Abstract

Temporary gatherings—such as conferences, trade shows, and festivals—create a forum where strangers meet one another, thereby helping people establish new social connections. The infusion of new social ties may permit knowledge transfers that facilitate recombinant search. However, knowledge transfer is not assured: ties newly formed at a temporary gathering may have low information carrying capacity. In this research project, we examine under what conditions temporary gatherings catalyze the creation of novel combinations. We study a large sample of novel technological combinations created at software development hackathons. We find that both the software developers’ prior knowledge and several attributes of hackathon context strongly influence the rate and direction of inventive activity at a hackathon.
INTRODUCTION

Temporary gatherings—a term we use to refer to formally organized, synchronous events such as conferences, trade shows, and festivals—create a forum where strangers meet one another, thereby helping people establish new social connections (Feld, 1981). The infusion of new social ties can have important consequences for knowledge exchange and innovation. A growing body of social science research finds temporary gatherings affect a range of important outcomes, including the emergence of cooperation (Rao & Dutta, 2012), network formation (Stam, 2010), knowledge diffusion (Fang et al., 2021), and the initiation of scientific collaborations (Chai & Freeman, 2019; Lane et al., 2021).

Novel projects that emerge from temporary gatherings are sociologically interesting because attendees may hail from distinct social worlds and bring distant knowledge bases to the gathering. This creates an opportunity for recombinant search, which we define as an attempt to create something useful or valuable by combining components in a new way (Fleming, 2001; Schumpeter, 1939). Taking the view that inventions arise from novel combinations of existing knowledge (Galunic & Rodan, 1998), recombinant search is more likely to yield novel inventions when search draws on a knowledge base composed of more diverse and distant knowledge elements (Kauffman, 1993). Recombining more distant knowledge elements also leads to higher variance in the usefulness of resulting innovations (Afuah & Tucci, 2012; Jeppesen & Lakhani, 2010; March, 1991) therefore raising the possibility of breakthrough innovation (Kneeland et al., 2020; Singh & Fleming, 2010).

However, recombining diverse knowledge is difficult (Piezunka & Dahlander, 2015). For two or more social acquaintances’ distinct knowledge elements to be recombined, at minimum one acquaintance must transfer some of their knowledge to another. Transferring tacit or complex knowledge requires social ties that have high carrying capacity for information flow
In network theory, a tie’s information carrying capacity is referred to as its bandwidth (Aral & Van Alstyne, 2011). Prior research finds tie bandwidth is higher when a social tie is stronger (i.e., longer duration, more frequent contact, and subjectively closer) (Brashears & Quintane, 2018). However, acquaintances with a strong social tie rarely possess diverse knowledge because they likely belong to the same social contexts. This leads to a “diversity-bandwidth trade-off,” where weak ties—including ties newly formed at a temporary gathering—tend to be high on knowledge diversity but low on knowledge transfer bandwidth (Aral & Van Alstyne, 2011).

The diversity-bandwidth trade-off therefore poses a compelling theoretical puzzle for recombinant search at temporary gatherings: knowledge diversity at a gathering is high because participants have differing social backgrounds, but those differences—alongside the fact that attendees are often meeting for the first time—imply that knowledge transfer bandwidth could be low. In this research project, we therefore ask when do temporary gatherings catalyze the creation of novel combinations? We employ an empirical context—software development hackathons—where participants are motivated and incentivized to search for useful innovations. We utilize granular data on the hackathon attendees’ software development activities to identify which combinations of components originated at a hackathon. We investigate what factors—those specific to the software developer and those arising from the hackathon context—are associated with the generation of novel software combinations. We supplement our quantitative analyses with a series of qualitative interviews to understand what drives recombinant search at hackathons.

Our findings both affirm and extend prior theory. In line with earlier studies measuring novel combinations (e.g., Uzzi et al., 2013), we find the distribution of innovation at hackathons is highly skewed. Roughly one third of software developers at a hackathon experiment with a
combination of two or more platform technologies that they have not previously combined themselves; two thirds of developers stick to familiar combinations. Roughly one sixth of software developers at a hackathon introduce a technological combination that had not previously been used by anyone who attended the hackathon, i.e., a technological combination that is new relative to the local context. Within this subset of software developers who introduce “new to hackathon” combinations, 20% of developers account for 66% of new combinations. This skewed distribution in innovation suggests that the diversity of knowledge present at a hackathon only helps a limited subset of software developers generate novel combinations.

Nearly all (98%) of the new technological combinations introduced by software developers at a hackathon involve at least one technological component that the developer had not used prior to the hackathon. The developer therefore adopts one or two new components at the hackathon, and immediately uses them to create (novel) technological combinations. We can trace many of these new adoptions to contextual cues that the developer experiences at the hackathon, such as the presence of a platform company as a hackathon sponsor, or the diffusion of knowledge about a component from peer developers at the hackathon.

One main theoretical argument we make in this manuscript is that—for recombinant search at temporary gatherings—the diversity of peers’ knowledge and the communication bandwidth between peers at the gathering act as strategic complements. As a result, software developers with higher capacity to absorb knowledge from peers will gain more from the diversity of peers’ knowledge bases. This allows a limited subset of software developers at a hackathon—those with the greatest technical expertise—transcend the diversity-bandwidth trade-off and immediately act as technology brokers (Hargadon & Sutton, 1997): these developers learn about technological components from peers that they have just met at the hackathon, and immediately recombine them into creative software projects.
LITERATURE BACKGROUND

Diversity, Bandwidth, and Recombinant Search

A large body of innovation research supports the proposition that novelty arises when existing ideas or technologies are assembled into new combinations (Xiao et al., 2021). The generative power of recombination has been documented across a variety of domains, including patenting (e.g., Fleming, 2001; Vestal & Danneels, 2022), academic research (e.g., Leahey et al., 2017; Uzzi et al., 2013), corporate structure (Karim & Kaul, 2015), and social movement tactics (Wang & Soule, 2016). When two existing ideas are combined for the first time, audiences tend to perceive the combination as creative (Godart et al., 2020). The novel combination may garner positive audience evaluations, especially in contexts where novelty is valued for its sake (Phillips, 2011). Audience evaluations of novel combinations tend to exhibit high variance: some novel combinations disappoint, whereas others represent technical or creative breakthroughs (Kneeland et al., 2020; Singh & Fleming, 2010).

Since ideas or technological components—which we will refer to as knowledge elements—are the feedstock for recombinant search, larger initial pools of knowledge elements can be combined in more numerous ways (Fleming, 2001). We follow past literature in treating knowledge elements as cognitive building blocks, akin to atoms, that a person can learn and then assemble into knowledge bundles, akin to molecules. This conceptualization allows us to define individual-level knowledge diversity as the number of distinct knowledge elements a person knows (Page, 2010). It allows us to define collective knowledge diversity as the heterogeneity among the knowledge bases of several people. Thus, an individual with diverse knowledge can produce numerous combinations by themselves. A group with diverse collective knowledge may have some latent capacity to produce many novel combinations, but for that capacity to be realized, the knowledge must be transferred among the individuals (Hansen, 2002).
Much existing research linking knowledge diversity to recombinant search examines actors’ social networks, treating social ties—between individuals or between firms—as conduits through which knowledge can flow (Phelps et al., 2012). Diverse network ties can then be a source of diverse knowledge; for example, managers whose network of contacts exhibits greater heterogeneity tend to exhibit greater innovative performance (Burt, 2004; Rodan & Galunic, 2004). However, the benefits connections to diverse sources of knowledge are only realized under certain contingencies (Reagans & Zuckerman, 2001; Ter Wal et al., 2016). Knowledge elements may exhibit tacitness or complexity (Fleming & Sorenson, 2001; Yayavaram & Ahuja, 2008) making them hard to transfer between individuals (Szulanski, 1996). In turn, social ties may have limited carrying capacity—i.e., bandwidth—for knowledge transfer. Even when collective knowledge diversity is high, if the bandwidth for knowledge transfer is low, there is no assurance that the diverse knowledge will be recombined (Aral & Van Alstyne, 2011).

Much prior research on recombinant search studies innovators as members of formal organizations; the innovators have an established structural position in an intra- and/or inter-organizational network (Xiao et al., 2021). In contrast, very limited research has considered whether and how temporary gatherings—where innovators meet and interact with strangers—facilitate recombinant search. For the reasons laid out in the Introduction, we find this an important omission. In the theory development section that follows, we re-examine how key mechanisms from the recombinant search literature play out in the context of temporary gatherings. We develop four deductive hypotheses which become the focus of our empirical study.
THEORY AND HYPOTHESES

We define temporary gatherings as formally organized, synchronous events that last between several hours and several days.¹ We refer to the individuals who attend a temporary gathering as attendees. Examples of temporary gatherings (and their corresponding attendees) include: academic conferences (research scientists), trade shows (businesspeople), art festivals (artists and art dealers), summit meetings (diplomats), and hackathons (software developers).

Temporary gatherings vary in the extent to which innovation is an explicit goal of the gathering. At some temporary gatherings, such as academic conferences and hackathons, knowledge creation is central to the gathering’s purpose. At other temporary gatherings, recombinant search may manifest in other important ways, such as collaborative problem-solving by buyers and suppliers at a trade show (Von Hippel & Von Krogh, 2016) or integrative (i.e., positive sum) bargaining at a summit meeting (Schüssler et al., 2014). Our theory development assumes that attendees at a gathering share at least some of their knowledge with one another, but in other respects our theory development is agnostic to the explicit goals of a given gathering.

Knowledge Reuse at Temporary Gatherings

A natural starting point in a study of recombinant search is to examine the existing knowledge base of the actor that undertakes search. New knowledge combinations often re-use an actor’s pre-existing knowledge: the actor could combine two or more familiar knowledge elements into a novel combination or combine one familiar knowledge element with one or more newly acquired knowledge elements (Capaldo et al., 2017). Referred to in prior research as search depth (Katila & Ahuja, 2002), the re-use of prior knowledge in new combinations helps

¹ This definition includes both in-person and virtual events. Our theory development assumes open mixing of attendees at an event, which is not always afforded in contemporary virtual events, but it is in principle possible.
innovators generate a higher volume of ideas. A direct corollary of the tendency for innovators to reuse knowledge elements they are familiar with is that innovators with greater knowledge diversity (i.e., a larger repertoire of knowledge elements) have greater capacity to generate novel combinations at a temporary gathering.

\[ H1. \text{An attendee’s prior familiarity with a knowledge element raises the likelihood that the attendee will use the knowledge element in a novel combination at a temporary gathering.} \]

The Knowledge Frontier at a Temporary Gathering

Prior research uses the term knowledge frontier to refer to the boundary between what is already known—by a given set of people working within a given field—and what remains to be discovered (Boudreau et al., 2016). To define a knowledge frontier we need to circumscribe the set of people we consider relevant to our definition. The knowledge frontier can therefore advance in two ways. First, the people in the field can create new knowledge, often by combining existing knowledge that lies within the frontier in a new way. For example, Andrew Wiles proved Fermat’s Last Theory by combining the existing Ribet’s theory with a then-recently discovered Euler system. Second, expanding the set of people in the field can expand the knowledge frontier. For example, the knowledge frontier in the field of theoretical mathematics can be defined by the population of research active mathematicians contributing to the international scholarly literature: an influx of mathematicians from the (formerly isolated) Soviet Union pushed the knowledge frontier in theoretical mathematics forward (Teodoridis et al., 2019).

To analyze a temporary gathering, we can define a local knowledge frontier based on the pre-existing knowledge of the set of people who attend the temporary gathering. Figure 1 depicts three knowledge sets from the perspective of a given attendee. The inner most knowledge set,
\{A\}, is the attendee’s own knowledge base. Each attendee’s knowledge base is nested within the collective knowledge base of all attendees, \{B\}. The edge of the attendees’ collective knowledge base is the knowledge frontier for this specific temporary gathering. The attendees’ collective knowledge base is nested within a broader space of latent knowledge to be discovered. This broader knowledge space can be defined, \textit{ex post}, by an analyst, but is by definition not known to the gathering attendees at the time of the temporary gathering.\(^2\)

*** Insert Figure 1 about here ***

Taking a knowledge-frontier perspective assists our understanding of recombinant search at a temporary gathering in two ways. First, it helps us to define how novel is a given knowledge combination made by a given attendee. The combination may be new-to-the-attendee yet lie within the knowledge frontier of other attendees. Alternately, it may be new-to-the-attendees and also lie outside the gathering’s knowledge frontier. In this case—which we refer to as a \textit{locally new combination}—the knowledge is new relative to the collective the knowledge base of all attendees at the gathering.

Second, the idea that a gathering can have a locally defined knowledge frontier allows us to conceptualize a given attendee’s \textit{distance} from the gathering’s knowledge frontier. A software developer who is further behind the local knowledge frontier may experiment with combinations that are new to the developer (i.e., undertake search that, from the developer’s perspective is exploratory) without contributing knowledge that is new relative to the broader hackathon context. In line with the idea that proximity to the knowledge frontier makes it easier for an innovator to make novel combinations that expand the frontier (Teodoridis et al., 2019), we

\(^2\) Knowledge that lies outside the knowledge frontier but proximate to it has been labeled the “adjacent possible” by complexity theorists such as Kauffman (Felin et al., 2014).
expect attendees at temporary gatherings to make more locally new combinations the closer they are to the local knowledge frontier at the gathering.

H2. Attendees whose pre-existing knowledge base places them closer to the local knowledge frontier tend to generate a larger number of locally new combinations at a temporary gathering.

Knowledge Diversity at Temporary Gatherings

Temporary gatherings are social foci (Feld, 1981) that draw together people from distinct social worlds. Any given pair of attendees inherently—by virtue of their attendance—share an interest in whatever topic the temporary gathering is about (Feld & Grofman, 2009). But aside from this one shared interest, the pair of attendees may differ on myriad other dimensions (e.g., demographic attributes, knowledge base, network milieu). Temporary gatherings bring together people from distant network neighborhoods. Despite potential reluctance among some attendees to mix with strangers (Ingram & Morris, 2007), the design of most temporary gatherings encourages the formation of instrumental network ties between strangers (Stam, 2010). These new encounters between people of differing backgrounds create a lot of latent potential for knowledge recombination. For example, Chai and Freeman (2019) find that attendees of a scientific conference who lack pre-existing coauthor ties to other attendees tend to subsequently initiate coauthoring relationships. Foerderer (2020) finds that software developers who attended Apple’s developer conference subsequently increased the pace and quality of app updates, suggesting the developers absorbed knowledge that catalyzed innovation.

We expect that the collective knowledge diversity of peers at a temporary gathering can enhance recombinant search by a focal attendee. The focal attendee may acquire knowledge elements from peers at a temporary gathering through a diffusion process (Fang et al., 2021).
Alternatively, peers’ knowledge might act as a contextual cue that reminds the attendee to re-use a given knowledge element that they had used at some point in the past.

_H3. The collective familiarity of other attendees with a given knowledge element raises the likelihood that a focal attendee will use the knowledge element in a novel combination at a temporary gathering._

**Individual-level Bandwidth at Temporary Gatherings**

Temporary gatherings bring together people with distant knowledge bases, and this knowledge distance can make transfer—and subsequent recombination—difficult. Where knowledge is distant, an individual may lack absorptive capacity (Cohen & Levinthal, 1990); where knowledge is tacit or complex there may be insufficient time or communication bandwidth to transfer it within the limited duration of the temporary gathering (Aral & Van Alstyne, 2011).

For two reasons, we expect individuals with greater technical expertise to be best placed to harness diverse collective knowledge at a temporary gathering. First, individuals with greater technical expertise possess cognitive frameworks that let them absorb new knowledge elements more readily (Cohen & Levinthal, 1990), i.e., the expertise effectively provides the individual with greater bandwidth for the absorbing knowledge from peers at a temporary gathering. Consistent with this argument, prior work finds that encounters between scientists at a temporary gathering fostered subsequent grant co-applications, knowledge transfers, and coauthor relationships more frequently when the scientists had overlapping knowledge bases (Boudreau et al. (2017); Lane et al., 2021).

Second, due to the combinatorial nature of innovation, an individual who absorbs a new knowledge element opens up a larger space of combinatorial possibilities when they start with a
larger knowledge base. Latent opportunities for novel combinations of knowledge elements rise geometrically with the size of the underlying knowledge base: just as more surface area is added when a pizza goes from 14” to 16” than when it goes form 10” to 12,” more novel combinations become possible when adding a knowledge element to a large knowledge base than to a small one.

\[ H4. \] The association between the collective familiarity of other attendees with a given knowledge element and the likelihood that a focal attendee will use the knowledge element in a novel combination is stronger for focal attendees who are technical experts.

**Contextual Sources of Bandwidth at Temporary Gatherings**

Temporary gatherings are—by definition—limited in duration: this limits the amount of knowledge transfer that can take place at a gathering. The pressures of innovating under the tight timeframe of a hackathon were investigated by Lifshitz-Assaf and colleagues (2021), who found that finding that classical, structured innovation processes backfire when applied to a temporary gathering context. Contextual limitations on bandwidth likely limit the effectiveness with which collective knowledge diversity at a temporary gathering generates novel combinations.

This raises the question of what contextual conditions at temporary gatherings might effectively catalyze generative knowledge transfers. We suggest that contextual factors including hackathon duration and the spatial density of participants influence how effective a temporary gathering is at generating novel combinations. We expect that longer gatherings result in greater capacity for knowledge transfer and therefore allow attendees to better harness collective knowledge diversity (NB. Analysis for this hypothesis is still TBD at the time of writing).
**H5.** *The association between the collective diversity of other attendees’ knowledge bases and the number of novel combinations a focal attendee generates is stronger for hackathons that are longer in duration.*

**EMPIRICAL SETTING**

**Recombinant Search in Software Development**

To write an application, software developers build on platform technologies that serve as inputs to the application. To access these inputs—such as cloud computing resources, datasets, and algorithms—developers write code that “calls” on web-based Application Programming Interfaces (APIs). Common APIs used by software developers include Amazon Web Services (cloud computing), Google Maps (mapping services), and PayPal (payment services). Simple pieces of software might solve a simple problem by drawing on just one API; for example, a note-taking App might simply store typed notes in the Cloud by calling on the Amazon Web Services API. More complex pieces of software combine the capabilities of multiple APIs in a complementary way. For example, a ride-hailing App such as Uber would need to call on APIs for cloud computing (e.g., AWS), mapping services (e.g., Google Maps), payment services (e.g., PayPal), and texting (e.g., Twilio).

Innovation in software development is therefore a recombinant search process in which software developers experiment with combinations of APIs to discover which combinations unlock complementarities that generate useful outputs (see also, Haefliger et al., 2008, and Boudreau, 2012). Because APIs are integral to modern software development, they are a large and growing component of the digital economy. API platforms allow software developers to

---

3 Developers also call hardware APIs to access computing resources and sensor inputs from the device the software will run on. We do not study hardware APIs in this project; we use the acronym API to refer to web-based APIs.
experiment with the API by making a low volume of calls for free; API platforms charge fees for high volumes of calls. These fees are a substantial revenue source for platform companies, e.g., Twilio makes most of its $2.8bn revenue from API fees.4

Our empirical study treats an API as a knowledge element for a software developer. Like many other forms of knowledge, learning how to use an API requires some up-front effort on behalf of the software developer. While there is some explicitly documented knowledge about how to use an API (e.g., on the API owner’s website), much of the necessary knowledge is more tacit: it is either learned through experience or learned via training by someone knowledgeable about the API. Once a developer has used an API at least once, we treat the API as an element in the developer’s knowledge base. In this way, software development resembles other contexts—such as patenting and scientific publication—where an individual exerts effort to learn various knowledge building blocks, and researchers make inferences about the individual’s knowledge base by measuring their creative outputs.

**Software Development Hackathons**

Software development hackathons are temporary gatherings, usually 12 to 72 hours in duration, where the attendees engage in focused work to write novel software programs. Hackathons are notionally structured as competitions, with a variety of prizes on offer: general prizes reward judges’ perception of overall innovation and sponsor-specific prizes reward the innovative use of a sponsoring company’s API. In our interviews with hackathon attendees, most reported that the possibility of winning prizes was only ever a minor aspect of the motivation to attend a hackathon. Most attendees reported that they are motivated to attend hackathons by the

---

4 Revenue figure is 2021 financial year; Twilio’s market capitalization is $26.7bn as of 18 March 2022.
opportunity to build something original and to undertake creative work in a socially energizing setting.

*Qualitative Accounts of Software Development at Hackathons*

To supplement the quantitative evidence we report below, we conducted qualitative interviews with 30 different hackathon stakeholders between May 2019 and August 2019. We interviewed all types of stakeholders—hackathon organizers, sponsors, and developers—who would have first-hand and observational experience of the development activity at hackathons. The interviewees developed software or observed developers at several hundred hackathons. One developer described his experience building a movie-phone application that combined multiple platforms:

*We built a movie-phone version of Instagram that used machine learning to describe the photos on someone's Instagram feed. It was pretty silly; it was obviously meant to be a joke... You dialed a phone number and then it describes a someone's Instagram feed to you [using computer vision] and you can press one to like [a post], and to comment [on a post], you can recite a comment and we'll write it down [using speech recognition]. We did all that with Twilio's API, and they were one of the sponsors.*

To build his application, the developer combined multiple platforms. First, the underlying data came from the Instagram API, but then he needed to use an Amazon Web Services API to do image recognition of the photo-centric Instagram data. From there, to deliver that data to the end user over the phone, he used the Twilio API to host the phone call with the end user, and the Twilio API allowed the user to engage by recognizing touchtone inputs and automatically transcribing spoken word.

Another developer described the enjoyment that came from experimenting with combining APIs:
The first time I used Twitter’s API was probably at a hackathon. There was also a company that does APIs for machine learning. It would do fairly simple stuff: if you sent it a picture, it would tell you what was in the picture, or if you sent it some text, it would give you the sentiment analysis, stuff like that. It was fun enough to play with, and it made it easy to string together a few other ideas into a pretty fun app.

One developer described the difficulty of pursuing novel combinations:

If you’re going to go for an obscure API that nobody is really using, it is also obscure for a reason. How many apps can you really build to incorporate a healthcare data API or like an insurance API? It is fairly difficult. It’s harder to shoehorn it, to use that expression. You can’t just like add it on. Like everyone can add a feature that uses Twilio and sends you a text message, but it’s very hard to use a feature that files an insurance claim in an unrelated app.

To better understand how software developers undertake recombinant search at hackathons, we undertake a quantitative study drawing on the code of a large sample of hackathon projects.

METHODS

Our quantitative dataset is constructed from the historic portfolios of software development projects posted on the website GitHub by a sample of 1,302 software developers who attended one of 167 software development hackathons between 2014 and 2017. We collect hackathon-level data from DevPost.com and developer-level data from GitHub and LinkedIn. From an initial sample of 1,287 hackathons, we exclude hackathons that have a single sponsor, those that took place virtually, and those with fewer than ten attendees.

We apply automated text analysis to the developers’ source code to identify which platform’s APIs each GitHub project refers to. While the universe of web-based APIs is very large, we construct our dataset focusing on the 29 most frequently used APIs. We update the
panel monthly, so our raw data is at the developer-platform-month level of analysis. We define a developer’s platform stock as the set of all APIs they have ever adopted up that point. The size (i.e., cardinality) of a developer’s platform stock therefore increases monotonically over time.

We define the developer’s knowledge base at the level of pairwise combinations of platforms. Specifically, the knowledge base for developer $i$ at time $t$, which we denote $K_{i,t}$, is defined as the set of all pairs of APIs that the developer has used together in the same month prior to and including month $t$. The developer’s knowledge base is time varying and, by definition, is monotonically (weakly) increasing in size over time. The combinatorial space of all possible pairs of platforms in our dataset, which we denote with $S$, is not time varying. Because we limit our study to 29 distinct APIs, the size of the combinatorial space is 406 ($= 29 \times 28 / 2 = |S|$) pairwise combinations of platforms. Each developer in our dataset attends one hackathon. The set of developers who attended hackathon $h$ is denoted $H_h$. The month in which hackathon $h$ occurred is denoted $m_h$.

**Dependent Variables**

**New-to-developer combinations.** This variable, specified at the developer-month level of analysis, refers to the count of new pairwise combinations that a developer uses for the first time in a given month. Formally, in set-theoretic notation, the new-to-developer combinations for developer $i$ in month $t$ is $|K_{i,t} \setminus K_{i,t-1}|$.

Figure 2 plots a time series of the mean of new-to-developer combinations, where the x-axis plots time relative to the developer’s hackathon month. In a typical non-hackathon month, a developer adds, on average, 1.7 platform combinations to their knowledge base. In a hackathon month, a developer adds, on average, 10.8 platform combinations to their knowledge base. This provides some initial evidence that hackathons are fertile forums for a given developer to try out platform combinations that are new to them.
However, most of these new-to-developer combinations refer to pairwise platform combinations that some other developer at the hackathon might have already used in the past. To study the generation of novel combinations at hackathons, we need a more restrictive definition of novelty that accounts for whether a given pairwise combination was used by any attending developer prior to the hackathon. This leads to our second dependent variable:

**Locally new combinations.** These are pairs of APIs used together by a developer in a hackathon month that have not been previously used together in one month by any of the hackathon attendees. Formally, in set-theoretic notation, the set of locally new combinations created by developer $i$ at hackathon $h$ is denoted $n_{i,h}$:

$$n_{i,h} \equiv K_{i,m_h} \setminus \bigcup_{j \in H_h} K_{j,m_{h-1}}$$

We analyze locally new combinations at the developer-API pair level of analysis. To analyze whether a given developer combines a given pair of platforms for the first time at a hackathon, we need to define the set of platform pairs that are ‘at risk’ of being combined for the first time; this risk set excludes all the platform pairs that have previously been combined by hackathon attendees, i.e.

$$\text{riskset}_h = S \setminus \bigcup_{j \in H_h} K_{j,m_{h-1}}$$

**Innovation outcomes.** Our main goal in this study is to understand what drives the generation of new technological combinations at software development hackathons. To help validate our main dependent variable, **Locally new combinations**, we analyze whether hackathon projects that contain locally new combinations are evaluated more favorably by hackathon judges. We define a binary variable, **Prize-winning project**, which is set to one for projects that win one of the general prizes at a hackathon. This measure captures the judges’ subjective
perception of whether the project displayed the greatest overall creativity or innovativeness amongst all the projects submitted at the hackathon.

**Independent Variables**

*Focal individual’s familiarity with knowledge elements.* In our analyses at the developer-API pair level of analysis, we measure whether the two focal APIs belong to the focal developer’s pre-hackathon knowledge stock using two binary variables. *Developer knows one API* is set to one if the developer’s pre-hackathon knowledge stock contains one API in the pair (and zero otherwise); *Developer knows both APIs* is set to one if the developer’s pre-hackathon knowledge stock contains both APIs in the pair.

*Proximity to knowledge frontier.* We define the variable *Developer proximity to local knowledge frontier* by dividing the size of the risk set of combinations that are new to the hackathon, $|S \setminus \bigcup_{j \in H_h} K_{j,m_h-1}|$ by the size of the risk set of combinations that are new to the developer, $|S \setminus K_{i,m_h-1}|$. This variable takes values between zero and one, where a value of one corresponds to a developer whose knowledge base is at the local knowledge frontier (i.e., the risk set that is new to the hackathon perfectly overlaps with the risk set that is new to the developer).

*Collective familiarity of other attendees with a given knowledge element.* For the two APIs in the focal API pair, we calculate the proportion of attendees at the hackathon who have the API in their pre-hackathon knowledge stock, the *Peers’ API adoption %.* In line with how dyadic variables are calculated for unordered dyads in the network analysis literature, we define two API-pair level variables capturing the *lower* of the two proportions (i.e., the minimum) and the *greater* of the two proportions (i.e., the maximum). These variables measure the extent to
which collective familiarity with the APIs in the pair is associated with the creation of novel combinations that include APIs in the pair.

**Moderating Variable**

*Developer expertise.* In order to categorize the software developers according to their level of expertise, we collect additional data on the developers’ educational backgrounds (e.g., highest degree level; whether they have a computer science degree) and work history (e.g., employment as a computer scientist) from their LinkedIn profiles. We also measure the developers’ overall levels of activity on GitHub, including their total number of projects. To reduce this high-dimensional data into a parsimonious variable that captures developer expertise, we apply Latent Class Analysis. This clustering-type method classifies observations into a discrete set of categories that minimizes differences within categories and maximizes differences between categories; it is well suited to dimensionality reduction tasks when some of the input data are categorical in nature. We find that a three-class model provides a strong fit to our developer-level dataset, and we label the three classes *novice, intermediate,* and *expert* level developers. We include dummy variables for these developer categories as control variables in our regression, and we use the *Expert developer* dummy as moderating variable that we interact with the *Peers API adoption %* variables.

**Control Variables**

The presence of the focal APIs’ platform owners as sponsors of the hackathon might act as a contextual cue and an incentive for the developer to employ the API in their hackathon project. We therefore measure whether the two focal APIs’ platform owners have a presence as sponsors of the hackathon using two binary variables, *One API is sponsor* and *Both APIs are sponsors*. Sponsoring companies often send employees to hackathons to act as technology
evangelists; for example, the employees might run coaching sessions to lower the learning costs for a developer adopting an API.

**PRELIMINARY RESULTS**

**Prize-winning Projects**

First, we validate our main dependent variable by showing that hackathon projects that contain locally new combinations are more likely to win hackathon prizes, indicating that novel technological combinations are evaluated favorably by an important audience (i.e., the judges) in the hackathon context. In a cross-sectional analysis of hackathon projects, there is a strong bivariate correlation between *Prize-winning projects* and *Locally new combinations*. Nine percent of the projects in our sample win a general hackathon prize. Amongst these prize-winning projects, 30.6% contain at least one locally new combination. Amongst non-winning projects, by comparison, just 15.8% contain at least one locally new combination. This correlational result helps validate that our main dependent variable, *Locally new combinations*, is a meaningful measure of innovation as perceived by hackathon judges.

**Locally New Combinations**

Table 1 reports results of preliminary regression analyses predicting *Locally new combinations* at the developer-API pair level of analysis. Each observation refers to an API pair that no developer at the hackathon has previously combined, meaning the API pair is at risk of being combined for the first time by the focal developer in their hackathon project. Table 1 reports one control variable model and four hypothesis testing models.

*** Insert Table 1 about here ***
The positive and statistically significant coefficient on *Developer knows one API* supports hypothesis 1. The coefficient indicates that a developer is more likely to recombine an API when the developer has previous familiarity with the API prior to the hackathon. To understand this result more clearly, we classify all the locally new combinations produced at hackathons according to whether one API, both APIs, or neither API had been used by the focal developer prior to the focal hackathon. We found that only 2% of the locally new combinations involve pairs of APIs where both are known to the developer prior to the hackathon. In 52% of locally new combinations, the developer combines one familiar API with one API that they newly adopt at the hackathon. In 46% of locally new combinations, the developer combines APIs that are both adopted for the first time at the hackathon. This suggests that knowledge elements that are newly adopted at the hackathon itself are a crucial input to recombinant search taking place at a hackathon. Contextual cues and peer-to-peer learning may therefore strongly impact which knowledge elements get recombined at temporary gatherings.

The positive and statistically significant coefficient on *Developer proximity to local knowledge frontier* in models 3 and 4 supports hypothesis 2. We note that this result holds even though we are controlling for the developer’s pre-existing knowledge over one or both APIs in the focal API pair. Thus, over and above their knowledge about specific APIs, proximity to the knowledge frontier helps developers identify combinations that are new relative to the local context. Developers further behind the frontier may be experimenting with knowledge combinations novel to them personally (i.e., “new-to-developer” combinations) but more of those combinations had already been previously used by another hackathon attendee.

Hypothesis 3 predicts that a focal developer is likely to recombine knowledge elements familiar to peers at the hackathon. The positive and significant coefficient on the variable *Peers’ API adoption % (lower of two)* in model 3 supports hypothesis 3. Pairs of APIs are more likely to
be recombined by a focal developer when both APIs are found within the knowledge stocks of peers at a hackathon. Interestingly, it is not sufficient if just one of the APIs is found within peers’ knowledge stock. Our interpretation of this coefficient is that software developers are being socially influenced to recombine pairs of APIs that are salient because peer developers are using them at the hackathon.

Hypothesis 4 predicts that expert developers are most sensitive to the social influence predicted in hypothesis 3. Model 4 in table 1 provides support for hypothesis 4. The coefficient on the interaction term Expert developer * Peers’ API adoption % (lower of two) is positive and statistically significant, while the coefficient on the main effect of Peers’ API adoption % (lower of two) in this model is smaller than in model 3 and indistinguishable from zero. This suggests that only expert developers—not novice or intermediate skill developers—are influenced to recombine APIs by their peers.

Turning to the control variables, table 1 reports that hackathon sponsors are sources of knowledge elements for a focal developer to recombine. This raises an interesting possibility that hackathon organizers (who, in our sample, are often volunteers motivated by an enthusiasm for technology) may be inadvertently affecting the direction of innovation when they solicit sponsorship from platform owners. Two platform owners who, independent of one another, accept an invitation to sponsor a given hackathon are disproportionately likely to have their APIs recombined in a hackathon project.

**DISCUSSION**

The picture emerging from our empirical analysis suggests that to generate locally new combinations at a temporary gathering, a large pre-existing stock of knowledge elements is not sufficient. The developers should also possess the capacity to quickly absorb and recombine newly acquired knowledge elements at the gathering itself, drawing on contextual stimuli such as
peer developers’ expertise and the presence of technical experts brought in by sponsors of the temporary gathering. The software developers in our study are engaging in technology brokering, in the sense described by Hargadon and Sutton (1997), but—intriguingly—they are doing so without the infrastructure of organizational routines and memory.

**Temporary Gatherings as a Generative Institutional Arrangement**

We have argued in this study that temporary gatherings are a theoretically distinctive and important phenomenon. We view temporary gatherings as an important institutional arrangement (i.e., an approach to coordinating economic activity, see Williamson (1991)) that supports recombinant search by both encouraging diverse knowledge bases to mix, while enabling sufficient knowledge transfer bandwidth for attendees to discover new combinations.

The two best studied institutional arrangements supporting recombinant search are formal organizations and interorganizational networks. Formal organizations, such as corporations and universities, support recombinant search by selecting members with specialized knowledge bases and creating structures for their collaboration. Formal organizations provide social infrastructure for repeated collaborations, which creates bandwidth for the transfer of complex knowledge (Hansen et al., 2005). Formal organizations also allow routines to emerge for storing and retrieving diverse ideas in an organization-level “memory” (Jain & Kogut, 2014; Walsh & Ungson, 1991). This facilitates technology brokering, in which solutions to past technical problems accumulate and get recombined during the search for solutions to newly surfaced problems (Hargadon & Sutton, 1997).

Interorganizational networks have long been recognized as an important locus of innovation. Alliances allow employees from different firms to collaborate over sustained periods of time (Ahuja, 2000; Powell et al., 1996; Schilling & Phelps, 2007). The allied organizations bring distinct knowledge bases to the collaboration, providing the requisite diversity of
knowledge that allows new combinations to arise (Phelps et al., 2012). Innovation alliances often employ governance structures—such as equity joint ventures—that allow deep collaboration between the firms’ employees (Mowery et al., 1996). Recombinant search in interorganizational networks is facilitated by the macro-level structure of ties: the network often takes on a “small world” topology (Baum et al., 2003; Gulati et al., 2012) which can sustain knowledge diversity even when knowledge gets transferred across the ties (Fang et al., 2010; Lazer & Friedman, 2007).

Our study puts forward temporary gatherings as a distinctive institutional arrangement that helps innovators overcome the diversity-bandwidth trade-off and engage in recombinant search. We have shown that software development hackathons—one prominent form of temporary gathering—act as melting pots that catalyze novel combinations in a way that is not deterministic but may be more predictable than previously thought. Future research is needed to understand whether these findings generalize beyond software developments to other types of temporary gatherings.
References


Figure 1. Nested sets of attendee knowledge and collective knowledge in a broader latent knowledge space

Notes: The three sets are weakly nested, such that \( \{A\} \subseteq \{B\} \subseteq \{C\} \). The choice to visually depict the knowledge space using squares is intentional: in our empirical analysis we define the search space as pairwise combinations of technological components, so that with \( n \) underlying technological component, the size of \( \{C\} \) is \( n^*(n-1)/2 \).
Figure 2. Time series of new-to-developer combinations

Mean of *New-to-developer combinations*

Months relative to hackathon
Table 1. Linear regression of *Locally new-combinations* created at hackathon; developer API-pair level observations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Controls</th>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
<th>Model 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate developer†</td>
<td>-0.003</td>
<td>-0.002</td>
<td>-0.002</td>
<td>-0.002</td>
<td>-0.002</td>
</tr>
<tr>
<td></td>
<td>(0.004)</td>
<td>(0.004)</td>
<td>(0.004)</td>
<td>(0.004)</td>
<td>(0.004)</td>
</tr>
<tr>
<td>Expert developer†</td>
<td>0.017**</td>
<td>0.011+</td>
<td>0.010+</td>
<td>0.010+</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>(0.006)</td>
<td>(0.006)</td>
<td>(0.006)</td>
<td>(0.006)</td>
<td>(0.006)</td>
</tr>
<tr>
<td>One API is sponsor</td>
<td>0.005+</td>
<td>0.005+</td>
<td>0.005+</td>
<td>0.005+</td>
<td>0.005+</td>
</tr>
<tr>
<td></td>
<td>(0.003)</td>
<td>(0.003)</td>
<td>(0.003)</td>
<td>(0.003)</td>
<td>(0.003)</td>
</tr>
<tr>
<td>Both APIs are sponsors</td>
<td>0.015*</td>
<td>0.014*</td>
<td>0.014*</td>
<td>0.014*</td>
<td>0.014*</td>
</tr>
<tr>
<td></td>
<td>(0.007)</td>
<td>(0.006)</td>
<td>(0.006)</td>
<td>(0.006)</td>
<td>(0.006)</td>
</tr>
<tr>
<td>Developer knows one API</td>
<td></td>
<td>0.024**</td>
<td>0.024**</td>
<td>0.022**</td>
<td>0.022**</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.004)</td>
<td>(0.004)</td>
<td>(0.004)</td>
<td>(0.005)</td>
</tr>
<tr>
<td>Developer knows both APIs</td>
<td></td>
<td>0.030+</td>
<td>0.030+</td>
<td>0.023</td>
<td>0.014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.017)</td>
<td>(0.017)</td>
<td>(0.015)</td>
<td>(0.014)</td>
</tr>
<tr>
<td>H1 Developer proximity to local knowledge frontier</td>
<td></td>
<td>0.014+</td>
<td>0.018*</td>
<td>0.017*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.009)</td>
<td>(0.008)</td>
<td>(0.008)</td>
<td></td>
</tr>
<tr>
<td>H2 Peers’ API adoption % (lower of two)</td>
<td></td>
<td></td>
<td></td>
<td>0.062*</td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.025)</td>
<td>(0.013)</td>
</tr>
<tr>
<td>H3 Peers’ API adoption % (higher of two)</td>
<td></td>
<td></td>
<td></td>
<td>0.007</td>
<td>-0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.007)</td>
<td>(0.006)</td>
</tr>
<tr>
<td>H4 Expert developer * Peers’ API adoption % (lower of two)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.171**</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.057)</td>
<td></td>
</tr>
<tr>
<td>H4 Expert developer * Peers’ API adoption % (higher of two)</td>
<td></td>
<td></td>
<td></td>
<td>0.021</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.015)</td>
<td></td>
</tr>
<tr>
<td>No. of observations</td>
<td>216,057</td>
<td>216,057</td>
<td>216,057</td>
<td>216,057</td>
<td>216,057</td>
</tr>
<tr>
<td>Adj. R²</td>
<td>0.005</td>
<td>0.011</td>
<td>0.012</td>
<td>0.013</td>
<td>0.015</td>
</tr>
</tbody>
</table>

† Dummy variables for developer type. The omitted category is *novice developer*.
Robust standard errors in parentheses, multi-way clustered by developer and both APIs in pair.
** p<0.01, * p<0.05, + p<0.1