ABSTRACT
App developers are constantly competing against each other to win more downloads for their apps. With hundreds of thousands of apps in these online stores, what strategy should a developer use to be successful? Should they innovate, make many similar apps, optimise their own apps or just copy the apps of others? Looking more deeply, how does a complex app ecosystem perform when developers choose to use different strategies? This paper investigates these questions using AppEco, the first Artificial Life model of mobile application ecosystems. In AppEco, developer agents build and upload apps to the app store; user agents browse the store and download the apps. A distinguishing feature of AppEco is the explicit modelling of apps as artefacts. In this work we use AppEco to simulate Apple’s iOS app ecosystem and investigate common developer strategies, evaluating them in terms of downloads received, app diversity, and adoption rate.

Categories and Subject Descriptors
1.6.3 [Simulation and Modelling]: Applications
1.6.5 [Simulation and Modelling]: Model Development

General Terms
Algorithms, Experimentation, Human Factors.

Keywords
App developers, strategies, app ecosystems, Artificial Life, mobile apps, agent-based simulation, app store, software ecosystems.

1. INTRODUCTION
It pays to be an app developer. Some of the world’s most recent millionaires made their money from mobile apps. For example, Ethan Nicholas made his million from his iShoot app in less than a year [1]. Rovio, the developer of Angry Birds, made a revenue of $100 million in 2011 [2]. The revenue generated from app sales is estimated to surpass $15 billion in 2011 and reach $58 billion by 2014 [3]. However, not all app developers are so lucky. In fact, the majority of developers make little or no profit from their apps. Reports suggested that 80% of paid apps in the Android Market have been downloaded less than 100 times [4].

While such problems may be familiar to those in the music or publishing industries, app ecosystems (comprising developers, users, and apps) face challenges that are brand new to the software industry. App store owners face the challenges of presenting the rapidly increasing app store content to the users and encouraging users to download apps. App users have difficulty in finding good apps amongst the vast number of alternatives. App developers find it increasingly difficult to make their apps stand out among hundreds of thousands of other apps in the app store, achieve downloads, and make profit.

Competition is so fierce and advertising space so congested that the question of how to be a successful app developer is now on the lips of thousands of programmers around the world. But which strategy is best? One approach to answering this question might be to experiment with a real app store: flood the store with thousands of new apps developed using specific strategies and measure their success. However, some strategies, such as copying the apps of others, are difficult to try in the real world. One developer faced a $12.5 million lawsuit for allegedly copying a $3 beer-drinking novelty app that allows users to virtually drink a pint by tilting their iPhone1. Consequently for this work we present an Artificial Life (Alife) agent-based model as an experimental tool to address such questions. Alife methods have proven their worth with many previous simulations of ecosystems.

In this paper, we present AppEco, a model of app ecosystems. AppEco models developers (agents that build apps) and users (agents that download apps). It simulates the app store environment, which hosts and organises content created by the developers, and enables users to browse and download apps. Significantly, AppEco also models apps – artefacts produced by the developers and downloaded by users – and their features. AppEco allows us to conduct experiments, test hypothesis about various processes in the ecosystem, and ask “what if” questions, all of which are difficult if not impossible to conduct in a real-world setting. Here, we use AppEco to simulate Apple’s iOS app ecosystem and investigate common strategies adopted by developers.

The rest of the paper is organised as follows. Section 2 describes existing work. Section 3 describes AppEco. Section 4 describes the application of AppEco to simulate the iOS app ecosystem, the experiments and results. Section 5 provides our conclusions based on the results.

1 http://www.wired.com/gadgetlab/2008/10/indie-iphone-de/
2. BACKGROUND

While the study of mobile app ecosystems is a current and significant topic for researchers, to date there has been little work focusing on the topic [5, 6]. However there is much related work that contextualises and informs our study.

One area related to app ecosystems is the study and prediction of app sales and usage. For example, Garg and Telang developed strategies to infer the current sales of an app based on its ranking on Apple’s iOS App Store Top Apps Chart [7]. Such work may enable investors to estimate likely profits should an app reach a specific rank, however there is no certainty that a new app will appear on the chart. Bohmer et al. developed a mobile app to collect mobile app usage information from over 4,100 users of Android devices [8]. Their research revealed interesting app usage behaviours among the users. For example, although users spend almost an hour a day using their phones, an average session with an app lasts less than a minute. They also found that news apps are most popular in the morning and games are at night, but communication apps dominate through most of the day [8]. These studies are informative, but they are limited to studying what is already out there, and “what-if” questions cannot be answered.

In the fields of Alife, Evolutionary Computing and Agent-Based Simulation, researchers have modelled various aspects of ecosystems such as evolutionary dynamics within interacting populations. Classic works in this area include studies by Axelrod and Hamilton on the evolution of cooperation [9] and Maynard Smith and Price on conflicts between animals of the same species [10]. More recently, Holland created Echo, a generic ecosystem model in which evolving agents are situated in a resource-limited environment [11]. Pachepsky et al. investigated the effect of ecological interactions between organisms on the evolutionary dynamics of a community [12]. There are also a growing number of studies on the emergent effects of human interaction at the population level. For example, Kohler et al. used models to understand the environmental and social factors that led to the disappearance of the Puebloan peoples of the North American Southwest [13]. Lux and Marchesi showed that the scaling of financial prices arises from interactions between a large number of market participants [14]. App stores have large populations of apps, developers, and users, and can benefit from similar studies.

Indirect interaction through mechanisms such as stigmergy is commonly studied by Alife researchers. In human society different kinds of objects and tools are often built, adopted, shared and used to support people in their work [15]. It is common for such artefacts to become media for communication (e.g., books, music, and software). One study relating to this topic is the use of robots to create music. In this work, Miranda developed a group of interactive autonomous singing robots that imitate each other to create music [16]. Despite such studies, which often focus on the evolution of human culture with reference to artefacts [17], there is a lack of models that study the development and consumption of artefacts by agents, and how the success of the artefact depends on the preferences of the agents.

3. APPECO

In an app ecosystem, coevolving systems of apps, developers, and users form complex relationships, filling niches, competing and cooperating, similar to species in a biological ecosystem [6]. The health of the app ecosystem is largely determined by the communities of developers that create innovative solutions that users want to buy [5, 18]. In an app ecosystem, application software (such as games, medical applications, and productivity tools) that is built for a mobile platform is sold via an app store running on the platform. The app store concept has democratised the software industry – almost anyone can build and sell apps. Once built, an app quickly becomes available to a worldwide market. Mobile device users can download the apps, use them immediately and provide feedback to the developers.

AppEco is an Artificial Life simulation of mobile app ecosystems. The model consists of agents that are abstractions of app users and developers, as well as artefacts that are abstractions of apps. Developer agents build and upload apps to the app store; user agents browse the store and download the apps, see Figure 1. Each download corresponds to a new sale. A distinguishing feature of the AppEco model compared to more traditional agent-based models is the explicit modelling of artefacts as well as the agents that produce and use the artefacts. Different from agents, artefacts are not autonomous, they represent passive entities of the system that are intentionally created and used by agents [15]. App artefacts are important in a model of an app ecosystem because the agents interact with one another via the apps.

![Figure 1. The interaction between developers, apps, and users in AppEco.](image)

3.1 AppEco Components

AppEco consists of app developers, apps, users, and the app store. Each component is described as follows.

3.1.1 Developers

In AppEco, a developer agent represents a solo developer or a team of developers working together to produce an app. Each developer agent has a development duration (devDuration, a random value between [dev_min, dev_max]), which specifies the number of days it needs to build an app. Each developer also records the number of days it has already spent building the app (daysTaken). Each developer is initially active (it continuously builds and upload apps to the app store) but may become inactive (it stops building apps) with probability P_inactive. This enables the modelling of part-time developers, hobbyists, and the tendency of developers to stop building apps. Every developer records the number of apps it has already developed and the number of downloads it has received. Each developer uses one of the following strategies to build apps:

- **S0 Innovator**: Builds an app with random features each time. This strategy models innovative developers. For example, iOS developer Shape Services produces different apps in a variety of categories such as social networking, business, utilities, and productivity.

---


3 http://www.shapeservices.com/
• **S1 Milker**: Makes a variation of own most recent app each time. This strategy models developers who “milk” a single app idea repeatedly. An extreme example is Brighthouse Labs which produced thousands of similar apps\(^4\), such as an app to provide news for each region in each country, and an app about each sports team for each sport.

• **S2 Optimiser**: Makes a variation of own best app each time. This strategy models developers who learn from downloads and improve on their best app. For example, Rovio developed many game apps before hitting the jackpot with Angry Birds. They then built on their success, releasing new apps such as Angry Birds Seasons, and Angry Birds Rio\(^5\).

• **S3 Copycat**: Copies an app in the Top Apps Chart. This strategy models developers who are less creative but want to achieve many downloads quickly. Angry Chickens and Angry Dogs are two example copycats of Angry Birds\(^6\).

• **S\(^\ast\) Flexible**: Developers begin with one of the strategies S0-S3. Each developer then has a 0.99 probability to randomly select an app from the Top Apps Chart and change strategy to be the same as the developer of the selected app. There is a 0.01 probability that a strategy is randomly selected.

These strategies are abstracted after consultation with app literature and developers. Few app developers perform market research before developing.

#### 3.1.2 Apps

Each app artefact is built and uploaded by a developer agent. The features of the app are abstracted as a 10x10 feature grid (F) for each app. If a cell in F is filled, then the app offers that particular feature. A grid is used so that feature similarity can be represented in the future, e.g., features that are similar can be represented as cells that are near to one another on the grid. For ranking purposes, each app keeps a record of the total number of downloads it has received to date and the number of downloads it has received on each of the previous seven days. For simplicity, the model currently assumes that all apps are sold at the same price; the model of variations in app pricing and categories of apps is left for future work. Each app also records the time when it is being uploaded.

The app feature grid F is filled depending on the strategy used by the app’s developer:

- **S0**: The cells in F are filled probabilistically, such that each cell in the grid has a probability \(P_{\text{fill}}\) of being filled.
- **S1**: The cells in F are filled probabilistically as in S0 if this is the developer’s first app. Otherwise, the developer copies the features of his own latest app with random mutation.
- **S2**: The cells in F are filled probabilistically as in S0 if this is the developer’s first app. Otherwise, the developer copies the features in his own best app (as determined by the highest daily average downloads) with random mutation. The choice of which app to copy occurs when the developer is starting to build the app. If no apps by this developer have downloads, the developer just copies his most recent app.

- **S3**: An app is randomly selected from the Top Apps Chart and its features are copied with random mutation. The choice of app to copy occurs when the developer is starting to build the app. There is a 0.5 probability that mutation occurs during a copy. Mutation is implemented by randomly selecting a filled cell in F and randomly “moving” it to an empty cell in F.

#### 3.1.3 Users

Inspired by the recommender systems literature [19], each user agent has preferences (or taste information) that determine the app features that it prefers. The preferences of a user agent are abstracted as a 10x10 preference grid (P). The cells in P are filled probabilistically, such that each cell in the grid has a probability \(P_{\text{fill}}\) of being filled. If a cell in P is filled, then the user agent desires the feature represented by that cell. If the feature grid F of an app has a cell in the same location filled, then it means the app offers a feature desired by the user agent. For example, in Figure 2, all four of the features offered by App 1 match the user agent’s preferences, but only two of the features offered by App 2 match the user agent’s preferences. For simplicity, preference matching is binary: filled cells either match or do not match. The top right quadrant in P is always empty in order to model some features that are undesirable to all users, see Figure 2. For example, no users want an app to have the features of a difficult-to-use or malicious program. Using the AppEco model, a popular app such as Angry Birds can be abstracted as an app with F that matches P of many users, while a less popular app has F that matches few or no users’ P. The developers are unaware of the users’ preferences.

![Figure 2. Matching app features with user preferences.](image)

Finally, a user agent keeps a record of the apps it has downloaded, the number of days between each browse of the app store \((\text{daysBetweenBrowse}, \text{a random value between } [\text{bro}_{\text{min}}, \text{bro}_{\text{max}}])\), and the number of days that have elapsed since it last browsed the app store \((\text{daysElapsed})\). \text{daysElapsed} is recorded so that the user agent knows when to browse the app store next. When users are initialised, \text{daysElapsed} is set to be a random number between \([0, \text{daysBetweenBrowse}]\) so that users don’t all browse at the same time when they start.

#### 3.1.4 App Store

The app store is the environment used by the agents to store and access apps. Its primary function is to provide a shop front for users and enable them to locate and download apps that match their preferences. To achieve this, it provides three browsing methods: the Top Apps Chart, the New Apps Chart, and Keyword Search. These three methods are modelled because they are

---


\(^5\)http://www.wired.co.uk/magazine/archive/2011/04/features/how-rovio-made-angry-birds-a-winner

\(^6\)http://techcrunch.com/2011/12/06/can-we-stop-the-copycat-apps/
common to many app stores, such as iOS, Android, and BlackBerry. The Top Apps Chart ranks apps based on the number of downloads the apps have received. The New Apps Chart displays new apps that have recently been uploaded by developer agents; only a small subset of new apps is chosen for this chart. Keyword Search returns a list of apps that match the keyword entered by the user agent. In AppEco, Keyword Search is abstracted as a random search for a random number of apps. It is implemented in this way because keywords may not correspond to features, so a matching keyword does not mean the app has desirable features for the user.

3.2 AppEco Algorithm

The AppEco algorithm models the daily interactions between the AppEco components described in the previous section. Each timestep in the algorithm represents a day in the real ecosystem.

Inspired by the ecology literature [20, 21], the population growth of user and developer agents is modelled using a sigmoid growth function commonly used to model the population growth in natural systems. The equation models the growth rate of user and developer agents in an app ecosystem declining as their population density increases, with the size of the ecosystem limited by the market share of the mobile platform. The population size at timestep \( t \), \( pop_t \), is defined by Eq. 1.

\[
pop_t = \text{MinPop} + \frac{(\text{MaxPop} - \text{MinPop})}{1 + e^{-S \cdot t}} \tag{Eq. 1}
\]

where MinPop is the minimum population, MaxPop is the maximum population, \( S \) determines the slope of the growth curve (\( S \) is negative for a growth curve), and \( D \) shifts the curve from left to right. Different growth formulas can be used to model different ecosystems [20, 21].

The AppEco algorithm, see Figure 3, is detailed as follows.

**Initialise ecosystem.** This step launches AppEco with the population of developer and user agents as defined in Eq. 1, with timestep \( t = 0 \). It is common for app stores to have apps before it is opened. For example, the iOS App Store had 500 apps the day it was launched. As such, this step also creates an initial number of app artefacts \( N_{\text{initApp}} \). The developers of these initial apps are randomly selected from the pool of initial developers of strategy S0, S1 or S2 (S3 waits for apps to be on Top Apps Chart before it builds apps). The attributes of initial developers, apps, and users are set as described in the previous section.

**Developer agents build and upload apps.** For each active developer, \( \text{daysTaken} \) is incremented by 1. If \( \text{daysTaken} \) exceeds this developer’s \( devDuration \), the app is completed. The developer then uploads the app to the store, resets \( \text{daysTaken} \) to 0, and decides on the next app to build. The feature grid \( F \) of the app is set depending on the developer’s strategy.

**Update app store.** The New Apps Chart is updated. When timestep \( t = 0 \), the New Apps Chart consists of a random selection of initial apps. In each following timestep, each new app has a probability \( P_{\text{OnNewChart}} \) of appearing on the New Apps Chart. Apps are randomly selected here because the selection criteria are not the focus of this work and real app stores do not reveal how they select apps for the New Apps Chart. The maximum number of apps in the chart is defined by \( N_{\text{MaxNewChart}} \). As newly selected apps are added to the chart, older apps appear lower in the chart and are no longer listed when their position exceeds the chart size. The Top Apps Chart is also updated. When timestep \( t = 0 \), the Top Apps Chart is empty because no apps have been downloaded yet. In each following timestep, apps are ranked in the order of decreasing score, calculated as \( 8 \times D_1 + 5 \times D_2 + 5 \times D_3 + 3 \times D_4 \) where \( D_n \) is the number of downloads received by the app on the nth day before the current day [22]. The maximum number of apps in the Top Apps Chart is defined by \( N_{\text{MaxTopChart}} \).

**User agents browse and download apps.** For each user, \( \text{daysElapsed} \) is incremented by 1. If \( \text{daysElapsed} \) exceeds \( \text{daysBtwBrowse} \), then the user browses the app store, and resets \( \text{daysElapsed} \) to 0. The user browses the New Apps Chart and the Top Apps Chart, and conducts Keyword Search (which returns a random number of apps between \( [\text{key}_{\text{min}}, \text{key}_{\text{max}}] \)). The user browses each app that it has not previously downloaded: the feature grid of the app is compared with the preference grid of the user. If all the features offered by the app match the user's preferences, then the user downloads the app. For example, in Figure 2, the user downloads App 1 but not App 2.

**Increase agent population.** This step increases the number of user and developer agents in the ecosystem for the next timestep, using Eq. 1.

AppEco is implemented in C++ and the code can be requested from the authors via email. It is developed to be highly configurable so that it can simulate various app ecosystems, such as iOS, Android, and BlackBerry.

4. EXPERIMENTS

In order to investigate the effects of developer strategies in AppEco, we must first calibrate the simulation to match, as much as is feasible, the behaviour of a real app store. We select Apple’s iOS App Store for our experiments, as it is one of the oldest and most established app stores. The calibration of AppEco to the iOS App Store is described in Section 4.1. We then investigate how different developer strategies affect individual and collective success in AppEco. Two experiments are conducted. Experiment 1 (E1) in Section 4.2 investigates the success of each individual strategy S0, S1, S2, and S3. Experiment 2 (E2) in Section 4.3 investigates the more realistic scenario of competing strategies by having developers select their own strategies over time. Thus in E2, developers have strategy \( 5^* \), and the overall success of the App Store is compared against E1.

4.1 Calibrating AppEco for iOS

We collected the following iOS data over a period of three years, from the start of the iOS ecosystem in July 2008 (Q4 2008) until the end of June 2011 (Q3 2011):

---

Footnote:

• **Number of iOS developers.** The number of iOS developers is based on the number of worldwide iOS developers month over month compiled by Gigaom⁸.

• **Number of iOS apps and downloads.** The number of apps and downloads is based on statistics provided in Apple press releases and Apple Events⁹. For example, in the Apple Special Event on 9th September 2009, Apple CEO Steve Jobs announced the App Store to reach 75,000 apps and 1.8 billion downloads, and Apple’s press release on 28th September 2009 announced that the App Store has achieved more than 85,000 apps and 2 billion downloads¹⁰.

• **Number of iOS users.** The number of iOS users is based on the number of iOS devices (iPod Touch, iPhone, and iPad) sold by Apple over time. The sales figures are available from Apple’s quarterly financial data¹¹, and for simplicity the calculation assumes that each user has one iOS device.

We calibrated AppEco to simulate the iOS app ecosystem. Table 1 summarises the calibrated values for the system constants. Most constants were set from publicly available data. For example, the total number of people in the world who use mobile devices is approximately 4 billion [23]. According to the International Data Corporation (IDC), Apple had 2.8% of the mobile device market share in Q1 2010 and 5% in Q1 2011¹². By assuming a maximum increase of market share to be 10%, our calculation gives us a MaxIncrease, of 400 million users. In order to match (curve-fit) the iOS user and developer growth rates, values such as D and S for users and developers were determined through tuning experiments. To ensure that the system is computationally feasible, one app represents one real app, and one developer agent represents one real developer, but one user agent represents 10,000 real users. This is because it is computationally infeasible in terms of memory to simulate hundreds of millions of users.

Table 1. Constant Values Resulting from iOS Calibration

<table>
<thead>
<tr>
<th>[P0PmaxUser, P0PmaxUser]</th>
<th>[1500, 40000]</th>
<th>[devmax, devmax]</th>
<th>[1, 180]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dinit</td>
<td>-4.0</td>
<td>Pinit</td>
<td>0.45</td>
</tr>
<tr>
<td>Sinit</td>
<td>0.0038</td>
<td>Pinit</td>
<td>0.04</td>
</tr>
<tr>
<td>[P0PmaxDev, P0PmaxDev]</td>
<td>[1000, 120000]</td>
<td>PmaxNewChart</td>
<td>0.001</td>
</tr>
<tr>
<td>Ddev</td>
<td>-4.0</td>
<td>NmaxNewChart</td>
<td>40</td>
</tr>
<tr>
<td>Sdev</td>
<td>-0.005</td>
<td>NmaxTopChart</td>
<td>50</td>
</tr>
<tr>
<td>NinitApp</td>
<td>500</td>
<td>Pactive</td>
<td>0.0027</td>
</tr>
<tr>
<td>[broinit, bromax]</td>
<td>[1, 360]</td>
<td>[keyinit, keymax]</td>
<td>[0, 50]</td>
</tr>
</tbody>
</table>

Figure 4 illustrates the actual and simulated number of users, developers, apps, and downloads. As can be seen, the behaviour of AppEco closely resembles the behaviour of the iOS ecosystem, including emergent rates such as the number of apps and downloads. A run of the simulation takes approximately 16 seconds CPU time on a MacBook Air with a 1.8GHz Intel Core i7 Processor and 4GB of 1333 MHz DDR3 memory. After three years (1080 timesteps assuming 30 days a month), the model typically contains more than 100,000 developer agents, 500,000 apps, 20,000 user agents (corresponding to 200m real users), and 1.5 million downloads (corresponding to 15bn real downloads).

4.2 **E1: Comparing Strategies**

4.2.1 **Objective and Setup**

In a mobile app ecosystem, developers compete with each other to earn more downloads. Some developers try many different ideas, some produce many similar apps, some gain experience from their previous successful apps, and some copy successful apps created by other developers. To learn how each strategy performs relative to one another, we ask the following research questions:

RQ1: Which developer strategy enables individual developers to become more successful?

RQ2: What is the diversity of apps produced by each strategy?

RQ3: Which developer strategy enables the developer to make more money as they develop more apps?

To answer RQ1, the following measurements are used: (1) **Average Downloads per App (AvgDl):** For each strategy S0 to S3, AvgDl is the total number of downloads received by the developers of the strategy, divided by total number of apps built by the developers of the same strategy. (2) **Top 20 Total Downloads (Top20TotDl):** The developers of all strategies are ranked based on the total number of downloads they received. For each strategy S0 to S3, Top20TotDl is the proportion of developers in the top 20 of this list that belong to the current strategy. (3) **Top 20 Average Downloads (Top20AvgDl):** The developers are ranked based on the average number of downloads they received per app. For each strategy S0 to S3, Top20AvgDl is the proportion of developers in the top 20 of this list that belong to the current strategy. Developers can receive more total downloads by building more apps – this measure identifies efficient developers who have many downloads with few apps. (4) **Zero Downloads (ZeroDl):** For each strategy S0 to S3, ZeroDl is the proportion of developers that belong to the current strategy who have received no downloads for any of their apps so far.

To answer RQ2, the **Feature Coefficient of Variation (FeatCV)** is used to measure the app coverage of features that are desired by users. For each cell in the desired region of feature grid F, we calculate the number of apps that offer that feature, forming a

---

⁹ http://www.apple.com/apple-events/
¹⁰ http://www.apple.com/pr/library/
¹¹ http://www.idc.com/getdoc.jsp?containerId=prUS22808211
combined feature grid $F^c$. $FeatCV$ is the coefficient of variation of grid $F^c$. $FeatCV$ is defined in Eq. 2 and expressed as a percentage.

$$FeatCV = \frac{\sigma}{\mu} \quad (Eq. \ 2)$$

where $\sigma$ is the standard deviation and $\mu$ is the mean of values in grid $F^c$. In this simulation, the user preference coefficient of variation in the desired region of $F$ is 0.688%, indicating that the mean preferences for the user population are evenly distributed over the mean $F$ feature grid. As such, a good strategy should have a low $FeatCV$, which means that all the apps have features that cover the desired region in $F$ evenly (in combination they better meet all the users’ needs).

To answer RQ3, we measure the Fitness of each strategy as its developers gain more experience in app development. For each strategy, we categorised the apps into classes corresponding to their developers’ first apps, second apps, third apps, and so on. These correspond to the apps created by the developers at experience level 1, 2, 3, and so on. For each app, we “survey” the users and ask if they would download the app: if all the features in the app match the user’s preferences then they would download the app. For each strategy, the Fitness of the strategy at experience level $L$ is defined in Eq. 3.

$$Fitness_L = \frac{AvgDl_L}{NumUsers} \quad (Eq. \ 3)$$

where $AvgDl_L$ is the number of potential downloads as reported by users in the survey for all the apps in experience level $L$ divided by the number of apps in $L$, and $NumUsers$ is the number of users who participated in the survey. $Fitness_L$ ranges from 0 to 1. The higher the value, the fitter the strategy.

AppEco was run with the settings described in Section 4.1. Throughout each run, developers in the ecosystem were randomly assigned strategies $S0$, $S1$, $S2$ or $S3$ in equal proportions to enable direct comparison of relative performance. AppEco was run for 1080 timesteps (corresponding to three years in the real world, assuming 30 days a month). The experiment was repeated 100 times. The results were averaged over the 100 runs.

4.2.2 Results and Analysis

RQ1: Which developer strategy enables individual developers to be most successful?

As can be seen in Table 2, the Copycat strategy $S3$ is the most successful, receiving the highest $AvgDl$, $Top20TotDl$ and $Top20AvgDl$, and the lowest $ZeroDl$. (The success of Copycats is well known in the real world — several Copycats who have paratisised Angry Birds have risen high in the Top Apps Chart). Although the Innovator strategy $S0$ performed worst with the lowest $AvgDl$ and $Top20TotDl$, it has a lower number of developers with zero downloads ($ZeroDl$) compared to strategies $S1$ and $S2$. This is because by randomly trying different ideas for apps, creative $S0$ avoids dwelling on ideas that do not work, unlike $S1$ Milker and $S2$ Optimiser that keep working on similar apps. $S1$ Milker has the lowest $Top20AvgDl$, illustrating that a strategy that produces many similar apps results in poor performance. (In the real App Store this strategy is also heavily criticised by other developers and users for its exploitative approach.)

Studies of individual runs show that although $S3$ Copycat dominates the top developers lists, individual developers from other strategies can become the most successful developer in the ecosystem, achieving either the highest number of downloads or the highest average downloads per app. However, the occurrence is by chance: the developer has to build the right app at the right time (e.g., the app has features preferred by many users, appears on the New Apps Chart, is downloaded by many users, appears on the Top Apps Chart, and continues to attract downloads).

Developers with high $Top20AvgDl$ tend to develop one app in the entire three years. This shows that it is difficult to repeat success (a fact well-known to app developers in the real world). The more apps a developer builds, the lower the average downloads tend to be. As such, the same developers rarely appear on both average and top downloads lists. In many runs, we observe that developers who built many apps can have the highest total number of downloads, but often have a low average.

Table 2. Results for RQ1 and RQ2 (Std. Dev. in Brackets)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>AvgDl</th>
<th>Top20TotDl</th>
<th>Top20AvgDl</th>
<th>ZeroDl</th>
<th>$FeatCV$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S0$ Innovator</td>
<td>1.18</td>
<td>3.85%</td>
<td>9.10%</td>
<td>26.51%</td>
<td>1.40%</td>
</tr>
<tr>
<td></td>
<td>(0.14)</td>
<td>(4.20%)</td>
<td>(5.52%)</td>
<td>(0.23%)</td>
<td>(0.13%)</td>
</tr>
<tr>
<td>$S1$ Milker</td>
<td>1.19</td>
<td>5.95%</td>
<td>8.80%</td>
<td>32.85%</td>
<td>4.27%</td>
</tr>
<tr>
<td></td>
<td>(0.14)</td>
<td>(5.16%)</td>
<td>(6.52%)</td>
<td>(0.26%)</td>
<td>(0.35%)</td>
</tr>
<tr>
<td>$S2$ Optimiser</td>
<td>1.41</td>
<td>7.40%</td>
<td>9.25%</td>
<td>32.90%</td>
<td>6.50%</td>
</tr>
<tr>
<td></td>
<td>(0.15)</td>
<td>(6.05%)</td>
<td>(7.30%)</td>
<td>(0.27%)</td>
<td>(0.62%)</td>
</tr>
<tr>
<td>$S3$ Copycat</td>
<td>8.22</td>
<td>82.80%</td>
<td>72.85%</td>
<td>7.74%</td>
<td>54.36%</td>
</tr>
<tr>
<td></td>
<td>(0.41)</td>
<td>(8.30%)</td>
<td>(10.03%)</td>
<td>(0.23%)</td>
<td>(6.78%)</td>
</tr>
</tbody>
</table>

RQ2: What is the diversity of apps produced by each strategy?

Developers who use the $S0$ Innovator strategy offer the most even coverage of features. Despite being the most successful strategy in terms of downloads, $S3$ Copycat has the highest $FeatCV$, which suggests that it only partially covers the preference space of users. Analysis shows that $S3$ produces an average of about 1 feature per app, while the other strategies have an average of 4 features per app. This is because users only download an app when all the features of the app match their preference, and as a result apps with fewer features will have a higher chance to be downloaded by many users and appear on the Top Apps Chart. Since $S3$ Copycat copies from Top Apps Chart, the Copycat developers are likely to copy apps with very few features. (This is consistent with the advice from successful app developers — apps with fewer features have a higher chance of being downloaded).

To investigate further, we group the apps by the experience level of their developers when the apps are built. We aggregate the features of the apps with the same strategy and experience level. We find that the coverage of features using $S3$ Copycat is worse than other strategies indicating that developers following this strategy are not satisfying as many users’ needs (Figure 5). In contrast, the developers using $S2$ Optimiser correctly avoid the top right corner as they become more experienced, while also consistently covering the features desired by the users.

RQ3: Which developer strategy enables the developer to become more successful as they develop more apps?

The only strategy that shows clear improvement as the developers become more experienced is $S2$ Optimiser. While $S3$ Copycat is the clear winner in terms of downloads, developers using the strategy are plagiarising the work of other developers rather than improving their own work. In $S2$ developers release new apps based on mutated copies of their most successful apps. As such this is similar to a $(1 + \lambda)$ Evolutionary Strategy. Figure 6 illustrates that among the four strategies, $S2$ shows a classic evolutionary curve. This demonstrates that developers who improve their own apps

---

based on download feedback should increasingly meet the needs of the users. In addition, among all four strategies, S2 also offers the highest number of features desired by the users.

![Figure 5. Heat maps for one run showing total app features over experience level.](image)

**Figure 5. Heat maps for one run showing total app features over experience level.**

**Table 3. Proportion of Developers at timestep $t = 1080$ (RQ4)**

<table>
<thead>
<tr>
<th>Proportion of Developers</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 Innovator</td>
<td>33.51%</td>
</tr>
<tr>
<td>S1 Milker</td>
<td>26.57%</td>
</tr>
<tr>
<td>S2 Optimiser</td>
<td>29.26%</td>
</tr>
<tr>
<td>S3 Copycat</td>
<td>10.67%</td>
</tr>
</tbody>
</table>

Indeed, studies from individual runs reveal that while S3 Copycat consistently falls quickly to 10% of the developer population, the winning strategies fluctuate unpredictably, at times with S0 Innovator, S1 Milker or S2 Optimiser each dominating the population, see Figure 7. It is interesting to note that the two most widely hated strategies in real life: S1 Milker and S3 Copycat, appear to be used the least in the ecosystem, with S3 Copycat clearly in the minority. To assess whether S3 could ever become widely adopted, we repeated E2 by only allowing developers to change strategies after 2 months in order to give S3 more opportunity to work. There was no change to the result: S3 again becomes unpopular. Even when E2 is repeated with 50% developers using S3 Copycat, most developers subsequently avoid choosing S3. In fact, in E2, the only way to guarantee that S3 will dominate the ecosystem is to force developers not to change strategies until after 1 year, by which time there are plenty of good apps to copy. This demonstrates that S3 is only viable as a minority strategy in a healthy ecosystem. Copycats rely on good apps created by other strategies; it is extremely difficult for an ecosystem to support a large proportion of Copycats. The result mirrors the app stores in the real world – Copycat developers regularly appear and take advantage of the success of others, but nevertheless their strategy remains in the minority.

**Figure 7. Proportion of developers in two example runs.**

**RQ5: What is the diversity of apps produced?** When developers can choose their strategy, the app ecosystem has a higher diversity of apps. E1 resulted in a $\text{FeatCV}$ of 6.28% with a standard deviation of 0.72%; E2 resulted in $\text{FeatCV}$ of 2.87% with a standard deviation of 2.33%. This shows that in E2, the apps

4.3 **E2: Ecosystem Health**

4.3.1 **Objective and Setup**

In real life, there are no fixed strategies. Developers can choose the strategy they want to use. With all developers free to choose, the strategies directly compete with each other in the ecosystem. If a strategy were more effective then it might quickly dominate all others; less effective strategies might become a tiny minority. However, in many ecosystems, individual success is not always reproducible for many [9]. It is therefore of great interest to study how the number of developers using each strategy varies over time. Our research questions are thus:

**RQ4:** When strategies compete, how often is each strategy chosen by developers?

**RQ5:** What is the diversity of apps produced?

**RQ6:** Is an app ecosystem that comprises competing strategies able to improve its performance in the long term?

To answer **RQ4**, we measure the proportion of developers using each strategy over time. To answer **RQ5**, we use $\text{FeatCV}$ (Eq. 2) on the aggregated app features in E2 and compare the results with the aggregated app features in E1. To answer **RQ6**, we use the $\text{Fitness}$ measure (Eq. 3) on E2 and compare the outcome with E1. AppEco was run with the settings described in Section 4.1, with all developers initialised with flexible strategy S*. E2 was also run for 1080 timesteps and repeated 100 times. The results were averaged over the 100 runs.

4.3.2 **Results and Analysis**

**RQ4:** When strategies compete, how often is each strategy chosen by developers? The choice of strategy depends on the proportion of other strategies in the population, but S3 Copycat is the least frequently chosen strategy (Table 3), despite appearing to be the best strategy from E1. When developers have a choice, very quickly the Copycat strategy is dropped in favour of the other strategies. As is evident from Table 3, strategy S0 Innovator is the most popular choice, followed by S2 Optimiser and then S1 Milker. However, the standard deviations for S0, S1 and S2 are very high, indicating that the percentage of developers from those strategies differs greatly in different runs (Table 3).

**Figure 6. Fitness of a strategy as its developers become more experienced.** Later data is more sparse as fewer developers create large number of apps, resulting in more noise.
evenly cover the users’ preference space. Analysis of all features for E1 and E2 shows that E2 has a more evenly distributed feature set. This means that in E1, users may find that the app store does not have any apps that meet some of their preferences.

RQ6: Is an app ecosystem that comprises competing strategies able to improve its performance in the long term? When we plot total fitness against developer experience level, there is a clear improvement evident for the more realistic ecosystem where developers are free to choose their strategies. Figure 8 illustrates that the developers in E2 improve more as they become more experienced, indicating that flexible developers who are not all locked into one development strategy will collectively perform better as they develop more apps.

![Figure 8. Fitness as developers become more experienced.](image)

## 5. CONCLUSIONS

It is the dream of many developers to make their fortune with a clever app. But when the chances of success are now similar to the chances of winning the lottery – and falling daily as new apps are released – how can anyone be a successful app developer?

In this work, we presented AppEco, an Artificial Life agent-based model that simulates app ecosystems, and investigated these issues. AppEco models developers (agents that build apps) and users (agents that download apps). It simulates the app store environment and the population growth of the agents and apps. Significantly, AppEco also models apps (artefacts produced by the developers and downloaded by users) and their features. AppEco is a complex ecosystem where developers, users and apps interact continuously, and interaction strategies may continuously change. Our experiments investigated different developer strategies: Innovators, Milkers, Optimisers, and Copycats.

In a complex ecosystem no strategy can be a guaranteed winner, but our results indicate that some strategies should be chosen more frequently than others. Innovators produce diverse apps, but they are hit or miss – some apps will be popular, some will not. Milkers may dwell on average or bad apps as they churn out new variations of the same idea. Optimisers produce diverse apps and tailor their development towards users’ needs. Finally, Copycats may seem like the best strategy to guarantee downloads in an app ecosystem, but the strategy can only work when there are enough other strategies to copy from. In addition, this strategy can only exist in a minority, otherwise app diversity will decrease (many duplicated apps result in a scarcity of some features desired by users) and the fitness of the ecosystem will suffer.

This study is one of many we will be undertaking with AppEco. We plan to study the effect of publicity on app downloads, and understand how a user might best locate desirable apps and communicate their requirements and feedback about the apps to developers, and how these feedback can influence app development. AppEco can also be calibrated to study other app ecosystems, such as Android and Blackberry, and extended to model web-based platforms such as Facebook and Chrome.

## 6. REFERENCES