מתודולוגיית פיתוח פרויקט - 2019

1.כללי:

פיתוח המערכת מורכב מ-4 גרסאות וגרסה מקדימה, כאשר הגרסה המקדימה (גרסה 0) אינה כוללת פיתוח ומטרתה למדל את שכבת ה-domain של המערכת. גישת הפיתוח מאופיינת בסבבי פיתוח קצרים (Agile development), והיא במסגרת Model Based Software Engineering (MBSE).

הגרסאות מבוססות על מסמך הדרישות הכללי. מסמך זה מהווה את האפיון הראשוני של המערכת ומתמצת את מהות המערכת ותפקידו לסייע בקבלת החלטות בשלב הפיתוח. עם זאת, הגרסאות מרחיבות מסמך זה ואף עשויות לשנות חלקים ממנו, באופן שאינו סותר את התיאור המהותי של המערכת. השינויים נגרמים בשל טעויות באפיון הראשוני למול רצון הלקוח או שינויים אותם הלקוח דורש במהלך התהליך. כל גרסה בנויה על קודמותיה ומורכבת מתוכן חדש לפיתוח ושינויים לדרישות עבר.

2. עקרונות הפיתוח:

1. ניהול פרויקט וחלוקת תפקידים בצוותים.
2. תכנון, ניהול ותחזוקת מודלים עדכניים.
3. ניהול קוד מקור.
4. ניהול וכתיבת בדיקות
5. מעקב אחר דרישות (traceability).

2.1 מבנה הצוות ותפקידיו:

בפיתוח כל גרסה, חברי הצוות יקבלו תפקיד של מנהל צוות, מפתחים ובודק מטעם הלקוח. התפקידים השונים מתחלפים בין חברי הצוות בכל גרסה. בגרסה הראשונה תפקיד הבודק מטעם הלקוח יהיה ייחודי ובגרסאות העוקבות, הוא יעשה על ידי מנהל הגרסה.

2.1.1 תחומי אחריותו של מנהל הצוות בגרסה:

- הגדרת משימות, תעדופן, תזמונן, וחלוקתן בין המפתחים..
- מעקב אחרי התקדמות הגרסה ועדכון חלוקת המשימות והזמנים בהתאם להתקדמות בפועל.
- בגרסה 1: בניית ארכיטקטורת בדיקות (שנכתבות על ידי המפתחים), ובניית מנגנון ל-regression testing.
- מיפוי (מעקב) בין הדרישות השונות לקוד המתאים ולבדיקות.
- לוודא שכל תרחיש משתמש מכוסה ע"י בדיקות.
- תכנון שינויים במודלים עבור הגרסה, ובדיקת התאמת המודלים למימוש.
- ניהול קוד המקור וביצוע code review לחברי הצוות כמפורט בהמשך..

על הצוות לנהל את הפרויקט באמצעות **כלי יעודי לניהול פרויקט,** שיבחר ויוצג בפגישה הראשונה עם מנחה הקבוצה.

2.1.2 תחומי אחריותו של מפתח:

- ביצוע המשימות לפי הגדרת מנהל הצוות: פיתוח ועדכון מודלים וקוד.
- כתיבה והרצה של בדיקות יחידה (unit tests) לקוד אותו המפתח כתב.
- כתיבה והרצה של בדיקות אינטגרציה (integration tests) בין רכיבים שונים בהם היה שותף פעיל לתהליך הקידוד.

2.1.3 תחומי בודק התוכנה מצד הלקוח:

- בגרסה 1: בניית ארכיטקטורת בדיקות לקוח המתממשקת עם המערכת' ובניית מנגנון ל-regression testing.
- כתיבת בדיקות קבלה לדרישות המפורטות בגרסה.
- הובלת בניית ארכיטקטורת בדיקות היחידה והשילוב בשיתוף חברי הצוות, כמפורט בסעיף הבדיקות.

בגרסאות 2-4, מנהל הצוות ישמש גם כבודק תוכנה מטעם הלקוח.

בנוסף, בכל מפגש יקבל חבר צוות אחר משימה של הצגת נושא לשאר חברי הצוות (ראו נספח א' להנחיות מפורטות).

2.2 תהליך עבודה באמצעות מערכת source control:

במהלך הפרויקט צפוי להיכתב קוד רב, וחייבים לאפשר עבודה בו-זמנית לחברי צוות. מסיבה זו, יש להשתמש בתוכנה לניהול קוד מקור המבוססת על Git. להלן מתואר תהליך המקובל בחברות תוכנה רבות משום שהוא מסייע לשמור על עקרונות הפיתוח (למשל, קוד קריא) ולמזער תקלות הנגרמות עקב ניהול קוד לקוי.

1. למנהל הצוות יש תפקיד של "עורך ראשי". הוא היחיד שיכול לבצע commit ל-master branch.
2. כל חברי הצוות עובדים על feature branch (המוכרים גם כ-topic branch). אלה הם ענפים קצרי טווח המשמשים לפיתוח feature בודד, לתקן באג בודד או למשימות ספציפיות אחרות.
3. כאשר העבודה על feature branch מסתיימת, המפתח יבצע pull request ל-master branch.
4. מנהל הצוות בוחן את הבקשה ונדרש לקבל החלטה:
   - אישור הבקשה תחת התנאים הבאים:
     a. הוא מאמין כי הקוד נכתב היטב (The Art of Code Review), למשל, מבלי לדבר עם המפתח הוא יכול להבין את מטרת המחלקות והשיטות החדשות / המתוקנות (בהתבסס על מוסכמות למתן שמות ותיעוד).
     b. הוא סבור כי יש מספיק בדיקות בכדי לוודא את תקינות הקוד.
   - דחיית הבקשה ומתן נימוקים והסברים על השינויים הנדרשים בקוד.
5. אם הבקשה מאושרת, מנהל הפרויקט ממזג אותה ל-master branch.
6. המפתחים צריכים למשוך את ה-master branch האחרון בכדי לעבוד על קוד עדכני.

2.3 בדיקות (Tests):

בדיקת קוד היא גורם מרכזי בפיתוח תוכנה. על אף שעבודת כתיבת הבדיקות אינה נתפסת בפני רבים כמעניינת כמו שלב הקידוד, היא חלק בלתי נפרד מכתיבת מערכת ומהווה אסמכתא לאיכותה. מערכת ללא רכיב בדיקות פועל, נגיש וגמיש היא חסרת ערך.
אחריותה המשותפת של הקבוצה בהובלת בודק התוכנה מצד הלקוח **לבנות תשתית לכתיבת בדיקות** על מנת לאפשר יציבות, נגישות וגמישות. ***יציבות*** ניתנת למדידה באמצעות regression testing (מנגנון המאפשר הרצת כלל הטסטים לאחר שינויים בקוד). ***גמישות*** משמעותה הוספה/הסרה פשוטה של בדיקות הנדרשת בעקבות הוספה ושינוי של דרישות.
במערכת גדולה, איכות הבדיקות תלויה רבות בתשתית. על הקבוצה להיות מסוגלת לנמק החלטות הנוגעות לתשתית כתיבת הבדיקות (חלוקה ל-test suits, כתיבת הבדיקות בצמוד או בנפרד מהקוד וכו') ולהסביר כיצד תשתית זו מאפשרת יצירת בדיקות חדשות המבוססות על שימוש בבדיקות קיימות (מבלי לכתוב אותן מחדש).

סוגי הבדיקות: ישנן בדיקות יחידה, בדיקות שילוב ובדיקות קבלה.
כל מפתח אחראי לכתוב בדיקות יחידה לקוד אותו כתב ובדיקות שילוב בין רכיבים שונים אשר היה מעורב בכתיבתם (לפעמים נוח לפתח בשיטת TDD). בדיקות אלו יכולות להיות code driven או functionality driven. במקרה הראשון הבדיקות בודקות מרכיבים תחביריים, כמו פונקציות. במקרה השני, הבדיקות בודקות תרחישי שימוש או רמת שירות או מאפייני נכונות. בגרסה 1, מנהל הגרסה נדרש לבנות את התשתית לארכיטקטורת של מבחני ה"קופסה הלבנה", ואת המנגנון ל-regression tetsting של מבחנים אלו.
מבחני קבלה בודקים אך ורק את תרחישי השימוש. הם צריכים להיות קריאים ומובנים ללקוח, עליהם לשקף את הדרישות השונות, ולהיות מנותקים מיישום המערכת. מבחני לקוח נפרדים מהקוד, ומשתמשים במערכת כ-"קופסה שחורה". בגרסה 1, בודק התוכנה מצד הלקוח נדרש לבנות את התשתית לארכיטקטורה המאפשרת מאפיינים אלו, ואת המנגנון ל-regression tetsting של מבחני לקוח. לאחר גרסה זו, מנהל הצוות ישמש גם כבודק מטעם הלקוח.
בדיקות שלילה: כל סוגי הבדיקות צריכים לכלול גם תרחישים אסורים האמורים להיכשל.

2.4 מעקב אחר דרישות (traceability):

במערכות גדולות המאופיינות בקוד רב, קבוצות פיתוח רבות (עם תחלופת מפתחים) ודרישות משתנות, נדרש לשמור על מיפוי בין חלקי **הקוד** השונים ל**דרישות** ול**בדיקות** לצורך תחזוקה. המעקב מספק את דרישת ה***נגישות*** עבור בדיקות. המעקב צריך לכלול:

- עבור כל דרישה, אילו חלקים בקוד מממשים אותה, ומהם מבחני הקבלה, היחידה והאינטגרציה הבודקים אותה.
- כל בדיקה צריכה לכלול קישור לקוד אותו היא בודקת ואם רלוונטי גם לתרחיש אותו היא בודקת.
- הארכיטקטורה צריכה לאפשר מיפוי ברור בין קוד לבדיקות שלו ולדרישות אותן הוא מממש.

2.5 מודלים של תוכנה

מודלים של תוכנה מספקים הפשטה של התוכנה מנקודות מבט שונות.

- מודלים אינם כוללים פירטי מימוש שאינם רלוונטיים לרמת ההפשטה שלהם.
- בכל גרסה, המודלים חייבים להיות תואמים לפיתוח הפרוייקט (זו סיבה נוספת להימנעות מפרטי מימוש משניים במודלים).
- העדכניות של use cases צריכה להשתקף בקיומם בשכבת השירות של ה-domain component.
- העדכניות של מודל מחלקות מתבטאת בהתאמת המחלקות והקישורים לקוד, ובאכיפת האילוצים שהמודל מספק. רצוי לאכוף אילוצים כחלק מתהליך הבנייה של אובייקטים: על ידי בנאים של מחלקות (למשל אילוצי ריבוי, או אילוצי מעגליות של קישורים), או בהצהרה על טיפוסי שדות או שיטות (למשל אילוצי overriding).
- העדכניות של תרשים מצבים (statechart) צריכה להשתקף בקריאות לשיטות של מחלקות.

3. תוכן גרסה (מצטבר):

התוכן של כל גרסה מבוסס על קודמותיה. המודלים צריכים להיות מעודכנים למול הדרישות ותואמים לקוד.

תוכן גרסה כולל את הרכיבים הבאים:

1. מילון מונחים.
2. תרחישי שימוש עם קישורים לבדיקות ולקוד.
3. דרישות service level עם קישורים לבדיקות ולקוד.
4. מודל ארכיטקטורה.
5. מודל מחלקות ומודלים נוספים לפי הצורך.
6. בדיקות עם קישורים לקוד ולתרחישים.
7. קוד.

3.1 מילון מונחים (Glossary):

רשימה אלפביתית של מונחים המורכבת ממונחים המשמשים את המערכת או מהווים חלק מתיאורה. במילון המונחים יש להגדיר מונחים ייחודיים לתחום הפרויקט ומונחים כלליים שאינם נפוצים. מילון המונחים יוצר שפה משותפת בין הלקוח, מנתחי הדרישות, ארכיטקט המערכת וצוות הפיתוח. מילון המונחים יכול להשתנות תוך כדי הפיתוח.

3.2 תרחישי שימוש (Use Cases):

תרחישי השימוש מגדירים את האינטראקציות בין המערכת לבין הסביבה החיצונית. כל תרחיש שימוש מתאר דרישה פונקציונלית של המערכת ומדגימה את האינטראקציה בין שחקן (משתמש או מערכת חיצונית) לבין המערכת.

מבנה התיאור של תרחיש שימוש:

- תנאי קדם ותנאי בטר (post conditions).

- תהליך: תיאור מילולי מובנה (אם קצר) או activity diagram או system sequence diagram בין השחקנים למערכת. ה-sequence diagram שימושי במיוחד עבור יצירת מבחני קבלה.

תרחיש שימוש שאיננו כולל (כמעט) אינטראקציות ואין לו תנאי קדם/סיום "מעניינים" מספיק לתאר בקיצור באופן מילולי.

3.3 מודל ארכיטקטורה של המערכת:

ארכיטקטורה של רכיבים לוגיים ורכיבים פיזיים של המערכת. על הארכיטקטורה לשקף את התלויות בין הרכיבים והשכבות השונות במערכת. יש לתאר את הארכיטקטורה באמצעות דיאגרמת רכיבים (component diagram).

3.4 מודל מחלקות (Class Model):

מודל המחלקות משמש לתיאור מבני של מערכת ע"י הצגת המחלקות השונות במערכת, האילוצים והקשרים שביניהן, תוך שימוש בתיאור חזותי של אילוצים ב-UML. בנוסף, יש לציין אילוצים שמעבר לכוח הביטוי של דיאגרמת מחלקות. אפשר לכתוב אילוצים כאלו בשפה טבעית או ב-OCL. יש לשים לב כי תרשים המחלקות צריך לשקף את היחסים המתוארים בתרחישי השימוש השונים.
מודל המחלקות העיקרי צריך להיבנות בגישה של "דיאגרמה לבנה": רק מחלקות וקישורים ואילוצים ביניהם. פירוט נוסף של מודל זה מוסיף שדות, שיטות עיקריות, היררכיה של מחלקות, ואילוצים נגררים.

- על התרשים המפורט להציג **רק** שיטות עיקריות של מחלקות
- להשתמש בחלוקה לחבילות (packages) שונות.
- שיטות מימוש (כגון getters/setters או חיפוש באוספים) או מחלקות מימוש כגון factory אינן מופיעות בתרשים.

נספח א':

עקרונות להצגת נושא בכיתה - 10 דקות:

- הצגת נושא אינה מחייבת מצגת PowerPoint או Beamer. עם זאת, נדרש להציג את הנושא באופן שיהיה ברור ותמציתי.
- מומלץ להתמקד במספר נקודות מפתח ולדון בהם תוך שימוש בדוגמאות.
- יש להתמקד בהיבטים הפרקטיים של הנושא בהקשר של הפרויקט. דוגמאות שימוש מעשיות יתרמו לקבוצה להבין מדוע הנושא חשוב וכיצד הוא בא לידי ביטוי בפיתוח המערכת.
- מומלץ להתאמן על ההצגה טרם המפגש בדגש על עמידה בזמנים.
- אם תבחרו להשתמש במצגת PowerPoint, מומלץ להיעזר בעקרונות הבאים.