בודק מוסמך
תוכנית לימודים לרמת הבסיס

מהדורה עברית 2024.02

מבוססת על מהדורה 2023 (גרסה 4.0) באנגלית

הארגון הישראלי להסמכת
בודקי תוכנה

ITCB
ISRAELI TESTING CERTIFICATION BOARD

הארגון הבינלאומי להסמכת
בודקי תוכנה

ISTQB®
International Software
Testing Qualifications Board

Copyright Notice *הודעת זכויות יוצרים*

- Any individual or group of individuals may use this syllabus as the basis for articles and books, if the authors and the ISTQB® are acknowledged as the source and copyright owners of the syllabus.
- Any other use of this syllabus is prohibited without first obtaining the approval in writing of the ISTQB®.
- Any ISTQB®-recognized Member Board may translate this syllabus provided they reproduce the abovementioned Copyright Notice in the translated version of the syllabus.

זכויות יוצרים © 2024 עריכת המהדורה העברית 2024.01

מתרגמי ועורכי (ראה/י רשימת המתרגמים והעורכים בסעיף תודות בהמשך) המהדורה העברית מעבירים בזאת את הזכויות לארגון הישראלי להסמכת בודקי תוכנה (®ITCB). המתרגמים, העורכים ו-®ITCB הסכימו על תנאי השימוש במהדורה העברית כדלהלן:

1. כל אדם או חברת הדרכה יכולים להשתמש במהדורה עברית זו כבסיס לקורס הדרכה ו/או הכשרה מקצועית, וכל זאת בתנאי שעורכי המהדורה העברית, ®ITCB ו-®ISTQB מוזכרים בתור המקור ובעלי זכויות היוצרים של תוכנית הלימוד בשפה העברית. פרסום לקורס (ו/או הכשרה מקצועית) כזה יהיה רשאי להזכיר את תוכנית הלימוד ואת ארגוני ®ITCB ו-®ISTQB רק לאחר שהתקבלה הסמכה רשמית לכך מ-®ITCB.
2. כל אדם או קבוצה יכולים להשתמש במהדורה עברית זו כבסיס למאמרים, ספרים או דרכי חיבור אחרות של תוכן מקצועי, וכל זאת בתנאי שעורכי המהדורה העברית, ®ITCB ו-®ISTQB מוזכרים כמקור וכבעלי זכויות היוצרים שלה.

היסטוריית גרסאות

הסטוריית המהדורה בעברית

גרסה	תאריך	הערות
2018.01	1 פברואר 2019	גרסה ראשונה (2018) לתרגום העברי המלא של סילבוס הרמה הבסיסית 2018
2018.02	15 יולי 2019	תיקוני שגיאות
2018.03	1 ינואר 2020	התאמה לגרסה 3.1 של הסילבוס באנגלית
2018.04		תיקוני שגיאות
2024.01	1 אוקטובר 2024	גרסה ראשונה (2024) לתרגום העברי המלא של סילבוס הרמה הבסיסית גרסה 4.0
2024.02	11 נובמבר 2024	תיקוני שגיאות ושיפורי תרגום

Revision History

Version	Date	Remarks
CTFL v4.0	21.04.2023	CTFL v4.0 – General release version
CTFL v3.1.1	01.07.2021	CTFL v3.1.1 – Copyright and logo update
CTFL v3.1	11.11.2019	CTFL v3.1 – Maintenance release with minor updates
ISTQB 2018	27.04.2018	CTFL v3.0 – Candidate general release version
ISTQB 2011	1.04.2011	CTFL Syllabus Maintenance Release
ISTQB 2010	30.03.2010	CTFL Syllabus Maintenance Release
ISTQB 2007	01.05.2007	CTFL Syllabus Maintenance Release
ISTQB 2005	01.07.2005	Certified Tester Foundation Level Syllabus v1.0

תוכן העניינים

0	הקדמה לתוכנית לימודים זו	10
0.1	מטרת המסמך	10
0.2	רמת הבסיס של בודק מוסמך בבדיקות תוכנה	10
0.3	מסלול קריירה לבודקים	10
0.4	תוצאות עסקיות (Business Outcomes)	11
0.5	מטרות למידה הניתנות לבחינה ורמת ידע הקוגניטיבי	11
0.6	בחינת ההסמכה לרמת הבסיס	11
0.7	הסמכת ספקי הדרכה	12
0.8	טיפול בתקנים	12
0.9	להישאר מעודכן	12
0.10	רמת הפירוט	12
0.11	איך תוכנית הלימודים הזו מאורגנת	13
1	יסודות בדיקות התוכנה – 180 דקות	14
1.1	בדיקות, מהן?	15
1.1.1	מטרות הבדיקה	15
1.1.2	בדיקות וניפוי באגים (debugging)	16
1.2	מדוע הבדיקות נחוצות?	16
1.2.1	תרומת הבדיקות להצלחה	16
1.2.2	בדיקות והבטחת איכות (Quality Assurance – QA)	17
1.2.3	שגיאות, פגמים, כשלים ושורש הבעיות	17
1.3	עקרונות הבדיקה	18
1.4	פעילויות בדיקה, בּוֹדְקָה (Testware) ובעלי תפקידים בבדיקות	19
1.4.1	פעילויות ומשימות בדיקה	19
1.4.2	תהליך הבדיקות בהתאמה להֶקְשר	20
1.4.3	בּוֹדְקָה Testware	20
1.4.4	נֶעֱקְבוּת (Traceability) בין בסיס הבדיקות לבודקה	21
1.4.5	תפקידים בבדיקות	22
1.5	מיומנויות חיוניות ושיטות עבודה טובות בבדיקות	22
1.5.1	מיומנויות כלליות הנדרשות לבדיקה	22
1.5.2	גישת הצוות השלם (whole team approach)	23
1.5.3	עצמאות הבדיקות	23
2	בדיקות לאורך מחזור חיי פיתוח תוכנה – 130 דקות	25
2.1	בדיקות בהקשר של מחזור חיים של פיתוח תוכנה	26
2.1.1	השפעת מחזור החיים של פיתוח תוכנה על בדיקות	26
2.1.2	מחזור חיי פיתוח תוכנה ושיטות בדיקה יעילות ומוכחות	26
2.1.3	בדיקות כמנוע לפיתוח תוכנה	27
2.1.4	DevOps ובדיקות	27
2.1.5	גישת הזזה שמאלה (Shift – Left)	28
2.1.6	פגישות רטרוספקטיבה – (מפגשי ראיה כוללת) ושיפור תהליכים	29
2.2	רמות הבדיקה וסוגי הבדיקה	29
2.2.1	רמות הבדיקה	29

2.2.2	סוגי בדיקות	30
2.2.3	בדיקות אישור ובדיקות נסיגה (רגרסיה)	31
2.3	בדיקות תחזוקה - Maintenance Testing	32
3	בדיקות סטטיות – 80 דקות	33
3.1	יסודות הבדיקות הסטטיות	34
3.1.1	תוצרי עבודה שניתן לבדוק באמצעות בדיקות סטטיות	34
3.1.2	הערך של בדיקות סטטיות	34
3.1.3	הבדלים בין בדיקות סטטיות ובדיקות דינמיות	35
3.2	תהליך משוב וסקירה	36
3.2.1	יתרונות המשוב המוקדם והתכוף מבעלי העניין	36
3.2.2	פעילויות בתהליך הסקירה	36
3.2.3	תפקידים ותחומי אחריות בסקירות	37
3.2.4	סוגי סקירות	37
3.2.5	גורמי הצלחה לסקירות	38
4	ניתוח ועיצוב בדיקות – 390 דקות	39
4.1	טכניקות בדיקה - סקירה	40
4.2	טכניקות בדיקה מסוג קופסה שחורה	40
4.2.1	מחלקות שקילות	40
4.2.2	ניתוח ערכי גבול	41
4.2.3	בדיקות טבלת החלטה	42
4.2.4	בדיקות הֶחְלֵף מצבים	43
4.3	טכניקות בדיקה מסוג קופסה לבנה	44
4.3.1	בדיקות משפטים (פקודות) וכיסוי משפטים (פקודות)	44
4.3.2	בדיקות ענפים וכיסוי ענפים (Branch Coverage)	44
4.3.3	הערך של בדיקות קופסה לבנה	45
4.4	טכניקות בדיקה מבוססות ניסיון	45
4.4.1	ניחוש שגיאות	45
4.4.2	בדיקות חוקרות	46
4.4.3	בדיקות מבוססות על רשימות בדיקה (תיוג)	46
4.5	גישות בדיקה מבוססות שיתוף פעולה	46
4.5.1	כתיבה משותפת של סיפור המשתמש	47
4.5.2	קריטריוני קבלה	47
4.5.3	פיתוח מונחה בדיקות קבלה (ATDD)	48
5	ניהול פעילות הבדיקות – 335 דקות	49
5.1	תכנון בדיקות	50
5.1.1	המטרה והתוכן של תוכנית בדיקות	50
5.1.2	תרומתו של הבודק לתכנון איטרציה ולתכנון שחרור הגרסה	51
5.1.3	קריטריוני כניסה וקריטריוני יציאה	51
5.1.4	טכניקות אומדן בדיקות	51
5.1.5	תעדוף מקרי בדיקה	52
5.1.6	פירמידת בדיקות	53
5.1.7	בדיקת רביעים	53
5.2	ניהול סיכונים	54
5.2.1	הגדרות סיכונים ומאפייני סיכון	54

5.2.2	סיכוני פרויקט וסיכוני מוצר	54
5.2.3	ניתוח סיכוני מוצר	55
5.2.4	בקרת סיכוני מוצר	55
5.3	ניטור בדיקות, בקרת בדיקות והשלמת בדיקות	56
5.3.1	מדדים בשימוש הבדיקות	56
5.3.2	מטרות, תוכן וקהל היעד של דוחות הבדיקות	57
5.3.3	מסירת סטטוס הבדיקות	58
5.4	ניהול תצורה	58
5.5	ניהול פגמים	59
6	כלי בדיקה – 20 דקות	60
6.1	כלים תומכי בדיקות	61
6.2	יתרונות וסיכונים באוטומציה של בדיקות	61
7	הפניות / סימוכין	63
8	נספח א – יעדי לימוד / רמות ידע קוגניטיביות	66
9	נספח ב – מטריצת נֶעֱקָבוּת: התוצאות העסקיות (BOs) למול יעדי למידה (LOs)	68
10	נספח ג – הערות פרסום למהדורה (Release Notes)	74
11	נספח ד – הבהרות מיוחדות (עבור המהדורה בעברית בלבד)	76

תודות

עבור הגרסה בעברית

לצוות התרגום והעריכה של גרסה 4.0: ירון צוברי, מיכל טל, יונית אלבז, דני אלמוג, ניצן גולדנברג וגיל דנון. לצוות ההגהה על התרגום: אסתר צבר, מיכל טל, ירון צוברי ומיכאל שטאל.

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.

אלישבע הרשלר, שביצעה את התרגום הבסיסי של מהדורת 2011 במסירות רבה; ובנוסף, לדר' אבי עופר על תרגום המונחים לעברית, וכן לצוות התרגום ועדכון המונחים: דליה פילמוס, יעל פלמינגר, דני אלמוג וירון צוברי.

עבור הגרסה המקורית באנגלית

This document was formally released by the General Assembly of the ISTQB® on 21 April 2023

It was produced by a team from the ISTQB joint Foundation Level & Agile Working Groups: Laura Albert, Renzo Cerquozzi (vice chair), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (co-chair), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (co-chair), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (vice chair), Eshraka Zakaria.

The team thanks Stuart Reid, Patricia McQuaid and Leanne Howard for their technical review and the review team and the Member Boards for their suggestions and input.

The following persons participated in the reviewing, commenting and balloting of this syllabus: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo and Zsolt Hargitai.

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

ISTQB 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 Member Boards for the suggestions for the current version of the syllabus. Certified Tester Foundation Level

v4.0 Page 9 of 74 2023-04-21 © International Software Testing Qualifications Board

ISTQB 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 Member Boards for their suggestions.

ISTQB 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 Member Boards for their suggestions.

ISTQB 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. The core team thanks the review team and all Member 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 בזאת מידי הארגון הבינלאומי להסמכת בודקי תוכנה (ISTQB®)

1. למועצות שחברות בארגון (ISTQB® Member Boards), לשם תרגום של תוכנית הלימוד לשפתם המקומית והסמכה של ספקי הדרכה (accreditation). המועצות החברות רשאיות להתאים את תוכנית הלימוד לצרכי השפה המיוחדים שלהם ולשנות את ההפניות כך שיתאימו גם לפרסומים המקומיים שלהם.
2. לגופי בחינה והסמכה, לשם הפקת שאלות בחינה בשפתם המקומית, מותאמות ליעדי הלמידה (Learning Objectives) של תוכנית הלימוד.
3. לספקי הדרכה, לשם הפקת מערכי הדרכה ובחירת שיטות הוראה המתאימות.
4. למועמדים לבחינת ההסמכה, לשם הכנה לבחינת ההסמכה (בין אם במסגרת קורס או הכשרה ובין אם בלימוד עצמאי).
5. לקהילה הבינלאומית של מהנדסי התוכנה והמערכת, לשם קידום מקצוע בדיקות התוכנה והמערכת, וכבסיס לספרים ומאמרים.

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 ונערכו כך למניעת סרבול, אבל הם מופנים במידה שווה לנשים ולגברים, לבודקות ולבודקים.

0.2 רמת הבסיס של בודק מוסמך בבדיקות תוכנה

ההסמכה ברמת הבסיס מיועדת לכל מי שעוסק בבדיקות תוכנה. זה כולל אנשים בתפקידים כמו בודקים, מנתחי בדיקות, מהנדסי בדיקות, יועצי בדיקה, מנהלי בדיקות, מפתחי תוכנה וחברי צוות פיתוח. הסמכה זו ברמת הבסיס מתאימה גם לכל מי שמעוניין בהבנה הבסיסית של בדיקות תוכנה, כגון מנהלי פרויקטים, מנהלי איכות, בעלי מוצרים, מנהלי פיתוח תוכנה, אנליסטים עסקיים, מנהלי IT ויועצי ניהול. בעלי תעודת ההסמכה יוכלו להמשיך ולרכוש מיומנויות ויכולות בבדיקות תוכנה ברמה גבוהה יותר.

0.3 מסלול קריירה לבודקים

תוכנית ISTQB® מספקת תמיכה לאנשי מקצוע בכל שלבי הקריירה שלהם, ומציעה ידע רוחבי ולעומק. אלו שהשיגו את ההסמכה ברמת הבסיס ISTQB® Foundation עשויים להתעניין גם ברמות הליבה המתקדמות (מנתח בדיקות, מנתח טכני לבדיקות ומנהל בדיקות) ולאחר מכן ברמת מומחה (ניהול בדיקות או שיפור תהליך הבדיקות). כל מי שמבקש לפתח מיומנויות בשיטות בדיקה בסביבה אג'ילית יכול לשקול את ההסמכה של בודק טכני בסביבה אג'ילית או מנהיגות בבדיקות בסביבה אג'ילית לפי אמות מידה (ATLaS).

זרם ההתמחות הייעודי ה-Specialist מציע צלילה עמוקה לתוך תחומים שיש להם גישות לבדיקה ופעילויות בדיקה ספציפיות (למשל, אוטומציה של בדיקות, בדיקות למערכות AI, בדיקות מבוססות מודלים, בדיקת אפליקציות במכשירים ניידים), הקשורים לאזורי בדיקה ספציפיים (למשל, בדיקות ביצועים, בדיקות שמישות, בדיקות קבלה, בדיקות אבטחה), או ידע בבדיקת אשכולות עבור תחומים

מסוימים בתעשייה (למשל, עבור תעשיית הרכב או משחקים). אנא בקרו בכתובת www.istqb.org על מנת לקבל את המידע העדכני ביותר על מגוון התוכניות ללימוד והסמכות בבדיקות של ISTQB®.

0.4 תוצאות עסקיות (Business Outcomes)

סעיף זה מפרט את 14 התוצאות העסקיות הצפויות מאדם שהשיג את ההסמכה ברמת הבסיס.

בודק מוסמך ברמת הבסיס יכול:

FL-BO1 לְהַבִּין מהי בדיקה ולמה היא מועילה.

FL-BO2 לְהַבִּין מושגי יסוד של בדיקות תוכנה.

FL-BO3 לְזַהוֹת את גישת הבדיקות והפעילויות שיש ליישם בהתאם להקשר של הבדיקות.

FL-BO4 להעריך ולשפר את איכות התיעוד.

FL-BO5 לְהַגדִיל את האפקטיביות והיעילות של הבדיקות.

FL-BO6 ליישר/להתאים את תהליך הבדיקות עם מחזור החיים של פיתוח התוכנה.

FL-BO7 להבין את עקרונות ניהול בדיקות.

FL-BO8 לְכתוֹב ולתקשר דוחות פגמים באופן ברור ומובן.

FL-BO9 לְהַבִּין את הגורמים המשפיעים על סדרי העדיפויות והמאמצים הקשורים לבדיקות.

FL-BO10 לעבוד כחלק מצוות חוצה פעילויות צולבות.

FL-BO11 לְהבחין בסיכונים וביתרונות הקשורים לאוטומציה של בדיקות.

FL-BO12 לזהות מיומנויות חיוניות הנדרשות עבור הבודק.

FL-BO13 לְהַבִּין את השפעת הסיכון על הבדיקות.

FL-BO14 לדווח ביעילות על התקדמות הבדיקות ואיכותן.

0.5 מטרות למידה הניתנות לבחינה ורמת ידע הקוגניטיבי

מטרות הלמידה תומכות בתוצאות העסקיות ומשמשות ליצירת הבחינות ברמת ה- Certified Tester Foundation Level. ככלל, כל התכנים של פרקים 1-6 בתוכנית לימוד זו ניתנים לבחינה ברמה K1. כלומר, ניתן לבקש מהמועמד להכיר, לזכור או להיזכר במילת מפתח או מושג המוזכרים בכל אחד מששת הפרקים. רמות יעדי הלמידה הספציפיים מוצגות בתחילת כל פרק, ומסווגות כדלקמן:

- K1: לזכור.
- K2: להבין.
- K3: ליישם.

פרטים נוספים ודוגמאות של יעדי למידה ניתנים בנספח א'.

כל המונחים הרשומים כמילות מפתח וממוקמים ממש מתחת לכל ראש פרק יש לזכרם (K1), גם אם הם לא צוינו במפורש ביעדי הלמידה.

0.6 בחינת ההסמכה לרמת הבסיס

בחינת התעודה ברמת הבסיס מבוססת על תוכנית לימוד זו. תשובות לשאלות בחינה עשויות לדרוש שימוש בחומר המבוסס על יותר מסעיף אחד בתוכנית לימוד זו. כל חלקי תוכנית הלימוד ניתנים לבחינה, למעט המבוא והנספחים. תקנים וספרים נכללים כאסמכתא (פרק 7), אך תוכנם אינו ניתן לבחינה, מעבר למה שמסוכם בתוכנית הלימוד עצמה מתקנים וספרים כאלה.

ניתן לעיין במסמך המבנים והכללים של הבחינות ברמת הבסיס (Foundation Level Examination Structures and Rules).

הבחינה בנויה במתכונת של שאלון רב-ברירתי (בחינה אמריקאית), וכוללת 40 שאלות. כדי לעבור את הבחינה, יש לענות נכון על 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 מהווה תנאי מוקדם לצורך הזכאות לגשת לבחינה.

0.7 הסמכת ספקי הדרכה

מועצה החברה ב- ISTQB® (בישראל, ארגון ITCB®) רשאית להסמיך ספקי הדרכה והכשרה שחומר הקורס שלהם עומד בתנאי תוכנית הלימוד הזו. ספקי ההדרכה/ההכשרה צריכים לקבל הנחיות להסמכה מהמועצה החברה (בישראל מארגון ITCB®) או מהגוף שמבצע את ההסמכה (accreditation). קורס מוסמך הוא כזה שתואם את תוכנית הלימודים הזו, ורשאי להגיש את הלומדים בו לבחינת ההסמכה של ISTQB® כחלק מהקורס. הנחיות ההסמכה לתוכנית הלימוד הזו עוקבות אחר הנחיות ההסמכה הכלליות שפורסמו על ידי קבוצת העבודה לניהול תהליכים ותאימות של ISTQB®.

0.8 טיפול בתקנים

ישנם תקנים המוזכרים בתוכנית הלימוד לרמת הבסיס (למשל, תקני IEEE או ISO). הפניות אלו מספקות מסגרת (כדוגמת ההפניות ל-ISO 25010 לגבי מאפייני איכות) או לספק נוסף של מקור מידע באם ירצה הקורא בכך. מסמכי התקנים אינם מיועדים לבחינה.

עיין בפרק 7 למידע נוסף על התקנים.

0.9 להישאר מעודכן

תעשיית התוכנה משתנה במהירות. כדי להתמודד עם שינויים אלו וכדי לספק לבעלי העניין גישה למידע רלוונטי ועדכני, קבוצות העבודה של ISTQB® יצרו קישורים באתר www.istqb.org, המתייחסים לתיעוד תומך ולשינויים בתקנים. מידע זה אינו ניתן לבחינה במסגרת תוכנית הלימוד לרמת הבסיס.

0.10 רמת הפירוט

רמת הפירוט של תוכנית הלימודים הזו מאפשרת אחידות בינלאומית 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 הקוגניטיבית המבוקשת.
- תיאור של מושגי יסוד, כולל מקורות מהספרות המקצועית המקובלת ומתקנים בינלאומיים.

תוכן תוכנית הלימודים אינו תיאור של כל גוף הידע (body of knowledge) בבדיקות תוכנה, אלא משקף רמת פירוט שיש להכיל בקורסי הדרכה לרמת הבסיס. התוכנית מתמקדת במושגים וטכניקות של בדיקות שניתן ליישם בכל פרויקט תוכנה, ללא תלות במחזור החיים של הפיתוח (SDLC) שמופעל.

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).

0.11 איך תוכנית הלימודים הזו מאורגנת

קיימים שישה פרקים עם תוכן שניתן לבחון עליו בבחינת ההסמכה. הכותרת ברמה העליונה של כל פרק מציינת בין השאר את זמן ההכשרה המומלץ לפרק. תזמון לא מסופק מרמות שמתחת לרמת הפרק. עבור קורסי הכשרה מוסמכים, הסילבוס דורש מינימום של 1135 דקות (18 שעות ו-55 דקות) של הדרכה, המחולקת על פני ששת הפרקים כדלקמן:

- **פרק 1: יסודות בדיקות התוכנה (180 דקות)**
  - התלמיד לומד את העקרונות הבסיסיים הקשורים לבדיקות, את הסיבות לביצוע הבדיקות ומהן מטרות הבדיקות.
  - התלמיד מבין את תהליך הבדיקות, פעילויות הבדיקה העיקריות, והבודקה.
  - התלמיד מבין את המיומנויות החיוניות הנדרשות לבדיקות.
- **פרק 2: בדיקות לאורך מחזור חיי פיתוח תוכנה (130 דקות)**
  - התלמיד לומד כיצד בדיקה משולבת בגישות פיתוח שונות.
  - התלמיד לומד את המושגים הקשורים לגישות בדיקות תחילה (test first), כמו גם DevOps.
  - התלמיד לומד על רמות הבדיקה השונות, סוגי הבדיקות ובדיקות התחזוקה.
- **פרק 3: בדיקות סטטיות (80 דקות)**
  - התלמיד לומד על היסודות הנדרשים לביצוע בדיקות סטטיות, תהליך המשוב והסקירה.
- **פרק 4: ניתוח ועיצוב בדיקות (390 דקות)**
  - התלמיד לומד כיצד ליישם טכניקות בדיקות מבוססות-קופסה שחורה, לבנה ומתנסה ביצירת מקרי בדיקה מתוצרי עבודה שונים של תוכנה.
  - התלמיד לומד על גישת הבדיקה המבוססת על שיתוף פעולה.
- **פרק 5: ניהול פעילות הבדיקות (335 דקות)**
  - התלמיד לומד כיצד לתכנן בדיקות באופן כללי, וכיצד להעריך מאמץ בדיקות.
  - התלמיד לומד כיצד סיכונים יכולים להשפיע על היקף הבדיקות.
  - התלמיד לומד כיצד לנטר ולשלוט בפעילויות הבדיקה.
  - התלמיד לומד כיצד ניהול תצורה תומך בבדיקות.
  - התלמיד לומד כיצד לדווח על פגמים בצורה ברורה ומובנת.
- **פרק 6: כלי בדיקה (20 דקות)**
  - התלמיד לומד לסווג כלים ולהבין את הסיכונים והיתרונות באוטומציה של הבדיקות.

1 יסודות בדיקות התוכנה — 180 דקות

מושגים

איכות (quality), אימות (verification), בדיקות (Testing), בודקה[1](testware) , ביצוע בדיקות (test execution), בסיס בדיקות (test basis), בקרת בדיקות (test control), הבטחת איכות (quality assurance), סיום בדיקות (test completion), יישום בדיקות (test implementation), כיסוי (coverage), כשל (failure), מושא הבדיקות (test object), מטרת הבדיקות (test objective), מקרה בדיקה (test case), נוהל/הליך בדיקה (test procedure), ניטור הבדיקות (test monitoring), ניפוי באגים (debugging), נתוני בדיקות (test data), ניתוח בדיקות (test analysis), עיצוב בדיקות (test design), פגם (defect), שגיאה (error), שורש הבעיה (root cause), תוצאת בדיקה (test result), תיקוף (validation), תכנון בדיקות (test planning), תנאי בדיקה (test condition)

מטרות למידה לפרק 1

1.1 בדיקות, מהן?

FL-1.1.1 (K1) זהה מטרות בדיקה אופייניות

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 (K1) הסבר את הקשר בין בדיקות והבטחת איכות

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

1.3 עקרונות הבדיקות

FL-1.3.1 (K2) הסבר את שבעת עקרונות הבדיקה

1.4 פעילויות בדיקות, בודקה ותפקידים בבדיקות

FL-1.4.1 (K2) סכם את הפעילויות והמשימות השונות של הבדיקות

FL-1.4.2 (K2) הסבר את השפעת ההקשר של התוכנה על תהליך הבדיקות

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

FL-1.4.4 (K2) הסבר את החשיבות בשמירה על עקיבות

FL-1.4.5 (K2) השווה בין התפקידים השונים בבדיקות

1.5 מיומנויות חשובות ושיטות עבודה טובות בבדיקות

FL-1.5.1 (K2) תן דוגמאות למיומנויות הכלליות הנדרשות לבדיקות

FL-1.5.2 (K1) הסבר את היתרונות של גישת "צוות שלם"

FL-1.5.3 (K2) הבחן בין היתרונות והחסרונות של עצמאות הבדיקות

[1] הערת עורך - מכלול מרכיבי הבדיקה. ראה/י מילון מונחים

1.1 בדיקות, מהן?

מערכות התוכנה הן חלק בלתי נפרד מחיינו. רוב האנשים נתקלו בתוכנות שלא פעלו כצפוי. תוכנה שאינה פועלת כראוי יכולה לגרום לבעיות רבות, כולל אובדן כסף, זמן או מוניטין עסקי, ובמקרים קיצוניים, אף פציעה או מוות. בדיקות תוכנה מעריכות את איכות התוכנה ומסייעות להפחית את הסיכון לכשלים בתוכנה במהלך תפקודה.

בדיקות תוכנה היא סדרת פעילויות לגילוי פגמים ולהערכת איכות רכיבי תוכנה. רכיב תוכנה כזה – כאשר הוא נבדק – נקרא "אוביקט הבדיקה" (Test Object). תפיסה שגויה נפוצה לגבי בדיקות היא שבדיקות כוללות רק ביצוע בדיקות (כלומר, הפעלת התוכנה ובדיקת תוצאות הבדיקה). עם זאת, בדיקות תוכנה כוללות גם פעילויות נוספות וחייבות להיות מתואמות עם מחזור חיי הפיתוח של התוכנה (ראה פרק 2).

עוד תפיסה שגויה נפוצה בנוגע לבדיקות היא שהן מתמקדות לחלוטין באימות אוביקט הבדיקה. בעוד שבדיקות אכן כוללות אימות (verification), כלומר, בודקות האם המערכת עומדת בדרישות הספציפיות; הן כוללות בנוסף גם תיקוף (validation), שמשמעותו היא לבדוק האם המערכת – בסביבתה התפעולית – עומדת בצרכי המשתמשים ובעלי עניין אחרים.

בדיקה יכולה להיות דינמית או סטטית. בדיקה דינמית כוללת את הפעלת התוכנה, בעוד שבבדיקה סטטית אין הפעלת התוכנה. בדיקה סטטית כוללת סקירות (reviews) (ראה פרק 3) וניתוח סטטי. בדיקה דינמית כוללת שימוש בסוגים שונים של טכניקות וגישות להפקת מקרי בדיקה (ראה פרק 4).

בדיקה היא לא רק פעילות טכנית. היא צריכה גם להיות מתוכננת, מנוהלת, מוערכת, מנוטרת ומבוקרת כראוי (ראה פרק 5).

בודקים משתמשים בכלים (ראה פרק 6), אך חשוב לזכור שבדיקה היא בעיקר פעילות אינטלקטואלית, הדורשת מהבודקים מומחיות, שימוש בכישורים אנליטיים, חשיבה קריטית וחשיבה מערכתית (Myers 2011, Roman 2018).

התקן ISO/IEC/IEEE 29119-1 מספק מידע נוסף אודות תפיסות בבדיקות תוכנה.

1.1.1 מטרות הבדיקה

מטרות הבדיקה הנפוצות הן:

- הערכת תוצרי עבודה כגון דרישות, סיפורי משתמש, עיצובים וקוד
- גרימת כשלים ומציאת פגמים
- הבטחת כיסוי נדרש של אובייקט הבדיקה
- הפחתת רמת הסיכון של איכות תוכנה לא מספקת
- אימות עמידה בדרישות הספציפיות
- אימות כי אובייקט הבדיקה עומד בדרישות החוזיות, החוקיות והתקניות
- מתן מידע לבעלי העניין על מנת לאפשר להם לקבל החלטות מושכלות
- בניית אמון באיכות של אובייקט הבדיקה
- אימות האם אובייקט הבדיקה הינו שלם ופועל כפי שבעלי העניין מצפים

מטרות הבדיקה יכולות להשתנות, לפי ההקשר, הכולל את תוצר העבודה הנבדק, רמת הבדיקה, סיכונים, מחזור חיי פיתוח התוכנה (Software Development Life Cycle - SDLC) המיושם, וגורמים הקשורים להקשר העסקי, לדוגמה: המבנה הארגוני, שיקולים תחרותיים, או זמן השחרור של המוצר לשוק (time to market).

1.1.2 בדיקות וניפוי באגים (debugging)

בדיקות וניפוי באגים הן פעילויות נפרדות. בדיקות יכולות למצוא כשלים שנגרמים על ידי פגמים בתוכנה (בדיקות דינמיות) או לגלות ישירות פגמים באובייקט הבדיקה (בדיקות סטטיות). כשבדיקה דינמית (ראה פרק 4) מגלה כשל, ניפוי הבאגים כולל את מציאת הסיבות לכשל (לפגמים), ניתוח הסיבות הללו והסרתן. התהליך הנפוץ לניפוי באגים במקרה שכזה כולל:

- שחזור של הכשל
- אבחון (מציאת הגורם לבעיה)
- תיקון הסיבה[2]

בדיקות אישור (confirmation testing) לאחר התיקון בודקת האם התיקונים פתרו את הבעיה. עדיף שהאדם שביצע את הבדיקה הראשונית יהיה זה שיבצע את בדיקות האימות. בדיקות רגרסיה יכולות גם להתבצע, על מנת לבדוק האם התיקונים גורמים לכשלים בחלקים אחרים של אובייקט הבדיקה (ראה פרק 2.2.3 למידע נוסף על בדיקות אישור ובדיקות רגרסיה).

כאשר בדיקה סטטית מזהה פגם (defect), ניפוי הבאגים מתמקד בהסרתו. אין צורך בשחזור או אבחון, מאחר ובדיקה סטטית מוצאת פגמים ישירות[3] ואינה יכולה לגרום לכשלים (ראה פרק 3).

1.2 מדוע הבדיקות נחוצות?

בדיקות, כפעילות של בקרת איכות, מסייעות בהשגת המטרות המוסכמות, במסגרת התכולה, הזמן, האיכות ומגבלות התקציב שנקבעו. התרומה של הבדיקות להצלחה אינה צריכה להיות מוגבלת לפעילויות צוות הבדיקות. כל בעל עניין יכול לעזור להצלחת הפרויקט בעזרת יכולות הבדיקה שלו. בדיקות רכיבים, מערכות ומסמכים הקשורים לתוכנה עוזרים לזהות פגמים בתוכנה.

1.2.1 תרומת הבדיקות להצלחה

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

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

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

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

[2] הערת עורך – תיקון הסיבה לכשל

[3] הערת עורך - בקוד ו/או במסמכים

1.2.2 בדיקות והבטחת איכות (Quality Assurance - QA)

אנשים נוטים להשתמש במושג הבטחת איכות (QA) כשהם מדברים על בדיקות (Testing), אך בדיקות ו-QA אינן זהות. בדיקות הן סוג של בקרת איכות (Quality Control - QC).

QC (בקרת איכות) היא גישה שמתמקדת במוצר, היא גישה מתקנת המתמקדת בפעילויות התומכות בהשגת רמות האיכות הנדרשות. בדיקות הן סוג מרכזי של בקרת איכות, בעוד שסוגים אחרים כוללים שיטות רשמיות (בדיקות מודלים והוכחת נכונות), סימולציות ובניית אב טיפוס.

QA (הבטחת איכות) היא גישה שמתמקדת בתהליך העבודה, היא גישה מונעת המתמקדת ביישום ושיפור תהליכי עבודה. היא עובדת על פי העקרון שאם תהליך העבודה הינו טוב ומיושם כראוי, אזי הוא יפיק מוצר טוב. QA מתייחס לתהליכי הפיתוח והבדיקות, והוא באחריות כל אנשי הפרויקט.

תוצאות הבדיקות משמשות את ה-QA וגם את ה-QC. ב-QC הן משמשות לתיקון פגמים, בעוד שב-QA הן מספקות משוב על טיבם של תהליכי הפיתוח והבדיקות.

1.2.3 שגיאות, פגמים, כשלים ושורש הבעיות

בני אדם מבצעים שגיאות (טעויות, mistakes), אשר גורמות לפגמים (תקלות, באגים, faults), היכולים לגרום לכשלים (failures). בני אדם מבצעים שגיאות מסיבות שונות, כגון לחץ זמן, מורכבות תוצרי עבודה, תהליכים, תשתיות או יחסי גומלין, או פשוט עקב עייפות או מחסור בהכשרה מתאימה.

פגמים ניתן למצוא במסמכים, כגון מפרטי דרישות או תסריטי בדיקה, בקוד מקור או בתוך פריט תומך כגון קובץ בניית גרסת המוצר (build file). פגמים בפריטי עבודה שנוצרו בשלבים הראשונים של מחזור החיים של הפיתוח (SDLC), אם הם אינם מזוהים, הם יכולים להוביל לפריטי עבודה פגומים בשלב מאוחר יותר במחזור החיים של פיתוח התוכנה. אם קיים פגם בקוד שרץ, המערכת עשויה לא לבצע את מה שהיא אמורה לעשות, או לבצע משהו שהיא לא אמורה לעשות, ובכך לגרום לכשל. חלק מהפגמים תמיד יביאו לכשל אם הם רצים (מבוצעים), בעוד שפגמים אחרים יביאו לכשל בתנאים מסוימים בלבד, וכמה פגמים לעולם לא יביאו לכשל.

שגיאות ופגמים אינם הסיבה היחידה לכשלים. כשלים יכולים גם להיגרם כתוצאה מתנאים סביבתיים, למשל, כאשר קרינה או שדה אלקטרומגנטי גורמים לפגמים בקושחה (Firmware - תוכנה המשובצת בהתקן חומרה).

שורש הבעיה הינה הסיבה הבסיסית להתרחשות הבעיה (לדוגמה, מצב המוביל לשגיאה). שורשי הבעיה מזוהים באמצעות ניתוח שורש הבעיה (root cause analysis), שלרוב מתבצע כאשר נמצא כשל או זוהה פגם. ההנחה היא שניתן למנוע או להפחית את התקריות של כשלים או פגמים דומים נוספים על ידי טיפול בשורש הבעיה, למשל על ידי הסרתו.

1.3 עקרונות הבדיקות

לאורך השנים הוצעו מספר עקרונות לבדיקות המהווים הנחיות כלליות החלות על כל הבדיקות. תכנית לימודים זו מתארת שבעה עקרונות כאלה.

1. הבדיקות מצביעות על נוכחות של פגמים, לא על היעדרם (**Testing shows the presence, not the absence of defects**).
הבדיקות יכולות להראות שפגמים נמצאים באובייקט הבדיקה, אך אינן יכולות להוכיח שאין פגמים (Buxton 1970). הבדיקות מפחיתות את הסיכוי שיישארו פגמים חבויים באובייקט הבדיקה, אך גם אם לא נמצאו פגמים, הבדיקות אינן יכולות להוכיח את נכונות (correctness) אובייקט הבדיקה.

2. בלתי אפשרי לבצע בדיקות ממצות (**Exhaustive testing is impossible**).
בדיקה של הכל (כל צירופי הקלט והתנאים המוקדמים) אינה ברת-ביצוע, למעט במקרים טריוויאלים (פשוטים) (Manna 1978). במקום לנסות לבצע בדיקות ממצות, יש להשתמש בטכניקות בדיקה (ראה פרק 4), תיעדוף מקרי בדיקה (ראה סעיף 5.1.5) ושימוש בבדיקות מונחות סיכונים (ראה סעיף 5.2) על מנת למקד את מאמצי הבדיקות.

3. בדיקות מוקדמות חוסכות זמן וכסף (**Early testing saves time and money**). פגמים המוסרים בשלבים מוקדמים בתהליך הפיתוח לא יגרמו לפגמים נוספים בתוצרי העבודה המפותחים בהמשך תהליך הפיתוח. עלות האיכות תופחת מאחר ופחות כשלים יתרחשו מאוחר יותר בתהליך הפיתוח (Boehm 1981). על מנת למצוא פגמים בשלב מוקדם, יש להתחיל בבדיקות סטטיות (ראה פרק 3) ובדיקות דינמיות (ראה פרק 4) מוקדם ככל הניתן.

4. התקבצות פגמים (**Defects cluster together**). מספר קטן של רכיבי מערכת מכילים בדרך כלל את רוב הפגמים שנמצאו, או אחראיים לרוב הכשלים התפעוליים (Enders 1975). תופעה זו הינה הדגמה של עקרון פרטו (Pareto principle). מקבץ צפוי של פגמים וקבוצות פגמים שנצפו במהלך הבדיקות או במהלך פעולת המערכת, הינם קלט חשוב עבור בדיקות מבוססות סיכונים (ראה סעיף 5.2).

5. הבדיקות נשחקות (**Tests wear out**). אם אותן בדיקות מורצות פעמים רבות, רמת יעילותן בזיהוי פגמים חדשים יורדת בהדרגה (Beizer 1990). כדי להתגבר על האפקט הזה, יתכן שיהיה צורך לשנות בדיקות קיימות ונתוני בדיקות, ואף לכתוב בדיקות חדשות. עם זאת, במקרים מסוימים, חזרה על אותן בדיקות שוב ושוב עשויה להועיל, לדוגמה בבדיקות רגרסיה אוטומטיות (ראה סעיף 2.2.3)[4].

6. בדיקות הן תלויות הקשר (**Testing is context dependent**). אין גישה ישימה אוניברסלית אחת לבדיקות. בדיקות נעשות בצורות שונות בהקשרים שונים (Kaner 2011).

7. העדר שגיאות הוא סברה מוטעית (**Absence-of-defects fallacy**). זוהי סברה מוטעית (כלומר, תפיסה שגויה) לצפות שאימות התוכנה יבטיח את הצלחת המערכת. בדיקות יסודיות של כל הדרישות שהוגדרו ותיקון כל הפגמים שנמצאו יכולים עדיין לייצר מערכת שאינה עומדת בצרכי ובציפיות המשתמשים, שאינה תורמת להשגת המטרות העסקיות של הלקוח ושהיא נחותה ממערכות מתחרות אחרות. לכן, בנוסף לאימות (verification), יש לבצע גם ווידוא (validation). (מקור Boehm 1981).

[4] הערת עורך - עקרון זו ידוע גם כפרדוקס ההדברה

1.4 פעילויות בדיקה, בודְקָה (Testware) ובעלי תפקידים בבדיקות

בדיקות הינן תלויות הקשר, אך ברמה גבוהה, ישנן קבוצות של פעילויות נפוצות אשר בלעדיהן ישנה סבירות נמוכה יותר שהבדיקות תשגנה את מטרותיהן. קבוצות אלה של פעילויות בדיקה מהוות את תהליך הבדיקות. ניתן להתאים את תהליך הבדיקות למצב מסוים בהתבסס על גורמים שונים. ההחלטות לגבי אילו פעילויות בדיקה נכללות בתהליך הבדיקות, כיצד הן מיושמות, ומתי הן קורות, נקבעות בדרך כלל כחלק מתכנון הבדיקות (test planning) למצב הספציפי (ראה סעיף 5.1).

הסעיפים הבאים מתארים את ההיבטים הכלליים של תהליך הבדיקות לפי פעילויות ומשימות בדיקה, השפעת ההקשר, הבודקה (Testware - מכלול מרכיבי הבדיקות), העקיבות בין בסיס הבדיקות והבודקה (Testware), וכן בעלי תפקידים בבדיקות.

תקן ISO/IEC/IEEE 29119-2 מספק מידע נוסף בנוגע על תהליכי הבדיקות.

1.4.1 פעילויות ומשימות בדיקה

תהליך הבדיקות כולל בדרך כלל את קבוצות הפעילויות העיקריות שמתוארות מטה. אף שנראה כאילו רבות מפעילויות אלו מתרחשות באופן סדרתי (אחת אחר השנייה), לעיתים קרובות הן מיושמות באופן מחזורי או במקביל. בדרך כלל נדרש להתאים את הפעילויות הללו למערכת ולפרויקט.

תכנון הבדיקות (**Test Planning**) כולל את הגדרת מטרת הבדיקות ובחירת גישה המספקת באופן הטוב ביותר את המטרות הללו במסגרת המגבלות הקיימות להקשר הכולל. תכנון הבדיקות מוסבר לעומק בסעיף 5.1.

ניטור ובקרת הבדיקות (**Test monitoring and control**) ניטור הבדיקות כולל בדיקת-מצב (סטטוס) לפעולות הבדיקה באופן שוטף והשוואה בין ההתקדמות בפועל למול התכנית. בקרת הבדיקות כוללת נקיטת פעולות הנדרשות לצורך השגת המטרות שהוגדרו לבדיקות. ניטור ובקרה של הבדיקות מוסברים בפירוט בסעיף 5.3.

ניתוח הבדיקות (**Test Analysis**) כולל את ניתוח בסיס הבדיקות שמטרתו לזהות מאפיינים (features) הניתנים לבדיקה, להגדיר ולתעדף את תנאי בדיקה (Test conditions), את הסיכונים ורמות הסיכון הקשורים (ראה סעיף 5.2). בסיס הבדיקות ואובייקטי הבדיקות נבחנים גם על מנת לזהות פגמים שהם עשויים להכיל ולהעריך את מידת הבדיקתיות (testability) שלהם. ניתוח הבדיקות נתמך בדרך כלל על ידי שימוש בטכניקות בדיקה (ראה פרק 4). ניתוח הבדיקות עונה על השאלה "מה לבדוק?" במונחים של קריטריוני כיסוי מדידים.

עיצוב הבדיקות (**Test Design**) כולל את הרחבת תנאי הבדיקות למקרי בדיקות (Test Cases) ובודקות אחרות (כמו אמנות בדיקות test charters). פעילות זו כוללת לעיתים קרובות גם זיהוי של פריטי כיסוי, המשמשים לזיהוי ערכי קלט שישמשו במקרי הבדיקות. ניתן להשתמש בטכניקות בדיקה (ראה פרק 4) במהלך פעילות זו. עיצוב הבדיקות גם כולל את הגדרת הדרישות לגבי נתוני בדיקות (Test data) אותם יש להכין, עיצוב סביבת הבדיקות וזיהוי תשתיות וכלים נוספים הנדרשים לביצוע הבדיקות. עיצוב הבדיקות עונה על השאלה "איך לבדוק?".

יישום הבדיקות (**Test implementation**) כולל את יצירת ו/או רכישת הבודקה הנדרשת להרצת הבדיקות (לדוגמה, נתוני בדיקות). מקרי בדיקות יכולים לכלול נוהלי בדיקות (test procedures) (צעדים/הנחיות לביצוע הבדיקה) ולהיות מאורגנים בסדרת בדיקות (test suites). נכתבים תסריטי בדיקה ידניים ואוטומטיים. נוהלי הבדיקות מתועדפים ומסודרים לפי סדר חשיבות בתכנית הרצות לצורך הרצת בדיקות יעילה (ראה סעיף 5.1.5). סביבת הבדיקות מוקמת ומאומתת על מנת לוודא שנבנתה בצורה נכונה.

ביצוע הבדיקות (**Test Execution**) כולל את הרצת הבדיקות לפי תוכנית ההרצה (test runs). ביצוע הבדיקות יכול להיות ידני או אוטומטי. ביצוע הבדיקות יכול להתרחש במגוון דרכים, כולל בדיקות

מתמשכות/בהמשכים (continuous testing) או בדיקות בצמדים/בזוגות (pair testing). תוצאות הבדיקות בפועל מושוות לתוצאות הצפויות. תוצאות הבדיקות מתועדות. חריגות מהתוצאות הצפויות עוברות ניתוח על מנת לזהות את הגורמים להן. ניתוח זה מאפשר לנו לדווח על הפגמים על סמך הכשלים שנצפו (ראה סעיף 5.5).

השלמת הבדיקות (**Test completion**) פעילויות הנערכות בדרך כלל באבני דרך חשובות בפרויקט (למשל שחרור, סוף איטרציה, סיום רמת בדיקה) עבור פגמים שלא נפתרו, בקשות לשינוי או דרישות (product backlog items) שנוצרו. כל פריטי הבדיקות (testware) שעשויים להיות שימושיים בעתיד נשמרים או מועברים לצוותים הרלוונטיים. סביבת הבדיקות מושבתת בהתאם למצב מוסכם. פעילויות הבדיקה מנותחות לצורך זיהוי והפקת לקחים ולשיפור באיטרציות, שחרורים או פרויקטים עתידיים (ראה סעיף 2.1.6). מסמך סיכום הבדיקות (STR) נוצר ומועבר לבעלי העניין.

1.4.2 *תהליך הבדיקות בהתאמה להֶקְשֵר*

בדיקות אינן מבוצעות באופן שהוא מנותק. פעילויות בדיקה הן חלק בלתי נפרד מתהליכי הפיתוח המתבצעים בארגון. הבדיקות ממומנות על ידי בעלי העניין ומטרתן הסופית היא לסייע במימוש הצרכים העסקיים של בעלי העניין. לכן, הדרך בה הבדיקות מבוצעות תלויה במספר גורמים הכוללים את:

- בעלי העניין (צרכים, ציפיות, דרישות, נכונות לשתף פעולה, וכו')
- חברי הצוות (כישורים/מיומנויות, ידע, רמת נסיון, זמינות, צרכי הדרכה, וכו')
- התחום העסקי (קריטיות של אובייקט הבדיקה, סיכונים מזוהים, צרכי השוק, תקנות וחוקים ספציפיים (legal regulations), וכו'
- גורמים טכניים (סוג התוכנה, ארכיטקטורת המוצר, הטכנולוגיה בשימוש, וכו')
- אילוצי פרויקט (היקף, זמן, תקציב, משאבים, וכו')
- גורמים ארגוניים (מבנה הארגון, מדיניות קיימת, פרקטיקות בשימוש, וכו')
- מחזור חיי פיתוח התוכנה (שיטות עבודה בהנדסת תוכנה, שיטות פיתוח, וכו')
- כלים (זמינות, שימושיות, תאימות, וכו')[5]

לגורמים אלו תהיה השפעה על נושאים רבים הקשורים לבדיקות, לרבות: אסטרטגיית בדיקות, טכניקות לבדיקות שיהיו בשימוש, מידת האוטומציה של הבדיקות, רמת הכיסוי הנדרשת, רמת הפירוט הנדרשת של תיעוד הבדיקות, דיווח וכו'.

1.4.3 *בּוֹדְקָה Testware*

בּוֹדְקָה (testware) מהווה את כלל התוצרים המופקים במהלך הביצוע של פעילויות הבדיקה כפי שתוארו בסעיף 1.4.1. קיימת שונות רבה באופן שבו ארגונים שונים מייצרים, מעצבים, מכנים, מארגנים ומנהלים את תוצרי העבודה שלהם.

ניהול תצורה תקין (ראה סעיף 5.4) מבטיח עקביות ושלמות של תוצרי העבודה. להלן רשימה חלקית של תוצרי בדיקות:

- **תוצרי עבודת תכנון בדיקות** (**test planning**) כוללים: תכנית בדיקות, לוח זמנים לבדיקות, מאגר סיכונים, וקריטריוני כניסה ויציאה (ראה סעיף 5.1). מאגר הסיכונים הינה רשימת סיכונים

[5] הערת עורך - לצורך עמידה בסטנדרטים

הכוללת את סבירות הסיכונים, השפעת הסיכונים ומידע אודות הפחתת הסיכונים (ראה סעיף 5.2). לוח הזמנים לבדיקות, רשימת הסיכונים וקריטריוני כניסה ויציאה הם בדרך כלל חלק מתכנית הבדיקות (Test Plan).

- **תוצרי עבודת ניטור ובקרת הבדיקות (test monitoring and control)** כוללים: דוחות התקדמות הבדיקות (ראה סעיף 5.3.2), תיעוד של הוראות בקרה (ראה סעיף 5.3) ומידע אודות סיכונים (ראה סעיף 5.2).
- **תוצרי עבודת ניתוח בדיקות (test analysis)** כוללים: תנאי בדיקה (test conditions)מתועדפים (למשל, קריטריוני קבלה, ראה סעיף 4.5.2) ודוחות ליקויים לפגמים שנמצאו בבסיס הבדיקות (אם לא תוקנו ישירות).
- **תוצרי עבודת עיצוב בדיקות (test design)** כוללים: מקרי בדיקה (test cases) מתועדפים, אמנות בדיקה (test charters), פריטי כיסוי, רשימת נתוני בדיקה נדרשים ודרישות לסביבת הבדיקות.
- **תוצרי עבודת יישום מערך הבדיקות (test implementation)** כוללים: הליכי/נוהלי בדיקות (test procedures), סקריפטים של בדיקות אוטומטיות, סדרת בדיקות (test suites) , נתוני בדיקות (test data), לוח זמנים להרצת הבדיקות (test execution schedule) ורכיבי סביבת הבדיקות. דוגמאות לרכיבי סביבת הבדיקות כוללות: תותבים (stubs), דרייברים (drivers), סימולטורים וירטואליזציות של שירותים.
- **תוצרי עבודת ביצוע הבדיקות (test execution)** כוללים: יומני בדיקות (test logs) ודוחות פגמים (ראה סעיף 5.5).
- **תוצרי עבודת השלמת הבדיקות (test completion)** כוללים: דוח סיכום הבדיקות (ראה סעיף 5.3.2), פריטי פעולות לשיפור פרויקטים או איטרציות עתידיות, לקחים מתועדים שנלמדו, ובקשות לשינויים (change requests) (לדוגמה, דרישות חדשות בצבר דרישות המוצר (product backlog)).

1.4.4 *נֶעֱקָבוּת (Traceability) בין בסיס הבדיקות לבודקה*

על מנת ליישם ניטור ובקרה באופן יעיל בבדיקות, חשוב לבסס ולשמור על נֶעֱקָבוּת (traceability) לאורך תהליך הבדיקה בין רכיבי בסיס הבדיקה, הבודקה המשויכת לרכיבים אלו (למשל, תנאי בדיקות, סיכונים, מקרי בדיקות), תוצאות הבדיקות ופגמים שזוהו.

נֶעֱקָבוּת מדויקת עוזרת בהערכת הכיסוי, ולכן חשוב להגדיר קריטריוני כיסוי מדידים בבסיס הבדיקות. קריטריוני הכיסוי יכולים לשמש כמדדי ביצוע עיקריים (KPI) (key performance indicators) מהם ייגזרו הפעילויות המראות באיזו מידה הושגו מטרות הבדיקות (ראה סעיף 1.1.1). לדוגמה:

- נֶעֱקָבוּת בין מקרי בדיקה לדרישות יכולה לאמת שהדרישות מכוסות על ידי מקרי בדיקה.
- נֶעֱקָבוּת בין תוצאות הבדיקה לסיכונים יכולה לשמש להערכת רמת הסיכונים שנותרו באובייקט הבדיקה.

בנוסף להערכת הכיסוי, נֶעֱקָבוּת טובה מאפשרת לקבוע את ההשפעה של שינויים, מקלה על סקירות של הבדיקות, ועוזרת בעמידה בכללים פנימיים של הארגון (IT governance). נֶעֱקָבוּת טובה גם מאפשרת הבנה טובה יותר של דוחות התקדמות והשלמת הבדיקות על ידי הוספת מצב (סטטוס) רכיבי בסיס הבדיקות לדוחות המעקב. דבר זה יכול גם לסייע בהעברת היבטים טכניים של הבדיקות לבעלי העניין בצורה מובנת. נֶעֱקָבוּת מספקת מידע להערכת איכות המוצר, יכולת התהליך, והתקדמות הפרויקט למול המטרות העסקיות.

1.4.5 תפקידים בבדיקות

בסילבוס זה, מכוסים שני תפקידים עיקריים בתחום הבדיקות: תפקיד ניהול הבדיקות ותפקיד הבודק. הפעילויות והמשימות המוקצות לשני התפקידים הללו תלויות בגורמים כמו ההקשר של הפרויקט והמוצר, המיומנויות של האנשים הנושאים בתפקיד כזה או אחר, והארגון.

תפקיד ניהול הבדיקות מקבל אחריות כוללת על תהליך הבדיקות, צוות הבדיקות וההובלה של פעילויות הבדיקה. תפקיד ניהול הבדיקות מתמקד בעיקר בפעילויות תכנון הבדיקות (test planning), ניטור ובקרת הבדיקות (test monitoring and control) והשלמת הבדיקות (test completion). הדרך בה מבוצע תפקיד ניהול הבדיקות משתנה לפי ההקשר. לדוגמה, בפיתוח תוכנה זריז (אג'ילי), חלק ממשימות ניהול הבדיקות עשויות להיות באחריות הצוות האג'ילי. משימות המתפרשות על פני מספר רב של צוותים או על כל הארגון עשויות להתבצע על ידי מנהלי בדיקות מחוץ לצוות הפיתוח.

תפקיד הבודק כולל אחריות כוללת על ההיבט ההנדסי (הטכני) של הבדיקה. תפקיד הבודק מתמקד בעיקר בפעילויות של ניתוח הבדיקות (test analysis), עיצוב הבדיקות (test design), יישום הבדיקות (test implementation) וביצוע הבדיקות (test execution).

אנשים שונים יכולים למלא את התפקידים הללו בזמנים שונים. לדוגמה, תפקיד מנהל הבדיקות יכול להתבצע על-ידי ראש צוות, על-ידי מנהל בדיקות, על-ידי מנהל פיתוח ועוד. בנוסף, אפשרי שאדם אחד ימלא את תפקידי הבודק ומנהל הבדיקות בו זמנית.

1.5 מיומנויות חיוניות ושיטות עבודה טובות בבדיקות

מיומנות היא היכולת לבצע משהו באופן מיטבי שנגזר מהידע, התרגול והכישרון של האדם. לבודקים טובים צריכות להיות כמה מיומנויות חיוניות על מנת לבצע את משימתם היטב. בודקים טובים צריכים להיות חברי צוות יעילים ולהיות מסוגלים לבצע בדיקות ברמות שונות של עצמאות הבדיקות.

1.5.1 מיומנויות כלליות הנדרשות לבדיקה

בעוד שהמיומנויות הבאות הינן כלליות, הן רלוונטיות במיוחד לבודקים:

- ידע בתחום הבדיקות (להגברת היעילות של הבדיקות, לדוגמה באמצעות שימוש בטכניקות בדיקה)
- יסודיות, זהירות, סקרנות, תשומת לב לפרטים, שיטתיות (לזיהוי פגמים, בעיקר אלו שקשים למציאה)
- כישורי תקשורת טובים, האזנה פעילה, להיות שחקן צוות (ליצור אינטראקציה יעילה עם כל בעלי העניין, להעברת מידע לאחרים, להיות מובן ולדווח ולדון בפגמים)
- חשיבה אנליטית, חשיבה ביקורתית, יצירתיות (להגברת היעילות של הבדיקות)
- ידע טכני (להגברת היעילות של הבדיקות, לדוגמה באמצעות כלי בדיקות מתאימים)
- ידע בתחום הרלוונטי בו פועלת המערכת (פיננסים, בטחון, תקשורת, וכו') (כדי להיות מסוגל להבין ולתקשר עם משתמשי הקצה/הנציגים העסקיים)

לעיתים קרובות הבודקים הינם נושאי החדשות הרעות. תכונה אנושית נפוצה הינה להאשים את נושאי החדשות הרעות. מה שהופך את מיומנויות התקשורת לחיוניים עבור בודקים. דיווח תוצאות הבדיקות

יכול להיתפס כביקורת על המוצר ועל היוצר שלו. הטיית האישוש[6] (Confirmation bias) יכולה להקשות על קבלת מידע המנוגד לאמונות בהן מחזיק השומע. חלק מהאנשים יכול לראות בבדיקות פעילות הרסנית (destructive), למרות שהיא תורמת רבות להצלחת הפרויקט ולאיכות המוצר. על מנת לשפר את נקודת מבט זו, יש לדווח את המידע על פגמים וכשלים בצורה בונה.

1.5.2 *גישת הצוות השלם (whole team approach)*

אחת המיומנויות החשובות לבודק היא היכולת לעבוד באופן יעיל בהקשר הצוותי ולתרום באופן חיובי למטרות הצוות. גישת הצוות השלם – שיטת עבודה שמגיעה מגישת התכנות המהיר (Extreme Programming) (ראה פרק 2.1) - נבנתה לפי מיומנות זו.

בגישת הצוות השלם, כל חבר צוות בעל ידע ומיומנויות נדרשות יכול לבצע כל משימה, וכולם אחראים על האיכות. חברי הצוות חולקים את אותו מרחב העבודה (פיזי או וירטואלי), שכן מיקום משותף מקל על תקשורת ואינטראקציה. גישת הצוות השלם משפרת את הדינמיקה של הצוות, משפרת את התקשורת ושיתוף הפעולה בין חברי הצוות ויוצרת סינרגיה על ידי מינוף המיומנויות השונות בצוות לטובת הפרויקט.

בודקים עובדים בצמוד לחברי הצוות האחרים על מנת להבטיח שרמות האיכות הרצויות מושגות. דבר זה כולל שיתוף פעולה עם נציגים עסקיים כדי לסייע להם ביצירת בדיקות קבלה מתאימות ועבודה עם מפתחים כדי להסכים על אסטרטגיית הבדיקות ולהחליט על גישות לבדיקות אוטומטיות. בודקים יכולים כך להעביר ידע מתחום הבדיקות לחברי צוות אחרים ולהשפיע על פיתוח המוצר.

בהתאם להקשר, ייתכן שגישת הצוות השלם לא תמיד תהיה מתאימה. לדוגמה, במצבים מסוימים כמו מערכות קריטיות (לדוגמה, מערכות רפואיות, צבאיות, תעופה, רכב, וכו'), נדרשת רמת עצמאות גבוהה של הבדיקות.

1.5.3 *עצמאות הבדיקות*

מידה מסוימת של עצמאות הופכת את הבודק ליותר יעיל במציאת פגמים עקב השוני בין ההטיות הקוגניטיביות של המפתח והבודק (ראו Salman 1995). עם זאת, עצמאות אינה מחליפה את ההיכרות, לדוגמה, מפתחים יכולים למצוא באופן יעיל פגמים רבים בקוד שלהם.

תוצרי עבודה יכולים להיבדק על ידי המחבר שלהם (המפתח) (ללא עצמאות), על ידי מפתח אחר מאותו צוות (עמית) (עצמאות נמוכה), על ידי בודקים מחוץ לצוות הפיתוח אך בתוך הארגון (עצמאות גבוהה), או על ידי בודקים מחוץ לארגון (עצמאות גבוהה מאוד). לרוב הפרויקטים בדרך כלל, כדאי לבצע בדיקות עם רמות שונות של עצמאות (לדוגמה, המפתחים מבצעים בדיקות רכיבים ובדיקות אינטגרציה בין רכיבים, צוות הבדיקות מבצע בדיקות מערכת ובדיקות אינטגרציה בין מערכות, והנציגים העסקיים מבצעים בדיקות קבלה).

היתרון העיקרי בעצמאות הבדיקות הוא שבודקים עצמאיים עשויים לזהות סוגים שונים של כשלים ופגמים, בשל ההבדלים ברקע, נקודת מבט טכנית וההטיות השונות שלהם. בנוסף, בודק עצמאי (בלתי תלוי) יכול לאמת, לערער או להפריך הנחות שנעשו על ידי בעלי עניין במהלך הכנת המפרטים ויישום המערכת.

---

[6] הערת עורך - ויקיפדיה: "הטיית האישוש (Confirmation bias) היא הנטייה לחפש, לפרש, להעדיף, ולזכור מידע המאשש אמונות או השערות קודמות, תוך התעלמות ממידע סותר או כזה שתומך באפשרויות חלופיות."

אף על פי כך, ישנם גם חסרונות. בודקים עצמאיים עשויים להיות מבודדים מצוות הפיתוח, דבר היכול להוביל לחוסר שיתוף פעולה, בעיות תקשורת או ליחסים עוינים עם צוות הפיתוח. מפתחים עשויים לאבד את תחושת האחריות לאיכות. בודקים עצמאיים עשויים להיתפס כצוואר בקבוק או כאשמים בעיכובים בשחרור (release) גירסת התוכנה, האפליקציה או המערכת.

2 בדיקות לאורך מחזור חיי פיתוח תוכנה — 130 דקות

מושגים

אובייקט בדיקה (test object), בדיקות אישור (confirmation testing), בדיקות אינטגרציה (integration testing), בדיקת אינטגרציית מערכת (integration testing System), בדיקות לא-תפקודיות / לא-פונקציונליות (non-functional testing), בדיקות מערכת (system testing), בדיקות נסיגה (regression testing), בדיקות פונקציונליות (functional testing), בדיקות שילוב רכיבים (component integration testing), בדיקות קבלה (acceptance testing), בדיקות קופסה לבנה (white-box testing), בדיקות קופסה שחורה (black-box testing), בדיקות רכיבים (component testing), בדיקות תחזוקה (maintenance testing), הזזה לשמאל (shift left) , סוג בדיקה (test type), רמת בדיקה (test level).

יעדי הלימוד לפרק 2

2.1 בדיקות בהקשר של מחזור חיים לפיתוח תוכנה

FL-2.1.1 (K2) הסבר את ההשפעה של סוגי מחזור חיים לפיתוח תוכנה על הבדיקות

FL-2.1.2 (K1) הכר שיטות בדיקה שמתאימות לכל סוגי מחזור חיים של פיתוח תוכנה

FL-2.1.3 (K1) הכר דוגמאות של גישות בדיקות תחילה (test-first) לפיתוח

FL-2.1.4 (K2) סכם כיצד DevOps עשוי להשפיע על בדיקות

FL-2.1.5 (K2) הסבר את גישת הזזה לשמאל (shift left)

FL-2.1.6 (K2) הסבר כיצד סקירה במבט לאחור (retrospectives) יכולה לשמש כמנגנון לשיפור תהליכים

2.2 רמות בדיקה וסוגי בדיקה

FL-2.2.1 (K2) הבחן בין רמות הבדיקה השונות

FL-2.2.2 (K2) הבחן בין סוגי הבדיקות השונים

FL-2.2.3 (K2) הבחן בין בדיקת אישור לבדיקת רגרסיה

2.3 בדיקות תחזוקה

FL-2.3.1 (K2) סכם את הגורמים לבדיקות תחזוקה

2.1 בדיקות בהקשר של מחזור חיים של פיתוח תוכנה

מודל מחזור חיי פיתוח תוכנה (SDLC) הוא ייצוג מופשט ברמה גבוהה של תהליך הפיתוח לתוכנה. מודל SDLC מגדיר כיצד שלבי פיתוח שונים וסוגי פעילויות שונות מבוצעות בתהליך זה מתייחסים זה לזה, הן מבחינה לוגית והן מבחינה כרונולוגית. דוגמאות למודלים של SDLC כוללות: מודלים לפיתוח רציף (למשל, מודל מפל מים, מודל V), מודלים לפיתוח בסבבים (iterative) (לדוגמה, מודל ספירלי, אב טיפוס), ומודלים לפיתוח מצטבר (incremental) (לדוגמה, תהליך מאוחד (Unified Process)).

פעילויות מסוימות בתוך תהליכי פיתוח תוכנה יכולות להיות מפורטות יותר גם באמצעות שיטות פיתוח תוכנה מתקדמות יותר ושיטות פיתוח אג'יליות (Agile).
להלן דוגמאות: פיתוח מונחה מבחני קבלה (ATDD), פיתוח מונחה התנהגות (BDD), עיצוב מונחה תחום/ים (DDD), תכנות מהיר (XP), פיתוח מונחה תכונות (FDD), קאנבאן (Kanban), Lean IT, סקראם (Scrum) ופיתוח מונחה בדיקות (TDD).

2.1.1 השפעת מחזור החיים של פיתוח תוכנה (SDLC) על בדיקות

בכדי להצליח - הבדיקה חייבת להיות מותאמת ל-SDLC. הבחירה ב-SDLC משפיעה על:

- היקף ועיתוי פעילויות הבדיקה (למשל, רמות הבדיקה וסוגי הבדיקות)
- רמת הפירוט של תיעוד הבדיקות
- בחירת טכניקות בדיקה וגישה לבדיקות
- היקף אוטומציה של הבדיקות
- תפקידים ואחריות של הבודק

בשלבים הראשונים במודלים לפיתוח רציף, הבודקים משתתפים בדרך כלל בסקירת הדרישות, ניתוח בדיקות ועיצוב בדיקות. קוד המערכת עצמו נוצר בדרך כלל בשלבים מאוחרים יותר, ולכן בדרך כלל לא ניתן לבצע בדיקות דינמיות בשלב מוקדם ב- SDLC.

בכמה מודלים לפיתוח מחזורי או מצטבר, ההנחה היא שכל מחזור מספק אב טיפוס עובד או תוספת מוצר. משמעות הדבר היא כי בכל איטרציה (מחזור) ניתן לבצע בדיקות סטטיות ודינמיות בכל רמות הבדיקה. אספקת תוספות באופן תָכוף דורשת משוב מהיר ובדיקות רגרסיה נרחבות.

פיתוח תוכנה אג'ילי (זריז וגמיש) מניח שהשינויים עשויים להתרחש במהלך הפרויקט. לכן, נדרש תיעוד קל-משקל לתוצר העבודה ואוטומציה נרחבת של הבדיקות, על מנת להקל על בדיקות הרגרסיה המועדפות בפרוייקטים אג'יליים. כמו כן, רוב הבדיקות הידניות נוטות להיעשות באמצעות טכניקות בדיקה מבוססות ניסיון שאינן דורשות ניתוח ועיצוב בדיקות מקדימות מקיפות (ראה סעיף 4.4).

2.1.2 מחזור חיי פיתוח תוכנה ושיטות בדיקה יעילות ומוכחות

ללא תלות במודל SDLC שנבחר, שיטות בדיקה טובות כוללות את הדברים הבאים:

- לכל פעילות פיתוח תוכנה, קיימת פעילות בדיקה מתאימה, כך שכל פעילויות הפיתוח כפופות לבקרת איכות
- לרמות בדיקה שונות (ראה פרק 2.2.1) יש מטרות בדיקה ספציפיות ושונות, דבר שיאפשר שהבדיקות יהיו מקיפות ומספקות ובמקביל תמנע יתירות
- עבור רמת בדיקה נתונה, ניתוח הבדיקה ועיצובה מתחיל במשולב עם שלב הפיתוח המתאים למודל ה-SDLC, כך שהבדיקות יוכלו לעמוד בעקרון הבדיקה המוקדמת (ראה סעיף 1.3)

- ברגע שהטיוטות של תוצר מסוים זמינות, הבודקים מעורבים בסקירת תוצרי העבודה, ותומכים באסטרטגיית הזזה-שמאל (ראה סעיף 2.1.5) על ידי בדיקה וזיהוי פגמים בשלבים מוקדמים

2.1.3 *בדיקות כמנוע לפיתוח תוכנה*

TDD, ATDD ו-BDD הן גישות פיתוח דומות, שבהן הבדיקות מוגדרות כאמצעי להובלת הפיתוח[7]. כל אחת מהגישות אלו מיישמת את עקרון הבדיקה המוקדמת (ראו פרק 1.3) ונוקטת בגישת הזזה-שמאלה (ראה סעיף 2.1.5), שכן הבדיקות מוגדרות לפני שהקוד נכתב.
גישות אלו תומכות במודל פיתוח איטרטיבי, ומאופיינות כדלקמן:

פיתוח מונחה בדיקות (TDD):

- מנחה את הקידוד באמצעות מקרי בדיקה (במקום עיצוב תוכנה נרחב) (Beck 2003)
- בדיקות נכתבות תחילה, הקוד נכתב לאחר מכן כדי לעמוד בבדיקות, ולאחר מכן הבדיקות והקוד מעוצבים מחדש (refactored)

פיתוח מונחה בדיקות קבלה (ATDD) (ראה סעיף 4.5.3):

- גזירת בדיקות מקריטריוני הקבלה כחלק מתהליך תכנון המערכת (Gärtner 2011)
- בדיקות נכתבות לפני שמפתחים את החלק ביישום שמממש את מה שבדיקות אלה מוודאות

פיתוח מונחה התנהגות (BDD):

- מבטא את ההתנהגות הרצויה של יישום עם מקרי בדיקה[8] כתובים בשפה טבעית, קלה להבנה על ידי בעלי עניין – בדרך כלל באמצעות תבנית בהינתן/כאשר/אז (given/when/then) (Chelimsky 2010)
- מקרי בדיקה מתורגמים אוטומטית לבדיקות הניתנות להרצה

עבור כל הגישות לעיל, בדיקות עשויות להימשך כבדיקות אוטומטיות על מנת להבטיח את איכות הקוד גם לאחר התאמות ושינויים עתידיים.

2.1.4 *DevOps ובדיקות*

DevOps היא גישה ארגונית שמטרתה ליצור סינרגיה בין הפיתוח (Dev) (שכולל את הבדיקות) והתפעול (Ops) על ידי עבודה משותפת, בכדי להשיג קבוצה של מטרות משותפות. DevOps דורש שינוי תרבות אירגוני על מנת לגשר על הפערים בין פיתוח (כולל בדיקות) לבין תפעול תוך התייחסות שוות-ערך לתפקודיהם. DevOps מקדם אוטונומיה לצוותים, משוב מהיר, כלים תומכים ופרקטיקות טכניות כגון אינטגרציה רציפה (CI) ואספקה רציפה (CD). כל זה מאפשר לצוותים לבנות, לבדוק ולשחרר קוד באיכות גבוהה מהר יותר תוך שימוש בצנרת אספקה (DevOps pipeline) (Kim 2016).

חלק מהיתרונות של DevOps מנקודת המבט של הבדיקות, הם:

- משוב מהיר על איכות הקוד והאם שינויים משפיעים לרעה על קוד קיים
- אינטגרציה רציפה (CI) מקדמת גישה של הזזה-שמאל בבדיקות (ראה סעיף 2.1.5) על ידי עידוד מפתחים לפתח קוד באיכות גבוהה מלווה בבדיקות רכיבים וניתוח סטטי לקוד

[7] הערת עורך – גישה שבה הבדיקות מוגדרות כאמצעי להובלת הפיתוח נקראת גישת "בדיקות-תחילה" (test-first)
[8] הערת עורך – ראה: "נספח ד – הבהרות מיוחדות (עבור המהדורה בעברית בלבד)"

- מקדם תהליכים אוטומטיים כמו אינטגרציה רציפה/אספקה רציפה (CI/CD) המאפשרים הקמת סביבות בדיקה יציבות
- מגביר את בחינתם של מאפייני איכות לא פונקציונליים (למשל, ביצועים, אמינות)
- אוטומציה באמצעות צינור אספקה (pipeline) מפחיתה את הצורך בבדיקות ידניות חוזרות ונשנות
- הסיכון ברגרסיה ממוזער בשל קנה המידה והטווח[9] של בדיקות רגרסיה אוטומטיות

DevOps אינו חף מסיכונים ואתגרים, הכוללים:

- יש להגדיר וליישם את התהליך האוטומטי (pipeline) של DevOps
- יש ליישם ולתחזק את כלי ה-CI/CD
- אוטומציית בדיקות דורשת משאבים נוספים ועשויה להיות קשה להקמה ולתחזוקה

למרות שה-DevOps מגיע עם רמה גבוהה של בדיקות אוטומטיות, עדיין יהיה צורך בבדיקות ידניות – במיוחד מנקודת המבט של המשתמש.

2.1.5 *גישת הזזה שמאלה (Shift – Left)*

עקרון הבדיקה המוקדמת (ראה סעיף 1.3) מכונה לעיתים "הזזה שמאלה" משום שזאת גישה שבה הבדיקות מבוצעת מוקדם יותר ב- SDLC.
עקרון ה- Shift-Left מציע בדרך כלל כי הבדיקה צריכה להיעשות מוקדם יותר (למשל, לא לחכות לקוד שייושם או לשילוב רכיבים), אך אין זה אומר שיש לזנוח את הבדיקות בשלב מאוחר יותר ב-SDLC.

ישנן כמה שיטות עבודה טובות הממחישות כיצד להשיג "תזוזה שמאלה" בבדיקות:

- סקירת המפרט מנקודת מבט של הבדיקות. פעילויות סקירה במפרטים מוצאות לעתים קרובות פגמים פוטנציאליים, כגון אי-בהירויות, חוסר שלמות וחוסר עקביות
- כתיבת מקרי בדיקה לפני כתיבת הקוד והפעלת הקוד בסביבת בדיקה Test harness מבודדת במהלך כתיבת הקוד
- שימוש ב- CI ועוד יותר ב- CD טוב מכיוון שהוא מספק משוב מהיר ובדיקות רכיבים אוטומטיות הבודקות את קוד המקור ובעת שילובו במאגר המרכזי של קוד המערכת
- ביצוע ניתוח סטטי לקוד המקור לפני בדיקות דינמיות, או כחלק מתהליך בדיקה אוטומטי
- ביצוע בדיקות לא פונקציונליות החל מרמת בדיקת הרכיבים, במידת האפשר. זו צורה של הזזה שמאלה מכיוון שסוגי בדיקות לא פונקציונליות אלה נוטים להתבצע מאוחר יותר ב- SDLC כאשר קיימות מערכת שלמה וסביבת בדיקה המדמה את הסביבה המבצעית

גישה של הזזה שמאלה עשויה להוביל לצורך בהכשרה נוספת, למאמץ ו/או לעלויות בשלב מוקדם יותר של התהליך, אך היא צפויה לגרום לחסכון במאמצים ו/או בעלויות בהמשך התהליך.

חשוב לרתום את בעלי העניין לגישת הזזה-שמאלה.

[9] הערת עורך – קנה המידה (scale) והטווח (range) בהקשר של הכיסוי.

2.1.6 פגישות רטרוספקטיבה – (מפגשי ראיה כוללת) ושיפור תהליכים

פגישות רטרוספקטיבה (הידועות גם כ"פגישות סיום הפרויקט" ו"רטרוספקטיבות פרויקט") נערכות לעתים קרובות בסיום פרויקט או בסיום סבב (איטרציה) או באבן דרך של שחרור קוד, או שניתן לקיים בעת הצורך. העיתוי וארגון מפגשים אלו תלוי במודל SDLC המסוים שננקט.
בפגישות אלו פוגשים את כל המשתתפים (לא רק בודקים, אלא גם למשל, מפתחים, אדריכלים, בעלי מוצר, עסק אנליסטים) ודנים ב:

- מה הצליח, וצריך לשמר?
- מה לא הצליח, וניתן יהיה לשפר?
- כיצד לשלב את השיפורים ולשמר את ההצלחות בעתיד?

התוצאות צריכות להיות מתועדות והן בדרך כלל חלק מדוח סיכום הבדיקה (ראה סעיף 5.3.2).
פגישות אלו הכרחיות לשיפור מתמיד וחשוב ליישם את מסקנותיהן.

יתרונות אופייניים לבדיקות כוללים:

- הגברת האפקטיביות/יעילות הבדיקות (למשל, על ידי יישום הצעות לשיפור התהליך)
- שיפור איכות הבודקה (Testware) (למשל, על ידי סקירה משותפת של תהליכי הבדיקות)
- גיבוש צוות ולמידה (למשל, כתוצאה מההזדמנות להעלות נושאים ולהציע נקודות לשיפור)
- שיפור איכות של בסיס הבדיקה (למשל, ניתן לדון ולפתור בעיות איכות וחוסרים במסמך הדרישות)
- שיפור שיתוף הפעולה בין הפיתוח והבדיקות (למשל, כאשר שיתוף הפעולה נסקר ומשופר באופן קבוע)

2.2 רמות הבדיקה וסוגי הבדיקה

רמות בדיקה הן קבוצות של פעילויות בדיקה המאורגנות ומנוהלות יחד.
כל רמת בדיקה היא מופע של תהליך הבדיקה, המבוצע ביחס לתוכנה בשלב נתון של פיתוח, מהרכיב הבודד ועד למערכות שלמות או, במידת הצורך, למערכות של מערכות.

רמות הבדיקה קשורות לפעילויות אחרות בתוך SDLC. במודלים של SDLC רציפים, רמות הבדיקה מוגדרות לעתים קרובות כך שקריטריוני היציאה של רמה אחת הם חלק מקריטריוני הכניסה לרמה הבאה. אולם בחלק מהמודלים האיטרטיביים, ייתכן שמצב זה לא יתקיים.פעילויות הפיתוח עשויות להתפרש על פני מספר רמות בדיקה. בדיקות ברמות הבדיקה השונות עשויות להיות חופפות בזמן.

סוגי בדיקות הם קבוצות של פעילויות בדיקה הקשורות למאפייני איכות ספציפיים. את רוב סוגי הבדיקות הללו ניתן לבצע בכל רמת בדיקה.

2.2.1 רמות הבדיקה

בסילבוס זה מתוארות חמש רמות הבדיקה הבאות:

- **בדיקת רכיבים** (הידועה גם בשם בדיקת יחידה) מתמקדת בבדיקת רכיבים במבודד מרכיבים אחרים. לעתים קרובות דורשת הבדיקה תמיכה ספציפית, כגון סביבת פיתוח עם תמיכה בכתיבת והרצת טסטים (test harness) או מערכת לבדיקות יחידה (unit test framework). בדיקות רכיבים מבוצעות בדרך כלל על ידי מפתחים בסביבות הפיתוח שלהם.

- **בדיקת אינטגרציית רכיבים** (הידועה גם בשם בדיקת שילוב יחידות) מתמקדת בבדיקת ממשקים ואינטראקציות בין רכיבים. בדיקות אינטגרציית (שילוב) רכיבים תלויות במידה רבה בגישות אסטרטגיית האינטגרציה, גישות כגון: מלמטה למעלה (bottom-up), מלמעלה למטה (top-down) או המפץ הגדול (big-bang).
- **בדיקות מערכת** מתמקדות בהתנהגות וביכולות הכוללות של מערכת או מוצר שלם, לעתים קרובות כוללות בדיקות פונקציונליות של משימות מקצה לקצה ובדיקות לא פונקציונליות של מאפיינים שעדיף לבדוק אותם על מערכת שלמה בסביבת בדיקה מייצגת (למשל, שימושיות). שימוש בסימולציות של תת-מערכות אפשריות גם כן. בדיקות מערכת יכולות להתבצע על ידי צוות בדיקה בלתי תלוי, וקשורות לבדיקות על פי מפרטים של המערכת
- **בדיקות אינטגרציה** בין מערכות מתמקדות בבדיקת ממשקים של המערכת הנבדקת עם מערכות ושירותים חיצוניים. בדיקת אינטגרציית (שילוב) מערכות דורשת סביבות בדיקה מתאימות, רצוי בכזו שדומה לסביבה המבצעית.
- **בדיקות קבלה** מתמקדות בתיקוף ובהוכחת מוכנות המערכת להפעלה, שמשמעותה שהמערכת ממלאת את הצרכים העסקיים של המשתמש. באופן אידיאלי בדיקות קבלה צריכות להתבצע על ידי המשתמשים המיועדים. הצורות העיקריות של בדיקות קבלה הן: בדיקות קבלת על ידי משתמשים (UAT), בדיקות קבלה תפעוליות, בדיקות קבלה חוזיות ורגולטוריות, בדיקות אלפא ובדיקות בטא.

רמות הבדיקה נבדלות על ידי רשימת התכונות הלא ממצה הבאה, כדי למנוע חפיפה של פעילויות הבדיקה:

- אובייקט הבדיקה
- מטרות הבדיקה
- בסיס הבדיקה
- פגמים וכשלים
- גישה ואחריות

2.2.2 *סוגי בדיקות*

קיימים סוגי בדיקות רבים וניתן ליישם אותם בפרויקטים. בסילבוס זה, מתייחס לארבעה סוגי הבדיקות.

בדיקות תיפקוד / פונקציונליות מעריכות את הפונקציות שרכיב או מערכת צריכים לבצע. הפונקציות הן בבחינת "מה" אובייקט הבדיקה צריך לעשות. המטרה העיקרית של בדיקות פונקציונליות היא בדיקת שלמות, תקינות תפקודית והתאמה תפקודית.

בדיקות לא-תפקודיות / לא-פונקציונליות מעריכות תכונות שאינן מאפיינים פונקציונליים של רכיב או מערכת. בדיקה לא פונקציונלית היא בדיקה של "כמה טוב המערכת מתנהגת". המטרה העיקרית של בדיקות לא פונקציונליות היא בדיקת מאפייני איכות התוכנה שאינם פונקציונליים. תקן ISO/IEC 25010 מסווג את מאפייני איכות התוכנה שאינם פונקציונליים:

- יעילות ביצועים
- תאימות
- שימושיות
- אמינות

- אבטחת מידע
- תחזוקתיות
- ניידות

לעיתים עדיף שבדיקות לא תפקודיות תתחלנה בשלב מוקדם במחזור החיים (למשל, כחלק מסקירת ובדיקות רכיבים או בדיקות מערכת). בדיקות לא תפקודיות רבות נגזרות מבדיקות תפקודיות כאשר הן משתמשות באותן בדיקות תפקודיות, אך יבדקו כי בעת ביצוע הפונקציה, המערכת עומדת באילוץ לא-תפקודי (למשל, בדיקה שפונקציה מבוצעת בתוך זמן מוגדר, או שפונקציה יכולה להיות מנוידת לפלטפורמה חדשה). גילוי מאוחר של פגמים לא תפקודיים יכול להוות איום רציני על הצלחה של פרויקט. בדיקות לא תפקודיות דורשות לעיתים סביבת בדיקה ספציפית מאוד, כגון מעבדת שימושיות לבדיקת שימושיות.

בדיקות קופסה שחורה (ראה סעיף 4.2) מבוססות מפרטים ונגזרות מתיעוד חיצוני לאובייקט הבדיקה. המטרה העיקרית של בדיקות קופסה שחורה היא לבדוק את התנהגות המערכת מול מפרטים

בדיקות קופסה לבנה (ראה סעיף 4.3) מבוססות מבנה ונגזרות מהמבנה פנימי של המערכת (לדוגמה, קוד, ארכיטקטורה, זרימות עבודה וזרימות נתונים). המטרה הראשית של בדיקות קופסא לבנה הינה לכסות את המבנה הפנימי על ידי בדיקות, עד להגעה לרמה מוסכמת של כיסוי

ניתן ליישם את כל ארבעת סוגי הבדיקות שהוזכרו לעיל על כל רמות הבדיקה, אם כי המיקוד יהיה שונה בכל רמה. ניתן להשתמש בטכניקות בדיקה שונות כדי לגזור תנאי בדיקה ומקרי בדיקה עבור כל סוגי הבדיקות שהוזכרו.

2.2.3 *בדיקות אישור ובדיקות נסיגה (רגרסיה)*

שינויים מתבצעים בדרך כלל ברכיב או במערכת כדי לשפר אותם על-ידי הוספת תכונה חדשה או על-ידי הסרת פגם. הבדיקות צריכות לכלול גם בדיקות אישור ובדיקות רגרסיה.

בדיקות אישור מאשרות כי הפגם המקורי תוקן בהצלחה. בהתאם לסיכון, ניתן לבדוק את הגרסה המתוקנת של התוכנה במספר דרכים, כולל:

- ביצוע כל מקרי הבדיקה שנכשלו בעבר עקב הפגם, או גם על ידיו
- הוספת בדיקות חדשות כדי לכסות את כל השינויים שנדרשו כדי לתקן את הפגם

עם זאת, כאשר חסר זמן או כסף בעת תיקון פגמים, בדיקות האישור עשויות להיות מוגבלות לביצוע הצעדים שאמורים לשחזר את הכשל שנגרם על ידי הפגם ולבדוק כי הכשל אינו חוזר.

בדיקות נסיגה (רגרסיה) מאשרות כי לא נגרמו תוצאות שליליות כתוצאה משינוי, כולל תיקון שכבר נבדק בבדיקות אישור. השלכות שליליות אלה עלולות להשפיע על אותו הרכיב שבו בוצע השינוי, על רכיבים אחרים באותה מערכת, או אפילו על מערכות מחוברות. בדיקת רגרסיה אינה מוגבלת לאובייקט הבדיקה עצמו, ויכולה להיות גם קשורה לסביבה. מומלץ תחילה לבצע ניתוח השפעה (Impact Analysis) כדי לייעל את היקף בדיקות הרגרסיה. ניתוח ההשפעה מראה אילו חלקים של התוכנה עלולים להיות מושפעים.

החבילות של בדיקות הרגרסיה מורצות פעמים רבות ובדרך כלל מספר מקרי בדיקות הרגרסיה יגדל עם כל איטרציה או שחרור, כך שבדיקת רגרסיה היא מועמדת חזקה לאוטומציה. אוטומציה של בדיקות אלה צריכה להתחיל בשלב מוקדם של הפרויקט. כאשר נעשה שימוש ב-CI, כגון ב-DevOps (ראה סעיף 2.1.4), מומלץ לכלול גם בדיקות רגרסיה אוטומטיות. בהתאם למצב, זה עשוי לכלול בדיקות רגרסיה ברמות שונות.

בדיקות אישור ו/או בדיקת רגרסיה עבור אובייקט הבדיקה נדרשות בכל רמות הבדיקה אם פגמים מתוקנים ו/או אם מתבצעים שינויים ברמות בדיקה אלו.

2.3 בדיקות תחזוקה - *Maintenance Testing*

ישנן קטגוריות שונות של תחזוקה, כגון תיקונים, התאמות לשינויים בסביבה או שיפור ביצועים או תחזוקתיות (לפרטים נוספים ראה תקן ISO/IEC 14764), כך שתחזוקה יכולה לכלול גרסאות מתוכננות וגרסאות לא מתוכנות (כגון: גרסת תיקונים חמים - hot fixes). ניתוח ההשפעה עשוי להיעשות לפני ביצוע השינוי, כדי לעזור להחליט אם השינוי צריך להתבצע, בהתבסס על ההשלכות האפשריות בתחומים אחרים של המערכת. בדיקת השינויים במערכת המבצעית (ב-Production) כוללת הן את הערכת הצלחת יישום השינוי והן בדיקות רגרסיה על חלקים של המערכת שנשארים ללא שינוי (בדרך כלל מרבית המערכת).

היקף בדיקות התחזוקה תלוי בדרך כלל ב:

- מידת הסיכון לשינוי
- גודל המערכת הקיימת
- גודל השינוי

ניתן לסווג את הגורמים לביצוע פעולות תחזוקה ובדיקות ותחזוקה באופן הבא:

- שינויים, כגון שיפורים מתוכננים (כלומר, כחלק מגרסת תוכנה), תיקונים או תיקונים חמים.
- שדרוגים או הגירות (migrations) של הסביבה התפעולית, כגון מפלטפורמה אחת לאחרת, אשר דורשות בדיקות הקשורות לסביבה החדשה, כמו גם לתוכנה שעברה שינוי, או בדיקות של המרת נתונים כאשר נתונים מיישום אחר מועברים למערכת עליה מבצעים תחזוקה.
- פרישה (retirement), כגון כאשר אפליקציה מגיעה לסוף חייה (end-of-life). כאשר מערכת יוצאת מכלל שימוש, נדרשת בדיקה של אחסון הנתונים בארכיון במידה ונדרשות תקופות ארוכות של שמירת נתונים. ייתכן שיהיה גם צורך בבדיקות לאחזור נתונים לאחר אחסון בארכיון במקרה שבו יש צורך בנתונים מסוימים.

3 בדיקות סטטיות — 80 דקות

מושגים

אנומליה (anomaly), בדיקות דינמיות (dynamic testing), בדיקות סטטיות (static testing), ביקורת (inspection), דיון מודרך (walkthrough), ניתוח סטטי (static analysis), סקירה (review), סקירה טכנית (technical review), סקירה לא רשמית/לא פורמלית (informal review), סקירה רשמית/פורמלית (formal review).

מטרות למידה לפרק 3:

3.1 יסודות הבדיקות הסטטיות

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

FL-3.1.2 (K2) הסבר את הערך (the value) של הבדיקות הסטטיות

FL-3.1.3 (K2) השווה ומצא ניגודים בין בדיקות סטטיות ובדיקות דינמיות

3.2 תהליך משוב וסקירה

FL-3.2.1 (K1) זהה את היתרונות של משוב מוקדם וכזה שמבוצע לעיתים תכופות מצד בעלי העניין

FL-3.2.2 (K2) סכם את פעילויות תהליך הסקירה

FL-3.2.3 (K1) זכור אילו תחומי אחריות מוקצים לתפקידים הראשיים בעת ביצוע סקירות

FL-3.2.4 (K2) השווה ומצא ניגודים בין סוגי הסקירות השונים

FL-3.2.5 (K1) זכור את הגורמים שתורמים להצלחת הסקירה

3.1 יסודות הבדיקות הסטטיות

בניגוד לבדיקות דינמיות, בבדיקות סטטיות אין צורך להריץ את התוכנה הנבדקת.

קוד, מפרטי תהליכים, מפרטי ארכיטקטורת המערכת או תוצרי עבודה אחרים עוברים בחינה ידנית (לדוגמה: סקירות) או בעזרת כלי (לדוגמה: ניתוח סטטי). מטרות הבדיקות הסטטיות כוללות שיפור האיכות, איתור פגמים והערכת מאפיינים כמו קְרִיאוּת (readability), שְׁלֵמוּת (completeness), נכונות (correctness), בדיקתיות (testability) ועֲקֵבִיוּת (consistency). ניתן לבצע בדיקות סטטיות לצורך אימות (verification) ווידוא (validation).

בודקים, נציגים עסקיים ומפתחים עובדים ביחד במהלך מיפוי הדוגמאות, כתיבת סיפורי המשתמש ומפגשי דיוק/חידוד הצבר (ה-backlog) הדרישות, על מנת להבטיח שסיפורי משתמש והתוצרים הקשורים עומדים בקריטריונים מוגדרים, לדוגמה: ה"הגדרה של מוכן" (Definition of Ready) (ראה סעיף 5.1.3). ניתן ליישם טכניקות של סקירות על מנת להבטיח שסיפורי המשתמש שלמים ומובנים וכוללים קריטריוני קבלה (acceptance criteria) הניתנים לבדיקה. על ידי שאלת השאלות הנכונות, הבודקים חוקרים, מאתגרים ומסייעים לשפר את סיפורי המשתמש המוצעים.

ניתוח סטטי יכול לזהות בעיות לפני הבדיקות הדינמיות ובדרך כלל דורש פחות מאמץ, מאחר ואין צורך בשימוש במקרי בדיקה, כיוון שבדרך כלל הוא נעשה באמצעות שימוש בכלים (ראה פרק 6). לעיתים קרובות הניתוח הסטטי מוטמע בתוך תהליך אינטגרציה רציפה (CI - continuous integration) (ראה סעיף 2.1.4). בעוד שניתוח סטטי בדרך כלל משמש לגילוי פגמים ספציפיים בקוד הוא משמש בנוסף גם להערכת מידת התחזוקתיות והאבטחה של הקוד. דוגמאות נוספות לכלי ניתוח סטטי הינם כלי "איות" הקוד (Spelling checkers), וכלים הבודקים את מידת המובנות וקריאות הקוד.

3.1.1 תוצרי עבודה שניתן לבדוק באמצעות בדיקות סטטיות

ניתן לבדוק כמעט את כל תוצרי העבודה באמצעות בדיקות סטטיות. לדוגמה: מסמכים של מפרטי הדרישות, קוד המקור, תוכניות בדיקה (test plans), מקרי בדיקה (test cases), פריטים בצבר המוצר (product backlog items), מסמכי אמנת בדיקות (test charters), מסמכי תיעוד הפרויקט, חוזים ומודלים.

ניתן לבצע סקירות לכל תוצרי העבודה שניתן לקרוא ולהבין. עם זאת, על מנת לבצע ניתוח סטטי, על תוצרי העבודה להיות בעלי תבנית מולה ניתן יהיה לבדוק אותם (לדוגמה: מודלים, קוד או טקסט עם תחביר פורמלי).

תוצרי עבודה שאינם מתאימים לבדיקות סטטיות כוללים את אלו שקשה לפרשם על ידי בני אדם, ושאין לנתח אותם על ידי כלים (לדוגמה: קוד הרצה שהתקבל מספק צד ג' - עקב סיבות משפטיות).

3.1.2 הערך של בדיקות סטטיות

בדיקות סטטיות יכולות לזהות פגמים בשלבים המוקדמים ביותר של מחזור חיי הפיתוח של התוכנה (SDLC) ולממש את העקרון של בדיקות מוקדמות (ראה סעיף 1.3). הן יכולות גם לזהות פגמים שאינם ניתנים לאיתור באמצעות בדיקות דינמיות (לדוגמה: קוד שלא ניתן להגיע אליו, דפוסי עיצוב שאינם מיושמים כנדרש ופגמים בתוצרי עבודה שאינם ניתנים להרצה).

בדיקות סטטיות מספקות את היכולת להעריך את איכות תוצרי העבודה ולבנות את האמון בהם. על ידי אימות הדרישות המתועדות, בעלי העניין יכולים גם לוודא שדרישות אלו תואמות את צרכיהם האמיתיים. מכיוון שהבדיקות הסטטיות נערכות בשלב מוקדם של מחזור חיי פיתוח התוכנה (SDLC), ניתן להגיע להבנה משותפת בין בעלי העניין המעורבים. ובכך התקשורת גם היא תשתפר בין בעלי העניין המעורבים. מסיבה זו מומלץ לערב מגוון רחב של בעלי עניין בבדיקות הסטטיות.

למרות שיתכן כי הסקירות עשויות להיות יקרות ליישום, עלויות הפרויקט הכלליות נמוכות יותר בדרך כלל לעומת מצב בו סקירות אינן מתקיימות כלל, מכיון שיש צורך בפחות זמן ומאמץ לתיקון פגמים במהלך הפרויקט.

ניתן לזהות פגמים בקוד באמצעות ניתוח סטטי ביעילות גבוהה יותר מאשר בבדיקות דינמיות, ובדרך כלל התוצאה היא פחות פגמים בקוד ומאמץ כולל נמוך יותר של הפיתוח.

3.1.3 *הבדלים בין בדיקות סטטיות ובדיקות דינמיות*

בדיקות סטטיות ושיטות בדיקה דינמיות משלימות זו את זו. יש להן מטרות דומות, כגון סיוע בגילוי פגמים בתוצרי עבודה (ראה סעיף 1.1.1), אך ישנם גם הבדלים מסוימים ביניהן, כגון:

- בדיקות סטטיות ודינמיות (יחד עם ניתוח כשלים) יכולות שתיהן להוביל לאיתור פגמים, אולם ישנם סוגים של פגמים שניתן למצוא רק באמצעות בדיקות סטטיות או בדיקות דינמיות
- בדיקות סטטיות מוצאת פגמים באופן ישיר, בעוד שבדיקות דינמיות גורמות לכשלים ורק לאחר שבוצע ניתוח לכשלים, מוגדרים הפגמים הרלוונטיים
- בדיקות סטטיות עשויות לזהות פגמים שנמצאים במסלולי קוד שמריצים אותם לעיתים נדירות או שקשה להגיע אליהם באמצעות בדיקות דינמיות
- ניתן לבצע בדיקות סטטיות על תוצרי עבודה שאינם ניתנים להרצה, בעוד שבדיקות דינמיות ניתנות לביצוע אך ורק על תוצרי העבודה הניתנים להרצה
- ניתן להשתמש בבדיקות סטטיות למדידת מאפייני איכות שאינם תלויים בהרצת קוד (לדוגמה: תחזוקתיות הקוד) בעוד שבבדיקות דינמיות ניתן להשתמש למדידת מאפייני איכות שתלויים בהרצת קוד (לדוגמה: בדיקות ביצועים ועומסים)

פגמים טיפוסיים שקל יותר ו/או זול יותר למצוא באמצעות בדיקות סטטיות כוללים:

- פגמים בדרישות (לדוגמה: חוסר עקביות, אי בהירות, סתירות, חוסרים, אי דיוקים וכפילויות)
- פגמי עיצוב (לדוגמה: מבנים לא יעילים של מסדי הנתונים, מודולריות לקויה)
- סוגים מסוימים של פגמים בקוד (לדוגמה: משתנים עם ערכים בלתי מוגדרים, משתנים ללא הכרזה (undeclared variables), קוד מת (שאינו רץ לעולם) או כפול, מורכבות גבוהה של הקוד)
- חריגות מהסטנדרטים (לדוגמה: אי עמידה במוסכמות של שמות המשתנים והרכיבים בקוד כפי שמופיעים בתקני כתיבת קוד)
- מפרטי ממשקים שגויים (לדוגמה: חוסר התאמה במספר, בסוג או בסדר של הפרמטרים)
- סוגים מסוימים של פרצות אבטחה (לדוגמה: גלישת חוצץ (buffer overflows))
- פערים או אי דיוקים בכיסוי של בסיס הבדיקות (לדוגמה: מחסור בבדיקות עבור קריטריון קבלה (acceptance criterion))

3.2 תהליך משוב וסקירה

3.2.1 *יתרונות המשוב המוקדם והתכוף מבעלי העניין*

משוב מוקדם ותכוף מאפשר זיהוי ותיקשור מוקדם של בעיות איכות אפשריות. במידה וישנה מעורבות מועטה של בעלי העניין במהלך חיי פיתוח התוכנה (SDLC), המוצר המפותח עשוי שלא לעמוד בחזון המקורי או הנוכחי של בעלי העניין. כישלון במסירת מוצר אותו רצו בעלי העניין, יכול לגרום לעבודה חוזרת יקרה, אי עמידה בלוחות זמנים, האשמות הדדיות ואף להוביל לכישלון מוחלט של הפרויקט.

משוב תכוף של בעלי העניין לאורך מחזור חיי פיתוח התוכנה (SDLC) יכול למנוע אי הבנות לגבי הדרישות ולהבטיח ששינויים בדרישות הובנו ויושמו מוקדם יותר. זה עוזר לצוות הפיתוח לשפר את ההבנה שלהם לגבי מה שהם בונים. זה מאפשר להם להתמקד בתכונות שמספקות את הערך הרב ביותר לבעלי העניין ושיש להם יכולת השפעה חיובית ביותר על הסיכונים שזוהו.

3.2.2 *פעילויות בתהליך הסקירה*

התקן ISO/IEC 20246 מגדיר תהליך סקירה כללי שמספק מסגרת מובנית אך גמישה לתהליך סקירה ממנה ניתן לבנות תהליך סקירה ספציפי למצב מסוים. אם הסקירה הנדרשת היא יותר פורמאלית, אז יהיה צורך ביותר מהמשימות המתוארות עבור הפעילויות השונות.

מספר רב של תוצרי עבודה הופכים למכלול רחב וגדול מכדי שסקירה בודדת תוכל לכסות אותם בשלמותם. תהליך הסקירה עשוי להיות מופעל מספר פעמים כדי להשלים את הסקירה על תוצר העבודה כולו.

הפעילויות בתהליך הסקירה הן:

- **תכנון** (**planning**). בשלב התכנון יוגדרו היקף הסקירה, שכולל את המטרה, תוצר העבודה שיש לסקור, מאפייני האיכות שיש לבדוק, איזורים בהם יש להתמקד, קריטריוני יציאה, מידע תומך כגון סטנדרטים, מאמץ נדרש ולוחות זמנים לסקירה.
- **התנעת/ייזום הסקירה** (**Review initiation**). במהלך שלב זה, המטרה היא לוודא שכל הנושאים והמעורבים מוכנים להתחיל בסקירה. מוודאים שלכל משתתף יש גישה לתוצר העבודה שיעבור סקירה, שכל משתתף מבין את תפקידו והאחריות שלו ומקבל את כל הדרוש על מנת שיוכל לבצע את הסקירה.
- **סקירה אישית (Individual review)**. כל בודק מבצע סקירה באופן אישי על מנת להעריך את איכות תוצר העבודה שנבדק ולזהות חריגות/אנומליות, לתת המלצות ולשאול שאלות על ידי שימוש בטכניקות סקירה אחת או יותר (לדוגמה: סקירה מבוססת רשימות בדיקה (checklist-based reviewing), סקירה מבוססת תרחישים (scenario-based reviewing)). התקן ISO/IEC 20246 מכיל פירוט נוסף בהקשר לטכניקות הסקירה השונות. הסוקרים רושמים את כל החריגות/האנומליות, ואת כל ההמלצות והשאלות שזיהו.
- **תקשורת וניתוח** (**Communication and analysis**). מאחר והחריגות/האנומליות שזוהו במהלך הסקירה האינדיבידואלית אינן בהכרח פגמים, יש לנתח ולדון בחריגות הללו. לכל חריגה/אנומליה, יש לקבוע את מצבה (status), מי אחראי לטפל בה והפעולות הנדרשות עבורה. ניתוח זה נעשה בדרך כלל בפגישת סקירה, שבמהלכה המשתתפים גם מחליטים על רמת האיכות של תוצר העבודה שנבדק ואילו פעולות המשך נדרשות. לעיתים יש לבצע סקירת המשך להשלמת שאר הפעולות.
- **תיקון ודיווח** (**Fixing and reporting**). עבור כל פגם, יש ליצור דו"ח פגם על מנת שניתן יהיה לעקוב אחר פעילויות התיקון. לאחר שהושגו קריטריוני היציאה ניתן לאשר את תוצר העבודה. תוצאות הסקירה מדווחות.

3.2.3 *תפקידים ותחומי אחריות בסקירות*

בסקירות מעורבים מגוון של בעלי עניין, שיכולים לשמש במספר תפקידים. להלן התפקידים הראשיים והאחריות שלהם:

- מנהל (Manager) - מחליט מה ייסקר ומקצה משאבים, כגון צוות וזמן, לצורך הסקירה
- מחבר (Author) - יוצר ומתקן את תוצר העבודה שעובר סקירה
- מנחה (Moderator, Facilitator) - מבטיח הפעלה יעילה של מפגשי הסקירה, כולל גישור, ניהול זמן ויצירת סביבת סקירה בטוחה בה כל אחד יכול לדבר בחופשיות
- רשם/מקליט (Scribe, Recorder) - מתעד חריגות/אנומליות שדווחו על ידי הסוקרים ורושם מידע על מהלך הסקירה, כגון החלטות וחריגות/אנומליות חדשות שנמצאו במהלך מפגש הסקירה
- סוקר (Reviewer) – מבצע סקירות. סוקר יכול להיות כל אחד מאלו שעובדים בפרויקט, מומחה בתחום או כל בעל עניין אחר
- מוביל הסקירה (Review leader) – בעל אחריות כוללת על תהליך הסקירה, כגון: החלטה מי יהיה מעורב, וארגון של מתי ואיפה תתקיים הסקירה

אפשרי למצוא תפקידים אחרים ומפורטים יותר, כמתואר בתקן ISO/IEC 20246.

3.2.4 *סוגי סקירות*

קיימים סוגי סקירות רבים החל מסקירה לא פורמאלית (לא רשמית) ועד לסקירות פורמליות (רשמיות). רמת הפורמאליות הנדרשת תלויה בגורמים כגון מודל מחזור חיי התוכנה (SDLC) שבשימוש, בשלות תהליך הפיתוח, מידת הקריטיות והמורכבות של תוצר העבודה שנסקר, דרישות חוקיות או רגולטוריות, והצורך בתיעוד תהליך הסקירה. אותו תוצר עבודה יכול להיבדק באמצעות סוגי סקירה שונים, לדוגמה: תחילה בסקירה לא פורמלית ולאחר מכן בסקירה פורמלית יותר.

בחירת סוג הסקירה הנכון הינה גורם מרכזי בהשגת מטרות הסקירה הנדרשות (ראו סעיף 3.2.5). הבחירה אינה מתבצעת רק על פי המטרות, אלא גם על פי גורמים כגון צרכי הפרויקט, המשאבים הזמינים, סוג תוצר העבודה והסיכונים, התחום העסקי והתרבות הארגונית.

כמה סוגי סקירות נפוצים:

- **סקירה לא פורמאלית (Informal review).** סקירות לא פורמליות אינן עוקבות אחר תהליך מוגדר ואינן מחייבות תיעוד פורמאלי. המטרה העיקרית הינה זיהוי חריגות/אנומליות.
- **דיון מודרך (Walkthrough).** דיון מודרך, אותו מוביל המחבר (author), יכול לשרת מטרות רבות, כגון: הערכת איכות ובניית אמון בתוצר העבודה, הכשרה ולימוד של הסוקרים, השגת הסכמה כוללת, יצירת רעיונות חדשים, העלאת מוטיבציה ומתן אפשרות למחברים לשפר ולאתר חריגות/אנומליות. הסוקרים יכולים לבצע סקירה פרטנית/אישית לפני הדיון המודרך, אך אין חובה שזה יתבצע בדיוק כך.

- **סקירה טכנית (Technical Review).** סקירה טכנית מבוצעת על ידי סוקרים שהם טכניים בהכשרתם ומנוהלת על ידי מנחה (moderator). מטרות הסקירה הטכנית הן להגיע להסכמה ולקבל החלטות לגבי בעיה טכנית, אך גם לזהות חריגות/אנומליות, להעריך את האיכות ולבנות אמון בתוצר העבודה, לייצר רעיונות חדשים ולהעלות מוטיבציה ולאפשר למחברים להשתפר.

- **ביקורת (Inspection).** ביקורות הן הסוג הפורמלי ביותר של סקירות והן עוקבות אחר תהליך הסקירה הכללי במלואו (ראו סעיף 3.2.2). המטרה העיקרית היא למצוא את המספר הגדול ביותר של החריגות/האנומליות. מטרות נוספות הן להעריך את האיכות, לבנות אמון בתוצר העבודה, להעלות מוטיבציה ולאפשר למחברים להשתפר. מדדים נאספים ונעשה בהם שימוש לשיפור מחזור חיי פיתוח התוכנה ,(SDLC) כולל תהליך הסקירה עצמו. במהלך הביקורת, המחבר אינו יכול לשמש כמוביל הסקירה או כרשם.

3.2.5 *גורמי הצלחה לסקירות*

ישנם מספר גורמים הקובעים את הצלחת הסקירות:

- הגדרת מטרות ברורות וקריטריוני יציאה מדידים. הערכת המשתתפים בסקירה לעולם איננה צריכה להיות אחת ממטרות הסקירה
- בחירת סוג הסקירה המתאים לשם השגת המטרות, ובהתאמה לסוג תוצר העבודה, המשתתפים בסקירה, צרכי הפרויקט וההקשר
- ביצוע סקירות על נתחים קטנים של תוכן, על מנת שהסוקרים לא יאבדו ריכוז במהלך סקירה פרטנית ו/או במהלך פגישת הסקירה (כאשר היא מתקיימת)
- מתן משוב מהסוקרים לבעלי העניין ולמחברים כך שיוכלו לשפר את התוצר ואת פעילויותיהם (ראו סעיף 3.2.1)
- מתן כמות זמן מספקת למשתתפים להתכונן לסקירה
- תמיכה מההנהלה בביצוע תהליך הסקירה
- הפיכת הסקירות לחלק מהתרבות הארגונית, על מנת לקדם למידה ושיפור תהליכים
- מתן הכשרה מספקת לכל המשתתפים, כך שידעו כיצד למלא את תפקידם
- הנחיית הפגישות

4 ניתוח ועיצוב בדיקות — 390 דקות

מושגים

בדיקות חקרניות (exploratory testing), בדיקות מבוססות רשימת תיוג (checklist-based testing), בדיקת טבלת החלטות (decision table testing), בדיקת מעבר בין מצבים (state transition testing), גישת בדיקה מבוססת שיתוף פעולה (collaboration-based test approach), טכניקת בדיקה (test technique), טכניקת בדיקות מבוססת ניסיון (experience-based test technique), טכניקת בדיקת קופסה לבנה (white-box test technique), טכניקת בדיקת קופסה שחורה (black-box test technique), כיסוי (coverage), כיסוי הצהרות (statement coverage), כיסוי ענפים (branch coverage), מחלקות שקילות (equivalence partitioning), ניחוש שגיאות (error guessing), ניתוח ערכי גבול (boundary value analysis), פיתוח מונחה בדיקות קבלה (acceptance test-driven development), פריט כיסוי (coverage item), קריטריוני קבלה (acceptance criteria)

מטרות למידה לפרק 4

4.1 טכניקות בדיקה – סקירה כללית

FL-4.1.1 (K2) הבחן בין טכניקות בדיקה מסוג קופסה שחורה, לבנה וטכניקות בדיקה מבוססות ניסיון

4.2 טכניקות בדיקה מסוג קופסה שחורה

FL-4.2.1 (K3) השתמש במחלקות שקילות כדי לגזור מקרי בדיקה

FL-4.2.2 (K3) השתמש בניתוח ערכי גבול כדי לגזור מקרי בדיקה

FL-4.2.3 (K3) השתמש בבדיקת טבלת החלטות כדי לגזור מקרי בדיקה

FL-4.2.4 (K3) השתמש בבדיקת הֶחְלֵף מצבים כדי לגזור מקרי בדיקה

4.3 טכניקות בדיקה מסוג קופסה לבנה

FL-4.3.1 (K2) הסבר על בדיקות משפטים

FL-4.3.2 (K2) הסבר על בדיקות ענפים

FL-4.3.3 (K2) הסבר את הערך (the value) של בדיקה מסוג קופסה לבנה

4.4 טכניקות בדיקה מבוססות ניסיון

FL-4.4.1 (K2) הסבר על ניחוש שגיאות

FL-4.4.2 (K2) הסבר על בדיקות חוקרות

FL-4.4.3 (K2) הסבר על בדיקות מבוססות רשימות תיוג

4.5 גישות בדיקה מבוססות שיתוף פעולה

FL-4.5.1 (K2) הסבר כיצד לכתוב סיפורי משתמשים בשיתוף עם מפתחים ונציגים מהצד העסקי

FL-4.5.2 (K2) סווג את האפשרויות השונות לכתיבת קריטריוני קבלה

FL-4.5.3 (K3) השתמש בפיתוח מונחה בדיקות קבלה (ATDD) לגזירת מקרי בדיקה

4.1 טכניקות בדיקה - סקירה

טכניקות בדיקה מסייעות לבודק התוכנה בניתוח הבדיקות (מה לבדוק) ובתכנון בדיקה (איך לבדוק). טכניקות הבדיקה עוזרות לפתח (להגדיר) קבוצה קטנה יחסית, אך מספקת, של מקרי בדיקה בצורה שיטתית. טכניקות הבדיקה עוזרות לבודק התוכנה להגדיר תנאי בדיקה, לזהות את פריטי הכיסוי ולזהות נתוני בדיקה במהלך ניתוח ועיצוב הבדיקה. מידע נוסף על טכניקות בדיקה והתאמתן ניתן למצוא בתקן ISO/IEC/IEEE 29119-4, וב- (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

בסילבוס זה, טכניקות הבדיקה מסווגות כטכניקות מסוג קופסה שחורה, טכניקות מסוג קופסה לבנה וטכניקות מבוססות ניסיון.

טכניקות בדיקה מסוג קופסה שחורה (ידועות גם בשם טכניקות מבוססות מפרטים) מבוססות על ניתוח ההתנהגות המצופה של אובייקט הבדיקה ללא התייחסות למבנה הפנימי שלו. לכן, מקרי הבדיקה אינם תלויים באופן יישום התוכנה. כתוצאה מכך, אם המימוש משתנה אבל ההתנהגות הנדרשת נשארת זהה, מקרי הבדיקה עדיין שימושיים.

טכניקות בדיקה מסוג קופסה לבנה (הידועות גם בשם טכניקות מבוססות מבנה) מבוססות על ניתוח המבנה של אובייקט הבדיקה ופעולות העיבוד שממומשות בו. מכיוון שמקרי הבדיקה תלויים באופן מימוש התוכנה, ניתן ליצור אותם רק לאחר התכנון או היישום של אובייקט הבדיקה.

טכניקות בדיקה מבוססות ניסיון משתמשות בפועל בידע ובניסיון של בודקים עבור תכנון ויישום מקרי בדיקה. היעילות של טכניקות אלה תלויה במידה רבה בכישורי הבודק. טכניקות בדיקה מבוססות ניסיון יכולות לזהות פגמים שעלולים להחמיץ אותם באמצעות שימוש בטכניקות בדיקה מסוג קופסה שחורה ולבנה. לפיכך, טכניקות בדיקה מבוססות ניסיון משלימות את טכניקות בדיקה מסוג קופסה שחורה וקופסה לבנה.

4.2 טכניקות בדיקה מסוג קופסה שחורה

טכניקות בדיקה מסוג קופסה שחורה הנפוצות שנדונות בסעיפים הבאים הן:

- מחלקות שקילות
- ניתוח ערכי גבול
- בדיקת טבלת החלטות
- בדיקת מעבר בין מצבים

4.2.1 מחלקות שקילות

חלוקה למחלקות שקילות (EP) מחלקת נתונים לקבוצות (הידועות כמחלקות שקילות) על סמך הציפייה כי כל הרכיבים של קבוצה נתונה יעובדו באופן זהה ויציגו התנהגות זהה בתוצאה הסופית. התיאוריה מאחורי טכניקה זו היא שאם מקרה בדיקה בודק ערך אחד מתוך מחלקת השקילות ומזהה פגם, פגם זה צריך להיות מזוהה גם על ידי מקרי בדיקה הבודקים כל ערך אחר באותה המחלקה. לכן מספיקה בדיקה אחת לכל מחלקת שקילות[10].

ניתן לזהות מחלקות שקילות עבור כל רכיב נתונים הקשור לאובייקט הבדיקה, כולל קלט, פלטים, פריטי תצורה, ערכים פנימיים וערכים הקשורים לזמן ולפרמטרים של הממשק. המחלקות יכולות להיות רציפות

[10] הערת עורך – ראה: "נספח ד – הבהרות מיוחדות (עבור המהדורה בעברית בלבד)"

או ללא רצף, מסודרות או לא מסודרות, סופיות או אינסופיות. אסור שתהיינה מחלקות ריקות (שלא מכילות אף ערך).

עבור אובייקטי בדיקה פשוטים חלוקה למחלקות שקילות יכולה להיות קלה, אבל בפועל, להבין איך אובייקט הבדיקה יטפל בערכים שונים עלול להיות מסובך. לכן, החלוקה לקבוצות שקילות צריכה להיעשות בתשומת לב.

מחלקה המכילה ערכים חוקיים נקראת מחלקה חוקית. מחלקה המכילה ערכים לא חוקיים נקראת מחלקה לא חוקית. ההגדרות של ערכים חוקיים ובלתי חוקיים עשויות להשתנות בין צוותים וארגונים. לדוגמה, ערכים חוקיים עשויים להתפרש ככאלה שאמורים להיות מעובדים על ידי האובייקט הנבדק או ככאלה שעבורם המפרט מגדיר את עיבודם. ערכים לא חוקיים עשויים להתפרש ככאלה שהמערכת תתעלם או תדחה אותם על ידי האובייקט הנבדק או ככאלה שלא הוגדר עבורם עיבוד במפרט האובייקט הנבדק.

במחלקות שקילות (ב-EP), פריטי הכיסוי הם מחלקות השקילות. כדי להשיג 100% כיסוי עם טכניקה זו, מקרי בדיקה חייבים לממש את כל המחלקות שזוהו (כולל מחלקות לא חוקיות) על ידי כיסוי כל מחלקה לפחות פעם אחת. הכיסוי נמדד במספר המחלקות המופעלות על ידי לפחות מקרה בדיקה אחד, מחולק לפי המספר הכולל של המחלקות שזוהו, והוא מבוטא באחוזים.

אובייקטי בדיקה רבים כוללים מספר קבוצות של מחלקות (למשל, אובייקטי בדיקה עם יותר מפרמטר קלט אחד), כלומר, מקרה בדיקה יכסה ערכים מקבוצות שקילות שונות. קריטריון הכיסוי הפשוט ביותר במקרה של מספר קבוצות של מחלקות נקרא כיסוי - "כל אפשרות" (Ammann 2016). כיסוי "כל אפשרות" דורש מקרי בדיקה למימוש כל מחלקה מכל סט של מחלקות לפחות פעם אחת. כיסוי "כל אפשרות" אינו לוקח בחשבון שילובים של מחלקות.

4.2.2 *ניתוח ערכי גבול*

ניתוח ערכי גבול (BVA) היא טכניקה המבוססת על שימוש בגבולות של מחלקות השקילות. לכן, ניתן להשתמש ב-BVA רק עבור מחלקות שקילות שהוגדרו. ערכי המינימום והמקסימום של מחלקת שקילות הם ערכי הגבול שלה. במקרה של ערכי גבול, אם שני אלמנטים שייכים לאותה מחלקה, כל האלמנטים ביניהם חייבים להיות שייכים גם למחלקת שקילות זו.

ניתוח ערכי גבול BVA מתמקד בערכי הגבול של המחלקות מכיוון שמפתחים נוטים יותר לבצע שגיאות בנקודות אלו.

פגמים אופייניים שנמצאו על ידי ניתוח ערכי גבול ממוקמים במקום שבו הגבולות המיושמים אינם ממוקמים מעל או מתחת למיקומם המיועד או שהם מושמטים לחלוטין.

סילבוס זה מכסה שתי גרסאות של ניתוח ערכי גבול: ניתוח ערכי גבול בעל 2 ערכים וניתוח ערכי גבול בעל 3 ערכים. הם נבדלים במונחים של פריטי כיסוי לכל גבול שיש לממש כדי להשיג 100% כיסוי.

בניתוח ערכי גבול בעל 2 ערכים (טווח) (Craig 2002, Myers 2011), לכל ערך גבול ישנם שני פריטי כיסוי: ערך הגבול של המחלקה והערך של השכן הקרוב ביותר שלו השייך למחלקת שקילות סמוכה. כדי להשיג 100% כיסוי בניתוח ערכי הגבול של 2 ערכים, מקרי הבדיקה חייבים לממש את כל פריטי הכיסוי, כלומר, את כל ערכי הגבול המזוהים. הכיסוי נמדד במספר ערכי הגבול שהופעלו, חלקי המספר הכולל של ערכי הגבול שזוהו, והוא מבוטא באחוזים.

בניתוח ערכי גבול בעל 3 ערכים (Koomen 2006, O'Regan 2019), לכל ערך גבול ישנם שלושה פריטי כיסוי: נקודת הגבול ושני הערכים הצמודים לה מימין ומשמאל. לכן, בניתוח ערכי גבול מסוג זה ייתכן שחלק מפריטי הכיסוי אינם נקודות גבול. כדי להשיג כיסוי של 100% בניתוח ערכי גבול בעל 3 ערכים, מקרי הבדיקה חייבים לממש את כל פריטי הכיסוי, כלומר נקודת הגבול המזוהה והנקודות

הצמודות לה - נקודה אחת מתחת ואחת מעל(הערך הצמוד מימין והערך הצמוד משמאל). הכיסוי נמדד במספר ערכי הגבול ושכניהם המופעלים, חלקי המספר הכולל של ערכי הגבול המזוהים ושכניהם, והוא מבוטא באחוזים.

מימוש טכניקת ניתוח ערכי גבול בעלת 3 ערכים הינה קפדנית יותר מאשר טכניקת ניתוח ערכי גבול בעלת 2 ערכים, מכיון שבטכניקה זו ניתן לזהות פגמים שאנו עלולים לפספס או להתעלם מהם בניתוח ערכי גבול בעל 2 ערכים.

לדוגמה, אם ההחלטה:

"אם (X<=10)..." תמומש באופן שגוי כ"אם (X=10)...", לא יגזרו נתוני בדיקה מנקודות גבול בעלת 2-ערכים (X=10, X=11) ובכך לא יזהו את הפגם. לעומת זאת הערך X=9 שנגזר מנקודת גבול בעלת 3-ערכים, צפוי לזהות אותו.

4.2.3 *בדיקות טבלת החלטה*

טבלאות החלטה משמשות לבדיקת יישום דרישות המערכת שמפרטות כיצד שילובים שונים של תנאים מובילים לתוצאות שונות. טבלאות החלטה הן הדרך היעילה לרישום לוגיקה מורכבת כגון כללים/חוקים עסקיים.

בעת יצירת טבלאות החלטה, מוגדרים התנאים והפעולות (התוצאות הסופיות) של המערכת. אלה יוצרים את שורות הטבלה. כל תנאי מייצג שורה בטבלה וכל עמודה תואמת לכלל החלטה המגדירה שילוב ייחודי של תנאים, יחד עם הפעולות (התוצאות) הנלוות. בטבלאות החלטה פשוטות (המורכבת מתנאים פשוטים) כל הערכים של התנאים והפעולות (התוצאות הסופיות) (למעט תנאים לא רלוונטיים או בלתי אפשריים; ראה להלן) מוצגים כערכים בוליאניים (אמת או שקר). לחילופין, בטבלאות החלטה מורחבות עשויים חלק מהתנאים והפעולות או כולם לקבל גם ערכים מרובים (למשל, טווחי מספרים, מחלקות שקילות, ערכים נפרדים).

הסימון לתנאים הוא כדלקמן: "T" (נכון) פירושו שהתנאי מתקיים. "F" (לא נכון) פירושו שהתנאי לא מתקיים. "–" פירושו שערך התנאי אינו רלוונטי לפעולת התוצאה. "לא רלוונטי" פירושו שהתנאי אינו בר ביצוע עבור כלל (rule) נתון. עבור פעולות: "X" אומר שהפעולה צריכה להתרחש. ריק פירושו שהפעולה לא צריכה להתרחש. ניתן להשתמש גם בסימונים אחרים.

טבלת החלטות מלאה מכילה את כלל העמודות המכסות את כלל שילובי התנאים. הטבלה ניתנת למיקוד ודיוק באמצעות הסרת/מחיקת עמודות המכילות שילובים בלתי אפשריים של תנאים. הטבלה עשויה להצטמצם באמצעות מיזוג עמודות, שבהן תנאים מסוימים אינם משפיעים על התוצאה. אלגוריתמים למיזעור טבלת החלטה אינם נידונים במסגרת סילבוס זה.

בבדיקת טבלת החלטה, פריטי הכיסוי הם העמודות המכילות שילובים אפשריים של תנאים. כדי להשיג כיסוי של 100% בטכניקה זו, מקרי הבדיקה חייבים לממש את כל העמודות הללו. הכיסוי נמדד כמספר העמודות הממומשות, חלקי המספר הכולל של העמודות האפשריות, והוא מבוטא באחוזים.

החוזק של בדיקה באמצעות טבלת החלטות היא בכך שהיא מספקת גישה שיטתית לזיהוי כל שילובי התנאים, שבמצבים אחרים עלולים להתעלם מחלקם. זה גם עוזר למצוא פערים או סתירות בדרישות. אם ישנם תנאים רבים, מימוש כל כללי ההחלטה עלול לארוך זמן רב, מכיוון שמספר הכללים גדל באופן אקספוננציאלי עם מספר התנאים. במקרה כזה, כדי לצמצם את מספר הכללים שיש להפעיל, ניתן להשתמש בטבלת החלטות ממוזערת או בגישה מבוססת סיכונים.

4.2.4 בדיקות הֶחְלֵף מצבים

תרשים (דיאגרמת) הֶחְלֵף מצבים[11] מייצגת את ההתנהגות של מערכת על ידי הצגת המצבים האפשריים של המערכת ואת המעברים התקפים בה. מעבר ממצב אחד לאחר נגרם על ידי אירוע, אשר עשוי להיות מוגדר באמצעות תנאי סף. ההנחה היא שהמעברים הם מיידיים ולעיתים גורמים לפעולה המבוצעת על ידי התוכנה הנבדקת. התחביר הנפוץ לציון המעבר הוא כדלקמן: "אירוע [תנאי סף] / פעולה". ניתן להשמיט תנאי הסף או את הפעולה אם הן אינם קיימים או אינם רלוונטיים עבור הבודק.

טבלת מצבים היא מודל שווה ערך לתרשים הֶחְלֵף מצבים. השורות מייצגות מצבים, והעמודות מייצגות אירועים (יחד עם התנאים, במידה והם קיימים). ערכי הטבלה (תאים בטבלה) מייצגים מעברים, ומכילים את מצב היעד, כמו גם את הפעולות המתקבלות, אם מוגדרות. בניגוד לתרשים הֶחְלֵף מצבים, טבלת המצבים מציגה במפורש גם מעברים לא חוקיים, המיוצגים על ידי תאים ריקים.

מקרה בדיקה המבוסס על תרשים הֶחְלֵף מצבים או טבלת מצבים מיוצג בדרך כלל כרצף של אירועים, אשר מביאים לרצף של שינויי מצב (ופעולות, במידת הצורך). מקרה בדיקה אחד עשוי, ובדרך כלל יכסה מספר מעברים בין מצבים.

קיימים קריטריוני כיסוי רבים לבדיקת מעבר בין מצבים. סילבוס זה דן בשלושה מהם.

בכיסוי של כל המצבים, פריטי הכיסוי הם המצבים. כדי להשיג 100% כיסוי של כל המצבים, מקרי הבדיקה חייבים להבטיח שכל המצבים יבדקו. הכיסוי נמדד כמספר המצבים בהן עברו (ביקרו) חלקי המספר הכולל של המצבים, והוא מבוטא באחוזים.

בכיסוי מעברים חוקי (נקרא גם כיסוי switch-0), פריטי הכיסוי התקפים הם כל המעברים הבודדים. כדי להשיג כיסוי מעברים תקף של 100%, מקרי בדיקה חייבים לממש את כל המעברים התקפים. הכיסוי נמדד כמספר המעברים התקפים שעברו בהם (שהופעלו) חלקי המספר הכולל של המעברים התקפים, והוא מבוטא באחוזים.

בכיסוי כל המעברים, פריטי הכיסוי הם כל המעברים המוצגים בטבלת המצבים. כדי להשיג 100% כיסוי כל המעברים, מקרי הבדיקה חייבים לממש את כל המעברים התקפים ולנסות לבצע מעברים לא חוקיים. בדיקת מעבר לא חוקי אחד בלבד - במקרה בדיקה בודד - עוזרת למנוע מיסוך/הסתרת תקלות, כלומר, מצב בו פגם אחד מונע את גילויו של האחר. הכיסוי נמדד כמספר המעברים התקפים והבלתי חוקיים שעברו בהם (שהופעלו) או שהיה ניסיון לכסות אותם במקרי הבדיקה שבוצעו, חלקי המספר הכולל של המעברים החוקיים והבלתי חוקיים, והוא מבוטא באחוזים.

הכיסוי של כל המצבים חלש יותר מכיסוי מעברים תקף, מכיוון שבדרך כלל ניתן להשיג אותו ללא מימוש של כל המעברים. כיסוי מעברים תקפים הוא קריטריון הכיסוי הנפוץ ביותר. השגת כיסוי מעברים תקפים במלואם מבטיחה כיסוי מלא של כל המעברים .בהשגת כיסוי של כל המעברים במלואם הכיסוי מבטיח הן כיסוי מלא של כל המצבים והן כיסוי מלא של מעברים תקפים ועליו לשמש כדרישת סף (מינימום) עבור תוכנה שהיא קריטית למשימות ובטיחות.

[11] הערת עורך - הֶחְלֵף מצבים = מעבר בין מצבים

4.3 טכניקות בדיקה מסוג קופסה לבנה

בגלל הפופולריות והפשטות שלהם, סעיף זה מתמקד בשתי טכניקות בדיקה של קופסה לבנה הקשורות לקוד:

- בדיקות משפטים
- בדיקות ענפים

ישנן טכניקות קפדניות יותר המשמשות בבדיקות סביבות (מערכות) קריטיות שבהן יש צורך בכיסוי יסודי יותר של הקוד .ישנן גם טכניקות בדיקה מסוג קופסה לבנה שניתנות לשימוש ברמות בדיקה גבוהות יותר (לדוגמה, בדיקת API), או שימוש בכיסוי שאינו קשור לקוד (למשל, כיסוי נוירונים ב- בדיקת רשתות עצביות). טכניקות אלו אינן נדונות בסילבוס זה.

4.3.1 בדיקות משפטים (פקודות) וכיסוי משפטים (פקודות)

בבדיקת משפטים, פריטי הכיסוי הם משפטים (פקודות) הניתנים (הניתנות) להפעלה. המטרה היא לתכנן מקרי בדיקה שמשתמשים במשפטים שבקוד עד להשגת רמת כיסוי מקובלת. הכיסוי נמדד כמספר המשפטים שהופעלו על ידי מקרי הבדיקה חלקי המספר הכולל של המשפטים הניתנים להפעלה בקוד, והוא מבוטא באחוזים.

כאשר מושג כיסוי של 100% של המשפטים, זה מבטיח שכל המשפטים בקוד הורצו בבדיקות לפחות פעם אחת. זה אומר שכל הצהרה עם פגם תיבדק, דבר שעשוי לגרום לכשל המעיד על קיומו של הפגם. עם זאת, מימוש הצהרה באמצעות מקרה בדיקה לא יזהה פגמים בכל המקרים. לדוגמה, ייתכן שהוא לא יזהה פגמים שהם תלויי נתונים (למשל, חלוקה באפס שנכשלת רק כאשר המכנה שווה לאפס). כמו כן, 100% כיסוי הצהרות אינו מבטיח שכל היגיון ההחלטות נבדק, לדוגמה, ייתכן שלא נממש את כל הענפים (ראה פרק 4.3.2) בקוד.

4.3.2 בדיקת ענפים וכיסוי ענפים (Branch Coverage)

ענף הינו העברת שליטה[12] בין שני צמתים בתרשים הזרימה, המראה את אפשרויות הרצפים שבהם הפקודות שבקוד המקור מבוצעות באובייקט הבדיקה. כל העברת שליטה יכולה להיות העברה בלתי מותנית (כלומר, קוד המבוצע בקו ישר) או מותנית (כלומר, תוצאה של החלטה).

בבדיקת ענפים פריטי הכיסוי הם הענפים והמטרה היא לעצב מקרי בדיקה למימוש הענפים בקוד עד להשגת רמת כיסוי מקובלת. הכיסוי נמדד כמספר הענפים שעברו בהם (שהופעלו) על ידי מקרי הבדיקה חלקי המספר הכולל של הענפים בקוד, והוא מבוטא באחוזים.

בהשגת כיסוי של 100%, כל הענפים בקוד, מותנים ולא-מותנים, מופעלים על ידי מקרי הבדיקה. ענפים מותנים בדרך כלל קשורים לתוצאת "אמת" או "שקר" (True or False) מנקודת החלטה של "אם...אז", תוצאה של משפט switch/case או החלטה לצאת או להמשיך בלולאה.

עם זאת, הפעלת ענף באמצעות מקרה בדיקה לא יאתר ליקויים בכל המקרים. לדוגמה, ייתכן שלא נזהה פגמים הדורשים ביצוע של נתיב ספציפי בקוד.

כיסוי ענפים מתבסס על כיסוי משפטים (פקודות). המשמעות היא שכל קבוצה של מקרי בדיקה שמשיגה 100% של כיסוי ענפים, משיגה גם כיסוי של 100% משפטים (אך לא להיפך).

[12] הערת עורך – ראה: "נספח ד – הבהרות מיוחדות (עבור המהדורה בעברית בלבד)"

4.3.3 *הערך של בדיקות קופסה לבנה*

החוזק הבסיסי שכל טכניקות הקופסה הלבנה חולקות הוא, שהטמעת התוכנה כולה נלקחת בחשבון במהלך הבדיקה, מה שמקל על זיהוי פגמים גם כאשר מפרט התוכנה מעורפל, מיושן או לא שלם. במקביל החולשה של בדיקות מסוג קופסה לבנה היא שאם התוכנה לא מיישמת דרישה אחת או יותר, ייתכן שבדיקת קופסה לבנה לא תזהה את הפגמים הנובעים מההשמטה (Watson 1996).

ניתן להשתמש בטכניקות מסוג קופסה לבנה בבדיקות סטטיות (למשל, במהלך ריצות יבשות של קוד). הן מתאימות היטב גם לסקירת קוד שעדיין לא מוכן להרצה (Hetzel 1988), כמו גם פסאודוקוד (pseudocode) ולוגיקה אחרת ברמה גבוהה (high-level) או מלמעלה למטה (top-down) שניתן לעצב באמצעות תרשים זרימה.

ביצוע בדיקות מסוג קופסה שחורה בלבד, אינו מספק מדד לכיסוי הקוד בפועל. מדדי כיסוי לבדיקות מסוג קופסה לבנה מספקים מדידה אובייקטיבית של הכיסוי ומספקים את המידע הדרוש כדי לאפשר יצירת בדיקות נוספות ולהגדיל את הכיסוי, ולאחר מכן להגדיל את האמון בקוד.

4.4 *טכניקות בדיקה מבוססות ניסיון*

טכניקות בדיקה מבוססות ניסיון הנפוצות שנדונות בסעיפים הבאים הן:

- ניחוש שגיאות
- בדיקות חוקרות
- בדיקה המבוססת על רשימות בדיקה

4.4.1 *ניחוש שגיאות*

ניחוש שגיאות הינה טכניקה המשמשת לחזות את התרחשותם של שגיאות, פגמים וכשלים, בהתבסס על הידע של הבודק, ובעיקר כוללת:

- איך האפליקציה עבדה בעבר?
- סוגי הטעויות שהמפתחים נוטים לעשות וסוגי התקלות הנובעות משגיאות אלו
- סוגי הכשלים שהתרחשו ביישומים אחרים, או דומים

באופן כללי, שגיאות, פגמים וכשלים עלולים להיות קשורים לקלט (למשל, קלט נכון לא מתקבל, פרמטרים שגויים או חסרים), לפלט (לדוגמה, פורמט שגוי, תוצאה שגויה), ללוגיקה (לדוגמה, מקרים חסרים, אופרטורים שגויים), לחישוב (לדוגמה ,נתון שגוי, חישוב שגוי), לממשקים (לדוגמה, פרמטר לא תואם, סוגים לא תואמים), או לנתונים (לדוגמה, אתחול שגוי, טיפוס נתונים שגוי).

מתקפת תקלות (fault attacks) היא גישה שיטתית ליישום ניחוש שגיאות. טכניקה זו דורשת מהבודק ליצור רשימה של שגיאות, פגמים וכשלים אפשריים, ולתכנן בדיקות שיאפשרו לזהות פגמים הקשורים לשגיאות, לחשוף את הליקויים או לגרום לכשלים. רשימות אלה יכולות להיבנות על סמך ניסיון, נתוני פגמים וכשלים או מתוך ידע נפוץ על הסיבה לכשל בתוכנה.

ראה (Whittaker 2002, Whittaker 2003, Andrews 2006) למידע נוסף על ניחוש שגיאות ומתקפת תקלות.

4.4.2 בדיקות חוקרות

בבדיקות חוקרות, הבדיקות מתוכננות, מבוצעות ומוערכות בו זמנית בזמן שהבודק לומד את אובייקט הבדיקה. הבדיקות משמשות כדי ללמוד יותר על אובייקט הבדיקה, כדי לחקור אותו יותר לעומק באמצעות בדיקות ממוקדות וליצור בדיקות לאיזורים שלא נבדקו.

כדי לבנות את הבדיקה בדיקות חוקרות מבוצעות לפעמים באמצעות בדיקות מבוססות מפגשים. בגישה מבוססת מפגשים, הבדיקות החוקרות נערכות במפגשים תחומים בזמן (time-boxed). הבודק משתמש באמנת בדיקה (Test Charter) המכילה יעדי בדיקה כדי להנחות את הבדיקה. לאחר הבדיקה מתקיים בדרך כלל תחקיר הכולל דיון בין הבודק לבין בעלי עניין המעוניינים בתוצאות הבדיקה. בגישה זו ניתן להתייחס ליעדי הבדיקה כאל תנאי בדיקה ברמה גבוהה. פריטי הכיסוי מזוהים ומופעלים במהלך הבדיקה. הבודק עשוי להשתמש בדפי תיעוד לבדיקה על מנת לתעד את השלבים שבוצעו ואת התוצאות שהתגלו.

בדיקות חוקרות מועילות כאשר יש מפרטים מועטים או לא מספקים או כשיש לחץ זמן משמעותי על הבדיקות. בדיקות חוקרות מועילות גם כדי להשלים בדיקות שהתבצעו בטכניקות בדיקות רשמיות שמקובלות. בדיקות חוקרות תהינה אפקטיביות יותר אם הבודק מנוסה, בעל ידע בתחום ובעל רמה גבוהה של מיומנויות חיוניות, כמו כישורים אנליטיים, סקרנות ויצירתיות (ראה סעיף 1.5.1).

בדיקות חוקרות יכולות לשלב שימוש בטכניקות בדיקה אחרות (למשל, קבוצות שקילות). יותר מידע על בדיקות חוקרות ניתן למצוא ב- (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3 בדיקה המבוססת על רשימות בדיקה (תיוג)

בבדיקות מבוססות רשימות בדיקה, הבודק מתכנן, מיישם ומבצע בדיקות כדי לכסות את תנאי הבדיקות מרשימת התיוג. ניתן לבנות רשימות תיוג על סמך הניסיון, הידע על מה שחשוב למשתמש, או ההבנה מדוע וכיצד התוכנה נכשלת. רשימות תיוג לא אמורות להכיל פריטים שיכולים להיבדק אוטומטית, פריטים המתאימים יותר כקריטריוני כניסה/יציאה או פריטים כלליים מדי (Brykczynski 1999).

פריטי הרשימה מנוסחים לרוב בצורה של שאלה. צריך שיהיה אפשרי לבדוק כל פריט בנפרד ובאופן ישיר. פריטים אלה עשויים להתייחס לדרישות, מאפייני ממשק גרפי, איכות מאפיינים או צורות אחרות של תנאי בדיקה. ניתן ליצור רשימות כדי לתמוך בסוגי בדיקות שונים, כולל בדיקות פונקציונליות ולא פונקציונליות (למשל, 10 היוריסטיקות לבדיקת שמישות (Nielsen 1994)).

רשימות מסוימות עם הזמן עשויות להפוך בהדרגה פחות אפקטיביות מכיוון שהמפתחים ילמדו כדי להימנע מביצוע אותן שגיאות. ייתכן שיהיה צורך להוסיף גם פריטים חדשים כדי לכלול פגמים חמורים שנמצאו בתוכנה. לכן, יש לעדכן באופן שוטף את רשימות התיוג על סמך ניתוח פגמים. עם זאת, יש להימנע מלאפשר לרשימת התיוג להיות ארוכה מדי (Gawande 2009).

בהיעדר מקרי בדיקה מפורטים, בדיקה מבוססת רשימות תיוג יכולה לספק הנחיות ורמה מסוימת של עקביות עבור הבדיקה. אם רשימות התיוג הן ברמה גבוהה, סביר להניח ששונות מסוימת בבדיקה עשויה להתרחש בפועל, וכתוצאה מכך הפוטנציאל לכיסוי יהיה גדול יותר אך תתאפשר פחות חזרתיות.

4.5 גישות בדיקה מבוססות שיתוף פעולה

לכל אחת מהטכניקות שהוזכרו לעיל (ראה סעיפים 4.2, 4.3, 4.4) יש מטרה מסוימת לאיתור פגמים. גישות המבוססות על שיתוף פעולה, לעומת זאת, מתמקדות גם במניעת הפגם על ידי שיתוף פעולה ותקשורת.

4.5.1 *כתיבה משותפת של סיפור המשתמש*

סיפור משתמש מייצג תכונה שתהיה בעלת ערך למשתמש או לרוכש של מערכת או תוֹכנָה. לסיפורי משתמשים יש שלושה היבטים קריטיים (Jeffries 2000), המכונים יחד (באנגלית) "3 C's":

- כרטיס (card) - האמצעי שבאמצעותו מתארים את סיפור המשתמש (למשל, כרטיס אינדקס, ערך בלוח אלקטרוני)
- שיחה (conversation) – ההסבר לאופן השימוש בתוכנה (ניתן לתעד או מילולי)
- אישור (confirmation) – קריטריוני הקבלה של סיפור המשתמש (ראה סעיף 4.5.2)

הפורמט הנפוץ ביותר לסיפור משתמש הוא "בתור [תפקיד], אני רוצה ש[יעד יושג], כדי שאוכל [הערך העסקי המתקבל לתפקיד]", ואחריו קריטריוני הקבלה. לצורך יצירה משותפת של סיפור המשתמש ניתן להשתמש בטכניקות כמו סיעור מוחות (brainstorming) ומפת חשיבה (mind mapping).

שיתוף הפעולה מאפשר לצוות לקבל חזון משותף של מה צריך להימסר, על ידי התחשבות בשלוש נקודות מבט נקודות שונות: עסקית, פיתוח ובדיקות.

סיפורי משתמשים טובים צריכים להיות: עצמאיים, ניתנים למשא ומתן, בעלי ערך, ניתנים להערכה, קטנים/קצרים וניתנים לבדיקה. אם בעל עניין אינו יודע כיצד לבדוק סיפור משתמש, זה עשוי להצביע על כך שסיפור המשתמש אינו ברור מספיק, או שאינו משקף משהו בעל ערך עבורו או שבעל העניין פשוט צריך עזרה בבדיקה (Wake 2003).

4.5.2 *קריטריוני קבלה*

קריטריוני הקבלה לסיפור המשתמש הם התנאים שהיישום של סיפור המשתמש חייב לעמוד בהם על מנת להתקבל על ידי בעלי העניין. מנקודת מבט זו, ניתן לראות את קריטריוני הקבלה כתנאי בדיקה שצריך לממש במסגרת הבדיקות. קריטריוני קבלה הם בדרך כלל תוצאה של שלב השיחה (ראה סעיף 4.5.1).

קריטריוני קבלה משמשים:

- להגדרת היקף סיפור המשתמש
- להגעה לקונצנזוס בין בעלי העניין
- לתיאור תרחישים חיוביים ושליליים
- כבסיס לבדיקות קבלה של סיפור המשתמש (ראה סעיף 4.5.3)
- לאפשר תכנון ואומדן מדויקים

ישנן מספר דרכים לכתוב קריטריוני קבלה לסיפור המשתמש. שני הפורמטים הנפוצים ביותר הם:

- מוכווני תרחיש (לדוגמה, פורמט בהינתן/כאשר/אז שנמצא בשימוש ב-BDD, ראה סעיף 2.1.3)
- מונחה כללים (למשל, רשימת הנחיות או טבלת מיפוי קלט-פלט)

ניתן לתעד את רוב קריטריוני הקבלה באחד משני הפורמטים הללו. עם זאת, הצוות עשוי להשתמש בפורמט אחר, מותאם אישית, כל עוד וקריטריוני הקבלה מוגדרים היטב וחד משמעיים.

4.5.3 *פיתוח מונחה בדיקות קבלה (ATDD)*

ATDD היא גישת בדוק-תחילה (ראה סעיף 2.1.3). מקרי הבדיקה נוצרים לפני היישום של סיפור המשתמש. מקרי הבדיקה נוצרים על ידי חברי צוות עם נקודות מבט שונות, למשל, לקוחות, מפתחים, ובודקים (Adzic 2009). מקרי הבדיקה עשויים להתבצע באופן ידני או אוטומטי.

השלב הראשון הוא לקיים סדנת מפרט שבה סיפור המשתמש וקריטריוני הקבלה שלו (אם טרם הוגדרו) מנותחים, נדונים ונכתבים על ידי חברי הצוות. חוסר שלמות, אי בהירות, או פגמים בסיפור המשתמש נפתרים במהלך תהליך זה. השלב הבא הוא יצירת מקרי הבדיקה. שלב זֶה יכול להיעשות על ידי הצוות כולו או על ידי הבודק בנפרד. מקרי הבדיקה מבוססים על קריטריוני קבלה וניתן לראות בהם דוגמאות לאופן פעולת התוכנה. מצב זה יעזור לצוות ליישם את סיפור המשתמש בצורה נכונה.

מכיוון שדוגמאות ובדיקות זהים, מונחים אלה משמשים לרוב לסירוגין. בזמן עיצוב הבדיקות ניתן ליישם את טכניקות הבדיקות המתוארות בסעיפים 4.2, 4.3 ו-4.4.

בדרך כלל, מקרי הבדיקה הראשונים חיוביים, ומאשרים את ההתנהגות הנכונה ללא חריגים או תנאים שגויים, וכוללים את רצף הפעילויות המתבצעות אם הכל הולך כמצופה. לאחר ביצוע מקרי הבדיקה החיוביים, הצוות צריך לבצע בדיקות שליליות. לבסוף, הצוות צריך לכסות גם מאפייני איכות לא פונקציונליים (למשל, יעילות ביצועים, שימושיות). מקרי הבדיקה צריכים להיות כתובים בצורה מובנת לבעלי העניין. בדרך כלל, מקרי הבדיקה מכילים משפטים בשפה טבעית הכוללים את התנאים המוקדמים הדרושים (אם יש), את הקלטים ואת תנאי הסיום.

מקרי הבדיקה חייבים לכסות את כל המאפיינים של סיפור המשתמש ולא צריכים לחרוג מהסיפור. עם זאת, קריטריוני הקבלה עשויים לפרט חלק מהבעיות המתוארות בסיפור המשתמש. בנוסף, שני מקרי בדיקה אינם צריכים לתאר את אותם מאפיינים הקיימים בסיפור המשתמש.

כתיבת מקרי בדיקה בפורמט הנתמך על ידי מסגרת בדיקות אוטומטיות (test automation framework) מאפשרת למפתחים להפוך את מקרי הבדיקה לאוטומטים על ידי כתיבת הקוד למקרים אלו תוך שהם מיישמים את התכונה המתוארת בסיפור המשתמש. כך למעשה הופכות בדיקות הקבלה לדרישה הניתנת להרצה.

5 ניהול פעילות הבדיקות — 335 דקות

מושגים

בדיקות מבוססות סיכון (risk-based testing), בקרת בדיקות (test control), בקרת סיכונים (risk control), גישת בדיקה (test approach), דוח השלמת בדיקות (test completion report), דוח התקדמות בדיקות (test progress report), דוח פגמים (defect report), הערכת סיכונים (risk assessment), הפחתת סיכונים (risk mitigation), זיהוי סיכונים (risk identification), ניהול סיכונים (risk management), ניהול פגמים (defect management), ניטור בדיקות (test monitoring), ניטור סיכונים (risk monitoring), ניתוח סיכונים (risk analysis), סיכון (risk), סיכון מוצר (product risk), סיכון פרויקט (project risk), פירמידת בדיקות (test pyramid), קריטריוני יציאה (exit criteria), קריטריוני כניסה (entry criteria), רבעי בדיקה (testing quadrants), רמת סיכון (risk level), תוכנית בדיקות (test plan), תכנון בדיקות (test planning)

מטרות הלמידה לפרק 5

5.1 תכנון בדיקות

FL-5.1.1 (K2) הדגם את המטרה והתוכן של תוכנית בדיקה

FL-5.1.2 (K1) זהה כיצד בודק מוסיף ערך לאיטרציה ולתכנון שחרור גרסה (release planning)

FL-5.1.3 (K2) השווה והבדל בין קריטריוני כניסה וקריטריוני יציאה

FL-5.1.4 (K3) השתמש בטכניקות אומדן כדי לחשב את מאמץ הבדיקה הנדרש

FL-5.1.5 (K3) יישם תעדוף מקרי בדיקה

FL-5.1.6 (K1) זכור את המושגים של פירמידת הבדיקות

FL-5.1.7 (K2) סכם את רבעי הבדיקה ואת הקשרים שלהם עם רמות הבדיקה וסוגי הבדיקות

5.2 ניהול סיכונים

FL-5.2.1 (K1) זהה את רמת הסיכון באמצעות סבירות הסיכון והשפעתו

FL-5.2.2 (K2) הבחן בין סיכוני פרויקט לסיכוני מוצר

FL-5.2.3 (K2) הסבר כיצד ניתוח סיכוני מוצר עשוי להשפיע על היסודיות והיקף הבדיקה

FL-5.2.4 (K2) הסבר אילו אמצעים ניתן לנקוט בתגובה לסיכוני מוצר שנותחו

5.3 ניטור בדיקות, בקרת בדיקות והשלמת בדיקות

FL-5.3.1 (K1) זכור מדדים המשמשים לבדיקות

FL-5.3.2 (K2) סכם את המטרות, התוכן והקהלים עבור דוחות בדיקה

FL-5.3.3 (K2) הדגם כיצד לתקשר את מצב הבדיקה

5.4 ניהול תצורה

FL-5.4.1 (K2) סכם את האופן שבו ניהול תצורה תומך בבדיקות

5.5 ניהול פגמים

FL-5.5.1 (K3) הכן דוח פגמים

5.1 תכנון בדיקות

5.1.1 המטרה והתוכן של תוכנית בדיקות

תוכנית בדיקות מתארת את היעדים, המשאבים והתהליכים עבור פרויקט בדיקות. תוכנית בדיקות:

- מתעדת את האמצעים ולוח הזמנים להשגת מטרות הבדיקות
- מסייעת להבטיח כי פעילויות הבדיקה שיבוצעו יעמדו בקריטריונים שנקבעו
- משמשת כאמצעי תקשורת עם חברי צוות ובעלי עניין אחרים
- מוודאת שהבדיקות יצייתו למדיניות הבדיקות הקיימת ולאסטרטגיית הבדיקות הקיימת (או מסבירה מדוע הבדיקות יחרגו מהן)

תכנון הבדיקות מנחה את חשיבת הבודקים ומאלץ אותם להתמודד עם אתגרים עתידיים הקשורים לסיכונים, לוחות זמנים, אנשים, כלים, עלויות, מאמץ וכו'. תהליך הכנת תוכנית הבדיקות הוא דרך שימושית לחשוב על המאמצים הדרושים להשגת מטרות פרויקט הבדיקות.

התוכן האופייני של תוכנית בדיקה כולל:

- הֶקשֵר הבדיקה (למשל, היקף, מטרות הבדיקה, בסיס הבדיקות)
- הנחות ואילוצים של הפרויקט הנבדק
- בעלי עניין (למשל, תפקידים, תחומי אחריות, רלוונטיות לבדיקה, גיוס והכשרה)
- תקשורת (למשל, טפסים ותדירות של תקשורת, תבניות תיעוד)
- רישום סיכונים (למשל סיכוני מוצר, סיכוני פרויקט)
- גישת הבדיקה (למשל, רמות בדיקה, סוגי בדיקות, טכניקות בדיקה, תוצרי בדיקה, קריטריוני כניסה וקריטריוני יציאה, עצמאות הבודקים, מדדים שיש לאסוף, דרישות נתוני בדיקה, דרישות סביבת בדיקה, סטיות ממדיניות הבדיקות הארגונית ומאסטרטגיית הבדיקה)
- תקציב ולוח זמנים

פרטים נוספים על תוכנית הבדיקות ותכניה ניתן למצוא בתקן ISO/IEC/IEEE 29119-3.

5.1.2 תרומתו של הבודק לתכנון איטרציה ולתכנון שחרור הגרסה

ב- SDLCs איטרטיביים, בדרך כלל מתרחשים שני סוגים של תכנון: תכנון שחרור ותכנון איטרציה.

תכנון השחרור (של הגרסה) מסתכל קדימה לעבר שחרור המוצר, מגדיר וגם מגדיר-מחדש את צבר השיפורים והתיקונים של המוצר, ועשוי לכלול זיקוק של סיפורי משתמש גדולים יותר לקבוצות של סיפורי משתמש קטנים יותר. הוא משמש גם כבסיס לגישת הבדיקות ולתוכנית הבדיקות בכל האיטראציות. בודקים המעורבים בתכנון שחרור הגרסה משתתפים בכתיבת סיפורי המשתמש הניתנים לבדיקה וקריטריוני קבלה (ראה סעיף 4.5), משתתפים בניתוחי סיכוני הפרויקט וסיכוני האיכות (ראה סעיף 5.2), מעריכים את מאמצי הבדיקה בהתאם לסיפורי המשתמש (ראה סעיף 5.1.4), קובעים את גישת הבדיקות ומתכננים את הבדיקות לגרסה המשוחררת.

תכנון איטרציה מסתכל קדימה לסופה של איטרציה יחידה ועוסק בתכולת האיטרציה. בודקים המעורבים בתכנון האיטרציה משתתפים בניתוח סיכונים מפורט של סיפורי המשתמש, קובעים את הבדיקתיות של סיפורי המשתמש, מפרקים סיפורי המשתמש למשימות (במיוחד משימות בדיקות), מעריכים את מאמץ הבדיקות עבור כל משימות הבדיקות ומזהים ומחדדים היבטים פונקציונליים ולא פונקציונליים של אובייקט הבדיקה.

5.1.3 *קריטריוני כניסה וקריטריוני יציאה*

קריטריוני כניסה מגדירים את התנאים המוקדמים לביצוע פעילות נתונה. אם לא יעמדו בקריטריוני הכניסה, סביר להניח שהפעילות תתברר כקשה יותר, גוזלת זמן, יקרה ומסוכנת יותר. קריטריוני יציאה מגדירים מה צריך להשיג כדי להכריז על פעילות שהושלמה. יש להגדיר את קריטריוני הכניסה ואת קריטריוני היציאה לכל רמת בדיקה, והם ישתנו בהתאם למטרות הבדיקה.

קריטריוני כניסה אופייניים כוללים: זמינות משאבים (למשל, אנשים, כלים, סביבות, נתוני בדיקה, תקציב, זמן), זמינות של הבודקה (למשל, בסיס בדיקות , דרישות ניתנות לבדיקה, סיפורי משתמש, מקרי בדיקות) ורמת האיכות הראשונית של אובייקט הבדיקה (למשל, שכל בדיקות השפיות עברו).

קריטריוני יציאה אופייניים כוללים: מדדים המעידים על יסודיות (למשל, רמת הכיסוי שהושגה, מספר הפגמים שלא נפתרו, צפיפות פגמים, מספר מקרי הבדיקה שנכשלו) וקריטריונים להשלמת משימות (למשל, שבוצעו בדיקות מתוכננות, שבוצעו בדיקות סטטיות, שדווחו כל הפגמים שנמצאו, שכל בדיקות הרגרסיה הפכו להיות אוטומטיות).

ניתן לראות בזמן או בתקציב שנגמר גם כקריטריוני יציאה תקפים. גם מבלי שקריטריונים אחרים ליציאה מתקיימים, זה יכול להיות מקובל לסיים את הבדיקות בנסיבות כאלה, אם בעלי העניין בחנו וקיבלו על עצמם את הסיכון לעלות לאוויר ללא בדיקות נוספות.

בפיתוח תוכנה זריז וגמיש (אג'ילי), קריטריוני היציאה נקראים לעתים קרובות Definition of Done, המגדירים את מדדי המטרה של הצוות עבור פריט שניתן לשחרור. קריטריוני כניסה שסיפור משתמש חייב לעמוד בהם כדי להתחיל את פעילויות הפיתוח ו/או הבדיקה נקראיםDefinition of Ready.

5.1.4 *טכניקות אומדן בדיקות*

הערכת מאמץ הבדיקה כרוכה בחיזוי כמות העבודה הקשורה לבדיקות בכדי לעמוד ביעדים של פרויקט הבדיקות. חשוב להבהיר לבעלי העניין כי האומדן מבוסס על מספר הנחות ונתון תמיד לשגיאות אומדן. הערכה עבור משימות קטנות היא בדרך כלל מדויקת יותר מאשר עבור הגדולות. לכן, בעת הערכת משימה גדולה, ניתן לפרק אותה לקבוצה של משימות קטנות יותר אשר כל אחת מהן בתורן ניתן להעריך בצורה מדויקת יותר.

בתוכנית לימודים זו מתוארות ארבע טכניקות האומדן הבאות.

הערכה המבוססת על יחסים. בטכניקה מבוססת מדדים זו, נאספים נתונים מפרויקטים קודמים בארגון, מה שמאפשר לגזור יחסים "סטנדרטיים" לפרויקטים דומים. היחסים בין הפרויקטים של הארגון עצמו (למשל, נלקחו מנתונים היסטוריים) הם בדרך כלל המקור הטוב ביותר לשימוש בתהליך האומדן. לאחר מכן ניתן להשתמש ביחסים סטנדרטיים אלה כדי להעריך את מאמץ הבדיקה עבור הפרויקט החדש. למשל, אם בפרויקט הקודם יחס מאמץ הפיתוח לבדיקה היה 3:2, ובפרויקט הנוכחי מאמץ הפיתוח צפוי להיות 600 ימי אדם, ניתן להעריך את מאמץ הבדיקה כ-400 ימי אדם.

אקסטרפולציה (אומדן משוער). בטכניקה מבוססת מדדים זו, המדידות נעשות מוקדם ככל האפשר בפרויקט הנוכחי כדי לאסוף את הנתונים. לאחר מספיק תצפיות, המאמץ הנדרש לעבודה הנותרת יכול להיות משוער על ידי אקסטרפולציה של נתונים אלה (בדרך כלל על ידי יישום מודל מתמטי).

שיטה זו מתאימה מאוד עבור SDLCs איטרטיביים. למשל, הצוות עשוי להסיק את מאמץ הבדיקה באיטרציה הקרובה כמאמץ הממוצע משלושת האיטרציות האחרונות.

שיטת פס רחב (דלפי). בטכניקה איטרטיבית זו, המבוססת על מומחים, מומחים מבצעים הערכות מבוססות ניסיון. כל מומחה, בבידוד, מעריך את המאמץ. התוצאות נאספות ואם יש חריגות שהן מחוץ לטווח הגבולות המוסכמים, המומחים דנים באומדנים הנוכחיים שלהם. לאחר מכן כל מומחה מתבקש לבצע הערכה מחודשת על סמך משוב זה, שוב בבידוד. תהליך זה חוזר על עצמו עד שמגיעים להסכמה. תכנון פוקר הוא גרסה של Wideband Delphi, בדרך כלל ב-Agile. בפיתוח בתכנון פוקר, ההערכות נעשות בדרך כלל באמצעות קלפים עם מספרים המייצגים את גודל המאמץ.

הערכה של שלוש-נקודות. בטכניקה מבוססת מומחים זו, שלוש הערכות נעשות על ידי המומחים:

האומדן האופטימי ביותר (a), האומדן הסביר ביותר (m) והאומדן הפסימי ביותר (b). האומדן הסופי (E) שהוא הממוצע האריתמטי המשוקלל שלהם. בגרסה הפופולרית ביותר של טכניקה זו, האומדן מחושב כ- E = (a + 4 * m + b) / 6. היתרון של טכניקה זו הוא בזה שהיא מאפשרת למומחים לחשב את שגיאת המדידה (סטיית התקן): SD = (b – a) / 6. למשל, אם האומדנים (בשעות אדם) הם: a=6, m=9 ו-b=18, אז האומדן הסופי הוא 10±2 שעות אדם (כלומר, בין 8 ל-12 שעות אדם), על כן: E = (6 + 4*9 + 18) / 6 = 10 ו-SD = (18 – 6) / 6 = 2.

עבור טכניקות אלה ורבות אחרות להערכת בדיקות ראה (Kan 2003, Koomen 2006, Westfall 2009)

5.1.5 *תעדוף מקרי בדיקה*

לאחר שמקרי הבדיקה והליכי הבדיקות מוגדרים ומורכבים לסדרת בדיקות (test suites), ניתן לארגן את סדרות הבדיקה הללו בלוח זמנים לביצוע (להרצות) המגדיר את הסדר שבו יש להריצן. כאשר מתעדפים מקרי בדיקה, ניתן לקחת בחשבון גורמים שונים. אסטרטגיות תעדוף מקרי הבדיקה הנפוצות ביותר הן כדלקמן:

- תעדוף מבוסס סיכון, כאשר סדר ביצוע הבדיקות מבוסס על תוצאות ניתוח סיכונים (ראה סעיף 5.2.3). מקרי בדיקה המכסים את הסיכונים החשובים ביותר מבוצעים תחילה.
- תעדוף מבוסס כיסוי, כאשר סדר ביצוע הבדיקות מבוסס על כיסוי (למשל, כיסוי שורות קוד). מקרי בדיקה המשיגים את הכיסוי הגבוה ביותר מבוצעים ראשונים. בגרסה אחרת, הנקראת תעדוף כיסוי נוסף, מקרה הבדיקה המשיג את הכיסוי הגבוה ביותר מבוצע ראשון; כל מקרה בדיקה עוקב (הנובע מקודמו) הוא זה שמשיג את הכיסוי הנוסף הגבוה ביותר.
- תעדוף מבוסס דרישות, כאשר סדר ביצוע הבדיקות מבוסס על סדרי העדיפויות של הדרישות המיוחסות למקרי הבדיקה המתאימים. סדרי העדיפויות של הדרישות מוגדרים על ידי בעלי העניין. כאשר מקרי בדיקה הקשורים לדרישות החשובות ביותר מבוצעים ראשונים.

באופן אידיאלי, מקרי בדיקה יתוזמנו להרצה על בסיס רמות העדיפות שלהם, תוך שימוש, למשל, באחת מאסטרטגיות התעדוף שהוזכרו לעיל. עם זאת, ייתכן שנוהג זה לא יפעל אם מקרי הבדיקה או התכונות הנבדקות תלויים במקרים אחרים. אם מקרה בדיקה בעדיפות גבוהה יותר תלוי במקרה בדיקה בעדיפות נמוכה יותר, יש לבצע תחילה את מקרה הבדיקה בעדיפות נמוכה יותר.

סדר ביצוע הבדיקה חייב לקחת בחשבון גם את זמינות המשאבים. למשל, כלי הבדיקה הנדרשים, סביבות הבדיקה או האנשים שעשויים להיות זמינים רק בחלון זמן ספציפי.

5.1.6 *פירמידת בדיקות*

פירמידת הבדיקות היא מודל המראה כי בדיקות שונות עשויות להיות בעלות רמת פירוט שונה. מודל פירמידת הבדיקות תומך בצוות בצורך שבבדיקות האוטומטיות ובהקצאת מאמצי בדיקה בכך שהוא מראה שיעדים שונים נתמכים על ידי רמות שונות של אוטומצית בדיקות. שכבות הפירמידה מייצגות קבוצות של בדיקות. ככל שהשכבה גבוהה יותר, כך רמת הפירוט של הבדיקה, בידוד הבדיקה וזמן ביצוע הבדיקה נמוכים יותר. הבדיקות בשכבה התחתונה הן קטנות, מבודדות, מהירות, ובודקות פיסת פונקציונליות קטנה, כך שבדרך כלל יש צורך בהרבה מהן כדי להשיג כיסוי סביר. השכבה העליונה מייצגת בדיקות מורכבות, ברמה גבוהה, ומקצה לקצה. בדיקות ברמה גבוהה הן בדרך כלל איטיות יותר מהבדיקות מהשכבות התחתונות, והן בדרך כלל בודקות חלק גדול של פונקציונליות, כך שבדרך כלל יש צורך רק בכמה מהן כדי להשיג כיסוי סביר. מספר השכבות ושמותיהן עשויים להיות שונים. למשל, מודל פירמידת הבדיקות המקורי (Cohn 2009) מגדיר שלוש שכבות: "בדיקות יחידה", "בדיקות שירות" ו"בדיקות ממשק משתמש". מודל פופולרי נוסף מגדיר בדיקות יחידה (רכיבים), בדיקות אינטגרציה (שילוב רכיבים) ובדיקות מקצה לקצה. ניתן להשתמש גם ברמות בדיקה אחרות (ראה סעיף 2.2.1).

5.1.7 *בדיקת רביעים*

רבעי הבדיקה, שהוגדרו על ידי בריאן מאריק (Marick 2003, Crispin 2008), מקבצים את רמות הבדיקה עם סוגי הבדיקה, הפעילויות, טכניקות הבדיקה ומוצרי העבודה המתאימים בפיתוח התוכנה במודל Agile.

המודל תומך בניהול בדיקות ובהצגתם כדי להבטיח שכל סוגי הבדיקות ורמות הבדיקה המתאימים נכללים ב-SDLC ובהבנה שסוגי בדיקות מסוימים רלוונטיים יותר לרמות בדיקה מסוימות מאחרות. מודל זה מספק גם דרך לבדל ולתאר את סוגי הבדיקות עבורכל בעלי העניין, כולל מפתחים, בודקים ונציגים עסקיים.

במודל זה, בדיקות יכולות להיות מכוונות עסקים או מכוונות טכנולוגיה. בדיקות יכולות גם לתמוך בצוות (כלומר, להנחות את הפיתוח) או לבקר את המוצר (כלומר, למדוד את התנהגות המוצר מול הציפיות). השילוב של שתי נקודות מבט אלה קובע את ארבעת הרבעים:

- רביע Quadrant Q1 (מכוון טכנולוגיה, תמיכה בצוות). רביע זה מכיל בדיקות רכיבים ואינטגרציה בין רכיבים. בדיקות אלה צריכות להיות אוטומטיות ולהיכלל בתהליך CI
- רביע Q2 (מכוון עסקים, תמיכה בצוות). רביע זה מכיל בדיקות פונקציונליות, דוגמאות, בדיקות סיפור משתמש, אבות טיפוס של חוויית משתמש, בדיקות API והדמיות. בדיקות אלה בודקות את קריטריוני הקבלה ויכולות להיות ידניות או אוטומטיות
- רביע Q3 (מכוון עסקים, בוחן את המוצר). רביע זה מכיל בדיקות חוקרות (exploratory), בדיקות שימושיות, בדיקות קבלה של המשתמשים (user acceptance tests). בדיקות אלה הן מונחות משתמש ולעתים קרובות הן ידניות
- רביע Q4 (מכוון טכנולוגיה, בוחן את המוצר). רביע זה מכיל בדיקות עשן (smoke tests) ובדיקות לא פונקציונליות (למעט בדיקות שימושיות). בדיקות אלה הן לעתים קרובות אוטומטיות

5.2 ניהול סיכונים

ארגונים מתמודדים עם גורמים פנימיים וחיצוניים רבים שבגינם אינם בטוחים אם ומתי ישיגו את יעדיהם (ISO 31000). ניהול סיכונים מאפשר לארגונים להגדיל את הסבירות להשגת יעדים, לשפר את איכות המוצרים שלהם ולהגביר את הביטחון והאמון של בעלי העניין.

פעילויות ניהול הסיכונים העיקריות הן:

- ניתוח סיכונים (הכולל זיהוי סיכונים והערכת סיכונים; ראה סעיף 5.2.3)
- בקרת סיכונים (הכוללת הפחתת סיכונים וניטור סיכונים; ראה סעיף 5.2.4)

גישת הבדיקות, שבה פעילויות הבדיקות נבחרות, מתועדפות ומנוהלות על בסיס ניתוח סיכונים ובקרת סיכונים, נקראת בדיקות מונחות סיכונים (risk-based testing).

5.2.1 הגדרות סיכונים ומאפייני סיכון

סיכון הוא אירוע, סכנה, איום או מצב פוטנציאליים שהתרחשותם תגרום להשפעה שלילית. סיכון יכול להיות מאופיין בשני גורמים:

- סבירות הסיכון – ההסתברות להתרחשות הסיכון (גדולה מאפס וקטנה מאחת)
- השפעת הסיכון (הנזק) – ההשלכות של התרחשות זו

שני גורמים אלה מבטאים את רמת הסיכון, שהיא מדד לסיכון. ככל שרמת הסיכון גבוהה יותר, כך הטיפול בו חשוב יותר.

5.2.2 סיכוני פרויקט וסיכוני מוצר

בבדיקות תוכנה עוסקים בדרך כלל בשני סוגים של סיכונים: סיכוני פרויקט וסיכוני מוצר.

סיכוני הפרויקט קשורים לניהול ובקרה של הפרויקט. סיכוני הפרויקט כוללים:

- בעיות ארגוניות (למשל, עיכובים באספקת תוצרי עבודה, הערכות לא מדויקות, קיצוץ עלויות)
- בעיות הקשורות לאנשים (למשל, מיומנויות לא מספיקות, קונפליקטים, בעיות תקשורת, מחסור בצוות)
- בעיות טכניות (למשל, זחילת היקף (התרחבות בלתי מבוקרת של היקף הפרויקט), תמיכה מועטה בכלים)
- בעיות ספקים (למשל, כשל במסירה של צד שלישי, פשיטת רגל של החברה התומכת)

לסיכוני פרויקט, כאשר הם מתרחשים, עלולה להיות השפעה על לוח הזמנים, התקציב או ההיקף של הפרויקט, דבר המשפיע על יכולתו של הפרויקט להשיג את יעדיו.

סיכוני המוצר קשורים למאפייני איכות המוצר (למשל, המתוארים במודל האיכות ב ISO 25010). דוגמאות לסיכוני מוצר כוללות: פונקציונליות חסרה או שגויה, חישובים שגויים, שגיאות זמן ריצה, ארכיטקטורה גרועה, אלגוריתמים לא יעילים, זמן תגובה לא מספק, חוויית משתמש גרועה, פגיעויות אבטחה. סיכוני המוצר, כאשר הם מתרחשים, עלולים לגרום להשלכות שליליות שונות, כולל:

- חוסר שביעות רצון של המשתמשים
- אובדן הכנסות, אמון, מוניטין

- נזק לצדדים שלישיים
- עלויות תחזוקה גבוהות, עומס יתר על מוקד התמיכה
- עונשים פליליים
- במקרים קיצוניים, נזק פיזי, פציעות או אפילו מוות.

5.2.3 *ניתוח סיכוני מוצר*

מנקודת מבט של בדיקות, המטרה של ניתוח סיכוני מוצר היא לספק מודעות לסיכוני המוצר על מנת למקד את מאמץ הבדיקה באופן שממזער את הרמה השיורית של סיכון המוצר. באופן אידיאלי, ניתוח סיכוני מוצר מתחיל בשלב מוקדם של ה-SDLC.

ניתוח סיכוני מוצר מורכב מזיהוי סיכונים והערכת סיכונים. זיהוי סיכונים משמעו יצירת רשימה מקיפה של סיכונים. בעלי עניין יכולים לזהות סיכונים באמצעות טכניקות וכלים שונים, כגון סיעור מוחות, סדנאות, ראיונות או דיאגרמות סיבה-תוצאה. הערכת סיכונים כוללת: סיווג סיכונים מזוהים, קביעת סבירות הסיכון שלהם, השפעת הסיכון ורמתו, תעדוף והצעת דרכים להתמודד איתם. סיווג לקטגוריות מסייע בהקצאת פעולות להפחתת סיכונים, מכיוון שבדרך כלל ניתן להפחית סיכונים המשתייכים לאותה קטגוריה באמצעות גישה דומה.

הערכת סיכונים יכולה להשתמש בגישה כמותית או איכותית, או שילוב שלהם. בגישה הכמותית רמת הסיכון מחושבת כמכפיל סבירות הסיכון והשפעת הסיכון. בגישה האיכותית ניתן לקבוע את רמת הסיכון באמצעות מטריצת סיכון.

ניתוח סיכוני מוצר עשוי להשפיע על היסודיות וההיקף של הבדיקות. תוצאותיו משמשות ל:

- קביעת היקף הבדיקות שיש לבצע
- קביעת רמות הבדיקה המסוימות והצעת סוגי בדיקות לביצוע
- קביעת טכניקות הבדיקה שיש להשתמש בהן ואת הכיסוי שיש להשיג
- הערכת מאמץ הבדיקה הנדרש עבור כל משימה
- תעדוף בדיקות בניסיון לאתר את הליקויים הקריטיים מוקדם ככל האפשר
- קביעה האם ניתן להשתמש בפעילויות כלשהן בנוסף לבדיקות כדי להפחית את הסיכון

5.2.4 *בקרת סיכוני מוצר*

בקרת סיכוני מוצר כוללת את כל האמצעים הנקטים בתגובה לסיכוני מוצר מזוהים ומוערכים. בקרת סיכוני מוצר מורכבת מהפחתת סיכונים וניטור סיכונים. הפחתת סיכונים כרוכה ביישום הפעולות המוצעות בהערכת סיכונים להפחתת רמת הסיכון. מטרת ניטור הסיכונים היא להבטיח שפעולות ההפחתה יהיו יעילות, לקבל מידע נוסף לשיפור הערכת הסיכונים ולזהות סיכונים מתפתחים.

בהתייחסות לבקרת סיכוני מוצר, לאחר ניתוח הסיכון, מספר אפשרויות תגובה לסיכון אפשריות, למשל, הפחתת סיכונים על ידי בדיקות, קבלת סיכונים, העברת סיכונים או תוכנית מגירה (Veenendaal 2012). על מנת להפחית את סיכוני המוצר על ידי בדיקות ניתן לבצע את הפעולות הבאות:

- לבחור את הבודקים עם רמת הניסיון והכישורים הנכונים, המתאימים לסוג סיכון נתון
- ליישם רמה מתאימה של עצמאות הבדיקות

- לבצע ביקורות וניתוח סטטי
- ליישם את טכניקות הבדיקה ואת רמות הכיסוי המתאימות
- להחיל את סוגי הבדיקות המתאימים תוך התייחסות למאפייני האיכות המושפעים
- לבצע בדיקות דינמיות, כולל בדיקות רגרסיה

5.3 ניטור בדיקות, בקרת בדיקות והשלמת בדיקות

ניטור בדיקות עוסק באיסוף מידע על בדיקות. מידע זה משמש להערכת התקדמות הבדיקות ולמדידה האם קריטריוני היציאה מהבדיקות או משימות הבדיקות המשויכות לקריטריוני היציאה מתקיימים, כגון: עמידה ביעדים לכיסוי סיכוני מוצר, דרישות או קריטריוני קבלה.

בקרת הבדיקה משתמשת במידע מניטור הבדיקות כדי לספק, בצורה של הוראות בקרה, הדרכה ופעולות מתקנות הדרושות כדי להשיג את הבדיקות היעילות ביותר. דוגמאות להוראות בקרה כוללות:

- בדיקות חוזרות כאשר הסיכון המזוהה הופך לבעיה
- הערכה מחדש אם פריט בדיקה עומד בקריטריוני כניסה או בקריטריוני יציאה עקב עבודה חוזרת
- התאמת לוח הזמנים של הבדיקות לטיפול בעיכוב במסירת סביבת הבדיקות
- הוספת משאבים חדשים בהתאם לנדרש בזמן ובמקום

בשלב השלמת הבדיקות אוספים נתונים מפעילויות בדיקות שהושלמו כדי לאחד את הכל למקום אחד, כגון: ניסיון, בודקה (testware) וכל מידע רלוונטי אחר. פעילויות השלמת בדיקות מתרחשות כחלק מאבני הדרך של פרויקט, כגון: כאשר רמת בדיקה הושלמה, איטרציה בפיתוח אג'ילי מסתיימת, פרויקט בדיקות הושלם (או בוטל), מערכת תוכנה משוחררת או מהדורת תחזוקה הושלמה.

5.3.1 מדדים בשימוש הבדיקות

מדדי הבדיקות נאספים כדי להראות את: ההתקדמות מול לוח הזמנים והתקציב המתוכננים, האיכות הנוכחית של אובייקט הבדיקה והיעילות של פעילויות הבדיקה ביחס ליעדים או ליעד איטרציה.

ניטור בדיקות אוסף מגוון מדדים כדי לתמוך בבקרת הבדיקות ובהשלמת הבדיקות.

מדדי בדיקות נפוצים כוללים:

- מדדי התקדמות פרויקט (למשל, השלמת פעילויות, שימוש במשאבים, מאמצי בדיקה)
- מדדי התקדמות הבדיקות (למשל, התקדמות יישום מקרה הבדיקה, התקדמות הכנה בסביבת הבדיקה, מספר מקרי הבדיקה רצים/לא פועלים, עבר/נכשל, זמן ביצוע בדיקה)
- מדדי איכות מוצר (למשל, זמינות, זמן תגובה, זמן ממוצע לכישלון)
- מדדי פגמים (למשל, מספר ותיעדוף של פגמים שנמצאו/תוקנו, צפיפות פגמים, אחוז גילוי פגמים)
- מדדי סיכון (למשל, רמת סיכון שיורית)
- מדדי כיסוי (למשל, כיסוי דרישות, כיסוי קוד)
- מדדי עלות (למשל, עלות הבדיקה, עלות האיכות הארגונית)

5.3.2 מטרות, תוכן וקהל היעד של דוחות הבדיקות

דוח הבדיקות מסכם ומעביר את פרטי הבדיקות במהלך הבדיקות. דוחות התקדמות הבדיקות תומכים בבקרה השוטפת של הבדיקות וחייבים לספק מספיק מידע כדי לבצע שינויים בלוח הזמנים של הבדיקות, במשאבים או בתוכנית הבדיקות, כאשר שינויים כאלה נדרשים עקב חריגה מהתוכנית או שינוי נסיבות. דוחות השלמת בדיקות מסכמים שלב ספציפי של בדיקה (למשל, רמת בדיקה, סבב, איטרציה) ויכולים לספק מידע לבדיקות הבאות.

במהלך ניטור ובקרת בדיקות, צוות הבדיקה מפיק דוחות התקדמות בדיקות עבור בעלי עניין כדי לעדכן אותם. דוחות התקדמות הבדיקות מופקים בדרך כלל על בסיס קבוע (למשל, יומי, שבועי וכו') וכוללים:

- את תקופת הבדיקות
- התקדמות הבדיקות (למשל, לפני או אחרי לוח הזמנים), כולל סטיות בולטות
- מכשולים לבדיקות והדרכים לעקיפת הבעיה
- מדדי בדיקות (ראה דוגמאות בסעיף 5.3.1)
- סיכונים חדשים ומשתנים בתקופת הבדיקות
- בדיקות מתוכננות לתקופה הבאה

דוח סיכום בדיקות נכתב במהלך שלב השלמת הבדיקות, כאשר פרויקט, רמת בדיקה או סוג בדיקה הושלמו וכאשר, באופן אידיאלי, קריטריוני היציאה שלו התקיימו. דוח זה משתמש בדוחות התקדמות בדיקות ובנתונים אחרים. דוחות אופייניים להשלמת בדיקות כוללים:

- סיכום הבדיקות
- בדיקות והערכת איכות המוצר בהתבסס על תוכנית הבדיקה המקורית (כלומר, יעדי הבדיקה וקריטריוני יציאה)
- חריגות מתוכנית הבדיקה (למשל, הבדלים מלוח הזמנים המתוכנן, משך הזמן והמאמץ)
- מכשולים שקרו במהלך הבדיקות ודרכים לעקיפת הבעיות
- מדדי בדיקה המבוססים על דוחות התקדמות הבדיקה
- סיכונים שלא טופלו, פגמים שלא תוקנו
- לקחים רלוונטיים לבדיקות

קהלים שונים דורשים מידע שונה בדוחות, ומשפיעים על מידת הפורמליות ותדירות הדיווח. דיווח על התקדמות הבדיקה לאחרים הנמצאים באותו הצוות לעתים קרובות מתבצע בתכיפות ואינו רשמי, בעוד שדיווח על בדיקות עבור פרויקט שהושלם מיושם או מתבצע באמצעות תבנית מוגדרת ומתרחש פעם אחת בלבד.

תקן ISO/IEC/IEEE 29119-3 כולל תבניות ודוגמאות לדוחות התקדמות בדיקה (הנקראים דוחות מצב בדיקה) ודוחות השלמת בדיקות.

5.3.3 *מסירת סטטוס הבדיקות*

האמצעים הטובים ביותר לתקשורת סטטוס הבדיקות משתנים: בהתאם לשיקולים בניהול בדיקות, אסטרטגיות בדיקות ארגוניות, תקנים רגולטוריים, או, במקרה של צוותים המנהלים את עצמם (ראה סעיף 1.5.2), בצוות עצמו. האפשרויות כוללות:

- תקשורת מילולית עם חברי צוות ובעלי עניין אחרים
- לוחות מחוונים (למשל, לוחות מחוונים של CI/CD, לוחות משימות ותרשימי התקדמות)
- ערוצי תקשורת אלקטרוניים (למשל, דוא"ל, צ'אט)
- תיעוד מקוון
- דוחות בדיקות רשמיים (ראה סעיף 5.3.2)

ניתן להשתמש באחת או יותר מאפשרויות אלה. תקשורת רשמית יותר עשויה להתאים יותר לצוותים מבוזרים שבהם תקשורת ישירה פנים אל פנים אינה תמיד אפשרית בגלל מרחק גיאוגרפי או הבדלי זמן. בדרך כלל, בעלי עניין שונים מעוניינים בסוגים שונים של מידע, ולכן התקשורת צריכה להיות מותאמת בהתאם.

5.4 *ניהול תצורה*

בתחום הבדיקות, ניהול תצורה (CM) מספק שיטה לזיהוי, בקרה ומעקב אחר תוצרי עבודה כגון תוכניות בדיקה, אסטרטגיות בדיקה, תנאי בדיקה, מקרי בדיקה, סקריפטים לבדיקה, תוצאות בדיקות, יומני בדיקות ודוחות בדיקה כפריטי ניהול תצורה.

עבור פריט תצורה מורכב (למשל, סביבת בדיקה), CM מתעד את הפריטים שהוא מורכב מהם, את קשרי הגומלין שלהם ואת הגרסאות. כאשר פריט התצורה מאושר לבדיקה, הוא הופך לנקודת המוצא וניתן לשנות אותו רק באמצעות תהליך בקרת שינויים רשמי.

ניהול תצורה שומר תיעוד של פריטי תצורה שהשתנו בעת יצירת תוכנית בסיסית חדשה. ניתן לחזור לנקודת המוצא הקודמת (baseline) כדי לשחזר תוצאות בדיקה קודמות.

על מנת לתמוך כראוי בבדיקות, CM מבטיח את הדברים הבאים:

- כל פריטי התצורה, כולל פריטי בדיקה (חלקים בודדים של אובייקט הבדיקה), מזוהים באופן ייחודי, נשלטים על-ידי גרסאות, נמצאים במעקב אחר שינויים וקשורים לפריטי תצורה אחרים, כך שניתן יהיה לשמור על עקיבות לאורך כל תהליך הבדיקה
- כל התיעוד המזוהה ופריטי התוכנה מוזכרים באופן חד משמעי בתיעוד הבדיקה

אינטגרציה רציפה, אספקה רציפה, פריסה רציפה (CI/CD) והבדיקות הנלוות מיושמים בדרך כלל כחלק מצינור DevOps אוטומטי (pipeline) (ראה סעיף 2.1.4), שבו CM אוטומטי נכלל בדרך כלל.

5.5 ניהול פגמים

מכיוון שאחת ממטרות הבדיקה העיקריות היא למצוא פגמים, תהליך ניהול פגמים מבוסס הוא חיוני. למרות שאנו מתייחסים כאן ל"פגמים", החריגות המדווחות עשויות להתברר כפגמים אמיתיים או משהו אחר (למשל, חיובי כוזב, בקשת שינוי) - זה נפתר במהלך הטיפול בדוחות הפגמים. ניתן לדווח על חריגות במהלך כל שלב של ה-SDLC ופורמט (או תבנית) הטופס תלוי ב- SDLC. לכל הפחות, תהליך ניהול הפגמים כולל תרשים זרימת עבודה לטיפול בחריגות הבודדות מגילוין ועד לסגירתן וכללים לסיווגן. תרשים זרימת העבודה כולל בדרך כלל פעילויות שיעודן: לרשום את החריגות שדווחו, לנתח ולסווג אותן, להחליט על תגובה מתאימה כגון לתקן או להשאיר אותה כפי שהיא ולבסוף לסגור את דוח הפגמים. התהליך חייב להיות מלווה על ידי כל בעלי העניין המעורבים. מומלץ לטפל בפגמים מבדיקות סטטיות (בעיקר ניתוח סטטי) באופן דומה.

לדוח פגמים אופייני יש את המטרות הבאות:

- לספק לאחראים לטיפול ולפתרון בפגמים שדווחו מספיק מידע כדי לפתור את הבעיה
- לספק אמצעי מעקב אחר איכות תוצר העבודה
- לספק רעיונות לשיפור תהליך הפיתוח והבדיקה

דוח פגמים שנרשם במהלך בדיקות דינמיות כולל בדרך כלל:

- מזהה ייחודי
- כותרת עם סיכום קצר של האנומליה המדווחת
- תאריך שבו נצפתה האנומליה, הארגון המדווח והמחבר, כולל תפקידם
- זיהוי אובייקט הבדיקה וסביבת הבדיקה
- ההקשר של הפגם (למשל, מקרה בדיקה מופעל, פעילות בדיקה מבוצעת, שלב ב-SDLC ומידע רלוונטי אחר כגון טכניקת הבדיקה, רשימת פעולות לביצוע או נתוני הבדיקה הנמצאים בשימוש)
- תיאור הכשל המאפשר שחזור ופתרון, כולל השלבים שזיהו את החריגה, וכל יומני הבדיקה, קבצי ה-dump של מסד הנתונים, צילומי המסך או ההקלטות הרלוונטיים
- תוצאות צפויות ותוצאות בפועל
- חומרת הפגם (מידת ההשפעה) על האינטרסים של בעלי העניין או הדרישות
- עדיפות לתיקון
- סטטוס הפגם (למשל, פתוח, נדחה, כפול, ממתין לתיקון, ממתין לבדיקת אישור, נפתח מחדש, סגור, נדחה)
- הפניות (למשל, למקרה בדיקה)

חלק מנתונים אלה עשויים להיכלל באופן אוטומטי בעת שימוש בכלי ניהול פגמים (למשל, מזהה, תאריך, מחבר וסטטוס התחלתי). תבניות מסמכים עבור דוח פגמים ודוחות פגמים למשל ניתן למצוא ב- ISO/IEC/IEEE 29119-3.

6 כלי בדיקה — 20 דקות

מילות מפתח

בדיקות אוטומציה (test automation)

מטרות הלמידה של פרק כלי בדיקה

6.1 כלים תומכי בדיקות

FL-6.1.1 (K2) הסבר כיצד סוגים שונים של כלי בדיקה תומכים בבדיקות

6.2 יתרונות וסיכונים באוטומציה של בדיקות

FL-6.2.1 (K1) זהה את היתרונות והסיכונים באוטומציה של בדיקות

6.1 כלים תומכי בדיקות

כלי בדיקה תומכים ומקלים על פעילויות בדיקה רבות. דוגמאות לכך שכוללות, אך לא מוגבלות לכתוב:

- כלים לניהול – מגדילים את יעילות תהליך הבדיקות על ידי הקלה בניהול: מחזור החיים של פיתוח תוכנה (SDLC), דרישות, בדיקות, תקלות, ותצורה
- כלים לבדיקות סטטיות – תומכים בבודק במהלך ביצוע סקירות וניתוחים סטטיים (static analysis)
- כלים לעיצוב ויישום בדיקות – מסייעים ביצירת מקרי בדיקה, נתוני בדיקה ונהלי/הליכי בדיקה
- כלים לביצוע וכיסוי בדיקות – מקלים על ביצוע אוטומטי של בדיקות ומדידת כיסוי
- כלים לביצוע בדיקות לא-פונקציונליות – מאפשרים לבודק לבצע בדיקות לא-פונקציונליות שקשה או בלתי אפשרי לבצע ידנית
- כלי DevOps – תומכים בצינור האספקה של ה-DevOps, מעקב אחר זרימת העבודה, תהליך בנייה אוטומטי, אינטגרציה רציפה והפצה רציפה (CI/CD)
- כלים לשיתוף פעולה – מקלים על התקשורת
- כלים התומכים במדרגיות וסטנדרטיזציה של פריסה (לדוגמה, מכונות וירטואליות, כלים לניהול מכולות[13](containerization tools))
- כל כלי אחר המסייע בבדיקות (לדוגמה, גיליון אלקטרוני המשמש ככלי בדיקה בהקשר של בדיקות)

6.2 יתרונות וסיכונים באוטומציה של בדיקות

רכישת כלי לבדה אינה מבטיחה הצלחה. כל כלי חדש ידרוש מאמץ להשגת יתרונות אמיתיים ומתמשכים (למשל, עבור החדרת הכלי, תחזוקה והכשרה). ישנם גם סיכונים מסוימים הדורשים ניתוח ופעולות מנע (mitigation).

היתרונות האפשריים של שימוש בכלי בדיקה אוטומטיים כוללים:

- חיסכון בזמן על ידי הפחתת עבודה ידנית חוזרת ונשנית (למשל, ביצוע בדיקות נסיגה (regression), הזנת חוזרת ונשנית של נתוני בדיקה, השוואת תוצאות צפויות לעומת תוצאות בפועל, ובדיקה למול תקני הקידוד)
- מניעת טעויות אנוש פשוטות באמצעות עקביות ויכולת שחזור גבוהה יותר (למשל, בדיקות הנגזרות בעקביות מהדרישות, נתוני בדיקה הנוצרים בצורה שיטתית ובדיקות המבוצעות על ידי הכלי באותו סדר ובאותה תדירות)
- הערכה אובייקטיבית יותר (למשל, כיסוי) ואספקת מדדים שהם מסובכים מדי להפקה באופן ידני על-ידי בני אדם

[13] הערת עורך - **Docker** היא פלטפורמה עבור קונטיינריזציה. היא מאפשרת לארוז את האפליקציה עם התלויות שלה ב-container ולהנגיש אותו בצורה נוחה עם פיצ'רים רבים. כך, נוכל להפחית את כמות "הבאגים המייאשים" ולעזור לאפליקציה לרוץ בצורה חלקה בכל סביבה." https://thedevopscore.com/docker-basics/

- גישה קלה יותר למידע על בדיקות, לתמיכה בניהול בדיקות ולדיווח על בדיקות (למשל, סטטיסטיקות, גרפים ונתונים מצטברים על התקדמות הבדיקות, קצב פתיחת פגמים ומשך ביצוע הבדיקות)
- הפחתת זמני ביצוע הבדיקות כדי לאפשר זיהוי פגמים ותקלות מוקדם יותר, משוב מהיר יותר וזמן יציאה לשוק קצר יותר
- יותר זמן לבודקים לתכנון בדיקות חדשות, מעמיקות ויעילות יותר

הסיכונים האפשריים בשימוש בבדיקות אוטומציה כוללים:

- ציפיות לא מציאותיות לגבי יתרונות הכלי (ובכללם הפונקציונליות וקלות השימוש בו)
- הערכות לא מדויקות של זמן, עלויות והמאמצים הנדרשים בהחדרת הכלי, תחזוקת סקריפטים של בדיקות ושינוי תהליך הבדיקה הידני הקיים
- שימוש בכלי בדיקה כאשר בדיקה ידנית מתאימה יותר
- הסתמכות יתר על כלי הבדיקה. לדוגמה, התעלמות מהצורך בחשיבה ביקורתית אנושית
- תלות בספק הכלי שעלול לפשוט רגל, להפסיק לתמוך בכלי, למכור את הכלי לספק אחר או לספק תמיכה לקויה (למשל, תגובות איטיות לשאלות או פניות, שדרוגים ותיקוני פגמים)
- שימוש בתוכנת קוד פתוח (open-source software) שעלולה להינטש כלומר, לא יהיו עדכונים נוספים זמינים, או שהרכיבים הפנימיים שלה עלולים לדרוש עדכונים תכופים למדי כפיתוח נוסף
- אי תאימות של כלי האוטומציה לפלטפורמת הפיתוח
- בחירת כלי לא מתאים שלא עמד בדרישות הרגולציה ו/או בתקני הבטיחות

7 הפניות / סימוכין

תקנים

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) 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

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

ספרים ומאמרים

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley 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: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

מאמרים ותוכן מדפי אינטרנט

Brykczynski, B. (1999) “A survey of software inspection checklists,” ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” IEEE Transactions on Software Engineering 1(2), pp. 140-149

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” IEEE Transactions on Software Engineering 4(3), pp. 199-229

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158

Salman. I. (2016) “Cognitive biases in software quality and testing,” Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8 נספח א – יעדי לימוד / רמות ידע קוגניטיביות

יעדי הלמידה הבאים מוגדרים ככאלו שחלים על תוכנית לימוד (סילבוס) זו. כל נושא בתוכנית הלימוד ייבחן בהתאם ליעד הלמידה שנקבע עבורו. יעדי הלמידה מתחילים עם פועל (שהיא בבחינת מילת מפתח המייצגת פעולה) התואם לרמת הידע הקוגניטיבי המדרש עבורו, כמפורט להלן.

רמה 1: לזכור (K1)

המועמד יזהה ויזכור נושא או מושג.

פעלים (מילות מפתח): לזהות, להיזכר, לזכור, לשחזר, להכיר, לדעת.

דוגמאות

- "זהה יעדי בדיקה טיפוסיים."
- "זכור את המושגים של פירמידת הבדיקות."
- "זהה כיצד בודק מוסיף ערך לתכנון איטרציה ושחרור"

רמה 2: להבין (K2)

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

פעלים (מילות מפתח): לסווג, להשוות, להבדיל, להבחין, להדגים, להסביר, לתת דוגמאות, לפרש, לסכם.

דוגמאות:

- "סווג את האפשרויות השונות לכתיבת קריטריוני קבלה."
- "השווה בין התפקידים השונים בבדיקה" (חפש דמיון, הבדלים או שניהם).
- "הבחן בין סיכוני פרויקט לסיכוני מוצר" (מאפשר להבדיל בין מושגים).
- "הדגם את המטרה והתוכן של תוכנית בדיקות"
- "הסבר את השפעת ההקשר על תהליך הבדיקות"
- "סכם את הפעילויות של תהליך הבדיקה"

רמה 3: ליישם (K3)

המועמד יכול לבצע הליך כאשר הוא מתמודד עם משימה מוכרת, או לבחור את ההליך הנכון ולהחיל אותו בהקשר נתון.

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

דוגמאות:

- "החל תעדוף מקרי בדיקה" (צריך להתייחס לנוהל, טכניקה, תהליך, אלגוריתם וכו').
- "הכן דוח פגמים"
- "השתמש בניתוח ערכי גבולות כדי לגזור מקרי בדיקה"

הפניות / סימוכין (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

9 נספח ב – מטריצת נֶעֱקָבוּת: התוצאות העסקיות (BOs) למול יעדי למידה (LOs)

נספח זה מפרט את מספר יעדי הלמידה (learning objectives) ברמת הבסיס הקשורים לתוצאות העסקיות (business outcomes) - שהוגדרו כחלק ממטרות תוכנית לימוד זו - ואת העקיבות ביניהם (בין התוצאות העסקיות ברמת הבסיס לבין יעדי הלמידה ברמת הבסיס).

Business Outcomes: Foundation Level		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Understand what testing is and why it is beneficial	6													
BO2	Understand fundamental concepts of software testing		22												
BO3	Identify the test approach and activities to be implemented depending on the context of testing			6											
BO4	Assess and improve the quality of documentation				9										
BO5	Increase the effectiveness and efficiency of testing					20									
BO6	Align the test process with the software development lifecycle						6								
BO7	Understand test management principles							6							
BO8	Write and communicate clear and understandable defect reports								1						
BO9	Understand the factors that influence the priorities and efforts related to testing									7					
BO10	Work as part of a cross-functional team										8				
BO11	Know risks and benefits related to test automation.											1			
BO12	Identify essential skills required for testing												5		
BO13	Understand the impact of risk on testing													4	
BO14	Effectively report on test progress and quality														4

Chapter/ section/ subsection	Learning objective	K-level	BUSINESS OUTCOMES													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Chapter 1	**Fundamentals of Testing**															
1.1	**What is Testing?**															
1.1.1	Identify typical test objectives	K1	X													
1.1.2	Differentiate testing from debugging	K2		X												
1.2	**Why is Testing Necessary?**															
1.2.1	Exemplify why testing is necessary	K2	X													
1.2.2	Recall the relation between testing and quality assurance	K1		X												
1.2.3	Distinguish between root cause, error, defect, and failure	K2		X												
1.3	**Testing Principles**															
1.3.1	Explain the seven testing principles	K2		X												
1.4	**Test Activities, Testware and Test Roles**															
1.4.1	Summarize the different test activities and tasks	K2			X											
1.4.2	Explain the impact of context on the test process	K2			X			X								
1.4.3	Differentiate the testware that support the test activities	K2			X											
1.4.4	Explain the value of maintaining traceability	K2				X	X									
1.4.5	Compare the different roles in testing	K2										X				

1.5	**Essential Skills and Good Practices in Testing**																
1.5.1	Give examples of the generic skills required for testing	K2												X			
1.5.2	Recall the advantages of the whole team approach	K1										X					
1.5.3	Distinguish the benefits and drawbacks of independence of testing	K2			X												
Chapter 2	**Testing Throughout the Software Development Lifecycle**																
2.1	**Testing in the Context of a Software Development Lifecycle**																
2.1.1	Explain the impact of the chosen software development lifecycle on testing	K2						X									
2.1.2	Recall good testing practices that apply to all software development lifecycles	K1						X									
2.1.3	Recall the examples of test-first approaches to development	K1					X										
2.1.4	Summarize how DevOps might have an impact on testing	K2					X	X			X	X					
2.1.5	Explain the shift-left approach	K2					X	X									
2.1.6	Explain how retrospectives can be used as a mechanism for process improvement	K2					X					X					
2.2	**Test Levels and Test Types**																
2.2.1	Distinguish the different test levels	K2		X	X												
2.2.2	Distinguish the different test types	K2		X													
2.2.3	Distinguish confirmation testing from regression testing	K2		X													
2.3	**Maintenance Testing**																
2.3.1	Summarize maintenance testing and its triggers	K2		X					X								
Chapter 3	**Static Testing**																
3.1	**Static Testing Basics**																

1.5	**Essential Skills and Good Practices in Testing**																
1.5.1	Give examples of the generic skills required for testing	K2												X			
1.5.2	Recall the advantages of the whole team approach	K1										X					
1.5.3	Distinguish the benefits and drawbacks of independence of testing	K2			X												
Chapter 2	**Testing Throughout the Software Development Lifecycle**																
2.1	**Testing in the Context of a Software Development Lifecycle**																
2.1.1	Explain the impact of the chosen software development lifecycle on testing	K2						X									
2.1.2	Recall good testing practices that apply to all software development lifecycles	K1						X									
2.1.3	Recall the examples of test-first approaches to development	K1					X										
2.1.4	Summarize how DevOps might have an impact on testing	K2					X	X			X	X					
2.1.5	Explain the shift-left approach	K2					X	X									
2.1.6	Explain how retrospectives can be used as a mechanism for process improvement	K2					X					X					
2.2	**Test Levels and Test Types**																
2.2.1	Distinguish the different test levels	K2		X	X												
2.2.2	Distinguish the different test types	K2		X													
2.2.3	Distinguish confirmation testing from regression testing	K2		X													
2.3	**Maintenance Testing**																
2.3.1	Summarize maintenance testing and its triggers	K2		X					X								
Chapter 3	**Static Testing**																
3.1	**Static Testing Basics**																

4.3.2	Explain branch testing	K2		X												
4.3.3	Explain the value of white box testing	K2	X	X												
4.4	**Experience-based Test Techniques**															
4.4.1	Explain error guessing	K2		X												
4.4.2	Explain exploratory testing	K2		X												
4.4.3	Explain checklist-based testing	K2		X												
4.5	**Collaboration-based Test Approaches**															
4.5.1	Explain how to write user stories in collaboration with developers and business representatives	K2				X						X				
4.5.2	Classify the different options for writing acceptance criteria	K2										X				
4.5.3	Use acceptance test-driven development (ATDD) to derive test cases	K3					X									
Chapter 5	**Managing the Test Activities**															
5.1	**Test Planning**															
5.1.1	Exemplify the purpose and content of a test plan	K2		X					X							
5.1.2	Recognize how a tester adds value to iteration and release planning	K1	X									X		X		
5.1.3	Compare and contrast entry criteria and exit criteria	K2				X		X								X
5.1.4	Use estimation techniques to calculate the required test effort	K3							X		X					
5.1.5	Apply test case prioritization	K3							X		X					
5.1.6	Recall the concepts of the test pyramid	K1		X												
5.1.7	Summarize the testing quadrants and their relationships with test levels and test types	K2		X							X					
5.2	**Risk Management**															

5.2.1	Identify risk level by using risk likelihood and risk impact	K1							X						X	
5.2.2	Distinguish between project risks and product risks	K2		X											X	
5.2.3	Explain how product risk analysis may influence thoroughness and scope of testing	K2					X				X				X	
5.2.4	Explain what measures can be taken in response to analyzed product risks	K2		X			X								X	
5.3	**Test Monitoring, Test Control and Test Completion**															
5.3.1	Recall metrics used for testing	K1									X					X
5.3.2	Summarize the purposes, content, and audiences for test reports	K2					X				X					X
5.3.3	Exemplify how to communicate the status of testing	K2												X		X
5.4	**Configuration Management**															
5.4.1	Summarize how configuration management supports testing	K2					X		X							
5.5	**Defect Management**															
5.5.1	Prepare a defect report	K3		X						X						
Chapter 6	**Test Tools**															
6.1	**Tool Support for Testing**															
6.1.1	Explain how different types of test tools support testing	K2					X									
6.2	**Benefits and Risks of Test Automation**															
6.2.1	Recall the benefits and risks of test automation	K1					X						X			

10 נספח ג – הערות פרסום למהדורה (Release Notes)

גירסת ISTQB® Foundation Syllabus v4.0 הינה גירסת עדכון מז'ורית (major update) המבוססת על תוכנית הלימוד ברמת הבסיס (v3.1.1) ותוכנית הלימוד של בודק אג'ייל Agile Tester 2014. מסיבה זו, אין הערות פרסום מפורטות לכל פרק וסעיף. עם זאת, להלן סיכום השינויים העיקריים. בנוסף, קיים מסמך נפרד של הערות פרסות למהדורה, בו ארגון ISTQB® מספק מעקב בין יעדי הלימוד (LOs) בגרסה 3.1.1 של תוכנית הלימוד ברמת הבסיס, גרסת 2014 של תוכנית הלימוד של ה- Agile Tester, לבין יעדי הלמידה בגרסה החדשה של תוכנית הלימוד ברמת הבסיס v4.0, המראה אילו יעדי הלימוד (LOs) נוספו, עודכנו או הוסרו.

בזמן כתיבת תוכנית הלימוד (2022-2023) נמדד כי ניגשו כבר יותר ממיליון אנשים ביותר מ-100 מדינות לבחינה ברמת הבסיס, ויותר מ-800,000 בודקים הוסמכו ברחבי העולם. הציפייה שכולם קראו את תוכנית הלימוד ברמת הבסיס כדי לעבור את הבחינה, הופכת את תוכנית הלימוד ברמת הבסיס למסמך בדיקות התוכנה הנקרא ביותר אי פעם! עדכון מז'ורי זה נעשה ביחס למורשת זו וכן בכדי לשפר את השקפותיהם של מאות אלפי אנשים נוספים ברמת האיכות שארגון ISTQB® מספק לקהילת הבדיקות העולמית.

בגרסה זו כל יעדי הלימוד (LOs) נערכו מחדש על מנת להפוך אותם ליותר אטומיים, וכדי ליצור מעקב אחד לאחד בין יעדי הלימוד (LOs) והסעיפים של תוכנית הלימוד, ובכך למנוע מצב שבו יהיו תכנים שאין להם LOs מתאימים. המטרה היא להפוך את הגרסה הזו לקלה יותר לקריאה, להבנה, ללמידה ולתרגום, תוך התמקדות בהגדלת התועלת המעשית ובאיזון בין ידע ומיומנויות.

מהדורה מז'ורית זו כוללת את השינויים הבאים:

- הפחתת הגודל הכולל של תוכנית הלימוד. תוכנית הלימוד אינה ספר לימוד, אלא מסמך המשמש כמתווה עבור המרכיבים הבסיסיים של קורס וכמבוא לבדיקות תוכנה; לרבות באילו נושאים יש להתמקד ובאיזו רמה. לכן, יש לשים לב במיוחד לכך:
  - שברוב המקרים הטקסט אינו מכיל דוגמאות. אספקת דוגמאות זוהי משימה שספקי הדרכות וההכשרות נדרשים לספק, כמו גם את התרגילים שיש לבצע במהלך התירגול.
  - שבוצע מעקב ל"רשימת הביקורת לכתיבת תוכנית הלימוד", רשימה המציעה את גודל הטקסט המרבי עבור יעדי הלימוד (LOs) בכל רמת לימוד K-Level (K1 = מקסימום 10 שורות, K2 = מקסימום 15 שורות, K3 = מקסימום 25 שורות)
- הפחתת מספר LOs בהשוואה לתוכנית הלימוד של Foundation v3.1.1 ו-Agile v2014
  - 14 LOs של K1 בהשוואה ל-21 LOs שקיימים ב-FL v3.1.1 (15) וב-AT 2014 (6)
  - 42 LOs של K2 בהשוואה ל-53 LOs שקיימים ב-FL v3.1.1 (40) וב-AT 2014 (13)
  - 8 LOs של K3 בהשוואה ל-15 LOs שקיימים ב-FL v3.1.1 (7) וב-AT 2014 (8)
- ישנן הפניות נרחבות יותר לספרים ומאמרים קלאסיים ו/או מכובדים על בדיקות תוכנה והנושאים הקשורים.
- **שינויים עיקריים בפרק 1 (יסודות בדיקות התוכנה)**
  - תת-הסעיף על מיומנויות בדיקה הורחב ושופר
  - התווסף תת-סעיף על הצוות השלם (K1)
  - תת-סעיף עצמאות הבדיקות הועבר מפרק 5 לפרק 1

- **שינויים עיקריים בפרק 2 (בדיקות לאורך מחזור חיי פיתוח תוכנה)**
  - תתי-הסעיפים 2.1.1 ו-2.1.2 שוכתבו ושופרו, וה-LOs התואמים עודכנו
  - התמקדות רחבה יותר בפרקטיקות כמו: גישת בדיקות תחילה (K1), הזזה-לשמאל (K2), רטרוספקטיבות (K2)
  - תת-סעיף חדש על בדיקות בהקשר של DevOps (K2)
  - רמת בדיקת האינטגרציה מפוצלת לשתי רמות בדיקה נפרדות: בדיקת אינטגרציה של רכיבים ובדיקת אינטגרציה של מערכת
- **שינויים עיקריים בפרק 3 (בדיקות סטטיות)**
  - הוסר תת-הסעיף על טכניקות סקירה, ביחד עם יעד הלימוד ברמה של K3 (ממש טכניקת סקירה)
- **שינויים עיקריים בפרק 4 (ניתוח ועיצוב בדיקות)**
  - בדיקת מקרי השימוש הוסרה (אך עדיין קיימת בתוכנית הלימוד של מנתח בדיקות מתקדם – Advanced Level Test Analyst)
  - התמקדות מורחבת בגישה המבוססת על שיתוף פעולה בבדיקות: התווסף יעד לימוד חדש ברמה של K3 בנושא השימוש ב-ATDD כדי להפיק מקרי בדיקה, ושני יעדי לימוד חדשים ברמה של K2 בנושא סיפורי משתמשים וקריטריוני קבלה
  - בדיקה וכיסוי החלטות הוחלפו בבדיקה וכיסוי ענפים (ראשית כיוון, שכיסוי ענפי נפוץ יותר בפועל; שנית, סטנדרטים שונים מגדירים את ההחלטה בצורה שונה, בניגוד ל"ענף"; שלישית, זה פותר פגם עדין אך חמור מגירסת תוכנית הלימוד FL 2018 הישנה שבה נטען ש"כיסוי החלטות של 100% מרמז על כיסוי של 100% של המשפטים" - המשפט הזה אינו נכון במקרה של תוכניות ללא החלטות)
  - שיפור של החלק הקשור לערך המוסף של בדיקת קופסה לבנה
- **שינויים עיקריים בפרק 5 (ניהול פעילות הבדיקות)**
  - תת-סעיף על אסטרטגיות/גישות לבדיקות הוסר
  - התווסף יעד לימוד חדש ברמה של K3 בנושא של טכניקות אומדן להערכת מאמץ הבדיקות
  - התמקדות מורחבת במושגים ובכלים הידועים הקשורים ל-Agile בניהול בדיקות: תכנון איטרציה ושחרור (K1), פירמידת בדיקות (K1) ובדיקות רביעים (K2)
  - החלק שקשור לניהול סיכונים בנוי טוב יותר, ומתואר על ידי ארבע פעילויות עיקריות: זיהוי סיכונים, הערכת סיכונים, הפחתת סיכונים וניטור סיכונים
- **שינויים עיקריים בפרק 6 (כלים תומכי בדיקות)**
  - התוכן לחלק מבעיות הקשורות לאוטומציה של בדיקות הצטמצם, כיוון שהוא בבחינת ידע מתקדם מדי עבור רמת הבסיס
  - הוסרו החלקים בנושאים של: בחירת כלים, ביצוע פרויקטי פיילוט והכנסת כלים לארגון

11 נספח ד – הבהרות מיוחדות (עבור המהדורה בעברית בלבד)

מטרת נספח זה היא להוסיף מידע ו/או לתת הסברים מפורטים (שחלקם אף כוללים דוגמאות) לתכנים שנידונו בפרקים שבתוכנית הלימוד. הסיבה העיקרית לקיומו של נספח זה נעוצה בכך שעורך המהדורה בעברית מצא לנכון שהתוכן המקורי (באנגלית) הוא חסר ועל כן הוא נדרש להבהרות ו/או להרחבות.

ההבהרות ו/או ההרחבות קיימים אך ורק בגרסה בעברית ואינם נדרשים בבחינת ההסמכה!

להלן רשימת המקומות שעבורן מצא העורך לנכון להבהיר ו/או להרחיב:

תת-סעיף: 2.1.3 בדיקות כמנוע לפיתוח תוכנה

ההבהרה הנדרשת: להבדל בין ATDD לבין BDD, ובעיקר בנוגע לשימוש ב: Given-When-Then

ההבדל העיקרי הוא שATDD מיועד לגזירת בדיקות מבוססות קריטריוני קבלה, בעוד שBDD הוא תיאור הדרישות באופן כזה שיהיו קלים להבנה ויכללו בדיקות (קבלה).

התבנית "Given-When-Then" ("בהינתן-כאשר-אז") מיועדת בעיקר עבור הבדיקות, ומשמשת הן את בדיקות הקבלה המוגדרות בATDD והן את הבדיקות (הקבלה) המוגדרות עבור התכונה (feature) בBDD. לעומתה התבנית "As a-I want to be able to-So that" ("בתור כ-אני רוצה להיות מסוגל-אזי") מיועדת להגדרת התכונה (feature) הנדרשת וקיימת אך ורק לשימוש בBDD.

המטרה העיקרית של הBDD היא, כיצד המערכת צריכה לעבוד, מהצד של המשתמש.

המטרה העיקרית של הATDD היא, להתמקד בעיקר בהבטחת התכונה שתענה על הצרכים העסקיים, ולרוב מונעת על ידי קריטריוני הקבלה המוסכמים על ידי האנליסטים העסקיים, הבודקים והמפתחים.

הBDD בחלק שכולל את בדיקות הקבלה של התכונה (feature), יהיה זהה לבדיקות הקבלה שהוגדרו במסגרת הATDD.

דוגמה להמחשת ההבדלים:

The feature which described in the BDD:

Feature: Login to the system

As a registered user

I want to be able to log in to the system

So that I can access my account

The acceptance tests which exist both in the acceptance criteria of the described feature in the BDD, and the test scenarios in the ATDD:

Scenario 1: Successful login

Given the user is on the login page

When the user enters a valid username and password

and clicks the login button

Then the user should be redirected to the dashboard

Scenario 2: Failed login due to invalid credentials

Given the user is on the login page

When the user enters an invalid username or password

and clicks the login button

Then the user should see an error message

תת-סעיף: 4.2.1 מחלקות שקילות

ההבהרה הנדרשת: תוכן הפסקה הראשונה

טכניקת הבדיקה של מחלקות שקילות מבוססת על כך שניתן לחלק את מרחב הקלטים שתוכנה מקבלת לקבוצות. כל קבוצה מכילה ערכי קלט שעבורם התוכנה מריצה בדיוק את אותן שורות קוד. הרעיון המרכזי הוא שאם יש פגם בשורת קוד מסוימת, כל ערך קלט שניקח מתוך הקבוצה שמריצה שורה זו, יזהה את הפגם. גם אם הקבוצה מכילה הרבה מאוד ערכי קלט שונים, מספיקה בדיקה אחת בודדת עם ערך אחד מהקבוצה, על מנת לזהות את הפגם.

תת-סעיף: 4.3.2 בדיקת ענפים וכיסוי ענפים (Branch Coverage)

ההבהרה הנדרשת: כיסוי ענפים (Branch Coverage) וכיסוי החלטות (Decision Coverage)

בכיסוי ענפים יש לכסות את כל העברות השליטה בקוד הנבדק. ענף הינו העברת שליטה בין שני צמתים בתרשים הזרימה, כלומר ענף הינו חץ בתרשים הזרימה, המייצג העברת שליטה בין שתי הצהרות או צמתים. פריטי הכיסוי הינם הענפים, כלומר החיצים בתרשים הזרימה. ענף יכול להכיל משפטים ללא התניות (ענף **בלתי מותנה**, כלומר, קוד המבוצע בקו ישר ורציף), או משפטים הכוללים התניות (ענף **מותנה**, כלומר, ענף הכולל נקודת/ות החלטה). במקרה של ענף מותנה, יש לכסות את כל התוצאות של נקודות ההחלטה, כלומר נתיב ה-True ונתיב ה-False.

כיסוי החלטות Decision Coverage הינה תת-טכניקה של כיסוי ענפים, לפיה יש לכסות את כל היציאות של נקודות ההחלטה בקוד, כלומר **היא מכסה את הענפים המותנים בלבד**.