Creating the List of App Companies

To investigate the Danish app development ecosystem, we began with a list of 33 companies identified through prior ethnographic research. These served as our first-tier companies and provided the foundation for a snowball sampling strategy aimed at uncovering additional relevant actors.

Using GitHub's advanced search function, we queried each company to identify user profiles potentially affiliated with them. These profiles, retrieved through keyword and location-based queries, comprised our first-tier users.

For companies with common or international names – such as PWC – initial searches returned thousands of results, only a fraction of which could be confidently linked to the company. To improve precision, we applied geographic filters to the search using GitHub's query qualifiers (see: https://docs.github.com/en/search-github/searching-on-github/searching-users), incorporating keywords for the 10 biggest Danish cities, and variants of "Denmark," "Copenhagen". For instance, our query for Netcompany looked as the following:

"netcompany location:dk location:Denmark location:Danmark location:CPH location:KBH location:Copenhagen location:Cop location:København location:Odense location:Aarhus location:Århus location:Aalborg location:Ålborg location:Esbjerg location:Randers location:Kolding location:Horsens location:Vejle location:Roskilde location:Herning"

In a few cases – such as FrontIt, Greener Pastures, and Adform – no users were returned when location filters were applied. In these instances, we removed the filters to identify at least one affiliated user per company (which we manually verified). Although this reduced geographic certainty, it was deemed necessary for initiating our snowball sampling process.

From the first-tier users, we identified second-tier users by examining their GitHub activity and connections. We manually reviewed the biographical data of these second-tier users to identify any mention of additional companies. New companies were added to our final sample if they met two criteria: (1) they were Danish or had a Danish branch, and (2) they were involved in app development or related data infrastructure services.

This iterative process led to the identification of 31 additional companies. Each was queried in the same manner as the original set, producing further tiers of users and expanding our sample.

Detailing the Scraping of Users from the GitHub REST API

For each user, we retrieved two categories of data from GitHub's REST API: (1) biographical information and (2) relational data based on user activity. Biographical fields included login, type, company, email, location, bio, and blog.

Except for location, which was used to infer geographic affiliation, these were free-text fields that helped us identify connections to specific companies. All data were drawn exclusively from public repositories and publicly visible profiles.

Relational data represented ties between users through four types of activity: starring, watching, forking, and following. While following is directed at other users, the remaining actions target repositories. For each user, we collected both inbound (interactions received) and outbound (interactions initiated) connections. These ties formed the basis of the network used in our analysis.

Data collection was conducted using the PyGithub package (v1.58.0) [documentation: https://pygithub.readthedocs.io/en/stable/introduction.html], a Python wrapper for GitHub's REST API v3 [https://docs.github.com/en/rest?apiVersion=2022-11-28]. The API imposes rate limits based on authentication status: up to 30 search requests per minute and 5,000 general requests per hour with an authenticated personal access token. To comply with these limits, we implemented automated pauses to align with reset intervals.

An overview of the GitHub data collection workflow is provided in the flowchart below:

% Insert figure

Sorting for Danish App Company Profiles

All user profiles were filtered based on two criteria: (1) geographic location in Denmark and (2) affiliation with a company from our sample. For first-tier users, only company affiliation was checked, as geographic location had already been addressed during the initial GitHub search.

Geographic filtering proceeded in two steps. First, a regular expression (regex) was applied to all biographical fields to match against a predefined list of Danish city names (as used in the GitHub queries described earlier). If this step yielded no match, and the user had a non-empty location field, the location string was evaluated using the GeoNames database (https://www.geonames.org/). Users that matched in either step were categorized as based in Denmark; those without a match were excluded from the sample.

To determine company affiliation, each company name was matched against the biographical fields of the user profile—including the company, bio, and email fields—using a company-specific regex. These regex patterns were maintained in a dedicated script (resources/regex_company_patterns.py). In cases where a single user profile matched multiple company patterns, company affiliation was manually reviewed and assigned.

% Gør det her? Subsequently, all the cases with an inferred company were reviewed manually to check for false positives. In total, 19 out of 480 (~4%) DK-based users turned out to be false positives. In the end, 461 users are categorized as organizational- or employee profiles associated with one of 71 app companies, and they are referred to as Danish app profiles.

Creating the Networks

For each user in the dataset, inbound and outbound ties were recorded for four types of GitHub actions: forking, starring, watching, and following. This produced eight directional tie variables: forks_in, forks_out, stars_in, stars_out, follows_in, follows_out, watches_in, and watches_out. Each entry in these variables is a list of dictionaries, where each dictionary represents a connection and includes the creation time, the name of the targeted repository (if applicable), and the GitHub username of the counterpart involved in the interaction.

To construct the edgelists, we first created a unique list of Danish GitHub users affiliated with companies in our sample. We then looped over each user's activity data across the eight tie types. For each connection, a lookup was performed to check whether the other user was in the unique user list. If so, a directed tie was added to the edgelist, annotated with metadata including: company affiliations of both users, whether the tie was intra- or inter-company, the action type, repository name, and timestamp.

After removing duplicates, the dataset was split into two distinct edgelists:

- A collaboration network, composed solely of fork ties
- An attention network, composed of stars, watches, and follows

Each edgelist was then transformed into a weighted user-to-user edgelist. For every directed pair of users, the number of observed ties in that direction was used as the weight. In the attention network, the three action types were combined into a single edge, but separate counts for each type (star, watch, follow) were stored as edge attributes.

Finally, both user-level networks were aggregated into company-level weighted edgelists. In these, edge weights represent the number of unique, directed user-to-user ties occurring between two companies. Counts of the underlying GitHub actions were also aggregated as edge attributes. Self-loops were allowed to capture intra-company ties and understand internal interaction patterns.

These two company-level edgelists form the basis for the directed network analyses presented in the main analysis section.

An overview of the edgelist construction process is provided in the flowchart below:

Plotting the networks and calculating network metrics

To visualize the collaboration and attention networks, we used the Python library matplotlib, the standard plotting tool in conjunction with the NetworkX network analysis package.

In the plotted networks, nodes are colored according to the four company categories:

1. Digital and marketing consultancies
2. Bespoke app companies
3. Data brokers and infrastructure providers
4. Companies with app-related services/products as part of a broader offering

Edge thickness is proportional to edge weight: higher weights correspond to thicker lines. Self-loops are included to represent intra-company interactions.

For layout generation, we used the Fruchterman–Reingold algorithm, a force-directed layout algorithm implemented in NetworkX. This algorithm simulates a physical system where nodes repel each other like particles, while edges act as springs pulling connected nodes together. The simulation iterates until a stable equilibrium is reached, resulting in a layout that is typically more interpretable than deterministic or grid-based methods.

Two key parameters control the layout:

- k: determines repulsion strength between nodes. A higher value produces a more spread-out graph. We set k = 1.5 for the attention network and k = 1.1 for the collaboration network.
- iterations: controls the number of iterations used to compute equilibrium. We set this to 50, which was sufficient given the relatively small size of the networks.

In addition to the visualizations, we report several summary metrics:

- Number of users: Unique GitHub users in the network
- Number of companies: Unique companies represented by users
- Unique user-to-user edges (directed): Includes self-loops
- Total inter-company GitHub actions: Sum of GitHub actions where users belong to different companies
- Total intra-company GitHub actions: Sum of GitHub actions within the same company
- Unique inter-company edges (directed): Number of directed edges between different companies
- Weighted density: Mean edge weight, indicating the average strength of ties in the network