קוד פתוח - סיכום

הערות:

- להצעות, טענות, תיקונים ובקשות, בבקשה לשלוח במסמך הבא: ללחוץ כאן.
- בשיעור 8 ו-9 הגעתי באיחור, אם למישהו יש מושג מה הוא אמר שם, אשמח אם יכתוב בקישור מעלה.
- זה לא באמת סיכום, זה יותר תמלול, אז מצטער אם הוא לא נשמע הגיוני בחלקים מסוימים.
- אני לא פירטתי פה ממש על הפקודות של לינוקס ו-git כי מה שהוא לימד הכרתי קודם. אם יש מישהו שלא מרגיש שליטה בהם, ממליץ לעבור על זה בלי קשר למה שכתבתי.
- היה את ה-design patterns והרישיונות השונים של הקוד הפתוח. לא פירטתי על זה..

שיעור 1 - מה זה קוד פתוח

מה זה קוד פתוח?

לינוקס זה קוד פתוח. מה שהוביל את המהפכה של הקוד פתוח היה הלינוקס.

שנה שעברה חשבו שניתן לעבוד בלינוקס, אבל הוא החליט שהשנה לא כי אנשים לא רגילים לזה, אבל הוא מאוד ישמח אם נעשה זאת על לינוקס.

נלמד את ההתאמות ולמה הם חשובות.

נושאים שנעבור עליהם בקורס

- להשתמש בקוד פתוח - הבסיס לכל העולם הזה של הקוד הפתוח הוא קודם כל השימוש בו. קוד שלא משתמשים בו הוא קוד מת.
- הפיתוח של הקוד השיתופי - נראה איך עובדים ביחד. לא כולנו יודעים לעבוד ביחד, ולא כל המתכנתים האחרים אוהבים לעשות את זה.
- פורק (fork) - מה זה פורק של קוד פתוח. ולמה תחרות זה דבר טוב.
- נדבר על מודלים עסקיים - איך זה יכול לעבוד? איך מתפרנסים בקוד פתוח ולא רק איך תורמים
- רשיונות - קוד פתוח היא קודם כל קוד תוכנה אשר מונגש באמצעות רישיונות. למרות שאנחנו לא עורכי דין אנחנו צריכים להבין אותם לפני שנוכל להשתמש בהם.
- מה קורה בממשלות? למה זה חשוב? איך זה משתלב עם ערכים של שיתופיות ושקיפות? קוד פתוח זה קודם כל שקיפות. ועל כן, האם מדינה צריכה לשחרר את הקוד שלה לציבור כחלק מעקרון השקיפות? מהם הסכנות והיתרונות של השקיפות? מצד אחד השקיפות מסוכנת ומאפשרת גישה לאנשים שלא כדאי שיהיה בידיהם גישה לקוד. מצד שני האזרחים יודעים על מה הם משלמים.
- קהילות - קהילות מפתחים שמקורן בקהילות קוד פתוח. חוקרים רואים זאת כמקור ו-קטר לחדשנות.
- שיתוף מידע - סטנדרטים של שיתוף מידע. כבר 30 שנה מפתחים את הלינוקס, איך לפני 30 שנה ידעו במה צריך לתמוך? צריך לעבוד בצורה כזו שאנחנו נאפשר ונחזה אילו שירותים שעדיין לא נוצרו יוכלו להתחבר אלינו. או למשל אם אנחנו בעצמנו נרצה להוסיף פיצ'רים לקוד ישן שכתבנו.
- מחקרים מתקדמים - איפה המקום של הקוד פתוח בתחום הזה? למשל, לא קיימים פרוטוקולים של בלוקצ'ין שאינם בקוד פתוח. אוניברסיטאות רוצות בעלות על הקוד שהסטודנטים שלהם יוצרים, ועל כן יש

התנגשות בין האוניברסיטאות לבין החוקרים בה על הזכויות על הקוד. כיום כשמפרסמים מחקרים נדרש לפרסם גם את הקוד.

- סכנות - צריך להכיר את הסכנות בשימוש בקוד הפתוח. אם לפני כמה שנים יכלנו לסמוך על הקהילה כדי לשמור עלינו מוירוסים ודברים כאלו. בגלל הנפח של הקוד הפתוח שיש כיום, יותר ויותר אנחנו סומכים על תוכנות שמבצעים סריקות שבודקות אם הקוד שאנחנו לוקחים זה הקוד שאנחנו מקבלים. היו לא מעט מקרים של אבטחת תוכנה בשנים האחרונות שאנחנו נסקור.

הערכה: לדעתו, פטנטים מאטים את הקדמה הטכנולוגית. לדעתו זה תחום מיותר. וללא הקוד פתוח לא היה לנו לינוקס או SQL, והיינו צריכים לשלם לבעלי אינטרסים.

אג'ייל - כל אדם יכול לבקש שיעשו משהו, לא בהכרח משהו שבאמת צריך. וכל אחד יכול להגיע ולממש את זה, ויכול להיות שזה יתקבל. אבל זה לא בהכרח יהיה מיידי. ויכול להיות שעוד אנשים יבדקו זאת. (הערה: אג'ייל גם קיים בעולם העסקי בצורה דומה, לא רק בקוד פתוח).

שיעור שני - לינוקס ותוכנות ניהול קוד

- **ריצ'רד סטולמן** - "הדרך היחידה לדעת מה באמת עושה התוכנה שאנחנו מריצים על המחשב, זה לקרוא את הקוד שלה" (הזכויות שלנו כמשתמשים). פעיל קוד פתוח ידוע, הקים ב-1983 את פרוייקט ה-GNU.
- GPLR - כניראה התכוונתי ל-GPL, שזה רישיונות General Public License.
- אין חינם - אם המוצר בחינם, אנחנו המוצר.
- פקודות לינוקס (הפעלנו מכונה וירטואלית על מחשבי האקווריום באמצעות rundeb9, זה מריץ הפצת דאביאן - סיסמאת הרוט היא toor).
  - טאב - משלים אוטומטית, מתחילים להקליד, לוחצים על tab, ואם הוא מוצא משהו הוא ישלים, ואם יש יותר אפשרויות הוא יראה רשימה
  - סימון ה-| - pipe - לוקח את הפלט מצד שמאל, וזורק אותו כקלט לצד ימין.
  - ls - בודק את הקבצים בתיקיה
  - touch - יוצרת קובץ
  - vim / emacs / nano - תוכנות לכתיבת טקסט ללא קוד. במכונה שהפעלנו בשיעור אין vim, מותקן אבל גרסה עתיקה שלו - vi.
  - more - נותן לקרוא קובץ טקסט עם buffering, כלומר, לא מדפיס למסך הכל ומחכה שתלחץ על enter כדי להמשיך לקרוא. אבל לא נראה שיש אותו על כל הפצת לינוקס.
  - less - כמו more, רק דרך pipe.
  - tail / head - שימושי ללוגים, מציג את ה-n שורות האחרונות \ ראשונות
  - cut -
  - grep -
  - find -
  - man - המדריך של הפקודות, כותבים man ולאחר מכן את שם הפקודה כדי לקבל הסבר מלא על השימוש בפקודה.
  - rm - מחיקת קובץ
  - mv - הזזת קובץ או שינוי השם
  - mkdir - יצירת תיקיה

- cp - העתקת קובץ
- ps - רשימת התהליכים
- kill - שולח סיגנל לתהליך עם המספר שלו (-9 הורג אותו ללא יכולת להתנגד, אם אנחנו לא הבעלים של התהליך צריך root לביצוע).
- exit - יציאה
- apt-get - הפקודה כדי להתקין או להסיר תוכנות בהפצות מבוצצות dabian.
- su - שינוי משתמש
- sudo - הרצת פקודה בהרשאות מנהל.
- whereis - מחזיר את המיקום של הפקודה

- שימוש בממשק משתמש גרפי (GUI) עלול לפגום באבטחת מידע. אנשים שמשתמשים בשרתים לרוב יעבדו דרך הטרמינל מהסיבה הזו.
- לפני שהיה git היו משתמשים בתוכנות לניהול קוד בו היה שרת מרכזי, שכל אחד מקבל את הקבצים אליו, ולאחר השינוי, הוא שולח בחזרה לשרת, אבל אם מישהו שלח עדכון, צריך לתקן את ההתנגשות לפני השליחה החוזרת.
- לינוס הגה את git - הרפוסיטורי הוא מבוזר, כולם מחזיקים את הקוד כאילו הוא הבעלים של הרפוסיטורי. הכנסת ה-git האיצה את קצב הפיתוח של קוד. מאפשר לעבוד על מספר גרסאות במקביל, למשל גרסת פיתוח \ אינטגרציה \ פרודקשן, ניתן לנהל את כל הגרסאות הללו מעל גיט אחד.
  - הגיט שומר את השינויים, לא את הקבצים שוב ושוב ושוב. ובגלל זה זה יותר יעיל. בתיקיה הראשית הוא שומר סוג של אינדקס אשר מכיל את כל המידע על הקבצים והשינויים.
  - init - בתיקיה הראשית של הפרויקט משתמשים בפקודה הזו כדי לאתחל בתיקיה ריפוסיטורי.
  - status - מציג את מצב הקבצים ברפוסיטורי.
  - add - מוסיף את הקובץ למעקב (אפשר להשתמש ב* או . כדי להוסיף את כל מה שיש בתיקיה, אפשר גם להשתמש ב--all אבל הוא לא הזכיר זאת). הערה חשובה שכדאי לזכור, אנחנו מוסיפים עם add שינויים, לא קבצים, כלומר, גם אם מחקת קובץ, צריך לעשות לו add, כי אחרת ה-git לא ידע שמחקת את הקובץ.
  - pull - מבקש את כל השינויים לרפוסיטורי המקומי. אם אף אחד לא נגע בקובץ אז לא נצטרך לעשות כלום, והוא יעשה merge לוקאלי בתחנת עבודה.
  - push - לדחוף את השינויים המקומיים אל האחרים. חשוב לציין שלא אמורה להיות בעיה בשלב הזה, כי הרגע עשית pull, ותיקנת את ההתנגשויות, אם היו שינויים, ה-push יכשל מקומית.
  - commit - לקחת את השינויים בתיקיה ולשמור את המצב שלהם.
  - clone - לשבט (להוריד) ריפוסיטורי שנמצא במקום אחר.
- אנחנו בקורס נשתמש ב-Github של האוניברסיטה. ב-Github לא עושים בדיוק push, מבקשים pull request במקום.

שיעורי בית: לפתוח ריפוסיטורי עם 3 קבצים ולכתוב באנגלית:

- אפשרויות לפרוייקטים שאנחנו רוצים לעשות, בכל קובץ בקצרה 2-3 שורות מה הפרוייקט שאנחנו רוצים לעשות ומה המטרה.
- לשלוח מייל עם השם שלנו, החשבון גיטאב שלנו milloa

שיעור 3 - רישיונות

בהתחלה היו מכנים את הקוד כ-Free code, כלומר קוד חופשי. אך אנשים חשבו על זה בתור קוד שניתן להשתמש בו בחינם, ולא דווקא קוד שאפשר להשתמש בו באופן חופשי. מאוחר יותר טבעו את השם קוד פתוח.

בהתחלה היה גוף מרוכז שניפק רישיונות של קוד פתוח. משום שמדובר בקהילה, ובקהילות יש סכסוכים ומלחמות, כיום יש המון רישינות שונים של גופים שונים שמנפקים אותם. חלקם נותנים זכויות מלאות על הקוד, וחלקם מגבילים את הזכויות.

רוב הנושאים שהרישיונות הללו מגדירים הם זכויות קניין, זכויות הפצה וכו'.

כמעט לכל פרוייקט ב-Github יש אייקון של רישיון. אם אין אחד כזה, כנראה שיש לו משהו פחות סטנדרטי. בכל פרוייקט כמעט יופיע הרישיון שימוש.

כל אחד יכול להגדיר לעצמו רישיון, זה פשוט דורש לשלם לעורך דין. למשל פייסבוק יצרה רישיון משלהם, אך בגלל לחץ מהקהילה הם חזרו בהם.

החלפת הרישיון היא דבר בעייתי, משום שהתורמים תרמו קוד בידיעה שהקוד נמצא ברישיון מסוים. כדי להחליף רישיון צריך אישור של כל התורמים. יש כל מיני טריקים שחברות לפעמים עושות בשביל להתמודד עם זה.

Copyleft - ברישיון בתוכנה מסוג זה, מי שמשתמש בתוכנה מחויב למסור לכל מי שהוא מנגיש (או מפיץ) אליו את התוכנה את קוד המקור של מה שהוא מריץ.

עבודה נגזרת - למשל לקחתי את ElasticSearch ומהקוד שלי קראתי ל-ElasticSearch, האם זה מחובר? זה סלע מחלוקת האם גם הקוד שקרא ל-ElasticSearch הוא עבודה נגזרת? יש חברות רבות שנמנעות מלהשתמש ב-GPL בגלל זה, משום שמבחינתם זה "מדבק", כלומר, הקוד שלהם הופך גם ל-GPL.

רישיונות משתנים עם הזמן, GPL2 למשל נוצר כאשר קוד היה מופץ על דיסקים, והקוד לא הורד מהאינטרנט.

פרויקטים ללא רישיון הם הכי מסוכנים, כי יכולים להוסיף להם רישיון מאוחר יותר, גם אם נסכים לו וגם אם לא.

רישיונות

- MIT
- GPL
- BSD
- אפאצ'י

בעיות נוספות עם רישיונות, שהם לא תמיד תואמים אחד את השני. למשל אפאצ'י אומר באופן מפורש שהוא לא תומך ב-GPL ולא מוכן שיחברו עם GPL (הערה, כנראה מדובר באפאצ'י 2.0 מול GPLv2, בגרסא GPL3 שגם ידועה בתור הגרסה שאף אחד לא משתמש בה אין כבר את הבעיה הזו מסתבר..).

קיימים גם רישיונות תרומה לפרוייקטים, Collaborator License Agreement או בקיצור CLA, שצריך להצהיר שאם אנחנו תורמים קוד לפרוייקט מסויים, הממונים עלינו יודעים שאנחנו תורמים.

עם הרישיונות הללו יש כל מיני בעיות: מה אם עזבתי את החברה, ועכשיו אני בחברה אחרת, ושם הם לא יודעים,אבל כבר חתמתי על ה-CLA.

DCO רישיון שאני חותם עליו כל commit.

מה לעשות בבית?
כל קבוצה שהחליטה שהיא קבוצה צריכה לקחת את השבוע הקרוב בשביל לבדוק את התשתיות קוד פתוח שיכולות לשמש אותה. ולראות היכן הדברים נמצאים והאם קיימות תשתיות קוד פתוח כאלו והאם אנחנו צריכים לממש אותן ואז זה יכול להיות קצת קשה.

שיעור שלישי
Fork - כמו clone, העתק של הפרוייקט והתחלת פרוייקט אחר. כאשר אנחנו רוצים להוסיף פיצ'ר נוסף, הבעלים של הפרוייקט יצטרך לאשר את השינויים הללו. כאשר מדובר ב-fork אנחנו יכולים לעשות מה שאנחנו רוצים, בלי להפריע למיינטרים של הפרוייקט המקורי. בדרך כלל חברות גדולות עושות זאת. הבעיה שבדרך כלל יש הרבה שינויים שרצים, ובדרך כלל רוצים לשמור על סנכרון עם הפרוייקט המקורי, אך לפעמים השינויים של הפרוייקט המקורי יתנגשו בפרוייקט ה-fork.

BDFL או **Benevolent dictator for life** - פרוייקטים שהתחילו דרך בן אדם אחד, וכאשר הוא התחיל לגדול מישהו היה צריך להחליט מה ייכנס לפרוייקט ומה לא. אותם אנשים הם בעצם דיקטטורים של קוד פתוח ומחליטים עבור כולם מה צריך לעשות. זה גורם לויכוחים ופיצולים בפרוייקטים.

תסריטי Fork אפשריים:
1. שניהם מצליחים - לדוגמא, MySQL ו-MariaSQL. כאשר אורקל קנו את MySQL, הם פחדו שאורקל קנו את המוצר כדי להרוג אותו. אז היוצר של MySQL (שעל שמה של בתו הגדולה נקרא הפרויקט) עשה fork לפרוייקט, ופתח את MariaSQL על שמה של בתו השנייה. שני הפרויקטים מצליחים.
2. המקור מת וה-fork חי - כאשר קבוצה גדולה נוטשת את המקור, ועוברת למתחרים. לדוגמא open office ו-libra office. הפיצול קרה כאשר open office נרכשה על ידי אורקל.
3. המקור נשאר חזק וה-fork נכשל - למשל ביטקוין, מלא מטבעות, מלא שמות, רובם נעלמו ואף אחד לא זוכר שהיו קיימים.
4. שניהם נכשלו - בעקבות הפורק הקהילה הצטמצמה ממש והכל הלך לפח.

מאוד קל להרים ולהגיד שאני יודע, קשה מאוד לבצע ולהצליח (השוואה את זה ללהקות של קופים בטבע). אנשים זוכרים מי השפיל אותם, גורם לעבוד עם אנשים חדשים.

פיצולים מסיבות אידיאלוגיות
1. אותוריום ואותוריום קלאסי
2. מנגנון System B
3. sugarCRM ו-sweetCRM

Container - תהליך שעוטף תהליך ומדמה סביבה וירטואלית סביבה. זו לא מכונה וירטואלית. Docker זה קוד פתוח שעוטף containers. משום שזה מסובך לנהל כאלו containerים ודockerים אז קמו כל מיני קוד פתוח שמתעסקים בניהול שלהם.

שיעור 4 - דיזיין פטרנס

דיבור על Design Patterns. הוא מסביר שבפרויקטים של קוד פתוח, אם לא נעבוד בצורה נכונה של דייזיין פטרן אנשים פחות ירצו לתרום מהקוד שלהם לפרוייקט.

ביקש לקרוא על Behavioral Patterns

- Observer - נשים העתק אחד שיכול לייצר אירועים וכל מיני דברים, ונשים אובייקטים אחרים שנרשמים עליו והוא מאזין עליהן ושולח את האיוונטים האלו דרכו.

דיבר על API עבור תקשורת בין מכונות. לפני שנוגעים בקוד, צריך להגדיר את ה-API. ומלבד שיעבדו על APIים, צריך שהם יעבדו על פי סטנדרטים. משום שלמה שמישהו אחר יתאים את עצמו לסטנדרט חדש שהמצאנו, שעלול להשתנות, ואם אנחנו נשנה אותו, לאחרים הקוד יפסיק לעבוד. דוגמאות לסטנדרטים של API, למשל RESTful. גם פורמטים של פלטים צריכים להיות לפי סטנדרטים.

מנוע החיפוש של גוגל הכתיב את הפורמט בו בונים אתרים, כי יש דרך בה הוא קורא אותם. ואם הם לא קריאים, אז לא ימצאו אותם. רוב האתרים כיום בנויים על CMSים מערכות לניהול תוכן שמאפשרות לנו לבנות אתרי תוכן בצורה דינאמית, משום שהתוכן מתעדכן. אנחנו לא יכולים לדרוש ממי שכותב את התוכן לדעת HTML, ולכן זה קיים.

לפעמים תמיכה בסטנדרט עלול להיות overhead רציני, מערכות מתיישנות.

שיעור 5 - תיעוד

דיבר על תיעוד

- קבצי Readme
- Commits
- תיעוד בתוך הקוד
- תיעוד מחוץ לקוד - Bug trackers, מערכות לניהול קוד
- לוגים

כיום נהוג לתעד בעיקר את ה-input/output בראש הפרוצדורה. שפות מסוימות שולפות את התיעוד מתוך הקוד בזמן הקמפול. אבל הוא אומר שתיעוד הרבה פעמים לא רלוונטי בתוך הקוד, למשל כאשר הקוד תועד ונכתב על ידי מישהו, ואז מישהו עדכן את הקוד, אבל לא עדכן את התיעוד..

היום שנכנסים לפרויקט של קוד פתוח, לרוב אנחנו נדרש להתחיל במשימות של שיפורי דוקומנטציה לפני שאנחנו בכלל נוגעים בקוד.

התיעוד חשוב לפרוייקט - לא בשבילי בתור מפתח, אלא בשביל הפרויקט עצמו. אחרת הפרויקט לא יגיע לשום מקום.

לפרוייקט צריך להיות שפה משותפת, לא בהכרח אנשים שיושבים במקומות אחרים בעולם יבינו את השפה המקומית שאנחנו משתמשים בה. מתכנתים צריכים להכיר את הדרישות העסקיות (כמו גם מנהלי המוצר והבודקים).

הזמן שלוקח כדי להחליט האם לדלג על אתר הוא בערך 2 שניות. קובץ ה-readme משמש כדי להשאיר את הגולש ולתאר את הפרויקט כדי שמי שרוצה להתקין את התוכנה יבין למה זה טוב לו ואיך זה מביא אותו למקום שהוא יכול להתחיל להשתמש.

הוא אמר שה-readme יהיה שווה הרבה נקודות בתרגיל.

קומיטים - זה גם תיעוד. כשאנחנו עושים תוספת או תיקון אנחנו צריכים לתעד אותה. בדרך כלל כשמתחילים לעבוד, אנשים בודקים קודם מה השתנה. הדרך לבדוק זאת היא דרך הקומיטים. יש מקומות עבודה שדורשים שכל תיקון יהיה בקומיט, ובכל קומיט יהיה קישור לתוכנת ניהול באגים. הרבה פעמים מי שמתקן את הבאג זה אותו אדם שכתב את הקוד לפני שבועיים. אז צריך שהקוד יהיה מספיק ברור כדי שאנחנו נבין, וגם מפתח אחר.

NL Programing - תכנות שפה טבעית, בעתיד יהיה אפשר לתת לבינה מלאכותית כדי לתכנת בעצמה, כנראה יהיה אפשר ללמד אותה על ידי קריאת commitים של אנשים והשינויים שבוצעו שם.

קהילה (comunity) - רשתות חברתיות, meetupים, מקומות ששואלים שאלות. צריך לוודא שאנשים לא כתבו שם שטויות, כי אי אפשר לתת לכל אחד לכתוב מה שהוא רוצה.

שיעור 6 - באונטיז

חוק לינוס - מיוחס ללינוס, "בהינתן מספיק עיניים שמסתכלות על הקוד, כל הבאגים יהיו רדודים" (קלים, לא רציניים).
תפיסת העולם הזו נפוצה בקוד הפתוח, אך היא שגוייה. אנשים נוטים לא לבדוק את הקוד הפתוח שהם לוקחים.
יש הרבה נושאים כמו אבטחה שלא מספיק מעבר בעיניים, צריך להבין גם בקוד וגם באותו התחום.
שקיפות זה כוח, לא בהכרח מי שמקבל את הכוח ישתמש בו למטרות טובות.

אליאוט אלדרסון (שם בדוי) - האקר צרפתי שהצליח להתחבר לתוכנת מסרים מאובטחת שפיתחה ממשלת צרפת לצרכים פנימיים המבוססת על קוד פתוח (ווטסאפ).

Bounties - דרך החיסון של פרויקטים בקוד פתוח, מציעה פרסים להאקרים תמורת מציאת פרצות אבטחה בפרויקטים. למשל הוא הזכיר את האתר HackerOne אשר מציג תוכניות Bounty. כאשר מפרסמים Bounties בדרך כלל כבר מוצאים פרצות תוך כמה שעות.
יש חברות ישראליות שמפתחות תוכנות שמוצאות פרצות בצורה אוטומטית.

קוד פתוח הוביל לגישה שתוכנות כל הזמן משתנות. בעבר היו מפיצים את התוכנה על דיסק בתצורה הסופית.

שיעור 7 - תאימות וסביבות

אחת הנקודות הבולטות בקוד פתוח היא התאמה לסביבות. גם אם אנחנו לא יודעים לפתח לכל סביבה, כדאי להשאיר מקום בשביל שאם מתישהו מישהו יוסיף תמיכה בפלטפורמות נוספות..

JAVA היתה הראשונה שאפשרה להריץ את הקוד על פלטפורמות אחרות. יש לה את הסביבה הוירטואלית עליה הקוד רץ, והסביבה הוירטואלית הזו יכולה לרוץ ולעבוד על פלטפורמות רבות.

בשלב מסוים הגיעו כלים שיודעים לעבוד עם ריבוי פלטפורמות, גדלי מסכים שונים וכו'.

JS רץ על כל דפדפן. שפה שכולם שונאים בגלל שהיא מכוערת ולא נוחה, הפכה להיות השפה הכי פופולארית כי היא רצה על כל דבר. לרוב כיום אנשים לא יודעים JS, אלא את הפריימוורק שמבוסס עליו, כמו ריאקט node אנגולר וכו'.

יש דפדפנים רבים שצריך לתמוך בהם, כרום ופיירפוקס מעודכנים באופן קבוע ע"פ תקינה, בעוד אינטרנט אקספלורר לא מתעדכן ומצריך התאמות שבדרך כלל ארוכות יותר מכל הקוד.
יש גם אפליקציות אנדרואיד לעומת אייפון, יש לאנדרואיד גם אלפי גרסאות שונות שצריך לתמוך בהן. יש כיום פלטפורמות כמו למשל unity אשר מאפשרות לפתח לשתי הסביבות בלי להכיר את שתיהן.

פעם רוב הזמן של הפיתוח התבזבז על הזמן קומפילציה (build), במערכות גדולות יותר כך זמן ה-build גדול יותר. על מנת לקצר את התהליך הזה המציאו את ה-cdi, שרתי build אשר מקמפלים את הקוד באופן קבוע על כל שינוי שמבצע המתכנת ומריצה בדיקות.
Jenkins הוא קוד פתוח שעושה את זה.

Docker - הדבר הכי חם כיום בתחום התוכנה, כיום כולם משתמשים, עוברים או מתכננים לעבור. ארכיטקטורה המאפשרת לארוז קוד ולהעביר לסביבה אחרת.

סביבות עבודה כל הזמן משתנות, ותמיד יהיה כלים שחוסכים במקום דקה שתי דקות, ואנשים יעדיפו את הכלי שחוסך יותר זמן, וינטשו את הקודם.

תרגול - התרגול הוא בזוגות, להחליף קבוצות. בשלב הראשון כל אחד מהצוות יציג לחבר החדש את הפרויקט של הצוות שלו, יתקין אותו על סביבת העבודה שהוא עובד עליה ויראה לו מה הוא עושה. ואחרי זה להתחלף.

שיעור 8 - חזרה על שיעורים קודמים

לכל מוצר אמור להיות צורך, אני לא הייתי פה כשהוא דיבר על זה, אבל הוא אמר משהו על דודה שהיתה צריכה משהו, ואם המוצר לא מספק את זה?

הגדרות דרישות agile-יות

בגיט, אנחנו תמיד עושים קודם pull ורק אח"כ push, כי לפני שאנחנו שולחים אנחנו צריכים לבדוק אם יש שינויים חדשים, עושה merge אצלי במכונה, ודוחף בחזרה לשרת.

רישיונות שימוש - אחד הנושאים המשעממים בקורס, אבל אנחנו לא משפטנים וחשוב לדעת מה כל רישיון כזה אומר, ומה המחויבות שלנו לרישיון הזה.. כמו כן, דיברנו על ההסכמים שאנחנו צריכים לחתום עליהם, ולמה קיימת הדרישה הזו.

ההבדל בין copyright ל-copyleft - ב-copyright אני קובע מי יכול ומי לא יכול להשתמש בקוד הזה. ב-copyleft אני קובע מה הזכויות של מי שרוצה להפיץ את התוכנה. בדרך כלל, אם אנחנו מפיצים את התוכנה, אנחנו צריכים להפיץ יחד איתה את קוד המקור ואם עשינו שינויים, אנחנו צריכים להפיץ אותם יחד עם קוד המקור.

Fork, הזכיר את הנושא של חברות שקנו open source וגרמו ל-fork של אותו פרוייקט (כמו mariaDB מ-MySql). כמו כן קיימים forkים שלא קשורים לקניות, למשל כל הפצות הלינוקס או ביטקויין.

מכונות מדברות - מה החשיבות של הקישוריות הזו בין תוכנות בקוד פתוח. תוכנות עושות משימות קטנות, אם יש לנו צורך גדול, אנחנו משתמשים בהרבה תוכונת קטנות והן צריכות לדבר ביניהן.

דיברנו על החשיבות של design patterns. הזכירו את ה-observer, visitor ועוד…

תיעוד - לא כותבים תיעוד בתוך הקוד, למעט בראשית של פונקציות ופרוצדורות. עדיף לא להסתמך על התיעוד הזה, משום שיכול להיות שמישהו לא עדכן אותו אחרי שעשה שינויים.

מערכות מודרניות משתמשות ב-micro services, או dockers כיום. וכל אחת מהן מייצרת לוגים שונים בפורמטים שונים, ואחד האתגרים הגדולים האלו בעולם התוכנה הוא לשלוף את כולם ולחבר ביניהם את ההודעות.

תיעוד של commitים, מכילים את השינויים שביצענו בקוד. סוג נוסף של תיעוד הוא wiki או README כדי שאנשים חדשים לא יציפו אותנו עם שאלות. בדרך כלל אנשים חדשים שנכנסים לפרוייקט מתחילים בתיעודו כדי ללמוד אותו.

היתרון של קוד פתוח הוא קבלה של פיצ'רים חדשים מהיוזרים.

יש גם את נושא ה-bounties, קהילות ההאקרים פחות משתפות קהילה כאשר אין בזה כסף - ואין מתנות חינם.

חוק לינוס - יותר עיניים עוברות על הקוד פחות באגים.

האידולס - אלו שרק מסתכלים על הקוד, לא באמת נכנסים לתוך הקוד ומוצאים את הבעיות אבטחה בהן.
קיימים פרוייקטי קוד פתוח אשר קיימים בשביל לעזור לפתור בעיות תאימות של חברות מסוימות, כמו למשל האי תאימות של האקספלורר ל-html מודרני.

וירטואליזציה - נחלק את משאבי המחשב שלי למספר מכונות וירטואליות

<u>גישות ברנצ'רים</u>

- **המסורתית** - כל אדם פותח ברנצ' משלו, מסיים את העבודה שלו ועושה merge חזרה.
- **טראנק בייס** - יש ברנצ' עבור הגרסה בפרודקשן, ויש ברנצ' בשביל הפיתוח שמתקדם ממנו, אם מתגלת תקלה בפרודקשן, אפשר לתקן שם ולמשוך לפיתוח, עד שתצא גרסה חדשה והגרסה תעבור מהפיתוח לפרודקשן.
- **צ'ריי-פיק** - המנהל בוחר את השינויים שמעניין אותו להכניס לגרסה והוא עולה להם merge.

אפאצ'י היא לא חברה בפני עצמה - יש להם עובדים ומשכורות, אבל זו לא חברה.

שיעורי 9 - מודלים כלכליים

הגעתי באיחור, דיבר על הענן כשהגעתי.

דיבר על כך שמיקרוסופט היו נלחמים משפטית בפרויקטים של קוד פתוח. כשהתחלף המנכ"ל הם שינו כיוון.

יש כיום shift ברשיונות כדי למנוע ממה שחברות גדולות כמו Amazon עושות - מציעות שירותי ענן המבוססים על פרוייקטי קוד פתוח. גוגל הלכו בכיוון השני ויצאו במודל בו דיברו מול מיזמי הקוד הפתוח והגיעו להסדרים כספיים

מודל שלישי - open core, חברת הקוד הפתוח מציעה את הליבה שלה בקוד פתוח (למשל Elastic) ומציעה מוצרי מעטפת ופלאגינים בתשלום.

חברה ישראלית jFrog - היא מחזיקה פרוייקט קוד פתוח עם גרסת קהילה וגרסה נוספת עם רישיון לשימוש מסחרי שמאפשר למשל להשתמש ב-Dockers.

רישיון כפול - רישיון אחד לשימוש קהילה, ורישיון אחר לשימוש מסחרי.

תמיכה - מודלים מסויימים שמאפשרים תמיכה בתשלום.

אפאצ'י - מכניסים רק מתרומות, דיבר על referral מול מייקרוסופט וגוגל כשגוגל ניצחו והציעו יותר כסף או משהו כזה..

מרצ'נדייז - קניית חולצות וסמלים עם מותג הפרויקט.

באונטיז - אפשר לעשות מזה כסף, לא מומלץ לפרוייקט.

לינוקס - הבסיס של פרוייקט הלינוקס עובד לחלוטין בגיוס תרומות, גם כאשר הם יוצאים למלחמות משפטיות ואין לפרוייקט הלינוקס כסף בשביל לעשות את זה כל מיני עמותות וארגונים מגייסים את הכסף כדי לעזור

פרסומות - צריך להגיע לנפחים מאוד גדולים בשביל להגיע לרווחים מזה..

כשחברה רוצה להפוך משהו שלהם לקוד פתוח, בדרך כלל אין אצלם ידע על איך לנהל פרויקטים כאלו של קוד פתוח. כך שיש להם שתי אפשרויות, לשלוח את הקוד לאוויר העולם וייתכן שעם מזל מישהו יקח את זה לאיפשהו. או שהם יכולים לשלם לאנשים מקהילת הקוד הפתוח המון כסף בשביל שינהלו את זה עבורם. הוא אומר שאפשר לתת את זה ל-apache foundation שיש להם ניסיון בניהול פרויקטים כאלו. פרוייקטים חזקים יכולים לעזור לפרויקטים אחרים.

שיעור 10 - בלוקצ'יין

כל בלוק מכיל את המידע של עצמו, וקישור לבלוק הבא. הסדר מוגדר על פי מספרי הבלוק.

אי אפשר לשנות את מה שכבר היה.

מתלהבים מזה בגלל ביטקויין.

החורף של הבלוקצ'יין - בזמן האחרון קצת פחות מראים עניין בבלוקציין.

איטריום - המייסד מוזר אבל מדבר לעניין. במקום שאנחנו ניקח את העבודה של נהג מונית, ניקח את העבודה של Ober.

פונקציית hash - לוקחת משהו מאוד גדול והופכת אותו למשהו בגודל קבוע (תמונות, מספרים, טקסטים). בהינתן הפלט, רוצים לדעת מה היה הקלט, זו משימה מאוד קשה. בדרך כלל אם הוא ממש יתאמץ, הוא ימצא הרבה אפשרויות שונות לקלטים.

טרנזקציות - בבלוק אנחנו רושמים טרנזקציות. טרנזקציה בביטקוין מכילה ארנק המקור, ארנק היעד ואת הסכום. יש רשתות פומביות כמו ביטקוין או איטריום. כל מי שיש לו מחשב יכול להפוך להיות node של ביטקויין בעצמו. יש רשתות שיש להן permissions, שכל node הוא משלו, למשל בנקים ורשויות מיסים למשל יכולות להיות.

בעיית הגנרל הביזנטי - יש שני גנרלים, אם רק אחד מהם תוקף הוא מפסיד, ועל כן הוא רוצה לדעת שהצד השני תוקף. יש מספר סיכונים, למשל החבר'ה באמצע שינו את ההודעה באמצע. זו בעיה לא פתירה, אי אפשר להיות בטוחים במאה אחוז האם לתקוף או לא לתקוף. פתרון לבעיה הוצע בשנות ה-80, אלגוריתם שעובד עם ⅔ מהם לא בוגדים. כל מי שמקבל את ההודעה מפיץ לכולם את ההודעה. כלומר, בביטקויין, כל אחד מודיע לכל האחרים את העסקה, ולא מבצע עד שכולם מאשרים.

רוב המטבעות משתמשות באלגוריתם שכתב סטושי
https://satoshi.nakamotoinstitute.org/emails/cryptography/11/

אתה שם את כל הטרנזקציות שאתה רוצה לבצע על אותו הבלוק, ניקח את הבלוק נוסיף לו איזה מספר, נבצע עליו את פונקציית ה-hash, וכל אחד יכול להוסיף את המספר הזה ולבדוק. הראשון שמוצא את המספר הזה הכי מהר נקרא "כורה", והוא יקבל על זה פיצוי בדמות כספית. סטושי רצה שכל 10 בלוקים תבוצע בדיקה, וכל פעם התוכנה תגדיל את הסיבוכיות כדי לא ליצור אינפלציה על כל 1000.

כיום יש מחשבים מיוחדים שמיועדים בשביל לבצע את זה, אנשים מתחברים ל-poolים שביחד מוצאים את המספר. Proof of work (הרבה חשמל מתבזבז על כלום).

יש אלגוריתמים אחרים כמו Proof of state שנועדו כדי לפתור את הבעיה הזו, למשל באיטריום.

חוזים חכמים (או סוכנים אוטומטים) - גילו עם הזמן שאותה שפת סקריפטים של הביטקויין שנועדה כדי לבדוק אכיפה של החוקים (למשל לבדוק שבאמת יש לך 100 ביטקויין לפני שאתה מעביר אותם - בודקים את הארנק ובודק את כל העסקאות שבוצעו על הארנק).
שפת הסולידיטי - אפשר לכתוב בה כמו בכל שפת תכנות אחרת.

שרוצים לעבור לגרסה אחרת, צריך שכל ה-node-ים יעברו לגרסה החדשה.
שינוי של סקאלביליות - חסם של ביצועיים, הדבר הזה לא סוחב, הוא לא יעיל, הוא בטוח. כלומר, אף אחד לא יחכה 10 דקות בשביל לקנות פחית קולה או פלאפל.

שיעור 11 - CMS ו-Web

משתמשים לא מתקינים PyCharm, הם משתמשים ב-Web, או מתקינים אפליקציה. אין טעם בקוד שלנו אם אף אחד לא ישתמש בו, ועל כן צריך להנגיש אותו.

צריך לתת להם כמה שיותר מוקדם להשתמש בזה, ולקבל מהם פידבק, ולכן ה-Web הוא מאוד חשוב כי הוא חוסך לנו הרבה מהבעיות המאפיינות את עולם התוכנה.

ה-Web לוקח אותנו למקום שלכולם יש דפדפן או טלפון. אנשים לא אוהבים להתקין דברים, אם הם לא מתכנתים, הכי הרבה שהם מסכימים זה להוריד את האפליקציה מהחנות.

צריך לעבוד לפי הסטנדרטים, הדפדפן יודע סך הכל להפעיל HTML, CSS, JS, ושום דבר אחר מעבר לא. לכן זה כל מה שאנחנו צריכים ללמוד, אבל (לדעתו, HTML זה קשה ומבעס, ו-CSS זה גם קשה, ומבעס ולמצוא פתרונות להכל זה הרבה עבודה). לכן קיימות הרבה תשתיות וספריות שעושות את זה.

יש לנו jQuery, Bootstap, react, Angular

ויש עוד כל מיני לעסקים ספציפים, כמו למשל גרפים (dfree), ואם אנחנו רוצים למובייל נשתמש ב-jQuery Mobile.

CMS או **Content Management System** סביבות פיתוח אשר מספקות חבילה של כל החלקים אשר נדרשים בהקמת אתר תוכן. הכי הרבה זמן לוקח בווב זה החלק של העיצוב, וזה משהו שמגיע כחלק מה-CMS.
מי שמייצר את ה-HTML זה השרת.

מפתחים משתמשים ב-F12, ה-Dev Tools של הדפדפן.

יש שרתי Web שהמטרה שלהם לתת רק תשובות. קיימים שרתי Proxy שמיועדים כדי להתמודד עם הרבה בקשות, כדי שהשרת Web המרכזי לא יצטרך לבצע את אותן פעולות הרבה פעמים עם תוכן סטטי (דפים שכבר יצרנו, לא צריך לייצר אותו מחדש, ניתן לשלוף מהמטמון - Cache).
אך יש עם זה בעיות לפעמים, מה קורה אם ביקשנו דף שהמטמון לא מכיר, או כאשר משתמשים שונים אמורים לקבל תוכן שונה באותה כתובת?

למה CMS הוא בדרך כלל קוד פתוח?
30% מהאתרים באינטרנט רצים על WordPress ו-60% מהאתרים עם CMS רצים על Wordpress. וורדפרס היא הדרך הכי מהירה כדי לעלות אתר לאינטרנט כמו למשל Wix, בשונה מ-Wix שאנחנו מחזיקים את האתר על שרת שלנו. גם ל-Wordpress.com יש שירות המאפשר Wordpress מאוכסן בתשלום.

מספר 2 זה ג'ומלה, ומספר 3 זה דרופל. הם יותר מתאימים לאתרים עם דרישות יותר מסובכות, הם כולם כתובים ב-PHP, בעיקר כי במקור ל-PHP היה מנגנון Caching מאוד חזק (החליפו אותם מאז, זה מלפני כ-10-15 שנה).
חברות שיושבות מאחורי ה-Open Source האלו שוות מיליונים כיום.
קוד פתוח מריץ יותר מ-60% מהווב. C# לעומת זאת פחות מ-20%.

יש חבילות כמו LAMP או XAMP אשר נותנות סביבות של Apache, MySql ו-PHP על השרת יש חבילות שמחליפות את ה-MySQL למשל ב-MariaDB. מסדי הנתונים האלו עובדים לדעתו יותר טוב ממסדי נתונים בתשלום של אורקל ו-MS SQL. את כל הדברים הללו אפשר לעשות גם על Windows (אבל הרבה פעמים זה לא עובד טוב).

Ruby זה קצת פאסה כיום לדעתו. כיום רוב העבודה של הווב נמצאת ב-client side, מה שנקרא Full Stack היום. אלו שזה ההכשרה שלהם יודעים טיפה Server אבל רוב העבודה נמצאת בקליינט. עבודה בעיקר של להציג את המידע טוב ויפה, שזו עבודה ב-JS בעיקר כי רוב העבודה על ה-Server כבר בעיקרה כבר נעשתה עבורנו.
כיום משתמשים בעיקר ב-NodeJS שעובד מאוד מהר וטוב.

ציונים של התרגול:
יש שני דברים מאוד חשובים בעבודה המעשית שלנו:
1. מה שאנחנו עושים
2. מה שאחרים עושים בשבילנו

הפרויקטים ימדדו לפי כמה שהם שימושיים - הדרך לדעת כמה הם שימושיים זה כמה fork‌ים, commit‌ים מספר ה-issue‌ים הפתוחים.
הוא אומר שהציון הפרויקטלי הוא זהה **גם למי שלא עשה כלום**

ציון אישי: על פי תרומות לפרויקטים שלא שלנו
הציון הוא חצי-חצי, חצי על מה שעשינו מהצד שלנו, וחצי על מה שעשינו בפרויקטים אחרים.

צריך להגיש דוח מסכם של כל הפעילות שלנו בפרויקטים אחרים והפרוייקטים שלנו.
יש לכתוב למעלה "שכל מה שעשיתי הוא נכון ואמיתי" ויש לשלוח אותו לשבוע הבא.
תרומות של 10 דברים זה נחשב טוב...
צריך לשלוח את זה עד שבוע הבא.

הוא אמר שהוא יהיה בבעיה אם מישהו לא עשה 10 קומייטים הוא בבעיה..

הפרמטר הוא כמה שורות תרמנו...

עבודה קבוצתית:
ראש הקבוצה צריך לשלוח את הדו"ח הקבוצתי שבו הוא יציין את הפעילות של חברי הקבוצה עשו בתוך הקבוצה ואת התרומות שקיבלנו בפרויקט. הוא אומר שניתן לצ'פר אם יש מישהו שנתן המון עבודה על הפרויקט וצריך לציין את זה (מישהו מבחוץ).

שיעור 12 - סיכום

"המבחן יהיה פה", אמר חמש דקות אחרי שנגמר השיעור...