CHAPTER 8.

SUCCESSFUL AND ABANDONED SOURCEFORGE.NET PROJECTS IN THE *INITIATION STAGE*

Chapter 6 provided an open source project success and abandonment dependent variable. Chapter 7 described data available in the Sourceforge.net repository and linked these data to various independent variable concepts and hypotheses presented in the theoretical part of this book. Chapter 7 also described the Classification Tree and Random Forest statistical approaches we use in this and the following chapter. This chapter presents the results of the Classification Tree analysis for successful and abandoned projects in the Initiation Stage, which in Chapter 3 (Figure 3.2), we defined as the period before and up to the time when a project completes a first release of its software. Readers are encouraged to review Chapter 6 (especially Table 6.1) for specifics on how we operationalized this definition as well as the other Initiation Stage dependent variable categories (e.g., Abandoned in Initiation, Indeterminate in Initiation).

Results

Trees and Random Forest Variable Importance Plots for the Initiation Stage reveal the importance of the Project Information Index (PII) for discriminating projects that were successful in the Initiation Stage. Here we provide a classification tree (Figure 8.1) that is generally representative of the results we encountered after a number of different samples from our SF dataset and tree generation. We then use Random Forests and a Variable Importance Plot to verify the results of this tree. In the last part of this section, we examine the

effect of projects with missing or incomplete categorical data.

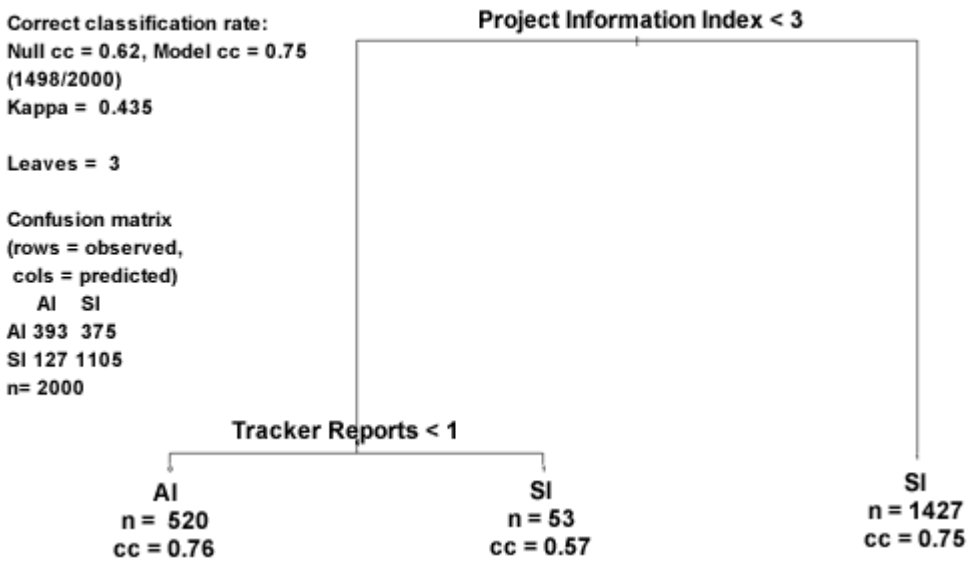

Figure 8.1
Initiation Stage Tree using Sampling Strategy #2
(see Chapter 7, Table 7.5)

We constructed the above using a random sample of 2,000 projects taken from all successful and abandoned projects in the Initiation Stage, but we excluded the Page Visits and Downloads variables, because, as described in Chapter 7, Downloads are closely linked with the definition of our dependent variable in the Initiation Stage, and Page Visits are highly correlated with Downloads. As can be seen in this figure, the Project Information Index (PII) – the total number of categorical subcategories chosen by project administrators to describe the project – was the main splitting variable. Tracker Reports also helped to distinguish between successful and abandoned projects. Recall that projects for which the splitting variable expression evaluates to “true” are sorted into the left node of the tree (Figure 8.1). In other

words, 76% of the 520 projects that had a value less than 3 for the PII and also had less than one Tracker Report were correctly classified as Abandoned in Initiation (AI). Using these two variables – PII and Tracker Reports alone – seventy-five percent of the projects were correctly classified (as shown by the “Model cc = 0.75” at top left in Figure 8.1). The Kappa of 0.435 in Figure 8.1 indicates that the model improved the classification accuracy by about forty-three percent over chance.

Although the overall classification accuracy is only fair, the Confusion Matrix in Figure 8.1 shows that our model classifies successful projects much better than abandoned projects. Of the successful Initiation Stage projects, 1105 were correctly classified, while only 127 were incorrectly classified. On the other hand, only 393 AI projects were correctly classified while 375 were incorrectly classified. In other words, the model classified about ninety percent of the Successful in the Initiation stage projects correctly, but only classified a little better than fifty percent of the Abandoned in the Initiation projects correctly. However, before considering in detail what these results mean, we need to look a little closer to evaluate the importance of some of our other variables.

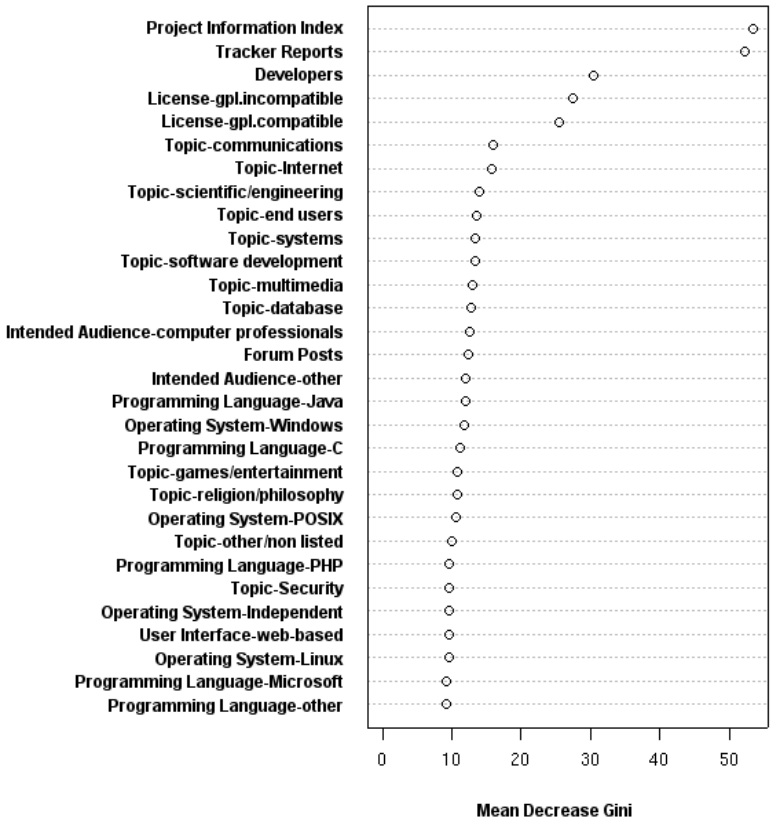

Figure 8.2
Initiation Stage Variable Importance Plot using Sampling Strategy #2
(n=2000, Based on 500 Trees, see Chapter 7, Table 7.5)

We constructed the Variable Importance Plot (VIP) shown in Figure 8.2 using the same data selection criteria used for the Classification Tree shown Figure 8.1 (i.e., Sampling Strategy #2). Figure 8.2 shows the Mean Decrease in the Gini Index, which indicates the relative importance of variables having the most effect on the accuracy of over 500 trees generated using the Random Forests methodology described in Chapter 7. Variables decrease in importance going from the top of the plot to the bottom. The fact that the PII and Tracker Reports are the most important variables corroborates the results of the classification tree shown in Figure 8.1. We also see in Figure 8.2 that Developers and Project License

(gpl.compatible and gpl.incompatible) appeared as important splitting variables in the Random Forest generated by this methodology.

We will talk more about the Developers variable in following sections of this chapter and the next, but as it turns out, Project License is highly correlated with PII in the Initiation Stage. The similarity between these two variables is revealed by the fact that almost all of the projects with a PII less than 3 (the splitting number for the top node in Figure 8.1) have not selected a license. Thus, the classification tree process might choose either a PII less than three *or* the fact that a project has not chosen a license to make the split, depending on relatively small changes in the characteristics of a sample from our dataset. A more intuitive way to understand how these variables are surrogates is to reason that project administrators who fail to make the important choice of a license are unlikely to choose other descriptive categories for the project. Consequently, the license variables are surrogates for the PII, making the PII even more important than it appears to be at first glance. At this juncture we should emphasize another important point related to the license variables. They appear near the top of the Figure 8.2 VIP not because one license type (GPL incompatible or GPL compatible) helps to distinguish successful and abandoned projects in the Initiation Stage, but rather because *selecting a license*[1] *compared to not selecting a license* helps to make this distinction.

Finally, examining the VIP in Figure 8.2, we should take note that almost all of the categorical variables except licenses (from “Topic – Communications” through “Programming Language-Other”) fail to make an important contribution to discriminating between successful

1 Note that it is not always one or the other: GPL compatible or GPL incompatible. Some (1906 to be exact) projects in our Sourceforge.net dataset have selected both options.

and abandoned projects in the Initiation Stage. Stated a more positive way, it appears that both success and abandonment of open source projects are widely distributed across intended audiences, operating systems, programming languages, project topics and the other categories that our categorical variables represent. To illustrate this point we provide Table 8.1, which includes all the projects in our database (n=107,747) and shows the number of projects in each dependent variable class for every categorical variable subcategory. This table shows that all the subcategories have a substantial number of projects in all the dependent variable classes.

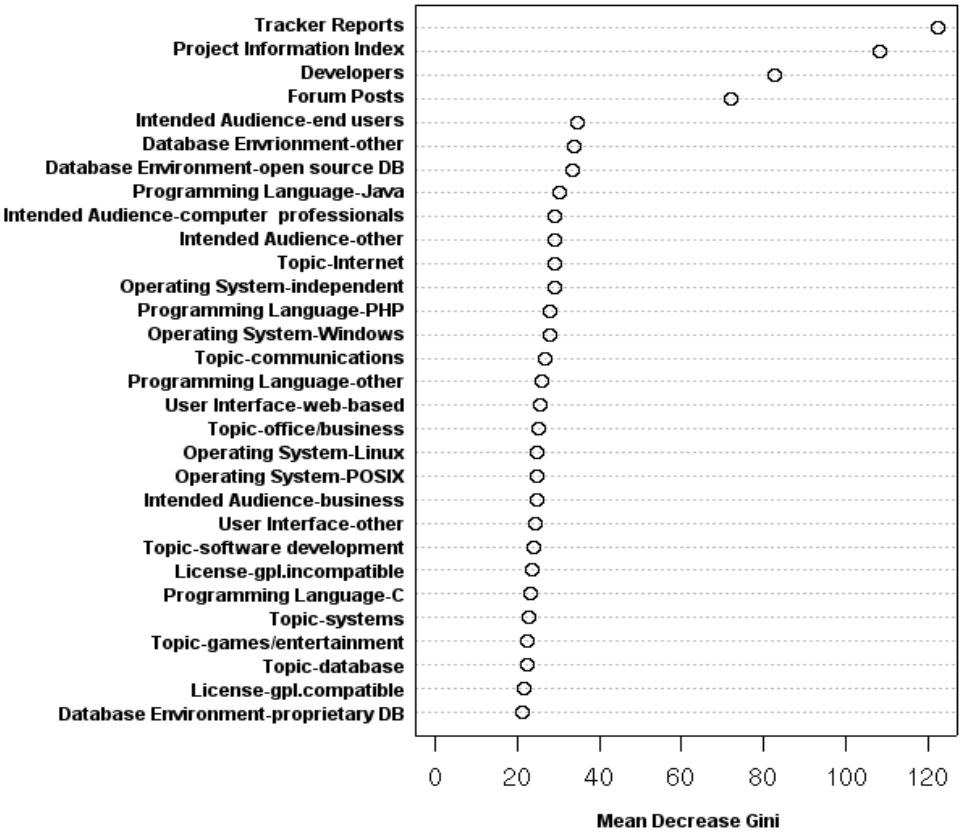

Figure 8.3
Variable Importance Plot using Sampling Strategy #3
(n=5037, see Chapter 7, Table 7.5)

Before discussing our Initiation Stage results further, we want to investigate the effect that “missing” data not completed by project administrators has on the results shown in Figures 8.1 and 8.2. In order to do this, we constructed the VIP shown in Figure 8.3 using the same set of independent variables as was used to generate Figure 8.1 for all successful and abandoned projects in the Initiation Stage except we only sampled projects that had ”Complete Observations” (n=5037, see Table 7.5 Data Subset #3). Recall that we defined “Complete Observations,” in the “Handling of Missing Data” section in Chapter 7, to be projects that have at least one variable selected for each of our categorical variable groups. Figure 8.3 shows that the two most important variables are Tracker Reports and the PII, with Developers being the third most important variable. These findings coincide quite well with our earlier results in Figure 8.2 for the most important splitting variables and give us confidence that our results are meaningful despite any concerns about missing data.

Discussion / Findings

Now that we have verified our results and addressed concerns about missing data, we can discuss our findings for the Initiation Stage. Recall that in Chapter 7 we discussed all our independent variables and their association with hypotheses presented in earlier theoretical chapters. This section describes our findings along with their relationship to these hypotheses, where applicable.

Initiation Stage Finding #1: The Project Information Index lends support for hypotheses H-P1 (software requirements - "clearly defined vision"), H-P3a (software "utility"), and H-C6a and H-C6b (leadership) in Table 7.8.

Our most important finding is that a PII greater than 2 is highly correlated with success in the Initiation Stage. Recall that the PII is a metric we created that totals the number of subcategories of the categorical variables that a project's administrator had selected to describe the project. The highest possible value is 54 categories, and the highest PII score any one particular project had in our 107,747 dataset was 25.

So why would a project leader's simply selecting more descriptive subcategories in SF be associated with project collaborative success in the Initiation Stage? Our first thought was that the PII number may rise *after* a project becomes successful in the Initiation Stage, and thus may not have anything to do with a project *becoming* a Success in Initiation Stage (SI) project. In Chapter 6 we made the point that the SI class is any projects that have made it to the Growth Stage – that is, it includes *all* the Abandoned in Growth (AG), Indeterminate in Growth (II) and Successful in Growth (SG) projects. So our results might indicate that projects add categories as they pass through the Growth Stage. This would mean that a higher PII would not reflect a higher rate of success in the Initiation Stage, but rather it would reflect what happens to the PII *after* a project becomes successful in the Initiation Stage. Projects may list more information as they add more functionality over time (because the project uses more programming languages, runs on more operating systems, has more user interfaces, etc.), or perhaps it is as simple as a project administrator finally having time to do the "paperwork" of adding more detailed project descriptions.

But is this the case? Do projects add categories, and thus raise their PII as they progress through the Growth Stage, or do projects in the Initiation Stage with higher PII

values become successful? Fortunately, we were able to shed some light on this question, and it appears that the latter is supported – *projects in the Initiation Stage with high PIIs more often achieve collaborative success*.

Let us summarize briefly how we came to this conclusion. The key analytic question is whether the PII values for projects that just produced a first release (new Growth Stage projects) are substantially higher than projects that have not yet done so. Fortunately, using our 107,747 case dataset, we were able to investigate this question. Our categorical variables were part of the data we received from the FLOSSMole (2006) repository which represents the SF database on August 1, 2006. The PII value for each project reflects this date. However, we noted in Chapter 6 that the FLOSSMole (2006) dataset did not include release dates for projects. We had to collect or "spider" SF for that data ourselves, in October 2006. By having release date data taken from SF later in the year, we were able to query our database to extract projects that did have a release date close to (plus or minus 15 days) the July 31st, 2006 date our PII data were collected. These were very new SI projects that had little time to do further work and change the PII.[2] From this, we have two key pieces of information: PII values for all projects in our dataset, and which projects became SI right around the time that our PII was measured.

Table 8.2 shows statistics for the PII for all of the 107,747 projects in each dependent variable class in our dataset. We also provide a box-plot of this data in Figure 8.4 below. We performed an Analysis of Variance (ANOVA) on the data shown in Table 8.2 and found that all the classes differ significantly from one another (*P*-value less than 0.001) with regard to

[2] We made the assumption that the PII would not change much during a 30 day window of time. We also compared shorter periods of time (i.e., plus or minus 7 days) and the PII values for time periods before and time periods after the first release to verify that this assumption seems reasonable.

PII.[3] The center row of this table provides the statistics for "Very New Successful in Initiation Stage" projects (778 of them). These are projects that are all Successful in Initiation (they achieved a first release), but fall within the Indeterminate Growth class because they are so new that they couldn't yet be classified as either Abandoned (AG) or successful (SG) in the Growth Stage. This row in Table 8.2 shows that the mean PII for these brand new growth stage projects is 6.811. That is, between 6 and 7 categories were chosen for these brand-new Growth Stage projects (see Figure 8.4 as well). This can be compared to the first row in Table 8.2 that shows a smaller PII mean for Abandoned Initiation (AI) projects of 3.796.

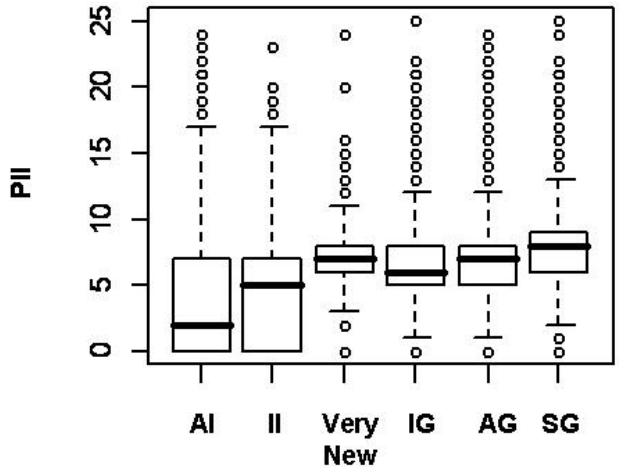

Figure 8.4
Box-plot of PII Scores for Dependent Variable Groups

The other group we should focus on in Table 8.2 and Figure 8.4 is the Indeterminate in

[3] Recall that in Chapter 6 we discussed the "population" of open source projects and the degree to which SF may be "representative" of this population. That discussion is relevant here because the ANOVA we performed is based on the assumption that our PII means are a random sample taken from a larger "population." Also note that our analysis is based on a single time period. Over longer periods of time, changes in factors like the design of the SF administrative interface (see Chapter 7), macroeconomic changes or the growth of the open source phenomenon itself warrant further investigation.

Initiation (II) class. These are projects in the Initiation Stage that do show some kind of development activity but have not yet produced a first release. These projects may or may not eventually be successful in Initiation – we simply can't tell at the time the data were collected. Table 8.2 shows us that the mean PII value for projects in the II class is 4.416, and as we just learned, the average PII value for "Very New Growth Stage" projects is 6.811. What this strongly suggests is that projects with high PII values in the Indeterminate in Initiation (II) class, on average, are the ones that become Successful in Initiation (SI) projects and enter the Indeterminate in Growth (IG) class. In fact, it appears that the "Very New Growth Stage" projects that "moved" from the II class into the IG class at the time of their first release had a mean PII just slightly below the cutoff point of the third quadrant of all II projects (a value of 7.0 in Table 8.2). In other words, projects with high PII values in the Initiation Stage became Successful in Initiation (SI) projects much more often than projects with lower PII values, at least for this point in time. The data in Table 8.2 also suggests that projects with high PII values in the Initiation Stage may often go on to become SG projects in a relatively short period of time. Since the argument supporting this idea is largely theoretical, we have included it as an endnote to this chapter.

This analysis provides strong evidence that Initiation Stage projects that make it to a first release (that is, the Growth Stage) tend to have higher PII values. But why would that be case? We can think of at least three possibilities that relate back to the theory and hypotheses discussed in Chapter 4.

First, we hypothesized that projects with a "clearly defined vision" would be more successful compared to ones lacking a clear vision (see Table 4.1, H-P1). The categories contained in the PII include things like the audiences that the project targets, the operating

systems that the project's software is designed to run on, the programming languages that the project uses, the user interface that the project will present and the project's topic or topics. It seems likely that projects with higher PII values would reflect a clearer plan or vision for the project compared to projects with a lower PII. This lends some support to H-PI for projects in the Initiation Stage.

Second, we hypothesized that projects would be more successful when the software was “considered useful” by end users (see Table 4.1, H-P3a). It seems logical that a software project intended for use by multiple audiences, intended to run on multiple operating systems, and capable of working with multiple user interfaces would have greater utility, by the classical definition of that word, compared to a project with less of these options. It also seems clear that such a project would tend to have higher PII scores (more selected categories). Consequently, the PII may provide a relatively straightforward measure of utility. It is important to remember, however, that because projects in the Initiation Stage have not actually produced software, this utility would be “potential” rather than actual. Nevertheless, since “potential” utility may help to recruit pre-release users or developers, we think that this PII result supports hypothesis H-P3a.

Finally, we'd expect that project administrators who were more diligent or committed, would tend to take more time to answer questions about the project's attributes. For example, a developer leading a serious programming project will more likely take the time to enter this information into SF, whereas, for example, a student at a university using SF to store his or her programming project for a class might not take the time to complete all the relevant categories. In the latter case, it might simply not be important to the developer to do this. In addition, having a clear vision or plan for the project could very well be related to setting

goals, so the PII may be capturing aspects of "project leadership," lending support to our hypotheses H-C6a and H-C6b in Table 4.2.

In sum, we think the reason for the PII's strong influence in distinguishing between successful and abandoned Initiation Stage projects is because it captures elements of a "clearly defined vision" (H-P1), software "utility" (H-P3a), and leadership (H-C6a, H-C6b) in Tables 4.1 and 4.2 in Chapter 4.

Initiation Stage Finding #2: Collaborative success and abandonment are widely distributed across SF categories.

The lack of importance of any categorical variables (VIP in Figure 8.2) shows that successful and abandoned open source projects are widely distributed across all categorical variables – intended audiences, operating systems, programming languages, project topics, and user interfaces, and the two main open source license types. This finding is true both for the Initiation Stage and the Growth Stage (Chapter 9). We wonder if this same result would have been true had this analysis been done even five years ago. To us, this finding suggests that open source as a collaborative paradigm may be "maturing," meaning that it is now entering a broader spectrum of software "topic areas," rather than focusing on traditional "open source" technologies, such as software projects around Gnu Linux, Apache, etc. We admit this last point is slightly speculative since we have no hard evidence to support it, but conceptually, it aligns well with the "open source ecosystem" points made back in Chapter 2.

Initiation Stage Finding #3: The collaborative infrastructure Bug Tracker and Forum Post categories do not help to distinguish between successful and abandoned projects in the Initiation Stage.

We included Tracker Reports in this analysis, even though it is the "pre-release" stage,

because it is conceivable that it could be used to communicate feature requests for the upcoming first release. Similar thought was given to Forum Posts as a form of project documentation. This connects back to the "collaborative infrastructure" research question presented in Chapter 4 (RQ-P1, Table 4.1). Figures 8.1-8.3 show that each of these variables help to distinguish between successful and abandoned projects in the Initiation Stage. However, we conducted a deeper analysis (similar to the PII analysis above) that shows in each of these cases the numbers rise *after* these projects get into the Growth Stage and it is these higher numbers for the projects representing Success in Initiation (all Growth Stage projects) that produce this result.

Initiation Stage Finding #4: Group Size does not help to distinguish between successful and abandoned projects in the Initiation Stage.

We included the Developer variable to investigate the three group size theories (Olson/Brooks, Linus' Law, and "Core Team") we discussed back in Chapter 4 (RQ-C4, Table 4.2). Both VIP Figures 8.2 and 8.3 show that developer count is an important variable for distinguishing between successful and abandoned Initiation Stage projects. However, like earlier findings, the question is whether this is true for the Initiation Stage or is it related to the fact that our "Success in Initiation data" are Growth Stage projects? We conducted a similar analysis to what we did for PII, and we discovered that the mean developer count for projects at the time of their first release (the 778 "Very New" projects in Table 8.2) is 1.56, which is very close to the mean developer count for all projects in the Initiation Stage (1.61). In contrast, the mean developer count for all Growth Stage projects is markedly higher (2.3) developers per project (see Figure 8.5).

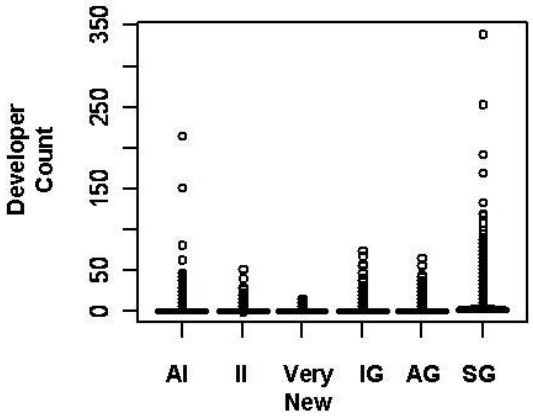

Figure 8.5
Box-plot of Developer Counts for Dependent Variable Groups

Consequently, while the VIPs in Figure 8.2 and 8.3 show that developer counts help to distinguish between successful and abandoned projects in the Initiation Stage, our further examination of the data shows that this finding is because developer counts get higher *after* a project becomes Successful in Initiation. It is an artifact of having Growth Stage projects in our dataset as cases of Success in Initiation. Group Size has important implications for the Growth Stage (as we will return to in Chapter 9), but our conclusion here is Group Size does not help to distinguish between success and abandonment when projects are in the Initiation Stage, and the average developer group size in Sourceforge in this stage is quite small (less than 2). This finding lends support to the "Core Team" theory discussed in which argued that small "core members" do the majority of the development work and consequently, the size of the group shouldn't matter.

Initiation Stage Finding #5: The "Computer Professionals" subcategory of Intended Audience does not help to distinguish between success and abandonment in the Initiation Stage. This "non-finding" suggests a broadening of von Hippel's "user-driven innovation" in the more complex open source ecosystem, where it is no longer just programmers developing for their own (more technical) use, but in addition, programmers developing software for use by other types of end users.

In our descriptions of the categorical variables in Chapter 7, we discussed how the "computer professionals" variable in the Intended Audience category might provide support for the "User-Driven Innovation" hypothesis (H-C1, Table 3.2) discussed in depth in Chapter 3. Our argument for this hypothesis, building on von Hippel (2005), was that open source collaboration is driven by developers programming to meet *their own, more technical* needs. That is, the successful open source Initiation Stage projects would more ones producing software that computer professionals themselves need and use (such as enhancements to a web server module that is used by professional web administrators). Following this logic, we expected to see the Computer Professionals subcategory to be an important indicator of success and show up as a splitting variable in our trees or on the VIPs, compared to the other four aggregated Intended Audience categories (end users, business, government, other; refer back to Table 7.1). Our data do not support this hypothesis. Only in the VIP shown in Figure 8.3 do any of the Intended Audience subcategories appear higher in the chart, and in this case it is the End Users category that is the first to appear, followed by Computer Professionals. However, their Gini coefficients are relatively low, meaning their differentiating power is small. Software written for computer professionals does not help to distinguish between successful and abandoned projects in the Initiation Stage, so von Hippel's user-driven innovation hypothesis is not supported.

But why not? Our first thought could be that our aggregation from nineteen categories

to five categories hides important information. But looking back at this aggregation and how we "clumped" subcategories (see Chapter 7, Intended Audience section), we don't think these aggregations were inappropriate or that changing them would make much of a difference. In our opinion, the best explanation for this finding is what we might call the "maturing nature" of open source: it is not just about computer professionals' needs anymore. The early years of open source (e.g., 1990's and into the early 2000's) were driven by programmers needing software for their own use as von Hippel (2005) has argued; however, in more recent years, such as 2006 when we collected our data, the open source paradigm has expanded, and programmers were willing to develop software outside of the domain of the Intended Audience of Computer Professionals. They still *may* be users of the software (e.g., the "end-user" category) but no longer are programmers limiting themselves to writing software that is intended for Computer Professionals' use (like the web server example noted above). They are writing code for other types of end users. Moreover, we think that this finding may be capturing the effect of a "broadening" open source ecosystem, emphasized in Chapter 2, where organizations – businesses, government agencies, non-profits – are now paying programmers to develop open source solutions to meet organizational needs (following the non-differentiating arguments made by Perens, 2005), which may not be the needs of computer professionals.

Initiation Stage Finding #6. Success and abandonment are widely distributed across all areas of software development, not just in "traditional open source" technologies. The "Helping the Open Source Cause" hypothesis is not supported.

As described in Chapter 7, several SF variables are thought to capture elements of the "helping the open source cause" hypothesis (H-C5, Table 3.2). These include: Operating

System, User Interface, Database Environment, and Project License. In each of these cases, none of their corresponding subcategories associated with "helping the cause" stood out in the trees we generated or in the Random Forest VIPs for the Initiation Stage. For example, if helping the open source cause was a strong driver for collaborative success, we would expect to see projects associated with Linux or BSD (Operating System subcategories), Open Source DB (Database subcategory), Gnome or KDE (User Interface subcategories) or GPL (Program License subcategory) to be splitting variables or to show up in VIPs. None of these were important splitting variables, providing another indication (like the earlier user-generated innovation discussion) that success in open source is influencing all areas of software, and not just in "traditional" open source technology areas – at least in the Initiation Stage.

Initiation Stage Finding #7: No Project Topics, Operating Systems or Programming Languages make a major contribution towards distinguishing between successful and abandoned projects in the Initiation Stage. The "critical infrastructure" (H-P3b, Table 4.1) and "preferred technologies" (H-P3c, Table 4.1) hypotheses are not supported.

Our earlier discussion on the Project Topic SF variable in Chapter 7 highlighted the "systems" subcategory and noted that this captures a component of the concept "critical or foundational infrastructure." The earlier hypothesis (H-P3b, Table 4.1) argued that projects working on foundation infrastructure like operating systems might be more successful than projects focusing on other topics. Alternatively, while not a stated hypothesis, we might find that the topic "games/entertainment" might stand out if the community of younger developers interested in gaming were driving a significant part of the successful open source collaborations on SF. Project Topic areas essentially tied for last place in terms of their importance in discriminating success and abandonment (Figures 8.2 and 8.3). Compared to

the top 4 or 5 variables (including Developers and PII), the Gini coefficients for Project Topic areas reflect little importance. In short, the "critical infrastructure" hypothesis not supported, based on our analysis.

We also were interested in investigating whether any "preferred technologies," like software for particular Operating Systems or ones that used particular Programming Languages, might be a major factor associated with Initiation Stage success or abandonment (H-P3c, Table 4.1). You may recall that the idea behind this is that programmers might be more interested in participating on projects related to one or more operating systems, or which use particular programming languages. The latter could be driven by learning motivations (see Chapter 3). But our analysis (Figures 8.1-8.3) shows that none of the subcategories for Operating System or Programming language are major factors that distinguish between successful and abandoned projects in the Initiation Stage.

Initiation Stage Finding #8: Success is not associated with GPL compatible or GPL incompatible licensing.

Finally, using our one institutional variable available from SF, Project License, we were able to investigate the research question (RQ-I3, Table 5.5) discussed in Chapter 5. Does the choice between a GPL compatible license and a GPL incompatible one affect the collaborative success of a project? For the Initiation Stage, our analysis suggests the answer is no – or at least not in an important way. However, given that these license variables appeared relatively high in the VIP of Figure 8.2, and even though we think we have explanations for this (e.g., the close relationship of the license variables and the PII and that this is capturing the influence of selecting a license at all versus compared to not selecting a

license), we intend to investigate this question further in our survey work in Part IV.

Conclusions

This chapter presented our results and findings of classification tree analysis for Sourceforge.net projects in the Initiation Stage. We will provide reflections on what these findings suggest for building and sustaining open source commons in the summary section of Part III of the book, which follows Chapter 9. But before we do this, we will turn to a similar analysis of Sourceforge.net Growth Stage projects in the next chapter.

Table 8.1
Number of Projects in Each Dependent Variable Class for Each Categorical Independent Variable (includes all 107,747 projects in our database)

	Project Class					
Independent variable	**AI**	**II**	**IG**	**AG**	**SG**	**Total**
Intended audience ia1- end users	8483	3684	4171	12161	7444	35943
ia2 – computer professionals	10875	4338	5058	16750	10130	47151
ia3 – business	1443	713	728	1377	976	5237
ia4 – other	3987	1553	1631	4911	3060	15142
ia5 – government/ non-profit	130	254	197	63	98	742
Operating System os1 – POSIX	7039	1303	1758	10677	6898	27675
os2 – independent	7426	2995	3518	9561	5718	29218
os3 – Linux	5636	1127	1534	8415	5275	21987
os4 – MS Windows	6157	2662	2876	7944	4776	24415
os5 – Mac	941	239	400	1086	1150	3816
os6 – BSD	843	253	313	1170	1002	3581
os7 – unix-like	481	54	131	867	755	2288
os8 - other	1131	382	425	1575	1141	4654
Programming Language pl1 – Java	5444	2688	2703	6411	3700	20946
pl2 – C	7112	2440	2809	10058	6931	29350
pl3 – PhP	3610	1579	1359	4057	1964	12569
pl4 – Perl	1365	338	474	2379	1396	5952
pl5 – Python	1214	529	639	1558	1085	5025
pl6 – Microsoft	1601	1007	951	1966	789	6314
pl7 – other	2139	938	1035	2980	2085	9177
pl8 - Assembly	500	87	101	465	303	1456
User Interface ui1 – web-based	5793	2182	1875	6586	3500	19936
ui2 – MS Windows	4278	842	1104	5837	3284	15345
ui3 – X Windows	2493	166	277	3478	2427	8841
ui4 – non-interactive	1122	288	338	1738	1171	4657
ui5 – console	632	989	1460	922	935	4938
ui6 – Java	565	907	961	536	575	3544
ui7 – Gnome	569	110	109	848	550	2186
ui8 – other	2056	1748	1845	2241	1957	9847
ui9 - KDE	480	80	92	635	447	1734

Table 8.1 **Number of Projects in Each Dependent Variable Class for Each Categorical Independent Variable (includes all 107,747 projects in our database)**						
Database environment de1 – open source DB	1117	1966	1397	805	829	6114
de2 – proprietary DB	203	441	318	180	224	1366
de3 - other	780	1218	972	611	601	4182
Project Topic t1 - communications	2828	1099	953	3750	2066	10696
t2 - database	1285	379	472	1718	1061	4915
t3 – desktop environ.	561	179	245	986	675	2646
t4 - education	810	439	487	1043	627	3406
t5 – formats and protocols	191	283	356	171	214	1215
t6 – games / entertain.	3076	1052	975	3232	1538	9873
t7 - Internet	4360	1525	1679	6176	3367	17107
t8 - multimedia	1784	699	960	3095	2099	8637
t9 – office / business	1330	663	616	1338	858	4805
t10 – other/ nonlisted	552	124	157	693	387	1913
t11 – printing	90	17	48	149	113	417
t12 – religion / philosophy	80	19	28	61	62	250
t13 – scientific / engineering	1755	790	1019	2432	1889	7885
t14 – security	504	190	278	907	531	2410
t15 – sociology	94	29	39	88	64	314
t16 – software development	3319	1601	1928	5149	3482	15479
t17 – systems	2925	1017	1408	4808	2975	13133
t18 – terminals	88	27	47	195	125	482
t19 – text editors	305	89	177	519	422	1512
Project License gpl_compatible	14814	6736	7381	21466	12340	62737
gpl_incompatible	3286	1863	1650	4062	2883	13744

Table 8.2
Project Information Index (PII) Variable Statistics by Dependent Variable Class
(Codes: AI- Abandoned in Initiation, II – Indeterminate in Initiation,
IG – Indeterminate in Growth, AG – Abandoned in Growth, SG – Successful in Growth)

Class	# of Projects	Minimum	1st Quadrant	Median	Mean	3rd Quadrant	Maximum
AI (0)	37,320	0	0	2.0	3.796	7.0	24.0
II (1)	13,342	0	0	5	4.416	7.0	23.0
"Very New" SI projects	778	0	6	7	6.811	8.0	24.0
IG (2)	10,711	0	5	6	6.018	8.0	25.0
AG (3)	30,592	0	5	7	6.305	8.0	24.0
SG (4)	15,782	0	6	8	7.664	9.0	25.0
All Projects	107,747	0	0	6	5.373	8	25.0

Endnote. The mean PII value for the IG stage (6.018) is significantly below the mean PII value for projects at the time of their first release (6.811), with a P-value < 0.001 as described above. Assuming that projects don't lower their PII very often (a seemingly reasonable assumption), this means that, on average, projects with higher PII values in the IG class tend to move rather quickly into the SG Class or the AG Class. (By our definitions, a project can only remain in the IG class for, at most, two years.) If this were not the case, then the IG Class, over less than a two year period, would have a mean PII value approximately equal to the mean value for projects entering that class at the time of their first release (i.e. 6.811), rather than its lower value of 6.018. Although it is possible that some high PII projects move into the AG class, if they all did, then over time the AG class would have an average PII value of 6.811. So, it appears that projects with high PII values in the Initiation Stage not only tend to become SI projects, but also become SG projects more often than projects with lower PII values. This reasoning is consistent with the observation that SG projects have the highest mean PII value (7.664). Projects probably also add information about the project as they grow. As mentioned previously, projects might add information because they increase their functionality (i.e., use more programming languages, run on more operating systems, have more user interfaces), or the projects' administrators may have simply not fully described the project earlier in its lifetime. Because, by our definitions, projects in the AG class must have been in the Growth Stage for at least one year; to some degree, the process of adding information over time may explain the fact that the AG class has a higher mean PII than the IG class (6.305 versus 6.018). If this reasoning is correct, then perhaps most of the projects entering the Growth Stage with a high PII move into the SG class. Admittedly, this argument is theoretical, and other explanations are possible, although they seem less likely. Fortunately, the longitudinal data necessary to verify or refute this argument is available from FLOSSmole and the Sourceforge.net research repository maintained at the University of Notre Dame (http://www.nd.edu/~oss/Data/data.html).