סיכום הבטחת איכות תוכנה

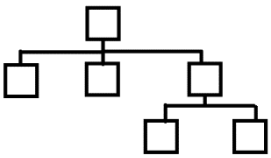
הסברים יותר מפורטים למדדים בהמשך

מדדים לאיכות תוכנה:

1. נכונות- התוכנה מפיקה תוצאות נכונות
2. אמינות- מערכת לא קורסת, עבודה רציפה לאורך זמן בתנאים קבועים, יתירות
3. שימושיות- חווית משתמש(עד כמה המערכת פשוטה להפעלה, השקעה של משאבים קטנים בשביל תפוקה גדולה)
4. נגישות- תמיכה בבעלי מוגבלות, תוכן נתפס(חליפות לתוכן נתפס), תוכן מובן, תוכן בר הפעלה, תוכן יציב (יציבות מערכת)
5. תחזוקה- שמירה על עבודה רציפה לאורך זמן. פעילות שמתבצעת כאשר המוצר אצל הלקוח
6. תחזוקתית/שימוש חוזר- היכולת של התוכנה לקבלת תחזוקה בעתיד
7. ביצועים- זמן תגובה עומס- פניות רבות בו זמנית של משתמשים רבים
   דחק- עומס קיצון נחפש את ה'גבולות' של יכולת התוכנה
   נפח- קבצי ענק
   התאוששות- חזרה לשגרה לאחר שהמערכת סופגת כשל
8. עקיבות- היכולת לקשר בין דרישות לבין המימוש שלהן בקוד(או בבדיקות). היכולת לזהות איפה לתקן ולעדכן, לדבר עם הלקוח בשפה שלו

בדיקות תוכנה בארגון:

1. ארגונים פונקציונליים-"היררכיה"

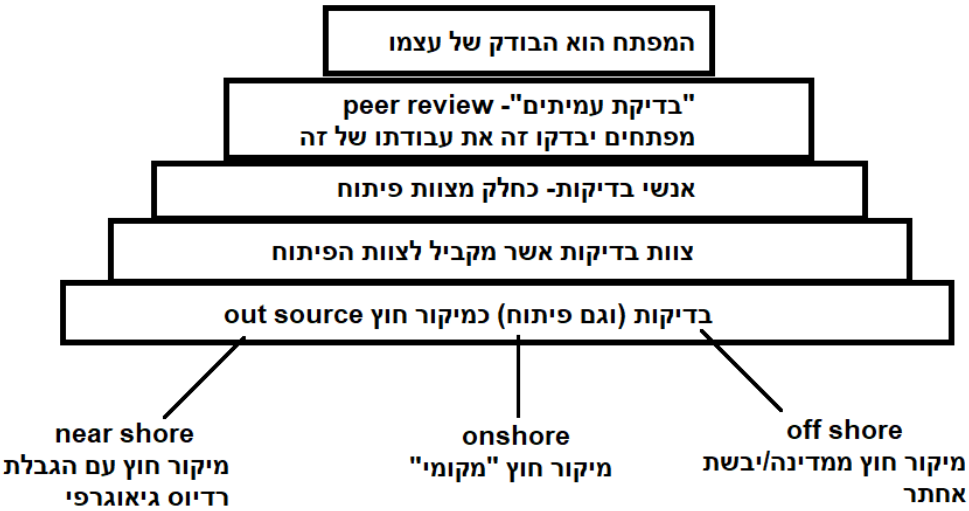


התמחות כללית "אחד עושה הכל"

2. ארגונים מטריציוניים- לעובד יש שני מנהלים- מנהל הפרויקט ומנהל מקצועי

התמחות יותר ספציפית "אחד עושה רק משהו ספציפי"

דרגת עצמאות= עד כמה הבודק עצמאי (רחוק) מהמפתח



ממשקים של אנשי בדיקות בארגון

1. הנהלה- לו"ז, תקציבים
2. פיתוח- קבלת קוד לבדיקה דיווח וטיפול בתקלות
3. גורם עסקי- (מכירות, שיווק) עדכונים על דרישות הלקוח
4. תמיכה תכנית- (שירות לקוחות) העברת ידע על גרסאות חדשות מענה למקרים ששירות לקוחות לא פותר לבד
5. כתיבה טכנית- מדריכים למשתמש מסכי עזרה
6. IT- שירות התקנות, סביבות מיוחדות
7. Devops

תהליך הבדיקות הבסיסי

שלב	הסבר
תכנון	פירוק פונקציונלי, בחירת אסטרטגיה לבדיקות, הגדרת נהלי עבודה, ניהול סיכונים, הגדרת לוחות זמנים...
ניתוח ועיצוב	ניתוח- נגדיר תנאי בדיקה (אם כאשר אזי) ומקרי בדיקה- הוראות בדיקה לבודק (צעד לבצוע מול תוצאה רצויה)
יישום ובצוע	יישום- התקנות, הכנת בסיסי נתונים, הכנת מנות בדיקה...בצוע- הרצת הבדיקות, ודווח תקלות
הערכה ודיווח	נוודא שאנו משיגים את יעדי הבדיקות, כפי שהוגדרו בשלב התכנון. דיווח ברמת סיכומים- דיווח להנהלת הפרויקט על סיום הבדיקות- הסקת מסקנות, סיכום הבדיקות
סגירה	סיום הפרויקט

קלט לתהליך בדיקות- תוכנה אפיון מסמך דרישות

תכנון plan-

- להגדיר מה נבדק ומה לא
- מתי ניתן להפסיק לבדוק? -קריטריון יציאה
- פירוק פונקציונאלי- "חלקת עבודה" לבודקים
- לוחות זמנים תכנית עבוד
- צרכים מיוחדים
- גישת בדיקות עקרונית

- שלב התכנון הוא באחריות מנהל הבדיקות STP-software test plan

שלב הניתוח והעיצוב-

מבוצע ע"י הבודקים עצמם המקבלים מסמך STP וגישת בדירות עקרונית

- ניתוח- נגדיר "תנאי בדיקה"-"תרגום" שח דרישות לפורמט בר בדיקה Test Condition לוקחים משפט רגיל והופכים אותו לסגנון של "אם:... כאשר:... אזי:...."
- עיצוב- נפתח מקרי בדיקה (נגזרים מהניתוח)

צעד לביצוע	תוצאה רצויה
נשתמש במתודולוגיה לעיצוב מקרי בדיקה	תיגזר מתוך הדרישות ומתוך תנאי הבדיקה שנוצרו בשלב הניתוח

המסמך שמגדיר את תנאי הבדיקה ומקרי הבדיקה נקרא STD-software test design

שלב היישום והביצוע-

- יישום: תוכנה מותקנת לבדיקה, Data (סינטטי או ערבול לData אמיתי)
מגדירים מנות בדיקה-test shits
סביבה תומכת תוכנה- רישיונות, התקנות. (הכנת כלי הבדיקה עצמם)
- ביצוע: מריצים בדיקות מדווחים על תקלות וממליצים על פתרונות, מדווחים סטאטוס התקדמות

שלב הערכה והדיווח

- הערכה: מנהל הבדיקות מבצע אומדן האם אנו קרובים לקריטריון יציאה
- דיווח: כתיבת מסמך סיכום הבדיקות נקרא STR-software test report המסמך מכיל את מצב הבדיקות (מה רץ, מה עבר) ומצב איכות התוכנה (אילו תקלות פתוחות מה לא יציב)

- שלב התכנון הוא באחריות מנהל הבדיקות STP-software test plan

פלט התהליך: תוכנה שעברה תיקונים + יכולת לדווח על מצב התוכנה הנבדקת + יכולת לדווח על מהלך הבדיקות שנעשו

עקרונות בדיקה

1. בדיקות מראות אחוז מסוים מהתקלות - לא ניתן לתכנן סט בדיקות שיבטיח שכל התקלות- יתגלו.
2. גילוי מוקדם- רצוי ויעיל - תמיד נעדיף לבדוק את התוכנה מוקדם בזמן.
3. "צרות באות בצרורות" Bug Clustering - אם מצאנו תקלה באזור מסוים בתוכנה- כנראה שיש תקלות נוספות.
4. פרדוקס מדביר החרקים - יש חשיבות לגוון בבדיקות לעדכן מקרי בדיקה וחלופות אחרות
5. בדיקות מתישות אינן ישימות - לא ניתן לבדוק כל קלט, כל קומבינציה, כל קונפיגורציה... לכן יש צורך במנגנון בחירה של מקרי בדיקה מייצגים.
6. בדיקות הינן תלות הקשר - יש חשיבות להבנת תחום הדעת (קהל היעד, מטרה, שוק) של התוכנה הנבדקת
7. "שקריות" העדר שגיאות - גם אם תוכנה הייתה נטולת פגמים משמעותיים אין זה מחייב שהתוכנה מתאימה למשתמשים.

סוגי בדיקות

- **ידני/אוטומטי**
  ידני: בדיקה ע"י אדם
  אוטומטי: בדיקה ע"י מכונה

- **דינמית/סטטית**
  דינמית: כוללת את הרצת קוד המערכת הנבדקת
  סטטית: קוד המערכת הנבדקת לא רץ

- **קופסא שחורה/קופסא לבנה/קופסא אפורה**
  קופסא שחורה: הקוד לא חשוף אלינו, אנחנו שולטים על הקלט ובודקים את הפלט
  קופסא לבנה: הקוד חשוף אלינו ואני שולטים על הקלט ועל הפלט ובודקים גם את הקוד עצמו
  קופסא אפורה: לא רואים את הקוד אבל רואים "עקבות" שהקוד משאיר (Log, הדפסות לקונסול, DB- קורים אחרי זמן ריצה) רואים בעצם את התנהגות הקוד תוך כדי הריצה

- **פונקציונאלי/לא פונקציונלי**
  פונקציונאליות: בודקים מה עושה המערכת אל מול מפרט הדרישות/האפיון
  לא פונקציונאליות: בדיקת שימושיות, ביצועים, אמינות, ניידות(פלטפורמות שונות), אבטחה. בדרך כלל מתבצעת לא מול דרישות אלה יותר מול סטנדרטים, תקנים

- **בדיקת ערכי קצה**

- **פרוגרסיה/רגרסיה/שפיות/אימות**
  פרוגרסיה: לבדוק את כל "הפיצ'רים"/היכולות החדשות במערכת הנבדקת (כלומר הדברים החדשים שנוספו מהגרסה הקודמת לחדשה)
  רגרסיה: לבדוק שיכולות ותיקות במערכת הנבדקת לא הושפעו עקב הוספת היכולות החדשות (הכי מתאים לבדוק ע"י אוטומציה)
  שפיות: בדיקה שנועדה לוודא מוכנות מערכת לבדיקה מעמיקה במעבדה
  אימות: בדיקה שנועדה לבדוק האם תיקון תקלה פתר את הבעיה

- **בדיקות קבלה- בדיקות שמתבצעות ע"י הלקוח**

בדיקות שימושיות:

סוג הבדיקה	הסבר
usability שימושיות	חווית משתמש- נוחות, יעילות, שביעות רצון. עד כמה התוכנה מובנת, אינטואיטיבית, קלה להפעלה.
נגישות	תחום בתוך שימושיות – תמיכה במשתמשים בעלי מוגבלויות. עבודה לפי תקן
לוקליזציה	תחום בתוך שימושיות- התאמה לשפה ותרבות מקומיים
אמינות	הצורך להבטיח שבכל הפעלה (גם אם לאורך זמן)- קבל את אותן התוצאות. מגנון של יתירות (גיבויים)
תחזוקה	עבודה לאורך זמן- במצב בו המערכת הנבדקת in production
בצועים (ועומסים)	להעמיס שרת (ביקושים רבים בו זמנית, קבצי ענק, דחק- עומס שיא)- ולוודא התנהלות. זמני בצוע, שימוש השרת במשאבים

שיטות עבודה לבדיקת שימושיות

1. להיעזר במשתמשים- לבדוק משתמשים במעבדה / שחרור גרסת "ביטא" לבדיקה אצלם בעיות אפשריות:
- אפקט הות'ורן- אפקט ריצוי המראיין אנשים יתנהגו בצורה שונה כאשר הם יודעים שבוחנים אותם
-ניסוי אש- לחץ חברתי, השפעת האחד על האחר. אי קבלת דעות אובייקטיבית
2. ייעוץ מומחים- פסיכולוגיה / ייעוץ לשוני
3. בודקים- " היוריסטיקת השימושיות של נילסון" (רשימת כללי אצבע בבדיקת שימושיות)
https://firststepsinux.wordpress.com/2012/05/05/jakob-nielsen-usability-heuristics/

בדיקת אמינות

מדד אמינות- MTBF זמן ממוצע בין כשלים

מה בודקים?

1. המשך עבודה רציפה
2. התאוששות המערכת מכשל
3. יתירות- יש יותר ממה שנדרש לשם המשכת עבודה רצופה
   - יתירות בתקשורת- מנגנונים כגון checksum
   - יתירות בחומרה- RAID מערך דיסקים כגיבוי
   - יתירות בתוכנה- N version programing- משלבים בתוכנה אלגוריתמים שונים שמיועדים לאותה בעיה

תכנון בדיקת אמינות
במהלך תכנון של כל בדיקה אנו מגדירים גם קריטריון יציאה.
בבדיקת האמינות קריטריון היציאה יהיה במונחי MTBF
נשלב בדיקות MTBF בעת עומסים
גישה שלילית: מייצרים אירועים שלילים למערכת כדי לוודא את משך וטיב ההתאוששות
סביבה: עבודה בלחות, חום גבוה, תוכנות עזר לא תקינות
בתקשורת: ניתוקים, הפרעות, השבתת שרת
בתוכנה: פרמטרים לא אמיתיים, שבירת לינקים, כפיית Time out
fault injection- הזרקת כשלים למערכת בזמן ריצה

בדיקות תחזוקה
המוצר אצל הלקוח, התנאים משתנים- אינם תנאי מעבדה

תועלת גבוהה+ עלות תיקון נמוכה **לתקן!**	תועלת גבוהה+ עלות תיקון גבוהה **?**
תועלת נמוכה+ עלות תיקון נמוכה **?**	תועלת גבוהה+ עלות תיקון גבוהה **להחליף!**

מוצר לא נשאר באותה משבצת לאורך זמן כי

1. התועלת יורדת (מבחינה עסקית)
2. קוד בן 5 שנים ומעלה
3. עבודה באמולציה- הרצת המערכת בפלטפורמה אחרת
4. MTBF מתקצר- תקלות נהיות תכופות יותר
5. מוצר שעבר תיקונים רבים-"ספגטי" קשה להבנה

אתגרים בבדיקות תחזוקה

1. מערכת ישנה- לא תמיד יש מי שמבין אותה לגמרי
2. בכל עדכון למערכת יש צורך לוודא שדבר לא התקלקל (סדיקות רגרסיה)
3. כמה רגרסיה לעשות ועל מה

Impact Analysis- דיון מקדים עד כמה התיקון המוצע ישפיע על המערכת

בדיקות תחזוקתיות

1. עד כמה מודולרית (כמה קל לפרק ולהרכיב)
2. עד כמה קל להחליף רכיבים
3. עד כמה קל לבודד בעיה
4. עד כמה קל לבדוק בעקבות בעיה

בדיקות הסבה/נדידה
העברת נתונים ממקום למקום
נדידה: העברה כמו שהם
הסבה: העברת נתונים תוך שינוי הנתונים ביעד
הדגש- וידוא מעבר תקין, וידוא שמה שלא צריך לעבור לא עבר, תיעוד טעויות
הבדיקה מתבצעת ע"י כתיבת שאילתות גם במקור וגם ביעד

עומסים

איך בודקים עומס
virtual user- מחולל קריאות לשרת דרך תוכנה ייעודית
אתגרים:

1. אפקט בדיקה(Probe effect): יש להפריד בין מחולל קריאות לשרת דרך תוכנה ייעודית
2. כדי "לרמות" את השרת שיראה בקשות ממקורות שונים (ip spoofing) – תחפושות שונות לאותו ip

בדיקת עומסים- כיוונון- fine tune- נשנה הגדרות בשרת בעת עומס ונבדוק את

1. היחס בין כמות הבקשות שהגיעו לשרת בשנייה 1 (request per second) לעומת כמות הבקשות שנענו ע"י השרת בשנייה אחת (hit pre second)
2. כמה זמן בממוצע לוקח להשלים בקשה (duration)

אפשרויות להתמודדות עם עומס

1. Proxy- שרת ששומר את התשובות הקודמות
2. Cluster- נאגד מס' שרתים יחד ונפנה לכלי אשר ינתב את הבקשות ביניהם
3. Mirror- אוסף שרתים זהים אשר אינם מחוברים ביניהם הלקוח בוחר למי לגשת

בדיקות סטטיות

סקרים

סוגי סקרים

1. Informal review: סקר לא רשמי, התייעצות בין עמיתים בד"כ אין תיעוד
2. Walkthrough: "קריאה מודרכת" מחבר המסמך מנחה דיון בו מבהירים את תכולת המסמך
3. סקר טכני: סקר שנועד להבנה, אישור של פעילות ינוהל ע"י מנחה שאינו מחבר המסמך למשל סקר נכונות, אנליזה
4. **Audit: סקר שנעשה ע"י ארגון חיצוני בעיקר למטרות קבלת תקן**

מחזור חיים של סקר

תכנון: בחירת נושא הסקר, שיטת הסקר, תזמון הסקר, קהל היעד.

↓

kick off: הנעה- פרסום, חלוקת עבודה

↓

הכנה אישית: משתתפי הסקר מכינים חומר רקע, שאלות

↓

פגישה (הסקר)

↓

תיקונים: בעקבות ההחלטה בפגישה (בסקר)

↓

מעקב: אם נדרש...

שילוב תהליכי בדיקה בתוך תהליכי פיתוח

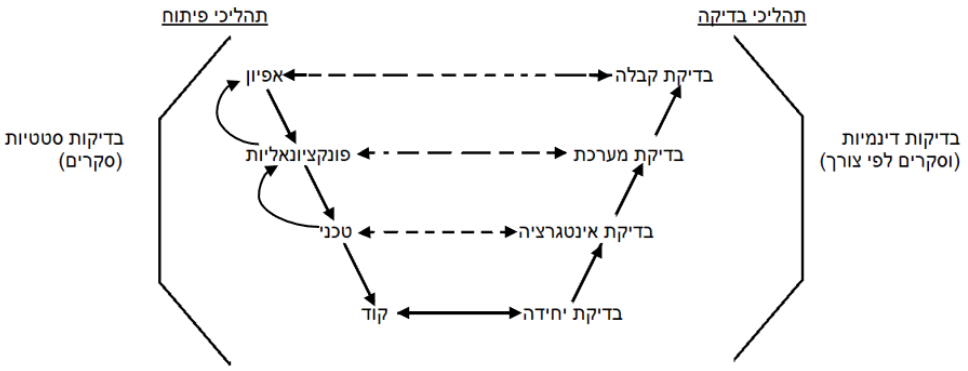
רמות הבדיקה (בסדר עולה)

1. **בדיקות יחידה**
Component test, unit test: בדיקות היחידה התכנותית הקטנה ביותר האפשרית
יבוצעו (בד"כ) ע"י קבוצות פיתוח, על סביבת פיתוח, שימוש בסימולטורים ודרייברים.
2. **בדיקות אינטגרציה**
בדיקת תקשורת בין יחידות שונות
יבוצעו ע"י יחידה מיוחדת לשם כך/בודקים/מפתחים (תלוי בארגון)
3. **בדיקת מערכת**
בדיקת מערכת כמכלול אחד התייחסות לדרישות לאפיונים לתנאי בדיקה יבוצעו ע"י קבוצת בדיקות
יבוצע בסביבה הקרובה ביותר לסביבת הלקוח
4. **בדיקות קבלה**
נבדוק שמה שמופיע באפיון תואם את המערכת בשונה מבדיקת מערכת לא נחפש תקלות אלה ננסה למצוא התאמה לאפיון
הבדיקה כוללת נציגי לקוח- בבדיקת האלפא הלקוח מגיע אל המעבדה שלנו, בבדיקת הביטא נגיע לסביבת הלקוח

ברמה מסוימת ניתן להפעיל מקרי בדיקה שונים
כל האיברים ברמת בדיקה כלשהי שווים
נעבור רמת בדיקה כאשר מתממש קריטריון יציאה אשר נקבע בתחילת התהליך

גישות בפיתוח

1. מודל קווי-
מסירת תוצר ללקוח פעם אחת.
הדרישות מוגדרות טרם תחילת הפיתוח. יותר בדיקות רגרסיה, יותר בדיקות התקנה.
2. מודל איטרטיבי-
דרישות מתפתחות תוך כדי עבודה
מסירה חוזרת ללקוח
3. מודל V-



בכל איטרציה יש בדיקת קבלה
האם אנו בונים את המערכת בצורה נכונה? Verification (צורך פנימי)
האם אנו בונים את המערכת הנכונה? Verification (צורך חיצוני)

צד ימין של מודל v: 4 רמות בדיקה, כל רמת בדיקה היא פעילות בדיקה אשר נועדה למטרה מסוימת
בכל רמת בדיקות נפעיל מחדש את תהליך הבדיקות הבסיסי

4. גישת "מפץ גדול"- להרכיב הכל בפעם אחת אם יש תקשורת- טוב ואם לא... צריך לחפש

5. גישת bottom-up, top-down- הרכבת מערכת לפי שכבות, כל פעם מוסיפים שכבה.
top-down- מתחילים מהGUI ומוסיפים שכבות עד שמגיעים לבסיס הנתונים

bottom-up: מתחילים מהdata base ומוסיפים שכבות עד לGUI

6. גישת increment- הרכבת המערכת לפי הפונקציונאליות שלה

טבלאות החלטה

שיטה טובה כאשר הפלט הוא תוצאה של כמה קלטים
- נפרוס את כל טבלת ההחלטה
- ננפה מקרי בדיקה מיותרים

כמות האפשרויות= הכפלת מס' האופציות של כל הפרמטרים

פרמטרים/ בדיקות	בדיקה1	...	בדיקהn
פרמטר1 . . . פרמטר N			
תוצאה רצויה			

דיאגרמת מצבים

שיטה טובה כאשר המערכת יכולה להיות במס' סופי של מצבים, לא ניתן להיות ביותר ממצב אחד בוז זמנית

- יש לזהות את המעברים
- יש לבדוק מעברים חוקיים ומעברים לא חוקיים

יישום וביצוע בדיקות

יישום בדיקות- הכנת סביבת העבודה בהתאם לרמת הבדיקה, מנות בדיקה, ארגון מקרי הבדיקה בקבוצות לשם קלות תפעול

שפיות- מנת בדיקה ספציפית, המכילה בדיקות "קצה-לקצה" של המערכת, בדיקות מדגמיות כדי לוודא מוכנות מערכת לבדיקה.

ביצוע בדיקות- ביצוע בדיקות לפי מקרי הבדיקה, דיווח על סטאטוס התקדמות הבדיקות, דיווח וטיפול בתקלות.

דיווח וטיפול בתקלות

תקלה- תוצאה רצויה ממקרה הבדיקה שונה מתוצאה בפועל (נצפית במהלך הבדיקה)

בידוד- גם אם נצפות מס' תופעות יחד יש לדווח עליהן בנפקד

דיוק- לתת תיאור מדויק של מה קרה האם תמיד קורה האם תלוי במשהו

מידע שלם- יש לצרף מידע שתומך בדיווח (log, צילומי מסך...)

אובייקטיביות- דיווח שמסתמך על מסמכי אפיון, דרישות ברורות

#טופס דיווח תקלה

שם:

תיאור:

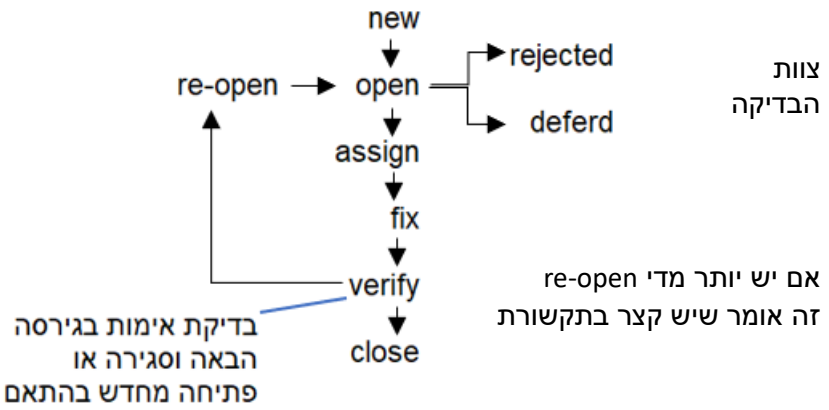
מס' גרסה:

אזור בתוכנה:

חומרה: (מידת ההשפעה של התקלה על המשתמש- נקבע ע"י הבודק)

עדיפות: (מידת ההשפעה של התקלה על הארגון-שיקול עסקי נקבע ע"י הנהלת הפרויקט)

מחזור חיים של דיווח תקלה



#תקלה נדחית- טעות בדיווח, טעות בהבנת התוכן, טעות בשימוש
#תקלה מוקפאת- עדיפות נמוכה, לא לתיקון במהלך הגרסה הזאת

- רק צוות הדיווח יכול להעביר מ open לclose

בדיקות קופסא לבנה

בדיקות בהן הקוד חשוף בפני הבודק כמו תקינות מבנה פנימי, יעילות

אחוז כיסוי- $\frac{\text{סך הבדיקות שבוצעו}}{\text{סך הבדיקות שיש לבצע}} \times 100$

גישות ל"סך הבדיקות שיש לבצע" בקופסא לבנה

1. Control-flow

כיסוי הצהרות: ייצור מקרי בדיקה לכל **הצהרה** בקוד. (**הצהרה-** פעולות לביצוע בקוד. בשמה הגדרה, קלט, פלט)

כיסוי החלטות: ייצור מקרי בדיקה כך שיפעלו על כל מוצאי ה**החלטות** בקוד. (**החלטות-** חישוב לשם שינוי כיוון: if, לולאות)

כיסוי תנאים מורכבים: בדיקת כל האופציות וכל תנאי נפרד גם אם תוצאת התנאי זהה (**תנאי מורכב-** החלטה אשר מכילה בתוכה תנאים הקשורים בקשרים לוגים) *בקוד העוסק בחיי אדם רוב התקנים מחייבים לבדוק תנאי זה*

כיסוי נתיבים: נחפש תנאי מעבר על הקוד מקצה לקצה

cyclomatic complexity(סיבוכיות ציקלומטית)(CC)- סך הנתיבים לכיסוי מלא של הקוד = סך ההחלטות הלוגיות בקוד +1

2. Data-flow

בשיטה זו עוקבים אחר "גלגולים" של data בקוד.

מצבים אפשריים של data:

d- defined	(int x;)
u-used	בחישוב(x=x+2), בתנאי ((if(x>4))
k-killed	לשחרר

שיטוח חיפוש מסלולים:

du: לחפש רק נתיבים בהם יש מעבר מהגדרה לשימוש

u: לחפש נתיב שימוש בלבד

3. חיפוש שימושים בתוך חישובים בלבד או בתוך תנאים בלבד

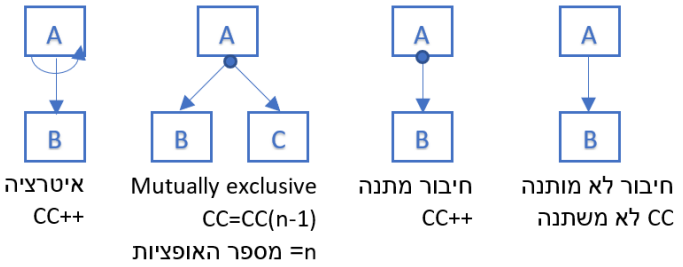
חיפוש נתיבים שיש בהם כשל לוגי כלומר u לפני d או k בהתחלה.

תמיד הכרחי וקורה תוך כדי עבודה

4. בדיקות באינטגרציה

בדיקת קישוריות בין יחידות קוד- המטרה לוודא תקשורת בין רכיבי תוכנה שונים

חיבורים בין יחידות



קביעת תוכנית עבודה

מודלים להערכת אומדנים

1. **גישה לפי משכי זמן קודמים- 3 point estimation**

ניקח 30 משימות עבר אשר קדמו למשימה

נגדיר 3 נקודות: a-משך זמן אופטימלי, b- משך זמן מקסימלי, m- משך זמן ממוצע

$E = \frac{a+4m+b}{6}$ סבירות של 68%

$sd = \frac{b-a}{6}$ כל תוספת sd לE תגדיל

את הסבירות במעט

*sd- התפלגות

2. **גישה לפי תוכן התוכנה- function point Jones**

- "מנקדים" את המערכת לפי סך הפונקציונליות שיש בה. על סמך הניקוד מחשבים את האומדנים למשל

f.p^(1.25) = סך תקלות שימצאו בפרויקט

f.p^(1.15) = מס' עמודי התיעוד הנדרשים למערכת

3. **מודל cocomo**

נסווג את הפרויקט ע"פ גודל הצוות, הטכנולוגיה, הניסיון הקודם

organic

semi-det ched לכל סוג יש טבלת מקדמים

embedded

Effort (person/month) = (2.94Xa) X KSLOC^br

- KSLOC = כמות שורות הקוד הנדרשת באלפים (תלוי בשפת התכנות).
- Effort Factor = a - גורם התאמת המאמץ (לפי סוג הפרויקט).
- Scaling Factor = b - גורם סדר גודל.

סיכונים

סיכון מוצר- תקלות במוצר

סיכון פרויקט- מצבים אשר יפריעו להמשך הפעילות (בודק יוצא במילואים, ספק לא העביר רישיון לכלי בדיקה, לקוח לא החלטי)

מטריצת סיכונים

זיהוי סיכון	מיון	הסתברות	נזק	RPN	התמודדות
סיכון אפשרי	גורם הבעיה	הסיכוי שיקרה	רמת נזק	כמה קריטי	מה נעשה

ignore- לא להתמודד עם הבעיה למצוא חלופה או לדחות לעתיד

mitigation- להתמודד עם הבעיה עכשיו

contingency- "ריכוך" השתמשות בתחליף לא פותר את הבעיה אבל מקטין את הנזק

transfer- העברת פתירת הבעיה לגורם חיצוני

סיכום פרויקט בדיקות

לאחר קבלת קריטריון יציאה שהוגדר בשלב התכנון.

דיווח בדיקות software test report (STR)

- עדכון על התנהלות התהליך: מה בוצע, מה תוכנן, הסבר לשינויים אם יש
- עדכון על מצב הגרסה בעת סיום הבדיקות, סטטיסטיקות על תקלות והרצות מקרים
- עדכון מסמכי בדיקות, ארכיב (ניהול גרסאות) וסביבות (למשל בחירת רגרסיה לבדיקה הבאה)
- הדרכה והשמעה ל"באים בתור"- הארגון (הצוות) שיעבוד בגרסה מעכשיו (לפי רמת הבדיקות שהסתיימה)
- סקר, דנים בסיכוני עבר, ניתוחים של סטטיסטיקות עבר

מודלים לשיפור תהליכי הבדיקות
- מודלים מרשמיים: שלבים מוגדרים לפי סדר מסוים
- מודלים לא מרשמיים: מודל גמיש שלא מחייב סדר אלה רק להגיע למטרה

Agile

הדרישות אינן מוגדרות בשלמות מראש אלא מתפתחות תוך כדי עבודה. מסירה חוזרת ללקוח.
הרבה יותר מסירות, התקנות וצורך ברגרסיה. יש צורך בניהול מדויק של גרסאות

נעדיף

1. אנשים ויחס גומלין על פני תהליכים וכלים ← צוותים מוגבלים בגודלם
2. תוכנה עובדת על פני תיעוד מפורט ← עדיף להוציא פחות יכולות בתוכנה ובפרט שהכל עובד
3. שת"פ עם לקוח על פני משא ומתן חוזר← הלקוח (נציגו) הינו חלק מצוות הפיתוח ממסירה למסירה יש עדכון ומשוב מהלקוח
4. תגובה לשינויי על פני מעקב אחרי התוכנית ← יישום הפיתוח צריך להיות כזה שמקבל שינויים בקלות

הבדלים עיקריים בין פרוייקט מסורתי לAgile

שלב	Agile	מסורתי
תכנון	תכנון מאקרו לכל גרסה ותכנון מיקרו לכל מסירה	תכנון אחד בתחילת הפרויקט
תיעוד	נמוך יותר	
רמות בדיקה	ערבוב רמות	לא מתחילים רמת בדיקה טרם הסתיימה הקודמת
קצב עבודה	קבוע	משתנה
עקיבות	הדרישות משתנות – אין טעם לתחזק מנגנון השוואה בין בדיקות ודרישות	דרישות קבועות- עקיבות גבוה
דרישה ובדיקה	מגדירים בדיקות לדרישה טרם החל הפיתוח עצמו	הבדיקה מתבצעת לאחר הפיתוח
עצמאות	נמוך יותר	

מתודולוגיות בAgile

scrum בעלי תפקידים עובדים בספרינטים

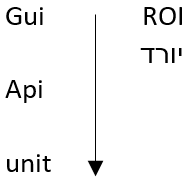
1. עבודה בתקופה תחומה בזמן קצר יחסית בסוף תקופה זו יש להוציא ללקוח תוכנה עובדת.
לכל ספרינט מגדירים את הדרישות, ההגדרות והבדיקות שהפיצ'ר חייב לעבור טרם מסירה ללקוח (user story)
2. גישה tdd- test driven design נממש בדיקות (לרוב אוטומטיות) טרם הקוד פותח
3. משוב רציף- תשתית אשר קולטת קוד בכל עת ומסוגלת לייצר גרסה עובדת מכל חלקי הקוד שנמזגים אליה

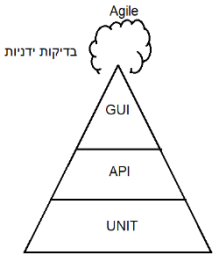
מושגים
Product owner- איש קשר מול הלקוח עוסק בהגדרה ועדכון של דרישות מספרינט לספרינט
scrum master- מאמן, מדריך שתפקידו לשמור על פעילות לפי הscrum
team- גישת אחריות של כל הצוות
product backlog- רשימת הפיצ'רים שתגיע ללקוח. מוגדר ע"י הproduct owner
sprint backlog- רשימת הפיצ'רים שיכלול הספרינט הנוכחי. מוגדר ע"י הצוות עם דגש על מוצר עובד
poker planning- הצגת הuser story לצוות כל אחד כותב לעצמו את הערכת זמן\מאמץ הנדרש לאחר חשיפה כל ההערכות מקבלים החלטה משותפת

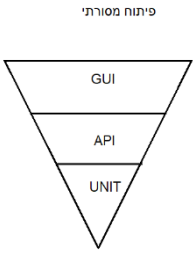
בדיקות בAgile
לרוב לא נכתוב מקרי בדיקה אלה רבעי בדיקה (test quadrants)

פירמידת הבדיקות (אוטומטיות)

ROI- החזר על ההשקעה



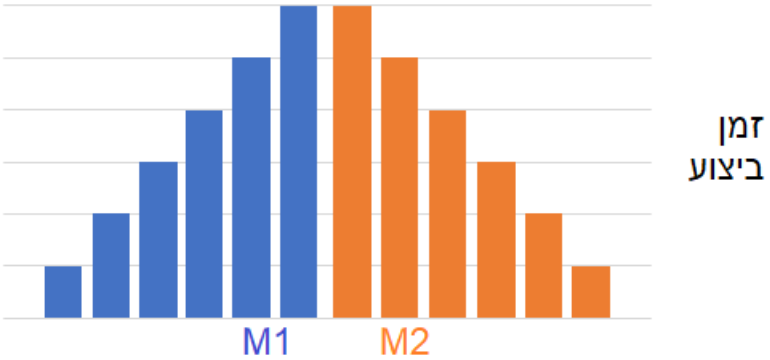




שיבוץ משאבים לפרויקט

אלגוריתם ג'ונסון

סידור על הפעולות כך שההתחלה והסוף יכללו את הפעולות הדורשות את הזמן הקצר ביותר לביצוע למשל צריך לבצע את פעולה M1 ופעולה M2 כך שM1 קודמת נסדר את ביצוע הפעולות של M1 כך שנתחיל מהקצרה ביותר ונעלה לארוכה ביותר ואת M2 נתחיל מהארוכה ביותר ונרד לקצרה ביותר



אלגוריתם ג'קסון

כאשר יש משימות מM1 הדורשות M2 ומשימות מM2 הדורשות M1

- נחלק את המשימות כך- קודם M1, קודם M2, M1 לבד,M2 לבד
- בתוך כל חלוקה נבצע ג'ונסון
- השיבוץ יבוצע כך

משימות בהן הפעולה ראשונה← משימות בהן המשימה לא תלויה במשימה אחרת← משימות בהן המשימה שנייה

-