DEVELOPER-CENTRIC SOFTWARE ASSESSMENT

Philip Makedonski
Jens Grabowski
SOFTWARE ASSESSMENT

• Software Assessment
  • “the process of posing specific questions about the software system under study and carrying out specialized analyses to answer these questions” (Nierstrasz, 2012)

• Agile Software Assessment
  • “a meta-tooling infrastructure and environment that allows rapid and cheap development of custom lightweight tools to support software assessment and program understanding” (Nierstrasz, 2012)
ARTIFACT-CENTRIC

- Metrics
- Clones
- Dependencies
- Domain
- ...

M1: Complexity

M2: Size

void checkSize(){
    if (c <= size){
        malloc(size);
        if (UNLIKELY){
            throw e;
        }
        bufEnd = c;
    }
    return iobuf;
}
CHANGE-CENTRIC

Revision N-1 → Revision N → Revision N+1

Churn Density

... Age Frequency

... Complexity +2
Size +5
Fix +1
DEVELOPER-CENTRIC

- Behavior
- Experience
- Interactions
- Ownership
- ...

Context → Change
DECENT
Developer-Centric Software Assessment
ABSTRACT

Modelling and understanding bugs has been the focus of much of the Software Engineering research today. However, organizations are interested in more than just bugs. In particular, they are more concerned about managing risk, i.e., the likelihood that a code or design change will cause a negative impact on their products and processes, regardless of whether or not it introduces a bug. In this paper, we conduct a year-long study involving more than 450 developers of a large enterprise, spanning more than 60 teams, to better understand risky changes, i.e., changes for which developers believe that additional attention is needed in the form of careful code or design reviewing and/or more testing. Our findings show that different developers and different teams have their own criteria for determining risky changes. Using factors extracted from the changes and the history of the files modified by the changes, we are able to accurately identify risky changes with a recall of more than 67%, and a precision improvement of 87% (using developer specific models) and 37% (using team specific models), over a random model. We find that the number of lines and chunks of code added by the change, the bugginess of the files being changed, the number of bug reports linked to a change and the developer experience are the best indicators of change risk. In addition, we find that when a change has many related changes, the reliability of developers in marking risky changes is negatively affected. Our findings and models are being used today in practice to manage the risk of software projects.

Categories and Subject Descriptors

D.2.8 [Computing Methodologies]: Artificial Intelligence—Expert Systems; Knowledge-Based Systems; Knowledge Representation; Machine Learning; Natural Language Processing; Problem Solving, Control Methods; Software Engineering—Change Management; Process Management. This is especially true for software projects. For example, a recent initiative on managing technical debt aims at studying how compromises that developers make today will affect their software in the future [30]. Risky changes could introduce bugs but they could also delay the release of projects, and/or negatively impact customer satisfaction. For example, changes that might have a widespread impact on the code (e.g., switching threading models) or on the user (e.g., making the software application autosave every 1 min instead of 30 seconds, for optimization reasons) are considered risky, regardless of whether or not they introduce bugs. The risk is caused by the uncertainty introduced by the changes.

A risky change ideally requires additional attention through careful code or design reviewing and possibly more testing. Our findings show that different developers and different teams have their own criteria for determining risky changes. Using factors extracted from the changes and the history of the files modified by the changes, we are able to accurately identify risky changes with a recall of more than 67%, and a precision improvement of 87% (using developer specific models) and 37% (using team specific models), over a random model. We find that the number of lines and chunks of code added by the change, the bugginess of the files being changed, the number of bug reports linked to a change and the developer experience are the best indicators of change risk. In addition, we find that when a change has many related changes, the reliability of developers in marking risky changes is negatively affected. Our findings and models are being used today in practice to manage the risk of software projects.

A survey of 600 firms showed that 35% of them had at least one runaway project [6]. Another study showed that, industry-wide, only 16.2% of software projects are on time and on budget. Of the rest, 52.7% are delivered with reduced functionality and 31.1% are cancelled before completion. The main reason for this large amount of late projects is the lack of proper software risk management (i.e., activities used to manage the possibility of harm or loss) [6, 10].

Due to the importance of risk management in the success of software projects, researchers and industry have become more interested and active in the area of software risk management [13, 23]. One line of work that received a large amount of attention recently is software bug prediction, where code and/or historical metrics are used to predict where bugs might appear in the future (e.g., [26, 35]). In fact a recent literature review showed that in the past ten years more than 200 papers were published on defect prediction alone [17].

However, organizations are interested in effective management of risk in general, which covers more than just bugs. For example, a recent initiative on managing technical debt aims at studying how compromises that developers make today will affect their software in the future [30]. Risky changes could introduce bugs but they could also delay the release of projects, and/or negatively impact customer satisfaction. For example, changes that might have a widespread impact on the code (e.g., switching threading models) or on the user (e.g., making the software application autosave every 1 min instead of 30 seconds, for optimization reasons) are considered risky, regardless of whether or not they introduce bugs. The risk is caused by the uncertainty introduced by the changes.

A risky change ideally requires additional attention through careful code or design reviewing and possibly more testing. Our findings show that different developers and different teams have their own criteria for determining risky changes. Using factors extracted from the changes and the history of the files modified by the changes, we are able to accurately identify risky changes with a recall of more than 67%, and a precision improvement of 87% (using developer specific models) and 37% (using team specific models), over a random model. We find that the number of lines and chunks of code added by the change, the bugginess of the files being changed, the number of bug reports linked to a change and the developer experience are the best indicators of change risk. In addition, we find that when a change has many related changes, the reliability of developers in marking risky changes is negatively affected. Our findings and models are being used today in practice to manage the risk of software projects.
ABSTRACT

Modelling and understanding bugs has been the focus of much of the Software Engineering research today. However, organizations are interested in more than just bugs. In particular, they are more concerned about managing risk, i.e., the likelihood that a code or design change will have a negative impact on their products and processes, regardless of whether or not it introduces a bug. In this paper, we conduct a year-long study involving more than 450 developers of a large enterprise, spanning more than 60 teams, to better understand risky changes, i.e., changes for which developers believe that additional attention is needed in the form of careful code or design reviewing and/or more testing. Our findings show that different developers and different teams have their own criteria for determining risky changes. Using factors extracted from the changes and the history of the files modified by the changes, we are able to accurately identify risky changes with a recall of more than 67%, and a precision improvement of 87% (using developer specific models) and 37% (using team specific models), over a random model. We find that the number of lines and chunks of code added by the change, the bugginess of the files being changed, the number of bug reports linked to a change and the developer experience are the best indicators of change risk. In addition, we find that when a change has many related changes, the reliability of developers in marking risky changes is negatively affected. Our findings and models are being used today in practice to manage the risk of software projects.

Categories and Subject Descriptors

- Year-long study with 450+ developers from 60+ teams at RIM
- Focus on risky rather than buggy changes
- Different developers and teams have their own criteria
- 23 factors across 6 dimensions
DECENT META-MODEL
Evidence-based Software Process Recovery:
A Post-doctoral View

Abram Hindle

Department of Computer Science
University of California, Davis
Davis, CA
ah@softwareprocess.es

Abstract—Software development processes are often viewed as a panacea for software quality: prescribe a process and a quality project will emerge. Unfortunately this has not been the case, as practitioners are prone to push against processes that they do not perceive as helpful, often much to the dismay of stakeholders such as their managers. Yet practitioners still tend to follow some sort of software development processes regardless of the prescribed processes. Thus if a team wants to recover the software development processes of a project or if team is trying to achieve a certification such as ISO9000 or CMM, the team will be tasked with describing their development processes. Previous research has tended to focus on modifying existing projects in order to extract process related information. In contrast, our approach of software process recovery attempts to analyze software artifacts extracted from software repositories in order to infer the underlying software development processes visible within these software repositories.

I. INTRODUCTION

If one approaches a developer and asks them what software development process are they following, how will they answer? Will they respond with the process that their management has demanded regardless of its appropriateness for the project? Will they reference the process that they believe their peers or community have dictated to them? Will they respond with the process that their manager or community has dictated to them? Will they respond with the process that their manager or community has dictated to them? If the answer to these questions is not any of these questions we require knowledge of the software and the process they were supposed to follow? To answer these questions from another source: artifacts of software development processes that they personally follow but others do not? Will they mention their project based on routines they like to follow. These actions

Authors can utilize software process recovery for numerous purposes:

- Managers can utilize software process recovery for numerous purposes: push developers to produce software of a consistent quality. As stakeholders such as their managers. Yet practitioners still tend to follow some sort of software development processes regardless of the prescribed processes. Thus if a team wants to recover the software development processes of a project or if team is trying to achieve a certification such as ISO9000 or CMM, the team will be tasked with describing their development processes. Previous research has tended to focus on modifying existing projects in order to extract process related information. In contrast, our approach of software process recovery attempts to analyze software artifacts extracted from software repositories in order to infer the underlying software development processes visible within these software repositories.

A. Stakeholder motivations

Recovering software development processes from existing projects is useful to many stakeholders who care about the system and also have some stake in the processes that govern its development.

Developers care about process in the sense that they are forced to follow it but also at the same time are forced to rely upon it. If developers act inconsistently, they create confusion based on the assumptions that other developers are making about development. Developers are surprised by behaviour that does not fit within an accepted process. Many developers would assume they do not follow any process at all. This is not the case as many developers, we would claim, follow a natural process based on routines they like to follow. These actions might result in greater software quality and thus motivate these
Abstract

—Software development processes are often viewed as a panacea for software quality: prescribe a process and a quality project will emerge. Unfortunately this has not been the case, as practitioners are prone to push against processes that they do not perceive as helpful, often much to the dismay of stakeholders such as their managers. Yet practitioners still tend to follow some sort of software development processes regardless of the prescribed processes. Thus if a team wants to recover the software development processes of a project or if a team is trying to achieve a certification such as ISO9000 or CMM, the team will be tasked with describing their development processes. Previous research has tended to focus on modifying existing projects in order to extract process related information. In contrast, our approach of software process recovery attempts to analyze software artifacts extracted from software repositories in order to infer the underlying software development processes visible within these software repositories.

I. INTRODUCTION

If one approaches a developer and asks them what software development process are they following, how will they answer? Will they respond with the processes that their managers have prescribed or might they answer that they do not follow any processes at all? This is not the case as many developers, we would claim, follow a natural process based on routines they like to follow. These actions might result in greater software quality and thus motivate these developers to perform better, but will not be recognized as processes. The problem is how do we deal with this situation? How do we deal with the gaps between what we can infer from evidence left behind and the reality of development?
OVERVIEW

DEVELOPER-CENTRIC

DECENT INFRASTRUCTURE

DECENT PREDICTION
DECENT META-MODEL
<table>
<thead>
<tr>
<th>Model</th>
<th>Summary</th>
<th>Context(cc)</th>
<th>Delta(cc)</th>
<th>Context(loc)</th>
<th>Delta(loc)</th>
<th>Context(com)</th>
<th>Delta(com)</th>
<th>Hunked Addec Removec Own changes (past)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>lueck (10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matgic78 (22)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>megabigbug (15)</td>
<td>575</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**settings.cpp**

<table>
<thead>
<tr>
<th>Location</th>
<th>0:0 =&gt; 53:53</th>
<th>Difference</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>3</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>0:0 =&gt; 112:115</td>
<td>Difference</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Affected Functions**

- Private(SettingsDialog *)

<table>
<thead>
<tr>
<th>Hits:</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>34</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
</table>

**webview.cpp**

<table>
<thead>
<tr>
<th>Location</th>
<th>125:125 =&gt; 125:125</th>
<th>Difference</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>24</th>
<th>59</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>0:0 =&gt; 127:127</td>
<td>Difference</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>0:0 =&gt; 130:131</td>
<td>Difference</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Nodes**

- Artifact
- ArtifactType
- Agent
- Attribute
- Value
- Activity
- ActivityValue

Different context menus:
- Link actions
- Actions for a text selected in field
- Actions for a field
- Actions for text selected in a page
- Actions for a page

--- a/src/webview.cpp
+++ b/src/webview.cpp
<table>
<thead>
<tr>
<th>Model</th>
<th>Summary</th>
<th>Context(cc)</th>
<th>Delta(cc)</th>
<th>Context(loc)</th>
<th>Delta(loc)</th>
<th>Context(com)</th>
<th>Delta(com)</th>
<th>Hunks(Adddel Retemo)</th>
<th>Own changes (past)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1720</td>
<td>Files: 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0</td>
<td>75</td>
<td>-1</td>
<td>32</td>
<td>-1</td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>37</td>
<td>0</td>
<td>237</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td></td>
<td></td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>3</td>
<td>80</td>
<td>4</td>
<td>41</td>
<td>4</td>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1721</td>
<td>Files: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1724</td>
<td>Files: 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>96</td>
<td>0</td>
<td>460</td>
<td>0</td>
<td>96</td>
<td>9</td>
<td></td>
<td></td>
<td>155</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>3</td>
<td>84</td>
<td>18</td>
<td>45</td>
<td>6</td>
<td></td>
<td></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>31</td>
</tr>
</tbody>
</table>

- Different context menus:
  - Link actions
  - Actions for a text selected in field
  - Actions for a field
  - Actions for text selected in a page
  - Actions for a page

--- a/src/webview.cpp
+++ b/src/webview.cpp
<table>
<thead>
<tr>
<th>Total changes (past)</th>
<th>Ra Ownership</th>
<th>Own fragments</th>
<th>Total fragments</th>
<th>Own fix commits</th>
<th>Total fix commits</th>
<th>Own bug commits</th>
<th>Total bug commits</th>
<th>Bug Confidence Weight</th>
<th>Fix</th>
<th>Bug</th>
<th>Clone</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.1666666666666666</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Issue Weight</td>
<td>Fix</td>
<td>Bug</td>
<td>Clone</td>
<td>Last change own</td>
<td>Distance to last own change</td>
<td>Distance to last</td>
<td>Average distance between own changes</td>
<td>Churn (added/loc)</td>
<td>Density (hunks/loc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-----------------</td>
<td>----------------------------</td>
<td>-----------------</td>
<td>---------------------------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>false</td>
<td>-1</td>
<td>53</td>
<td>-1</td>
<td>0</td>
<td>0.0</td>
<td>0.01333333333333334</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66666666</td>
<td>1</td>
<td>1</td>
<td>false</td>
<td>325</td>
<td>0</td>
<td>325</td>
<td>3</td>
<td>0.004219409282700</td>
<td>0.004219409282700422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>false</td>
<td>240</td>
<td>4</td>
<td>103</td>
<td>8</td>
<td>0.4375</td>
<td>0.0875</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>false</td>
<td>240</td>
<td>4</td>
<td>103</td>
<td>10</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>false</td>
<td>124</td>
<td>8</td>
<td>107</td>
<td>3</td>
<td>0.021739130434782</td>
<td>0.004347826086956522</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>true</td>
<td>4</td>
<td>4</td>
<td>138</td>
<td>14</td>
<td>0.511904761904761</td>
<td>0.10714285714285714</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>true</td>
<td>4</td>
<td>4</td>
<td>110</td>
<td>17</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DECENT PREDICTION

• Developers as first class citizens in software assessment
• Developer-specific factors contributing to risk
• Personalized and contextualized feedback
• Improvement of software assessment and quality
FURTHER APPLICATIONS
DEVELOPER-CENTRIC SOFTWARE ASSESSMENT

Philip Makedonski
makedonski@informatik.uni-goettingen.de

Jens Grabowski
grabowski@informatik.uni-goettingen.de

Software Engineering for Distributed Systems
Goldschmidtstr. 7
37077 Göttingen
Germany

www.swe.informatik.uni-goettingen.de
LITERATURE


FURTHER LITERATURE


FURTHER LITERATURE


history menu fix
--- a/src/modelmenu.cpp
+++ b/src/modelmenu.cpp
@@ -147,17 +147,19 @@