Data Mining Behavioral Transitions in Open Source Repositories

William N. Robinson  
Computer Information Systems Dept.  
Georgia State University  
wrobinson@gsu.edu

Tianjie Deng  
Computer Information Systems Dept.  
Georgia State University  
tdeng1@student.gsu.edu

Abstract

Open-source repository data can be automatically mined using sequence mining methods to provide high-level feedback on project status. GitHub.com projects are acquired, sequence-mined, clustered, and regressed to analyze project characteristics. Such results can be presented to project managers, as part of a display generated by an automated monitoring system. Such monitoring systems provide high-level feedback in real-time. This project is a preliminary step in a larger research project aimed at understanding and monitoring FLOSS projects using this process modeling approach.

1 Introduction

We introduce the kind of problem we aim to address with an illustrative scenario. Consider an open-source software development team working on their sixth release of Open Photo Booth v6. During the past two years, the team has gone through changes: new developers joined while old members dropped out, hundreds of bugs have been reported, the project’s vision has gone through several shifts, thousands of comments were made, and download numbers of the previous releases have varied. Release v4 was a huge success; however, the last release, v5, did not receive much attention. Now, before their sixth release, they are wondering, what is the likelihood that this version will be a success? Which development behaviors are effective at increasing downloads and project followers?

This illustrates an interesting issue about open source projects: due to the dynamic structure of the teams, and the unique ways of collaborating, projects can evolve substantially. Thus, analyzing a static snapshot does not allow us to understand an evolving trajectory. Evolutionary qualities are important, and may be a good guide to forecasting future qualities, such as project downloads and project followers.

Raising alerts on a project dashboard when a team appears to be losing its effectiveness is a goal of the work described herein. We rely on the team’s repository to observe their activities. Our data mining techniques provide for automated analysis, which can be incorporated into the dashboards common to Agile development.

This approach to team monitoring is as follows:

1. Developers use their standard development tools (e.g., Eclipse, GitHub, etc.) to develop software.
2. The Monitor continually monitors events logged on the team’s source code repository (e.g., GitHub).
3. The Monitor builds data-mined models, representing developer behaviors, from the repository events.
4. Analysis of the mined models is presented on a project status dashboard.
5. Managers and developers review the analysis as a guide to how their team is performing.

We describe our automation for steps 2 – 3 (model building). Much research remains to extend our models, by building on the data mining results and preliminary analysis. As a secondary consideration, we will address automation of steps 4 – 5 (model presentation and interpretation) in the future.

1.1 FLOSS Success

Our analysis reviews open source projects from GitHub.com. We use the common term, FLOSS, to reference them.

This primary work takes a takes data centric, abduction approach to exploring project characteristics. Eventually, we aim to find causal patterns between developer behavior and project outcomes and link that analysis to theoretical processes and constructs, such as project success. Toward that end, we are working on building software automation (data mining) for model building. Consequently, this paper is more about model building automation. Our outcome analysis herein is limited to simple project characteristics, such as downloads and followers, which are observable performance constructs that characterize aspects of success.

1.2 Recognizing Behaviors and Change

We aim to identify and analyze common sequences of actions by developers. Developers commit code, raise issues, and make comments. We would like to know, for example, if the pattern sequence (Issue x, Comment x, Commit x) occurs frequently in the stream of developer activities. We suspect that successful projects include this pattern more frequently than simple Commit x—that is, having no previously specified issue or comment for a code commit.

To uncover sequential patterns, sequential data mining techniques are commonly applied. We have applied such techniques to event streams generated by developers. When applied directly, the techniques
generate long lists of varied, low-level action sequences, which are not easily interpreted. To address this issue, we specify work constructs, which are recognized by a rule-base system. More generally, we are working towards a theory of FLOSS development by incrementally improving partial operational models, which recognize theoretical constructs in FLOSS event streams.

Our analysis proceeds as follows:
1. Specify recognition rules, derived from a theory of distributed cognition, for our theoretical constructs, such as an issue-based work unit.
2. Apply the rules to the repository event data, to recognize the theoretical constructs.
3. Data mine the recognized constructs, using sequence-mining techniques.
4. Apply model-differencing techniques, to recognize changes in behaviors over time.
This approach allows us to analyze developer behavior (as action sequences) and their changes over time.

1.3 Building a Process Theory
This work represents a case study in our efforts toward supporting the construction of process theories. A “process model explains development in terms of the order in which things occur and the stage in the process at which they occur.”[1] Abbott, for example, illustrates how time ordered events affect the lifecycle of individuals, which supports theorizing about process steps, and cause-effect relationships[2].

A process theory can be derived from data by analyzing event sequences. An event can be viewed as the change in state of some variable values[3]. An event sequence can be characterized by common metrics, such as length, entropy, subsequences, pattern frequency, and similarity to other sequences. Through abduction, a researcher can infer common constructs, higher level concepts, and eventually relate these terms to a theory that explains relationships among concepts within a set of boundary conditions[1]. Figure 1 illustrates these elements of a process theory (adapted from [1]).

The study presented herein illustrates this approach to abducting theory elements from sequence data. We have in mind some general theories which appear applicable to our analysis of development, such as distributed cognition[4]. To bridge the gap between our hypothesized theoretical elements and our data, we apply sequence data mining. Our minor contribution to building process theories is demonstrating how much of the abduction process can be automated.

1.4 Article Overview
Next, this article introduces reference theories that have guided our project Monitor and data mining. Section 3 presents a case study of sequence mining GitHub projects. The final sections present a discussion and conclusions.

2 Theoretical Lens and Reference Techniques
2.1 Distributed Cognition
As we begin this research project, to understand FLOSS development, we assume that information systems and computer science theories are relevant. As theories, they provide a conceptual lens to frame our analysis. We aim to make elements of the theories operational in the sense that we can establish a direct link between the data and a theoretical relationship. For us, an operational model establishes this direct, operational link between data and theoretical constructs.

Distributed cognition provides a theoretical lens to understand software development. As developed by Hutchins, it posits the perspective that the boundary of cognitive processes go beyond individuals to socio-technical systems[4]. It presents how cognitive processes are distributed socially, structurally, and temporally when the members of a distributed team collaborate on information processing tasks.

By conceptualizing cognition as “the propagation of representational state across representational media” ([5]: p. 118), distributed cognition expands the unit of cognitive analysis from that of the individual to that of the entire team attending to a specific task. With this shift in perspective on cognition, the theory asserts[4]: (1) thought processes are distributed among members of social groups, (2) cognition employs both internal and external structures, and (3) cognitive processes are distributed over time.

Most large software development efforts are team efforts[6]. Teams bring together individuals from a variety of technical and functional domains to address complex design challenges. For example, the cognitive task of arriving at a stable requirements set resides in the holistic process of cognition that enables requirements to emerge as a quality of the social system [7, 8].

In practical terms, this means that you can interpret the team member activities in distributive cognition terms. For example, team members may rely on the source-code commit log to share progress information about the project. When members fail to submit comments, then this form of distributed information sharing breaks down. Distributed cognition indicates, generally, the kinds of communications

![Figure 1 Illustration of a Theory.](image-url)
and breakdowns they commonly occur.

Other theories provide a background for the development of variables, constructs, and concepts needed to understand information processes in software development. For example, the recent theory of collaboration through open superposition suggests specific ways in which members collaborate in FLOSS projects[9]. Galbraith’s classic information processing view suggests the need to examine structural mechanisms, such as information buffers (e.g., a repository), to reduce information uncertainty[10].

In general, we are trying to demonstrate, adapt, and extend software development theories by encoding them in operational process models and using them to conduct exploratory analysis. Herein, we apply this approach to the analysis of GitHub projects. We introduce how we use a distributed cognition concept to encode some data, in section 3.4.

### 2.2 Sequence Stream-Mining

Developers have many interactions, directly or indirectly, through their tools. Some co-located developers will go to their computers to meet, thereby ensuring a record on their meeting (as well as providing access to development records).

Many FLOSS interactions are logged as event histories. For example, the history of source code changes is maintained by source control systems (e.g., CVS, Git). As each change is committed to the source repository, the new code and comments are recorded as a change event. Similarly, edits within a code editor (e.g., Eclipse), messages within a chat session, forum comments, feature requests, FAQ edits, etc., are all event sequences. Such sequences can be mined for patterns.

Sequence data mining concerns analysis of events in sequence. The event data are often nominal-valued or symbolic and the goal is to discover variables and their correlations[11, 12]. This contrasts to the well-studied domain of time series analysis, which considers real or complex-valued time series of known parameters using methods such as autoregressive integrated moving average (ARIMA) modeling. Sequence mining techniques address: (1) prediction, (2) classification, (3) clustering, (4) search and retrieval, and (5) pattern discovery.

We apply sequence mining in the context of stream mining. Stream mining aims to find interesting relationships over a sequence of data segments [13-15]. Stream mining algorithms can vary substantially from their more traditional forms: data may be analyzed incrementally rather than as a batch, old data may be discounted or removed in favor of newer data, the created model may be an approximation when compared to its traditional form[16]. A variety of techniques can be applied to stream data [17-20]—much of the work is focused on the efficiency of incrementally updating the model [21].

Stream mining can detect changes in the data-stream. Two types of algorithms are common: (1) distribution detection, which watches for changes in the data distributions, and (2) burst detection, which watches for sudden large and unusual changes in a data-stream. Distribution detection algorithms have two common forms: (a) data from two windows (current and reference) are compared using some distance measure, (b) a predictive model is created from a prior window and then its prediction is compared with the current window—high prediction error indicates a significant change. We apply both distribution detection techniques to discover changes, as well as model differencing, which is presented next.

### 2.3 Model Differencing

Model differencing provides a mean to recognize important changes occurring within an event stream. Consider a stream of repository events divided into data windows, \( w_1, w_2, \ldots, w_\lambda \). Transition identification marks each data window as either normal or transitional; for example, (normal, normal, transitional, normal, normal…). Transitional behavior is historically unusual behavior, according to some measure such as statistical variance. We use the term transitional because the behavior is unusual and transient, and thus interesting from a theoretical perspective, such as cognition or learning theory.

In our approach, a repository stream is divided into data windows. Each window is characterized by a model, \( \lambda \). Consider two models in sequence, \( \lambda_1 \) and \( \lambda_2 \). The software finds the difference of the models to characterize the change: \( \Delta \lambda = (\lambda_2 - \lambda_1) / (t_2 - t_1) \). If the difference \( \Delta \lambda \) is significant, by some measure, then we have found a transition point[22].

In this work, the model types \( \lambda \) vary; we include hidden Markov models (HMMs) for example. Now, because of this automated differencing technique, a monitoring system can quickly identify changes in the models. Thus, some intervention may be applied. In software development, this may be changing the project lead, increasing testing, or releasing the software, for example.

### 2.4 HMM Probabilities

Given data containing sequences, a common task is to find transition probabilities. That is, given an observed event A, what is the probably that the next event observed with be B or C? A hidden Markov model (HMM) can solve this problem by building a probability model from observed event sequences.

A hidden Markov model (HMM) is a stochastic signal model[23]. In our application to repository analysis, the signals are sequences of discrete typed
events (e.g., code commit). A HMM provides algorithms to solve three important problems:

1. \textit{Compute the probability} that an observed sequence, \(O\), is represented by a HMM, \(\lambda\) (using the Forward-Backward Procedure[24]).
2. \textit{Adjust the parameters of a HMM, \(\lambda\), to maximize the fit} to an observed sequence, \(O\) (using the Baum-Welch algorithm[25]).
3. \textit{Compute the optimal HMM state sequence} that best explains an observed sequence, \(O\) (using the Viterbi Algorithm[26]).

We use HMMs to model patterns of sequential events within the stream of FLOSS repository events.

HMM transition identification detects significant changes in modeled events between consecutive windows of event data. HMMs can be used to identify transitions by: (1) \textit{comparing consecutive HMMs} generated from the observation sequences, or (2) \textit{comparing consecutive acceptance probabilities}[22].

Technique 1 compares consecutive HMMs. This a model differencing technique is generally characterized as follows:

\[ \Delta \lambda = \lambda_2 - \lambda_1 \]  

(1)

Here, \(\lambda\) denotes a HMM. To find the distance between two HMMs, we apply the widely used Kullback-Leibler algorithm[27].

Technique 2 compares the acceptance probabilities of the observation sequences using the first HMM. The two techniques produce similar results.

### 2.5 Volatility Models

A variety of models can be applied to the sequential events found within the data windows of repositories. We consider two that measure variance in sequences: turbulence and optimal matching.

Given a sequence, \textit{turbulence} calculates a metric based on the number distinct subsequences within a data window[28]. Turbulence increases with the number distinct subsequences.

Optimal matching (OM) generates edit distances that are the minimal cost, in terms of insertions, deletions and substitutions, for transforming one sequence into another. OM can be applied to a repository data window to derive a measure of variance. Consider a data window with two observed sequences: \(O^1\) and \(O^{+1}\). Optimal matching, OM(\(O^1\), \(O^{+1}\)), is 0 if the two sequences are identical; OM increases with the differences between sequences.

Like HMM models, turbulence and OM can be calculated for each data window, as well as differenced between two consecutive windows. Thus, we can measure \(\Delta\text{HMM}, \Delta\text{Turbulence},\) and \(\Delta\text{OM}\) between consecutive windows of repository data.

### 2.6 Sequence Stream-Mining Tool Support

Project status monitoring is a practical use of sequence stream-mining for developers. For example, a project dashboard can be created showing the effectiveness of a team’s capability. Low achievement of goals could be met by interventions such as increased meetings and increased usage of project plans and summaries.

Our series of monitoring projects has resulted in tools that have increased capability for stream mining and model difference analysis. The case study reported herein is an application of our tools, summarizes the tools. The first column presents the concepts, which are defined in the second column, while the third column presents representative references.

<table>
<thead>
<tr>
<th>#</th>
<th>Concept</th>
<th>Definition</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compliance checking</td>
<td>Identify absent events compared to specified event sequence.</td>
<td>Deviation from sequences implied by plan—see survey: [29]; Deviation from event-stream properties (e.g., sequences, data predicates, etc.])30-34</td>
</tr>
<tr>
<td>2</td>
<td>Aggregate behavior transition identification</td>
<td>Identify a change in event distribution compared to a model of past events.</td>
<td>Decision tree differencing[35]</td>
</tr>
<tr>
<td>3</td>
<td>Aggregate behavior transition diagnosis</td>
<td>Identify the specific events (types and quantities) that have changed from one observation window to the next.</td>
<td>Decision tree differences and enumerate of associated difference metrics[36, 37]</td>
</tr>
<tr>
<td>4</td>
<td>Aggregate sequential behavior transition identification</td>
<td>Identify a change in event-sequence distributions compared to a model of past event sequences.</td>
<td>Simple difference metrics [38]; Markov probability distribution differences[22]</td>
</tr>
<tr>
<td>5</td>
<td>Aggregate sequential behavior transition diagnosis</td>
<td>Identify the specific in event-sequence (types and quantities) that have changed from one observation window to the next.</td>
<td>Markov probability differences enumerate as event-sequence metrics[22]</td>
</tr>
<tr>
<td>6</td>
<td>Integrative sequence mining</td>
<td>Combine prior mining tools with workflow to automate event acquisition to mined model applications.</td>
<td>This paper</td>
</tr>
</tbody>
</table>

Most monitoring tools check for compliance with properties. We call this compliance checking because the properties imply or explicitly specify a sequence of actions that shall occur.

Another approach to monitoring analyzes trends of aggregated software events. Rather than enumerating and checking all properties, this approach focuses on finding (row #2) and diagnosing (row #3) unusual events, assuming that most software events are normal. Where there are many potential acceptable event sequences, this approach can be more efficient in specification and monitoring effort. Conversely, where there a few potential acceptable event sequences, the
3 Data-Mining Approach
We analyzed 103 projects from GitHub, the most popular open-source code repository site. Founded in 2008, GitHub had over 3 million users and over 5 million repositories as of January 2013. After selecting projects, our KNIME workflow generated sequence models and then clustered the projects. We then characterized the clusters and regressed them with summary project measures. Finally, we validated some results by comparing the automated analysis with qualitative information. The results seem to suggest that our sequence mining is helpful in clustering projects and detecting interesting transitions within a project.

3.1 Data Selection
We used the search function of the GitHub web site to enumerate projects varied by popularity:
1. stars:>10000 forks:>1000
2. less than #1 but stars:>5000 forks:>750
3. less than #2 but stars:>1000 forks:>500
We used number of stars and number of forks as proxy for level of popularity. We included an additional group of Java projects to investigate if language plays a role in the development patterns. (Most GitHub projects use scripting languages.) Subsequent processing reduced the final set to 103 projects. Projects were dropped because their data was minimal or erroneous. Many Java projects, for example, maintain their issue database outside of GitHub. Our analysis was limited to projects contained within GitHub.com. The appendix enumerates the projects.

3.2 Data Preparation
A workflow automated the data acquisition and preparation. GitHub.com data was obtained from two sources:
1. GHTorrent provides access to a GitHub database[40]. That MongoDB database is the result of GHTorrent monitoring the GitHub public event timeline.
2. GitHub API provides direct access to the project data by sending JSON over HTTPS.

GHTorrent provided the basis for the data. The GitHub API was used to validate the data, and in some cases provide missing data.

The GitHub data is comprised of a 16 collections, which we combined, through filtering and joining, into a single table. Our data was derived mainly from these collections: issues, issue events, issues comments, pull requests, and pull request comments. The resulting table consists of these fields: Issue number, body, diff hunk, path, position, original position, commit id, original comment id, create at, updated at, event comments, milestone, title, assignee, closed at, state, merged at, head, base, and actor. Each record in the table provides a vector for input into our data mining process.

The table represents a sequence of Git events. Of the 18 Git events, we focused on six, which most closely associated with teamwork:
1. IssuesEvent: An issue is created, closed, or reopened.
2. PushEvent: Code is committed (pushed) to the repository.
3. PullRequestEvent: A user requests that new code be pushed to the repository.
4. IssueCommentEvent: A comment is associated with an issue.
5. CommitCommentEvent: A comment is associated with a commit (PushEvent).
6. PullRequestReviewCommentEvent: A comment is associated with a PullRequest.

Other events, such as watch events, in which users subscribe to a repository to get updates, do not concern teamwork. They are thus excluded from our study. Our prepared table of sequential Git events is further process to represent elements of teamwork.

3.3 Work Constructs
Git events, such as push and commit, represent work; however, the context of the work is missing. For example, it seems that 10 code commits for the same issue is different than 10 code commits, each for a single issue. A rule-based system is applied to the prepared event data to derive a table of abstracted work events.

Work in most GitHub projects begins with an IssueEvent or a PullRequestEvent. Both represent a typical unit of development work, which may be scheduled, opened, closed, reopened, etc. An IssueEvent typically represents a bug or enhancement. It follows a common lifecycle of being opened,
followed by code changes represented by commits, and then an issue close. For example:
IssuesEvent.open, PushEvent, PushEvent, IssuesEvent.close
Of course, other events may intervene (e.g., comment events), as well as the issue may be reopened or never closed.

The PullRequestEvent is similar to the IssueEvent, but the subsequent work events are related to integrating the new code into the project’s code repository.

A rule-based system is applied to recognize event sequences beginning with IssueEvent or a PullRequestEvent. We think about them as mini-workflows, which are initiated in response to a work request (e.g., issue or pull request). However, we use the more neutral term, motif, to indicate recognition of these common sequence patterns.

The rule-based system recognizes two kinds of work motifs in the prepared table of sequential Git events. The basic form is as follows:
1. (IssueEvent | PullRequestEvent). *
2. (Reopen (of #1)). *

As indicated above, a work motif begins with either an IssueEvent or PullRequestEvent, followed by any other Git event that references the initiating event (by number). The motif records the initial event, and all subsequent events (and their attributes). When either an IssueEvent or PullRequestEvent is reopened, it is considered a new instance of the second motif pattern (above). Thus, open and reopen are each considered the beginning of a work motif. Our subsequent analysis (§4) shows common event sequences; however, those are Git event sequences within the context of these work motifs.

These work motifs are derived from the prepared data in support of our theoretical background—tasks of distributed cognition in particular. Thus, we call these derived, abstracted elements work constructs, to be consistent with theorizing process theories [1].

### 3.4 Sequence Feature Construction

Before the work motifs can be sequence mined, they are encoded. Most sequence data-mining algorithms process event sequences, where events are identified as members of a fixed alphabet. An event sequence, for example, could be A-B-A-B-B-C. Few algorithms directly address object sequences, where each sequence member is comprised of an information object. An object sequence, for example, could be:
- [Issue.ID=1,Issue.Author=a1,Issue.State=open]
- [Commit.ID=10,Issue.Author=a1,Commit.ID=1]
- [Issue.ID=1,Issue.Author=a1,Issue.State=close]. Object sequences can be processed by dropping information. For example, just processing object type information: Issue-Commit-Issue. More information can be processed by first encoding the object information into another alphabet; for example, [Issue.ID=1,Issue.Author=a1] becomes I1A1. We applied this transformation concept to the sequence of work motifs.

Given the work-motif sequences for a GitHub project, we used k-means clustering to transform each work motif into one of 50 clusters. (Size 50 clusters retained the information represented by the great variety of sequences, as indicated by cluster metrics, while simplifying the subsequent sequence analysis.)

The work motif is represented by a vector of these attributes:

- IssueEvent | PullRequestEvent (open|reopen), ID, openDate, events, actors(same|different), number of actors, state(open|closed), merged(true/false), duration, number of comments, turbulence. Our rationale for selecting those attributes is: different instantiations of the same work motif might present different levels of collaboration. For example, in an IssueEvent→CommentEvent→PushEvent work motif, the level of collaboration would be different between an instantiation in which the same actor performed these three activities, and an instantiation in which three different actors performed these three activities. Distributed cognition theory, which considers how team members collaborate, seems to be an appropriate lens to interpret these data. Thus, we used concepts from this theory to extract relevant attributes of a work motif. We reviewed the indicators of distributed cognition based on existing literatures [41-43]. From these indicators, we identified relevant attributes in our work motifs.

### 3.5 Sequence Modeling

Given sequences of clusters representing work motifs, our software constructs models of (a) sequence pattern probabilities, (b) entropy within sequences, and (c) changes in the patterns and entropy over data windows.

Sequence pattern frequencies are determined by simply counting pattern occurrence. For example, Figure 2 illustrates the frequency of the top 20 sequential patterns from the bootstrap project. Additionally, a hidden Markov model (HMM) calculates the sequence pattern probabilities. Optimal matching (OM) is applied to pairs of sequences, to determine a matching distance.

Turbulence and OM are applied to sequential data windows of work motifs (themselves sequences). A series of data windows results in a series of values ($Turb_b, OM_1, \ldots, Turb_{b+n}, OM_{1+n}$) indicating kinds of sequence entropy over time.
Change in sequences over time can be calculated for turbulence, OM, and HMMs. The general equation is simply $\Delta \lambda = \lambda_2 - \lambda_1$, where $\lambda$ represents either turbulence, OM, and HMM models. We calculate the models and their differences for each data window for each project. After some preliminary analysis, we choose four weeks for the data window size—it contains sufficient data and represents a common unit of work for open source development methodologies. The subsequent illustrates $\Delta$HMM for a project from each cluster we derived.

### 3.6 Project Clustering

Projects can be clustered by their changes in their sequences, as represented by models of the prior section. Each project has sequences of change in turbulence, OM, and HMMs; that is, $\Delta \lambda$, where the model $\lambda$ is one of sequence turbulence, OM, and HMM. Given the projects, each represented by a sequence of values, we pairwise compare them using optimal matching to generate a distance matrix. We apply hierarchical clustering to the distance matrix to derive our clusters. (The appendix includes the cluster number for each project.)

Table II summarizes the cluster characteristics. Clusters 3 and 4 have relatively more stars, forks, open issues, and have relative short cycles. We reviewed a project from each cluster. From clusters 3 and 4, ember.js and brackets, respectively, are relatively active. Ember.js has a 15 day release cycle, and brackets has a 17 day release cycle—relatively small compared to the average of 97 days among all projects. Ember.js also has a big community of 360 contributors and has very frequent group meetings through Google Hangout. They used various communication channels including discussion forum, blogs, GitHub site and stackoverflow.com. Project information from such sources provides detailed projects information.

<table>
<thead>
<tr>
<th>cluster</th>
<th>forks</th>
<th>stars</th>
<th>daily forks</th>
<th>daily stars</th>
<th>release time(days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1363.6</td>
<td>5952.6</td>
<td>1.4</td>
<td>6.0</td>
<td>141.7</td>
</tr>
<tr>
<td>1</td>
<td>1094.0</td>
<td>5453.7</td>
<td>0.7</td>
<td>3.5</td>
<td>137.2</td>
</tr>
<tr>
<td>2</td>
<td>885.1</td>
<td>5239.4</td>
<td>1.8</td>
<td>10.8</td>
<td>40.6</td>
</tr>
<tr>
<td>3</td>
<td>2432.7</td>
<td>14103.3</td>
<td>2.3</td>
<td>14.2</td>
<td>28.0</td>
</tr>
<tr>
<td>4</td>
<td>2025.9</td>
<td>10131.3</td>
<td>3.6</td>
<td>19.9</td>
<td>8.25</td>
</tr>
<tr>
<td>mean</td>
<td>1560.2</td>
<td>8176.0</td>
<td>2.0</td>
<td>10.9</td>
<td>101.3</td>
</tr>
</tbody>
</table>

The other three clusters (cluster 0, cluster 1 and cluster 2) are similar in terms of popularity—they have relatively similar number of forks, starts and watchers. However, among these three clusters, the daily stars of cluster 2 suggest that projects in this cluster were able to attract stars faster than the projects in the other two clusters. Cluster 2 also has a shorter release cycle time: 40 days compared to 141 days in cluster 0 and 137 days in cluster 1.

Projects in clusters 0 - 1 are the least popular—they have the lowest forks rate, and star rate. They also grow relatively slower—both of them have smaller open issues rate than others, and have longer release cycles.

### 4 Analysis

Our goal is to monitor a FLOSS project by observing and analyzing its activities, thus, we focused on two analysis tasks: (1) detecting behavioral transitions within a project, to achieve step 3 in the Introduction, and (2) investigating the relationship between sequential behaviors and project measures. Understanding this relationship can help us achieve step 5, that of inferring the relationship between behavioral patterns and success.

---

1. Space limitations prevents us from showing all the cluster attributes.
The descriptive characteristics of the clusters suggest that our KNIME workflow for sequence mining is useful for the classification of GitHub.com projects by their sequential behaviors. We aim to relate the lower-level constructs represented encoded in the clusters to higher-level theoretical concepts. To test the feasibility of this, we ran a linear regression of the clusters with summary project features. Finally, as an added level of validation, we sampled the clusters and correlated their mined characteristics with independent information.

4.1 Regression

For exploration, we applied linear regression to determine if the clusters were related to project summary characteristics, some of which can be interpreted as project performance measurements.

To run the regression, we created dummy variables for the clusters and then ran the linear regression between these dummy variables and project summary variables. We included project duration and number of collaborators as control variables. The results show that the clusters are related to daily stars (adjusted $R^2$ of 0.576) and daily forks (adjusted $R^2$ of 0.477). Table III below summarizes the regression results. The control variable collaborator is not significant, and therefore it is not included in the result table.

<table>
<thead>
<tr>
<th>Independent Variable</th>
<th>Daily Number of Stars</th>
<th>Daily Number of Forks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unstandardized Coefficient</td>
<td>p-value</td>
</tr>
<tr>
<td>Constant</td>
<td>19.892</td>
<td>0.000</td>
</tr>
<tr>
<td>Group0</td>
<td>-11.09</td>
<td>0.000</td>
</tr>
<tr>
<td>Group1</td>
<td>-10.90</td>
<td>0.000</td>
</tr>
<tr>
<td>Group2</td>
<td>-9.05</td>
<td>0.000</td>
</tr>
<tr>
<td>Group3</td>
<td>-2.92</td>
<td>0.221</td>
</tr>
<tr>
<td>Day</td>
<td>-0.005</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*Group0, Group1, Group2 and Group3 are dummy variables created for the 5 cluster groups.

We are also interested on the relationship between programming language and development behaviors. In particular, we investigated if projects that use Java would present different development behavioral patterns. We compared the usage of Java among the five groups. We found that projects in cluster 0 use Java significantly more frequently than projects in other clusters. Interestingly, as discussed before, projects in this cluster are the least popular, with slow growth rate. It will be interesting to investigate this further.

4.2 Behavioral Transitions

Transition identification detects significant changes in modeled events between consecutive windows of event data. Space limitations prevent us from showing all projects, or even all of a single project. However, shows five projects, one from each cluster. Two projects discussed here, ember.js and bootstrap, are the first and second projects.

The x-axis represents the Kullback-Leibler comparison of HMMs generated from the data windows. Each point represents the comparison between two HMMs, each representing a month of data. The trend values are more important than the specific HMM comparison values. Notice that all projects have periods of transition, where their behavior models change significantly, as shown by the spikes. This figure illustrates how well HMM differencing discriminates unusual periods of sequential behaviors from the more common background.

The transitions (spikes) displayed in represent real changes in developer behavior—the developers have changed their patterns of their work motifs. We have correlated those changes with web data to validate that interesting team behaviors are being monitored.

Table IV summarizes bootstrap blog entries that correlate to the transitions presented in . (The x-axis is window count, not date.) These entries provide corroborating evidence that the transitions capture meaningful team behavior.

For example, there is a spike at week 41 (x-axis 6). In the week, the bootstrap team posted a blog entry introducing a new plan of Bootstrap 3. Therefore, an explanation for the spike is that the team began work for Bootstrap 3 after the announcement, bringing a change on their development behavior. Another example is the spike of week 68. Several big events occurred during this period: on the Sunday of the first week, the team called for help for contributors to test the coming new version 2.2.2, on their blog. The version finally came out on that Saturday. Therefore, we can expect that the first week would be a busy week for the team, compared to the following week, after 2.2.2 was released. A similar example occurred at week 34, when the team went through another release announcement—calls for testing in the first week, and then a new release in the early days of the second week. Other spikes can be similarly explained.

5287
Table IV  Blog entries for bootstrap transitions.

<table>
<thead>
<tr>
<th>Week</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>On 14th, there was an announcement about a future release of 2.0.3 on the blog. On 15th, the team called for help on testing 2.0.3 and 2.0.3 was released on 24th.</td>
</tr>
<tr>
<td>41</td>
<td>After the release of v2.0.4 on June 01, 2012</td>
</tr>
<tr>
<td>57</td>
<td>Bootstrap announced on their blog on Sept 29th that they were leaving twitter on 9/29/2012</td>
</tr>
<tr>
<td>63</td>
<td>Nov 9th the team informed that 2.2.2 won’t include glyphicons font on their blog.</td>
</tr>
<tr>
<td>64</td>
<td>11/09: inform 2.2.2 won’t include glyphicons font on the blog</td>
</tr>
<tr>
<td>67</td>
<td>Social distribution: Distribution of social actors among the projects</td>
</tr>
<tr>
<td>68</td>
<td>Structural distribution: Distribution of internal and external(material or environmental) structure</td>
</tr>
<tr>
<td>69</td>
<td>Temporal distribution: Outcome of earlier actions influence the cognitive processes enacted in later efforts</td>
</tr>
<tr>
<td>70</td>
<td>After two pull requests were posted on github on Dec 20th: pull request for bootstrap 3 was posted on github, which would be the next major release with lots of changes, and the pull request of 2.3.0</td>
</tr>
</tbody>
</table>

5 Discussion

This data mining study is aimed at understanding the sequential behaviors of developers in software projects. The experiment produced meaningful clusters from 103 GitHub.com projects. Our regression of clusters with daily stars and daily forks is encouraging, but limited. It demonstrates how automatically mined constructs may be linked to higher-level theoretical concepts. This experiment is an instance of our overall methodology for theory exploration.

Automation was implicit in our discussion; however, all the steps, from project data retrieval through the regression analysis are automated. Retrieval of the project records is, by far, the slowest part of the analysis. On an ongoing basis, a dashboard can update many projects every minute.

Based on these results, the dashboard could cluster projects (roughly) into those that are more popular (having more daily downloads and stars). Note that such characterization is obtained exclusively from the sequential behaviors of the developers. This suggests that in the future, with more complete models, we may produce more refined behavioral analysis on what is most effective for team success.

For future research, we propose to investigate transitions within projects. For example, we could examine if the team size and team structure changes around those transition points. We will then apply the theory of distribute cognition to explore the nature and reasons of those changes, to aid constructing a process model of development behaviors.

Limitations of this study include: (1) sampling, in that the projects may not be representative of FLOSS projects in general; and (2) modeling, in that the measurement of and constructs for sequential behaviors are incomplete. Future work will seek to diminish these limitations.

6 Conclusions

FLOSS developers generate many events, through their tools, which can be used to monitor their progress and predict their results. A carefully constructed data mining workflow can automate the acquisition and analysis of repository events to present a dashboard of clustered projects, highlight when significant changes in developer behaviors have occurred. Now, such automation is of great help to researchers who seek to demonstrate, adapt, and extend software development theories by encoding them in operational process models and using them to conduct exploratory analysis. When such research results become practical, then future dashboards will produce more refined behavioral analysis on what is most effective for FLOSS team success.

7 Acknowledgements

We are grateful for the support of Kalle Lyytinen, Sean Hansen, Aron Lindberg, and the National Science Foundation (IIS 1217552).

8 Appendix

List of project and their associated cluster:

- ActionBarSherlock,0; AFNetworking,0; android,0; android-bootstrap,0; android-bootstrap,1; Android-ViewPagerIndicator,2; AngularJS-Learning,2; annotated_redis_source,1; async,2; atom,1; authlogic,0; AwesomeMenu,0; backbone-boilerplate,0; backbone-fundamentals,1; bash-it,2; bootstrap-sass,4; brackets,1; capistrano,0; chosen,0; cocos2d-html51,1; CodeIgniter-Ion-Auth,1; coffeescript,2; colour-schemes,1; compass,2; coursework,0; cw-omniaus,3; devise,4; discourse,4; docker,0; elasticsearch,3; ember.js,1; fabric.js,0; fastclick,2; FlatUIKIT,2; flight,3; Font-Awesome,0; Front-end-Developer-Interview-Questions,4; Ghost,1; gitflow,4; gitlabhq,0; GMGridView,0; grunt,1; guzzle,4; hackathon-starter,1; handlebars.js,1; highlight.js,1; history.js,0; idiomatic.js,2; intro.js,1; jade,1; jasmine,0; javascript-patterns,1; jekyll,1; jqGrid,3; jQuery-menu-aim,2; jquery-pjax,1; jquerytools,1; scrollPane,1; KineticJS,0; less.js,0; libgd,2; masonry,1; meteor,3; metrics,2; Modernizr,1; moment,0; MWFedParser,1; netty,0; NewsBlur,1; node-webkit,3; normalize.css,1; onepage-scroll,2; OpenTLD,1; parallax,2; phantomjs,0; phonegap-plugins,1; platform_frameworks_base,0; Probabilistic-Programming-and-Bayesian-Methods-for-Hackers,2; ProjectTox-Core,2; pure,2; raphael,1; ratchet,2; ReactiveCocoa,2; reddit,1; resque,1; retire,1; rubinius,0; select2,3; Semantic-UI,4; sizzle,1; SlidingMenu,0; socket.io,0; statsd,1; storm,0; tag-it,1; Telescope,4; TimelineJS,0; typohead.js,2; underscore,1; Vundle.vim,1; wysihtml5,0; x-editable,2; zepto,1

9 References
