Entscheidungs- & Machtstrukturen

Dominik Diekmann

18.06.2018

Inhaltsverzeichnis

1 Einleitung

1.1 Warum Entscheidungs- & Machtstrukturen?

Die meisten Open-Source Projekte entstehen durch ein Ein-Mensch-Projekt. Jemand hat eine Idee zu einem Projekt, setzt diese in Grundzügen um und veröffentlicht diese Basis. Nun finden sich andere Leute, die an dem Softwareprojekt Interesse haben und für dieses vielleicht selber einen Nutzen haben. Doch da jeder Nutzer einen leicht anderen Anwendungszeck hat, wird das Projekt erweitert und allgemeiner implementiert. Für diese Arbeiten melden sich weitere Entwickler/innen, die ihren Beitrag leisten wollen. Das Projekt wächst. Ohne genaue Planung ergeben sich doch einige Probleme: Es gibt keine klare Richtung in die sich die Software entwickeln soll, da keine Priorisierung der kommenden Features festgelegt ist. Jeder entwickelt nach seinem eigenen Belieben an dem Projekt. Außerdem ist die Einstiegshürde für neue Freiwillige extrem hoch. Niemand weiß, an welcher Stelle Neuerungen eingepflegt werden und wer festlegt, welche Änderungen tatsächlich in eine Release-Version übernommen werden. Eine Teststruktur, die für stabile Releases sorgt, ist ebenfalls nicht vorhanden. Diese Zustände werden für Unzufriedenheit sorgen und einer produktiven Weiterentwicklung des Projektes im Weg stehen.

Um die Meisten dieser Probleme zu beheben, empfiehlt es sich, eine Gewaltenstruktur einzuführen. Es wird genau festgelegt, wer das letzte Wort bei Entscheidungen hat und über welche Wege Entscheidungen getroffen werden. Werden alle Entscheidungen direkt an den Leiter übergeben? Oder gibt es eine Art „Bereichsleiter“, die eigene Entscheidungen treffen und diese nur im Zweifelsfall an den Hauptleiter weitergeben? Außerdem sollten Rollen verteilt werden. Nicht nur im Hinblick auf die hierarchischen Positionen, diese werden im nächsten Abschnitt genauer beschrieben, sondern auch auf die Zugehörigkeit in einem Bereich (Backend, Design, Dokumentation etc.) im Projekt. Um von den jeweiligen Mitglie-

dern des Projekts neue Beiträge einzugliedern, sollte genau geregelt sein, wie diese bereitgestellt werden müssen und wie über Annahme oder Ablehnung dieses Beitrages entschieden wird. Wichtig es ebenfalls, dass all diese Strukturen festgehalten und öffentlich verfügbar gemacht werden. [1]

1.2 Rollen

In den folgenden Abschnitten werden die typischen Rollen in Open-Source-Projekten kurz beschrieben. Je nach Gewaltenstruktur entscheiden sich jedoch die einzelnen Rollen ein wenig.

Der *Maintainer* ist die hierarchisch höchste Position in einem Open-Source-Projekt. Personen, die diese Position innehalten sind die einzigen, die entscheiden dürfen, welche Beträge in der Release Version eines Projektes veröffentlicht werden. Sie sind somit auch rechtlich für das Projekt verantwortlich und müssen im Zweifelsfall haften. Häufig handelt es sich bei den Maintainern um Programmierer/innen. Das muss jedoch nicht zwangsläufig so sein. Maintainer können sich komplett auf organisatorische Aufgaben beschränken. Es gilt das Projekt zu verbreiten, Entscheidungen zu treffen und die Verantwortlichkeit für die Community zu übernehmen.

Die *Contributer* sind Mitglieder der Community, die auf irgendeine Weise zu dem Projekt beigetragen haben. Auch hierbei beschränken sich die Aufgaben nicht nur auf Programmierarbeit. Zum Beispiel das Reagieren und Einordnen von gemeldeten Fehlern („Issues“) oder das Managen von Projektevents könnte zu den organisatorischen Aufgaben der Contributer gehören. Je nach Machtstruktur wird ein User bereits durch das Melden eines Fehlers zum Contributer.

Die *Committer* sind eine Untergruppe der Contributer. Sie besteht aus Personen, die bereits mehrfach und regelmäßig zum Projekt beigetragen haben und somit mit dem Projekt oder auch nur mit einem Teilbereich des Projektes sehr vertraut sind. Üblicherweise haben die Committer „Commitrechte“ auf der Codeverwaltung des Projektes. Sie können also eigenen Code oder Codebeiträge von Contributern direkt zum Projekt hinzufügen.

Die *User* sind die wichtigsten Mitglieder der Community. Ohne sie hätte das Projekt keinen Zweck. Neben dem Nutzen der Software unterstützen die User häufig noch im Support und Verbreiten das Projekt.

Weitere typische, aber seltenere Rollen, sind weitere, spezifischere Leiterrollen (auch nicht technische), spezifiziertere Arten der Contributer und „kleinere“ Rollen als der Contributer (z.B. reine Rolle für das Melden von Fehlern). [1]

2 Strukturen

In den folgenden Abschnitten werden die üblichsten Gewaltenstrukturen in Open-Source-Projekten genauer beschrieben. Dabei vergrößert sich mit jedem beschriebenen Modell die Anzahl der Personen die in den jeweiligen Projekten Entscheidungsgewalt besitzen.

Beim *Benovelent Dictator Governance Model* werden alle endgültigen Entscheidungen von einer einzigen Person getroffen. Dieses Modell wird häufig bei kleineren Projekten genutzt, da der Erfinder des Projektes sich das Recht vorbehält, über das gesamte Projekt zu verfügen. Die *Meritocratic Governance* erweitert den Kreis der Entscheidenden und wird häufig in mittelgroßen bis großen Projekten angewandt. Eine Gruppe von Leuten hat in dieser die Entscheidungsgewalt. Sämtliche Hierarchien versucht die *Liberal Contribution Policy* abzubauen. Alle Entscheidungen sollen hierbei durch möglichst viele Mitglieder der Community getroffen werden.

2.1 Benovelent Dictator Government Model

Beim *BDGM* wird das Projekt von einem Leiter gesteuert. Die Anforderungen an diesen sind sehr hoch. Er muss sehr gute Leiterfähigkeiten, wie Team-, Communitybuiling- und diplomatische Fähigkeiten,

und gleichzeitig ein sehr detailliertes und technisches Wissen in allen Bereichen des Projektes besitzen. Außerdem ist, gerade in größeren Projekten, sehr viel Einsatz und Hingabe erforderlich. Das ganze Projekt ist von diesem einen Leiter abhängig.

Trotz dieser Abhängigkeit muss die Community eine Möglichkeit haben, ihren Leiter zu kontrollieren. Diese Kontrolle entsteht bei Open-Source-Projekten durch die Möglichkeit das Projekt zu „forken“. Das bedeutet, den gesamten Code des Projekts zu kopieren und ein eigenes Projekt zu starten. Durch diese Möglichkeit ist der Leiter gezwungen, Entscheidungen zu fällen, die auch im Sinne der Community sind oder seine Entscheidungen zumindest so zu erklären, dass der Großteil der Community mit diesen einverstanden ist.

Der „Benovelent Dictator“ ist dabei der *Maintainer* und Leiter des Projektes. Dabei hat sich dieser typischerweise selbst dazu ernannt. In kleinen Projekten entsteht dies dadurch, dass der Erfinder des Projektes sich automatisch zum Leiter ernennt und in größeren Projekten hat jeder das Recht, das Projekt zu forken und sich selbst zum Leiter zu ernennen. Er ist der Verantwortliche für Projekt und Community und muss diese deshalb auch komplett verstehen. In den meisten Fällen ist er eher ein Diplomat als ein Diktator, da er Konflikte in der Community lösen und Lösungen für Probleme innerhalb des Projektes finden muss. Um seine eigenen Vorstellungen und Ziele des Projektes durchsetzen zu können, muss er diese der Community vermitteln können und den Leuten Einfluss geben, die seine Visionen teilen, damit diese die richtigen Entscheidungen für das Projekt treffen.

Wie in allen Modellen sind auch im BDGM die *Committer* Contributer, die bereits häufiger zum Projekt beigetragen haben. Zu ihren Aufgaben gehört es, eigenen Code beizusteuern und den Code von anderen Contributern zu kontrollieren. Um Committer zu werden arbeitet sich ein Contributer in den meisten Fällen in einen bestimmten Bereich des Projektes ein und wird Experte in diesem. Er erarbeitet sich dadurch den Respekt der Community und vor allem des Leiters. Falls dieser beginnt, den Contributer nach Rat und Unterstützung zu fragen, kann dieser in die Rolle eines Committers gehoben werden. Trotzdem hat er weiterhin keine endgültigen Entscheidungsrechte. Zu seinen weiteren Aufgaben gehört es, dem Diktator die Bedürfnisse und Ziele der Community bewusst zu machen und in dem ihm anvertrauten Bereich im Projekt Entscheidungen im Sinne des Leiters zu treffen.

Die *Contributer* sind alle Mitglieder der Community, die Beiträge beisteuern, aber nicht die Kompetenz eines Committers besitzen. Es gibt keine Voraussetzungen Contributer zu werden, wie zum Beispiel ein Mindestmaß an Hingabe, ein minimales Skillevel oder einen Auswahlprozess. Es ist nur ein Beitrag zu leisten, der das Projekt voranbringt. Zu den Aufgaben gehören unter anderem das Programmieren, Design, die Infrastruktur des Projektes zu unterstützen, Dokumentation schreiben, Bugs melden, Anforderungen ermitteln oder auch die User zu unterstützen.

Die *User* sind die wichtigsten Mitglieder der Community. Zu ihnen gehören alle, die einen Nutzen in dem Projekt haben. Obwohl sie sich nicht direkt an der Projektentwicklung beteiligen, erfüllen sie trotzdem wichtige Aufgaben in der Community. Überzeugte User verbreiten das Projekt ganz automatisch und vergrößern somit die Community. Sie informieren die Entwickler über Stärken und Schwächen des Projekts und beeinflussen somit auch die Entwicklung. Ein nicht unwesentlicher Teil ist auch der Support für die Entwickler, sowohl auf moralische als auch auf finanzielle Weise. Außerdem sind die ein Teil des Supports für andere User. Bei Fragen oder Hilfestellung ist die gesamte Community, von Maintainer bis User, dazu aufgefordert, den Enduser zu unterstützen.

Die *Entscheidungsfindung* im BDGM ist kein kompliziertes System. Committer dürfen Entscheidungen in ihren eigenen Bereichen nach Vorgabe des Leiters treffen. Dazu gehören Entscheidungen, welche neuen Funktionen implementiert, auf welche Weise Fehler behoben und welche Codebeiträge der Contributer angenommen werden. Zweifelt ein Contributer oder ein anderer Committer diese Entscheidung an, wird diese dem Leiter gemeldet. Er hat im Zweifelsfall das letzte Wort und kann die ursprüngliche Entscheidung des Committers rückgängig machen. [2]

2.2 Meritocratic Governance

Im Meritocratic Governance Model haben eine Reihe von ausgewählten Community Mitgliedern die Entscheidungsgewalt. Innerhalb dieser Mitglieder wird über Entscheidungen abgestimmt. Trotz dieser Abstimmung ist das meritokratische Modell keine Demokratie. Abstimmungsrecht bekommt man nur, wenn man genug wertvolle Beiträge geleistet hat und mit den Visionen der bisherigen Entscheider übereinstimmt. Obwohl es mehrere Entscheider gibt, bleibt die Struktur flach, da alle Entscheider die gleiche Autorität haben. Die Struktur des meritokratischen Modells entsteht häufig in Projekten, die mit einem Entscheider begonnen haben. Dieser sucht sich jedoch mit der Zeit Gleichgesinnte und teilt seine Rechte, da er nicht mehr alleine über das Projekt bestimmen möchte. Auch in diesem Modell werden die ausgewählten Mitglieder kontrolliert durch die Möglichkeit der Community das Projekt zu forken. Stimmt ein Großteil der Community nicht mehr mit den Beschlüssen der Entscheider überein, beginnen diese Mitglieder der Community einfach auf der gleichen Codebasis ihr eigenes Projekt und zwingen die ausgewählten Mitglieder somit, Entscheidungen im Sinne der Community zu treffen.

Da das meritokratische Modell sehr viel Ähnlichkeit zum BDGM hat, unterscheiden sich die Aufgaben in den unteren Bereichen der Hierarchie, und somit der User und der Contributer, kaum. Beim *Committer* jedoch werden erste Unterschiede sichtbar. Wie beim BDGM zeigt auch im meritokratischen Modell die Rolle des Committers, dass bereits viele wichtige Beiträge beigesteuert wurden. Die Committer haben direkt Zugriff auf den Code ohne Umwege über Pull Requests. Trotzdem werden ihre Änderungen vor Veröffentlichung im offiziellen Release begutachtet und sie haben somit nicht mehr Rechte als die Contributer. Der Unterschied ist nur die Reihenfolge zwischen Commit und Begutachtung. Bei Contributern wird der Code begutachtet und bewertet und anschließend bei positivem Feedback in die Codeverwaltung eingepflegt. Bei Committern ist dies andersrum: Der Code wird zunächst hinzugefügt und erst hinterher im Bedarfsfall bewertet. Dieses Verfahren soll für mehr Effizienz sorgen.

Um vom Contributer zum Committer aufzusteigen müssen eine Reihe von Voraussetzungen erfüllt sein: Es müssen regelmäßig wertvolle Beiträge geliefert worden und der Wille vorhanden sein, dies auch weiterhin zu tun. Außerdem werden die technische und ein gewisses Maß an Teamfähigkeit vorausgesetzt. Der Kandidat sollte ebenfalls ein grundlegendes Verständnis für das gesamte Projekt, seiner Ziele und Strategien haben. Jeder Committer kann einen Contributer, der diese Voraussetzungen erfüllt, nominieren. Das Project Management Comittee (s. unten) stimmt geheim über die Ernennung ab. Das Ergebnis dieser Abstimmung wird anschließend veröffentlicht. Der Kandidat hat das Recht auf eine Erklärung zu jeder Nein-Stimme, die vom Vorsitz der Project Management Comittees abgegeben wird. Ist die Abstimmung jedoch positiv, wird der Kandidat zum Committer befördert.

Das *Project Management Comittee* („PMC“) besteht aus den ausgewählten Mitgliedern, die Entscheidungsgewalt innerhalb des Projektes haben. Zu ihren Aufgaben gehört es Codebeiträge zu begutachten, an der strategischen Planung des Projektes teilzunehmen, Änderungen an Kontrollstrukturen zu beschließen, die Klärung von rechtlichen Fragen und, wie oben beschrieben, die Ernennung neuer Committer. Trotzdem haben die Mitglieder in den meisten Fällen nicht mehr Rechte als der Rest der Community, da das PMC häufig nur bei zwiespältigen Diskussionen unter den Contributern das letzte Wort haben. Für die Mitglieder des PMC besteht ein privater Kommunikationskanal. Hierüber laufen jedoch nur personenbezogenen Diskussionen und Abstimmungen und die Klärung rechtlicher Angelegenheiten, die nicht in der Öffentlichkeit besprochen werden können. Alle andere Kommunikation, wie Projektmanagement und Planung, findet unbedingt öffentlich statt. Neue Mitglieder können über eine Nominierung von einem derzeitigen PMC Mitglied bestimmt werden. Auch hier stimmt das gesamte Komitee über die Ernennung ab.

Das gesamte PMC wird von einem Vorstand (*PMC Chair*) geleitet. Auch dieser wird von den restlichen Mitgliedern gewählt. Er hält dieses Amt solange inne, bis er Abtritt oder von einer zwei Drittel Mehrheit der Mitglieder abgewählt wird. Er hat keine besondere Autorität über die anderen Mitglieder, sondern handelt eher als Koordinator und Moderator.

Entscheidungen werden in dem meritokratischen Modell meisten durch Diskussionen in der Community getroffen. Um einer sehr lange Entscheidungsfindung vorzubeugen, wird nach dem „Lazy-Consensus"-Prinzip entschieden. Dabei wird ein Diskussionspunkt oder ein Codebeitrag öffentlich eingebracht. Alle Mitglieder der Community sind aufgefordert den Beitrag zu prüfen und über ihn zu diskutieren. Dabei sollten die Mitglieder eine Entscheidung fällen. Kommen jedoch keine oder nur positive Rückmeldungen, gilt der Beitrag nach einem bestimmten Zeitraum automatisch als bewilligt. Kann über dieses System keine endgültige Entscheidung getroffen werden, stimmen die Mitglieder des PMC über diesen Punkt ab. [3]

2.3 Liberal Contribution Policy

Die Liberal Contribution Policy („LCP") versucht ein System aufzubauen, bei dem sämtliche Hierarchien abgebaut sind und die Entscheidungsgewalt in der gesamten Community liegt. Das heißt, alle Mitglieder haben die gleichen Rechte.

Der Schritt vom User zum *Contriuter* passiert in der LCP schneller als in den bisherigen Modellen. Sobald ein Nutzer einen Fehler meldet, auf diesen reagiert oder einen Beitrag leistet oder diesen auch nur bewertet wird er zum Contributer ernannt.

Auch für die Beförderung zum *Committer* muss der Contributer weniger Aufwand betreiben. Es reicht der Beitrag einer Lösung einer einzigen nicht-trivialen Aufgabe um Schreibrechte auf der Codeverwaltung zu erlangen. Diese schnelle Erweiterung der Rechte eine Contributers soll Anreize schaffen auch weiterhin wertvolle Beiträge für das Projekt zu schaffen. Bei zerstörerischen Beiträgen können Änderungen bei heutigen Codeverwaltungen ganz einfach wieder rückgängig gemacht werden.

Obwohl möglichst alle Entscheidungen von der gesamten Community getroffen werden sollten, gibt es eine Instanz oberhalb der meisten Mitglieder. Das *Technical Committee* („TC") besteht aus einer Gruppe von Committern mit besonderem technischen Fachwissen. Es wird genutzt um festgefahrene Konflikte in der Community zu lösen.

Damit ein Contributer einen Beitrag leisten kann, muss er diesen durch einen Pull Request anmelden. Jeder geleistete Beitrag zieht eine Diskussion und Überprüfung der gesamten Community nach sich. Kein Committer darf Code ohne Diskussion annehmen. Auch in der LCP laufen Diskussionen nach dem Lazy-Consensus-Prinzip ab, das bedeutet, nach dem Verstreichen einer Frist, gilt ein Pull Request automatisch als angenommen. Allerdings können Mitglieder der Community ein „LGTM" („Looks good to me") von einem Experten verlangen. Dieser Experte muss dann zunächst sein Einverständnis geben, damit der Code angenommen wird. Ein Beispiel dafür ist die Implementierung von sicherheitskritischem Code. Wird ein LGTM von einem Mitglied verlangt, muss zunächst mindestens ein Experte auf dem Gebiet Sicherheit den Beitrag begutachten und für gut befinden. Sollte ein Problem nicht von der Community gelöst werden können, wird das Problem an das TC weitergeleitet. Dies sollte jedoch nur in wenigen Fällen nötig sein.

Ein Problem, das im TC ankommt wird versucht einstimmig zu lösen. Falls es auch so zu keiner Einigung kommt, wird unter den Mitgliedern darüber abgestimmt. Eine weitere Möglichkeit, das Problem zu lösen, ist, dass der TC das Problem zurück an die Community gibt, mit Hinweisen, wie das Problem gelöst werden könnte.

Neue Mitglieder im TC können von jedem Committer nominiert werden. Das TC entscheidet dann mit dem oben beschriebenen Prozess über die Ernennung des Kandidaten. Von Mitgliedern, die nicht mehr regelmäßig beitragen wird erwartet, dass sie zurücktreten. [4]

3 Fazit

Alle drei vorgestellten Modelle der Gewaltenstrukturen in Open-Source-Projekte haben ihre Vor- und Nachteile und für alle gibt es funktionierende Beispiele.

Das BDGM wird genauso wie beschrieben für das Linux-Kernel-Projekt verwendet, das von Linus Torvalds geleitet wird. Das BDGM benötigt einen einzelnen sehr talentierten Leiter, der sowohl einzigartige technische als auch soziale Kompetenzen besitzen muss. Ist dieser jedoch gefunden, kann diese Modell mit seiner einfachen und sehr klaren Struktur sehr gut funktionieren.

Die Meritocratic Governance hat eine klarere und transparentere Entscheidungsstruktur. Obwohl auch hier keine Demokratie herrscht kann jedes Mitglied der Community Teil des Entscheidungskomitees zu werden. Sämtliche Projekte der Apache Foundation zeigen, dass dieses System trotz steigender Bürokratie gut funktionieren kann.

In der Liberal Contribution Policy hat jeder die gleichen Rechte. Das bringt viele Vorteile für jeden Einzelnen kann aber eine Last sein, wenn das Projekt nicht vorankommt, Entscheidungen keine klare Linie haben oder Diskussionen endlos werden. Darauf muss geachtet und ein System geschaffen werden, dass diese Probleme möglichst beseitigt. Wenn es jedoch funktioniert, kann die LCP für eine sehr zufriedene Community sorgen und die Node.js Foundation ist ein gutes Beispiel für dessen Funktionsweise.

Die Frage nach dem richtigen Modell ist sicherlich von den gegebenen Umständen in jedem einzelnen Projekt abhängig. Trotzdem denke ich, dass mit dem meritokratischen System für die meisten Projekte ein guter Mittelweg gefunden ist. Es herrscht nicht der Eindruck einer Diktatur, aber es gibt trotzdem eine Gruppe von Entscheidern, die im Zweifelsfall dafür sorgen, dass das Projekt mit einer klaren Richtung weiterentwickelt wird.

Literatur

[1] *Leadership and Governance*. URL: https://opensource.guide/leadership-and-governance/ (besucht am 08. 06. 2018).

[2] Ross Gardler und Gabriel Hanganu. *Benevolent Dictator Governance Model*. URL: http://oss-watch.ac.uk/resources/benevolentdictatorgovernancemodel (besucht am 08. 06. 2018).

[3] Ross Gardler und Gabriel Hanganu. *Meritocratic Governance Model*. URL: http://oss-watch.ac.uk/resources/meritocraticgovernancemodel (besucht am 08. 06. 2018).

[4] Mikeal Rogers. *Healthy Open Source*. URL: http://medium.com/the-node-js-collection/healthy-open-source-967fa8be7951 (besucht am 08. 06. 2018).