Deliverable 1 – Process Control Chart

DANILO DELL'ORCO 0300229

Roadmap

- Introduzione
- Progettazione
  - Ticket Jira
  - Commit Git
  - Merging Git w/ Jira
  - Analisi Ticket e Report
  - Process Control Chart
- Risultati
- Analisi dei Risultati
- Conclusioni

Introduzione (1/2)

- L'obiettivo di questa attivita è quello di realizzare un **Process Control Chart** per uno specifico attributo di un processo.

- Il Process Control Chart è uno strumento statistico che permette di monitorare l'evoluzione temporale e la stabilità di un qualsiasi processo di sviluppo
  - Si pone sull'asse X la metrica temporale considerata (#Revisione, Mese, Anno …)
  - Si pone sull'asse Y il valore dell'attributo da monitorare.

- Fissato l'attributo di interesse, si valuta il suo comportamento rispetto all'andamento medio
  - Si definiscono *Upper Limit* e *Lower Limit*
  - Si individuano gli **outlier**, ovvero i valori che escono da tali limiti

Introduzione (2/2)

- L'analisi della stabilità tramite Process Control Chart permette di individuare dei sottoperiodi in cui ricercare eventuali problemi nel processo software.
  - Definire il punto esatto in cui si verifica una anomalia
  - Studiare l'andamento nei periodi immediatamente precedenti/successivi
  - Determinare con maggiore precisione le cause che hanno portato al verificarsi di tali anomalie

- Durante questa attività è stato preso in esame il progetto **apache/daffodil**
  - Come metrica temporale si considerano i *mesi*.
  - Come attributo da monitorare si considera il *numero di bug fixati*.

Progettazione

- Per raccogliere i dati necessari all'analisi del processo è stato sviluppato un programma Java, organizzato in 4 fasi principali:
  1. Ottenimento dei ticket di tipo *fix bug* da **Jira**.
  2. Ottenimento dei *commit* da **GitHub** .
  3. Mapping tra *commit* e *ticket*.
  4. Analisi dei ticket e salvataggio del report su un file *.csv*

- Successivamente il *.csv* con tutti i dati relativi ai ticket viene importato all'interno di un fogliodi lavoro *Excel*, al fine di realizzare il *process control chart*

Progettazione – Jira Ticket

- Si ottengono da **Jira** tutti i *ticket* relativi a *bug fixati*, utilizzando le *REST API* messe a disposizione da Jirastesso.

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22" + projName
+ "%22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR"
+ "%22status%22=%22resolved%22)AND%22resolution%22=%22fixed%22&fields=key,resolutiondate,versions,created&startAt="
+ i.toString() + "&maxResults=" + j.toString();
```

- Per ogni *ticket* trovato viene istanziato un oggetto *JiraTicket*, identificato dai campi *key* e *date*

```
for (; i < total && i < j; i++) {
    String date = issues.getJSONObject(i % 1000).getJSONObject("fields").get("resolutiondate").toString();
    String key = issues.getJSONObject(i % 1000).get("key").toString();
    JiraTicket entry = new JiraTicket(key, date);
    fixedBugs.add(entry);
}
```

- Viene ritornato l'array contenente tutti i *JiraTicket* trovati

```
return fixedBugs;
```

Progettazione – Git Commit

- Si ottiene da **Github** la lista dei *commit*, utilizzando la libreria *JGit*.
  - Si apre in locale la repository tramite il comando *clone()* o *checkout()*

```
if (!Files.exists(Paths.get(repoDir))) {
    this.git = Git.cloneRepository()
    .setURI(gitUrl + projectName + ".git")
    .setDirectory(new File(repoDir))
    .call();
} else {
    try (Git gitRepo = Git.open(new File( repoDir + "/.git"))){
        this.git = gitRepo;
        gitRepo.checkout().setName(this.getDefaultBranch()).call();
        gitRepo.pull().call();
    }
}
```

  - Si ottiene la lista dei commit effettuati sul branch master tramite il comando *log()*. Per ogni commit individuato si istanzia un oggetto *GitCommit* e viene ritornata la lista di tutti i commit trovati.

```
this.commits = new ArrayList<>();
    Iterable<RevCommit> commitsLog = null;

    this.git.checkout().setName(this.getDefaultBranch()).call();
    commitsLog = git.log().call();

    for (RevCommit commit : commitsLog) {
        this.commits.add(new GitCommit(commit.getId(), new Date(commit.getCommitTime() * 1000L),
commit.getFullMessage()));
    }
    return this.commits;
```

Progettazione – Merging Git w/ Jira

- Si effettua il mapping tra i Ticket ed i Commit associati a quei ticket.
  - Si cerca il commit che contiene nel messaggio l'ID del Ticket
  - Si rendono coerenti le informazioni ottenute dalle due piattaforme, impostando sul *JiraTicket* la stessa data ottenuta dal *GitCommit*.

```
public static void bindJiraToGit(List<JiraTicket> tickets) {
    GitAPI git = new GitAPI(GIT_PROJECT_NAME, "output/");
    git.init();
    List<GitCommit> commits = git.getCommits();

    for (GitCommit c : commits) {
        for (JiraTicket t : tickets) {
            if (c.getComment().contains(t.getKey()) && (t.getDate().compareTo(c.getDate()) < 0)) {
                t.setDate(c.getDate());
            }
        }
    }
}
```

Progettazione – Analisi Ticket e Report

- Per ottenere il numero di *Bug Fixed* per ogni mese, si ottiene innanzitutto una lista contenente tutti i mesi sui quali ho dei JiraTicket validi.

```
List<String> months = new ArrayList<>();
for (JiraTicket t : tickets) {
    if (!months.contains(t.getMonth())) {
    months.add(t.getMonth());
    }
}
```

- Per ogni mese individuato, conto quante volte questo appare all'interno della lista di *JiraTicket*

```
for (String m : months) {
    int count = 0;
    for (JiraTicket t : fixedBugs) {
        if (t.getMonth().equals(m)) {
        count++;
    }
}
```

- Il report generato viene scritto su un file DAFFODIL_DATA.csv

Month	NumBugFixed
2012-03	3
2012-06	2
2012-07	6
2012-08	12

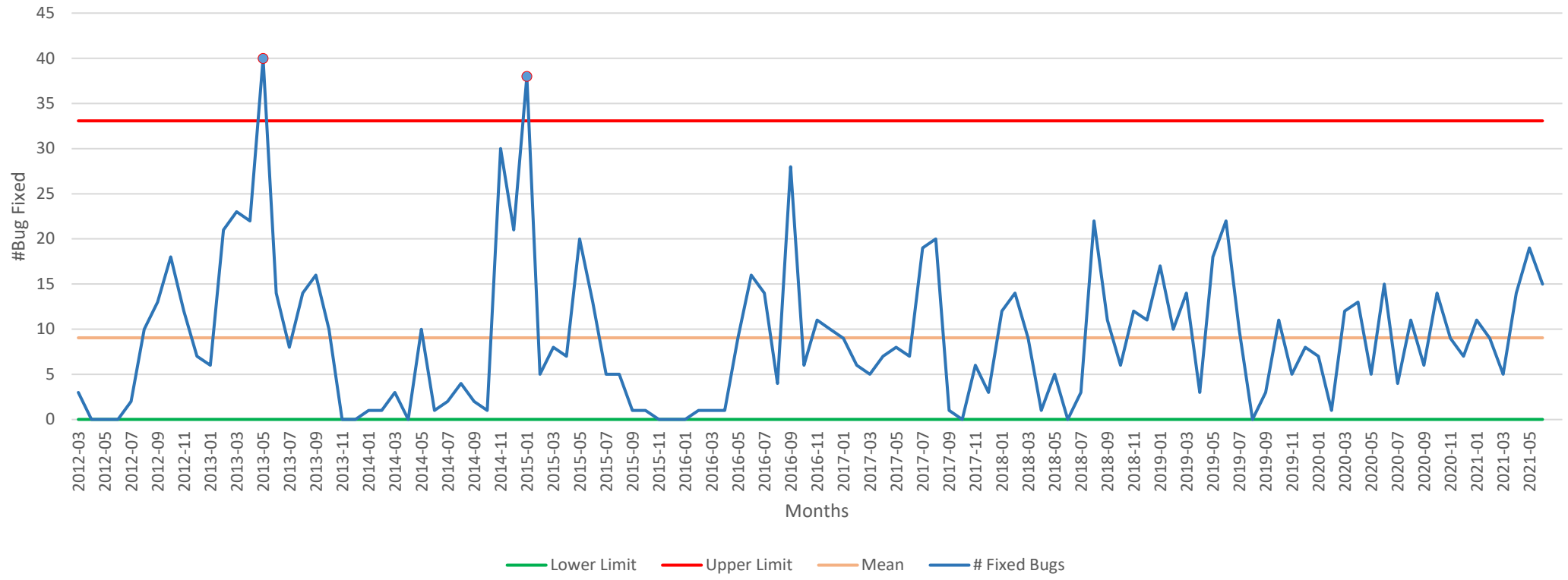
Progettazione - Process Control Chart

- I dati riportati dal file DAFFODIL_DATA.csv sono stati importati all'interno di un foglio *Excel*, in modo da poter calcolare tutti i valori necessari alla realizzazione del Process Control Chart.
  - Numero medio di *bug fixed* tra tutti i mesi considerati $\boldsymbol{E}$
  - Deviazione standard del campione $\boldsymbol{\sigma}$
  - Upper limit: $\boldsymbol{UL = E + 3\sigma}$
  - Lower limit: $\boldsymbol{LL = max\{0; E - 3\sigma\}}$

- A partire da questi valori, il process control chart è stato realizzato come un grafico a linee
  - Asse X: *Mesi*
  - Asse Y: *Numero di bug fix*.

- I valori di *Upper Limit*, *Lower Limit* e *Media* sono rappresentati sul grafico tramite una serie lineare.
  - Permette di visualizzare graficamente quando un punto si discosta dall'andamento medio

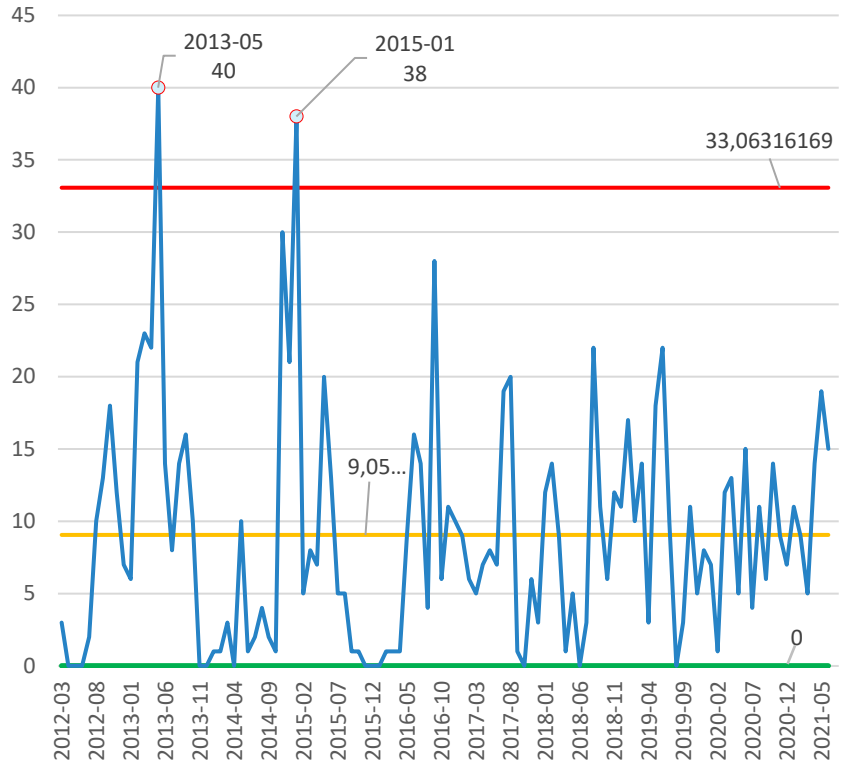
Process Control Chart



Process Control Chart sul numero di Bug Fixed nei vari mesi del progetto apache/daffodil

Risultati

- Periodo di osservazione di 112 mesi (2012-03/2021-06)
- 1014 Ticket di tipo Fixed Bug
- 23 Ticket con informazioni non conformi al commit associato che sono stati corretti.
- Il Process Control Chart essere complessivamente stabile
  - Solo per 2 punti su 112 totali il numero di bug fixed nel mese oltrepassa i limiti di stabilità $[LL; UL]$
- Maggio 2013
  - Numero di Bug Fixed 40, superiore di 7 unità rispetto all'Upper Limit considerato
- Gennaio 2015
  - Numero di Bug Fixed 38, superiore di 5 unità rispetto all'Upper Limit considerato



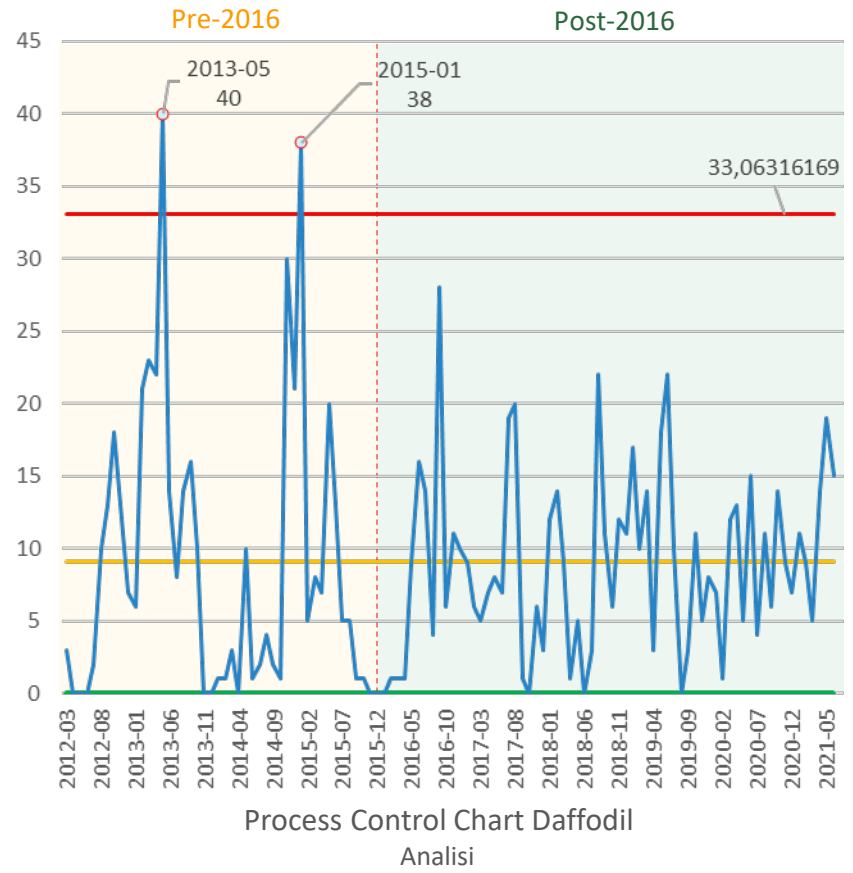
Process Control Chart Daffodil
Outliers Individuati

Analisi dei Risultati

- Analizzando i risultati forniti dal Process Control Chart si può osservare come sia possibile esaminare il processo di sviluppo suddividendolo in *due sottoperiodi*.

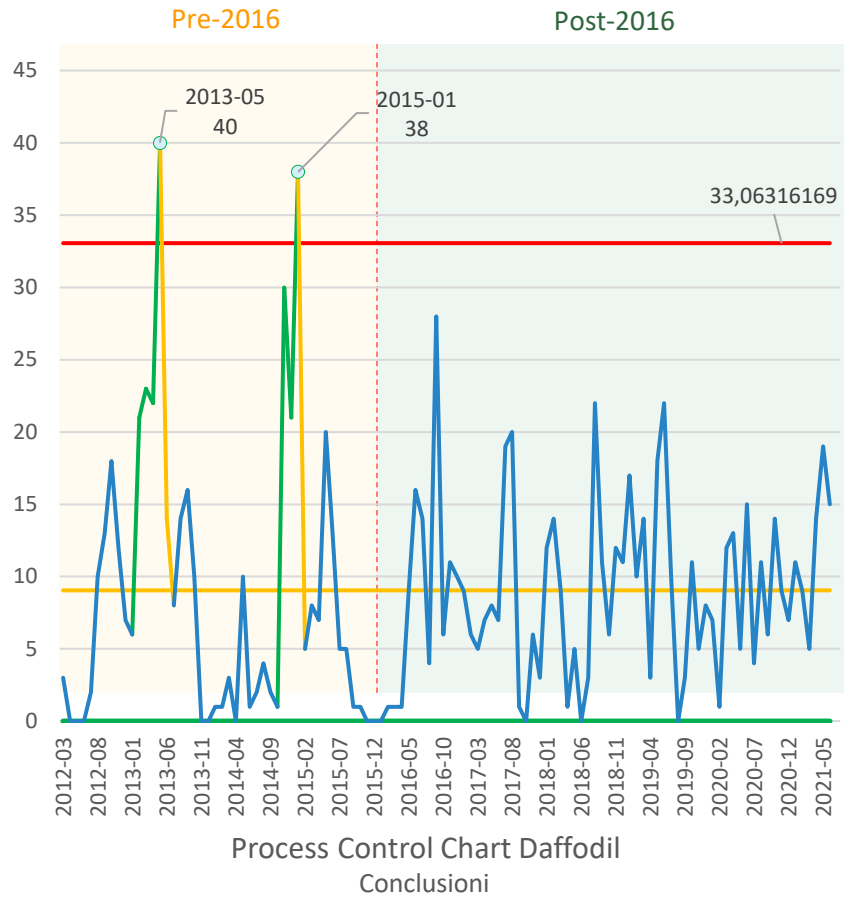
- Prima del 2016
  - Sono presenti gli unici due punti anomali individuati
  - L'andamento generale si discosta molto dalla media

- Dopo il 2016
  - Processo di sviluppo più stabile, nessun punto esce dai limiti di controllo
  - Le oscillazioni della curva risultano molto più vicine alla media rispetto a quelle pre-2016



Conclusioni

- L'andamento per il periodo pre-2016 mostra una *rapida crescita* fino al raggiungimento del punto anomalo, seguita da una *decrescita immediata* nell'intorno del valore medio.
- Questo evidenzia come nel processo di sviluppo *pre-2016* si siano alternate fasi di *debugging* a fasi di *developement*
  - Spostamento periodico dell'*effort* verso le attività di debugging
  - In questi periodi venivano effettuate più operazioni di correzione dei bug
- L'andamento invece per il periodo *post-2016* indica come molto probabilmente sia stato utilizzato un approccio più orientato al *Continous Integration* & *Continous Developement*
  - Si ha un'attività di debugging che segue di pari passo quelle di testing e developement
  - Il numero di bug fixati risulta quindi più costante nel tempo



Links

- Repository GitHub: https://github.com/danilo-dellorco/deliverable1
- Travis CI: https://travis-ci.com/github/danilo-dellorco/deliverable1
- SonarCloud: https://sonarcloud.io/dashboard?id=danilo-dellorco_deliverable1