Adapting new capabilities or enhancing functionality? Two sequence patterns of capability redeem in platform ecosystems

Philipp Hukal[1], Alexander Eck[2]

[1] Warwick Business School – The University of Warwick
Information Systems & Management Group
Gibbet Hill Road, CV7 4AL Coventry, United Kingdom
p.hukal@warwick.ac.uk

[2] University of St. Gallen
Institute of Information Management
Unterer Graben 21, 9000 St. Gallen, Switzerland
alexander.eck@unisg.ch

Abstract. Platform ecosystem participants often draw from capabilities provided to them by other actors in the ecosystem. While heralded as a driver of innovation, it is unclear how exactly platform evolution is affected by this dynamic. In this short paper we apply event sequence analysis to extract and analyse sequential patterns related to *capability redeem* – i.e. the internalization and subsequent utilization of external capabilities – on platform ecosystems. We find that development sequences differ in their order of events depending on whether a new capability is being integrated as opposed to the capability being put to use to create novel functionality. In particular, we find evidence for two sequence patterns. First, the platform adapting to new resources and incorporating fresh capabilities, leading to little changes in the functionality of the platform. Second, newly gained capabilities are put to use to enhance platform functionality, incurring substantial adjustments to its external behaviour. Located in information systems research on platform-ecosystems these initial findings present potential for future studies.

Keywords: capability redeem, digital platforms, platform ecosystems, sequence analysis, openstreetmap

1 Introduction

Ecosystem dynamics are key in understanding the innovative potential of digital platforms. Complex interdependencies fuel the continuous, iterative, and generative manner in which information resources are recombined across organizational boundaries in order to create digital innovations [1]–[3]. In attempts to sharpen that view, scholars have recently combined views on focal and non-focal actors as ecosystem participants that are not bound to a particular ecosystem, but chose to interact with a selected ecosystem in order to obtain capabilities [4]. A possible course of action for these actors is to extend their capabilities through what has been called *'capability redeem'* [4]. Core to this notion is the idea that ecosystems provide capabilities that are useful in extending participants' ability to innovate [4]. For example, developers of a platform could decide to implement authentication of client applications via a third party protocol such as *oauth*. With the new capability implemented, the platform offers the opportunity to its peripheral actors to develop functionalities that draw from features requiring authentication and authorisation.

While the literature has explored mechanisms of sharing capabilities across platform-based ecosystems, it is unclear how exactly the incorporation and subsequent utilization of ecosystem capabilities plays out and how changes to platform development can be understood. Understanding these developments in platform-ecosystems contexts is important as it would advance the basis for insights on distributed digital innovations.

In this short paper, we therefore demonstrate an approach to study the detailed changes that arise when platforms align their resource base and incorporate capabilities provided through others in their ecosystem. We address the research question; *What are sequences of events of capability redeem in platform-ecosystems?* Drawing on the notion of capability redeem, we are particularly interested in the chain of events during the integration of capabilities to be gained as well as their subsequent utilisation.

We present the case of the OpenStreetMap.org (OSM) platform to answer the above research questions and report initial evidence from a larger study on platform development. Conceptualising capability redeem as a two stage process that involves integrating as well as using the newly gained capability, we identified two focus episodes from the case. By applying computational event sequence analysis to each episode, we explicate trajectories of changes to understand how redeeming ecosystem capabilities influences the development of new functionality on the OSM platform.

The remainder of the paper is structured as follows. In the next section we frame our approach to capability redeem through ecosystem participation by briefly reviewing relevant literature. We then detail our approach by highlighting the empirical context as well as event sequence analysis methods. Finally, we present initial results before concluding and pointing to opportunities for further research.

2 Related Work

Co-evolving organisational capabilities through ecosystem participation is a notion firmly rooted in the information systems literature [3]. One commonly investigated ecosystem topology is that of a focal actor which – via a central platform – provides shared resources to a set of connected artefacts owned by non-focal actors. Innovation capability of a non-focal actor is thought to be a function of how well it combines shared resources with own assets to generate novelty. Meanwhile, the focal actor profits from innovations as they grow the overall ecosystem around the platform as well as the perceived innovation potential compared to other platforms [5]–[7]. IS scholars offer ample evidence on the benefits actors gain from platform-ecosystem participation and how they are actualized (e.g. [8], [9]).

Considerable work has gone into explicating the mechanisms by which the exchange of resources and capabilities foster innovation for participants. For instance, some point out that providing interfaces facilitates innovation and growth in platform ecosystems [10]. So-called *boundary resources* let the focal actor control what and how resources are available to non-focal actors. However, far from being unilaterally controlled, interfaces are subject to complex interaction between focal and non-focal actors. As reported in [11], the balancing act of *'accommodating and resisting'* peripheral interests through shared resources is crucial for platform innovation. In cases where capabilities provided to peripheral actors are too limited, the risk of impeding their innovative capability through ecosystem participation causes tensions.

Resolving such tensions is the purpose of governance, which aims to align detrimental interest between participants. To meet competing demands, authors have suggested different governance principles [12]. One such principle is to ensure *'stability as well as evolvability in outputs'* (p. 1199) of the ecosystem, with the goal to exercise control over undesirable variance which shall pave the way for desirable variance. For example, assuring interoperability with one ecosystem initially decreases variability, yet in the same time, it also enables variability in a different part of the platform ecosystem as freed up efforts can be directed towards innovation.

Relevant IS literature advocates benefits of ecosystem participation. Participants have a clear interest in drawing from capabilities offered through the ecosystem as far as this advances their ability to innovate [4]. The pre-condition of aligning and redeeming capabilities is implicit in the studies, but rarely a topic in its own right. While studies hint at the necessity to align capabilities in order to innovate, the precise activity involving capability redeem in order to innovate remains unexplored and the outcomes undertheorized.

For this paper, we draw on the idea of capability redeem [4] to understand platform innovation. The perspective provides a solid framing for our empirical context while affording flexibility for subsequent theorisation. Additionally, capability redeem is a useful notion in a number or ways. First, it allows the analysis of platform activity that is the same time both focal and non-focal to different ecosystems. Second, it provides a lens on platform capabilities as the ability to innovate by drawing from a suit of resources provided by another ecosystem actor. Thirdly, it enables a bi-directional view. That is, it acknowledges the flow of capability redeem as being gained from one ecosystem where the actor might be on the periphery and passing it on to a different ecosystem to which the actor might be core and hence provides the capability to others. In sum, capability redeem as formulated in [4] serves as a theoretical perspective with which to understand an actor's innovative ability as a function of redeeming external capability across digital ecosystems.

3 Research Design

We examine the case of the openstreetmap.org (OSM) platform to study how capability redeem in a platform ecosystem is unfolding. OSM, the platform, is an open source software artefact which produces the assets of OpenStreetMap, the geo-spatial data project, available both via programmable interfaces and via the browser. In operation since 2004, 'the Wikipedia of maps' [13] is considered the world's largest community-driven mapping project on the web with over 3.1 million registered users who to date contributed a total of 5.4bn geo-spatial data points[1].

OSM is a well-suited case to showcase our empirical approach and discuss capability redeem in an ecosystem context. Over time, the platform has been adapted and extended to draw on many technical solutions provided by other (mostly open source software) projects such as *leaflet.js* (a library for interactive maps), *Rails* (a web development framework), and *Mapnik* (a rendering engine). In turn, OSM provides actors in the OSM ecosystem various capabilities to handle geo-spatial data. These capabilities are immensely popular and several hundred commercial and non-commercial web services are drawing from OSM data and related data handling capabilities in their configurations, for instance, Craigslist, Wikipedia, and Foursquare[2].

The position of OSM as simultaneously a focal actor to its own ecosystem and non-focal actor in relation to other ecosystems (such as the ecosystem around leaflet.js, Rails, and Mapnik) presents a fit with the conceptual perspective on ecosystems proposed by [4]. The perspective offers an initial framing for this showcase without pre-empting subsequent avenues for theorisation. Moreover, the OSM case offers advantages in terms of accessibility. OSM source code is hosted on GitHub.com to coordinate development work among OSM developers. GitHub is based on the Git version control system and lets users share, propose, and discuss software code (cf. [14], [15]). In the case of an open source software project such as OSM, any GitHub user may openly access its codebase and propose source code changes (*'pull requests'*), in addition to discussing plans and issues on associated online forums. If a core member of the project agrees with the suggested changes, the proposal is *'committed'*, or merged with the original code base and the changes take effect (compare [16]). Github makes the content of commits together with a number of metrics publicly available.

To focus the study, we identified a pertinent episode of capability redeem in the context of OSM that demonstrably resulted in innovation. Our conceptualisation of capability redeem implies two related yet distinct streams of activity. First, the addition of the desired capability to the suite of platform resources. Second, the utilisation of the gained capability for innovation on the platform.

[1] Database statistics as of October 2016,
available at: http://www.openstreetmap.org/stats/data_stats.html

[2] http://wiki.openstreetmap.org/wiki/List_of_OSM-based_services

As such we chose the update of Ruby on Rails to version 3.0 and subsequently version 3.1 as our first episode of interest. As a subsequent episode of utilisation of newly gained capabilities related to the Rails update, we chose the extension of the OSM data model with *redactions*, i.e. the possibility to prevent the circulation of OSM data objects across all instances of the OpenStreetMap database. The Rails update episode and the resulting utilisation of the capabilities gained from it are a meaningful representation of capability redeem on OSM. The Rails update 3.0/3.1 focused substantially on the interaction between Rails components and database operations. Through the set of novel capabilities OSM was able to react swiftly to two external events that made adjustments to the platform necessary. First, the inclusion of redactions as an extension to the OSM data model, protected OSM from copyright infringement allegations, should data be distributed that violated intellectual property rights of others. Second, in 2010 the project underwent a license change from CC-BY-SA[3] towards ODbL[4] covering the OpenStreetMap data. This made it necessary to exclude all data points committed to OSM from distribution for which the new license agreements have not been accepted by the uploader yet. We conceptualise these examples as illustrations of our approach while framing the paper with the conceptual lens that capability redeem provides. As stated earlier, OSM is dependent on the Rails ecosystem to provide capabilities needed in operating OSM in its current form [17]. Rails powers the OSM web application responsible for the OSM website, application programming interfaces (APIs), as well as other components such as editors needed to handle geo-spatial data. As such, Rails is crucial for the OSM platform as it enables a number of key capabilities that the platform offers to other participants.

3.1 Data

We queried the GitHub web API and downloaded all source code changes committed to the OSM Rails port. Specifically, we focused on changes made to the *'rails controllers'*. In the Rails architecture, *controllers* manage any information processing activity in web applications (called business logic in other contexts). Controllers

[3] Creative Commons Attribution Share Alike
[4] Open Data Commons Open Database License (similar to CC-BY-SA, but specifically for data)

coordinate operations between the data *model* and the various *views* presented to the user [18]. The rails controllers thus present a manageable subset of OSM development data without sacrificing basis for inference.

With the constrains described above we used the downloaded commits to construct an event sequence dataset [19], [20]. Conceptually, sequences are finite sets of ordered elements such as events, states, or activities following a temporal, logical, or spatial ordering principle [19]–[22]. Using a quadripartite approach to conceptualise sequential data structures the following aspects guided data collection and transformation cf. [19]–[21]; (1) the unit of analysis, (2) records of events, (3) records of activity, and (4) a timeline of observation.

(1) Unit of Analysis: The unit of analyses in this paper are the temporarily ordered sequences of source code changes committed to selected OSM components (i.e. *rails controllers,* hereafter simply components). This level of detail allows analysis of the changes made to every single component and was necessary to avoid inaccuracies. A commit on GitHub often includes several changes at once and so a single commit may affect multiple components, but not necessarily to the same effect. In order to increase efficacy of the sequence method, we analysed each source code change on the basis of the component it relates to.

(2) Records of Events: Records of events form the first part of bi-modal data needed to construct sequences. Event sequences typically represent changes in states of the unit of analysis. For instance, in a life course study, the event 'graduation' changes the state of an individual from student to graduate. Here we consider every commit a separate event: Whenever one component was affected by a commit and hence its source code was changed, we recorded it as one event. The entirety of commits relating to one component makes up that component's development sequence.

Some data transformation was necessary to allow a meaningful analysis of the collected sequential data. First and foremost, manual filtering was necessary to eliminate noise that was introduced to the data by the temporal order of commits. All commits have been ordered by continuous calendar time based on the timestamps provided by GitHub metadata[5]. This consequently implied a relation

[5] Note: We use the timestamps the commit was added to the code base (as opposed to when the commit was authored)

between events (i.e. commits) by virtue of having occurred immediately before or after one another. This was the case regardless whether these events are causally related, that is, actually relating to the same source code issue. Against the backdrop of the identified episode of capability redeem this required to identify and derive a subset from the data that contained commits exclusively referring to the two episodes *'Rails Update'* and *'Redaction Implementation'* respectively.

Filtering of events was informed by the Rails change logs[6,7] to identify potential source code changes relating to aspects of the Rails update. Prolific changes that guided the data cleaning are detailed in Table 2. Consequently, the number of all commits associated with every component was reduced to commits that were clearly attributable to the 'Rails Update' ($n = 144$) and the 'Redaction Implementation' ($m = 37$) respectively.

(3) *Records of Activity:* The second part of bi-modal data for sequence analysis requires a finite set of mutually-exclusive categories of activity [23]. Tracked across time, events are selected from pre-defined categories able to qualify what kind of event the unit of analysis is subjected to. Therefore, the source code changes made to a component as part of a commit, constitute the activity of the event. To create a set of possible categories of source code changes, we classified each commit following the taxonomy of software changes developed by [24]. The classes relevant to this study include changes to the source code base and where applicable changes made to functionality and external behaviour of the software. In so doing, we examined each commit and assigned one of the category labels as described in [24]. The resulting sequence alphabet for this study has a size of five elements describing the types of events found in the data (*adaptive, corrective, enhancive, groomative, reductive*)[8] [25], [26]. Table 1 below summarises the applicable classifications, presents indicators as well as illustrative examples.

[6] http://guides.rubyonrails.org/v3.2/3_0_release_notes.html

[7] http://guides.rubyonrails.org/v3.2/3_1_release_notes.html

[8] Note:

Events of type 'documentation' were not found in the two focus episodes

Events of types 'preventive' and 'performance' were omitted from the analysis as they are effectively unrecognisable without more intimate knowledge of the change process; Also see [24] for further details.

Classification was conducted by examining each commit through the associated URL provided by GitHub, assessing the source code changes made to every file and deciding on a classification of the changes. The approach assured that commits changing multiple components are classified according to the source code changes they made to each respective component. Ties in classifications were resolved through discussions with agreement averaging 73.94% after three iterations of coding.

(4) *Timeline of Observation:* The rails update 3.0 was initiated on November 14th 2011. The inclusion of redactions to the data model began on April 5th 2012. By December 10th 2013, the Rails Port was once again updated to Rails 4.16, marking the end point of this study's observational timeline. To showcase our approach, we hence only consider events that have taken place in the 25 months between November 2011 and December 2013. This delimitation allows the construction of a clear timeline with well-defined start and end points.

In sum, the outlined approach resulted in the creation of an event sequence data set, comprising *27 sequences* (19 relating to Rails Update, 8 to Redact Implementation) each representing all relevant change made to one of the affected components in the time from November 2011 to December 2013. Figure 1 below presents an illustrative event sequence with the coded short label for each event type.

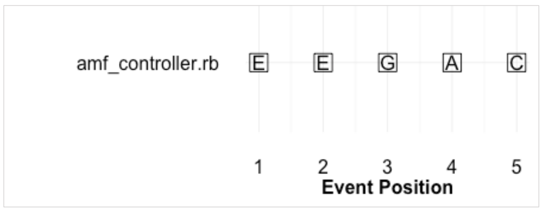

Fig. 1. Illustrative sequence selected from the dataset; Labels correspond to the coded event types of source code changes; A = Adaptive, C = Corrective; E = Enhancive; G = Groomative; R = Reductive

Table 1. Categories of source code changes based on classification developed by [24].

Functionality	Category	Explanation	Indicator
Unchanged	Groomative	Change with the purpose to make the source code more maintainable, understandable, or usable.	Refactoring, Renaming, Style Guide Compliance
Unchanged	Preventive	Changes aimed at preventing future malfunctioning	*Not applicable;* preventive changes are indistinguishable from other forms of changes unless explicitly stated
Unchanged	Performance	Changes aimed at improving system performance, such as increasing uptime, decrease resource usage	*Not applicable;* performance changes are indistinguishable from other forms of changes unless explicitly stated
Unchanged	Adaptive	Changes that alters the technologies or resources used, or that restore compatibility with platform or module due to changes in the artefact	Using new function calls, including new algorithms, drawing form external resources
Changed	Corrective	Changes that restore a defunct functionality or assure its anticipated behaviour	Bug fixes; dealing with edge cases
Changed	Reductive	Changes that eliminate or restrict functionality; with effects on external behaviour both visibly and invisibly	Redirecting data flow to exclude a user group from access to resources; deactivating a default function call or route
Changed	Enhancive	Changes that create new functionality with visible effects on external behaviour	Adding a feature, introducing new information flow, allowing novel interaction with the software

Table 2. Major changes in Rails Updates 3.0 / 3.1; adapted from Rails change logs (see footnotes 7 and 8 above).

Issue	Description
New APIs	Introduction of new APIs for web applications; e.g. Mailer API for mail notification capabilities provided by Rails
AREL Methods	Complementing Active Record methods in database queries by explicitly supporting AREL function calls in database handling
XHR Updates	Update of xml_to_html requests leads to changes in javascript pushstate functions used to interact with client application (e.g. browser history)
CSRF authentication	Adjustments to CSRF token check now activated by default and evoked in *“app/controllers/application_controller.rb”*
Default on jQuery	New default javascript framework for AJAX queries in Ruby

3.2 Data Analysis

We use computational event sequence analysis to the filtered, ordered and coded datasets to study the chain of events occurring in the identified episodes that we conceptualise as part of capability redeem. We are specifically interested in the piecemeal steps constituting patterns within and across sequences. We therefore organised the dataset in an event sequence structure and aligned sequences by event order. This allows the time agnostic analysis of individual events and prioritises the investigation of event order, shared sub-sequences and common event transitions over duration and timing of events [20]. In what follows we present our approach that uses the R package *'TraMineR'*. Aimed at the analysis of sequence data in the social sciences, *'TraMineR'* provides a range of functions for statistical programming in R[9] [27].

Two questions guide the analysis of the development sequences; First, do sub-sequence patterns differ across episodes? Second, what whole-sequence patterns are characteristic for each respective episode?

First, we extracted discriminant sub-sequences following an approach described in [20]. The approach searches for frequent items in an ordered set. Initial inspection of our dataset shows that sequences are of unequal length with a maximum of 18 and a minimum of 3 events per sequence (*mean: 6.7; median: 5.0; SD: 4.38*). Rooted in the empirical context and informed by our research focus, we therefore defined five search parameters which determined what and how many sub-sequences are to be included for analysis (for details see [20]).

(1) *Start of sub-sequence:* To include as many sequences as possible in the analysis, we limit the event position by which each sub-sequence has to have started to the second event. Given the minimal sequence length of some sequences in the set, this would still yield sub-sequences of length two with one transition between events.

(2) *Length of sub-sequence:* With no possible a priori assumptions about expected sequences, the minimum sub-sequence length of interest is two events. Since we are interested in maximising the sample of possible sequences to describe each episode, this results in the inclusion of every possible event transition.

[9] All data collection, transformation and analysis is done using R version 3.3.1; see R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. (https://www.R-project.org/)

(3) *Spread of sub-sequence:* Similar to *(2),* we do not restrict the maximum length of a sub-sequence (i.e. number of events) or the time window in which it has to be completed, meaning that a sub-sequence is counted even if it is interrupted by another event.
(4) *Overlap of sub-sequence:* We include overlap of spells so that a sub-sequence is counted if a meaningful transition occurs multiple times in varying time windows as long as *(1)* is observed (for details see [20]).
(5) *Support of sub-sequence:* Every subsequence that occurred at least once, conditional on *(1).*

The second stage of the analysis is concerned with the derivation of representative whole sequence patterns in each episode. To that end, we apply optimal matching analysis to the whole sequences. Albeit nascent in social science research, optimal matching is an established approach to dealing with sequential data with prominent applications in sociology and origins in biology [19], [28], [29]. Generally, optimal matching refers to techniques aimed at measuring the similarity of diverse kinds of ordered data. In event sequence analysis, this is achieved by aligning the data such that each sequence element position in one sequence aligns with the respective element position of the next sequence. Irrespective of the exact time point of an event this explicates the order in which events occurred in each sequence.

TraMineR uses the established *'Needleman–Wunsch'* algorithm to compute a similarity measure for each pair of sequences [30], [31]. *Needleman–Wunsch* tests which arrangement of pre-defined operations minimises the number of steps needed to transform one sequence into another. Two basic operations are available to manipulate sequences; insertion (i.e. adding), and deletion (i.e. excluding) of sequence elements. Each operation is assigned a cost representing the hypothetical effort needed to apply an operation to an element. We use the classical *Levenshtein Distance* in our analysis. As such, the cost for both operations is set to one unit, regardless whether a sequence element is inserted or deleted to align two sequences. Hence, the used similarity score simply represents the total number of operations needed to transform sequences.

The cost regime is not arbitrary as sequence alignment operations and their costs are best informed by subject matter expertise and research context [31], [32]. As such, assigning an equal cost to both operations is reasonable for our study: Lacking information as to what kind of source code changes to expect from capability redeem activity, no justification exists for setting varying alignment costs for a priori. For instance, setting costs based on observed or expected frequencies of events from the sequence alphabet is unmerited since we have no reason to believe that any kind of source code change is more or less likely.

Applying *Needleman-Wunsch* with the *Levenshtein* cost regime yields a dissimilarity matrix with the minimal similarity scores for each sequence. Following an approach described by [33] we normalized the similarity measures by maximum sequence length, ensuring that similarity scores are not biased due to varying number of elements in each sequence.

4 Results

The results indicate that the sequence of events in each episode differ substantially. Both, the analysis of sub-sequence elements as well as representative whole-sequences provide evidence for the existence of fundamental differences in the sequential patterns of source code changes in both selected episodes. We report these initial findings according to the two guiding questions on sub-sequence and whole-sequence differences.

4.1 Discriminant Sub-Sequence Patterns

First, we were interested in potential differences in the order of events in each development episode asking the question: *Do sub-sequence patterns differ across episodes?*

The result from the discriminant sub-sequence analysis clearly indicate that the two episodes differ in terms of the elements and sub-sequences that constitute the chain of events in each episode. Figure 2 below displays the 25 most determinative sub-sequences for each episode. For reasons of visual accessibility, we only plot sequences with maximum length of three elements. Given the low average length of sequences in our sample, this does not present a limitation.

In general, the events associated with the *'Rails Update'* episode predominantly display sub-sequences that begin with an 'adaptive' event (17 of 25), followed by various combinations of the events 'corrective', ‘groomative’, or ‘reductive’. The frequency of events of the ‘enhancive’ category (i.e. adding new functionality) are rare in that group (indicated by the downward direction of the bar representing the sign of the Pearson residual). Interestingly, no sub-sequence is statistically significant for this episode (statistical significance refers to chi square test results between episodes). This indicates that sequence patterns exist in too great a variety within the episode to be conclusive about what sub-sequences are determinative.

In contrast, consider the sub-sequence patterns that characterise the *‘Redaction Implementation’* episode. Here, sub-sequences beginning with ‘enhancive’ events are not only more frequent (6 of 25), but are also statistically significant in terms of their chi-square test results (indicated by the grey shading of bars) [33]. This indicates that a sequence within the second episode can be identified through the existence of sub-sequence elements of type ‘enhancive’. The patterns in positive and significant sub-sequences exclusively begin with an ‘enhancive’ event followed by combinations of ‘corrective’, ‘adaptive’, or ‘groomative’ events.

Sub-sequence elements with ‘adaptive’ events in the beginning or among the first sequence elements are common across the *‘Rails Update’* episode. In contrast they are less so in the *‘Redaction Implementation’* episode. In fact, two of the sub-sequence spells beginning with an ‘adaptive’ event are negative identifiers. That is, on a .05 significance level, a sub-sequence can be categorised as *not* being from the ‘Redaction Implementation’ episode if it contains these patterns.

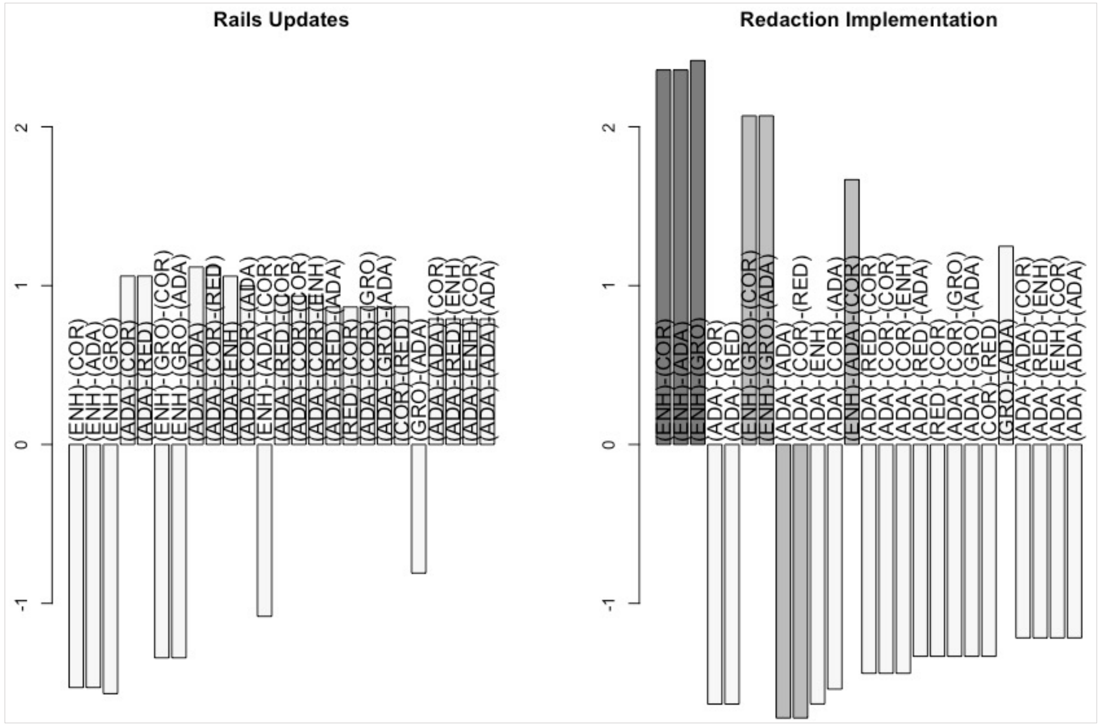

Fig. 2. Top 25 discriminant sub-sequence patterns by episode: Labels correspond to the coded event types of source code changes; ADA = Adaptive, COR = Corrective; ENH = Enhancive; GRO = Groomative; RED = Reductive; Shading represents significance on 0.01 level (dark grey), 0.05 level (medium), and 0 (light) based on Chi Square tests; Bar direction indicates residual sign of the Pearson Residual

4.2 Representative Whole-Sequence Patterns

Next, we turn to the analysis results of whole-sequences, asking; *What whole-sequence patterns are representative for each episode?* We therefore used the dissimilarity matrix as described above to derive sequences of source code changes that are representative of each episode. We extracted the five sequences from each episode with minimal dissimilarity scores. These sequences are referred to as representative sequences as they display cases for which the effort needed to transform them in any other sequence is lowest [33]. They are the cases with the highest degree of resemblance to all other sequences; hence representative of their set.

Consistent with the analysis of sub-sequence elements, the analysis of whole sequences confirms the differences of sequence within each episode. The five most representative sequences in the *'Rails Update'* episode all start with 'adaptive' events, followed by either 'reductive' or 'corrective' code changes. Only exception to this are two sequences (#4, #5) exhibiting an event of type 'enhancive' later in the sequence.

The pattern can be interpreted as incorporating external capabilities to the OSM platform artefact. The source code changes needed to *'adapt'* to the new capabilities gained from updating Rails initiate the sequences, while the rest of the development work is characterised by correcting functions by either fixing or restricting them ('corrective' or 'reductive' events) or as in one occasion maintaining the appearance or style of the code base. The top 5 sequences are illustrated in Figure 3 below.

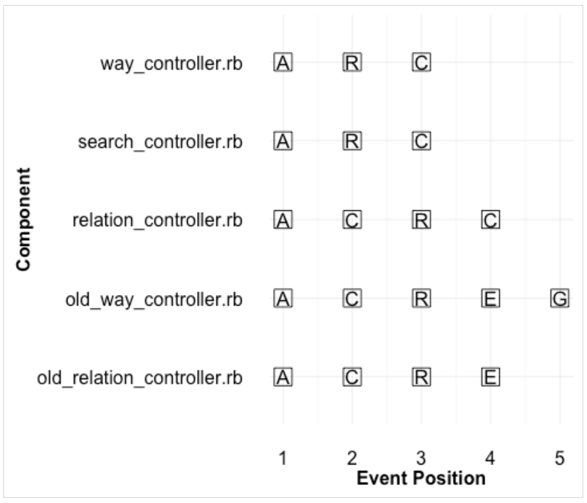

Fig. 3. Top 5 representative development sequences for the Rails 3.0/3.1 Updates per component; Labels correspond to the coded event types of source code changes; A = Adaptive, C = Corrective; E = Enhancive; G = Groomative; R = Reductive

The sequences in the *'Redaction Implementation'* episode stand in stark contrast to the ones in the *'Rails Update'*. Here, without exception, the most representative sequences all begin with an 'enhancive' event indicating the addition of new functionality to respective components. The sequences proceed with high variety of possible events before ending with variations of 'groomative', 'corrective' or 'adaptive' events. As in the analysis of sub-sequence patterns, the event of type 'reductive' is absent from this set of sequences. (see Figure 4)

The pattern can be interpreted as utilizing the newly gained capabilities by adding new functionality to the artefact. Related to the introduction of novelty ('enhancive' events) is the adjustment of resources indicated by 'adaptive' events. The majority of succeeding events deals with the aligning existing functionality indicated by 'corrective' events as well as maintaining standards in the source code base as represented by 'groomative' events. Given the small number of eight sequences in this episode, the pattern across the five representative sequences is especially remarkable.

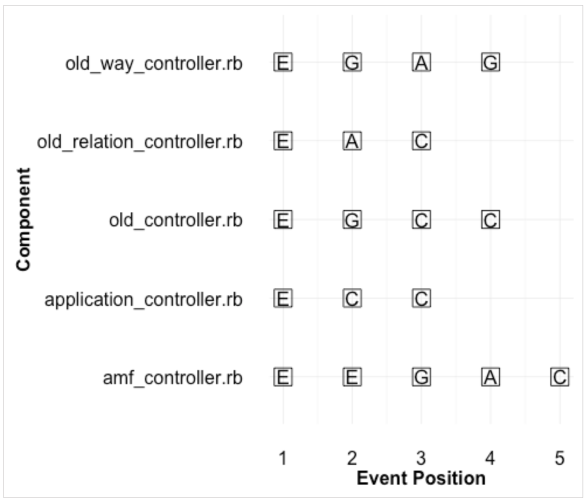

Fig. 4. Top 5 representative development sequences for the implementation of redaction resources; Labels correspond to the coded event types of source code changes; A = Adaptive, C = Corrective; E = Enhancive; G = Groomative; R = Reductive

5 Conclusion

In summary, the apparent differences in sub-sequence and whole-sequence patterns indicate contrasts in the unfolding event order during capability redeem affecting platform evolution. The sequences of source code changes in the *'Rails Update'* episode consists of events associated with maintaining of functionality by use of newly available resources ('adaptive' events). This adaptation is followed by dealing with edge cases, corrections, and restrictions of existing functionalities later in the development sequence ('corrective' and 'reductive' events). The addition of novel functionality is rare in the entire episode as refelcted by only two elements of type 'enhancive' in the representative sequences in figure 3. This is unsurprising, given that the alignment of the resource base through interaction with an ecosystem such as Rails entails adjusting the resources needed to perform functionality. From the perspective of capability redeem, OSM is a non-focal actor which needs to accommodate changes in the Rails ecosystem to keep profiting from redeeming capabilities from Rails.

Conversely, the *'Redaction Implementation'* episode is characterised by numerous additions of new functionality, followed by aligning resources as well as tending to maintainability of source code. This is evident from the prevalence of ‘enhancive’ events in the beginning of the representative sequences followed by combinations of ‘corrective’, ‘groomative’, and ‘adaptive’ events (compate figure 4).

Evidence from the *‘Rails Update’* episode of integrating capabilities gained from an ecosystem partner does not support conclusive statements at this point. None of the sub-sequence patterns are significantly different from the sequences contained in the second episode. While revealing interesting descriptions of the source code changes entailed in the integration of redeemed capabilities, this highlights a need for further investigation. Contrarily, the episode dealing with the utilisation of newly gained capabilities from ecosystem partners by implementing a new feature, offers promising avenues for further research. Implicit in the sequence of source code changes associated with the second episode is the notion that the inclusion of novel functionalities alters the external behaviour of the software. While not surprising in itself, this idea invites reflections about how the evolution of platform-ecosystems is affected by redeeming capability from ecosystem actors. The OSM platform clearly benefits from integrating capabilities and resources provided by the Rails ecosystem. While not advancing functionality of OSM software immediately during the implementation, our evidence suggests that subsequent design of novel functionality is substantially affected by the capabilities OSM developers gained by aligning with updated Rails resources.

Several avenues for further research on platform-ecosystems are possible based on initial insights showcased here. First, a perspective on digital innovation through path constitution theory ([34], [35]) may be connected with our findings. Key to the development of digital platforms are capabilities provided through ecosystem dependencies. It thus could be of great interest to path constitution researchers to investigate how technology paths are influenced by the tension between design agency and ecosystem dependencies. To that end, the event sequences presented here can serve a starting point for further analysis. Regarding individual commits and their effect on functionality of a platform as a step in a technology development path might be a promising approach: ‘Emergent designs’ [36] can be traced by analysing multiple simultaneous development sequences on a highly granular level. Sequence analysis as well as the detailed view on technology development in a platform context may be intriguing for anyone interested in extending path theoretic perspectives.

Furthermore, questions concerning changing design rules [37] might be of interest for platform researchers. Our results indicate that the sequences of events stemming from ecosystem dependencies have substantial effect on platform evolution as new functionality is influenced by capabilities provided by ecosystem actors. Altering the architecture of digital platforms triggers changes in crucial task and design structures and thus influences subsequent technological development and innovation on the platform. The malleability of digital technology is a matter of course for information systems researchers, but detailed insights into the minute design changes and trajectories remain to be theorised and are hence promising for future studies (e.g. [15], [21]).

Lastly, our approach highlights the feasibility of tracking platform development on the level of individual components. As such, development sequences can be related to questions of platform component composition (see [38]). For instance, the time needed for a component to fully adopt a new functionality change or the rate with which changes occur in a component present opportunities for investigations on platform development as a function of the ease with which its components incorporate changes. Using measures relating to sequential dynamics would thus allow to derive insights on platform evolution.

On a final note, we hope that despite its early stage character, this paper demonstrates computational sequence analysis as a potentially fruitful approach for anyone interested in understanding digital technology development in platform-ecosystem contexts.

Acknowledgments.
The authors would like to thank two anonymous reviewers as well as the participants of the Pre-ICIS event '*8th International Workshop on Software Ecosystems*' on December 10th in Dublin, Ireland for their helpful comments on an earlier version of this paper.

References

[1] Y. Yoo, O. Henfridsson, and K. Lyytinen, "The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research," *Inf. Syst. Res.*, vol. 21, no. 4, pp. 724–735, Dec. 2010.

[2] Y. Yoo, R. J. Boland, K. Lyytinen, and A. Majchrzak, "Organizing for Innovation in the Digitized World," *Organ. Sci.*, vol. 23, no. 5, pp. 1398–1408, 2012.

[3] G. Adomavicius, J. C. Bockstedt, A. Gupta, and R. J. Kauffman, "Technology roles and paths of influence in an ecosystem model of technology evolution," *Inf. Technol. Manag.*, vol. 8, no. 2, pp. 185–202, 2007.

[4] L. Selander, O. Henfridsson, and F. Svahn, "Capability search and redeem across digital ecosystems," *J. Inf. Technol.*, vol. 28, no. 3, pp. 183–197, May 2013.

[5] A. Tiwana, B. Konsynski, and A. A. Bush, "Research Commentary —Platform Evolution: Coevolution of Platform Architecture, Governance, and Environmental Dynamics," *Inf. Syst. Res.*, vol. 21, no. 4, pp. 675–687, Dec. 2010.

[6] A. Gawer and M. Cusumano, "Industry Platforms and Ecosystem Innovation," *J. Prod. Innov. Manag.*, vol. 31, no. 3, pp. 417–433, May 2014.

[7] C. Y. Baldwin and C. J. Woodard, "The architecture of platforms: a unified view," in *Platforms, Markets and Innovation*, A. Gawer, Ed. Cheltenham: Edward Elgar, 2009, pp. 19–44.

[8] M. Ceccagnoli, C. Forman, and P. Huang, "Cocreation of Value in a Platform Ecosystem: The Case of Enterpise Software," *MIS Q.*, vol. 36, no. 1, pp. 263–290, 2012.

[9] K. Boudreau, "Let a Thousand Flowers Bloom? An Early Look at Large Numbers of Software App Developers and Patterns of Innovation," *Organ. Sci.*, vol. 23, no. 5, pp. 1409–1427, 2012.

[10] A. Ghazawneh and O. Henfridsson, "Balancing platform control and external contribution in third-party development: the boundary resources model," *Inf. Syst. J.*, vol. 23, no. 2, pp. 173–192, Mar. 2013.

[11] B. Eaton, S. Elaluf-Calderwood, C. Sorensen, and Y. Yoo, "Distributed Tuning of Boundary Resources: The Case of Apple's iOS Service System," *MIS Q.*, vol. 39, no. 1, pp. 217–243, 2015.

[12] J. Wareham, P. B. Fox, and J. L. Cano Giner, "Technology

Ecosystem Governance," *Organ. Sci.*, vol. 25, no. 4, pp. 1195–1215, 2014.

[13] K. Fox, "OpenStreetMap: 'It's the Wikipedia of maps'," *The Guardian*, 18-Feb-2012.

[14] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social Coding in GitHub : Transparency and Collaboration in an Open Software Repository," in *CSCW Feburary 11-15*, 2012.

[15] A. Lindberg, N. Berente, J. Gaskin, and K. Lyytinen, "Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project," *Inf. Syst. Res.*, pp. 1–44.

[16] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *IWPSE Proceedings*, 2002, pp. 76–85.

[17] F. Ramm, J. Topf, and S. Chilton, *OpenStreetMap Using and Enhancing the Free Map of the World.* Cambridge, UK, 2010.

[18] S. Ruby, D. Thomas, and D. H. Hansson, *Agile Web Development with Rails, 4th Edition, Rails 3.2*, 4th Editio. Dallas, TX (US), 2011.

[19] A. Abbott, "A Primer on Sequence Methods," *Organ. Sci.*, vol. 1, no. 4, pp. 375–392, 1990.

[20] G. Ritschard, A. Gabadinho, N. S. Muller, and M. Studer, "Mining event histories: a social science perspective," *Int. J. Data Mining, Model. Manag.*, vol. 1, no. 1, p. 68, 2008.

[21] J. Gaskin, N. Berente, K. Lyytinen, and Y. Yoo, "Toward Generalizable Sociomaterial Inquiry: A Computational Approach for Zooming In and Out of Sociomaterial Routines," *MIS Q.*, vol. 38, no. 3, pp. 849–871, 2014.

[22] B. Cornwell, "Theoretical Foundations of Social Sequence Analysis," in *Social Sequence Analysis - Methods and Applications*, New York: Cambridge University Press, 2016, pp. 21–56.

[23] A. Abbott and A. Hrycak, "Measuring Resemblance in Sequence Data: An Optimal Matching Analysis of Musicians' Careers," *Am. J. Sociol.*, vol. 96, no. 1, pp. 144–185, 1990.

[24] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, "Types of software evolution and software maintenance," *J. Softw. Maint. Evol. Res. Pract.*, vol. 13, no. November 2015, pp. 3–30, 2001.

[25] G. Ritschard, “Exploring sequential data,” in *Discovery Science, 15th International Conference, Lecture Notes in Computer Science; DS 2012, Lyon, France, October 29-31, 2012*, 2012, vol. 7569 LNAI, pp. 3–6.

[26] J.-A. Gauthier, P. Blanchard, and F. Buhlmann, “Introduction: Sequence Analysis in 2014,” in *Advances in Sequence Analysis: Theory, Method, Applications*, P. Blanchard, F. Buhlmann, and J.-A. Gauthier, Eds. Heidelberg/New York: Springer, 2014, pp. 1–21.

[27] A. Gabadinho, G. Ritschard, and M. Studer, “Analyzing and Visualizing State Sequences in R with TraMineR,” *J. Stat. Softw.*, vol. 40, no. 4, pp. 1–37, 2011.

[28] T. Biemann and D. K. Datta, “Analyzing Sequence Data: Optimal Matching in Management Research,” *Organ. Res. Methods*, vol. 17, no. 1, pp. 51–76, 2014.

[29] A. Abbott, “Sequence Analysis: New Methods for Old Ideas,” *Annu. Rev. Sociol.*, vol. 21, pp. 93–113, 1995.

[30] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[31] B. Cornwell, “Whole-Sequence Comparison Methods,” in *Social Sequence Analysis - Methods and Applications*, New York: Cambridge University Press, 2016, pp. 109–144.

[32] L. Lesnard, *Setting Cost in Optimal Matching to Uncover Contemporaneous Socio-Temporal Patterns*, vol. 38, no. 3. 2010.

[33] A. Gabadinho, G. Ritschard, M. Studer, and N. S. Müller, “Extracting and rendering representative sequences,” *Commun. Comput. Inf. Sci.*, vol. 128, pp. 94–106, 2011.

[34] J. Sydow, A. Windeler, G. Müller-Seitz, and K. Lange, “Path Constitution Analysis: A Methodology for Understanding Path Dependence and Path Creation,” *BuR - Bus. Res.*, vol. 5, no. 2, pp. 155–176, 2012.

[35] R. Singh, L. Mathiassen, and A. Mishra, “Organizational Path Constitution in Technological Innovation: Evidence from Rural Telehealth,” *MIS Q.*, vol. 39, no. 3, pp. 643–666, 2015.

[36] R. Garud, A. Kumaraswamy, and V. Sambamurthy, “Emergent by Design: Performance and Transformation at Infosys Technologies,” *Organ. Sci.*, vol. 17, no. 2, pp. 277–286, 2006.

[37] C. Y. Baldwin and K. B. Clark, *Design Rules - the Power of*

Modularity. *Cambridge, MA: MIT Press*. Cambridge, MA: MIT Press, 2000.

[38] A. Tiwana, *Platform Ecosystems: Aligning Architecture, Governance, and Strategy*. Amsterdam: Morgan Kaufmann, 2014.