

המרצה: צביקה גיטרמן
אדמיניסטרטיבי: אין תרגילים. בסוף הקורס יש מבחן שיהווה 100% מהציון הסופי.

הקדמה

סקירה של חומר הלימוד בקורס:

- **SCM** – איך אוגרים קוד?
 למה צריך לאגור קוד? כי הרבה פעמים משנים אותו ולפעמים רוצים לחזור לגרסאות קודמות.
- **Build tools** – הצורה הכי פשוטה היא קומפילציה משורת פקודה. קצת יותר מסובך הוא ה-Makefile ויש עוד...
- **הנדסה הפוכה** – זה התהליך שבו חוזרים מקובץ בינארי ל-source code. צריך את זה הרבה פעמים אם המתכנת שכתב את התוכנה כבר לא עובד בחברה או שהתוכנה שייכת לחברה שלא קיימת יותר (אלה הם השימושים החוקיים...)
- **Bug reporting** – **באג** זה כשהאפליקציה לא עושה את מה שהמתכנת רצה שהיא תעשה. אנחנו נדבר על הנושא הזה מנקודת המבט של המתכנתים.
- **דוקומנטציה** – האם כדאי? מהם היתרונות והחסרונות של דוקומנטציה כחלק אינטגרלי מהשפה (למשל, **javadoc**)?
- **Logging tools** – סטנדרטיזציה של **logging**
- **היבטים משפטיים** – נדבר על זה מנקודת המבט של המתכנת. מה כדאי שיהיה בכל ההערות המשפטיות שרואים לפני שמתקינים תוכנה? מה זה בדיוק open source? האם זה אומר שאפשר להשתמש בזה מתי שרוצים ואיך שרוצים? התשובה היא לא טריוויאלית...

יש פער בין העולמות האקדמי למציאות. באוניברסיטה הכל תיאורטי וטוב ויפה אבל זה לא עובד ככה במציאות... אחת ממטרות הקורס היא לקשר בין העולם האקדמי לעולם האמיתי.

מה אנחנו מחפשים?

- איך עושים תוכנה? – כלים שונים שעוזרים לנו וחוסכים זמן.
- יציבות – אם מצאנו כלי מאוד מגניב זה לא ישר אומר שצריך להשתמש בו. יכול להיות שאנחנו הראשונים שמשתמשים בו ואנחנו לא רוצים להיות הראשונים שיימצאו באגים...
- תמיכה – מי תומך בכלים האלה? IBM או קהילת ה-open source?
- סטנדרטים ידועים – למשל XML זה סטנדרט לאגירת מידע שיש לו תמיכה בהמון שפות. אם משתמשים במשהו לא סטנדרטי תהיה בעיה בקופמטביליות. סטנדרט נוסף הוא למשל ה-web service. פשוט פונים דרך port 80 עם בקשה ובכלל לא חשוב מה עומד מאחוריו ואיך העניין ממומש.
- אינטגרציה עם כלים/פלטפורמות אחרות – כדי שיהיה קל בחיים
- פשטות – אנחנו רוצים שהכלים יהיו פשוטים לשימוש ולתחזוק. מ הסתם יש Trade-off בין פשטות ליכולות...
- מחיר – לא נדבר על זה בקורס אבל בחיים האמיתיים כדאי לחשוב על זה...
- התאמה לגודל הפרויקט – יש הבדל בעבודה בצוות של 3 אנשים בחדרון קטן לבין עבודה על פרוייקט עם עוד אלפי אנשים שמפוזרים בקמפוסים שונים.
- האם אנחנו רוצים שהרבה אנשים ישתמשו בטכנולוגיות שאנחנו משתמשים בהן? זה תלוי בכל מיני גורמים. למשל, אנחנו לא רוצים להיות הראשונים שיגלו באג בתוכנה של כור גרעיני. מצד שני, אם אנחנו מפתחים משהו חדש אולי לא נרצה שעוד אנשים יעשו את זה...

- כנ"ל לגבי open/closed source. בד"כ אומרים שמה שטוב בקוד פתוח הוא שאפשר לשנות אותו. אבל לא תמיד כדאי. אין הרבה אנשים ששינו את ה-scheduler של לינוקס...

מוטיבציה לקורס

- באג Y2K – בשנתיים שלפני הייתה תחושה מאוד לא נעימה והושקעו המון משאבים!
- ב-Win 2K יש 35 מיליון שורות קוד, 1400 מפתחים ו-QA 1700-ים!! במיקרוסופט מאוד משקיעים בבדיקות. יש להם יחס של כ-2:1 בין בודקים למפתחים לטובת הבודקים!! בלינוקס יש 6 מיליון שורות קוד!
- בפרוייקטים בסדי גודל כאלה אי אפשר לנקות את הקוד מבאגים לגמרי. כמות הקוד אולי עולה לינארית אבל כמות המצבים שהתוכנה יכולה להיות בהם עולה אקספוננציאלית. פשוט אי אפשר לבדוק את כל זה!! לכן גם תמיד לפני התקנת תוכנות המשתמש צריך לחתום על EULA – הסכם שאומר שאין על החברה שום אחריות ושהיא לא מתחייבת לשום דבר!!

Source Control Management

כשמפתחים תוכנה היא יכולה לכלול המון קבצים מסוגים שונים. עץ המחיצות (directory tree) הוא מאוד מורכב. חלק מהקבצים נוצרים בזמן הבניה (למשל, קבצי object, לוגים). נשאלות השאלות הבאות:

- מה מכל הבלאגן הזה אנחנו רוצים לשמור?
- באיזה מבנה אנחנו רוצים לשמור את זה?

דרישות

- ניהול התנגשויות בין קבצים – אם יש כמה מפתחים שעובדים על אותו הקבוצ צריך לדעת איך לאחד אותם
- מתן הרשאות שונות לאנשים שונים כדי שאף אחד סתם ככה ישנה דברים שלא צריך לשנות.
- אפשרות עבודה online ו-offline – כי יכול להיות שיש אנשים שלא מחוברים כל הזמן ואז יש להם גרסאות של הפרוייקט על המחשב האישי. במקרה זה, צריך לדאוג לסנכרון פעם בכמה זמן.
- יכולים להיות הרבה מוצרים שיוצאים מאותו קוד מקור – גרסאות שונות וכו'. צריכה להיות יכולת לנהל כמה מוצרים.
- תמיכה בעבודה בקמפוסים שונים שגם יכולים להיות ביניהם הבדלי שעות. צריך להחליט איך מעבירים את הקוד בין הקמפוסים – האם יש פס רחב שמקשר בין כולם אן שיש מבנה נתונים לכל קמפוס והם מסתנכרנים פעם בכמה זמן.
- ארכיבים ולוגים – צריך לשמור מתי עשינו מה (למשל, מתי הוספה פונקציה מסויימת). זאת גם דרך לראות מי כותב הכי הרבה ובאיזה שעות, מי פותר הכי הרבה באגים ומי יוצר אותם...

אוצר מילים

- repository – המאגר שבו אוגרים את הקוד
- working/local copy – עותק של הפרוייקט שנמצא במחשב האישי
- check-out – הוצאת גרסה מהמאגר
- check-in – הכנסת גרסה לתוך המאגר
- Version – גרסאות שונות של התוכנה
- Trunk – הגזע המרכזי של הפרוייקט וממנו יוצאים עצים קטנים
- Lock – נעילה של קבצים כדי שאף אחד אחר לא ייגע בהם
- Branch – הענפים של הגרסה המרכזית
- Tag, label – סימון לגרסאות (בד"כ מילולי) כדי שיהיה נוח לזכור ולחזור אליהן במקרה הצורך.

- Merge
 - 0 במקרה ששני אנשים עובדים על אותו קובץ צריך לאחד את שתי הגרסאות
 - 0 חיבור של ענף לגזע המרכזי. עדיף לחבר כמה שיותר כי אז למשל לא צריך לבדוק באגים פעמיים.
- Metadata – כל מיני נתונים כמו לוג, סביבת עבודה ועוד

המלצת הבית

לתכנן את הפרוייקט, להבין מה גודל הפרוייקט, כמה מפתחים צריך, QA, דוקומנטציה וכו'. זה שווה כל דקה שיושבים מראש. אחרי שכבר יש כמות גדולה של קוד זה מאוד מאוד מאוד time consuming לשנות עץ פרוייקט. לכן חשוב לתכנן מראש כך שיהיה נוח לשלוט. טוב לחלק להרבה חלקים ולא כדאי שבאותה תיקייה יהיו הרבה קבצים.

האם מכניסים לפרוייקט את כל הקבצים הבינאריים, object-ים, exe-ים וכו'? זה תלוי. אם זה משהו שלא רוצים לקמפל מחדש (כי זה גדול או מסיבות אחרות) אז כדאי להכניס. אחרת, אם עובדים על משהו קטן שמשתנה כל הזמן אז מן הסתם צריך לקמפל כל הזמן מחדש ואז לא צריך לשמור את קבצי ההרצה.

גרסאות

או שיש גרסה גלובאלית או שלכל ענף יש גרסה. המספר נותן אינדיקציה לבשלות הקוד. יש מוסכמה שמה שנכנס ל-source control הוא בשל. אבל "בשל" זה משהו סובייקטיבי. המכנה המשותף הנמוך ביותר הוא שהקוד מתקמפל.

בשבוע שעבר דיברנו על SCM, איך בונים את העץ, איך נותנים גרסאות וכו'. כששמים גרסאות צריך לשאוף שתהיה לזה משמעות וזה לא אמור להיות קשור בכלל לגרסאות של אנשי השיווק. למשל, בשיווק יכולים להחליט למכור את המוצר בתור גרסה 8 בעוד שזה בכלל ה-build הראשון שקיים.

יש עשרות חברות ייעוץ בינלאומיות שמטרתן לקחת מישהו שמתחיל לעשות פרוייקט תוכנה ולתת לו סטטיסטיקה על מה שקורה מבחינת טכנולוגיות וכאלה. העבודה שלהן מתחלקת לשני חלקים:

1. מידע סטטיסטי סיכומי על מה שנכון לאתמול (למשל, מעי ההפעלה הנפוצה בעולם היא חלונות XP שיש לה כ-70% מהשוק. השנייה בשכיחותה היא חלונות 2000 שמוקנת ב-7% מהמחשבים). בד"כ הנתונים של החברות די אמינים.

2. תחזית והשוואה בין מוצרים ובים חברות

בין המובילות בתחום הן [Forrester](#), [IDC](#), [Gartner](#) ועוד. [Tolly Group](#) מבצעת סקרים לפי בקשת הלקוח ואז יוצא שהסקרים מוטים משום שנדבקים רק חלק מהפרמטרים. לעומתה, Gartner עובדת לפי ה-agenda שלה ולא לפי מה שמבקשים מהם והיא נחשבת הכי אובייקטיבית ואמינה.

האנשים שעובדים בחברות האלה נקראים אנליסטים ויש להם השפעה רבה על מה שקורה. למשל, Gartner ממליצה לעבור ל-Vista רק כשייצא איזה pack נוסף. החברות נבדלות לא רק באמינות אלא גם בכמה שהמידע שהן נותנות הוא מדיד. יש חברות שכותבות כל מיני דברים מעורפלים שדי קשה לבצע החלטות לפיהם.

כלי SCM בולטים

- [RCS](#) – פותח ב-1980 וקיים ב-unix, linux, cygwin. הוא open source ויחידת העבודה שלו היא הקובץ. הוא פשוט משתמש ב-diff כדי להשוות קבצים שזה בזבזני עבור קבצים בינאריים וזה גם לא עובד לספריות שלמות מה שהופך את הכלי הזה לדי לא מעשי.
- [CVS](#) – משתמש ב-RCS ברמת הקבצים. גם כן open source והוא מתאים להרבה מערכות הפעלה. זה כלי מאוד נפוץ בעולם. יש לו הרבה plug-in ים ובברירת המחדל הוא non-locking – כלומר שני אנשים יכולים למשוך את אותה הגרסה ולעבוד עליה בו"ז. כמובן יהיו צרות ב-merge.
- [VSS](#) – של חברת [מיקרוסופט](#). זהו כלי מאוד נפוץ. האגירה שלו היא בצורה בינארית, כלומר הכלי הוא סגור. יש מאגר נתונים בינארי וזה לא טוב במקרה שמשוה נדפק בדיסק הקשיח כי אז התכונה לא מצליחה לפתוח את הקובץ וזהו. אמנם לאחרונה התפתחו כלים שיכולים לתקן את הנוק. ברירת במחדל שלו היא שהוא locking. ל-VSS יש אינטגרציה עם visual studio אבל חייבים לעבוד בחלונות. חוץ מזה הוא לא מתפקד טוב כל כך ברשת. מה שמשך הרבה אנשים לכלי הזה הוא שיש לו [GUI](#). נקודה מעניינת היא שמיקרוסופט עצמה לא משתמשת ב-VSS לפיתוח שלה.
- [Team System](#) – כלי חדש של מיקרוסופט.
- [Clear Case](#) – זה מוצר הדגל של חברה בשם [Rational](#) שנקנתה ע"י [IBM](#). זה קיים לכל מערכות ההפעלה ותומך גם בעבודה ב-Offline אבל ההתקנה היא מאוד כבדה. לאחרונה עובדים על גרסאות light. למוצר הזה יש שליטה בשוק ה-high end – כ-30%-40%.
- [Subversion](#) – נכתב במקור כמחליף של CVS בשנת 2000. הוא מבוסס על [Apache](#) וביצועי הרשת שלו טובים. יש לו גם תמיכה יותר טובה בבינאריים.

Build

מה עושים ב-build? לוקחים מלא קבצים של קוד והופכים אותו לבינארי. האמת היא שזה לא רק קוד, אלא גם תמונות, מפתחות הצפנה וכו'. בסוף יוצא מוצר – בינארי, תיעוד ועוד דברים נלווים. שיקול העצלנות הוא מוביל. רוצים שהתהליך יהיה כמה שיותר אוטומטי ושהסקריפטים יעבדו בשבילנו ולא להפך. מסביב המעטפת צריכה להיות נוחה. למשל, אם קרה משהו לא תקין אנחנו נרצה לדעת מזה. חוץ זה רצוי כמובן שתהיה אינטגרציה עם ה-SCM כדי שתהיה יכולת למשוך קבצים.

אוצר מילים

- Action – פעולה אוטומית, למשל קומפילציה
- Target – היעד של התוצרים – object, exe וכו'
- Dependencies – מי תלוי במי? זה חשוב מאוד לביצועים. כמובן היה אפשר לקמפל הכל מחדש כל פעם שעושים שינוי קטן אבל זה יכול לקחת מלא זמן. אם אנחנו יודעים איזה קובץ תלוי באיזה קובץ אפשר לקמפל רק את מה שרלוונטי.
- התלויות גם מגדירות את סדר תהליך ה-Build.
- Time-stamps – לכל קובץ יש חתימת זמן – יצירה, שינוי וקריאה אחרונה. Make ועוד הרבה מערכות בעבר בנו את כל תהליך ה-build לפי חותמות זמן. זו כמובן בעיה גדולה אם המערכת מבוזרת והקבצים בה יושבים על יותר ממכונה אחת ויש בעיה של סנכרון שעונים.

פתרונות

- אפשר לכתוב סקריפט שעושה את ה-Build
- כלים ויזואליים (כמו visual studio) שנותנים לעשות את ה-build דרכם. רק צריך לזכור שמתחת ל-visual studio יש בעצם את הקומפיילר ואת nmake שהו הגרסה של חלונות ל-make.
- Make – כלי שנכתב ע"י מי שהמציא את פורטרן (ולכן הסינטקס שלו כזה נוראי). זו אפליקציה Open source, היא עובדת בהרבה פלטפורמות ויש לה הרבה וריאציות. Make יודע רק את החוקים שאנחנו כותבים והוא עובד לפי שעון. הוא יודע שאם משהו השתנה אז הוא צריך לעשות את מה שאמרו לו. הוא לא מכיר את שפות התכנות עצמן אלא הוא עובד יחד עם המהדרים.
- יש ל-make הרבה חסרונות:
 - 0 הסינטקס שלו מעצבן והשפה לא קריאה.
 - 0 אם הפרוייקט מאוד גדול קשה לכתוב makefile לבד.
 - 0 בגלל שהוא משתמש בכלים של המערכת אותה גרסה לא תעבוד על פלטפורמות שונות.
 - 0 לא תומך ב-build מבוזר על פני מחשבים שונים (אבל יש לזה פתרונות)

Compiler cache

בגלל בעיות של תלויות אנשים מאבדים בזה אמון והרבה פעמים עושים את כל ה-build מחדש. זה כמובן פוגע מאוד בביצועים. אז יש טריק חדש – compiler cache. מה שגורם לבעיות הוא בד"כ התהליך של ה-preprocessing. זה לוקח קובץ cpp עם macro-ים למיניהם והופך אותו ל-c++ טהור ואת זה כבר אפשר לקמפל לקובץ object. אז כל פעם שהופכים c++ טהור ל-Object גם שומרים עותקים של המקור ושל התוצאה. ואז כשמקמפלים קובץ חדש בודקים אם הוא כבר קיים במאגר ואם כן לוקחים את ה-Object ששמור. זה מפשר את הביצועים עד פי 10!

6) 10.6.07
SET

הוצאות: 17/6 - אין שינוי
24/6 - שינוי המס
12/8 - מוצא

שינוי בית: (c) אמצע אתר הצו"ה של forrester & SCM
לדקדק אתו (צו"ה)

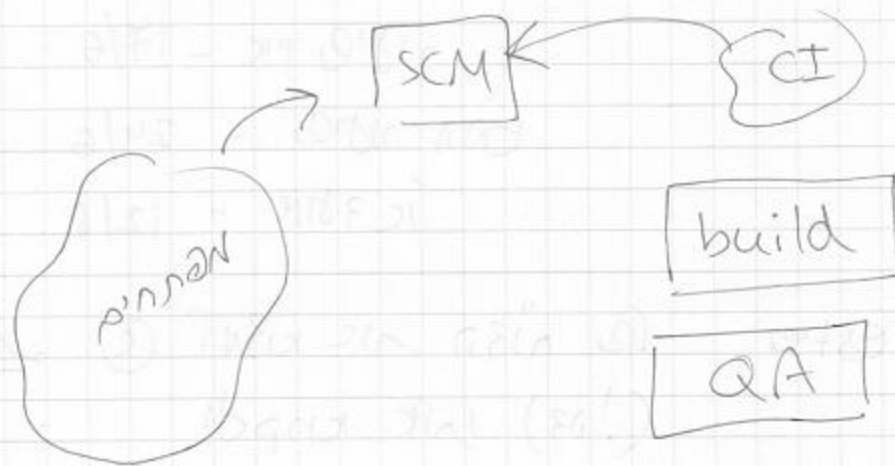
למנוע שגיאות בקודים של building tools. במקום הזה נכנסו
לפני כן אתר החברה הישנה XOR-EAX שמוקד בהיסוד
ההוא והתנ"ה.

ANT - בשנת 2000 Ant התחל לנהל הסטנדרט של
ה-build. open-sources. בהתחלה הוא התמקד בעזרת
ה-Java ותוכן עצמו גם כן. הוא אמר לפתור
ם מנייה קצו"ה שיש לו make. הוא גם מנהל המון דברים
בתוכו. הוא מוצע למאפיינים של השפה והוא עובד על
הפלטפורמה. אפשר גם לכתוב tasks משלנו. הוא יצר לעבוד
לפי SCM ויש לו plug-ins גם לרוב התוכנות המסחריות
(אולי התחלנו).

Continuous Integration

מה שאנו יקדוק לו הוא שצרכים במהלך האיש' על גרסה
דושימה להטעים ומצויות פנימה - SCM, ואז מניבים עוד
טעמים.

אז מה שיש תוכן איזה script שוקד גרסה אחת, קודם
אז, אחר כך טעמים ומצויות למפתחים הפונקציות.
הם נעשה אוטומטית כנגד שנתנה עולם check-in לקדם.



עושים את זה באמצעות CI או משהו כזה. נמנעו גם מלפתח יותר לעולמו build אצלנו. אבל הרעיון הוא שזה יהיה אוטומטי ולא ידני באמצעות.

עושים את זה באמצעות CI או משהו כזה. נמנעו גם מלפתח יותר לעולמו build אצלנו. אבל הרעיון הוא שזה יהיה אוטומטי ולא ידני באמצעות.

Doxygen היא חבילה שמאפשרת לעשות תיעוד ב-C++ ו-JavaDoc. זה דומה ל-JavaDoc, אבל, אפשר להוסיף את כל ספרייה ונתנו יומן את המבנה המלא של הליבררי והיוושה. יש גם סינטקס איות שמאפשר להוסיף הוראות בקוד ל-Doxygen וזה לפרסם את התוצאות.

בשנים האחרונות התפתחו הרבה טכנולוגיות ל-`language aware` וכן מאפשרות לעשות `refactoring` בקוד. למשל, לשנות שם של משתנה, להפוך משתנה למשתנה, למחוקה פני עצמה. ה-VS ו-C# יש הרבה `feature` וזה הוא מאוד מרשימים!

(4)

QA

COG זה מארג מארג תשובות! ב- 2001 תחשבונו הכי זבאה
 על צדד משנה הצבאר ביטולא הינה של אים הנסוי
 של תול האוויר הסבה ליק (אולי לא היוצג) היא להקיר
 אינו זה נובח תשובה.

חלק מתהליך ה-QA היא למצוא לגיטאל חלק היא ארגז אם
 התוכנה ענשה מה להיות אחרת לעשות.

אחר ההתאבדויות היא כמה להלקין ה-QA? בולצאי שלה
 תלוי אם זה משפיע על הי אדם או זה סתם צד צד
 לשמור אדלי. בפדויקטים ביד 90%-30% מה צבא
 היא על סטטים.

מה צדד לעשות כוילה-QA יהיה מוצח? צדק אחר
 היא לשים^ת בודקים. למשל במקדוסוסט של מתנה ים
 3 בודקים. צדק אחר היא לשים אנשים שחורשים מחוז
 ל-QA. ההציה היא לשון כ" אפה לאמה אזה הצורה
 מסוגרת. זה לא שיש תואר ה-QA מאוניבסיטה.
 חומקתיו רוצים לחל על נאפטרון להצדק בל אדם מן השם
 זה חסתי אפשי לתל על 2^n אפטרון ראש מ זה יספר
 תאי הדיכחן במחשב.

היה לקימה שמתקנים באג אחד הרבה פעמים
 מכניסים באימים חדשים. תהליך ה-QA לא נגמר אל פעם!!!
 החומרה יכולה לתמוך בהצדקה יש מקרים למארג קשה למצוא
 כישפעים צדמ וחציה משנה צברים. למשל בל המעבדים
 של אינטל יש feature-ים שמפשים לעשות מין
 קפיקה על ה-cache.

אוצר מילים

Bug, Fault, Error, Defect, ... אפזאים קטנים
 Feature. מתי? כשסמכים ערסה ומציאים אורה

דברים לצדד צרכים האלה ממש לא ישר לומר, אלא בהפך.

Upgrade, Update, Service Pack, Patch, Fix -

release candidate, beta, alpha :

black box, white-box, functional, regression -
קבוצות של קודים שאתם
תארו להם להמשיך עבודה

הצורה של קודים אלה הקוד הפנימי

הצורה של קודים אלה הקוד הפנימי

Unit - בדיקה של יחידה אחת של התוכנה

System - בדיקה של חבילה שלמה

Integration Testing - בדיקה של חבילה שלמה

מבחן - 12/8 - מועד א' } אוניברסיטה אקדמית
9/9 - מועד ב'

עצם זה שאנחנו עושים בדקדוק זה מלפני כחודש אחד, זה
הנכון עצמו. למשל, תוכנית הדקדוק משיימת על המילים
של המערכת אם אנחנו מוצאים מילים להחליף בעייתיות,
ואז הרבה פעמים מילים חומה מיוחדת בלבד לה.

חברה Akamai יש תוכן זהיר אופי ה proxy העולם.
יש להם המון שיתוף ברחבי העולם וזהם מפלגים קבצים.
למשל, מיקרוסופט מפיצה דרכם את ה patch-ים.
ערשיו, כלשהם חצים לבדק מלפני זה סיפור מאוד מורכב, כי
יש להם מלא שיתוף עם העולם. הוצאות:

- אפר סקרים כל העולם והם יבדקו את המערכת. את
המחירים של Mercury עולה בדיוק את זה. זה (קבץ Topaz
המפנינו בעולם שרתים והם עושים בדקדוק.

- יש גם אוטומטים שיוצרים סמלילי ל מני. בעיה של אינטרנט.

- את סמלילי Akamai עשו המו שלם השאירו שזה

אחוז מהליתים של מידע והיה QA וכשיש להם זרם

הכלה הם שמים את זה יק על השיתוף של הדקדוק.

אז כל מה של הדקדוק ש לפת אחיג. השורה של הדקדוק

עולה את זה, הלבד, אם לא נותן עילקוח. בדיוק

עניי שיתוף עילקוח השיתוף מלפני אתה תוצאה שלם.

זה דומה ל אק להדקדוק מניח את המילה היא debug

זה דומה release. הן לא מוציאים פועל בדיוק באותו אופן.

כלים סטטיים אבדיקה:

- הקאמפילר בודק הורה צבחים אמל ל gcc ול הורה
- flag - ים של הורה אמל Wall - יאז
- באדיקה סטטיים אמל Lint
- code review - אחר אדילס אמר אקס אה בקוד
- ול חתי הורה

כלים דינמיים אבדיקה:

- צבחים יטבאדיקה בלמן רילה:
- (יהו) ציסון - יושיה לא חק'א וכו'
- כלים למדידה צמנים (של פוק' וכו')
- Coverage - באדיקה אה משתמשים בלמה שותכנו
- אינה אולורם בקוד בסא אידי שימול ואילה לא

אוק ודעים כמה התורם וטא באמא? באציאה זה לאפלוט
נאובקאסיה מאונקבסטה זה לא שמו להתורם אובדה או לא
בחיים באה זה מושק הורה יארי פלאידי

אטרוקל דדיקה:

- אספור אה כמה הבאדים ליום / לפתח / צוה / מורה
- להגדיר כמה תומרה - ספים שונים של באדים
- אה לא עובר אה הסל עד מה לאפק א-feature
- regression tools - בצב חצים לבחירה הבאה תהיה
- יחוסבה מתנאיה
- acceptance testing - אפאדים האקוד חזה להגדיר מה לה מבח'אני
- אונה שותכמה זה קורה בצב למאזנים פוטניים שותכמים
- אפי בקלה ואז האקוד אדדור אה ה- acceptance testing
- בצב הם ים אמלמים אחרי שצוברים אה הבדיקה וזה
- אורו מוטיבציה אדבה

9

חתי מפסיקים לדבר?

- אשלי אפי ומוח חמדה

- או שקובעים מהו מחירון זמן או חקצב.

Sanity checks - סקסן של בדקור פשוטו כפיו
להחיל מהי והן אמורה לתת או התחולל למה מסדר.
אח' עובדים את ה - sanity כעין טעם להחליק
משורר הבדקור היותו אחת ומקיסות.

Test-first Design - לפי שחמקום קוד כותבים את
הבדקור. זה Black Box. פומר, לפני שכל
מחברנים את לכתוב, לפני שאנחנו מלוחצים, לכתוב
את הבדקור. איננו שכח הקוד יוצא יותר טוב.

Rational Purify

ג' שמיועד למצוא טעויות זמן ריצה ב-C/C++ מצבור
הדברים כמו פנייה איננה סימולטאנית, בעמרים וכו'.
היא לוקה את ה קוד אסמבלר ומשנה אותו בכל מקום
שיש בו אשליה לזיכרון. מחקר זה היה סטטי, פומר פיוק
קוד פוזיציה. כיום יש גם דסאור שפושט את זה דינמית.
גם אופן מול' ישרי ה' גרה חפיה ומחד שלל הבסיות

Valgrind - הקורב מלפחה ה open source
של Purify. אמת ומכירים אותו מ-lab C/C++.

דיווח (היה) לנינוח למוקד של המאמרים - אמרתי לדבר
עמן זאת אם יצי אספרי כמה יל.

אוצר מילים (Bug Reporting)

- חשבוניות - אמרתים / QA / מנהלים. למשתמשים לו תחיד
יש חשבוניות. ב open source יש לפרטים. מצד שני,

Windows מציגה אפילו bug report (תקלה) ושלח, אבל זה לא הסתם אז אפשר להיכנס לאתר (תקנים של מיקרוסופט) שבו נשלחו כל התקלות.

זה דוגמה של האפילו והבטא אי שמתחילים הם סוג של שותפים - הם מסתתרים בשימוש בתוכן יציקה ואם באים והם מצווחים על כל התקלות.

תל מזה משתמשים יכולים למלא bug-report. התבוננות היא בהרשאה שיש לכל אחד לעבור / לעלות. למשל אין שום סיבה להשתמש ובו אבל התארים לקיימים הם רק מצווחים על האדם. בעצמם שרתים 'כל' לצוות האדם. אבל לשנות יכול רק אי שיתל להבטיח שהאדם הבאית והוא אקייים. זה ממש התקנים של התארים אפשר להעלות שאלות זה ממש לתקנים דוגמה.

Bugzilla - שותפה עם אתם אנשים כמו Mozilla.

התכנה והפארה

מטרה של בינארי וחצים (אחד עליו, או שיתבנו קאדפני 4 עלים ואנחנו ערשיו כוצים להבין אותו).

למה?

- דוקומנטציה גבוהה אל נשיונל בעל
- נדון לצדד איך מוציאה מחנה עובד
- התחבש על תוכנה, טווח אותה זה plug-in ים.
- security

10

לכתוב תוכנה ולחשוף source code זו משימה קלה מאוד
(NP-Complete). קומפילציה זה לא תהליך הפיך.
זה תהליך של תוכנה לאט לאט אתה שופך עליה מאסמבלר
ואתה יש עוד כמה משימות אופטימיזציה, אולי לחשוף
משפט מאוד קשה.

אבל, יש מקרים שבהם זה יותר קל. למשל, ה-Virtual Machines.
חור מנה, "חור קל" לחשוף אותה debug version.

כלים:

- ניתוח קוד אסימבלי
- כלים סטטיים לניתוח (זה מה שיש בדבג)
- כלים דינמיים לניתוח כמו debugger

IDA - disassembler שיוצא לקדם דברים דינמיים
של הרבה משימות
סטטיים, שלמות של משימות, לולאות, שורות בשילוש,
פונקציות - מ' קראו לאי וכו'.
זה כל סטטי אבל כיום הם מוסיפים אסמבלר דינמי.

Softice - debugger גרמני ה kernel. הוא מאפשר
לדבג את שורה של האסמבלר.

Copy protection - מן הסתם אנחנו לא רוצים לשתף
אנו את התוכנה. אבל אפשרות זה מושגת להחליף אפליקציה
מכונה של דבר של המערכת שלנו אסמבלר ומה שזו מתחבטת
(הפעם הראשונה לא קיימת בתוכנה לרצות בלילה, כי הקוד בעל
אולי רץ מחדש שלנו) מה שכן אפשר לחשוף הוא אסמבלר וקוד
אולי הדיבג של מ' לפתור את התקלות.

יפתח הקורס זימון על הנצחה הפוכה וההשלכה של זה.
יש גם שאלות (obfuscators) שאלות על
הדיוק והפולציה. קצב אנשים אחר זה כמות של הקוד
לפניהם גם האוסמפיה. זה (פול) ייתר בעולם כמו
java ! - C# שיש יחסיה קוד לכתוב מהבדלים
הקוד עצמו. מובן מי לכתוב את ה-obfuscator
כדי לקוד לבוא לא מוכנים באיזה.

ע"י (נ"י) מלפנים

כולנו מכירים את ה Help>About יש שם כל מיני פירוט
משפטים ואיך צברים להתקין אתמנה קודם.

נשמע תוכנה תוכנה הניסיון מספר את כל היחסים בין האב
ל תוכנה - מה אחת בדיוק לעשה איתך ולעשה אנשים / זמן.

שקולו:

- איך נתקיים את השורה? - אולי מוצל או שזה באינטרנט?
- כמה עושים את הכסף? - משתמשים / פרסומות / שירותים
מקדמים / השכרה המוצר?
- איך מתחרים את המוצר? - לפי זמן / כמה משתמשים?
- תמסודים - נותנים למישהו וירטואלית לעבוד עם המוצר
ומקבלים תמסודים מההכנסות.
- באמצעות - השותפים למכרה ולשליח בעצמיה למכרה
או המוצר.
- קרדיט - האם אנונימי בשמיה מי אחראי על המוצר ומאיזו
כמה?
- האם נותנים את הקוד או לא? האם נותנים אישור לשלוח
או לא? האם יש אישור זהירות?

- code-escrow - מה אם חברה גייסרה דברים הסכם
ששנים אחר כך באים מקום שלישי, נמנעו או משהו
ואז אם החברה נסגרה אז הקוד שיק אמיטלם או
העסקה.

open source - העקרון הבסיסי קוד חופשי -
בהתכנות - וזאת מה שזה הקוד והדגם אפשר לשנות
אותו את מי שרואה שזה open source.
זה מאוד מאוד נפוץ (linux, apache, perl, php ועוד)

העניינים החשובים אלו הם העניינים השונים (הפנים) דואלים.
יש חברה שיש open source אבל הן גם דגש על
דול, אלה ע' מהן תמיכה מסחרית (Redhat, MySQL)

יתרונות

- חופש
- יכולת לשנות/שדרוך
- קל לשדרוך - הקוד פתוח ואם מישהו שגה צריך
- אספקת
- אם תלוי ב- vendor (ארכי)

חסרונות

- אין אחריות/אחריות בעתיד
- אין קצב חופה לגבי תמיכה
- תמיכה פחותה וקניסון מאוד מוגבל או השירות
- משרת הפיתוח (יול) השתנה
- נראה ראוי - אין 'תו' מוכרי

(12)

אתר הרשימה - הנפוצים ביותר GPL ו-GPL
אפשר לשלוח את שירותים שבהם הקוד ב- GPL
שירותים ו-GPL. יש להן תקנה זרימה
אנשים link או לא. אם סתם שירותים מול DLL
זה בסדר. אבל אם לקחת קוד ושילובו ליישום
הקוד שלך אז זה צריך להיות GPL.

Black Duck - תוכנה לעזרה של מאג הקוד
ומזכיר את יש עם שימוש ב- opensource לעולם
לדבר העזרה מילפיון.

פסנט - המוטיבציה (ואם לא מילפיון ממילפיון וזה נכון)
עם העולם להשתמש ובתמורה מקבל מילפיון או מילפיון.
השמו אושם פסנט צריך לעמוד בתנאי תנאים
- הרצון צריך להיות חדש לאיזה וזה יצא קודם
- העליון מתחיל בשירותים שירותים לציבור קיימים
- פסנטים טובים / - עם שנה וזה לא מציור בנפרד (אם
אפשר הם הסכמים)
- עליונות התחלתית נחמה - מאחר צריך בשביל להגיש את
ההצעה. אחרי עזר טובים על זה ומעבדים את זה וזה
יבחר מילפיון לאדם בולרים.

סוף

מבחן - מוצא 12/8
שאלה אחת אחרת (תשובה אחת נחמה קפוק)
שאלה קטנה - 22/7 - לשלוח או מילפיון לציבור (אם)