או מגרסה אחת לאחרת.

יתר על כן, למרות שנראה שרבות מפעילויות אלה מסודרות בסדר הגיוני רציף, הרי שפעמים רבות הן מיושמות בצורה מחזורית. למשל, פיתוח אג'ילי כולל מספר מחזורים של עיצוב תוכנה, בנייה ובדיקות שנעשים בצורה רציפה, כשהתכנון נעשה במקביל וברציפות. כך שפעילויות בדיקה נעשות גם על בסיס מחזורי ומתמשך במסגרת גישת פיתוח זו. גם בפיתוח סדרתי (sequential) פעילויות הבדיקה ההגיוניות והרציפות, עשויות לחפוף זו את זו, להיות משולבות זו בזו, להתבצע בו זמנית, או שניתן להשמיט חלק מהן. בדרך כלל יש להתאים את הפעילויות המרכזיות האלו בהתאם להקשר של המערכת והפרויקט.

[2] התרגום העברי למושג "קריטריון" הוא "אמת מידה". בתוכנית לימודים זו נשתמש במושג הלועזי מאחר והמושג העברי אינו בשימוש יומיומי

תכנון בדיקות (Test planning)

תכנון בדיקות כולל פעילויות שמגדירות את מטרות הבדיקות ואת הגישה להשגת מטרות הבדיקות במסגרת המגבלות הקיימות (לדוגמה, הגדרת טכניקות בדיקה ומטלות בדיקה מתאימות ויצירת לוח זמנים לבדיקות שיעמוד בתאריך היעד). ניתן לשוב ולעדכן את תכניות הבדיקה בהתאם למשוב שיתקבל מפעילויות הניטור והבקרה.

תכנון הבדיקות מוסבר בפירוט בפרק 5.2.

ניטור ובקרה של בדיקות (Test monitoring and control)

ניטור בדיקות כולל השוואה מתמשכת של ההתקדמות בפועל אל מול תוכנית הבדיקות, תוך שימוש בכל מדד ניטור שהוגדר בתוכנית הבדיקות. בקרה של בדיקות כוללת ביצוע הפעולות ההכרחיות להשגת מטרות תוכנית הבדיקות (שעשויה להתעדכן מפעם לפעם). ניטור ובקרת בדיקות תומכות בהערכת קריטריון יציאה, שמוגדר כהגדרה של "נקודת הסיום" ("Done") בחלק ממחזורי החיים (ראו ISTQB-AT Foundation Level Agile Tester Extension Syllabus). לדוגמה, הערכת קריטריון יציאה לביצוע בדיקות ברמת בדיקות נתונה עשוי לכלול:

- בדיקת תוצאות הבדיקה והלוגים אל מול קריטריון הכיסוי
- הערכת רמת האיכות של הרכיב או המערכת בהתבסס על תוצאות הבדיקה והלוגים
- קבלת החלטה על בדיקות נוספות (למשל, אם הבדיקות לא השיגו את רמת הכיסוי המתוכננת של סיכון המוצר, דבר שיצריך כתיבה וביצוע של בדיקות נוספות)

התקדמות הבדיקות אל מול התוכנית מדווחת לבעלי העניין בדוחות התקדמות, כולל דיווח על סטייה מהתוכנית ומידע שיתמוך בקבלת החלטה אם המוצר נבדק מספיק.

ניטור ובקרה של הבדיקות מוספר בפירוט בפרק 5.3.

ניתוח בדיקות (Test analysis)

בשלב ניתוח הבדיקות, נעשה ניתוח לבסיס הבדיקות במטרה לזהות תכונות שניתן לבדוק (testable) ולהגדיר את מצבי הבדיקה המשויכים לתכונות אלה. במלים אחרות, ניתוח הבדיקות מגדיר "מה צריך לבדוק" במושגים של קריטריון כיסוי בר-מדידה.

ניתוח בדיקות כולל את הפעילויות המרכזיות הבאות:

- ניתוח בסיס הבדיקות המתאים לשלב הבדיקות המתוכנן, לדוגמה:
  - מפרטי דרישות (requirement specification) כגון דרישות עסקיות, דרישות פונקציונליות, דרישות מערכת, סיפורי משתמש, אפוסים (epic), מקרי שימוש, או תוצרי עבודה דומים שמגדירים את המאפיינים הפונקציונליים והלא-פונקציונליים הרצויים להתנהגות המערכת או הרכיב
  - עיצוב ויישום של מידע, כגון מסמכי או תרשימי ארכיטקטורה של מערכת או תוכנה, מאפייני עיצוב, תרשימי זרימה, תרשימי מודל (למשל UML או תרשימי קשרים בין ישויות (entity relationship diagrams)), מפרטי ממשקים, או תוצרי עבודה דומים שמגדירים את מבנה המערכת או הרכיב
  - דוחות ניתוח סיכונים, שעשויים לכלול היבטים פונקציונליים, לא-פונקציונליים ומבניים של המערכת או הרכיב
- הערכת בסיס הבדיקות ופריטי הבדיקות (test item) במטרה לזהות פגמים מסוגים שונים, לדוגמה:

- דו-משמעות
- פרטים חסרים
- חוסר עקביות
- אי דיוקים
- סתירות
- משפטים מיותרים

- זיהוי תכונות וקבוצות של תכונות שיש לבדוק
- הגדרה ותעדוף של מצבי בדיקה לכל תכונה בהתבסס על ניתוח בסיס הבדיקות, זיהוי מאפיינים פונקציונליים, לא-פונקציונליים ומבניים, גורמים עסקיים וטכניים אחרים, ורמות סיכון
- יצירת נֶעֱקָבוּת דו-כיוונית בין כל פריט בבסיס הבדיקות ובין מצבי הבדיקה המשויכים לפריט זה (ראו פרקים 1.4.3 ו-1.4.4)

יישום של טכניקות בדיקה מסוג קופסה שחורה, קופסה לבנה וטכניקות מבוססות ניסיון יכול לסייע לתהליך ניתוח הבדיקות (ראו פרק 4) בהפחתת הסבירות שמקרי בדיקה חשובים יושמטו ובהגדרה של מקרי בדיקה מדויקים יותר.

במקרים אחדים, ניתוח בדיקות מייצר מצבי בדיקה שישמשו כמטרות בדיקה באמנות בדיקה (test charters). אמנות בדיקות הן תוצרי עבודה טיפוסיים במספר סוגים של בדיקות מבוססות ניסיון (ראו פרק 4.4.2). כשניתן לעקוב ממטרות בדיקה אלו אל בסיס הבדיקות, ניתן למדוד את הכיסוי המושג בבדיקות מבוססות ניסיון.

תועלת חשובה שניתן להפיק בזמן ניתוח הבדיקות היא זיהוי פגמים, בייחוד אם לא נעשה כל תהליך סקירה אחר ו/או כאשר תהליך הבדיקות מהווה חלק מתהליך הסקירה. פעילויות ניתוח בדיקה לא רק מאמתות שהדרישות עקביות ומבוטאות בצורה נכונה ושלמה, הן גם מוודאות שהדרישות עונות בצורה נכונה על צרכי הלקוח, המשתמש ובעלי עניין אחרים. לדוגמה, טכניקות כמו פיתוח מונחה התנהגות (BDD – Behavior Driven Development) ופיתוח מונחה בדיקות קבלה (ATDD – Acceptance Test Driven Development). טכניקות אלה כוללות יצירת מצבי בדיקה ומקרי בדיקה מסיפורי משתמש וקביעת קריטריון קבלה קודם לכתיבת קוד. כתוצאת אגב מפעילות זו הן גם מאמתות, נותנות תוקף, ומזהות פגמים בסיפורי המשתמש ובקריטריון הקבלה (ראו ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

עיצוב בדיקות (Test design)

בשלב עיצוב הבדיקות, מצבי הבדיקה מורחבים ומפורטים למקרי בדיקה, סדרות של מקרי בדיקה (test sets) ובּוֹדְקָה נוספת. בעוד שניתוח בדיקות עונה על השאלה "מה צריך לבדוק?" הרי שעיצוב הבדיקות מגדיר "איך צריך לבדוק?".

עיצוב הבדיקות כולל את הפעילויות המרכזיות הבאות:

- עיצוב ותעדוף מקרי בדיקה וסדרות של מקרי בדיקה
- זיהוי נתוני בדיקה דרושים לתמיכה במצבי הבדיקה ובמקרי הבדיקה
- עיצוב סביבת הבדיקות וזיהוי כל התשתיות והכלים הנדרשים
- יצירת נֶעֱקָבוּת דו-כיוונית בין בסיס הבדיקות, מצבי הבדיקה, מקרי הבדיקה והליכי הבדיקה (ראו פרק 1.4.4)

הרחבת מצבי הבדיקה למקרי בדיקה וסדרות של מקרי בדיקה כרוכה לעתים קרובות בשימוש בטכניקות בדיקה (ראו פרק 4).

שלב עיצוב הבדיקות עשוי לזהות את אותו סוג של פגמים ששלב ניתוח הבדיקות מגלה בבסיס הבדיקות. וכמו בשלב ניתוח הבדיקות גם בשלב זה, זיהוי פגמים הוא יתרון פוטנציאלי חשוב.

יישום מערך הבדיקות (Test implementation)

בשלב יישום מערך הבדיקות, מייצרים ו/או משלימים את הבּוֹדֶקָה הנדרשת לביצוע הבדיקות, כולל עריכת מקרי הבדיקה בהליכי בדיקה. בעוד שעיצוב הבדיקות עונה על השאלה "איך צריך לבדוק?", שלב יישום מערך הבדיקות עונה על השאלה "האם יש לנו כל מה שנדרש כדי לבצע את הבדיקות?"

יישום מערך הבדיקות כולל את הפעילויות המרכזיות הבאות:

- יצירה ותעדוף של הליכי בדיקה, ויצירת תסריטי בדיקה (test script) אוטומטיים, אם קיימת האפשרות לכך
- יצירת סדרות בדיקה (test suite) מהליכי הבדיקה ותסריטי הבדיקה האוטומטיים (אם ישנם כאלה)
- ארגון תסריטי הבדיקה בלוח זמנים לביצוע הבדיקות בצורה כזו שביצוע הבדיקות יהיה יעיל (ראו פרק 5.2.4)
- בניית סביבת הבדיקות (כולל, אם ניתן, ריתמת בדיקות (test harness), וירטואליזציה של שירותים (service virtualization), סימולטורים ורכיבי תשתיות אחרים) ווידוא שכל הנדרש לצרכי הבדיקות נבנה בצורה נכונה
- הכנת נתוני בדיקה ווידוא שהם נטענים לסביבת הבדיקות
- וידוא ועדכון נֶעֱקָבוּת דו-כיוונית בין בסיס הבדיקות, מצבי הבדיקה, מקרי הבדיקה, הלכי הבדיקה וסדרות הבדיקה (ראו פרק 1.4.4)

שלבי עיצוב הבדיקות ויישום מערך הבדיקות משולבים זה בזה לעיתים קרובות.

בבדיקות חוקרות (exploratory testing) וסוגים אחרים של בדיקות מבוססות ניסיון, עיצוב בדיקות ויישום מערך הבדיקות נעשים, ומתועדים אם צריך, כחלק מביצוע הבדיקות. בדיקות חוקרות עשויות להתבסס על אמנות בדיקה (test charters) (שנכתבות בשלב ניתוח הבדיקות), ומבוצעות תוך כדי עיצוב הבדיקות ויישום הבדיקות (ראו פרק 4.4.2).

ביצוע בדיקות (Test execution)

בשלב ביצוע הבדיקות מריצים את סדרות הבדיקות בהתאם ללוח הזמנים לביצוע הבדיקות.

ביצוע הבדיקות כולל את הפעילויות המרכזיות הבאות:

- זיהוי ותיעוד הגרסאות של פריטי הבדיקות או מושא הבדיקות, כלי הבדיקות ומכלול מרכיבי הבדיקות (testware)
- הרצת בדיקות בצורה ידנית או באמצעות כלי הרצת בדיקות
- השוואה בין תוצאות הבדיקה שהתקבלו לתוצאות המצופות
- ניתוח חריגות למציאת הגורמים להן (לדוגמה, כשלים עשויים לקרות בשל פגמים בקוד אבל יתכן וקיבלנו תוצאה חיובית כוזבת [ראו פרק 1.2.3])
- דיווח על פגמים בהתבסס על הכשלים שהתגלו (ראו פרק 5.6)

- תיעוד תוצאות ביצוע הבדיקות (לדוגמה, בדיקה עברה, בדיקה נכשלה, בדיקה נחסמה)
- חזרה על פעילויות בדיקה בין אם כתוצאה של פעולה שנעשתה לתיקון חריגה, או כחלק מבדיקות מתוכננות (לדוגמה, ביצוע בדיקה שתוקנה, בדיקות אימות (confirmation testing), ו/או בדיקות נסיגה (regression testing))
- וידוא ועדכון נֶעֱקְבוּת דו-כיוונית בין בסיס הבדיקות, מצבי הבדיקה, מקרי הבדיקה, הליכי הבדיקה ותוצאות הבדיקה

השלמת הבדיקות (Test completion)

פעילויות השלמת הבדיקות כוללות איסוף נתונים מפעילויות הבדיקה שהושלמו, לתעד את הניסיון, את מכלול מרכיבי הבדיקות וכל מידע רלוונטי אחר. פעילויות השלמת בדיקות נעשות כשהפרויקט מגיע לאבן דרך, לדוגמה, כאשר מערכת תוכנה משוחררת, פרויקט בדיקות הסתיים (או בוטל), מחזור של פרויקט אג'ילי הגיע לסיום (לדוגמה, במסגרת פגישה לצורך הפקת לקחים), השלמת שלב בדיקות, או כאשר גרסת תחזוקה הושלמה.

השלמת הבדיקות כוללת את הפעילויות המרכזיות הבאות:

- בדיקה שכל דוחות הפגמים נסגרו, או לחילופין הוגדרו בקשות לשינוי או הוכנסו פריטים לעתודת המוצר (product backlog) עבור כל פגם שלא תוקן בסיום ביצוע הבדיקות
- כתיבת דוח השלמת הבדיקות שיימסר לבעלי העניין
- סיום וארכון סביבת הבדיקות, נתוני הבדיקה, תשתית הבדיקה ומכלול מערך הבדיקות לשימוש בשלב מאוחר יותר
- העברת מכלול מערך הבדיקות לידי צוותי התחזוקה, צוותי פרויקט אחרים ו/או בעלי עניין אחרים שעשויים להפיק תועלת מכך
- ניתוח לקחים (lessons learned) מפעילויות הבדיקה שהסתיימו במטרה להחליט על ביצוע שינויים למחזורים, גרסאות ופרויקטים הבאים
- שימוש במידע שנאסף לשיפור בגרות תהליך הבדיקות

1.4.3 *תוצרי עבודת הבדיקות*

תוצרי עבודת הבדיקות נוצרים כחלק מתהליך הבדיקות. כשם שיש הבדלים משמעותיים בדרך בה ארגונים מיישמים את תהליך הבדיקות, ישנם גם הבדלים משמעותיים בסוגי תוצרי הבדיקות שנוצרים במהלך התהליך, בדרך בה התוצרים מאורגנים ומנוהלים, ובשמות שניתנים לתוצרים אלה. תוכנית הלימודים הזו הולכת בעקבות תהליך הבדיקות שתואר למעלה ותוצרי עבודת הבדיקות שמתוארים כאן ובמילון המונחים של ISTQB. תקן ISO (ISO/IEC/IEEE 29119-3) עשוי גם הוא לשמש כקו מנחה עבור תוצרי עבודת הבדיקות.

רבים מתוצרי עבודת הבדיקות המתוארים בפרק זה ניתן ליצור ולנהל בעזרת כלי ניהול בדיקות וכלי ניהול פגמים (ראו פרק 6).

תוצרי עבודת תכנון בדיקות

תוצרי עבודת תכנון הבדיקות כוללים בדרך כלל תוכנית בדיקות אחת או יותר. תוכנית הבדיקות כוללת מידע על בסיס הבדיקות, אליו יתייחסו יתר תוצרי עבודת הבדיקות באמצעות נֶעֱקְבוּת (ראו בהמשך ובחלק 1.4.4), כמו גם על קריטריון היציאה (או הגדרה של "הושלם") בו יעשה שימוש בשלב ניטור ובקרת הבדיקות. תכניות בדיקה מתוארות בפרק 5.2.

תוצרי עבודת ניטור ובקרת הבדיקות

תוצרי עבודה טיפוסיים לשלב ניטור ובקרת הבדיקות כוללים סוגים שונים של דוחות בדיקות, כולל דוחות התקדמות הבדיקות (שמיוצר על בסיס מתמשך ו/או סדיר) ודוחות סיכום בדיקות (שמיוצרים בסיום אבני דרך שונים). על כל דוחות הבדיקות לכלול מידע על תהליך הבדיקות נכון ליום הדו"ח, כולל סיכום תוצאות הרצת הבדיקות כשאלה זמינים. המידע צריך להתאים לקהל היעד.

תוצרי עבודת ניטור ובקרת הבדיקות צריכים לענות על הנושאים החשובים להנהלת הפרויקט, כגון השלמת משימות, הקצאת משאבים, ניצול משאבים ומאמץ.

ניטור ובקרת הבדיקות, כמו גם תוצרי העבודה שנוצרים במהלך הפעילויות האלה, מוסברים בהרחבה בפרק 5.3 של תוכנית לימודים זו.

תוצרי עבודת ניתוח בדיקות

תוצרי עבודה טיפוסיים לשלב ניתוח הבדיקות כוללים הגדרה ותעדוף של מצבי הבדיקה, שעקרונית כל אחד מהם מתייחס לפריט (או פריטים) מסוים בבסיס הבדיקות באמצעות נֶעֱקָבוּת דו-כיוונית. ניתוח בדיקות עבור בדיקות חוקרות יכלול יצירה של אמנות בדיקה. ניתוח בדיקות עשוי לגלות פגמים בבסיס הבדיקות, עליהם יש לדווח.

תוצרי עבודת עיצוב בדיקות

תוצרי העבודה בשלב עיצוב בדיקות כוללים מקרי בדיקה וסדרות של מקרי בדיקה שמיישמים את מצבי הבדיקה שהוגדרו בשלב ניתוח הבדיקות. בשלב זה נכון יהיה ליצור שלד של מקרי בדיקה, ללא ערכי קלט ממשיים וללא תוצאות צפויות. ניתן להשתמש במקרי בדיקה אלה בסבבי בדיקות רבים עם ערכי קלט שונים, תוך תיעוד מדוקדק של מקרה הבדיקה. עקרונית, כל מקרה בדיקה יתייחס למצב בדיקה (אחד או יותר) בעזרת נֶעֱקָבוּת דו-כיוונית.

עיצוב הבדיקות כולל עיצוב ו/או זיהוי של נתוני הבדיקה הדרושים, עיצוב סביבת הבדיקות, וזיהוי התשתית והכלים, למרות שהפירוט בה כל אלה מתועדים יהיה שונה בצורה משמעותית.

בשלב עיצוב הבדיקות יתכן ויחודדו מצבי הבדיקה שהוגדרו בשלב ניתוח הבדיקות.

תוצרי עבודת יישום מערך הבדיקות

תוצרי העבודה בשלב יישום מערך הבדיקות כוללים:

- הליכי בדיקה וסדר הרצת הליכי הבדיקה הללו
- סדרות בדיקה (test suites)
- לוח זמנים לביצוע הבדיקות

עקרונית, לאחר סיום יישום מערך הבדיקות, ניתן להראות את רמת הכיסוי שהושגה בהתאם לקריטריון שהוגדר בתוכנית הבדיקות, בעזרה נֶעֱקָבוּת דו-כיוונית בין הליכי הבדיקה ופריטים מסוימים בבסיס הבדיקות, דרך מקרי הבדיקה ומצבי הבדיקה.

ישנם מקרים בהם יישום מערך הבדיקות כולל יצירת תוצרי עבודה תוך שימוש, או תוצרים שישתמשו, בכלים, כגון וירטואליזציה של שירותים ותסריטי בדיקה אוטומטיים.

יישום מערך הבדיקות כולל גם יצירה ווידוא של נתוני בדיקה וסביבת הבדיקות. תכולת אימות הנתונים ו/או הסביבה עשויים להיות מתועדים בצורות שונות משמעותית.

נתוני הבדיקה משמשים להשמת ערכי קלט ממשיים ותוצאות צפויות תואמות במקרי הבדיקה. ערכים ממשיים אלה, בשילוב הוראות מפורשות על השימוש בערכים אלה, הופכים את השלד של מקרי הבדיקה

למקרי בדיקה שניתן להריץ. מקרי בדיקה עשויים להשתמש בנתוני בדיקה שונים, כאשר יבוצעו על גרסאות שונות של מושא הבדיקות. בעזרת שימוש באורקל בדיקות, מזהים את תוצאות הבדיקה הצפויות שתואמות לנתוני הבדיקה הממשיים.

בבדיקות חוקרות, חלק מתוצרי עיצוב הבדיקות ויישום מערך הבדיקות, נוצרים בשלב ביצוע הבדיקות, למרות שההיקף בהם מתועדות הבדיקות החוקרות (ואיתם הנֶעֱקבוּת לפריטים ספציפיים בבסיס הבדיקות) עשוי להשתנות בצורה משמעותית.

בשלב יישום מערך הבדיקות יתכן ויחודדו מצבי הבדיקה שהוגדרו בשלב ניתוח הבדיקות.

תוצרי עבודת ביצוע הבדיקות

תוצרי העבודה בשלב ביצוע הבדיקות כוללים:

- תיעוד סטטוס מקרי בדיקה או הליכי בדיקה יחידים (לדוגמה, מוכן לביצוע, עבר, נכשל, נחסם, נפסח בכוונה, וכו')
- דוחות פגמים (ראו פרק 5.6)
- תיעוד פריט(י) הבדיקות, מושא(י) הבדיקות, כלי הבדיקות ומכלול מרכיבי הבדיקות שנכללו בבדיקות

עקרונית, לאחר סיום ביצוע הבדיקות, ניתן לקבוע ולדווח על מצבו של כל אחד מהפריטים בבסיס הבדיקות באמצעות נֶעֱקבוּת דו-כיוונית עם הליכי הבדיקות המתאימים. לדוגמה, נוכל להגיד אילו דרישות עברו את כל הבדיקות המתוכננות, לאילו דרישות יש בדיקות שנכשלו ו/או יש פגמים שקשורים בהן, ולאילו דרישות יש בדיקות מתוכננות שמחכות לביצוע. כך ניתן לאמת את השגת קריטריון הכיסוי, ולדווח את תוצאות הבדיקה במושגים שיהיה מובנים לבעלי העניין.

תוצרי עבודת השלמת הבדיקות

תוצרי העבודה בשלב השלמת הבדיקות כוללים דוחות סיכום הבדיקות, משימות לשיפור בפרויקטים או מחזורים הבאים (למשל, כמו שנעשה במפגשי הפקת לקחים במתודולוגית אג'ייל), בקשות לשינוי או הכנסת פריטים לעתודת המוצר, וסגירת מכלול מרכיבי הבדיקות.

1.4.4 *נֶעֱקבוּת בין בסיס הבדיקות ותוצרי עבודת הבדיקות*

כמו שמוזכר בפרק 1.4.3, תוצרי עבודה הבדיקות והשמות של תוצרים אלה עשויים להיות שונים בצורה משמעותית. אך ללא קשר להבדלים אלה, בכדי ליישם תהליך ניטור ובקרת בדיקות יעילים, חשוב להגדיר ולתחזק נֶעֱקבוּת בין כל פריט בבסיס הבדיקות ותוצרי הבדיקות הקשורים לפריט זה, לכל אורך תהליך הבדיקות, כמו שמתואר למעלה. בנוסף להערכת כיסוי הבדיקות, נֶעֱקבוּת טובה תומכת ב:

- ניתוח השפעת השינויים
- הכנת הבדיקות לביקורת
- עמידה בכללים פנימיים של הארגון (IT governance)
- שיפור הבנת דוחות התקדמות הבדיקות ודוחות סיכום הבדיקות, שיכללו את הסטטוס של פריטים בבסיס הבדיקות (לדוגמה, דרישות שהבדיקות שלהן עברו, דרישות שהבדיקות שלהן נכשלו, ודרישות שהבדיקות שלהן ממתינות לביצוע)
- הסברת ההיבטים הטכניים של הבדיקות לבעלי העניין במושגים שהם יוכלו להבין
- אספקת מידע לשם הערכת איכות המוצר, יכולת התהליך, והתקדמות הפרויקט אל מול המטרות העסקיות

חלק מכלי ניהול הבדיקות מספקים מודלים לתוצרי תוצאות הבדיקות שמתאימים לחלק או לכל התוצרים שמוגדרים בפרק זה. ישנם ארגונים שבונים בעצמם את מערכות הניהול לארגון תוצרי הבדיקות ואלה מספקים את מידע הנֶעֱקָבוּת הנדרש.

1.5 הפסיכולוגיה של הבדיקות

פיתוח תוכנה, כולל בדיקות תוכנה, נעשה על ידי בני אדם. לכן, לפסיכולוגיה של אנשים ישנה השפעה חשובה על בדיקות תוכנה.

1.5.1 *פסיכולוגיה של אנשים ובדיקות*

זיהוי פגמים באמצעות בדיקה סטטית כמו סקירה (review) של דרישות או סדנת חידוד לסיפורי משתמש, או זיהוי כשלים במהלך ביצוע בדיקות דינמיות, עשוי להיתפס כביקורת על המוצר ועל הכותב. גורם בפסיכולוגיה של אנשים שנקרא הטיית אישור (confirmation bias) עשוי להקשות על קבלת מידע שלא מסכים עם הדעות שמחזיק האדם. לדוגמה, מאחר ומפתחים מצפים שהקוד אותו כתבו יהיה תקין, יש להם הטיית אישור שמקשה עליהם לקבל שהקוד לא תקין. בנוסף להטיית אישור, הטיות קוגניטיביות אחרות יקשו על אנשים להבין או לקבל מידע שמתקבל על ידי בדיקות. יותר מזה, קיימת תכונה אנושית שמאשימה את מביא החדשות הרעות, ומידע שמתקבל על ידי בדיקות נחשב חדשות רעות.

כתוצאה מגורמים פסיכולוגיים אלה, ישנם אנשים שרואים בבדיקות גורם הרסני, למרות תרומתן הגדולה להתקדמות הפרויקט ולאיכות המוצר (ראה פרקים 1.1 ו-1.2). כדי להחליש תפיסה זו, יש להעביר מידע על פגמים וכשלים בדרך בונה. כך, יופחת המתח בין בודקים ומנתחי מערכת, מנהל המוצר, מעצבים ומפתחים. גישה זו נכונה גם בבדיקות סטטיות וגם בבדיקות דינמיות.

כישורים בינאישיים טובים הם תכונה חשובה לבודקים ומנהלי בדיקות והיא תאפשר להם לתקשר ביעילות מידע על פגמים, כשלים, תוצאות בדיקה, התקדמות הבדיקות וסיכונים, ולבנות יחסים חיוביים עם עמיתים. דוגמאות לדרכי תקשורת יעילות:

- התחילו עם שיתוף פעולה במקום קרבות. הזכירו לכולם שהמטרה המשותפת היא מערכות עם איכות טובה יותר.
- הדגישו את היתרונות של הבדיקות. לדוגמה, מידע על פגמים יכול לעזור לכותבים לשפר את תוצרי העבודה שלהם ואת היכולות שלהם. עבור הארגון, פגמים שהתגלו ותוקנו בזמן הבדיקות יחסכו זמן וכסף ויפחיתו את הסיכון הכולל לאיכות המוצר.
- דווחו על תוצאות הבדיקה וממצאים אחרים בדרך ניטרלית, מבוססת עובדות, ללא ביקורות על האדם שיצר את הפריט הפגום. כתבו דוחות פגמים וממצאי סקירות בצורה אובייקטיבית ומבוססת עובדות.
- נסו להבין איך מרגיש האדם האחר והסיבות שיגרמו לו להגיב בצורה שלילית על המידע.
- וודאו שהאדם השני הבין מה שנאמר ולהיפך.

מטרות בדיקה טיפוסיות הוגדרו מוקדם יותר (ראו פרק 1.1). להגדרה ברורה של מטרות הבדיקה יש השלכות פסיכולוגיות חשובות. מרבית האנשים נוטים להתאים את התוכניות וההתנהגות שלהם עם המטרות שנקבעו על ידי הקבוצה, ההנהלה, ובעלי עניין אחרים. חשוב גם שבודקים יתאימו עצמם למטרות אלה עם הטיה אישית מינימלית.

1.5.2 *דרך חשיבה של בודקים ומפתחים*

במקרים רבים בודקים ומפתחים חושבים בצורה שונה. המטרה העיקרית של פיתוח היא לעצב ולבנות את המוצר. כמו שנידון קודם לכן, מטרות הבדיקות כוללות אימות ותיקוף של המוצר, מציאת פגמים קודם לשחרור המוצר, וכדומה. אלו מטרות שונות שדורשות דרך חשיבה שונה. קירוב דרכי החשיבה זו לזו יעזור להשיג רמה גבוהה של איכות המוצר.

דרך חשיבה משליכה על ההנחות שאדם עושה ועל השיטות המועדפות עליו בעת קבלת החלטות ופתרון בעיות. דרך החשיבה של בודק צריכה לכלול סקרנות, פסימיזם מקצועי, מבט ביקורתי, שימת לב לפרטים ומוטיבציה לתקשורת ויחסים טובים וחיוביים. דרך החשיבה של בודק נוטה להתפתח ולהתבגר ככל שהבודק צובר ניסיון.

דרך החשיבה של מפתח כוללת כמה גורמים של דרך החשיבה של בודק, אבל פעמים רבות מפתחים מצליחים מעוניינים יותר לעצב ולבנות פתרונות, מאשר לחשוב על מה עלול להשתבש עם הפתרונות הללו. בנוסף, הטיית אישור מקשה על מציאת שגיאות בעבודה שלהם עצמם.

עם דרך החשיבה הנכונה, מפתחים יכולים לבדוק את הקוד של עצמם. לעיתים יש למודלים שונים של מחזור חיי תוכנה דרכים שונות לארגן את הבודקים ואת פעילויות הבדיקה. כשחלק מפעילויות הבדיקה נעשות בידי בודקים עצמאיים, הדבר מגדיל את יעילות מציאת הפגמים, דבר חשוב במיוחד למערכת גדולות, מסובכות וחיוניות. בודקים עצמאיים מביאים נקודת מבט שונה מהכותבים השונים של תוצרי העבודה (מנתחים עסקיים, מנהלי המוצר, מעצבים ומפתחים), מאחר ויש להם הטיות קוגניטיביות שונות מכותבים אלה.

2 בדיקות לאורך מחזור חיי תוכנה — 100 דקות

מושגים

בדיקות קבלה (acceptance testing), בדיקות אלפא (alpha testing) בדיקות ביתא (beta testing), מוצר מדף ((commercial off-the-shelf (COTS), בדיקות אינטגרציית רכיבים (component integration testing), בדיקות רכיבים (component testing), בדיקות אימות (confirmation testing), בדיקות קבלה חוזיות (contractual acceptance testing), בדיקות פונקציונליות[3] (functional testing), ניתוח השפעה (impact analysis), בדיקות אינטגרציה (integration testing), בדיקות תחזוקה (maintenance testing), בדיקות לא פונקציונליות (non-functional testing), בדיקות קבלה תפעוליות (operational acceptance testing), בדיקות נסיגה (regression testing), בדיקות קבלה רגולטוריות (regulatory acceptance testing), מודל פיתוח סדרתי (sequential development model), בדיקות אינטגרציית מערכת (system integration testing), בדיקות מערכת (system testing), בסיס בדיקות (test basis), מקרה בדיקה (test case), סביבת בדיקות (test environment), רמת בדיקה (test level), מושא הבדיקות (test object), מטרת הבדיקות (test objective), סוג הבדיקה (test type), בדיקות קבלה ע"י משתמש (user acceptance testing), בדיקות קופסה לבנה (white-box testing)

יעדי הלימוד של פרק בדיקות לאורך מחזור חיי תוכנה

2.1 מודלים של מחזור חיי תוכנה

FL-2.1.1 (K2) הסבר את הקשרים בין פעילויות פיתוח תוכנה ופעילויות בדיקות במחזור חיי התוכנה

FL-2.1.2 (K1) זהה את הסיבות לכך שיש להתאים מודלים של מחזור חיי תוכנה להקשר ולמאפייני הפרויקט והמוצר

2.2 רמות בדיקה

FL-2.2.1 (K2) השווה בין רמות הבדיקה השונות מנקודת המבט של מטרות הבדיקה, בסיס הבדיקה, מושאי הבדיקה, פגמים וכשלים אופייניים, גישות ותחומי אחריות

2.3 סוגי בדיקות

FL-2.3.1 (K2) השווה בין בדיקות פונקציונליות, בדיקות לא פונקציונליות ובדיקות קופסה לבנה

FL-2.3.2 (K1) דע שניתן לבצע בדיקות פונקציונליות, בדיקות לא פונקציונליות ובדיקות קופסה לבנה בכל רמת בדיקה

FL-2.3.3 (K2) השווה בין מטרות בדיקות אימות ומטרות בדיקות נסיגה

2.4 בדיקות תחזוקה

FL-2.4.1 (K2) סכם את הגורמים לבדיקות תחזוקה

FL-2.4.2 (K2) תאר את תפקיד ניתוח ההשפעה בבדיקות תחזוקה

[3] התרגום העברי למושג "פונקציונליות" הוא "תפקודיות". בתוכנית לימודים זו נשתמש במושג הלועזי מאחר והמושג העברי אינו בשימוש יומיומי

2.1 מודלים למחזור חיי תוכנה

מודל מחזור חיי תוכנה מתאר את סוגי הפעילויות המבוצעות בכל שלב בפרויקט פיתוח תוכנה, ואיך פעילויות אלה מתייחסות אחת לשנייה בצורה לוגית וכרונולוגית. ישנם מספר מודלים שונים של מחזור חיי תוכנה, וכל אחד מהם דורש גישה שונה לבדיקות.

2.1.1 *פיתוח תוכנה ובדיקות תוכנה*

חלק חשוב מתפקידו של בודק הוא להכיר מודלים אופייניים למחזור חיי תוכנה בכדי שיוכל לבצע פעילויות בדיקה מתאימות.

בכל מודל למחזור חיי תוכנה, יש מספר מאפיינים של תהליך בדיקה טוב:

- לכל פעילות פיתוח ישנה פעילות בדיקה תואמת
- לכל רמת בדיקה ישנן מטרות בדיקה המיוחדות לאותה רמה
- פעילויות ניתוח בדיקות ועיצוב בדיקות לרמת בדיקה נתונה מתחילות במקביל לפעילות הפיתוח התואמת
- בודקים משתתפים בדיונים להגדרה ושיפור הדרישות והעיצוב, ומעורבים בסקירה של תוצרי העבודה (לדוגמה, דרישות, עיצוב, סיפורי משתמש וכו') כבר מהגרסאות הראשונות.

לא משנה איזה מודל למחזור חיי תוכנה נבחר, על פעילויות הבדיקות להתחיל בשלבים הראשונים של מחזור החיים, בהתאם לעקרון הבדיקה של בדיקות מוקדמות.

תוכנית לימודים זו מבחינה בשתי קטגוריות כלליות למחזור חיי תוכנה :

- מודלים סדרתיים (sequential)
- מודלים של פיתוח בסבבים (iterative) ומודלים מצטברים (incremental)

מודלים סדרתיים לפיתוח מתארים את תהליך פיתוח התוכנה כרצף פעילויות המשכי וליניארי. הכוונה היא שכל שלב בתהליך הפיתוח אמור להתחיל כשהשלב הקודם מסתיים. בתאוריה, אין חפיפה של שלבים, אך למעשה, מועיל מאוד לקבל משוב מוקדם מהשלב הקודם.

במודל מפל המים (Waterfall model), פעילויות הפיתוח (לדוגמה, ניתוח דרישות, עיצוב, כתיבת קוד, בדיקות) מסתיימות אחת אחרי השנייה. במודל זה, פעילויות הבדיקה מתבצעות רק אחרי שכל פעילויות הפיתוח מסתיימות.

שלא כמודל מפל המים, מודל ה-V משלב את פעילויות הבדיקה במהלך תהליך הפיתוח, תוך יישום עקרון הבדיקות המוקדמות. יותר מכך, מודל ה-V כולל רמות בדיקות שקשורות לכל שלב פיתוח תואם, כך שעקרון הבדיקות המוקדמות נתמך עוד יותר (ראו פרק 2.2 על רמות בדיקה). במודל זה, ביצוע בדיקות הקשורות לכל שלב בדיקות מתבצע בצורה המשכית, אך ישנם מקרים בהם קיימת חפיפה.

במודלים סדרתיים לפיתוח התוכנה הנמסרת מכילה את הסט המלא של המאפיינים והדרישות שאופיינו לפרויקט, אך בדרך כלל נדרשים חודשים או שנים למסירה לבעלי העניין והמשתמשים.

בפיתוח בשלבים מצטברים (incremental development), קביעת הדרישות, עיצוב, בניית המערכת ובדיקת המערכת נעשית בשלבים, כלומר יכולות המערכת גדלות בצורה מצטברת. גודל השינוי אינו קבוע, כאשר בשיטות אחדות כל שלב מאגד בתוכו שינויים רבים; בשיטות אחרות השינויים קטנים יותר. ההבדל יכול להיות קטן ממש: שינוי קטן בממשק המשתמש או אפשרות חיפוש חדשה.

פיתוח בסבבים נעשה כשקבוצות של מאפיינים מוגדרים, מעוצבים, נבנים ונבדקים ביחד, בסדרה של סבבים, בדרך כלל בפרקי זמן קבועים. סבבים עשויים לכלול שינויים למאפיינים שפותחו בסבבים

מוקדמים, וכן שינויים בהיקף הפרויקט. בכל סבב נמסרת תוכנה עובדת שמכילה יותר ויותר מסך המאפיינים שהוגדרו למערכת, עד שנמסרת הגרסה הסופית של התוכנה או שהפיתוח נפסק.

דוגמאות למודלי פיתוח מצטברים ופיתוח בסבבים:

- Rational Unified Process: סבבים ארוכים יחסית (לדוגמה, בין חודשיים לשלושה) כאשר רמת השינויים המצטברים בכל מחזור גדולה בהתאמה, למשל שתיים או שלוש קבוצות של מאפיינים
- Scrum: כל מחזור הוא יחסית קצר (לדוגמה, שעות, ימים או מספר שבועות) ורמת השינויים המצטברת קטנה בהתאמה, למשל הרחבות קטנות ביכולות של מאפיין, או שניים-שלושה מאפיינים חדשים
- Kanban: מיושם עם או בלי פרקי זמן קבועים לכל מחזור, כאשר כל מחזור מכיל הרחבה בודדת או מאפיין שהושלמו, או מקבץ מספר מאפיינים ביחד
- Spiral (או בניית אב-טיפוס, prototyping): משלב יצירת קבוצות מאפיינים ניסיוניות, שחלקן יעברו שינויים ושיפורים או אפילו ייזנחו בהמשך עבודת הפיתוח

בפיתוח של רכיבים או מערכות באמצעות שיטות אלו, ישנה לעיתים קרובות חפיפה ושילוב של רמות בדיקה לאורך תהליך הפיתוח. עקרונית, כל מאפיין נבדק במספר רמות בדיקה כשהוא מתקדם לקראת מסירה. בחלק מהמקרים, הצוותים משתמשים במסירה מתמשכת (continuous delivery) או בפריסה מתמשכת (continuous deployment), שתי שיטות שכוללות שימוש משמעותי באוטומציה[4] של מספר רמות בדיקה כחלק מתהליך המסירה. כאשר משתמשים בשיטות אלה, מאמצי פיתוח רבים נעשים במסגרת של צוותים עצמאיים, דבר שיכול לשנות את הדרך בה מאורגנת עבודת הבדיקות כמו גם את היחסים בין הבודקים והמפתחים.

שיטות אלה יוצרות מערכת גדלה, שניתן למסור למשתמשי קצה עם סיום הפיתוח של כל מאפיין, בסיום כל מחזור פיתוח, או בצורה מסורתית יותר של מסירת גרסה גדולה. בין אם הגרסה נמסרת למשתמשי הקצה או לא, חשיבותן של בדיקות הנסיגה גדלה עם הגידול במערכת.

בניגוד למודלים סדרתיים, מודלים של פיתוח בסבבים ומודלים מצטברים עשויים למסור תוכנה עובדת תוך מספר שבועות או אפילו ימים, ואילו מסירה של המוצר הסופי והמלא תקרה רק לאחר תקופה של חודשים או אפילו שנים.

למידע נוסף על בדיקות תוכנה בהקשר של פיתוח אג'ילי ראו ISTQB-AT Foundation Level Agile Tester Extension, Black 2017, Crispin 2008 ו-Gregory 2015.

2.1.2 *התאמת מודל מחזור חיי תוכנה לפרויקט ולמוצר*

כשבוחרים מודלים למחזור חיי תוכנה יש להתאימם להקשר של מאפייני הפרויקט והמוצר. בחירת מודל מתאים למחזור חיי תוכנה והתאמת המודל תתבסס על מטרת הפרויקט, סוג המוצר המפותח, עדיפות עסקית (לדוגמה, הצורך להגיע ראשונים לשוק), וסיכוני המוצר והפרויקט שזוהו. לדוגמה, פיתוח ובדיקות של תוכנה לא קריטית לשימוש פנימי בארגון תהיה שונה מפיתוח ובדיקות של מערכת בטיחות חיונית כמו מערכת שליטה לבלמים של מכונית. דוגמה נוספת, ישנם מקרים בהם נורמות ארגוניות או תרבותיות מקשות על התקשורת בין חברי צוות, דבר שיקשה על שימוש במודל של פיתוח בסבבים.

בהתאם להקשר של הפרויקט, יתכן ויהיה צורך לשלב או לארגן מחדש את רמות הבדיקה ו/או את פעילויות הבדיקה. לדוגמה, בשילוב של תוכנת מוצר מדף לתוך מערכת גדולה יותר, הרוכש עשוי לבצע בדיקות יכולת פעולה-הדדית (interoperability testing) ברמת בדיקות אינטגרציית המערכת (לדוגמה,

[4] התרגום העברי למושג "אוטומציה" הוא "מיכון". בתוכנית לימודים זו נשתמש במושג הלועזי מאחר והמושג העברי אינו בשימוש יומיומי

אינטגרציה לתשתית ולמערכות אחרות) וברמת בדיקות הקבלה (בדיקות פונקציונליות ולא פונקציונליות, כמו גם בדיקות קבלת משתמש ובדיקות קבלה תפעוליות). ראו פרק 2.2 על רמות בדיקה ופרק 2.3 על סוגי בדיקה.

בנוסף, ניתן לשלב בין מודלים למחזור חיי תוכנה. לדוגמה, ניתן להשתמש במודל V לפיתוח, בדיקות ואינטגרציה של מערכות backend, ובמודל פיתוח אג'ילי לפיתוח ובדיקה של ממשק משתמש הקצה (UI – User Interface) ושל הפונקציונליות של המערכת. ניתן להשתמש באב טיפוס בשלב מוקדם של הפרויקט, ולעבור למודל פיתוח מצטבר כשמסתיים השלב הניסיוני.

בפיתוח מערכות אינטרנט של דברים (IoT, Internet of Things), שמורכבות ממספר רב של רכיבים שונים, כמו התקנים, מוצרים ושירותים, מקובל ליישם מודל מחזור חיי תוכנה נפרד לכל רכיב. דבר זה מהווה אתגר לפיתוח גרסאות של מערכות אינטרנט של דברים. בנוסף, מחזור חיי תוכנה של רכיבים כאלה שם דגש רב יותר על שלבים מאוחרים של מחזור חיי הפיתוח, לאחר שהרכיבים נכנסו לשימוש תפעולי (לדוגמה, שלבי תפעול, עדכון, והוצאה משימוש).

2.2 רמות בדיקה

רמות בדיקה הן קבוצות של פעילויות בדיקה שמאורגנות ומנוהלות ביחד. כל רמת בדיקה היא מופע של תהליך הבדיקות, ומכילה את הפעילויות שמתוארות בפרק 1.4. פעולות אלה מבוצעות בכל רמת נתונה של הפיתוח, מיחידות או רכיבים בודדים ועד למערכות שלמות, או מערכות המורכבות ממספר מערכות (systems of systems), במקרים בהם מתקיימת מערכת כזו. רמות בדיקה מתייחסות גם לפעילויות אחרות במסגרת מחזור חיי התוכנה. רמות הבדיקה המופיעות בתוכנית לימודים זו הן:

- בדיקות רכיבים (component testing)
- בדיקות אינטגרציה (integration testing)
- בדיקות מערכת (system testing)
- בדיקות קבלה (acceptance testing)

לרמות בדיקה יהיו המאפיינים הבאים:

- מטרות מוגדרות וייחודיות
- בסיס בדיקות שביחס אליו מגדירים מקרי בדיקה
- מושא בדיקות (כלומר, מה נבדק)
- פגמים וכשלים אופייניים
- גישות ספציפיות ותחומי אחריות

לכל רמת בדיקה נדרשת סביבת בדיקות מתאימה. בבדיקות קבלה, לדוגמה, רצוי להשתמש בסביבת בדיקות שמדמה את הסביבה התפעולית, בעוד שבבדיקות רכיבים המפתחים בדרך כלל משתמשים בסביבת הפיתוח שלהם.

2.2.1 *בדיקות רכיבים (component testing)*

המטרות של בדיקות רכיבים

בדיקות רכיבים (המוכרות גם בשם בדיקות יחידה (unit testing) ובדיקות מודולים (module testing)) מתמקדות ברכיבים שניתן לבדוק בנפרד. המטרות של בדיקות רכיבים כוללות:

- הפחתת סיכונים
- וידוא שההתנהגות הפונקציונלית והלא פונקציונלית של הרכיב הן בהתאם לעיצוב ולאפיון

- בניית אמון באיכות הרכיב
- מציאת פגמים ברכיב
- מניעת זליגה של פגמים לרמות בדיקה גבוהות יותר

במקרים מסוימים, בייחוד במודלי פיתוח מצטברים ופיתוח בסבבים (לדוגמה, אג'ייל), כשהקוד עובר שינויים תדירים, יש לבדיקות נסיגה אוטומטיות תפקיד מפתח בבניית אמון בכך שהשינויים שנעשו לא שברו רכיבים קיימים.

התוכן והשימוש של בדיקות רכיבים תלוי במודל מחזור חיי התוכנה ובמערכת. בדיקות רכיבים נעשות לעיתים קרובות בניתוק משאר המערכת, תוך שימוש באובייקטים מדומים (mock objects), וירטואליזציה של שירותים (services), רתמות (harness), תותבים (stubs), ומנהלי התקנים (drivers). בדיקות רכיבים עשויות לכסות מאפיינים פונקציונליים (לדוגמה, נכונות של חישובים), מאפיינים לא פונקציונליים (לדוגמה, חיפוש אחרי זליגת זיכרון), ותכונות מבניות (לדוגמה, בדיקות החלטות (decision testing)).

בסיס הבדיקות (test basis)

דוגמאות לתוצרים שיכולים לשמש כבסיס בדיקות לבדיקות רכיבים כוללות:

- עיצוב מפורט
- קוד
- מודל נתונים
- מפרט הרכיב

מושא הבדיקות (test objects)

מושאי בדיקות אופייניים לבדיקות רכיבים כוללים:

- רכיבים, יחידות או מודולים
- קוד ומבני נתונים
- מחלקות (classes)
- מודולים של בסיסי נתונים

פגמים וכשלים אופייניים

דוגמאות לפגמים וכשלים שבדיקות רכיבים עוזרות לגלות, כוללות:

- פונקציונליות שגויה (לדוגמה, לא כפי שמתואר במפרטי העיצוב)
- בעיות של זרימת נתונים
- קוד ולוגיקה שגויים

הפגמים מתוקנים בדרך כלל ברגע שהם מתגלים, לרוב ללא ניהול פגמים פורמלי. אולם, כאשר המפתחים כן מדווחים על הפגמים, הדבר מספק מידע חשוב עבור ניתוח שורש הבעיות ושיפור תהליכים.

גישות ספציפיות ותחומי אחריות

בדיקות רכיבים מבוצעות בדרך כלל על ידי המפתח שכתב את הקוד. בדיקות אלה ניתנות לביצוע גם על ידי מישהו אחר, בתנאי שיש לבודק גישה לקוד. מפתחים עשויים לבצע לסירוגין פיתוח רכיבים ומציאה

ותיקון של פגמים. בדרך כלל המפתחים כותבים ומבצעים בדיקות אחרי שכתבו את הקוד לרכיב. אולם, כתיבת בדיקות אוטומטיות יכולה לבוא לפני כתיבת הקוד של הרכיב, בייחוד בפיתוח אג'ילי.

דוגמה לכך היא שיטת פיתוח מונחה בדיקות (TDD – Test Driven Development). פיתוח מונחה בדיקות הינו פיתוח בסבבים המבוסס על סבבים של פיתוח מקרי בדיקה אוטומטים, לאחריהן בנייה ואינטגרציה של יחידות קטנות של קוד, הרצה של בדיקות הרכיב, תיקון בעיות שהתגלו, ולבסוף ארגון הקוד מחדש (refactoring). תהליך זה נמשך עד שהרכיב נבנה בשלמותו וכל בדיקות הרכיב עברו בהצלחה. פיתוח מונחה בדיקות הוא דוגמה לגישה שבה פיתוח הבדיקות קודם לפיתוח הקוד. פיתוח מונחה בדיקות שנוצר במקור כחלק מגישת eXtreme Programming (XP), התפשט לגישות אחרות של אג'ייל וגם למודלים של פיתוח סדרתי (ראו ISTQB-AT Foundation Level Agile Tester Extension).

2.2.2 בדיקות אינטגרציה *(integration testing)*

המטרות של בדיקות אינטגרציה

בדיקות אינטגרציה מתמקדות בממשקים בין רכיבים או מערכות. המטרות של בדיקות אינטגרציה כוללות:

- הפחתת סיכונים
- וידוא שההתנהגות הפונקציונלית והלא פונקציונלית של הממשקים הן בהתאם לעיצוב ולאפיון
- בניית אמון באיכות הממשקים
- מציאת פגמים (שעשויים להיות בממשקים עצמם או בתוך הרכיבים או המערכות)
- מניעת זליגה של פגמים לרמות בדיקה גבוהות יותר

כמו בבדיקות רכיבים, במקרים בהם מיישמים בדיקות נסיגה אוטומטיות ברמה זו, הן מספקות אמון בכך שהשינויים שנעשו לא שברו מערכות, ממשקים או רכיבים קיימים.

תוכנית הלימודים הזו מתארת שתי רמות שונות של בדיקות ממשקים, שניתן לבצע על מושאי בדיקות בגדלים שונים, כדלקמן:

- בדיקות אינטגרציה של רכיבים מתמקדות בממשקים ובפעולות (interactions) המתקיימות בין רכיבים. בדיקות אינטגרציה של רכיבים מבוצעות לאחר בדיקות רכיבים, ובדרך כלל הן בדיקות אוטומטיות. בפיתוח בסבבים ובפיתוח מצטבר, בדיקות אינטגרציה של רכיבים הן בדרך כלל חלק מתהליך אינטגרציה רציפה (continuous integration).
- בדיקות אינטגרציה של מערכות מתמקדות בממשקים וביחסי הגומלין בין מערכות, חבילות ו-microservices. בדיקות אינטגרציה של מערכות יכולות לכסות גם ממשקים ויחסי גומלין עם ארגונים חיצוניים (למשל, שירותים המתקבלים דרך הרשת). במקרה כזה, הארגון המפתח לא שולט בממשקים החיצוניים, דבר שיכול להציב אתגרים שונים לבדיקות (לדוגמה, להבטיח שפגמים בקוד של הארגון החיצוני שחוסמים הרצת חלק מהבדיקות, ייפתרו; הקמה של סביבות בדיקה, וכו'). ניתן לבצע את בדיקות האינטגרציה של מערכות אחרי בדיקות המערכת או במקביל לביצוע בדיקות המערכת (דבר זה נכון גם בפיתוח המשכי וגם בפיתוח בסבבים ובפיתוח מצטבר).

בסיס הבדיקות (test basis)

דוגמאות לתוצרים שיכולים לשמש כבסיס הבדיקות לבדיקות אינטגרציה כוללות:

- עיצוב תוכנה ומערכת

- תרשימי רצף (sequence diagram)
- מפרטי פרוטוקולים של ממשקים ותקשורת
- מקרי שימוש (use case)
- ארכיטקטורה של רמת הרכיב או המערכת
- תיאור התהליכים שהמערכת מבצעת (workflow)
- הגדרות ממשקים חיצוניים

מושא הבדיקות (test objects)

מושאי בדיקות אופייניים לבדיקות אינטגרציה כוללים:

- תת מערכות
- בסיסי נתונים
- תשתית
- ממשקים
- ממשקי תכנות יישומים (Application Programming Interface; API)
- Microservices

פגמים וכשלים אופייניים

דוגמאות לפגמים וכשלים שבדיקות אינטגרציה של רכיבים עוזרות לגלות, כוללות:

- נתונים שגויים, נתונים חסרים, או קידוד נתונים שגוי
- רצף שגוי או תזמון שגוי של קריאות לממשקים
- אי התאמה בין ממשקים
- כשלים בתקשורת בין רכיבים
- כשלים שלא מטופלים או מטופלים בצורה לא נכונה בתקשורת בין רכיבים
- הנחות שגויות על משמעות, יחידות המדידה, או תחום הערכים המועבר בין רכיבים

דוגמאות לפגמים וכשלים אופייניים שבדיקות אינטגרציה של מערכות עוזרות לגלות, כוללות:

- חוסר עקביות במבני של הודעות בין מערכות
- נתונים שגויים, נתונים חסרים או קידוד נתונים שגוי
- אי התאמה בין ממשקים
- כשלים בתקשורת בין מערכות
- כשלים שלא מטופלים או מטופלים בצורה לא נכונה בתקשורת בין מערכות
- הנחות שגויות על משמעות, יחידות המדידה, או תחום הערכים המועבר בין מערכות
- אי עמידה בתקני בטיחות מחייבים

גישות ספציפיות ותחומי אחריות

בדיקות אינטגרציה של רכיבים ובדיקות אינטגרציה של מערכות צריכות להתמקד בשילוב (אינטגרציה) עצמו. לדוגמה, אם מודול A משתלב עם מודל B, הבדיקות אמורות להתמקד בתקשורת בין המודולים ולא בפונקציונליות של כל מודול בנפרד, מאחר וזה אמור היה להיות מכוסה בשלב בדיקות הרכיבים. אם מערכת X משתלבת עם מערכת Y, הבדיקות אמורות להתמקד בתקשורת בין המערכות, ולא בפונקציונליות של כל מערכת בנפרד, מאחר וזה אמור היה להיות מכוסה בשלב בדיקות המערכת.

בדיקות פונקציונליות, בדיקות לא פונקציונליות ובדיקות מבניות מתאימות ברמה זו.

במקרים רבים, בדיקות אינטגרציה של רכיבים הן באחריות המפתחים. בדיקות אינטגרציה של מערכות הן בדרך כלל באחריות הבודקים. עקרונית, בודקים המבצעים בדיקות אינטגרציה של מערכות אמורים להבין את ארכיטקטורת המערכת והיו אמורים להשפיע על תכנון האינטגרציה.

אם התכנון של בדיקות האינטגרציה ואסטרטגיית האינטגרציה נעשה לפני שהרכיבים או המערכות נבנים, ניתן לבנות את הרכיבים או המערכות בסדר שיעשה את הבדיקות יעילות. אסטרטגיות אינטגרציה שיטתיות עשויות להתבסס על הארכיטקטורה של המערכת (לדוגמה, מלמעלה למטה או מלמטה למעלה), על משימות פונקציונליות, רצף עיבוד פעולות (transaction), או כל היבט אחר של המערכת או הרכיב. כדי לפשט את בידוד הפגמים והגילוי המוקדם של פגמים, על האינטגרציה להיות מצטברת (כלומר, הוספה של מספר קטן של רכיבים או מערכות בכל פעם) ולא ב"מפץ גדול" (כלומר, שילוב של הרכיבים או המערכות בצעד בודד אחד). ניתוח סיכונים של הממשקים המסובכים ביותר יכול לעזור ולמקד את בדיקות האינטגרציה.

ככל שתכולת האינטגרציה גדלה, כך נהיה קשה יותר לשייך פגמים לרכיב או מערכת מסוימים, דבר שיוביל להגדלת הסיכון ולהוספת זמן לפתרון תקלות. זו אחת הסיבות שאינטגרציה מתמשכת, שבה התוכנה משתלבת רכיב אחרי רכיב (כלומר, אינטגרציה פונקציונלית), נהיתה שיטה נפוצה. אינטגרציה מתמשכת כוללת במקרים רבים בדיקות נסיגה אוטומטיות, רצוי במספר רמות בדיקה.

2.2.3 ***בדיקות מערכת (system testing)***

המטרות של בדיקות מערכת

בדיקות מערכת מתמקדות בהתנהגות וביכולות של המערכת או המוצר בשלמותם, לעיתים קרובות תוך התמקדות במשימות "קצה לקצה" (end-to-end) שהמערכת יכולה לבצע וההתנהגות הלא פונקציונלית שהמערכת מציגה תוך ביצוע משימות אלה. המטרות של בדיקות מערכות כוללות:

- הפחתת סיכונים
- וידוא שההתנהגות הפונקציונלית והלא פונקציונלית של המערכת הן בהתאם לאפיון ולעיצוב
- ווידוא שהמערכת פותחה בשלמותה ושהיא עובדת כמצופה
- בניית אמון באיכות המערכת בשלמותה
- מציאת פגמים
- מניעת זליגה של פגמים לרמות בדיקה גבוהות יותר או לסביבה התפעולית
- במערכות מסוימות: אימות איכות הנתונים

בדומה לבדיקות רכיבים ובדיקות אינטגרציה, בדיקות נסיגה (אוטומטיות או ידניות) מספקות אמון בכך שהשינויים שנעשו לא פגעו בתכונות קיימות או ביכולות מקצה לקצה. בדרך כלל אלו בדיקות המערכת שמספקות מידע לבעלי העניין לצורך קבלת החלטה לגבי מסירת המערכת. בדיקות מערכת עשויות גם למלא דרישות חוקיות או רגולטוריות ודרישות תקנים.

עקרונית, סביבת הבדיקות צריכה להיות דומה לסביבת המטרה או התפעול הסופיות.

בסיס הבדיקות (test basis)

דוגמאות לתוצרים שיכולים לשמש כבסיס הבדיקות לבדיקות מערכת כוללות:

- מפרטי דרישות של התוכנה והמערכת (פונקציונליים ולא פונקציונליים)
- דוחות ניתוח סיכונים
- מקרי שימוש (use case)
- אפוסים (epic) וסיפורי משתמש (בפיתוח אג'ילי)
- מודלים של התנהגות מערכת
- תרשימי מצבים
- מדריך למשתמש ומדריכים לשימוש במערכת

מושא הבדיקות (test objects)

מושאי בדיקות אופייניים לבדיקות מערכת כוללים:

- יישומים
- מערכות חומרה/תוכנה
- מערכות הפעלה
- המערכת הנבדקת (SUT – System Under Test)
- תצורת מערכת (system configuration) והערכים בהם משתמשים לקונפיגורציה של המערכת (configuration data)

פגמים וכשלים אופייניים

דוגמאות לפגמים וכשלים אופייניים שבדיקות מערכת עוזרות לגלות, כוללות:

- חישובים שגויים
- התנהגות לא נכונה או לא צפויה של המערכת (פונקציונלית ולא פונקציונלית)
- זרימת בקרה ו/או זרימת נתונים שגויים בתוך המערכת
- כשל בביצוע משימות פונקציונליות מקצה לקצה בצורה שלמה ונכונה
- כשל של המערכת לעבוד בצורה נכונה בסביבה התפעולית
- כשל של המערכת לעבוד כמתואר במדריך למשתמש ובמדריכי השימוש במערכת

גישות ספציפיות ותחומי אחריות

בדיקות מערכת צריכות להתמקד בהתנהגות הפונקציונלית והלא פונקציונלית הכוללת, מקצה לקצה, של המערכת בשלמותה. על בדיקות מערכות להשתמש בטכניקות המתאימות ביותר (ראו פרק 4) להיבטים של המערכת הנבדקת. לדוגמה, ניתן ליצור טבלת החלטות כדי לאמת שהתנהגות פונקציונלית מסוימת תואמת למתואר בחוקים העסקיים.

בדיקות מערכת מבוצעות בדרך כלל על ידי בודקים עצמאיים. פגמים במפרטים (לדוגמה, סיפורי משתמש חסרים, דרישות עסקיות שגויות, וכו') יכולים לגרום לחוסר הבנה או לחוסר הסכמה לגבי

ההתנהגות הצפויה של המערכת. מקרים כאלה יכולים לגרום לתוצאות שגויות (false positive or false negative), דבר שיגרום לבזבוז זמן ולהפחתת יעילות גילוי הפגמים. שילוב מוקדם של בודקים בהגדרה מדויקת ומפורטת של סיפורי משתמש או בפעילויות בדיקה סטטיות, כמו סקירות (review), יעזרו להפחית מקרים כאלה.

2.2.4 *בדיקות קבלה (acceptance testing)*

המטרות של בדיקות קבלה

בדיקות קבלה, כמו בדיקות מערכת, מתמקדות בדרך כלל בהתנהגות וביכולות של המערכת או המוצר בשלמותם. המטרות של בדיקות קבלה כוללות:

- בניית אמון באיכות המערכת בשלמותה
- ווידוא שהמערכת פותחה בשלמותה ושהיא עובדת כמצופה
- וידוא שההתנהגות הפונקציונלית והלא פונקציונלית של המערכת הן בהתאם לאפיון

בדיקות קבלה עשויות לספק מידע שישמש להערכת מוכנות המערכת לפריסה ולשימוש על ידי הלקוח (משתמש הקצה). פגמים עשויים להתגלות בזמן בדיקות קבלה, אבל במקרים רבים מציאת פגמים אינה המטרה ומציאת מספר משמעותי של פגמים בזמן בדיקות קבלה עלול במקרים מסוימים להיחשב כסיכון פרויקט משמעותי. בדיקות קבלה עשויות למלא דרישות חוקיות או רגולטוריות ודרישות תקנים.

צורות נפוצות של בדיקות קבלה כוללות:

- בדיקות קבלת משתמש
- בדיקות קבלה תפעוליות
- בדיקות קבלה חוזיות ורגולטוריות
- בדיקות אלפא ובדיקות ביתא

כל אלו מתוארות בסעיפים הבאים

בדיקות קבלת משתמש (UAT – User Acceptance Testing)

בדיקות קבלת משתמש מתמקדת בדרך כלל בבדיקת התאמת המערכת לשימוש (fitness for use) על ידי המשתמשים הצפויים בסביבת עבודה תפעולית אמיתית או בסימולציה. המטרה המרכזית היא בניית אמון בכך שהמשתמשים יכולים להשתמש במערכת על פי צרכיהם, למלא את הדרישות ולבצע תהליכים עסקיים עם מינימום קשיים, עלות וסיכון.

בדיקות קבלה תפעוליות (OAT – Operational Acceptance Testing)

בדיקות הקבלה של המערכת בידי מנהלי תפעול או מנהלי מערכת מבוצעים בדרך כלל בסביבה תפעולית (או העתק של הסביבה התפעולית המשמש לפיתוח ובדיקות – "sandbox"). הבדיקות מתמקדות בהיבטים תפעוליים, ועשויים לכלול:

- בדיקות גיבוי ושיחזור
- התקנה, הסרה ושדרוג
- התאוששות מאסון
- ניהול משתמשים
- משימות תחזוקה

- משימות עומס נתונים והגירת נתונים (data migration)
- בדיקות חולשות אבטחה
- בדיקות ביצועים

המטרה המרכזית של בדיקות קבלה תפעוליות היא בניית אמון (של המפעילים או מנהלי המערכת) בכך שהמערכת תעבוד בצורה נכונה עבור המשתמשים בסביבה התפעולית, גם תחת תנאים קשים ויוצאי דופן.

בדיקות קבלה חוזיות ורגולטוריות

בדיקות קבלה חוזיות מבוצעות אל מול קריטריון הקבלה שנקבע בחוזה לפיתוח תוכנה ייחודית. . יש להגדיר קריטריון קבלה כשהצדדים מסכמים את תנאי החוזה. בדיקות קבלה חוזיות מתבצע בדרך כלל על ידי משתמשים או על ידי בודקים עצמאיים.

בדיקות קבלה רגולטוריות מבוצעות כדי לוודא שהמערכת ממלאת דרישות של תקנות בהן היא אמורה לעמוד, כמו למשל תקנות ממשלתיות, תקנות חוקיות או תקני בטיחות. בדיקות קבלה רגולטוריות מבוצעות בדרך כלל על ידי משתמשים או על ידי בודקים עצמאיים, כשלעיתים נציגי ארגון התקינה נוכחים בשעת ביצוע הבדיקות או מבצעים ביקורת של התוצאות לאחר ביצוען.

המטרה המרכזית של בדיקות קבלה חוזיות ורגולטוריות היא בניית אמון בכך שדרישות החוזה או התקינה התמלאו.

בדיקות אלפא וביתא

בדיקות אלפא וביתא אופייניות לפיתוח של תוכנה כ"מוצר מדף" מסחרי, כשהארגון המפתח רוצה לקבל משוב ממשתמשים קיימים או פוטנציאלים, מלקוחות ו/או ממפעילים לפני שמוצר התוכנה יוצא לשוק. בדיקות אלפא נערכות באתר הארגון המפתח, לא על ידי צוות הפיתוח, אלא על ידי לקוחות ו/או מפעילים קיימים או פוטנציאלים, או על ידי צוות בדיקות עצמאי. בדיקות ביתא נערכות על ידי לקוחות ו/או מפעילים קיימים או פוטנציאלים באתר שלהם עצמם. בדיקות ביתא יכולות להתבצע אחרי בדיקות אלפא, או להיעשות גם בלי שבדיקות אלפא בוצעו קודם לכן.

מטרה אחת של בדיקות אלפא וביתא היא בניית אמון אצל לקוחות ו/או מפעילים קיימים ופוטנציאלים, בכך שהם יוכלו להשתמש במערכת במצבים רגילים, יומיומיים, ובסביבה התפעולית שלהם, כדי להשיג את מטרותיהם עם מינימום קשיים, עלויות וסיכונים. מטרה נוספת עשויה להיות זיהוי פגמים הקשורים לתנאים והסביבות בהם המערכת תהיה בשימוש, במיוחד כשקשה לצוות הפיתוח לשכפל תנאים וסביבות אלה.

בסיס הבדיקות (test basis)

דוגמאות לתוצרים שיכולים לשמש כבסיס הבדיקות לכל צורה של בדיקות קבלה כוללות:

- תהליכים עסקיים
- דרישות משתמש או דרישות עסקיות
- תקינה (רגולציה), חוזים חוקיים ותקנים
- מקרי שימוש
- דרישות מערכת
- מדריך למשתמש ומדריכים לשימוש במערכת
- תהליכי התקנה

- דוחות ניתוח סיכונים

בנוסף, אחד או יותר מהתוצרים הבאים יכול לשמש כבסיס בדיקות להגדרת מקרי בדיקה עבור בדיקות קבלה תפעוליות:

- תהליכי גיבוי ושחזור
- תהליכי התאוששות מאסון
- דרישות לא פונקציונליות
- מסמכים תפעוליים
- הוראות התקנה ופריסה
- יעדי ביצועים
- בסיסי נתונים
- תקינה (רגולציה) ותקני אבטחה

מושאי בדיקות אופייניים (test objects)

מושאי הבדיקות לכל צורה של בדיקות קבלה כוללות:

- המערכת הנבדקת
- תצורת מערכת (system configuration) והערכים בהם משתמשים לקונפיגורציה של המערכת (configuration data)
- תהליכים עסקיים למערכת משולבת במלואה
- מערכות התאוששות ואתרים חלופיים (להמשכיות עסקית ובדיקות התאוששות מאסון)
- תהליכי תפעול ותחזוקה
- טפסים
- דוחות
- נתונים תפעוליים אמיתיים או מוסבים[5]

פגמים וכשלים אופייניים

דוגמאות לפגמים וכשלים אופייניים שבדיקות קבלה עוזרות לגלות, כוללות:

- תהליכי עבודה במערכת אינם עונים על הדרישות העסקיות או דרישות המשתמש
- חוקים עסקיים לא מיושמים נכון
- המערכת לא עונה על דרישות חוזיות או רגולטוריות
- כשלים לא פונקציונליים כגון חולשות אבטחה, ביצועים לא יעילים תחת עומסים גבוהים או כשל בפעולה על פלטפורמה נתמכת

[5] נתונים מוסבים הם נתונים אמיתיים שעברו התאמה על מנת להגן על פרטים חסויים

גישות ספציפיות ותחומי אחריות

האחריות לבדיקות קבלה נופלת לעיתים קרובות על הלקוחות, המשתמשים העסקיים, מנהלי מוצר או המפעילים של המערכת; גם בעלי עניין אחרים עשויים להיות מעורבים.

בדיקות קבלה נתפסות לעיתים קרובות השלב האחרון במחזור חיי פיתוח המשכי, אבל הן עשויות להתבצע גם בשלבים אחרים, לדוגמה:

- בדיקות קבלה של מוצר מדף עשויות להתבצע בזמן התקנת המוצר או שילובו בסביבה תפעולית
- בדיקות קבלה של פונקציונלית חדשה שהתווספה למערכת קיימת עשויות להתבצע לפני בדיקות מערכת

בפיתוח בסבבים, צוותי פרויקט עשויים ליישם צורות שונות של בדיקות קבלה במהלך או בסוף כל מחזור, למשל כאלו שמתמקדות באימות של מאפיין חדש אל מול קריטריון הקבלה שלו וכאלו שמתמקדות בבדיקה שמאפיין חדש עונה על צרכי המשתמשים. בנוסף, בדיקות אלפא וביתא עשויות להתבצע לקראת סוף כל סבב, אחרי השלמת כל סבב, או אחרי סדרה של סבבים. בדיקות קבלת משתמש, בדיקות קבלה תפעוליות, בדיקות קבלה רגולטוריות ובדיקות קבלה חוזיות עשויות גם הן להתבצע לקראת סוף כל סבב, אחרי השלמת כל סבב או אחרי סדרה של סבבים.

2.3 סוגי בדיקות

סוג בדיקה הינו קבוצה של פעילויות בדיקה המכוונות לבדיקה של מאפיינים ספציפיים של מערכת תוכנה, או חלק ממערכת, בהתבסס על מטרות בדיקה מוגדרות וייחודיות. מטרות אלה עשויות לכלול:

- הערכת איכות מאפיינים פונקציונליים, כגון מימוש מלא ונכון והתאמה לצרכים
- הערכת איכות מאפיינים לא פונקציונליים, כגון אמינות, יעילות ביצועים, אבטחה, תאימות (compatibility), ושימושיות (usability)
- הערכה האם מבנה או ארכיטקטורת הרכיב או המערכת נעשו בהתאם לאפיון, באופן מלא ובצורה נכונה
- הערכת השפעת שינויים, לדוגמה אימות תיקון של פגמים (בדיקות אימות) וחיפוש של שינויים לא מכוונים בהתנהגות כתוצאה משינויים בתוכנה או בסביבה (בדיקות נסיגה)

2.3.1 *בדיקות פונקציונליות*

בדיקות פונקציונליות של מערכת כוללות בדיקות שמעריכות פונקציות שהמערכת אמורה לבצע. דרישות פונקציונליות עשויות להיות מתוארות בתוצרי עבודה כגון מפרט דרישות עסקיות, אפוסים, סיפורי משתמש, מקרי שימוש או מפרטים פונקציונליים. יתכן גם שהדרישות לא יהיו מתועדות. הפונקציות הן "מה" המערכת אמורה לעשות.

בדיקות פונקציונליות אמורות להתבצע בכל רמות הבדיקות (לדוגמה, בדיקות של רכיבים עשויות להתבסס על מפרטי הרכיב), אם כי המיקוד שונה בכל רמה (ראו פרק 2.2).

בדיקות פונקציונליות בודקות את התנהגות התוכנה, כך שניתן להשתמש בטכניקות קופסה שחורה להפקת תנאי בדיקה ומקרי בדיקה לבדיקת התפקוד של רכיב או מערכת (ראו פרק 4.2).

ניתן למדוד את עד כמה הבדיקות הפונקציונליות יסודיות באמצעות כיסוי פונקציונלי. כיסוי פונקציונלי נותן הערכה עד כמה פריט פונקציונלי הופעל על ידי הבדיקות, ומבוטא באחוזי הכיסוי של אותו פריט פונקציונלי. לדוגמה, באמצעות נֶעֱקָבוּת בין בדיקות ודרישות פונקציונליות, ניתן לחשב את אחוז הדרישות שכוסו על ידי בדיקות, וכך לגלות פערים בכיסוי.

עיצוב וביצוע בדיקות פונקציונליות עשויים לדרוש מיומנויות או ידע מיוחדים, כגון ידע על הבעיה העסקית שהתוכנה פותרת (לדוגמה, תוכנה למודלים גיאולוגיים לתעשיית הגז והנפט) או על התחום העסקי אליו שייכת התוכנה (לדוגמה, תוכנה למשחקי מחשב שמספקת חוויה אינטראקטיבית).

2.3.2 *בדיקות לא פונקציונליות*

בדיקות לא פונקציונליות של מערכת מעריכות מאפייני מערכת ותוכנה כגון שימושיות, יעילות הביצועים, ואבטחה. עוד על סיווג של מאפייני איכות של מוצר תוכנה ראו תקן ISO (ISO/IEC 25010). בדיקות לא פונקציונליות בודקות "כמה טוב" המערכת מתפקדת.

בניגוד לתפיסה הקיימת, ניתן ולעיתים אף רצוי לבצע בדיקות לא פונקציונליות בכל רמות הבדיקה, ויש לבצע אותן מוקדם ככל האפשר. גילוי מאוחר של פגמים לא פונקציונליים עשוי להיות סיכון גבוה עבור הצלחת הפרויקט.

ניתן להשתמש בטכניקות קופסה שחורה (ראו פרק 4.2) כדי להפיק תנאי בדיקה ומקרי בדיקה עבור בדיקות לא פונקציונליות. לדוגמה, ניתן להשתמש בניתוח ערכי גבול להגדרת מצבי מאמץ (stress) לבדיקות ביצועים.

ניתן למדוד עד כמה הבדיקות הלא פונקציונליות יסודיות בעזרת כיסוי לא פונקציונלי. כיסוי לא פונקציונלי נותן הערכה עד כמה פריט לא-פונקציונלי הופעל על ידי הבדיקות, ומבוטא באחוזי הכיסוי של אותו פריט לא פונקציונלי. לדוגמה, בהינתן רשימה של מכשירים ניידים שאמורים להיתמך על ידי אפליקציה ,ניתן לחשב את אחוז המכשירים שנבדקו בבדיקות תאימות (compatibility testing), ולגלות פערים בכיסוי, אם ישנם.

עיצוב וביצוע בדיקות לא פונקציונליות עשויים לדרוש מיומנויות או ידע מיוחדים, כגון ידע על חולשות שטבועות בעיצוב או בטכנולוגיה (לדוגמה, חולשות אבטחה הקשורות לשפת תכנות מסוימת) או על בסיס משתמשים מסוים (לדוגמה, המאפיינים של משתמשים במערכות ניהול מתקנים רפואיים).

למידע נוסף על בדיקות של מאפייני איכות לא פונקציונליים ראו את תכניות הלימודים של ISTQB:

- ISTQB-ATA Advanced Level Test Analyst
- ISTQB-ATTA Advanced Level Technical Test Analyst
- ISTQB-SEC Advanced Level Security Tester

ותוכניות לימודים נוספות ממסלול המתמחה (specialist) של ISTQB.

2.3.3 *בדיקות קופסה לבנה (white-box testing)*

בדיקות קופסה לבנה מגדירות בדיקות שמבוססות על המבנה הפנימי של המערכת או על יישום המערכת. מבנה פנימי עשוי לכלול קוד, ארכיטקטורה, תהליכים שהמערכת מבצעת (workflow) ו/או זרימת נתונים בתוך המערכת (ראו פרק 4.3).

ניתן למדוד את היקף בדיקות הקופסה הלבנה באמצעות כיסוי מבני. כיסוי מבני הוא המידה בה מספר סוגים של פריטי מבנה מומשו על ידי בדיקות, ומבוטא באחוזי הכיסוי של אותו סוג פריט.

ברמת בדיקות הרכיבים, כיסוי קוד מבוסס על אחוז הקוד ברכיב שנבדק, וניתן למדידה מהיבטים שונים של הקוד (פריטי כיסוי), כגון אחוז המשפטים שנבדקו, או אחוז תוצאות ההחלטה שנבדקו. סוגי כיסוי אלה נקראים ביחד כיסוי קוד. ברמת בדיקות אינטגרציה של רכיבים, בדיקות קופסה לבנה עשויות להתבסס על ארכיטקטורת המערכת, לדוגמה ממשקים בין רכיבים, וכיסוי מבנה יימדד במונחים של אחוז הממשקים שמומשו על ידי בדיקות.

עיצוב והרצת בדיקות קופסה לבנה עשויים לדרוש מיומנויות או ידע מיוחדים, כגון ידע על הצורה בה הקוד בנוי (לדוגמה, לצרכי שימוש בכלי כיסוי קוד), איך נתונים נשמרים (לדוגמה, להערכת שאילתות אפשריות על בסיס נתונים) ואיך להשתמש בכלי כיסוי ולפרש את התוצאות בצורה נכונה.

2.3.4 *בדיקות הקשורות בשינויים (change-related testing)*

כאשר נעשים שינויים במערכת, בין אם כדי לתקן פגם או בגלל שינוי בפונקציונליות או הכנסת תכונה חדשה, יש לבצע בדיקות שיאמתו שהשינויים אכן תיקנו את הפגם או שיישום התכונה החדשה נעשה נכון, ולא גרמו השפעות שלילות לא צפויות.

- בדיקות אימות: לאחר שתוקן פגם, ניתן לבדוק את המערכת עם כל מקרי הבדיקה שנכשלו בשל אותו פגם, כלומר לבצע את הבדיקות מחדש על גרסת התוכנה החדשה. ניתן לבדוק את המערכת גם עם בדיקות חדשות, אם, לדוגמה, הפגם היה תכונה חסרה. לכל הפחות, יש לבצע מחדש את הצעדים לשחזור הכשל שנגרם על ידי הפגם, על גרסה התוכנה החדשה. המטרה של בדיקות אימות הן לאמת שהפגם המקורי תוקן בהצלחה.
- בדיקות נסיגה: יתכן ששינוי שנעשה בחלק אחד של הקוד, בין אם תיקון או כל שינוי אחר, ישפיע בטעות על התנהגות של חלקים אחרים של הקוד, בין אם באותו רכיב, ברכיבים אחרים של אותה מערכת, או אפילו במערכות אחרות. שינויים עשויים לכלול שינויים לסביבה, לדוגמה גרסה חדשה של מערכת ההפעלה או מערכת לניהול בסיס נתונים. תופעות לוואי לא מכוונות אלה נקראות נסיגה. בדיקות נסיגה כוללות הרצת בדיקות לגילוי תופעות לוואי לא מכוונות אלה.

בדיקות אימות ובדיקות נסיגה מבוצעות בכל רמות הבדיקה.

במיוחד במודלים של פיתוח מצטבר ופיתוח בסבבים (לדוגמה, אג'ייל), תכונות חדשות, שינויים לתכונות קיימות וארגון הקוד מחדש (refactor) יוצרים שינויים תכופים בקוד, דבר שדורש בדיקות מבוססות שינוי. בשל האופי המתפתח של המערכת, יש חשיבות גבוהה לבדיקות אימות ובדיקות נסיגה. הדבר נכון במיוחד למערכת אינטרנט של דברים (IoT) בהן פריטים יחידים (לדוגמה, מכשירים) מתעדכנים ומוחלפים בתדירות גבוהה.

סדרות של בדיקות נסיגה מבוצעות מספר רב של פעמים ובעיקרון מתפתחות בקצב איטי, כך שבדיקות נסיגה הן מועמד מושלם לאוטומציה. יש להתחיל אוטומציה של בדיקות אלה מוקדם בחיי הפרויקט (ראו פרק 6).

2.3.5 *סוגי בדיקות ורמות בדיקה*

ניתן לבצע כל אחד מסוגי הבדיקות שהוזכרו למעלה בכל אחד מרמות הבדיקה. כדי להדגים, יינתנו דוגמאות לבדיקות פונקציונליות, בדיקות לא פונקציונליות, בדיקות קופסה לבנה ובדיקות מבוססות שינוי לכל הרמות, עבור יישומון של בנק, ונתחיל בבדיקות פונקציונליות:

- עבור בדיקות רכיבים מעוצבות בדיקות בהתבסס על הדרך בה אמור הרכיב לחשב ריבית דריבית.
- עבור בדיקות אינטגרציה של רכיבים מעוצבות בדיקות בהתבסס על הדרך בה מידע משתמש שמתקבל בממשק המשתמש מועבר ללוגיקה העסקית.
- עבור בדיקות המערכת מעוצבות בדיקות בהתבסס על הדרך בה מחזיקי חשבון יכולים לבקש אשראי בחשבונות העו"ש שלהם.
- עבור בדיקות אינטגרציה של מערכות מעוצבות בדיקות בהתבסס על הדרך בה המערכת משתמשת ב-microservice חיצוני כדי לבדוק את דירוג האשראי של מחזיק חשבון.
- עבור בדיקות קבלה מעוצבות בדיקות בהתבסס על הדרך בה הבנקאי מאשר או דוחה בקשה לקבלת אשראי.

הדוגמאות הבאות הן לבדיקות לא פונקציונליות:

- עבור בדיקות רכיבים מעוצבות בדיקות ביצועים להערכת מספר מחזורי המעבד (CPU) הנדרשים לביצוע חישוב מסובך של ריבית.
- עבוד בדיקות אינטגרציה של רכיבים מעוצבות בדיקות אבטחה לבדיקת חולשה מסוג גלישת חוצץ (buffer overflow) עקב נתונים שעוברים ממשק המשתמש ללוגיקה העסקית.
- עבור בדיקות המערכת מעוצבות בדיקות ניידות תוכנה (portability testing) שבודקות אם שכבת הייצוג (presentation layer) עובדת עם כל הדפדפנים הנתמכים וכל המכשירים הניידים הנתמכים.
- עבור בדיקות אינטגרציה של מערכות מעוצבות בדיקות מהימנות (reliability testing) להערכת עמידות המערכת במקרה בו ה-microservice שאמור לתת את דירוג האשראי לא מגיב.
- עבור בדיקות קבלה מעוצבות בדיקות שימושיות להערכת הנגישות של ממשק תהליך עיבוד האשראי על ידי הבנקאי לאנשים עם מוגבלויות.

הדוגמאות הבאות הן לבדיקות קופסה לבנה:

- עבור בדיקות רכיבים מעוצבות בדיקות להשגת כיסוי משפטים וכיסוי החלטות מלאים (ראו פרק 4.3) עבור כל הרכיבים שמבצעים חישובים פיננסיים.
- עבור בדיקות אינטגרציה של רכיבים מעוצבות בדיקות להעברת נתונים מכל מסך בממשק הדפדפן למסך הבא וללוגיקה העסקית.
- עבור בדיקות מערכת מעוצבות בדיקות לכיסוי רצפים של עמודי אינטרנט שעשויים להתרחש בזמן כתיבת בקשה לקו אשראי.
- עבור בדיקות אינטגרציה של מערכות מעוצבות בדיקות שיריצו את כל סוגי השאילתות האפשריות שנשלחות ל-microservice של דירוג האשראי.
- עבור בדיקות קבלה מעוצבות בדיקות שמכסות את כל המבנים הנתמכים של קבצי הנתונים הפיננסיים וטווחי הערכים עבור העברות בין בנקים.

לבסוף, הדוגמאות הבאות הן לבדיקות מבוססות שינוי:

- עבור בדיקות רכיבים נבנות בדיקות רגרסיה אוטומטיות עבור כל רכיב והן נכללות במערכת האינטגרציה המתמשכת.
- עבור בדיקות אינטגרציה של רכיבים מעוצבות בדיקות שמאמתות תיקונים של פגמים בממשקים כשהתיקונים מועברים למאגר הקוד (code repository).
- עבור בדיקות מערכת, כל הבדיקות של תהליכי עבודה נתונים מבוצעות מחדש אם מסך כלשהו ברצף משתנה.
- עבור בדיקות אינטגרציה של מערכות, בדיקות של המערכת שמתממשקת עם ה-microservice של דירוג האשראי מבוצעות מידי יום כחלק מהפריסה המתמשכת של אותו microservice.
- עבור בדיקות קבלה, כל הבדיקות שנכשלו קודם לכן מבוצעות מחדש לאחר שתוקן פגם שנתגלה בבדיקות קבלה.

למרות שפרק זה מספק דוגמאות לכל סוג בדיקה בכל רמת בדיקה, אין זה הכרחי שלכל תוכנה יהיו את כל סוגי הבדיקות בכל רמות הבדיקה. אולם חשוב להריץ את כל סוגי הבדיקות הרלוונטיים לכל רמת בדיקה, בייחוד ברמת הבדיקה המוקדמת ביותר בה נמצא סוג הבדיקה.

2.4 בדיקות תחזוקה

כשהתוכנה או המערכת מותקנת בסביבות התפעוליות, יש לתחזק אותן. שינויים מכל מיני סוגים הם כמעט בלתי נמנעים במערכות או בתוכנות אחרי מסירה, בין אם כדי לתקן פגמים שהתגלו בזמן שימוש, להוסיף תכונות חדשות או להסיר או לשנות תכונות שנמסרו. תחזוקה נדרשת גם לשיפור מאפייני איכות לא פונקציונליים של רכיב או מערכת במשך חייהם, בייחוד שיפור של יעילות ביצועים, תאימות, מהימנות, אבטחה וניידות תוכנה.

כאשר נעשים שינויים כלשהם כחלק מתחזוקה, יש לבצע בדיקות תחזוקה, הן לשם הערכת מידת ההצלחה בעשיית השינויים והן לבדיקה של תופעות לוואי אפשריות (לדוגמה, נסיגות) בחלקים של המערכת בהם לא נעשו שינויים (שהם בדרך כלל רוב המערכת). בדיקות תחזוקה מתמקדות בבדיקות השינויים שנעשו במערכת, כמו גם בדיקות החלקים שלא השתנו שעשויים להיות מושפעים מהשינויים. תחזוקה יכולה לכלול גרסאות מתוכננות וגרסאות לא מתוכננות (תיקוני חירום, hot fixes).

גרסת תחזוקה עשויה לדרוש בדיקות תחזוקה במספר רמות בדיקה, תוך שימוש בסוגי בדיקות שונים, בהתבסס על תכולת השינוי. תכולת בדיקות התחזוקה תלויה ב:

- רמת הסיכון של השינוי, לדוגמה המידה בה האזור שהשתנה בתוכנה מתקשר עם רכיבים אחרים או מערכות אחרות
- גודל המערכת הקיימת
- גודל השינוי

2.4.1 *גורמים לתחזוקה*

ישנן מספר סיבות לתחזוקה של תוכנה, ובעקבותיה לבדיקות תחזוקה, עבור שינויים מתוכננים ושינויים לא מתוכננים.

ניתן לסווג את הגורמים לתחזוקה כדלהלן:

- שינוי (modification), כגון הרחבות מתוכננות (לדוגמה, גרסה חדשה), שינויים מתקנים ושינויי חירום, שינויים לסביבה התפעולית (כגון עדכונים מתוכננים של מערכת ההפעלה או בסיסי נתונים), עדכונים של תוכנת מוצר מדף, וטלאים לתיקון פגמים וחולשות
- הגירה (migration), כגון מעבר מפלטפורמה אחת לאחרת, דבר שדורש בדיקות תפעוליות של הסביבה החדשה כמו גם של התוכנה שהשתנתה, או בדיקות של המרת נתונים כאשר נתונים ממערכת אחרת יועברו למערכת המתוחזקת
- פרישה (retirement), כאשר מערכת מגיעה לסוף חייה

כאשר יישום או מערכת פורשים, נדרשות בדיקות להגירה של נתונים או העברת נתונים לארכיון במידה ונדרשות תקופות שמירת נתונים ארוכות. יתכן וידרשו גם בדיקות של תהליכי אחזור/שחזור אחרי ארכון לתקופות ארוכות. בנוסף, יתכן וידרשו בדיקות נסיגה לוודא שכל פונקציונליות שנשמרת ממשיכה לעבוד.

עבור מערכות אינטרנט של דברים (IoT), בדיקות תחזוקה עשויות להיגרם על ידי הוספה של דברים חדשים לגמרי או כאלה שעברו שינוי, כגון התקני חומרה או שירותי תוכנה, לתוך המערכת הכוללת. בדיקות התחזוקה למערכת כאלה שמות דגש מיוחד על בדיקות אינטגרציה ברמות שונות (לדוגמה, ברמת הרשת, ברמת היישום) ועל היבטי אבטחה, בייחוד כאלה שקשורים לנתונים אישיים.

2.4.2 *ניתוח השפעה עבור תחזוקה*

ניתוח השפעה מעריך את השינויים שנעשו לגרסת תחזוקה כדי לזהות את ההשלכות המתוכננות כמו גם תופעות לוואי מתוכננות ותופעות לוואי אפשריות של שינוי. בנוסף יש לזהות את האזורים במערכת שיושפעו מהשינוי. ניתוח השפעה יכול גם לעזור בזיהוי השפעת השינוי על בדיקות קיימות. יש לבדוק את

תופעות הלוואי ואת האזורים במערכת שיושפעו מהשינוי לנסיגות אפשריות, אחרי שמעדכנים בדיקות קיימות שעשויות להיות מושפעות מהשינוי.

ניתן לעשות את ניתוח ההשפעה לפני שנעשה השינוי, כדי להחליט אם יש לבצע את השינוי, בהתבסס על ההשלכות האפשריות על אזורים אחרים במערכת.

ניתוח השפעה יכול להיות קשה אם:

- מפרטים (לדוגמה, דרישות עסקיות, סיפורי משתמש, ארכיטקטורה) אינם מעודכנים או חסרים
- מקרי בדיקה לא מתועדים או חסרים
- נֶעֱקָבוּת דו כיוונית בין הבדיקות ובסיס הבדיקות לא תוחזקה
- אין כלים תומכים או שהם חלשים
- האנשים המעורבים חסרים ידע על המערכת או בתחום הטכנולוגי או העסקי
- לא ניתנה תשומת לב מספקת לתחזוקתיות של התוכנה בזמן הפיתוח

3 בדיקות סטטיות — 135 דקות

מושגים

סקירה אד-הוק (ad hoc reviewing), סקירה מבוססת רשימות ביקורת (checklist-based reviewing), בדיקות דינמיות (dynamic testing), סקירה פורמלית (רשמית; formal review), סקירה לא פורמלית (לא פורמלית; informal review), ביקורת (inspection), קריאה מבוססת נקודת מבט (perspective-based reading), סקירה (review), סקירה מבוססת תפקיד (role-based review), סקירה מבוססת תרחיש (scenario-based review), ניתוח סטטי (static analysis), בדיקות סטטיות (static testing), סקירה טכנית (technical review), דיון מודרך (walkthrough)

יעדי הלימוד של פרק בדיקות סטטיות

3.1 יסודות הבדיקות הסטטיות

FL-3.1.1 (K1) זהה סוגי תוצרים שניתן לבחון בעזרת טכניקות שונות של בדיקות סטטיות

FL-3.1.2 (K2) הסבר את הערך של בדיקות סטטיות בעזרת דוגמאות

FL-3.1.3 (K2) הסבר את ההבדלים בין טכניקות סטטיות ודינמיות, תוך התחשבות בהבדלים בין המטרות, סוגי הפגמים שניתן לזהות, והתפקיד של טכניקות אלו במחזור חיי התוכנה

3.2 תהליך סקירה

FL-3.2.1 (K2) סכם את הפעילויות של תהליך סקירת התוצרים

FL-3.2.2 (K1) זהה את התפקידים השונים ואת תחומי האחריות בסקירה רשמית

FL-3.2.3 (K2) הסבר את ההבדלים בין סוגי הסקירה השונים: סקירה לא רשמית, דיון מודרך, סקירה טכנית וביקורת

FL-3.2.4 (K3) יישם טכניקת סקירה על תוצר בכדי למצוא פגמים

FL-3.2.5 (K2) הסבר את הגורמים שתורמים לסקירה מוצלחת

3.1 יסודות הבדיקות הסטטיות

בניגוד לבדיקות דינמיות, שדורשות הרצה של התוכנה הנבדקת, בדיקות סטטיות נעשות על ידי בחינה ידנית של התוצרים (כלומר, סקירות) או שימוש בכלי לניתוח קוד או כלי לניתוח תוצרים אחרים (כלומר, ניתוח סטטי). שני הסוגים של הבדיקות הסטטיות בוחנות את הקוד או תוצר אחר שנבדק מבלי להריץ את הקוד או את התוצר האחר.

ניתוח סטטי חושב למערכת מחשב חיוניות (לדוגמה, תוכנה לתעופה, למערכות רפואיות או מערכות גרעיניות), אבל ניתוח סטטי נהיה חשוב ונפוץ גם במערכות אחרות. לדוגמה, ניתוח סטטי הוא חלק חשוב מבדיקות אבטחה (security testing). ניתוח סטטי משולב לעתים קרובות גם במערכת בניית תוכנה ומסירה אוטומטיות, למשל בפיתוח אג'ילי, מסירה מתמשכת ופריסה מתמשכת.

3.1.1 תוצרים שניתן לבחון על ידי בדיקות סטטיות

ניתן לבדוק כמעט כל תוצר באמצעות בדיקות סטטיות (סקירות ו/או ניתוח סטטי), לדוגמה:

- מפרטים, כולל דרישות עסקיות, דרישות פונקציונליות, ודרישות אבטחה
- אפוסים, סיפורי משתמש, וקריטריון קבלה
- מפרטי ארכיטקטורה ועיצוב
- קוד
- מכלול מרכיבי הבדיקות (בּוֹדְקָה), כולל תכניות בדיקה, מקרי בדיקה, הליכי בדיקה, ותסריטי בדיקה אוטומטיים
- מדריכי משתמש
- עמודי אינטרנט
- חוזים, תכניות פרויקט, לוחות זמנים ותקציבים
- מודלים, כמו תרשימי פעילות, שניתן להשתמש עבור בדיקות מבוססות מודל (Model-Based testing) (ראו ISTQB-MBT Foundation Level Model-Based Tester Extension ו- Kramer 2016)

ניתן ליישם סקירות עבור כל תוצר שהמשתתפים יודעים איך לקרוא ולהבין. ניתן ליישם ניתוח סטטי בצורה יעילה לכל תוצר עם מבנה פורמלי (בדרך כלל קוד ומודלים) עבורו קיים כלי מתאים לניתוח סטטי. ניתן ליישם ניתוח סטטי גם עם כלים שמעריכים תוצרים שכתובים בשפה טבעית, כמו דרישות (לדוגמה, בדיקת איות, דקדוק והקריאות של הטקסט (readability)).

3.1.2 יתרונות של בדיקות סטטיות

טכניקות לבדיקות סטטיות מספקות מגוון של יתרונות. כאשר הן מיושמות מוקדם במחזור חיי התוכנה, בדיקות סטטיות מאפשרות גילוי מוקדם של פגמים לפני ביצוע בדיקות דינמיות (למשל, בסקירות של דרישות או מפרטי עיצוב, חידוד עתודת המוצר, וכו'). איתור פגמים בשלב מוקדם בתהליך הפיתוח יוזיל בדרך כלל את עלות תיקון הפגמים, בייחוד בהשוואה לפגמים שמתגלים אחרי שהתוכנה נמסרה ונמצאת בשימוש. שימוש בטכניקות בדיקה סטטיות לאיתור פגמים ותיקונם המיידי, כמעט תמיד יהיה יותר זול לארגון מאשר תיקון פגמים שנמצאו בעזרת בדיקות דינמיות, במיוחד כשמתחשבים בעלויות הנוספות הקשורות בעדכון תוצרים אחרים וביצוע בדיקות אימות ובדיקות נסיגה.

יתרונות נוספים של בדיקות סטטיות עשויים לכלול:

- זיהוי ותיקון של פגמים בצורה יעילה יותר, לפני לביצוע בדיקות דינמיות

- זיהוי פגמים שלא קל לזהות באמצעות בדיקות דינמיות
- מניעה של פגמים בעיצוב או בקוד על ידי מציאת חוסר עקביות, עמימות, סתירות, חוסרים, אי דיוקים, ודרישות מיותרות
- הגדלת אפקטיביות הפיתוח (לדוגמה, בעזרת עיצוב משופר, קוד קל יותר לתחזוקה)
- הפחתת עלויות וזמני הפיתוח
- הפחתת עלויות וזמני הבדיקות
- הפחתת העלות הכוללת של האיכות למשך חיי התוכנה באמצעות הפחתת כמות התקלות שמתרחשות מאוחר במחזור החיים, או תוך שימוש שוטף
- שיפור התקשורת בין חברי הצוות במהלך ההשתתפות בסקירות

3.1.3 *הבדלים בין בדיקות סטטיות ובדיקות דינמיות*

לבדיקות סטטיות ולבדיקות דינמיות יכולות להיות אותן מטרות (ראו פרק 1.1.1), לדוגמה להעריך את איכות התוצר או לזהות פגמים מוקדם ככל האפשר. בדיקות סטטיות ובדיקות דינמיות משלימות אחת את השנייה על ידי זיהוי סוגים שונים של פגמים.

הבדל מרכזי אחד הוא שבדיקות סטטיות מוצאות פגמים ישירות בתוצרים ואילו בדיקות דינמיות מוצאות כשלים בעת הרצת המוצר הנובעים מפגמים אלה. פגם יכול להימצא בתוצר כלשהו במשך זמן רב מבלי לגרום לכשל כלשהו. יתכן רצף הפעולות הנדרש לצורך איתור הפגם כמעט ולא נמצא בשימוש או שקשה להגיע אליו, כך שלא יהיה קל לבנות ולהריץ בדיקה דינמית שתחשוף אותו. בדיקות סטטיות יוכלו אולי למצוא את הפגם עם הרבה פחות מאמץ.

הבדל נוסף הוא שניתן להשתמש בבדיקות סטטיות לשיפור העקביות והאיכות הפנימית של התוצרים, בעוד שבדיקות דינמיות בדרך כלל מתמקדות בהתנהגות חיצונית נראית לעין.

בהשוואה לבדיקות דינמיות, פגמים טיפוסיים שקל וזול יותר למצוא ולתקן באמצעות בדיקות סטטיות כוללים:

- פגמים בדרישות (לדוגמה, חוסר עקביות, עמימות, סתירות, חוסרים, אי דיוקים ועודפים)
- פגמים בעיצוב (לדוגמה, אלגוריתמים או מבנים של בסיס נתונים לא יעילים, צימוד גבוה (high coupling) או לכידות נמוכה (low cohesion))
- פגמים בקוד (לדוגמה, משתנים עם ערכים לא מוגדרים, משתנים שהוגדרו אך לא נעשה בהם שימוש, קוד לא נגיש, קוד כפול)
- סטייה מתקנים (לדוגמה, אי ציות לתקן כתיבת קוד)
- מפרטי ממשקים שגויים (לדוגמה, שימוש ביחידות מידה שונות במערכת שקוראת ובמערכת שנקראת)
- חולשות אבטחה (לדוגמה, רגישות לגלישת חוצץ, דהיינו כתיבה של יותר מידע מהמותר באיזור שהוקצה בזיכרון)
- פערים או אי דיוקים בנֶעֱקבוּת או כיסוי בסיס הבדיקות (לדוגמה, חוסר בבדיקות לקריטריון קבלה)

יתר על כן, ניתן לזהות את מרבית הסוגים של פגמים בתחזוקתיות באמצעות בדיקות סטטיות (לדוגמה, מידול לא מתאים, שימוש חוזר מועט ברכיבים, קוד קשה לניתוח ושינוי בלי יצירת פגמים חדשים).

3.2 תהליך סקירה

סקירות משתנות בין סקירות רשמיות ללא רשמיות. סקירות לא רשמיות מאופיינות בכך שהן לא נעשות על בסיס תהליך מוגדר ואין להן פלט מתועד רשמי. סקירות רשמיות מאופיינות בהשתתפות של צוות, בתוצאות סקירה מתועדות, ובהליכים מתועדים לעריכת הסקירה. הרשמיות של תהליך סקירה קשורה לגורמים כמו מודל מחזור חיי התוכנה, הבגרות של תהליך הפיתוח, מורכבות התוצר אותו יש לסקור, דרישות חוקיות או של תקינה, ו/או הצורך בתיעוד הביקורת (audit trail).

המוקד של סקירה תלוי במטרות המוסכמות של הסקירה (לדוגמה, מציאת פגמים, הבנה, לימוד משתתפים כמו בודקים או חברי צוות חדשים, או דיון וקבלת החלטה עם קונצנזוס).

תקן ISO (ISO/IEC 20246) מכיל תיאור יותר מעמיק של תהליך הסקירה של תוצרים, כולל תפקידים וטכניקות לסקירה.

3.2.1 תהליך סקירה של תוצרים

תהליך הסקירה כולל את הפעילויות הראשית הבאות:

תכנון

- הגדרת תכולת הסקירה, כולל מטרת הסקירה, אילו מסמכים או חלקי מסמכים יסקרו, ומאפייני האיכות אותם יש להעריך
- הערכת מאמץ ומסגרת זמן
- זיהוי מאפייני הסקירה כמו סוג הסקירה עם תפקידים, פעילויות ורשימות ביקורת (checklist)
- בחירת האנשים שישתתפו בסקירה והקצאת תפקידים
- הגדרת קריטריון כניסה וקריטריון יציאה לסוגי סקירות רשמיות יותר (לדוגמה, ביקורת)
- בדיקה שהושג קריטריון כניסה (לסוגי סקירות רשמיות יותר)

תחילת הסקירה

- הפצת התוצר (פיזית או באמצעים אלקטרוניים) וחומר נוסף, כמו טפסים לרישום הערות, רשימות בדיקה, וכל תוצרי עבודה קשורים
- הסברת התכולה, המטרות, התהליך, התפקידים, והתוצרים למשתתפים
- מתן מענה לשאלות שיתכן ויהיו למשתתפים לגבי הסקירה

סקירה פרטנית

- סקירה של כל התוצר או חלקו
- תיעוד פגמים אפשריים, המלצות ושאלות

תקשורת וניתוח

- שיתוף הפגמים האפשריים שהתגלו (למשל, בפגישת סקירה)
- ניתוח פגמים אפשריים, האצלת בעלות עליהם וקביעת הסטטוס שלהם
- הערכה ותיעוד של מאפייני איכות
- הערכת ממצאי הסקירה אל מול קריטריון היציאה לצרכי קבלת החלטות (דחייה; יש צורך בשינויים משמעותיים; קבלה, אפשר עם שינויים קלים)

תיקון ודיווח

- כתיבת דוחות פגמים עבור הממצאים שדורשים שינויים
- תיקון פגמים שנמצאו (בדרך כלל נעשה על ידי הכותב) בתוצר שנסקר
- דיווח פגמים לאדם או הצוות המתאימים (אם נמצאו בתוצר מקושר לתוצר שנסקר)
- תיעוד סטטוס מעודכן של הפגמים (בסקירה רשמית), עם הסכמה של מי שכתב את ההערה
- איסוף מדדים (עבור סוגי סקירות רשמיות)
- קבלת התוצר כשעומד בקריטריון היציאה

התוצאות של סקירות תוצרים יכולות להשתנות כתלות בסוג הסקירה וברמת הרשמיות שלה, כמו שמתואר בפרק 3.2.3.

3.2.2 *תפקידים ותחומי אחריות בסקירה רשמית*

סקירה רשמית טיפוסית תכלול את התפקידים הבאים:

מחבר (author)

- יוצר את התוצר הנסקר
- מתקן פגמים בתוצר הנסקר (אם נדרש לכך)

הנהלה

- אחראית לתכנון הסקירה
- מחליטה על קיום הסקירות
- מקצה אנשים, תקציב וזמן
- מעקב שוטף אחרי עלות-תועלת
- קבלת החלטות בקרה במידה ומתקבלות תוצאות לקויות

מנחה (facilitator, moderator)

- מבטיח תהליך יעיל של פגישות סקירה (כשנערכות כאלה)
- מגשר, אם צריך, בין נקודות מבט שונות
- לעתים הצלחת הסקירה תלויה בו

מוביל הסקירה

- בעל אחריות כוללת על הסקירה
- מחליט מי יהיה מעורב ומתאם את מקום וזמן הפגישה

סוקרים (reviewer)

- עשויים להיות מומחים בתחום (subject matter expert), אנשים שעובדים על הפרויקט, בעלי עניין בתוצר, ו/או אנשים עם רקע טכנולוגי או עסקי ייחודיים
- יזהו פגמים אפשריים בתוצר הנסקר

- עשויים לייצג נקודות מבט שונות (לדוגמה, בודק, מתכנת, משתמש, מפעיל, מנתח מערכות, מומחה שימושיות, וכו')

רשם (scribe, recorder)

- אוסף את הפגמים האפשריים בשלב הסקירה הפרטנית
- מתעד פגמים אפשריים חדשים, נקודות פתוחות והחלטות שנאספות בפגישת הסקירה (אם נערכת)

בחלק מסוגי הסקירות, אדם אחד יכול למלא יותר מתפקיד אחד, והפעילויות המקושרות לכל תפקיד עשויות להשתנות בהתאם לסוג הסקירה. בנוסף, עם הופעת כלים שתומכים בתהליך הסקירה, בייחוד כאלה שמתעדים פגמים, נקודות פתוחות והחלטות, לעתים אין צורך במתעד.

ניתן לכלול תפקידים נוספים, כמו שמתואר בתקן ISO (ISO/IEC 20246).

3.2.3 *סוגי סקירה*

למרות שיש לסקירות מספר מטרות, אחת המטרות המרכזיות היא לחשוף פגמים. כל סוגי הסקירות יכולים לסייע בזיהוי פגמים, וסוג הסקירה הנבחר יתבסס, בין השאר, על צרכי הפרויקט, המשאבים הזמינים, סוג המוצר וסיכוני המוצר, התחום העסקי והתרבות הארגונית, כמו גם מאפייני בחירה נוספים.

ניתן לסווג סקירות על פי מספר מאפיינים. הרשימות הבאות מציגות את ארבעת סוגי הסקירות המקובלים ביותר, ואת המאפיינים הקשורים אליהם.

סקירה לא פורמלית (לא רשמית; informal review)

(לדוגמה, סקירת חברים (buddy check), צימוד (pairing), סקירה בזוגות (pair review))

- מטרה עיקרית: זיהוי פגמים אפשריים
- מטרות אפשריות נוספות: העלאת רעיונות או פתרונות חדשים, פתרון מהיר לבעיות קטנות
- לא מבוססת על תהליך (מתועד) רשמי
- יתכן ולא תכלול פגישת סקירה
- יתכן ותתבצע על ידי עמית של המחבר (סקירת חברים) או על ידי אנשים נוספים
- ניתן לתעד את התוצאות
- רמת היעילות משתנה בהתאם לסוקרים
- ניתן להשתמש ברשימות בדיקה
- נפוץ בפיתוח אג'ילי

דיון מודרך

- מטרות עיקריות: מציאת פגמים, שיפור של תוצר התוכנה, בחינת אפשרויות יישום חלופיות, הערכת התאמה לתקנים ומפרטים
- מטרות אפשריות נוספות: דיון על טכניקות וסגנונות חלופיים, הדרכה של המשתתפים, השגת הסכמה (consensus)
- אפשרי כהכנה פרטנית לפני פגישת סקירה
- פגישת הסקירה מובלת בדרך כלל על ידי המחבר של התוצר הנסקר

- חובה להשתמש במתעד
- ניתן להשתמש ברשימות בדיקה
- ניתן לעשות בדרך של תרחישים, הרצה על יבש (dry run) או הדמיה (simulation)
- ניתן להוציא דוחות על פגמים אפשריים ודוחות סקירה
- יתכן ויבוצע בדרך לא רשמית או בדרך מאוד רשמית

סקירה טכנית

- מטרה עיקרית: השגת הסכמה, זיהוי פגמים אפשריים
- מטרות אפשריות נוספות: הערכת האיכות ובניית אמון בתוצר, העלאת רעיונות חדשים, עידוד מחברים ומתן אפשרות לשיפור תוצרים עתידיים, בחינת אפשרויות יישום חלופיות
- הסוקרים צריכים להיות עמיתים טכניים של המחבר, ומומחים טכנולוגיים בתחום או בתחומים שונים
- נדרשת הכנה פרטנית לפני פגישת הסקירה
- ניתן לקיים פגישת סגירה, רצוי שתהיה מובלת על ידי מנחה מקצועי (בדרך כלל לא המחבר)
- חובה להשתמש במתעד, רצוי אדם אחר ולא המחבר
- ניתן להשתמש ברשימות בדיקה
- בדרך כלל מופקים דוחות על פגמים אפשריים ודוחות סקירה

ביקורת

- מטרות עיקריות: זיהוי פגמים אפשריים, הערכת האיכות ובניית אמון בתוצר, מניעת פגמים דומים בעתיד באמצעות לימוד של המחבר וניתוח שורש הבעיה
- מטרות אפשריות נוספות: עידוד מחברים ומתן אפשרות לשיפור תוצרים עתידיים, שיפור תהליך פיתוח התוכנה, השגת הסכמה
- נעשית בעזרת תהליך מוגדר עם פלטים מתועדים רשמיים, בהתבסס על חוקים ורשימות בדיקה
- חובה לעשות שימוש בתפקידים מוגדרים בבירור, כמו אלה שמוגדרים בפרק 3.2.2, וניתן להשתמש באדם שיקרא את התוצר בקול בזמן פגישת הסקירה
- נדרשת הכנה פרטנית לפני פגישת הסקירה
- הסוקרים הם עמיתים של המחבר או מומחים בשיטות שונות שרלוונטיים לתוצר
- יש להגדיר קריטריון כניסה וקריטריון יציאה ייחודיים
- חובה להשתמש במתעד
- פגישת הסקירה מובלת על ידי מנחה מקצועי (לא המחבר)
- המחבר יכול לשמש כמוביל הסקירה, כקורא או כמתעד
- מופקים דוחות פגמים ודוחות סקירה
- נאספים מדדים בהם ישתמשו לשיפור תהליך פיתוח התוכנה כולו, כולל תהליך הביקורת

תוצר אחד יכול לעבור יותר מסוג אחד של סקירה. במקרים בהם משתמשים ביותר מסקירה אחת, סדר הסקירות יכול להשתנות. לדוגמה, ניתן לערוך סקירה לא רשמית לפני סקירה טכנית בכדי לוודא שהתוצר מוכן לסקירה הטכנית.

ניתן לעשו את סוגי הסקירות שמתוארים למעלה כסקרי עמיתים (peer review), כלומר, כאלה שנעשים על ידי עמיתים מאותה דרגה במבנה הארגוני.

סוגי הפגמים המתגלים בסקירה משתנים, ותלויים בייחוד בתוצר שנסקר. ראו דוגמאות לפגמים שניתן למצוא על ידי סקירה בתוצרים שונים בפרק 3.1.3, וראו Gilb 1993 למידע נוסף על ביקורת רשמית.

3.2.4 *יישום טכניקות סקירה*

ישנן מספר טכניקות סקירה שניתן ליישם במהלך סקירה פרטנית (כלומר, הכנה פרטנית) לצורך גילוי פגמים. ניתן להשתמש בטכניקות אלה עם כל סוגי הסקירות שתוארו למעלה. היעילות של הטכניקות עשויה להשתנות בהתאם לסוג הסקירה. דוגמאות לטכניקות סקירה פרטנית שונות לסוגי סקירות שונות מפורטות להלן.

אד-הוק

בסקירה אד-הוק הסוקרים לא מקבלים הדרכה או מקבלים הדרכה מועטה איך לבצע את המשימה. הסוקרים לעיתים קוראים את התוצר ברצף, מזהים ומתעדים בעיות שהם נתקלים בהם. סקירה אד-הוק היא טכניקה נפוצה שזקוקה להכנה מעטה. טכניקה זו תלויה מאוד בכישורי הסוקר ועשויה להיתקל בדיווח כפול של בעיות שהתגלו על ידי סוקרים שונים.

סקירה מבוססת רשימות בדיקה

סקירה מבוססת רשימות בדיקה היא טכניקה שיטתית, שבה הסוקרים מזהים בעיות בהתבסס על רשימות בדיקה המופצות בתחילת הסקירה (לדוגמה על ידי המנחה). רשימת בדיקה לסקירה כוללת סדרה של שאלות מבוססות על פגמים אפשריים, שהופקה על סמך ניסיון. רשימות בדיקה צריכה להיות ייחודיות לסוג התוצר הנסקר ויש לתחזק אותם על בסיס קבע כדי לגלות סוגים של בעיות שלא התגלו בסקירות קודמות. היתרון הגדול של סקירה מבוססת רשימה הוא כיסוי שיטתי של סוגי פגמים טיפוסיים. יש לדאוג שהסוקרים לא רק יעבדו לפי הרשימה במהלך הסקירה הפרטנית, אלא גם יחפשו פגמים שלא מופיעים ברשימה.

תרחישים והרצה על יבש

בסקירה מבוססת תרחיש, ניתנות לסוקרים הנחיות מובנות איך לקרוא את התוצר. גישה מכוונת תרחיש מעודדת סוקרים לבצע "הרצה על יבש" על התוצר בדרך שתתבסס על השימוש הצפוי בתוצר (אם התוצר מתועד בדרך מתאימה כמו באמצעות מקרי שימוש). תרחישים אלה מספקים לסוקרים הנחיות שיסייעו בזיהוי סוגי פגמים ייחודיים בצורה טובה יותר מרשימות בדיקה פשוטות. בדומה לסקירות מבוססות רשימות בדיקה, כדי לא לפספס סוגים אחרים של פגמים (לדוגמה, תכונות חסרות), אין להגביל את הסוקרים רק לתרחישים המתועדים.

סקירה מבוססת תפקיד

סקירה מבוססת תפקיד היא טכניקה שבה הסוקרים מעריכים את התוצר מנקודת מבט של בעל עניין יחיד. תפקידים אופייניים כוללים סוגים מסוימים של משתמשי קצה (מנוסים, לא מנוסים, בכירים, ילדים וכו'), ותפקיד ייחודיים בארגון (אחראי ניהול משתמשים, מנהל מערכת, בודק ביצועים וכו').

סקירה מבוססת נקודת מבט

בקריאה מבוססת נקודת מבט, בדומה לסקירה מבוססת תפקיד, הסקירה הפרטנית נעשית מנקודות מבט שונות. נקודות מבט אופייניות כוללות משתמשי קצה, שיווק, מעצב, בודק או תפעול. שימוש

בנקודות מבט של בעלי עניין שונים נותן עומק בסקירה הפרטנית, כשמתגלות פחות כפילויות בנושאים שמתגלים על ידי הסוקרים השונים.

בנוסף, קריאה מבוססת נקודת מבט דורשת מהסוקרים לנסות ולהשתמש בתוצר הנסקר כדי להפיק את התוצר שהם ייצרו מהתוצר הנסקר. למשל, בודק ינסה להפיק טיוטה של בדיקות קבלה במידה ומבצע קריאה מבוססת נקודת מבט למפרט דרישות בכדי לראות שכל המידע הנדרש כלול. מעבר לכך, בסקירה מבוססת נקודת מבט יש להשתמש ברשימות בדיקה.

מחקרים אמפיריים הראו שקריאה מבוססת נקודת מבט היא הטכניקה הכללית היעילה ביותר עבור סקירה של דרישות ותוצרים טכניים. גורם הצלחה חשוב הוא לכלול ולשקלל בצורה נכונה נקודות מבט של בעלי עניין שונים, בהתבסס על סיכונים. לפרטים על קריאה מבוססת נקודת מבט ראו Shul 2000 ולפרטים על יעילות של סוגי סקירות שונים ראו Sauer 2000.

3.2.5 *גורמי הצלחה לסקירות*

כדי להצליח בסקירה, יש ליישם את הסוג הנכון של הסקירה ואת הטכניקות הנכונות. בנוסף, ישנם מספר גורמים נוספים שישפיעו על תוצאות הסקירה.

גורמי הצלחה ארגוניים כוללים:

- לכל סקירה יש מטרות ברורות, מוגדרות בשלב תכנון הסקירה, שניתן להשתמש בהן כקריטריון יציאה בר מדידה
- סוגי סקירה המיושמים מתאימים להשגת המטרות ולסוג ורמת תוצר התוכנה והמשתתפים בסקירה
- כל טכניקת סקירה בה משתמשים, למשל מבוססת רשימת בדיקות או מבוססת תפקיד, מתאימה לזיהוי יעיל של פגמים בתוצר הנסקר
- כל רשימות הבדיקה מעודכנות ועונות על הסיכונים העיקריים
- מסמכים גדולים נכתבים ונסקרים בחלקים קטנים, כך שבקרת איכות מיושמת ומספקת למחברים משוב מוקדם ותכוף על פגמים
- יש למשתתפים מספיק זמן כדי להתכונן
- הסקירות נקבעות מספיק זמן מראש
- ההנהלה תומכת בתהליך הסקירה (לדוגמה, על ידי הקצאת זמן מספיק לפעילויות הסקירה בתוכנית הפרויקט)

גורמי הצלחה שתלויים באנשים כוללים:

- האנשים הנכונים להשגת מטרות הסקירה מעורבים, לדוגמה אנשים עם כישורים שונים או נקודות מבט שונות, שיכולים להשתמש במסמך כקלט
- בודקים נחשבים כסוקרים בעלי ערך שתורמים לסקירה ולומדים על התוצר, דבר שמאפשר להם להכין בדיקות יעילות יותר ומוקדם יותר
- המשתתפים מקדישים מספיק זמן ושימת לב לפרטים
- הסקירות נעשות על חלקים קטנים, כדי שהסוקרים לא יאבדו ריכוז במהלך הסקירה הפרטנית ו/או במהלך פגישת הסקירה (אם יש כזו)
- מתקבלות הכרה והערכה לפגמים שהתגלו, והם מטופלים בצורה אובייקטיבית
- הפגישה מנוהלת נכון, כך שהמשתתפים יחשיבו אותה כניצול יעיל של זמנם

- הסקירה מנוהלת באווירה של אמון; תוצאות הסקירה לא ישמשו להערכה של המשתתפים
- על המשתתפים להימנע משפת גוף והתנהגות שעשויה להראות על שעמום, רוגז או עוינות כלפי משתתפים אחרים
- ניתנת הדרכה מתאימה, בייחוד עבור סוגי סקירות רשמיות יותר, כמו ביקורת
- הארגון מעודד תרבות של למידה ושיפור תהליכים

לפרטים נוספים על סקירות מוצלחות ראו Gilb 1993, Wiegers 2002 ו-Van Veenendaal 2004.

4 טכניקות בדיקה — 330 דקות

מושגים

טכניקת בדיקה מסוג קופסה שחורה (black-box test technique), ניתוח ערכי גבול (boundary value analysis), בדיקות מבוססות רשימות ביקורת (checklist-based testing), כיסוי (coverage), כיסוי החלטות (decision coverage), בדיקות טבלת החלטה (decision table testing), ניחוש שגיאות (error guessing), מחלקות שקילות (equivalence partitioning), טכניקת בדיקה מבוססת ניסיון (experience-based test technique), בדיקות חוקרות (exploratory testing), בדיקות הֶחְלֵף מצבים (state transition testing), כיסוי משפטים (statement coverage), טכניקת בדיקה (test technique), בדיקות מקרי שימוש (use case testing), טכניקת בדיקה מסוג קופסה לבנה (white-box test technique)

יעדי הלימוד של פרק טכניקות בדיקה

4.1 קטגוריות של טכניקות בדיקה

FL-4.1.1 (K2) הסבר את המאפיינים, המשותף והשונה בין טכניקות בדיקה מסוג קופסה שחורה, טכניקות בדיקה מסוג קופסה לבנה וטכניקות בדיקה מבוססות ניסיון

4.2 טכניקות בדיקה קופסה שחורה

FL-4.2.1 (K3) גזור מקרי בדיקה מדרישות נתונות באמצעות יישום של חלוקה למחלקות שקילות

FL-4.2.2 (K3) גזור מקרי בדיקה מדרישות נתונות באמצעות יישום של ניתוח ערכי גבול

FL-4.2.3 (K3) גזור מקרי בדיקה מדרישות נתונות באמצעות יישום של בדיקות טבלת החלטה

FL-4.2.4 (K3) גזור מקרי בדיקה מדרישות נתונות באמצעות יישום של בדיקות הֶחְלֵף מצבים

FL-4.2.5 (K2) הסבר כיצד להפיק מקרי בדיקה ממקרי שימוש

4.3 טכניקות בדיקה קופסה לבנה

FL-4.3.1 (K2) הסבר מהו כיסוי משפטים

FL-4.3.2 (K2) הסבר מהו כיסוי החלטות

FL-4.3.3 (K2) הסבר את הערך ביישום כיסוי משפטים וכיסוי החלטות

4.4 טכניקות בדיקה מבוססות ניסיון

FL-4.4.1 (K2) הסבר מהו ניחוש שגיאות

FL-4.4.2 (K2) הסבר מהן בדיקות חוקרות

FL-4.4.3 (K2) הסבר מהן בדיקות מבוססות על רשימות ביקורת

4.1 קטגוריות של טכניקות בדיקה

מטרת טכניקת הבדיקה, כולל אלה שנידונות בפרק זה, היא לסייע בזיהוי מצבי בדיקה, מקרי בדיקה ונתוני בדיקה.

4.1.1 בחירה של טכניקת בדיקה

בחירת טכניקת הבדיקה לשימוש תלויה במספר גורמים, כולל הגורמים הבאים:

- סוג הרכיב או המערכת
- מורכבות הרכיב או המערכת
- תקנים רגולטורים
- דרישות לקוח או דרישות חוזיות
- רמות סיכון
- סוגי סיכון
- מטרות הבדיקה
- תיעוד זמין
- ידע וכישורי הבודק
- כלים זמינים
- זמן ותקציב
- מודל מחזור חיי התוכנה
- השימוש הצפוי בתוכנה
- ניסיון קודם בשימוש בטכניקות הבדיקה על הרכיב או המערכת הנבדקים
- סוגי הפגמים הצפויים ברכיב או במערכת

טכניקות מסוימות מתאימות יותר למצבים מסוימים ורמות בדיקה מסוימות; טכניקות אחרות מתאימות לכל רמות הבדיקה. ביצירת מקרי בדיקה, מבצעים הבודקים בדרך כלל שילוב של טכניקות בדיקה על מנת להשיג את התוצאות הטובות ביותר ממאמץ הבדיקות.

השימוש בטכניקות בדיקה בניתוח הבדיקות, עיצוב הבדיקות ויישום הבדיקות יכול לנוע בין שימוש לא פורמלי בעליל (כאשר חסר תיעוד) לבין שימוש רשמי ביותר. רמת הרשמיות המתאימה תלויה בהקשר של הבדיקות, כולל בגרות תהליכי הבדיקה והפיתוח, אילוצי זמן, דרישות בטיחות ודרישות רגולציה, הידע והכישורים של האנשים המעורבים, ומודל מחזור חיי התוכנה שבשימוש.

4.1.2 *קטגוריות של טכניקות בדיקה והמאפיינים שלהן*

בתוכנית לימודים זו, טכניקות בדיקה מסווגות כקופסה שחורה, קופסה לבנה או מבוססות ניסיון.

טכניקות בדיקה מסוג קופסה שחורה (נקראות גם התנהגותיות או טכניקות מבוססות התנהגות) מבוססות על ניתוח של בסיס בדיקות מתאים (לדוגמה, מסמכי דרישות פורמליות, מפרטים, מקרי שימוש, סיפורי משתמש או תהליכים עסקיים). טכניקות אלה ישימות גם לבדיקות פונקציונליות וגם לבדיקות לא פונקציונליות. טכניקות בדיקה מסוג קופסה שחורה מתמקדות בקלט ובפלט של מושא הבדיקות, מבלי להתייחס למבנה הפנימי שלו.

טכניקות בדיקה מסוג קופסה לבנה (נקראות גם מבניות או טכניקות מבוססות מבנה) מבוססות על ניתוח של הארכיטקטורה, עיצוב מפורט, המבנה הפנימי או הקוד של מושא הבדיקות. בניגוד לטכניקות בדיקה מסוג קופסה שחורה, טכניקות בדיקה מסוג קופסה לבנה מתמקדות במבנה ובתהליכים שבתוך מושא הבדיקות.

טכניקות בדיקה מבוססות ניסיון משתמשים בניסיון של המפתחים, הבודקים והמשתמשים לעיצוב, יישום וביצוע בדיקות. טכניקות אלה משולבות לעיתים עם טכניקות מסוג קופסה שחורה וטכניקות מסוג קופסה לבנה.

מאפיינים נפוצים של טכניקות בדיקה מסוג קופסה שחורה:

- מצבי בדיקה, מקרי בדיקה ונתוני בדיקה נגזרים מבסיס בדיקות שיכול לכלול דרישות תוכנה, מפרטים, מקרי שימוש וסיפורי משתמש
- ניתן להשתמש במקרי בדיקה לזהות פערים בין הדרישות והיישום של אותן דרישות, כמו גם סטייה מהדרישות
- מדידת הכיסוי מבוססת על הפריטים מבסיס הבדיקות שנבדקו ועל הטכניקות שיושמו על בסיס הבדיקות

מאפיינים נפוצים של טכניקות בדיקה מסוג קופסה לבנה:

- מצבי בדיקה, מקרי בדיקה ונתוני בדיקה נגזרים מבסיס בדיקות שיכול לכלול קוד, ארכיטקטורה של התוכנה, עיצוב מפורט, או כל מקור מידע אחר שמתייחס למבנה של התוכנה
- מדידת הכיסוי מבוססת על הפריטים מהמבנה הנבחר שנבדקו (לדוגמה, הקוד או הממשקים)
- מפרטים משמשים לעיתים כמקור מידע נוסף שעוזר להחליט מהן התוצאות הצפויות של מקר הבדיקה

מאפיינים נפוצים של טכניקות בדיקה מבוססות ניסיון:

- מצבי בדיקה, מקרי בדיקה ונתוני בדיקה נגזרים מבסיס בדיקות שיכול לכלול את הידע והניסיון של בודקים, מפתחים, משתמשים ובעלי עניין אחרים

ידע וניסיון אלה כוללים שימוש מצופה בתוכנה, הסיבה בה התוכנה פועלת, פגמים אפשריים והתפוצה של פגמים אלה.

התקן הבינלאומי ISO/IEC/IEEE 29119-4 מכיל תיאור של טכניקות בדיקה ושל מדדי הכיסוי המתאימים להן (למידע נוסף על טכניקות ראו Craig 2012 ו-Copeland 2004).

4.2 טכניקות בדיקה מסוג קופסה שחורה

4.2.1 מחלקות שקילות

טכניקת מחלקות שקילות מחלקת נתונים לקבוצות (או מחלקות שקילות) בצורה כזו שנצפה שכל האברים בקבוצה נתונה יעובדו באותה דרך (ראו Kaner 2013 ו-Jorgensen 2014). ישנן מחלקות שקילות עבור ערכים תקפים וערכים לא תקפים.

- ערכים תקפים הם ערכים שהרכיב או המערכת אמורים לקבל. מחלקת שקילות שמכילה ערכים תקפים נקראת "מחלקת שקילות תקפה".
- ערכים לא תקפים הם ערכים שהרכיב או המערכת אמורים לדחות. מחלקת שקילות שמכילה ערכים לא תקפים נקראת "מחלקת שקילות לא תקפה".
- ניתן לזהות מחלקות לכל רכיב נתונים שקשור למושא הבדיקות, כולל קלט, פלט, ערכים פנימיים, ערכים תלויי זמן, (לדוגמה, לפני אירוע או אחריו) ולנתונים של ממשקים (לדוגמה, רכיבים משולבים שנבדקים בזמן בדיקות אינטגרציה).
- במידת הצורך, ניתן לחלק כל מחלקה לתת מחלקות.
- כל ערך נתון יכול להשתייך למחלקת שקילות אחת ורק אחת.
- כאשר משתמשים במחלקות שקילות לא תקפות במקרה בדיקה, יש לבדוק מחלקות שקילות אלה בנפרד, כלומר, לא בשילוב עם מחלקות שקילות לא תקפות אחרות, וזאת כדי לוודא שכשלים לא ממוסכים. כשלים יכולים להיות ממוסכים כאשר מספר כשלים מתרחשים בו זמנית אבל רק אחד מהם גלוי ומזוהה, בעוד שהאחרים לא מזוהים.

כדי להשיג 100% כיסוי עם טכניקה זו, מקרי הבדיקה צריכים לכסות את כל המחלקות המזוהות (כולל מחלקות לא תקפות). כיסוי נעשה על ידי שימוש בערך אחד לפחות מתוך כל מחלקת שקילות. הכיסוי נמדד על ידי חלוקה של מספר מחלקות השקילות שנבדקו על ידי ערך אחד לפחות, בסך מחלקות השקילות שזיהינו. בדרך כלל מבטאים כיסוי זה באחוזים. ניתן ליישם חלוקה למחלקות שקילות בכל רמות הבדיקה.

4.2.2 ניתוח ערכי גבול

טכניקת ניתוח ערכי גבול הינה הרחבה של חלוקה למחלקות שקילות. ניתן להשתמש בטכניקה זו כאשר המחלקות מכילות ערכים מספריים או רציפים. ערכי המינימום והמקסימום (או הערך הראשון והערך האחרון) של מחלקה הינם ערכי הגבול (Beizer 1990).

לדוגמה, נניח ששדה קלט מקבלם מספר שלם (integer) חד ספרתי כקלט, כאשר משתמשים בלוח מקשים כדי להגביל ולמנוע הכנסה של קלט שאינו ערך מספרי. התחום התקף הוא בין 1 ל-5, כולל. כך נוצרים שלוש מחלקות שקילות: לא תקף (נמוך מידי); תקף; לא תקף (גבוה מידי). ערכי הגבול למחלקת השקילות התקפה הינם 1 ו-5. ערכי הגבול למחלקת השקילות הלא תקפה (גבוה מידי) הינם 6 ו-9. למחלקת השקילות הלא תקפה (נמוך מידי) יש רק ערך גבול אחד, 0, מאחר וזו מחלקה שיש בה רק אבר אחד.

בדוגמה למעלה, זיהינו שני ערכי גבול לכל גבול. הגבול בין המחלקה התקפה למחלקה הלא תקפה (נמוך מידי) נותן את ערכי הבדיקה 0 ו-1. הגבול בין המחלקה התקפה למחלקה הלא תקפה (גבוה מידי) נותן את ערכי הבדיקה 5 ו-6. גרסאות אחדות של הטכניקה הזו מזהות שלושה ערכי גבול לכל גבול: הערך שלפני הגבול, ערך הגבול וערך שאחרי הגבול. בדוגמה הקודמת, שימוש בשלושה ערכי גבול ייתן את ערכי הבדיקה 0, 1, ו-2 לגבול התחתון ואת ערכי הבדיקה 4, 5 ו-6 לגבול העליון (Jorgensen 2014).

הסיכוי גבוה יותר שהתנהגות המערכת תהיה לא נכונה בגבולות מחלקות השקילות מאשר בתוכן. חשוב לזכור שהגבולות שהוגדרו ויושמו עשויים לזוז למקום גבוה או נמוך יותר מאשר הגבול שהתכוונו אליו, יתכן והגבולות לא יוגדרו בכלל, או שייתכן ויהיו גבולות נוספים לא רצויים. בדיקות באמצעות ניתוח ערכי גבול יחשפו כמעט את כל הפגמים האלה בכך שיחשפו גבולות שלא מתנהגים כמו מחלקת השקילות אליה הם אמורים להשתייך.

ניתן ליישם ניתוח ערכי גבול בכל רמות הבדיקה. בדרך כלל משתמש הטכניקה הזו לבדיקה של דרישות שקוראות לטווח של מספרים (כולל תאריכים וזמנים). כיסוי גבולות של מחלקת שקילות נמדד כחלוקה של מספר ערכי הגבול שנבדקו בסך כל ערכי הגבול שזוהו. בדרך כלל מבטאים כיסוי זה באחוזים.

4.2.3 *בדיקות טבלת החלטה*

טכניקות בדיקה של צירופים הינן יעילות לבדיקה של יישום דרישות מערכת שמגדירות איך צירופים של תנאים שונים נותנים תוצאות שונות. גישה אחת לבדיקות כאלה נקראה בדיקות טבלת החלטה.

טבלאות החלטה הינן דרך טובה לתעד חוקים עסקיים מורכבים שעל המערכת ליישם. על בודק שיוצר טבלת החלטה לזהות את התנאים (לעיתים אלה נתוני הקלט) ואת הפעולות שנגזרות מהתנאים (לעיתים אלה התוצאות) של המערכת. אלה יוצרים את השורות בטבלה, בדרך כלל כשהתנאים בחלק העליון והפעולות החלק התחתון. כל טור מתייחס לחוק עסקי שמגדיר צירוף ייחודי של תנאים שגורם לביצוע של הפעולות שקשורות לחוק הזה. הערכים של התנאים והפעולות מוצגים בדרך כלל כערכים בוליאניים (True/False) או כערכים נפרדים (לדוגמה, אדום, ירוק, כחול), אבל יכולים להיות גם מספרים או טווח של מספרים. ניתן למצוא סוגים שונים אלה של תנאים ופעולות באותה טבלה.

הסימון המקובל של טבלאות החלטה הוא כלהלן:

עבור התנאים:

- כ (Y) אומר שהתנאי נכון (ניתן להציג גם כ-T או 1)
- ל (N) אומר שהתנאי אינו נכון (ניתן להציג גם כ-F או 0)
- – אומר שהערך של התנאי אינו רלוונטי (ניתן להציג גם כ-NA)

עבור הפעולות:

- X אומר שהפעולה צריכה להתבצע (ניתן להציג גם כ-Y או T או 1)
- תא ריק אומר שהפעולה לא צריכה להתבצע (ניתן להציג גם כסימן – או N או F או 0)

לטבלת החלטה מלאה יהיו מספיק טורים שיכסו כל צירוף אפשרי של תנאים. ניתן לצמצם את הטבלה על ידי ביטול של טורים שמכילים צירופי תנאים לא אפשריים, טורים שמכילים צירופי תנאים אפשריים אבל לא ישימים וטורים שבודקים צירופי תנאים שלא משפיעים על התוצאה. למידע נוסף על צמצום של טבלאות החלטה, ראו ISTQB-ATA Advanced Level Test Analyst.

הכיסוי המינימלי המקובל בבדיקות טבלת החלטה מתקבל על ידי יצירה של מקרה בדיקה אחד לפחות לכל מצב החלטה בטבלה. זה אומר בדרך כלל כיסוי כל צירופי התנאים. הכיסוי נמדד כחלוקה של מספר מצבי ההחלטה שנבדקים על ידי מקרה בדיקה אחד לפחות בסך כל מצבי ההחלטה. בדרך כלל מבטאים כיסוי זה באחוזים.

החוזק של טבלת החלטה הוא בכך שהיא עוזרת לזהות את כל צירופי התנאים החשובים, חלקם לא היו מזוהים בדרך אחרת. הטבלה מסייעת גם בזיהוי פערים בדרישות. ניתן ליישם טכניקה זו לכל המצבים בהם ההתנהגות של התוכנה תלויה בצירוף של תנאים ובכל רמות הבדיקה.

4.2.4 ***בדיקות הֶחְלֵף מצבים***

רכיבים או מערכות שעשויים להגיב לאירוע בצורה שונה שתלויה במצב הנוכחי או בהיסטוריה של המצבים (לדוגמה, האירועים שקרו מאז שהמערכת אותחלה). ניתן לסכם את ההיסטוריה בעזרת המושג "מצבים". תרשים הֶחְלֵף מצבים מציג את מצבי התוכנה האפשריים, כמו גם את הדרך בה התוכנה נכנסת למצבים, יוצאת ממצבים ועוברת בין מצבים. מעבר בין מצבים (הֶחְלֵף) נגרם על ידי אירוע (לדוגמה, ערך שמכניס משתמש לשדה). במצבים בהם אותו אירוע יכול לגרום לשני מעברים או יותר מאותו מצב, יש לציין יחד עם האירוע את התנאי או התנאים הנוספים שקובעים איזה מהמעברים יתבצע. תנאים אלה נקראים "תנאי הגנה" (guard conditions). השינוי במצב עשוי לגרום לתוכנה לבצע פעולה (לדוגמה, להציג תוצאה של חישוב או הודעת שגיאה).

טבלת הֶחְלֵף מצבים מציגה את כל מעברי המצבים התקפים ואת האירועים, תנאי ההגנה והפעולות שיבוצעו כתוצאה ממעבר בין מצבים תקפים. מצבים לא תקפים מיוצגים בטבלה ומסומנים בהתאם. תרשים הֶחְלֵף מצבים מציג בדרך כלל את מעברי המצבים התקפים ולא את מעברי המצבים הלא תקפים.

ניתן לתכנן בדיקות שיכסו רצף אופייני של מצבים, להפעיל את כל המצבים, להפעיל את כל מעברי המצבים, להפעיל סדרות מוגדרות של מעברי מצבים, או לנסות לגרום למעברי מצבים לא תקפים.

בדיקות הֶחְלֵף מצבים משמשות לאפליקציות מבוססות תפריט והן נפוצות מאוד בתעשיית התוכנה משובצת המחשב (embedded software). טכניקה זו מתאימה גם למודלים של תרחישים עסקיים שלהם מצבים מוגדרים או לבדיקות של ניווט בין מסכים. המושג "מצב" הוא מושג מופשט והוא יכול לייצג מספר שורות קוד או תהליך עסקי שלם.

כיסוי מקובל של בדיקות הֶחְלֵף מצבים נמדד על ידי חלוקה של מספר המצבים או מעברי המצבים שנבדקו בסך כל המצבים או מעברי המצבים שזוהו במושא הבדיקות. בדרך כלל מבטאים כיסוי זה באחוזים. למידע נוסף על קריטריון כיסוי לבדיקות הֶחְלֵף מצבים, ראו ISTQB-ATA Advanced Level Test Analyst.

4.2.5 ***בדיקות מקרי שימוש***

ניתן לגזור בדיקות ממקרי שימוש, שהן דרך לעיצוב יחסי גומלין (interactions) עם רכיבי תוכנה, ולשלב דרישות לתפקוד התוכנה המיוצגות על ידי מקרי שימוש. מקרי שימוש קשורים לשחקנים (actors) (משתמשים אנושיים, חומרה חיצונית, או רכיבים אחרים במערכת) ומושאים (subjects) (הרכיב או המערכת אליו מתייחס מקרה השימוש).

כל מקרה שימוש מגדיר התנהגות מסוימת שמושא הבדיקה יכול לבצע בשיתוף שחקן אחד או יותר (UML 2.5.1 2017). ניתן לתאר מקרה שימוש על ידי יחסי הגומלין והפעילויות שלו, כמו גם התנאים המקדימים, מצבי הסיום (post-condition) ובשפה טבעית איפה שניתן. יחסי גומלין בין שחקנים ומושא עשויים לגרום לשינויים במצב המושא. ניתן להציג את יחסי הגומלין בצורה גרפית באמצעות תרשימי זרימה, תרשימי פעילות, או של תהליכים עסקיים.

מקרה שימוש יכול להכיל חלופות אפשריות שונות של התנהגות המקרה הבסיסית, כולל התנהגות יוצאת דופן וניהול שגיאות (תגובת המערכת והתאוששות משגיאות תכנות, יישום ותקשורת וכו' שיגרמו להופעת הודעת שגיאה). הבדיקות מעוצבות להפעלת ההתנהגות המוגדרת (התנהגות בסיסית, התנהגות יוצאת דופן או חלופית וניהול שגיאות). הכיסוי נמדד על ידי חלוקה של מספר התנהגויות מקרי השימוש שנבדקו בסך כל התנהגויות מקרי השימוש. בדרך כלל מבטאים כיסוי זה באחוזים.

למידע נוסף על קריטריון כיסוי לבדיקות מקרי שימוש, ראו ISTQB-ATA Advanced Level Test Analyst.

4.3 טכניקות בדיקה מסוג קופסה לבנה

בדיקות מסוג קופסה לבנה מבוססות על המבנה הפנימי של מושא הבדיקות. ניתן להשתמש בטכניקות בדיקה מסוג קופסה לבנה בכל רמות הבדיקה, אולם שתי הטכניקות שיוצגו בפרק זה מבוססות קוד והשימוש בהן נפוץ בעיקר ברמת בדיקות הרכיבים. ישנן טכניקות בדיקה מתקדמות יותר שנמצאות בשימוש במערכות קריטיות (safety critical or mission critical) או בסביבות שדורשות אמינות גבוהה (high integrity environment), במטרה להשיג כיסוי יסודי יותר. למידע נוסף על טכניקות אלה, ראו ISTQB-ATTA Advanced Level Technical Test Analyst.

4.3.1 בדיקות וכיסוי משפטים

בדיקת משפטים מפעילה את משפטי הפקודה בקוד (לא כולל, למשל, הערות בקוד). הכיסוי נמדד על ידי חלוקה של המשפטים שהופעלו על ידי הבדיקה בסך משפטי הפקודה במושא הבדיקות. בדרך כלל מבטאים כיסוי זה באחוזים.

4.3.2 בדיקות וכיסוי החלטות

בבדיקת החלטות מריצים מקרי בדיקה שגורמים לכל נקודת החלטה בקוד (לדוגמה משפט IF) לקבל, לפחות פעם אחת ערך "אמת" (true) ולפחות פעם אחת ערך "שקר" (false). מקרי הבדיקה מונחים על ידי בקרת הזרימה (control flow) שנובעת מתוצאות ההחלטות (לדוגמה, למשפט IF, מקרה בדיקה אחד עבור תוצאת "אמת" (true) ומקרה בדיקה אחד עבור תוצאת "שקר" (false); למשפט CASE יידרשו מקרי בדיקה לכל התוצאות האפשריות, כולל תוצאת ברירת המחדל).

הכיסוי נמדד על ידי חלוקה של מספר תוצאות ההחלטה שהופעלו על ידי הבדיקות בסך כל תוצאות ההחלטה במושא הבדיקות. בדרך כל מבטאים כיסוי זה באחוזים.

4.3.3 הערך של בדיקות משפטים ובדיקות החלטות

100% כיסוי משפטים מבטיח שכל משפטי ההפעלה בקוד נבדקו לפחות פעם אחת, אבל לא מבטיח שכל לוגיקת ההחלטות נבדקה. מבין שתי טכניקות הבדיקה מסוג קופסה לבנה שמוצגות כאן, בדיקות משפטים מספקות כיסוי נמוך יותר מבדיקות החלטות.

100% כיסוי החלטות מבטיח שכל תוצאות ההחלטות הופעלו, כולל התוצאות החיוביות והתוצאות השליליות, גם כאשר אין משפטים מפורשים לתוצאה שלילית (לדוגמה, כאשר בקוד ישנו משפט IF ללא משפט ELSE). כיסוי משפטים עוזר למצוא פגמים בקוד שלא היה מופעל על ידי בדיקות אחרות. כיסוי החלטות עוזר למצוא פגמים בקוד במקומות בהם בדיקות אחרות לא בדקו גם את התוצאות החיוביות וגם את התוצאות השליליות.

100% כיסוי החלטות מבטיח 100% כיסוי משפטים (אך לא להיפך).

4.4 טכניקות בדיקה מבוססות ניסיון

כאשר מיישמים טכניקות בדיקה מבוססות ניסיון, מקרי הבדיקה נגזרים מהיכולות של הבודקים, והאינטואיציה שלהם ומהניסיון שלהם עם יישומים וטכנולוגיות דומים. טכניקות אלה יכולות לסייע בזיהוי בדיקות שלא ניתן לזהות בעזרת טכניקות אחרות, שיטתיות יותר. שיטות אלה ישיגו רמה שונה של כיסוי ושל יעילות, בהתאם לגישה ולניסיון של הבודק. כאשר משתמשים בטכניקות אלה קשה להעריך ולא ניתן למדוד את הכיסוי.

הפרקים הבאים מציגים טכניקות מבוססות ניסיון מקובלות:

4.4.1 ניחוש שגיאות

טכניקת ניחוש שגיאות משתמשת בידע של הבודק לחיזוי הופעת שגיאות, פגמים וכשלים. הידע הזה כולל:

- איך עבד היישום בעבר?
- איזה סוג של שגיאות נוטים המפתחים לעשות?
- כשלים שקרו ביישומים אחרים

ניתן לגשת לטכניקת ניחוש שגיאות בצורה שיטתית וליצור רשימה של שגיאות, פגמים וכשלים אפשריים, ולעצב בדיקות שיחשפו את הכשלים האלה ואת הפגמים שגורמים להם. ניתן לבנות רשימות שגיאות, פגמים וכשלים אלה על בסיס הניסיון של הבודק, נתוני על פגמים וכשלים, או על סמך ידע כללי על סיבות שכיחות לכשלים של תוכנה.

4.4.2 *בדיקות חוקרות*

בבדיקות חוקרות, בדיקות לא פורמליות (לא מתוכננות מראש) מעוצבות, מבוצעות, מתועדות ומוערכות בצורה דינמית תוך כדי ביצוע בדיקות. תוצאות הבדיקה משמשות ללימוד נוסף של הרכיב או המערכת, וליצירה של בדיקות עבור אזורים במערכת שנזקקים לבדיקות נוספות.

בדיקות חוקרות נערכות לעיתים תוך שימוש בבדיקות מבוססות מושב (session based testing) כדי לבנות את פעילות הבדיקה. בבדיקות מבוססות מושב, בדיקות חוקרות נערכות במסגרת של זמן מוגדר, והבודק משתמש באמנת בדיקות (test charter) שמכילות את מטרות הבדיקה ומשמשות כקו מנחה לבדיקות. ניתן להשתמש באמנות הבדיקה לתיעוד צעדי הבדיקה ואת הממצאים שהתגלו במהלך הבדיקה.

בדיקות חוקרות מועילות מאוד במקרים בהם יש מפרטים דלים או לא מספקים, או כאשר הבדיקות מוגבלות מאוד בזמן. בדיקות חוקרות מועילות גם כהשלמה לטכניקות בדיקה אחרות, פורמליות יותר.

בדיקות חוקרות קשורות לאסטרטגיות בדיקה תגובתיות (reactive test strategy) (ראו פרק 5.2.2). בדיקות חוקרות עשויות לשלב טכניקות קופסה שחורה, טכניקות קופסה לבנה וטכניקות מבוססות ניסיון, לעיתים באותו מקרה בדיקה.

4.4.3 *בדיקות מבוססות על רשימות בדיקה*

בבדיקות מבוססת על רשימות בדיקה, הבודקים מעצבים, מיישמים ומריצים בדיקות במטרה לכסות תנאי בדיקה שנמצאים ברשימות בדיקה. כחלק מניתוח הבדיקות, בודקים יוצרים רשימת בדיקה חדשה או מעדכנים ומרחיבים רשימה קיימת. ניתן גם להשתמש ברשימה קיימת מבלי לשנות אותה. ניתן לבנות רשימות בדיקה כאלה על סמך ניסיון, ידע על מה שחשוב למשתמש, או הבנה של מה גורם לכשלים בתוכנה ואיך כשלים כאלה נגרמים.

ניתן ליצור רשימות בדיקה כדי לתמוך בסוגי בדיקות שונים, כולל בדיקות פונקציונליות ובדיקות לא פונקציונליות. בהיעדר מקרי בדיקה מפורטים, בדיקות מבוססות על רשימות בדיקה יכולות לספק קו מנחה ורמה מסוימת של עקביות. מאחר ורשימות בדיקה אלה לא מפורטות לעומק, יתכן ויהיה קיים שוני בדרך בה יבוצעו הבדיקות בפועל, דבר שעשוי לתרום לכיסוי גבוה יותר אך להקטין את האפשרות לשחזר במדויק את צעדי הבדיקה.

5 ניהול בדיקות — 225 דקות

מושגים

ניהול תצורה (configuration management), ניהול פגמים (defect management), קריטריון כניסה (entry criteria), קריטריון יציאה (exit criteria), סיכון מוצר (product risk), סיכון פרויקט (project risk), סיכון (risk), רמת סיכון (risk level), בדיקות מבוססות סיכונים (risk-based testing), גישה לבדיקות (test approach), בקרת בדיקות (test control), אומדן בדיקות (test estimation), מנהל בדיקות (test manager), ניטור בדיקות (test monitoring), תוכנית בדיקות (test plan), תכנון בדיקות (test planning), דו"ח התקדמות בדיקות (test progress report), אסטרטגית בדיקות (test strategy), דו"ח סיכום בדיקות (test summary report), בודק (tester)

יעדי הלימוד של פרק ניהול בדיקות

5.1 ארגון הבדיקות

FL-5.1.1 (K2) הסבר את היתרונות והחסרונות של בדיקות עצמאיות

FL-5.1.2 (K1) זהה את המשימות של מנהל הבדיקות ושל הבודק

5.2 תכנון הבדיקות והערכת הבדיקות

FL-5.2.1 (K2) סכם את המטרה והתוכן של תוכנית הבדיקות

FL-5.2.2 (K2) הבחן בין אסטרטגיות בדיקות שונות

FL-5.2.3 (K2) הבא דוגמאות לקריטריון כניסה וקריטריון יציאה אפשריים

FL-5.2.4 (K3) יישם ידע על תיעדוף ותלויות טכניות ולוגיות על מנת לקבוע לוח זמנים לביצוע בדיקות עבור סדרה נתונה של מקרי בדיקה.

FL-5.2.5 (K1) זהה גורמים שמשפיעים על המאמץ הקשור בבדיקות.

FL-5.2.6 (K2) הסבר את ההבדלים בין שתי טכניקות לאומדן: טכניקה מבוססת מדדים וטכניקה מבוססת מומחים.

5.3 ניטור בדיקות ובקרת בדיקות

FL-5.3.1 (K1) הכר מדדים המשמשים בבדיקות

FL-5.3.2 (K2) סכם את המטרה, התוכן וקהל היעד של דוחות בדיקות

5.4 ניהול תצורה

FL-5.4.1 (K2) סכם איך מסייע ניהול תצורה לבדיקות

5.5 סיכונים ובדיקות

FL-5.5.1 (K1) הגדר רמת סיכון בעזרת סבירות והשפעה

FL-5.5.2 (K2) הבחן בין סיכוני מוצר וסיכוני פרויקט

FL-5.5.3 (K2) הסבר בעזרת דוגמאות איך ניתוח סיכוני מוצר עשוי להשפיע על היקף ויסודיות הבדיקות

5.6 ניהול פגמים

FL-5.6.1 (K3) כתוב דו"ח פגמים שמכסה את הפגמים שנמצאו במהלך הבדיקות

5.1 ארגון הבדיקות

5.1.1 בדיקות עצמאיות

משימות הבדיקות עשויות להיעשות על ידי אנשים בתפקידי בדיקות ייעודיים, או על ידי אנשים בתפקידים אחרים (לדוגמה, לקוחות). לעיתים, רמה מסוימת של עצמאות תעשה את הבודק יעיל יותר במציאות פגמים בשל ההבדל בהטיה הקוגניטיבית בין המחבר והבודק (ראו פרק 1.5). אולם עצמאות אינה תחליף להיכרות עם המערכת, ומפתחים יכולים להיות יעילים במציאת פגמים בקוד שלהם עצמם.

עצמאות הבדיקות תוגדר לפי הרמות הבאות (מרמת העצמאות הנמוכה אל הרמה הגבוהה):

- אין בודקים עצמאיים; הבדיקות היחידות שנעשות הן אלה שבהן המפתחים בודקים את הקוד של עצמם
- מפתחים עצמאיים או בודקים עצמאיים בתוך צוות הבדיקות או צוות הפרויקט; יתכן ויהיו אלה מפתחים שבודקים את התוצרים על עמיתיהם
- צוות בדיקות או קבוצת בדיקות עצמאיים בתוך הארגון, מדווחים להנהלת הפרויקט או להנהלה הבכירה
- צוות בדיקות עצמאי מתוך הארגון או מקהילת המשתמשים, או עם התמחות בסוג מסוים של בדיקות כדוגמה שימושיות, ביצועים, תאימות לתקינה, או ניידות תוכנה
- בודקים עצמאיים חיצוניים לארגון, כאלה שעובדים באתר הארגון (onsite, insourcing) או באתר חיצוני (מיקור חוץ, outsourcing)

במרבית סוגי הפרויקטים, עדיף בדרך כלל ליישם מספר רמות בדיקות, כאשר חלק מרמות בדיקה אלה מבוצעות על ידי בודקים עצמאיים. מפתחים צריכים להשתתף בבדיקות, בייחוד ברמות הנמוכות, כדי שתהיה להם שליטה על האיכות של התוצר שלהם.

הדרך בה מיושמת העצמאות של הבדיקות משתנה בהתאם למחזור חיי התוכנה. לדוגמה, בפיתוח אג'ילי, הבודקים עשויים להיות חלק מצוות הפיתוח. בחלק מהארגונים שמשתמשים בפיתוח אג'ילי, בודקים אלה יהיו גם חלק מצוות בדיקות עצמאי. בנוסף, מנהלי מוצר בארגונים כאלה עשויים לבצע בדיקות קבלה כדי לתקף סיפור משתמש בסיום כל מחזור.

יתרונות אפשריים לבדיקות עצמאיות כוללים:

- בודקים עצמאיים עשויים לזהות סוגי כשלים שונים בהשוואה למפתחים מאחר ויש להם רקע שונה, נקודת מבט טכנית שונה, והטיות שונות
- בודק עצמאי יכול לוודא, לאתגר או להפריך הנחות שנעשו על ידי בעלי עניין במהלך כתיבת המפרטים ויישום המערכת

חסרונות אפשריים לבדיקות עצמאיות כוללים:

- היבדלות מצוות הפיתוח שיכול להביא לחוסר שיתוף פעולה, עיכוב במתן משוב לצוות הפיתוח או יחסים עכורים עם צוות הפיתוח
- המפתחים עשויים לאבד תחושת אחריות לאיכות
- בודקים עצמאיים עשויים להיתפס כצוואר בקבוק ולהיות מואשמים בעיכוב שחרור המערכת
- מידע חשוב עשוי להיות חסר לבודקים עצמאיים (לדוגמה, על מושא הבדיקות)

ארגונים רבים מצליחים להשיג בהצלחה את היתרונות של בדיקות עצמאיות ולהימנע מהחסרונות.

5.1.2 *מטלות של מנהל בדיקות ושל בודק*

תוכנית לימודים זו מציגה שני תפקידי בדיקות, מנהל בדיקות ובודק. הפעילויות והמטלות שמבוצעות בידי שני תפקידים אלה תלויות בהקשר של הפרויקט ושל המוצר, בכישורים של האנשים שנמצאים בתפקידים אלה ובארגון.

למנהל בדיקות יש אחריות כוללת לתהליך הבדיקות והובלה מוצלחת של פעילויות הבדיקה. את תפקיד מנהל הבדיקות יכול לבצע מנהל בדיקות מקצועי או מנהל פרויקט, מנהל פיתוח או מנהל אבטחת איכות. בפרויקטים גדולים או בארגונים גדולים, מספר צוותי בדיקות יכולים לדווח למנהל בדיקות, מאמן בדיקות (test coach) או מתאם בדיקות (test coordinator), כשבראש כל צוות עומד מוביל בדיקות או בודק ראשי (lead tester).

מטלות אופייניות של מנהל בדיקות עשויות לכלול:

- פיתוח או סקירה של מדיניות בדיקות (test policy) ואסטרטגיית בדיקות לארגון
- תכנון פעילויות הבדיקה תוך התחשבות בהקשר של הבדיקות והבנת מטרות הבדיקה והסיכונים. מטלה זו תכלול בחירת גישה לבדיקות, אומדן זמני בדיקות, מאמץ ועלות הבדיקות, רכישת משאבים, הגדרת רמות בדיקה וסבבי בדיקה ותכנון ניהול פגמים
- כתיבה ועדכון של תוכני(ו)ת בדיקה
- תיאום תוכני(ו)ת הבדיקה עם מנהלי פרויקט, מנהלי מוצר ואחרים
- שיתוף נקודת המבט של הבדיקות עם פעילויות פרויקט אחרות, כמו תכנון אינטגרציה
- ייזום ניתוח, עיצוב, כתיבה וביצוע של בדיקות, ניטור התקדמות הבדיקות ותוצאות הבדיקה ובדיקת מצב קריטריון היציאה (או ההגדרה של "נקודת הסיום" (done))
- הכנה ושליחה של דוחות התקדמות הבדיקות ודוחות סיכום הבדיקות על פי המידע שנאסף
- התאמת תכניות על פי תוצאות הבדיקה והתקדמות הבדיקות (לעיתים מתועד בדוחות התקדמות הבדיקות ו/או בדוחות סיכום הבדיקות של בדיקות שהושלמו בפרויקט) ונקיטת צעדים הנדרשים לבקרת הבדיקות
- תמיכה בהקמת מערכת ניהול פגמים וניהול תצורה הולם למכלול מרכיבי הבדיקות (בּוֹדְקָה)
- הגדרת מדדים מתאימים למדידת התקדמות הבדיקות והערכת איכות הבדיקות ואיכות המוצר
- תמיכה בבחירה וביישום של כלים שיתמכו בתהליך הבדיקות, כולל המלצה על תקציב לבחירת כלי (כמו גם רכישה ו/או תמיכה), הקצאת זמן ומאמץ לביצוע פרויקט ניסיוני (פיילוט) ותמיכה שוטפת לשימוש בכלי(ם)
- החלטה על יישום של סביב(ו)ת בדיקה
- קידום הבודקים ותמיכה בהם, בצוות הבדיקות ובמקצוע הבדיקות בתוך הארגון
- פיתוח כישורי הבודקים והקריירה שלהם (למשל, באמצעות תכניות הדרכה, הערכת ביצועים, אימון ועוד)

הדרך בה מיושם תפקיד מנהל הבדיקות משתנה בהתאם למחזור חיי התוכנה. לדוגמה, בפיתוח אג'ילי, חלק מהמטלות המוזכרות לעיל מבוצעות על ידי צוות האג'ייל, בייחוד אלו הקשורות למטלות הבדיקה היומיומיות שנעשות בתוך הצוות, לעיתים על ידי בודק שעובד כחלק מהצוות. חלק מהמטלות שקשורות למספר צוותים או לכל הארגון, או כאלה שנוגעות לניהול אישי, ייעשו על ידי מנהלי בדיקות מחוץ לצוות הפיתוח, שלעיתים נקראים מאמני בדיקות. לעוד מידע על ניהול תהליך הבדיקות ראו Black 2009.

מטלות אופייניות של בודק עשויות לכלול:

- סקירה ותרומה לתכניות בדיקה
- ניתוח, סקירה והערכת הבדיקתיות (testability) של דרישות, סיפורי משתמש וקריטריון קבלה, מפרטים ומודלים (כלומר, בסיס הבדיקות)
- זיהוי ותיעוד מצבי בדיקה, והגדרת נֶעֱקָבוּת בין מקרי בדיקה, מצבי בדיקה ובסיס הבדיקות
- עיצוב, הקמה ווידוא של סביב(ו)ת הבדיקות, לעיתים תוך תיאום עם מנהלי מערכות (system administrators) ומנהלי רשת
- עיצוב ויישום של מקרי בדיקה והליכי בדיקה
- הכנה של נתוני בדיקה או השגה של נתונים מוכנים
- יצירת לוח זמנים מפורט לביצוע בדיקות
- ביצוע בדיקות, הערכת התוצאות ותיעוד סטיות מהתוצאות הצפויות
- שימוש בכלים מתאימים לסיוע בתהליך הבדיקות
- אוטומציה של בדיקות לפי הנדרש (יתכן ומטלה זו תיתמך על ידי מפתח או מומחה בדיקות אוטומטיות
- הערכת מאפיינים לא פונקציונליים כמו יעילות ביצועים, מהימנות, שימושיות, אבטחה, תאימות ונייידות תוכנה
- סקירה של בדיקות שמפותחות על ידי אחרים

אנשים שעובדים בניתוח בדיקות, עיצוב בדיקות, סוגי בדיקות מסוימים ואוטומציה של בדיקות עשויים להיות מומחים בתחומים אלה. בהתאם לסיכונים שקשורים למוצר ולפרויקט, כמו גם למודל חיי התוכנה שבשימוש, אנשים שונים עשויים להחזיק בתפקיד הבודק ברמות בדיקה שונות. לדוגמה, ברמת בדיקות הרכיבים וברמת בדיקות אינטגרציה של רכיבים, תפקיד הבודק נעשה במקרים רבים על ידי המפתחים. ברמת בדיקות הקבלה, תפקיד הבודק נעשה במקרים רבים על ידי מנתח עסקי (business analyst), מומחים בתחום (subject matter expert) ומשתמשים. ברמת בדיקות המערכת וברמת בדיקות אינטגרציה של מערכות תפקיד הבודק נעשה בדרך כלל על ידי צוות בדיקות עצמאי. ברמת בדיקות הקבלה התפעוליות, תפקיד הבודק נעשה לעיתים על ידי מנהלי המערכת.

5.2 תכנון בדיקות ואומדן בדיקות

5.2.1 מטרת תוכנית הבדיקות ותוכן תוכנית הבדיקות

תוכנית בדיקות מתארת את פעילויות הבדיקה עבור פרויקטים של פיתוח ותחזוקה. התכנון מושפע ממדיניות הבדיקות ומאסטרטגיית הבדיקות של הארגון, ממחזור חיי הפיתוח ומהשיטות שמונהגות בארגון (ראו פרק 2.1), הגדרת היקף הבדיקות[6], מטרות, סיכונים, מגבלות, קריטיות, בדיקתיות, וזמינות המשאבים.

ככל שמתקדמים תכנון הפרויקט ותכנון הבדיקות, עוד מידע נהיה זמין וניתן לכלול פרטים נוספים בתוכנית הבדיקות. תכנון בדיקות היא פעילות מתמשכת שמבוצעת לאורך כל מחזור חיי המוצר. (שימו לב שמחזור חיי המוצר נמשך גם לאחר סיום הפרויקט וכולל את שלב התחזוקה). ניתן להשתמש במשוב על פעילויות הבדיקה כדי לזהות שינויים בסיכונים וכך להתאים את התכנון. ניתן לתעד את תכנון הבדיקות בתוכנית בדיקה כוללת של הפרויקט (master test plan) ובתכניות בדיקה נפרדות, כמו בדיקות שימושיות או בדיקות ביצועים. פעילויות תכניות בדיקה עשויות לכלול את הפעילויות הבאות, שאת חלקן ניתן לתעד במסמך תוכנית הבדיקות:

- קביעת תכולת הבדיקות, מטרות הבדיקות והסיכונים בבדיקות
- הגדרת הגישה הכללית לבדיקות
- שילוב ותיאום פעילות הבדיקה עם פעילויות מחזור חיי התוכנה
- קבלת החלטות על מה ייבדק, האנשים והמשאבים האחרים הדרושים לביצוע פעילויות הבדיקה השונות ואיך יבוצעו פעילויות אלה
- תזמון ניתוח הבדיקות, עיצוב, יישום, ביצוע והערכת הפעילויות, בין אם בתאריכים מסוימים (לדוגמה, במודלים של פיתוח סדרתי) או בהקשר של כל מחזור (לדוגמה, בפיתוח בסבבים)
- בחירת מדדים לניטור ובקרת בדיקות
- תקצוב פעילויות הבדיקה
- הגדרת המבנה של מסמכי הבדיקות ורמת הפירוט הנדרשת בהם (לדוגמה, על ידי יצירה של מסמכי דוגמה או תבנית (template) של המסמך)

התוכן של תכניות בדיקה משתנה ויכול להתרחב מעבר לנושאים שהוזכרו לעיל. ניתן למצוא תכניות בדיקה לדוגמה בתקן ISO (ISO/IEC/IEEE 29119-3).

5.2.2 אסטרטגית בדיקות וגישה לבדיקות

אסטרטגית בדיקות מספקת תיאור כללי של תהליך הבדיקות, בדרך כלל ברמת המוצר או הארגון. סוגים נפוצים של אסטרטגיות בדיקה כוללים:

- **ניתוחית** (analytical): סוג זה של אסטרטגיית בדיקות מבוסס על ניתוח של מספר גורמים (לדוגמה, דרישות או סיכונים). בדיקות מבוססות סיכונים הן דוגמה לגישה ניתוחית, בה עיצוב הבדיקות ותיעדוף הבדיקות נעשה על בסיס רמת הסיכון.
- **מבוססת מודל** (model based): בסוג זה של אסטרטגיית בדיקות, עיצוב הבדיקות מתבסס על מודל של היבט כלשהו של המוצר, כמו פונקציה, תהליך עסקי, מבנה פנימי או מאפיין לא פונקציונלי (לדוגמה, מהימנות). דוגמאות למודלים כאלה כוללים מודלים של תהליכים עסקיים, מודל מצבים ומודלים שיפור באמינות המוצר (reliability growth model).

[6] הערת מתרגם: היקף הבדיקות (test scope) הוא סך כל הפעילויות בפרויקט שמוגדרות באחריות קבוצת הבדיקות

- **שיטתית** (methodical): סוג זה של אסטרטגיית בדיקות מתבסס על שימוש שיטתי בסדרה של בדיקות או תנאי בדיקות מוגדרים מראש, כמו טקסונומיה (taxonomy) של סוגים נפוצים של כשלים, רשימת של מאפייני איכות חשובים או תקנים לתצורה אחידה (מבחינת מראה ואופן אינטראקציה) ליישומים עבור מכשירים ניידים או אתרי רשת.
- **תואמת תהליך** (process compliant) (או תואמת תקן, standard compliant): סוג זה של אסטרטגיית בדיקות כולל ניתוח, עיצוב ויישום בדיקות מבוססות על חוקים ותקנים חיצוניים, כגון תקנים ייחודים לתעשייה, תיעוד תהליכים, זיהוי קפדני ושימוש בבסיס הבדיקות או כל תהליך או תקן שהארגון מחויב או בוחר להשתמש בו.
- **מכוונת** (directed) (או מייעצת, consultative): סוג זה של אסטרטגיית בדיקות מכוון בעיקר על ידי עצה, הכוונה או הוראות של בעלי עניין, מומחים בתחום העסקי, או מומחים טכנולוגיים, שיכולים לבוא מחוץ לצוות הבדיקות או מחוץ לארגון עצמו.
- **מניעת נסיגה** (regression-averse): סוג זה של אסטרטגיית בדיקות מונע על ידי הרצון להימנע מנסיגה של יכולות קיימות. אסטרטגיית בדיקות זו כוללת שימוש חוזר במכלול בדיקות קיים (במיוחד מקרי בדיקה ונתוני בדיקה), שימוש נרחב באוטומציה ובדיקות נסיגה וסדרות בדיקה רגילות.
- **מגיבה** (reactive): בסוג זה של אסטרטגיית בדיקות, הבדיקות מגיבות לרכיב או למערכת הנבדקים, ולאירועים המתרחשים בזמן ביצוע הבדיקות, במקום להיות מתוכננים מראש (כמו שקורה בסוגים הקודמים). הבדיקות מתוכננות ומיושמות וניתנות לביצוע מיד תוך שימוש בידע שנצבר מתוצאות בדיקה קודמות. בדיקות חוקרות הן טכניקה נפוצה שמיושמת באסטרטגיה מגיבה.

אסטרטגית בדיקות מתאימה נוצרת לעיתים תוך שילוב מספר סוגים של אסטרטגיות בדיקה. לדוגמה, ניתן לשלב בדיקות מבוססות ניסיון (אסטרטגיה ניתוחית) עם בדיקות חוקרות (אסטרטגיה מגיבה); הן משלימות זו את זו ויכולות להיות יעילות יותר אם משתמשים בהן ביחד.

בעוד שאסטרטגית הבדיקות מספקת תיאור כללי של תהליך הבדיקות, גישת הבדיקות מתאימה את אסטרטגית הבדיקות לפרויקט או גרסה מסוימים. גישת הבדיקות היא נקודת הפתיחה לבחירה של טכניקות בדיקה, רמות בדיקה וסוגי בדיקה, כמו גם להגדרת קריטריון כניסה וקריטריון יציאה (או הגדרה של "נקודת התחלה" (ready) והגדרה של "נקודת הסיום" (done) בהתאמה). התאמת אסטרטגית הבדיקות מבוססת על קבלת החלטות ביחס למורכבות הפרויקט ולמטרות הפרויקט, סוג המוצר המפותח וניתוח סיכוני המוצר. הגישה הנבחרת תלויה בהקשר ולוקחת בחשבון גורמים כגון סיכונים, בטיחות, זמינות ומיומנויות של משאבים, טכנולוגיה, סוג המערכת (לדוגמה מערכת שנבנית בהתאמה או מוצר מדף), מטרות הבדיקות ותקינה.

5.2.3 *קריטריון כניסה וקריטריון יציאה (הגדרה נקודת התחלה ונקודת סיום)*

בכדי ליישם בקרה יעילה על איכות התוכנה והבדיקות, רצוי להגדיר קריטריון שיציין מתי פעילות בדיקה אמורה להתחיל ומתי היא הסתיימה. קריטריון כניסה (שבפיתוח אג'ילי נקראת "הגדרה של נקודת התחלה") מגדיר את תנאי הקדם להתחלה של פעילות בדיקות נתונה. אם קריטריון הכניסה לא מתקיים, סביר שהפעילות תהיה מורכבת יותר, תימשך זמן רב יותר, תעלה יותר כסף ותהיה בעלת סיכון גבוה יותר. קריטריון יציאה (שבפיתוח אג'ילי נקראת "הגדרה של נקודת סיום") מגדיר אילו תנאים יש להשיג כדי להכריז על השלמה של רמת בדיקה או סדרה של בדיקות. יש להגדיר קריטריון כניסה וקריטריון יציאה לכל רמת בדיקה וסוג בדיקה. הקריטריונים ישתנו בהתאם למטרות הבדיקה.

קריטריונים אופייניים לכניסה כוללים:

- זמינות של דרישות ניתנות לבדיקה, סיפורי משתמש ו/או מודלים (לדוגמה, כאשר משתמשים באסטרטגיה של בדיקות מבוססות מודלים)
- זמינות של פריטי בדיקות שעמדו בקריטריון יציאה של כל שלבי הבדיקות הקודמים
- זמינות סביבת הבדיקות
- זמינות כלי הבדיקות הנדרשים
- זמינות נתוני הבדיקות וכל משאבים נדרשים אחרים

קריטריונים אופייניים ליציאה כוללים:

- בוצעו הבדיקות המתוכננות
- הושגה רמת הכיסוי שהוגדרה (לדוגמה, כיסוי של דרישות, סיפורי משתמש, קריטריון קבלה, סיכונים, קוד)
- מספר הפגמים הלא פתורים נמצא בתחום מוגדר מראש
- מספר הפגמים הצפוי שנשאר בתוכנה נמוך
- רמות מאפייני האיכות (אמינות, יעילות ביצועים, שימושיות, אבטחה, ואחרים) מספקים

גם בלי שקריטריון היציאה מושג, לעיתים קרובות פעילויות הבדיקה מופסקות לפני שהגיעו לסיום מאחר שהסתיים התקציב, נגמר הזמן שתוכנן ו/או קיים לחץ להוציא את המוצר לשוק. הפסקת הבדיקות בתנאים אלה מקובלת, אם הסיכון בשחרור המוצר ללא בדיקות נוספות מוצר לבעלי העניין בפרויקט והמנהלים העסקיים והם מסכימים להמשיך.

5.2.4 *לוח זמנים לביצוע בדיקות*

לאחר שנכתבו מקרי הבדיקה והליכי הבדיקה (ומספר הליכי בדיקה מועמדים לאוטומציה) ואורגנו בסדרות בדיקה, ניתן לארגן את סדרות הבדיקות בלוח זמנים לביצוע בדיקות שמגדיר את הסדר בו יבוצעו סדרות בדיקה אלה. לוח הזמנים לביצוע בדיקות ייקח בחשבון גורמים כמו תיעדוף, תלויות, בדיקות אימות, בדיקות נסיגה ואת הסדר היעיל ביותר לביצוע בדיקות אלה.

עקרונית, יש לסדר את מקרי הבדיקה כך שיבוצעו על פי רמת החשיבות שלהם כך שמקרי הבדיקה החשובים יותר יבוצעו ראשונים. אולם, דרך זו לא תעבוד אם קיימות תלויות בין מקרי בדיקה או אם קיימות תלויות לתכונות נבדקות. אם מקרה בדיקה עם חשיבות גבוהה תלוי במקרה בדיקה עם חשיבות נמוכה יותר, יש לבצע קודם את מקרה הבדיקה הפחות חשוב. באופן דומה, אם קיימות תלויות בין מקרי בדיקה רבים, יש לסדר אותם בצורה נכונה, ללא תלות בחשיבות היחסית שלהם. יש לתעדף גם בדיקות אימות ובדיקות נסיגה, בהתבסס על החשיבות במשוב מהיר על שינויים, אך גם כאן יש להתחשב בתלויות השונות.

במקרים אחדים, יהיו מספר אפשרויות חלופיות לסדרות של בדיקות, כשלכל סדרה תהיה רמה שונה של יעילות. במקרים אלה, יש למצוא את הפשרה בין יעילות של ביצוע הבדיקות לבין חשיבות התיעדוף.

5.2.5 *גורמים המשפיעים על מאמץ הבדיקות*

הערכת מאמץ הבדיקות כרוכה בהערכת כמות עבודת הבדיקות שתידרש לשם השגת מטרות הבדיקות לפרויקט מסוים, לגרסה, או מחזור פיתוח. גורמים שמשפיעים על מאמץ הבדיקות עשויים לכלול את מאפייני המוצר, מאפייני תהליך הפיתוח, מאפייני האנשים, ותוצאות הבדיקות, כפי שיוצג להלן.

מאפייני מוצר

- הסיכונים הקשורים למוצר
- איכות בסיס הבדיקות
- גודל המוצר
- מורכבות תחום המוצר
- הדרישות למאפייני איכות (לדוגמה, אבטחה, אמינות)
- רמת הפירוט הנדרשת ממסמכי הבדיקות
- דרישות תאימות לחוקים או לתקינה

מאפייני תהליך פיתוח

- יציבות ובגרות הארגון
- מודל הפיתוח בשימוש
- הגישה לבדיקות
- הכלים בשימוש
- תהליך הבדיקות
- לחץ הזמן

מאפייני האנשים

- המיומנות והניסיון של האנשים המעורבים, במיוחד ניסיון בפרויקטים ומוצרים דומים (לדוגמה, ידע בתחום המוצר (domain knowledge))
- לכידות הצוות וההנהגה

תוצאות הבדיקה

- מספר הפגמים שנמצאים ורמת החומרה שלהם
- כמות העבודה החוזרת הנדרשת

5.2.6 *טכניקות לאומדן בדיקות*

ישנן מספר טכניקות לאומדן המשמשות לקביעת המאמץ הנדרש לבדיקות מספקות. שתיים מהטכניקות הנפוצות ביותר הן:

- טכניקה מבוססת מדדים: אומדן מאמץ הבדיקות מתבסס על מדדים של פרויקט דומה קודם או מתבסס על ערכים מקובלים
- טכניקה מבוססת מומחים: אומדן מאמץ הבדיקות מתבסס על הניסיון של המעורבים במשימות הבדיקה או של מומחים

לדוגמה, בפיתוח אג'ילי, תרשימי התקדמות (burndown chart) הם דוגמאות לגישה מבוסס מדדים שבה המאמץ מתועד ומדווח ואז מחושב לקביעת כמות העבודה שהצוות יכול לבצע במחזור הפיתוח הבא; לעומת זאת, פוקר תכנון (planning poker) הינו דוגמה לגישה מבוססת מומחים שבה חברי בצוות אומדים את המאמץ הנדרש לספק תכונה כשהם מתבססים על הניסיון שלהם (ראו ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

בפרויקטים המשכיים מודלים להסרת פגמים הם דוגמאות לגישה מבוססת מדדים שבה כמות הפגמים והזמן הנדרש להסיר אותם מתועדים ומדווחים ואז משמשים בסיס לאומדן פרויקטים עתידיים בעלי אופי דומה; לעומת זאת, טכניקת אומדן Wideband Delphi היא דוגמה לגישה מבוססת מומחים שבה קבוצות של מומחים מספקים אומדים בהתבסס על ניסיונם (ראו ISTQB-ATM Advanced Level Test Manager).

5.3 ניטור ובקרת בדיקות

המטרה של ניטור בדיקות היא לאסוף מידע ולספק משוב ונראות על פעילויות הבדיקה. המידע המנוטר יכול להיאסף ידנית או אוטומטית וישמש להערכת התקדמות הבדיקות ולמדידה אם הושג קריטריון היציאה מבדיקות, או הושגה נקודת הסיום במטלות בדיקה של פרויקט אג'ילי. השגת המטרה של כיסוי סיכוני מוצר, דרישות או קריטריון קבלה הן דוגמאות לקריטריונים כאלה.

בקרת בדיקות כוללת כל פעולת הנחיה או תיקון המבוצעות כתוצאה מהמידע והמדדים שנאספו ו/או דווחו. פעולות אלה עשויות להיות קשורות לכל פעילויות הבדיקה ולעיתים גם להשפיע על פעילויות פיתוח אחרות.

דוגמאות לפעולות בקרת בדיקות כוללות:

- תיעדוף מחדש של בדיקות כאשר סיכון שזוהה אכן מתרחש (לדוגמה, תוכנה נמסרת מאוחר)
- שינוי לוח זמנים של בדיקות בשל זמינות או אי זמינות של סביבת בדיקות או משאבים אחרים
- הערכה מחדש האם פריט בדיקות שעבר שינוי עומד בקריטריון היציאה או הכניסה

5.3.1 *מדדים בשימוש הבדיקות*

ניתן לאסוף מדדים במהלך פעילויות הבדיקות ובסיומן במטרה להעריך את

- התקדמות העבודה אל מול לוח הזמנים והתקציב המתוכננים
- האיכות הנוכחית של פריט בדיקות
- התאמת גישת הבדיקות לפריט הנבדק
- יעילות פעילויות הבדיקה ביחס למטרות הבדיקה

מדדי בדיקות נפוצים כוללים:

- אחוז העבודה שנעשתה בהכנת מקרי בדיקה אל מול התכנון (או אחוז מקרי הבדיקה שיושמו אל מול התכנון)
- אחוז העבודה שנעשתה בהכנת סביבת הבדיקה אל מול התכנון
- ביצוע מקרי בדיקה (לדוגמה, מספר מקרי הבדיקה שהורצו/לא הורצו, מקרי בדיקה שעברו/נכשלו ו/או מצבי בדיקה שנעברו/נכשלו)
- מידע על פגמים (לדוגמה, צפיפות הפגמים, פגמים שנמצאו ותוקנו, שיעור הכשלים ותוצאות בדיקות אימות)
- כיסוי הדרישות, סיפורי משתמש, קריטריון קבלה, סיכונים או קוד על ידי הבדיקות
- השלמת משימות, הקצאת משאבים ושימוש בהם, המאמץ שהושקע
- עלות הבדיקות, כולל השוואת עלות לתועלת במציאת הפגם הבא או השוואת עלות לתועלת בהרצת הבדיקה הבאה

5.3.2 *מטרות, תוכן וקהל היעד של דוחות הבדיקות*

המטרה של דיווח הבדיקות היא לסכם ולדווח את המידע על פעילות הבדיקות, הן במהלך פעילות הבדיקה והן בסיומה (לדוגמה, רמת בדיקה). ניתן להתייחס אל דו"ח בדיקה שנעשה במהלך פעילות הבדיקה כדו"ח התקדמות הבדיקה, בעוד שדו"ח בדיקות שנעשה בסיום פעילות הבדיקה יקרא דו"ח סיכום הבדיקות.

כחלק מניטור ובקרת הבדיקות, מנהל בדיקות מפרסם באופן קבוע דוחות התקדמות בדיקות לבעלי עניין. דוחות התקדמות בדיקות טיפוסיים עשויים לכלול:

- סטטוס פעילויות הבדיקה והתקדמות הפעילויות אל מול תוכנית הבדיקות
- גורמים המעכבים את הבדיקות
- הבדיקות המתכוננות לתקופת הדיווח הבאה
- איכות מושא הבדיקות

כשקריטריון היציאה מושג, מנהל הבדיקות מפרסם את דו"ח סיכום הבדיקות. הדו"ח מציג סיכום הבדיקות שבוצעו בהתבסס על דו"ח התקדמות הבדיקות האחרון וכל מידע רלוונטי נוסף.

דו"ח התדמות הבדיקות ודו"ח סיכום הבדיקות טיפוסיים עשויים לכלול:

- סיכום הבדיקות שבוצעו
- מידע על מה שהתרחש בזמן הבדיקות
- סטייה מהתכנון, כולל סטיות בלוח זמנים, משך הבדיקות, או המאמץ של פעילויות הבדיקה
- סטטוס הבדיקות ואיכות המוצר ביחס לקריטריון היציאה או נקודת הסיום המוגדרת
- גורמים שחסמו או ממשיכים לחסום את ההתקדמות
- מדדים של פגמים, מקרי בדיקה, כיסוי בדיקה, התקדמות הפעילות וצריכת משאבים (לדוגמה, מדדים שתוארו בפרק 5.3.1)
- סיכונים שנשארו
- תוצרי בדיקות שניתנים לשימוש חוזר

תוכן דו"ח הבדיקות ישתנה בהתאם לפרויקט, דרישות הארגון ומחזור חיי התוכנה. לדוגמה, פרויקט מורכב עם הרבה בעלי עניין או פרויקט שנדרש לעמוד בדרישות תקינה עשויים לחייב דיווח מפורט וקפדני יותר מאשר עדכון תוכנה מהיר. דוגמה נוספת, בפיתוח אג'ילי, דיווח על התקדמות הבדיקות עשוי להיות משולב בלוח המשימות, סיכום הפגמים ותרשימי התקדמות, עליהם ידונו במהלך הפגישות היומיות (stand up meeting) (ראו ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

בנוסף להתאמת דוחות הבדיקה להקשר של הפרויקט, הדוחות יותאמו גם לפי קהל היעד שלהם. סוג המידע וכמות המידע שימסרו לקהל יעד טכני או לצוות בדיקות עשוי להיות שונה ממה שיוצג בדוח סיכום להנהלה. במקרה הראשון, מידע מפורט על סוגי הפגמים ומגמות בפגמים עשוי להיות חשוב. במקרה השני, דו"ח ממבט על עשוי להיות מתאים יותר (לדוגמה, סיכום מצב הפגמים לפי חשיבות, תקציב, לוחות זמנים ומצבי בדיקה שעברו/נכשלו/לא נבדקו).

תקן ISO (ISO/IEC/IEEE 29119-3) מתייחס לשני סוגים של דוחות בדיקה: דוחות התקדמות בדיקות ודוחות השלמת בדיקות (test completion report; שימו לב שבתוכנית לימודים זו דוחות אלה נקראים דוחות סיכום בדיקות (test summary report)). התקן מכיל מבנה ודוגמאות לכל סוג.

5.4 ניהול תצורה

המטרה של ניהול תצורה היא לקבוע ולתחזק את מערכת היחסים בין רכיבי המערכת והבּוֹדְקָה לאורך מחזור חיי הפרויקט והמוצר.

כדי לתמוך בבדיקות בצורה נכונה, ניהול התצורה יהיה מעורב בווידוא הדברים הבאים:

- כל פריטי בדיקות בעלי זיהוי ייחודי, תחת בקרת גרסאות, מעקב אחרי שינויים ומתייחסים זה לזה
- כל פריטי הבּוֹדְקָה בעלי זיהוי ייחודי, תחת בקרת גרסאות, מעקב אחרי שינויים, מתייחסים זה לזה ומתייחסים לגרסאות של פריט(י) בדיקות כך שניתן לתחזק נָעֲקַבּוּת במשך כל תהליך הבדיקות
- מסמכי הבדיקות מתייחסים בצורה חד משמעית לכל המסמכים ופריטי התוכנה המזוהים

במהלך תכנון הבדיקות, יש לזהות וליישם הליכי ניהול תצורה ותשתיות לניהול תצורה (כלים).

5.5 סיכון ובדיקות

5.5.1 הגדרה של סיכון

סיכון קשור באפשרות להתרחשות אירוע עתידי שיהיו לו תוצאות שליליות. רמת הסיכון נקבע על ידי הסבירות להתרחשות האירוע וההשפעה (הנזק) מהתרחשות האירוע.

5.5.2 סיכוני מוצר וסיכוני פרויקט

סיכוני מוצר כוללים את האפשרות שתוצר (לדוגמה, מפרט, רכיב, מערכת או בדיקה) יכשלו לענות על הצרכים הלגיטימיים של המשתמשים או בעלי העניין שלהם. כאשר סיכוני המוצר קשורים עם מאפייני איכות מסוימים של המוצר (לדוגמה, התאמה פונקציונלית, אמינות, יעילות ביצועים, שימושיות, אבטחה, תאימות, תחזוקתיות וניידות), סיכוני מוצר נקראים סיכוני איכות. דוגמאות לסיכוני מוצר כוללים:

- תוכנה עשויה לא לבצע את הפונקציונליות המצופה ממנה על פי המפרט
- תוכנה עשויה לא לבצע את הפונקציונליות המצופה ממנה על פי צרכי המשתמש, הלקוח ו/או בעל העניין
- ארכיטקטורה של מערכת לא תתמוך בצורה נאותה בדרישות לא פונקציונליות
- חישוב מסוים עשוי להיעשות בצורה לא נכונה בתנאים מסוימים
- קוד של לולאת בקרה (loop control structure) עשוי להיכתב בצורה לא נכונה
- זמני תגובה עשויים לא להתאים למערכות עיבוד טרנסאקציות בעלות ביצועים גבוהים
- משוב חווית משתמש (UX) עשוי לא להתאים לציפיות מהמוצר

סיכוני פרויקט כוללים מצבים שבמידה ויתרחשו ישפיעו בצורה שלילית על יכולת הפרויקט להשיג את מטרותיו. דוגמאות לסיכוני פרויקט כוללים:

- בעיות פרויקט:
  - עיכובים שעשויים להתרחש במסירה, השלמת מטלות או השגת קריטריון יציאה או הגדרת נקודת סיום
  - אומדנים לא מדויקים, הקצאת משאבים לפרויקטים עם חשיבות גבוהה יותר או קיצוצי תקציב כלליים בכל הארגון עשויים לגרום לתקציב לא מתאים

- שינויים מאוחרים עשויים לגרום לכמות משמעותית של עבודה מחדש

- בעיות ארגוניות:
  - כישורים, הדרכות וכוח אדם עשויים להיות לא מתאימים
  - בעיות בין אישיות עשויות לגרום לחיכוכים ובעיות
  - משתמשים, צוות עסקי או מומחים בתחום עשויים להיות לא זמינים בשל סתירה בעדיפויות עסקיות

- בעיות פוליטיות:
  - הבודקים עשויים לא לפרסם להעביר את צרכיהם ו/או את תוצאות הבדיקה בצורה נאותה
  - מפתחים ו/או בודקים עשויים לא לפעול לפי מידע שמתקבל בבדיקות או בסקירות (לדוגמה, לא לשפר את שיטות הפיתוח והבדיקה)
  - יכולה להיות גישה לא נכונה, או ציפיות לא נכונות, מהבדיקה (לדוגמה, חוסר הערכה לערך של מציאת פגמים בזמן הבדיקות)

- בעיות טכניות
  - הדרישות עשויות להיות מוגדרות בצורה לא מספקת
  - הדרישות עשויות לא להיות מושגות, בהינתן מגבלות מסוימות
  - סביבת הבדיקות עשויה לא להיות מוכנה בזמן
  - המרת נתונים, תכניות להמרה, והתמיכה בכלים עשויים להגיע מאוחר מידי
  - חולשות בתהליך הפיתוח עשויות להשפיע על העקביות או איכות תוצרי הפרויקט כמו עיצוב, קוד, תצורה, נתוני בדיקות ומקרי בדיקות
  - ניהול פגמים לקוי ובעיות דומות עשויים לגרום למצבור של פגמים וחובות טכניים (technical debt) נוספים

- בעיות של ספק
  - צד שלישי עשוי להיכשל במסירה של תוצר או שרות הכרחיים, או לפשוט רגל
  - נושאים חוזיים עשויים לגרום בעיות לפרויקט

סיכוני פרויקט עשויים להשפיע על פעילויות הפיתוח כמו גם על פעילויות הבדיקה. במקרים מסוימים, מנהלי פרויקט אחראים לניהול כל סיכוני הפרויקט, אך זה לא בלתי שגרתי שלמנהלי הבדיקות תהיה אחריות לסיכוני פרויקט הקשורים בבדיקות.

5.5.3 *בדיקות מבוססות סיכונים ואיכות מוצר*

הסיכון משמש למקד את המאמץ הנדרש במהלך הבדיקות. הוא משמש לקבלת החלטה איפה ומתי להתחיל את הבדיקות ולזיהוי של אזורים שזקוקים לתשומת לב רבה יותר. הבדיקות משמשות להקטנת הסיכוי להתרחשות אירוע שלילי, או להפחתת ההשפעה של אירוע שלילי. הבדיקות משמשות כפעילות הקלת סיכונים (risk mitigation), מספקות מידע על סיכונים שזוהו ועל סיכונים שנשארו (לא נפתרו).

גישת בדיקות מבוססת סיכונים מספקת הזדמנויות יזומות להפחתת הרמות של סיכוני מוצר. גישה זו כוללת ניתוח סיכוני מוצר, שכוללת זיהוי סיכוני מוצר והערכת הסבירות וההשפעה של כל סיכון. המידע

המתקבל על סיכון המוצר משמש כמנחה לתכנון הבדיקות, למפרט מקרי הבדיקה, הכנתם וביצועם ולצרכי ניטור ובקרה הבדיקות. ניתוח מוקדם של סיכוני מוצר תורם להצלחת הפרויקט.

בגישה מבוססת סיכונים, התוצאות של ניתוח סיכוני מוצר משמשות ל:

- החלטה על השימוש בטכניקות בדיקה
- החלטה על רמות בדיקה וסוגי בדיקה מיוחדים שיש לבצע (לדוגמה, בדיקות אבטחה, בדיקות נגישות)
- החלטה על היקף בדיקות שיש לבצע
- תיעדוף בדיקות בניסיון למצוא את הפגמים הקריטיים מוקדם ככל האפשר
- החלטה אם ניתן ליישם כל פעילות אחרת בנוסף לבדיקות בכדי להקטין את הסיכון (לדוגמה, להדריך מעצבים לא מנוסים)

בדיקות מבוססות סיכונים מתבססות על הידע המצטבר ועל התובנות של בעלי העניין בפרויקט לשם ביצוע ניתוח סיכוני מוצר. כדי לוודא שהסבירות לכשל של המוצר קטנה למינימום, ניהול סיכונים מספק גישה מסודרת ל:

- ניתוח (והערכה מחודשת על בסיס קבוע) של מה יכול להשתבש (סיכונים)
- החלטה באיזה סיכונים חשוב לטפל
- יישום פעולות להקטין סיכונים אלה
- הכנת תכניות מגירה לטיפול בסיכונים במידה ויהפכו לאירועים אמיתיים

בנוסף, הבדיקות עשויות לגלות סיכונים חדשים, לעזור בקבלת החלטה איזה סיכונים יש להקטין ובהפחתת חוסר הודאות על סיכונים.

5.6 ניהול פגמים

מאחר ואחת המטרות של בדיקות היא למצוא פגמים, יש לתעד פגמים שנמצאו במהלך הבדיקות. הדרך בה יתועדו הפגמים עשויה להשתנות בהתאם להקשר הרכיב או המערכת הנבדקים, לרמת הבדיקה ולמחזור חיי התוכנה. יש לחקור כל פגם שמזוהה ולתעד מהגילוי והסיווג של הפגם ועד לפתרונו (לדוגמה, תיקון הפגם ובדיקת אימות מוצלחת של הפתרון, דחייה לגרסה מאוחרת יותר, קבלה כמגבלת מוצר קבועה, וכו'). כדי לנהל את כל הפגמים עד לשלב הפתרון, הארגון צריך להגדיר תהליך ניהול פגמים שיכלול תהליך עבודה וכללים לסיווג. כל המעורבים בניהול הפגמים צריכים להסכים על התהליך, כולל מעצבים, מפתחים, בודקים ומנהלי מוצר. במספר ארגונים, תיעוד ומעקב אחרי פגמים עשוי להיות מאוד לא פורמלי.

במהלך תהליך ניהול הפגמים, חלק מדוחות הפגמים עשויים לתאר תוצאה חיובית כוזבת, שאינה כשל שנגרם על ידי פעם. לדוגמה, בדיקה עשויה להיכשל כאשר חיבור הרשת מתנתק או איטי מהצפוי. התנהגות זו אינו נגמרת מפגם במושא הבדיקות, אך היא חריגה שיש לחקור. הבודקים צריכים לנסות ולהקטין למינימום את מספר התוצאות החיוביות כוזבות שמדווחות כפגמים.

ניתן לדווח על פגמים במהלך כתיבת הקוד, ניתוח סטטי, סקירות, בדיקות דינמיות או שימוש במוצר תוכנה. ניתן לדווח על פגמים בקוד או במערכות עובדות, או בכל סוג של מסמך, כולל דרישות, סיפורי משתמש וקריטריון קבלה, מסמכי פיתוח, מסמכי בדיקה, מדריכי משתמש או מדריכי התקנה. כדי שתהליך ניהול הפגמים יהיה יעיל, על הארגונים להגדיר את התכונות של הפגמים, את הסיווג שלהם ואת תהליכי מחזור החיים של הפגמים.

מטרות של דוחות פגמים טיפוסיים:

- לספק למפתחים ולאחרים מידע על כל אירוע שלילי שהתרחש, לאפשר להם לזהות תוצאות ספציפיות, לבודד את הבעיה בעזרת בדיקת שחזור מינימלית ולתקן את הפגם האפשרי במידת האפשר או לפתור את הבעיה בכל דרך שהיא.
- לספק למנהלי בדיקות אמצעים למעקב אחרי איכות התוצרים ואחרי ההשפעה על הבדיקות (לדוגמה, אם מדווחים פגמים רבים, הבודקים ישקיעו זמן רב על דיווח במקום להריץ בדיקות נוספות ויהיה צורך ביותר בדיקות אימות)
- לספק רעיונות לשיפור תהליכי פיתוח ובדיקות

דו"ח פגם טיפוסי שיתועד במהלך בדיקות דינמיות יכלול:

- מזהה
- כותרת ותיאור תמציתי קצר של הפגם המדווח
- תאריך הדיווח, הארגון שמדווח ומחבר הדו"ח
- זיהוי מושא הבדיקות (פריט התצורה הנבדק) והסביבה בה התגלה הפגם
- שלב מחזור חיי הפיתוח שבו התגלה הפגם
- תיאור הפגם בצורה שתאפשר שחזור ופתרון, כולל לוגים, עותק של בסיס נתונים (database dump), צילומי מסך והקלטות (אם הפגם התגלה במהלך ביצוע בדיקות)
- התוצאה הצפויה והתוצאה בפועל
- תחום או דרגת ההשפעה (חומרה) של הפגם
- דחיפות/חשיבות התיקון
- מצב דו"ח הפגם (לדוגמה, פתוח, דחוי, כפול, ממתין לתיקון, ממתין לבדיקות אימות, נפתח מחדש, סגור)
- מסקנות, המלצות ואישורים
- נושאים כלליים, למשל אזורים אחרים שעשויים להיות מושפעים מהשינוי שנעשה עקב הפגם
- היסטוריית שינויים, למשל רצף הפעולות שעשו חברי הצוות לצרוך בידוד הפגם, תיקון הפגם ואימות התיקון
- הפניות, כולל מקרה הבדיקה שגילה את הפגם

חלק מהפרטים הללו יכללו ו/או ינוהלו אוטומטית כאשר נשתמש בכלי ניהול בדיקות, למשל, הקצאה אוטומטית של מזהה, הקצאה ועדכון סטטוס דו"ח הפגם בזמן תהליך העבודה וכו'. פגמים שנמצאו במהלך בדיקות סטטיות, בייחוד סקירות, יתועדו בדרך כלל בצורה שונה, לדוגמה, בסיכום פגישת הסקירה.

ניתן למצוא דוגמה לתוכן של דו"ח פגם בתקן ISO (ISO/IEC/IEEE 29119-3) (שמתייחס לדו"ח פגם כדו"ח תקלה).

6 כלים תומכי בדיקות — 40 דקות

מושגים

בדיקות מונחות נתונים (data-driven testing), בדיקות מונחות מילות מפתח (keyword-driving testing), כלי לבדיקות ביצועים (performance testing tool), אוטומציה של בדיקות (test automation), כלי לביצוע בדיקות (test execution tool), כלי לניהול בדיקות (test management tool)

יעדי הלימוד של פרק כלים תומכי בדיקות

6.1 שיקולים בבחירת כלי בדיקות

FL-6.1.1 (K2) סווג כלי בדיקה בהתאם למטרה שלהם ולפעילויות הבדיקה בהם הם תומכים

FL-6.1.2 (K1) זהה את היתרונות והסיכונים באוטומציה של בדיקות

FL-6.1.3 (K1) הכר שיקולים מיוחדים של כלים להרצת בדיקות וכלים לניהול בדיקות

6.2 שימוש יעיל בכלים

FL-6.2.1 (K1) זהה את העקרונות העיקריים בבחירה של כלים

FL-6.2.2 (K1) הכר את המטרות של פרויקט ניסיוני (פיילוט) לשם הכנסת כלים לארגון

FL-6.2.3 (K1) זהה את גורמי ההצלחה להערכה, יישום, פריסה (deployment) ותמיכה שוטפת של כלי בדיקות בארגון

6.1 שיקולים בבחירת כלי בדיקות

ניתן להשתמש בכלי בדיקה לתמיכה בפעילות בדיקות אחת או יותר. כלים כאלה כוללים:

- כלים בשימוש ישיר בבדיקות, כמו כלים לביצוע בדיקות וכלים להכנת נתוני בדיקות
- כלים שמסייעים בניהול דרישות, מקרי בדיקה, הליכי בדיקה, תסריטי בדיקה אוטומטיים, תוצאות בדיקה, נתוני בדיקה ופגמים וכלים לדיווח וניטור ביצוע בדיקות
- כלים המשמשים לחקירה והערכה
- כל כלי המסייע בבדיקות (לצורך העניין גם גיליון אלקטרוני הוא כלי בדיקות)

6.1.1 סיווג של כלי בדיקות

לכלי בדיקות יהיו אחת מהמטרות הבאות או יותר, בהתאם להקשר:

- שיפור יעילות של פעילויות הבדיקה על ידי אוטומציה של פעולות חוזרות או פעולות שדורשות משאבים רבים כאשר הן נעשות באופן ידני (לדוגמה, הרצת בדיקות, בדיקות נסיגה)
- שיפור יעילות של פעילויות הבדיקה על ידי תמיכה בפעילויות בדיקה ידניות לאורך הליך הבדיקות (ראו פרק 1.4)
- שיפור איכות פעילויות הבדיקה על ידי שמירה על עקביות בבדיקות ורמה גבוהה של שיחזור פגמים
- אוטומציה של פעילויות שלא ניתן לבצע ידנית (לדוגמה, בדיקות ביצועים בהיקפים גדולים)
- הגדלת אמינות הבדיקות (לדוגמה, על ידי אוטומציה בהשוואת כמות גדולה של נתונים או סימולציה של התנהגות המערכת)

ניתן לסווג כלים על פי מספר קריטריונים כגון מטרה, מחיר, מודל רישיון (לדוגמה, מסחרי או קוד פתוח) והטכנולוגיה שבשימוש הכלי. בתוכנית לימודים זו יעשה הסיווג של הכלים בהתאם לפעילויות הבדיקה בהן הם תומכים.

חלק מהכלים תומכים בפעילות אחת עיקרית בצורה ברורה; אחרים עשויים לתמוך ביותר מפעילות אחת אבל יסווגו תחת הפעילות אליה הם קשורים ביותר. כלים מספק אחד, בעיקר כאלה שתוכננו לעבוד ביחד, עשויים להימסר כחלק מחבילה משולבת.

מספר סוגים של כלי בדיקה עשויים להיות פולשניים (intrusive), כלומר הם עשויים להשפיע על תוצאת הבדיקה. לדוגמה, זמן התגובה המעשי של יישום עשוי להיות שונה בשל ההוראות הנוספות שמבוצעות על ידי כלי בדיקת ביצועים, או שסך כיסוי הקוד שהושג עשוי להיות מסולף בשל השימוש בכלי לכיסוי. ההשלכות של שימוש בכלי פולשני נקראות אפקט הגשושית (probe effect).

מספר כלים מציעים תמיכה שמתאימה יותר למפתחים (לדוגמה, כלים שמשמשים לבדיקות רכיבים ואינטגרציה). כלים אלה מסומנים ב"(מ)" בחלק הבא.

כלים תומכים בניהול בדיקות ובוֹדְקָה (testware)

כלי ניהול בדיקות עשויים לתמוך בכל פעילויות בדיקה לאורך כל מחזור חיי פיתוח התוכנה. דוגמאות לכלים שתומכים בניהול בדיקות ובוֹדְקָה כוללות:

- כלים לניהול בדיקות וכלים לניהול מחזור חיים (ALM – Application Lifecycle Management)
- כלים לניהול דרישות (לדוגמה, נְעֲקָבוּת של מושאי בדיקות)

- כלים לניהול פגמים (defects)
- כלים לניהול תצורה (configuration management)
- כלים לאינטגרציה מתמשכת (continuous integration) (מ)

כלים תומכים בבדיקות סטטיות

כלים לבדיקות סטטיות מקושרים עם פעילויות ויתרונות המתוארים בפרק 3. דוגמות לכלים כאלה כוללות:

- כלים תומכי סקירות (reviews)
- כלי ניתוח סטטי (static analysis) (מ)

כלים תומכים בעיצוב ויישום בדיקות

כלים לעיצוב בדיקות מסייעים ביצירה של תוצרי עיצוב ויישום בדיקות הניתנים לתחזוקה, כולל מקרי בדיקה, הליכי בדיקה ונתוני בדיקה. דוגמאות לכלים כאלה כוללות:

- כלים לעיצוב בדיקות
- כלי בדיקות מבוססות מודל (model based testing)
- כלים להכנה של נתוני בדיקה
- כלים לפיתוח מונחה בדיקות קבלה (ATDD – Acceptance Test Driven Development) ופיתוח מונחה התנהגות (BDD – Business Driven Development)
- כלים לפיתוח מונחה בדיקות (TDD – Test Driven Development) (מ)

במקרים אחדים, כלים שתומכים בעיצוב ויישום בדיקות עשויים גם לתמוך בביצוע בדיקות ובתיעוד שלהם, או שהם מספקים את הפלט שלהם ישירות לכלים אחרים שתומכים בביצוע ובתיעוד של בדיקות.

כלים תומכים בביצוע ותיעוד של בדיקות

ישנם כלים רבים התומכים ומשפרים את פעילויות הרצת ותיעוד הבדיקות. דוגמאות לכלים אלה כוללות:

- כלי הרצת בדיקות (לדוגמה, להרצה של בדיקות נסיגה)
- כלי כיסוי (לדוגמה, כיסוי דרישות, כיסוי קוד (מ))
- רתמות בדיקה (test harnesses) (מ)
- כלי מסגרת בדיקות יחידה (unit test framework) (מ)

כלים תומכים במדידת ביצועים וניתוח דינמי

כלים למדידת ביצועים וניתוח דינמי חיוניים לתמיכה בפעילויות בדיקה של ביצועים ועומס, מאחר ולא ניתן לבצע פעילויות אלה בצורה ידנית ויעילה. דוגמאות לכליים אלה כוללות:

- כלים לבדיקות ביצועים
- כלים לניטור (monitoring)
- כלי ניתוח דינמי (dynamic analysis) (מ)

כלים תומכים בצרכי בדיקות מיוחדים

בנוסף לכלים שתומכים בתהליך הבדיקות הכללי, ישנם כלים רבים אחרים שתומכים נושאי בדיקה ייחודיים. דוגמאות לכלים כאלה כוללות כלים שמתמקדים ב:

- הערכת איכות נתונים
- המרה והגירה של נתונים (data conversion and migration)
- בדיקת שימושיות (usability testing)
- בדיקת נגישות (accessibility testing)
- בדיקות לוקליזציה (localization testing)
- בדיקות אבטחה
- בדיקות ניידות (portability testing) (לדוגמה, בדיקה של תוכנה על גבי מספר פלטפורמות נתמכות)

6.1.2 *יתרונות וסיכונים של אוטומציה של בדיקות*

רכישה של כלי לא מבטיחה הצלחה. כל כלי חדש שמוכנס לארגון ידרוש מאמץ להשגת יתרונות אמיתיים שיחזיקו לאורך זמן. שימוש בכלים לבדיקות פותח הזדמנויות ומכיל יתרונות, אבל ישנם גם סיכונים. הדבר נכון במיוחד לכלים לביצוע בדיקות (אליהם מתייחסים לעיתים כאוטומציה של בדיקות).

יתרונות אפשריים לשימוש בכלים לתמיכה בביצוע בדיקות כוללים:

- הפחתה בעבודה ידנית חוזרת (לדוגמה, הרצה של בדיקות נסיגה, מטלות של הקמה סביבת בדיקות והחזרתה לקדמותה, הכנסה חוזרת של נתוני בדיקות ובדיקה אל מול תקן כתיבת קוד) וכך לחסוך בזמן
- גידול בעקביות ויכולת חזרה (לדוגמה, יצירת נתוני בדיקה בצורה עקבית, בדיקות מבוצעות על ידי כלי באותו סדר ובאותו קצב ובדיקות שנגזרות מדרישות באופן עקבי)
- הערכה יותר אובייקטיבית (לדוגמה, מדידות סטטיות, כיסוי)
- נגישות קלה יותר למידע על בדיקות (לדוגמה, סטטיסטיקות וגרפים על התקדמות הבדיקות, שיעור הפגמים וביצועים)

סיכונים אפשריים בשימוש בכלים כוללים:

- ציפיות מהכלי עשויות להיות לא מציאותיות (כולל פונקציונליות וקלות השימוש)
- הערכת הזמן, העלות והמאמץ להכנסה ראשונית של הכלי עשויה להיות בחסר (כולל זמן ותקציב להדרכה ומומחיות חיצונית)
- הערכת הזמן והמאמץ הנדרשים להשגת יתרונות משמעותיים ונמשכים מהכלי עשויה להיות בחסר (כולל הצורך בשינויים בתהליך הבדיקות ושיפור מתמיד בדרך בה משתמשים בכלי)
- הערכת המאמץ הנדרש לתחזוקת נכסי הבדיקות (test assets) שנוצרים על ידי הכלי עשויה להיות בחסר
- התלות בכלי עשויה להיות גבוהה מידי (הכלי יראה כתחליף לעיצוב וביצוע בדיקות, או שיעשה שימוש בבדיקות אוטומטיות במקומות בהם עדיף להשתמש בבדיקות ידניות)
- בקרת גרסאות של נכסי הבדיקות עשויה להיזנח

- בעיות היחסים ההדדיים בין כלים קריטיים עשויה להיזנח, למשל כלי ניהול דרישות, כלי ניהול תצורה, כלי ניהול פגמים וכלים מספקים שונים
- ספק של כלי עשוי לסגור את העסק, להוציא את הכלי לפרישה (כלומר לא לתמוך בכלי יותר), או למכור את הכלי לספק אחר
- הספק עשוי להגיב לאט לבעיות תמיכה, עדכונים ותיקון פגמים בכלי
- פרויקט קוד פתוח עשוי להיפסק
- פלטפורמה חדשה או טכנולוגיה חדשה עשויות לא להיתמך על ידי הכלי
- יתכן ואין בעלות ברורה על הכלי (לדוגמה, לצורך הדרכה, עדכונים וכו')

6.1.3 *שיקולים מיוחדים לכלי ביצוע בדיקות וכלי ניהול בדיקות*

בכדי שיישום הכלי יהיה מוצלח ויעבור בצורה חלקה, ישנם מספר שיקולים שיש לקחת בחשבון כשבוחרים כלים לביצוע בדיקות ולניהול בדיקות ומשלבים אותם בארגון.

כלים לביצוע בדיקות

כלים לביצוע בדיקות מפעילים את מושאי הבדיקות באמצעות תסריטי בדיקה אוטומטיים. סוג זה של כלים דורש לעיתים מאמץ רב לשם השגת תועלת משמעותית.

יצירת בדיקות על ידי הקלטה של פעולות שמבצע בודק ידני נשמע אטרקטיבי, אבל גישה זו לא יכולה להתאים למספר רב של תסריטי בדיקה. תסריט מוקלט הוא ייצוג ישיר של מקרה בדיקה עם נתונים ופעולות מוגדרים לכל תסריט. סוג זה של תסריט עשוי להיות לא יציב אם מתרחש אירוע לא צפוי. הדור האחרון של כלים אלה, שמנצל את היתרונות של טכנולוגיה "חכמה" לזיהוי תמונות, הגדיל את התועלת המתקבלת מסוג זה של כלים, למרות שעדיין נדרשת תחזוקה שוטפת של התסריטים שנוצרים מההקלטה, מאחר וממשק המשתמש של המערכת מתפתח במשך הזמן.

גישת הבדיקות מונחות הנתונים מפרידה בין הקלט לבדיקות והתוצאות הצפויות, בדרך כלל בגיליון אלקטרוני, ומשתמשת בתסריט בדיקות כללי (גנרי) שיכול לקרוא את נתוני הקלט ולבצע את אותו תסריט הבדיקה עם נתונים שונים. בודקים שלא יודעים את השפה בה כתוב התסריט (שפת סקריפטינג) יכולים ליצור נתוני בדיקה חדשים לתסריטים אלה המוגדרים מראש.

בגישת הבדיקות מונחות מילות מפתח, תסריט כללי מעבד מילות מפתח המתארות את הפעולות שיש לבצע (נקראות גם מילות פעולה (action words)). מילות המפתח קוראות לתסריטים מתאימים שמעבדים את נתוני הבדיקה המשויכים. בודקים (גם אם לא מכירים את שפת התסריט) יכולים להגדיר בדיקות כשהם משתמשים במילות המפתח והנתונים המשויכים, שניתן להתאים למערכת הנבדקת. עוד פרטים ודוגמאות לבדיקות מונחות נתונים ובדיקות מונחות מילות מפתח ניתנות בתוכנית הלימודים ISTQB-TAE Advanced Level Test Automation Engineer Syllabus, ב-Fewster 1999 ו-Buwalda 2001.

הגישות שהובאו כאן דורשות מומחיות בשפת תסריטים (סקריפטינג) (של בודקים, מפתחים או מומחים באוטומציה של בדיקות). אולם ללא קשר לטכניקת התסריט שבשימוש, יש להשוות את התוצאות הצפויות של כל בדיקה לתוצאות שהתקבלו מהבדיקה, בצורה דינמית (כשהבדיקה עדיין מבוצעת) או שהיא תישמר להשוואה מאוחרת (אחרי הביצוע).

כלים לבדיקות מבוססות מודלים (MBT) מאפשרים להגדיר מפרט פונקציונלי בצורה של מודל, למשל תרשים פעילות (activity diagram). מטלה זו מבוצעת בדרך כלל על ידי מעצב מערכת. כלי ה-MBT מתרגם את המודל בכדי ליצור מפרט למקרי בדיקה שיכול להישמר בכלי ניהול בדיקות ו/או להיות מבוצע על ידי כלי לביצוע בדיקות (ראו ISTQB-MBT Foundation Level Model-Based Testing Syllabus).

כלים לניהול בדיקות

כלים לניהול בדיקות נדרשים לעיתים להתממשק עם כלים אחרים או עם גיליונות אלקטרוניים, מסיבות שונות, כולל:

- הפקה של מידע מועיל בתבנית (פורמט) שמתאים לצרכים של הארגון
- תחזוקה של נֶעֱקָבוּת לדרישות בכלי ניהול הדרישות
- קישור למידע על גרסת מושא הבדיקות בכלי ניהול תצורה

חשוב במיוחד לשים לב לשיקולים הנ"ל כאשר משתמשים בכלי משולב (לדוגמה כלי לניהול מחזור חיים (ALM)), שכולל כלי לניהול בדיקות (וככל הנראה גם כלי לניהול פגמים) כמו גם מודולים אחרים (לדוגמה, מידע על לוח זמנים ותקציב של פרויקט) בהם משתמשות קבוצות שונות בתוך הארגון.

6.2 שימוש יעיל בכלים

6.2.1 *עקרונות מרכזיים לבחירה של כלים*

השיקולים העיקריים בבחירה של כלי לארגון כוללים:

- הערכת בגרות הארגון, החוזקות והחולשות שלו
- זיהוי הזדמנויות לשיפור תהליך הבדיקות באמצעות הכלים
- הבנת הטכנולוגיה בה משתמש מושא(י) הבדיקות, כדי לבחור כלי שיתאים לטכנולוגיה הזו
- הבנת הכלים שנמצאים בשימוש הארגון לאינטגרציה מתמשכת (continuous integration) ובנייה מתמשכת (continuous build), כדי לוודא שהכלי מתאים ומשתלב עם הכלים הקיימים
- הערכת הכלי אל מול דרישות ברורות וקריטריונים אובייקטיבים
- בדיקה האם הכלי זמין לתקופה ניסיון חינמית (ולכמה זמן)
- הערכת הספק (כולל הדרכה, תמיכה והיבטים מסחריים) או הערכת התמיכה לכלים לא מסחריים (לדוגמה כלי קוד פתוח)
- זיהוי דרישות פנים-ארגוניות להדרכה וחונכות בשימוש בכלי
- הערכת צרכי ההדרכה, בהתחשב בכישורי הבדיקות (והאוטומציה של הבדיקות) של מי שיעבדו ישירות עם הכלי
- הערכת יתרונות וחסרונות של מודלים שונים של רישיון (לדוגמה מסחרי או קוד פתוח)
- הערכת יחס עלות-תועלת מבוססת על מקרה עסקי מסוים

כצעד אחרון יש לבצע פעילות של הוכחת היתכנות (proof of concept) בכדי לקבוע האם הביצועים של הכלי יעילים בעבודה עם התוכנה הנבדקת ועם התשתית הנוכחית, או, אם צריך, לזהות את השינויים לתשתית הדרושים לצורך עבודה יעילה עם הכלי.

6.2.2 *פרויקט ניסיוני להכנסת הכלי לארגון*

אחרי שמסתיימת בחירת הכלי ובוצעה הוכחת היתכנות מוצלחת, יש להכניס את הכלי לארגון. הדבר מתחיל בדרך כלל עם פרויקט ניסיוני, שלו המטרות הבאות:

- השגת ידע מעמיק על הכלי, הבנת החוזקות והחולשות שלו
- הערכת ההתאמה של הכלי לתהליכים ונהלים קיימים, והחלטה מה צריך לשנות
- החלטה על דרכים סטנדרטיות לשימוש, ניהול, אחסון ותחזוקה של הכלי והתוצרים הקשורים אליו (לדוגמה, החלטה על מוסכמות לשמות של קבצים ובדיקות, בחירה של תקנים לכתיבת קוד, יצירת ספריות והגדרה של מודולריות של סדרות הבדיקות)
- הערכה האם התועלת מהכלי תושג בעלות סבירה
- הבנה של המדדים שרצוי שהכלי יאסוף וידווח, והגדרת התצורה של הכלי כך שמדדים אלה אכן יאספו וידווחו

6.2.3 *גורמי הצלחה לכלים*

גורמי הצלחה להערכה, יישום, פריסה ותמיכה מתמשכת בכלים בתוך הארגון כוללת:

- הכנסת הכלי לשאר הארגון בצורה הדרגתית
- התאמת תהליכים ושיפורם כדי שיתאימו לשימוש בכלי
- מתן הדרכה, אימון וחונכות למשתמשי הכלי
- הגדרת הנחיות לשימוש בכלי (לדוגמה, תקן פנימי לאוטומציה)
- יישום מערכת לאיסוף מידע על השימוש בפועל בכלי
- ניטור השימוש בכלי והתועלת המופקת ממנו
- מתן תמיכה למשתמשים
- איסוף משוב וניתוח לקחים מכל המשתמשים

חשוב גם לוודא שהכלי משולב ברמה הטכנית והארגונית לתוך מחזור חיי התוכנה, דבר שיתכן ויכלול ארגונים נוספים, כגון DevOps או חברות חיצוניות שמפתחות חלקים מהתוכנה (3rd party).

למידע נוסף, עצות ושיתוף בידע על שימוש בכלי ביצוע בדיקות, ראו Graham 2012.

7 הפניות

תקנים

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

מסמכי ISTQB

ISTQB Glossary

ISTQB Foundation Level Overview 2018

ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB-AT Foundation Level Agile Tester Extension Syllabus

ISTQB-ATA Advanced Level Test Analyst Syllabus

ISTQB-ATM Advanced Level Test Manager Syllabus

ISTQB-SEC Advanced Level Security Tester Syllabus

ISTQB-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-ETM Expert Level Test Management Syllabus

ISTQB-EITP Expert Level Improving the Test Process Syllabus

ספרים ומאמרים

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp 1-*

Shull, F., Rus, I., Basili, V. July 2000. "How Perspective-Based Reading can Improve Requirement Inspections." *IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8 - 10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

מקורות אחרים (ללא הפניה ישירה בתוכנית הלימודים)

Black, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Spillner, A., Linz, T., and Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8 נספח א – הרקע לתוכנית הלימודים

היסטוריה של המסמך

מסמך זה הוא תוכנית הלימודים של ISTQB לרמת הבסיס (Certified Tester foundation Level), שהיא הרמה הראשונה בהכשרה הבינלאומית המאושרת על ידי ISTQB (www.istqb.org).

מסמך זה נכתב בין השנים 2014 ו-2018 על ידי קבוצת עבודה המורכבת מאנשים שהתמנו על ידי הארגון הבינלאומי להכשרות בבדיקות תוכנה (ISTQB). גרסת 2018 נסקרה על ידי נציגים מכל הוועדים החברים של ISTQB ולאחר מכן על ידי נציגים של קהילת בדיקות התוכנה הבינלאומית.

המטרות של ההכשרה הבסיסית

- להשיג הכרה בבדיקות כהתמחות חיונית ומקצועית של הנדסת תוכנה
- לספק מסגרת אחידה לפיתוח קריירה של בודקים
- לאפשר לבודקים מקצועיים הכרה על ידי מעסיקים, לקוחות ועמיתים ולהעלות את הפרופיל של הבודקים
- לקדם שיטות בדיקה טובות ואחידות עבור כל הדיסציפלינות של הנדסת תוכנה
- לזהות נושאי בדיקה רלוונטיים ובעלי ערך לתעשייה
- לאפשר לספקי תוכנה להעסיק בודקים מוסמכים ובכך להשיג יתרון מסחרי על מתחריהם עם פרסום מדיניות גיוס הבודקים שלהם
- לספק הזדמנות לבודקים ואלה שמתעניינים בבדיקות לרכוש הכשרה בינלאומית בנושא

מטרות ההכשרה הבינלאומית

- לאפשר השוואה של ידע בבדיקות בין מדינות שונות
- לאפשר לבודקים לנוע בין מדינות בצורה קלה יותר
- לאפשר לפרויקטים בינלאומיים/רב-לאומיים הבנה משותפת של נושאים בבדיקות
- להגדיל את מספר הבודקים המוסמכים ברחבי העולם
- להוות גורם משפיע/בעל ערך כיוזמה בינלאומית יותר מאשר כל גישה מבוססת מדינה
- לפתח גוף ידע בינלאומי משותף על בדיקות באמצעות תוכנית הלימודים ומילון המונחים\ ולהגדיל את רמת הידע על בדיקות עבור כל המשתתפים
- לקדם את הבדיקות כמקצוע ביותר מדינות
- לאפשר לבודקים לרכוש הכשרה מוכרת בשפת האם שלהם
- לאפשר שיתוף ידע ומשאבים בין מדינות
- לספק הכרה בינלאומית של בודקים והכשרה זו בעזרת השתתפות נציגים ממדינות רבות

דרישות כניסה להכשרה זו

קריטריון הכניסה לבחינה של תוכנית לימודים זו (ISTQB Certified Tester Foundation Level) היא שהמועמד יתעניין בבדיקות תוכנה. אולם, מומלץ גם שהמועמד:

- יהיה בעל רקע מינימלי בפיתוח תוכנה או בבדיקות תוכנה, לדוגמה ניסיון של שישה חודשים בבדיקות מערכת או בבדיקות קבלת משתמש
- יעבור קורס הדרכה מאושר על ידי אחד הוועדים החברים של ISTQB

רקע והיסטוריה של הסמכת הבסיס בבדיקות תוכנה

ההסכמה העצמאית בבדיקות תוכנה החלה בבריטניה על ידי Information System Examination Board (ISEB) שהוקמה על ידי British Computer Society (BCS), שהקימה ועד של בדיקות תוכנה ב-1998 (www.bcs.org.uk/iseb). בשנת 2002, ארגון ASQF בגרמניה התחיל לתמוך בתוכנית הכשרה גרמנית לבודקים (www.asqf.de). תוכנית לימודים זו מבוססת על תוכנית הלימודים של ISEB ושל ASQF; התוכנית כוללת תוכן שאורגן מחדש ועודכן כמו גם תוכן חדש, כשהדגש הושם על נושאים שיספקו את העזרה המעשית ביותר לבודקים.

הסמכות בסיס בבדיקות תוכנה (לדוגמה מ-ISEB, ASQF או מוועד חבר ב-ISTQB) שניתנו לפני שההסמכה הבינלאומית התפרסמה, יוכרו כמקבילים ושווים להסמכה הבינלאומית. הסמכת הבסיס אינה פוקעת ואין צורך לחדש אותה. תאריך הענקת ההסכמה מופיעה על תעודת ההסמכה.

היבטים מקומיים בכל מדינה נשלטים על ידי הוועד החבר ב-ISTQB. החובות של הוועדים החברים מוגדרות על ידי ISTQB, אך מיושמות בכל מדינה. החובות של וועדים חברים כוללות הסמכה של ספקי הדרכה וארגון בחינות.

9 נספח ב – יעדי לימוד/רמות ידע קוגניטיביות

כל נושא בתוכנית הלימודים ייבחן על פי יעדי הלימוד המתאימים לנושא.

רמה 1: לזכור (K1)

המועמד יזהה ויזכור נושא או מושג.

מילות מפתח: לזהות, לזכור, לשחזר, להכיר, לדעת

דוגמאות

המועמד יכול לזהות את ההגדרה של "כשל" כ:

- "אי הספקה של שירות למשתמש קצה או לכל בעל עניין אחר" או
- "סטייה של רכיב או מערכת מהשירות או התוצאה המצופה"

רמה 2: להבין (K2)

המועמד יוכל לבחור סיבות או הסברים למשפטים הקשורים לנושא, ויכול לסכם, להשוות, לסווג ולתת דוגמאות על מושג הבדיקות.

מילות מפתח: לסכם, להכליל, לפשט, לסווג, להשוות, למפות, להציג ניגודים, להדגים, לפרש, לתרגם, לייצג, להסיק, לבנות מודלים

דוגמאות:

המועמד יכול להסביר את הסיבה לכך שניתוח ועיצוב בדיקות יבוצעו מוקדם ככל האפשר:

- למצוא פגמים כאשר זול יותר להסיר אותם
- למצוא את הפגמים החשובים ראשונים

המועמד יכול להסביר את הדמיון וההבדלים בין בדיקות אינטגרציה ובדיקות מערכת:

- דמיון: מושאי הבדיקות עבור בדיקות אינטגרציה ובדיקות מערכת כוללים יותר מרכיב אחד ושני סוגי הבדיקות כוללים בדיקות לא פונקציונליות
- הבדלים: בדיקות אינטגרציה מתמקדות בממשקים ופעולות גומלין ובדיקות מערכת מתמקדות בהיבטים של המערכת השלמה, לדוגמה עיבוד מקצה לקצה

רמה 3: ליישם (K3)

המועמד יכול לבחור את היישום הנכון של מושג או טכניקה וליישם אותה בהקשר נתון.

מילות מפתח: ליישם, לבצע, להשתמש, לבצע הליך, ליישם הליך

דוגמאות:

- המועמד יכול לזהות ערכי גבול של מחלקות תקפות ולא תקפות
- המועד יכול לבחור מקרי בדיקה מתרשים מעבר בין מצבים נתון כדי לכסות את כל המעברים

הפניות (לרמות קוגניטיביות של יעדי לימוד)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon: Boston MA

10 נספח ג – Release Notes

תוכנית הלימודים לרמת הבסיס על פי ISTQB, גרסת 2018, הינה עדכון ושכתוב של גרסת 2011. מסיבה זו אין release notes מפורטים לכל פרט וחלק. סיכום של השינויים העיקריים מופיע כאן. בנוסף, במסמך release notes נפרד, ISTQB מספקת נעקבות בין יעדי הלימוד של גרסת 2011 לגרסת 2018 של תוכנית הלימודים לרמת הבסיס. המסמך מראה איזה יעדי לימוד נוספו, עודכנו או הוסרו (מסמך זה אינו מתורגם לעברית).

נכון לתחילת 2017 יותר מ-550,000 אנשים ביותר מ-100 מדינות נבחנו בבחינת הבסיס וישנם יותר מ-500,000 בודקים מוסמכים מסביב לעולם. בהנחה שכל הנבחנים קראו את תוכנית הלימודים הזו בכדי לעבור את הבחינה, זהו כנראה מסמך בדיקות התוכנה הנקרא ביותר מעולם!

עדכון גדול זה נעשה מתוך כבוד למורשת הזו וכדי לשפר את הערך המוסף ש-ISTQB יספק ל-500,000 האנשים הבאים בקהילת הבדיקות הבינלאומית.

בגרסה זו, כל יעדי הלימוד נערכו לפרטיהם, וליצור נעקבות ברורה מכל יעד לימוד אל התוכן של הפרק(ים) (ושאלות הבחינה) הקשורים ליעד הלימוד הזה, וליצור נעקבות ברורה מהתוכן של הפרק(ים) (ושאלות הבחינה) בחזרה אל יעד הלימוד הקשור. בנוסף, הזמן המוקצב לכל פרק הוגדר בדרך מציאותית יותר מאשר היה בגרסת 2011 של תוכנית הלימודים, על ידי שימוש בנוסחאות שנבחנו בתוכניות לימוד אחרות של ISTQB, מבוססות על ניתוח של יעדי לימוד שיש לכסות בכל פרק.

למרות שזוהי תוכנית לימודים לרמת הבסיס, שמציגה שיטות וטכניקות שעמדו במבחן הזמן, ערכנו שינויים כדי לחדש את הצגת החומר, בייחוד בהקשר של שיטות פיתוח התוכנה (לדוגמה, Scrum ופריסה מתמשכת (continuous deployment)) וטכנולוגיות (לדוגמה, האינטרנט של הדברים). עדכנו את התקנים אליהם התוכנית מתייחס כלהלן:

1. ISO/IEC/IEEE 29919 מחליף את IEEE Standard 829
2. ISO/IEC 25010 מחליף את ISO 9126
3. ISO/IEC 20246 מחליף את IEEE 1028

בנוסף, מאחר ומגוון תכניות הלימודים של ISTQB גדל מאוד במהלך העשור האחרון, הוספנו הפניות לתוכן קשור בתוכניות לימוד אחרות של ISTQB, במקומות הרלוונטיים, כמו גם ביצוע סקירה של ההתאמה בין כל תכניות הלימוד ובין מילון המונחים (glossary) של ISTQB. המטרה היא לעשות את הגרסה הזו קלה יותר לקריאה, להבנה, ללימוד ותרגום, תוך התמקדות בהגדלת התרומה המעשית של התוכנית ומציאת איזון בין ידע וכישורים.

לניתוח מפורט של השינויים שנעשו בגרסה זו, ראו ISTQB Certified Tester Foundation Level Overview 2018.