**目次**

[第1章 序論 0](#_Toc378534718)

[1.1. 本章の構成 1](#_Toc378534719)

[1.2. 研究背景 1](#_Toc378534720)

[1.3. 研究目的 2](#_Toc378534721)

[1.4. 研究方法 2](#_Toc378534722)

[1.5. プロジェクトマネジントとの関連 2](#_Toc378534723)

[1.6. 本論文の構成 2](#_Toc378534724)

[第2章 チケットについて 2](#_Toc378534725)

[2.1. 本章の構成 3](#_Toc378534726)

[2.2. チケットとは 3](#_Toc378534727)

[2.2.1. チケットの作成 5](#_Toc378534728)

[2.2.2. チケットの表示 6](#_Toc378534729)

[2.2.3. チケットに含まれるデータ 7](#_Toc378534730)

[2.3. チケット駆動開発 8](#_Toc378534731)

[2.3.1. チケット駆動開発の由来 8](#_Toc378534732)

[2.3.2. チケット駆動開発とは 8](#_Toc378534733)

[2.3.3. チケット駆動開発の開発サイクル 9](#_Toc378534734)

[2.3.4. チケット駆動開発と開発プロセス 10](#_Toc378534735)

[2.3.5. チケット駆動開発のメリット 11](#_Toc378534736)

[2.3.6. チケット駆動開発のデメリット 11](#_Toc378534737)

[第3章 GitHubについて 11](#_Toc378534738)

[3.1. 本章の構成 12](#_Toc378534739)

[3.2. 分散型バージョン管理システム 13](#_Toc378534740)

[3.3. Gitについて 14](#_Toc378534741)

[3.3.1. Gitの特徴 14](#_Toc378534742)

[3.4. GitHubとは 15](#_Toc378534743)

[3.4.1. GitHubの基本用語 15](#_Toc378534744)

[3.4.2. GitHubの機能 17](#_Toc378534745)

[3.5. APIについて 18](#_Toc378534746)

[3.5.1. APIを公開する・公開されるメリット 18](#_Toc378534747)

[3.6. GitHubのAPIの種類 19](#_Toc378534748)

[第4章 調査・開発 31](#_Toc378534749)

[4.1. 本章の構成 32](#_Toc378534750)

[4.2. 調査対象 32](#_Toc378534751)

[4.2.1. 調査対象データ 32](#_Toc378534752)

[4.2.2. 調査対象プロジェクト 33](#_Toc378534753)

[4.3. 環境構築・調査 34](#_Toc378534754)

[4.3.1. 調査環境構築 34](#_Toc378534755)

[4.3.2. 調査方法 36](#_Toc378534756)

[4.3.3. 調査 39](#_Toc378534757)

[4.3.4. 調査プログラムの実行 42](#_Toc378534758)

[第5章 調査結果考察 42](#_Toc378534759)

[5.2.1. 調査結果グラフ 43](#_Toc378534760)

[5.2.2. グラフのグループ分け・解析 62](#_Toc378534761)

図目次

[図 2‑1チケットのフロー 3](file:///C:\Users\kubo\Documents\GitHub\yabukilab\卒業論文\2012\久保孝樹\卒業論文(kubo).docx#_Toc378534762)

[図 2‑2チケットの例 5](#_Toc378534763)

[図 2‑3チケットの表示の例 6](#_Toc378534764)

[図 2‑4チケット駆動開発の開発サイクル 9](#_Toc378534765)

[図 2‑5開発プロセスとチケット駆動開発の関係 10](#_Toc378534766)

[図 3‑1分散型バージョン管理システム 13](file:///C:\Users\kubo\Documents\GitHub\yabukilab\卒業論文\2012\久保孝樹\卒業論文(kubo).docx#_Toc378534767)

[図 5‑1　Issueの時間変化グラフ/brackets 43](#_Toc378534768)

[図 5‑2 　Issueの時間変化グラフ/angular.js 44](#_Toc378534769)

[図 5‑3　 Issueの時間変化グラフ/phantoms 44](#_Toc378534770)

表目次

[表 2‑1チケットに含まれるデータ 7](#_Toc369191316)

[表 3‑1 IssueのAPIと意味 19](#_Toc369191317)

[表 3‑2 Issue EventsのAPIと意味 19](#_Toc369191318)

[表 3‑3 Issue CommentsのAPIと意味 20](#_Toc369191319)

[表 3‑4 EventsのAPIと意味 20](#_Toc369191320)

[表 3‑5 LabelのAPIと意味 21](#_Toc369191321)

[表 3‑6 GistsのAPIと意味 22](#_Toc369191322)

[表 3‑7 Gists CommentsのAPIと意味 22](#_Toc369191323)

[表 3‑8 GitDataのAPIと意味 23](#_Toc369191324)

[表 3‑9 Repo DownloadのAPIと意味 24](#_Toc369191325)

[表 3‑10 Repo Deploy KeyのAPIと意味 24](#_Toc369191326)

[表 3‑11 Repo ForkのAPIと意味 24](#_Toc369191327)

[表 3‑12 Repo HookのAPIと意味 25](#_Toc369191328)

[表 3‑13 PullRequestのAPIと意味 25](#_Toc369191329)

[表 3‑14 Pull Request Review CommentのAPIと意味 26](#_Toc369191330)

[表 3‑15 MilestoneのAPIと意味 26](#_Toc369191331)

[表 3‑16 UserのAPIと意味 26](#_Toc369191332)

[表 3‑17 User EmailのAPIと意味 27](#_Toc369191333)

[表 3‑18 User FollowerのAPIと意味 27](#_Toc369191334)

[表 3‑19 User Public KeyのAPIと意味 27](#_Toc369191335)

[表 3‑20 ReposのAPIと意味 28](#_Toc369191336)

[表 3‑21 Repo CommitのAPIと意味 29](#_Toc369191337)

[表 3‑22 Repos WatchingのAPIと意味 29](#_Toc369191338)

[表 3‑23 Repo CollaboratorのAPIと意味 30](#_Toc369191339)

[表 3‑24 Organization TeamのAPIと意味 30](#_Toc369191340)

[表 3‑25 OrganizationのAPIと意味 31](#_Toc369191341)

[表 3‑26 Organization MembersのAPIと意味 31](#_Toc369191342)

第1章

序論

# 序論

* 1. 本章の構成

　第1章では，本論文の序論を述べる．研究背景，研究目的，研究方法，プロジェクトマネジメントの関連，本論文について記述する．

* 1. 研究背景

近年スマートフォンやタブレット端末などの急速な普及により，ウェブアプリケーションやソーシャルゲームなどのソフトウェア開発が果敢に行われるようになった．そのようなソフトウェア開発プロジェクトでは開発中に顧客の要求や開発環境が変化し続けてしまうため，変化していく顧客の要求や開発環境の変化に柔軟に対応していかなければならない．そのため，開発プロセスもそのような事柄を考慮したものが求められている．

従来の開発プロセスであるウォーターフォール型では，要求定義や，外部設計，内部設計，開発，テスト，運用などの作業を各工程に分割して実行し，原則として前工程が終了しなければ次工程に進めることができない．顧客の要求の変化や環境への変化に対応していくためには，手戻りをしなくてはならないのである．そのため，要求や環境が変化すると，必然的にプロジェクトは遅延し，コストは超過してしまう．このような問題の解決策として，アジャイル型の開発プロセスが注目されているのである．アジャイル型の開発プロセスでは，1回のイテレーションを短い周期で行い，その中で要求定義，開発，テストを行うため，顧客の要求や開発環境が変化した場合，次イテレーションにて対応することが可能であり，手戻りを起こすことはなく，大きな遅延を起こすことはなくなる．アジャイル型の開発プロセスでは，環境の変化，顧客の要求やバグの修正など，様々なタスクが発生する．それらに柔軟に対応するため，発生したすべきことを明記する「チケット」と呼ばれるツールが活用され，チケットを中心として開発を行う「チケット駆動開発」というものが提唱されている．チケットは，すべきことと報告者，担当者，優先度，マイルストーン，種類，状況，コンポーネント，解決法，その他詳細をひとまとめにしてWeb上で共有するための仕組みである．Web上で管理されるため，プロジェクトメンバはいつでもチケットを参照，更新することが出来る．また，チケットと同様な機能を持つツールはソフトウェア開発を支えるツールの中に存在し，本論文ではGitHubに注目しその中で，チケットとして利用されるIssueを利用したバグ管理，バージョン管理を調査し，それらの環境で利用されているチケットはソフトウェア開発において大変便利なツールであると考えられ，チケットによってプロジェクトのスコープや進捗の管理，プロジェクトメンバの管理が効率的に行うことができ，チケットを利用することで，プロジェクトの成功率を向上させることが出来るのではないかと考えた．

* 1. 研究目的

チケットがどのようにソフトウェア開発プロジェクトで使用され，役立っているのかを調査することによってプロジェクトマネジメントにおいてどのように利用することで，プロジェクトの効率向上でき，成功率を上げることが出来るのかを知りたい．そのために，ソフトウェア開発プロジェクトにおいて，最もよく使われているバージョン管理サービスであるGitHubを利用し，GitHubにおけるチケット（Issueと呼ばれるもの）の使われ方を調査する．具体的には，GitHubで公開されているソフトウェア開発プロジェクトにおいて，Issueに関するデータを抽出し，Issueがどのように利用されているのか，Issueを中心として開発しているプロジェクトがどのくらい存在するのか，どのようなプロジェクトで使われているのかなどを調査することで， チケットがどのように活用されるかを調査し，ソフトウェア開発プロジェクトにおいてどのように便利なのかを明らかにする．

* 1. 研究方法

GitHubで公開されているソフトウェア開発プロジェクトにおけるIsuueに着目し，チケットが具体的にどのようなものなのか，どのように利用されているのか，どのようなメリット，デメリットがあるのかを調査する．また，Issueに含まれるデータの調査，理解をし，調査に必要であるデータの項目を洗い出し，それらのIssueに含まれるデータを収集するためのツールを開発し，データを解析しチケットの使われ方を明らかにする．その上で，チケットを利用することで，ソフトウェア開発プロジェクトにどのように影響を与えるのかを調査する．

* 1. プロジェクトマネジントとの関連

チケットはPMBOKが提唱する，スコープマネジメントに最も関連があるものである．チケット利用により，次々に発生するタスクの整理が出来，やるべきことを明確に可視化することが出来る．プロジェクトマネジメントにおいて，タスクを洗い出し，可視化することは重要であり，それらを効率的に行う事が出来るものがチケットであると考える．

* 1. 本論文の構成

第1章では序論，第2章では本論文の調査目的であるチケットについて，チケットがどのようなものなのか，チケットを利用した開発である，チケット駆動開発がどのようなものであり，どのように行われているのかなどを記述する．第3章ではバージョン管理システムについて，Gitを中心として記述し，本調査のチケットのデータを抽出する対象である，GitHubについての解説，GitHubのAPIの説明を記述する．第4章ではプロジェクトマネジメントとの関係を記述する．第5章では具体的な調査方法，調査経過，データを示し，データ解析を行い，第6章で考察，まとめを行う．

参考文献

[1] 小川明彦, 酒井誠. チケット駆動開発. 翔泳社, 2012-8-23.

[2] Project Management Institute, Inc. プロジェクトマネジメント知識体系ガイド(第4版). PMI, 2009-12.

[3] Jonathan Rasmusson. アジャイルサムライ―達人開発者への道. オーム社, 2011-11-25.

[4] 濱野純.入門Git, 秀和システム, 2009-9-25.

[5] 片岡巌. WEB+DB PRESS. 技術評論社, 2012-7-25, Vol.69. p.18-52.

第２章

チケットについて

# チケットについて

* 1. 本章の構成

本章では本論文の調査対象であるチケットの基本知識や使われ方を述べる．またチケットを利用したソフトウェア開発であるチケット駆動開発の基本知識並びに使用法などを述べる．

* 1. チケットとは

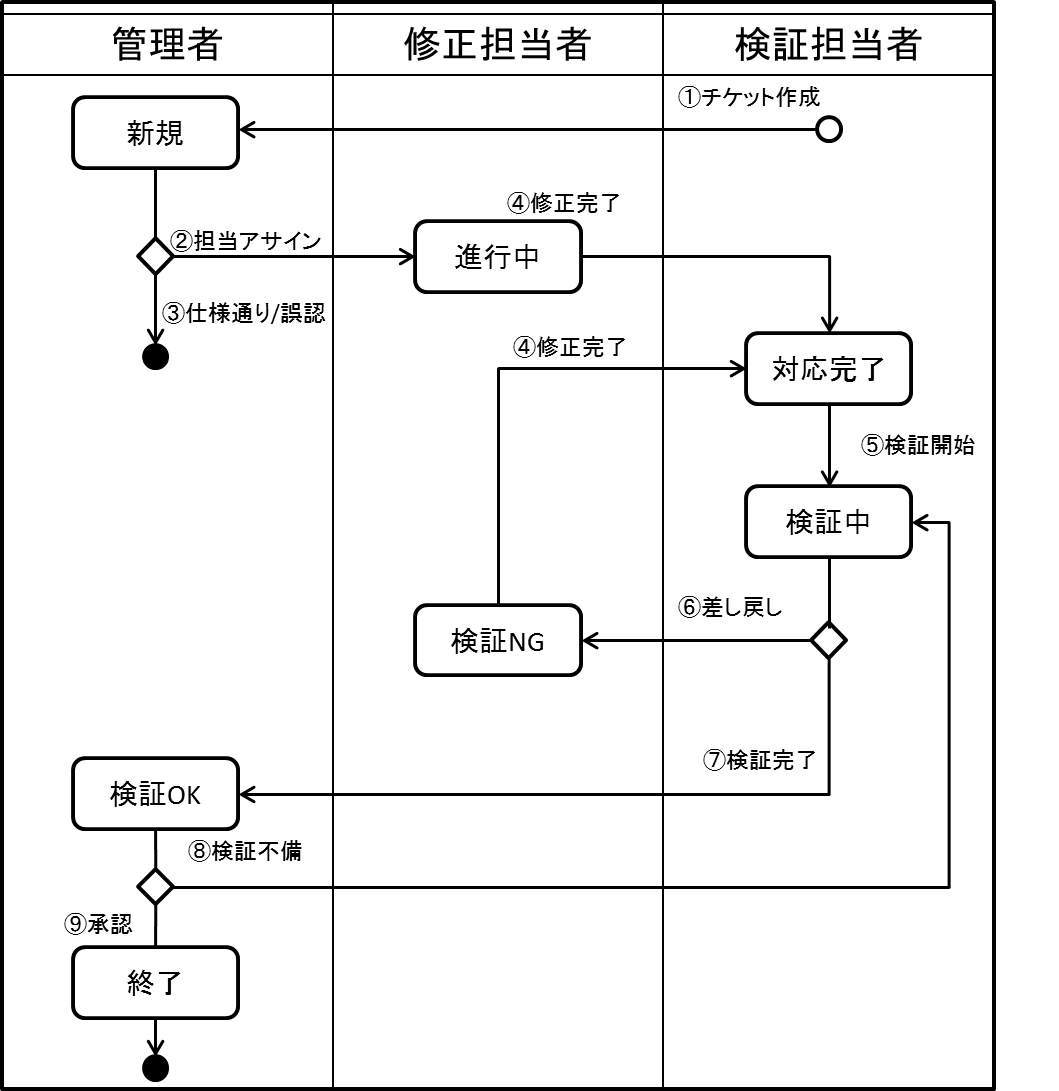
チケットは課題管理システム（Issue Tracking System＝ITS）バグ管理システム（Bug Tracking System＝BTS）において使われるツールであり，1つの課題や，バグを管理する単位を「チケット」と呼び，それぞれ課題やバグが発生したときに発行されるものである．チケットは発行された時点ではオープンとなり，修正など施され終了された時にはクローズとなる．チケットは以下のようなフローで処理される．

図 2‑1チケットのフロー

チケットのフロー

1. チケット作成［→ 新規］

検証担当者はバグを発見すると，バグの詳細を記述したチケットを新規に作成します．作成の際に担当者を管理者にすることで，バグが発見された通知が管理者に送られます．

1. 担当アサイン［新規 → 進行中］

管理者はチケットに記述されたバグ情報を確認し，修正担当者をアサインします．

1. 仕様通り／誤認［新規 → 却下］

チケットに記述されたバグ情報を確認し，そもそも仕様通りの場合や検証担当者の誤認だった場合は対応不要のため，理由を記述したうえでチケットのステータスを"却下"に変更し，終了します．

1. 修正完了［進行中 → 対応完了］［検証NG → 対応完了］

修正担当者はバグの修正対応を行い，その対応の詳細をチケットに記述します．再度検証を行うため，担当者をテスト担当者に変更します．

1. 検証開始［対応完了 → 検証中］

対応完了になっているチケットから，検証を開始するチケットを選択し「検証中」に変更します．

1. 差し戻し［検証中 → 検証NG］

バグの修正対応が不十分であった場合，「検証NG」として差し戻しを行います．

1. 検証完了［検証中 → 検証OK］

バグの修正対応に問題がないと確認できた場合「検証OK」とし，管理者に最終承認を行ってもらいます．

1. 検証不備［検証OK → 検証中］

検証結果を確認し，検証内容に不備があった場合や，追加で検証を行ってほしい項目などがあれば， 検証担当者に追加の検証を行ってもらいます．

1. 承認［検証OK → 終了］

検証結果に問題がなければ承認とし，チケットのステータスを「終了」にします．

* + 1. チケットの作成

チケットの作成は基本的にメンバーのだれでも行うことが出来，すべてのメンバーが発見したこと，報告をチケットで表すことが出来る．チケットの入力項目としては，タイトル，報告者，担当者，マイルストーン，優先度，属性，ステータス，内容詳細などを記載する

1つのタスクを1つのチケットとして登録し，各チケットにそれぞれコメントなどを書くことで，進捗管理や問題解決の議論などが行える．またチケットは柔軟な表現が可能で強調したい部分の文字の色を変えたり，文字の太さを変えたりすることが可能である．

以下の図はチケットの例として，GitHubでチケットと同様な機能を持ち使用されているIssueである．

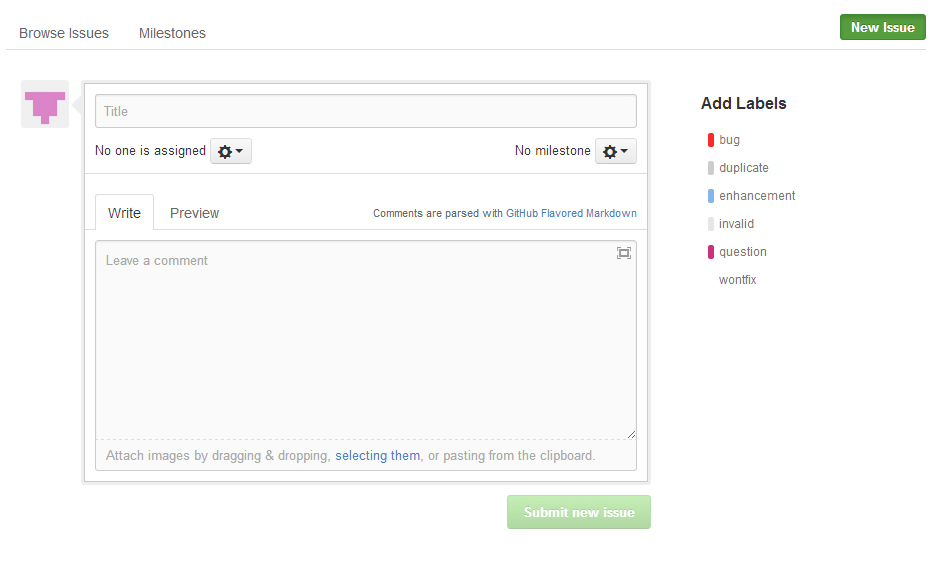


図 2‑2チケットの例

* + 1. チケットの表示

作成されたチケットは，レポート機能やクエリ機能で一覧表示や検索ができ，担当者，優先度，マイルストーンを設定することで，担当者別に表示したり，優先度順に並び替えたりなどが可能であり，誰がどのタスクを行っていくのか，いつまでに，どの順序で行うべきかを可視化することが出来るのである．

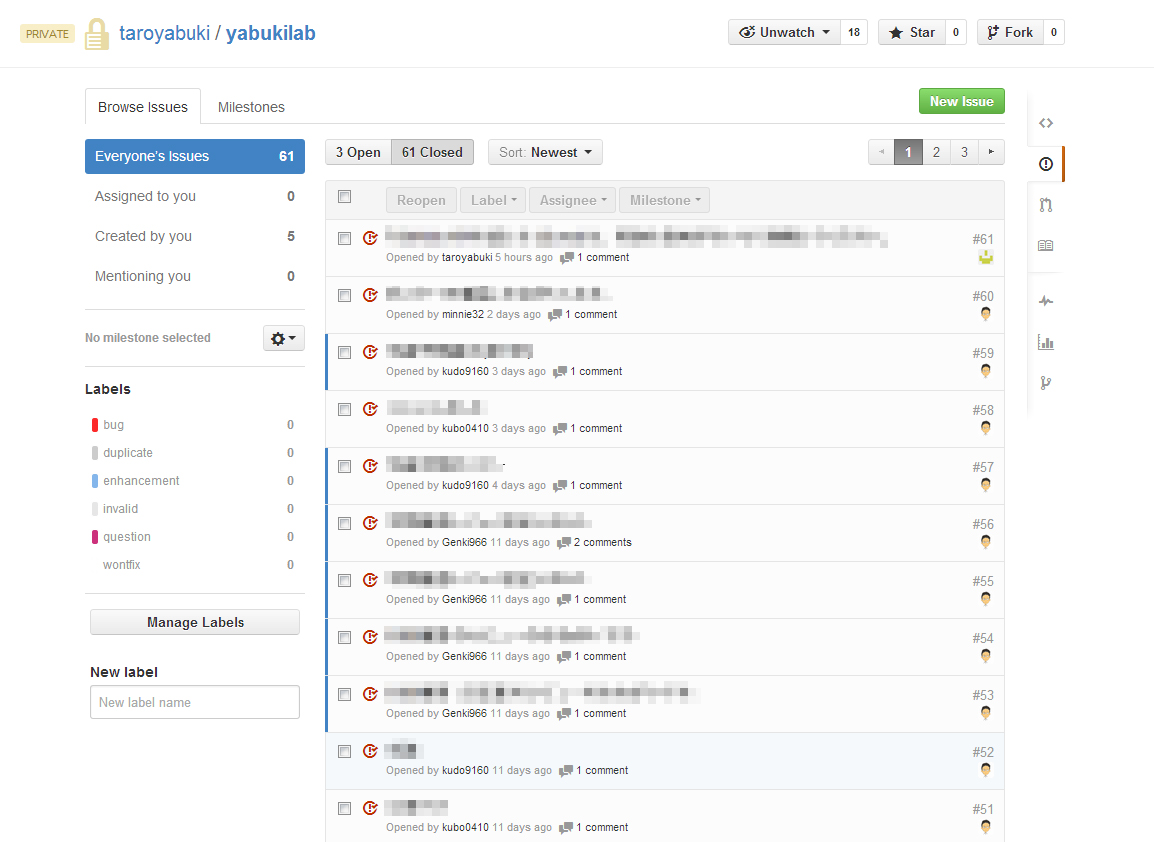


図 2‑3チケットの表示の例

* + 1. チケットに含まれるデータ

　チケットにはメンバーに伝えるために様々なデータが含まれる．チケットがどのようなものなのか，どのような状態なのかがわかるようになっている．

チケットに含まれるデータ，項目を以下の表に記述する．

表 2‑1チケットに含まれるデータ

|  |  |
| --- | --- |
| 項目名 | 内容 |
| タイトル | チケットのタイトルが記載され，どのような内容なのかわかるようにする． |
| 報告者 | チケットを発行したメンバーの名前が記載される． |
| 担当者 | チケットを割り振れられた担当者の名前が記載される． |
| チケット番号 | チケットが何番目に作られたのか，チケットの番号を表す． |
| マイルストーン | いつまでに作業を終わらせなければならないか期限を表す． |
| 優先度 | どのチケットを優先に行うべきかそのチケットの優先度を表す． |
| 属性 | そのチケットがどのような種類なのかを表す．バグ，質問，リクエストなどの種類を表す． |
| ステータス | チケットがオープンなのか，担当者が割り振られているのか，クローズなのかなど今どのような状態なのかを表す． |
| コメント・詳細 | チケットに対する詳細をコメントしたり，自由にコメントをしたり，伝えたいことを記載することが出来る． |
| 日時 | チケットがいつ発行されたのか，いつ更新されたのか，いつ終了したのかが表される． |

* 1. チケット駆動開発

　チケット駆動開発とは，チケットを開発の中心とし，開発を行っていく手法である．

* + 1. チケット駆動開発の由来

チケット駆動開発の基本となる構成管理と，障害管理の提携はツールを何も用いずに1980年代から行われていた．構成管理ではベースラインを決め，そこからの変更を管理し，障害は障害票で管理され連番が振られ，ベースラインからの変更理由として障害票の連番が必要とされていた．このようにチケット駆動開発の基本的な考えは20世紀から実践され，それらはプロジェクト管理の視点で行われていた．また，それらはすべてが自動化されておらず，手動で行われてきた．

チケット駆動開発は障害管理ツールであるTracの利用者により始まった．Tracで行われた理由には「チケットの種類に『タスク』が始めからあった」「構成管理ツールとチケットで簡単に連携できた」「Wikiが組み込まれており，その記法としてチケット番号やチェンジセットへのリンクがあった」というTracの特徴があった．

これらによって一般の開発が考慮され，管理者だけでなく開発者の利便性を意識した作りになり，Tracでの開発が開発者の中で話題となり，Tracを中心にし，これまで手作業で行ってきた作業をオンラインで行い，リアルタイムコミュニケーションが可能で効率的な開発が提唱され，その中で2007年に開催された「ITpro Challenge!」で，まちゅ氏がチケット駆動開発を提唱したのが始まりである．このようにチケット駆動開発は現場で開発する人から始まったのである．

* + 1. チケット駆動開発とは

チケット駆動開発は障害管理システムであるBTS（Bug Tracking System）を中心に用いたタスク管理が基本であり，チケットを開発の中心として行う開発方法である．障害管理ツールを用いてタスクを管理して開発を進めていくため，作業の漏れが少なくなり，予想外の作業が発生した場合も，メンバー間で共有して協力しながら計画的に進めることが出来る．

　チケット駆動開発を行う際のルールは基本的に1つであり「チケットなしのコミットは禁止『No Ticket, No commit!』」であり，個人的な管理ではなく構成管理上のコミットをしないということである．このルールによって開発メンバーの仕事が把握しやすくなるだけでなく，成果物の更新が必ずチケットと関連付くので変更理由が明確になる，というチケット駆動開発のメリットが生まれるのである．

* + 1. チケット駆動開発の開発サイクル

　チケット駆動開発では以下のようなサイクルを繰り返して開発が行われる．

1. 大まかなリリース計画を作る．
2. 仕事を細かいタスクに分割し，タスクを書き出す．(チケットの発行)
3. イテレーション単位でタスクをまとめて，イテレーション計画を作る．
4. タスクを一つ選び，実装する．
5. 差分をコミットし，完了する．(チケットのクローズ)
6. イテレーションに紐づくタスクがすべて終了ステータスになるとリリースする．
7. リリース後，開発チームで作業をふりかえる．
8. 次のイテレーション計画へ顧客の要望やふりかえりの内容を反映する

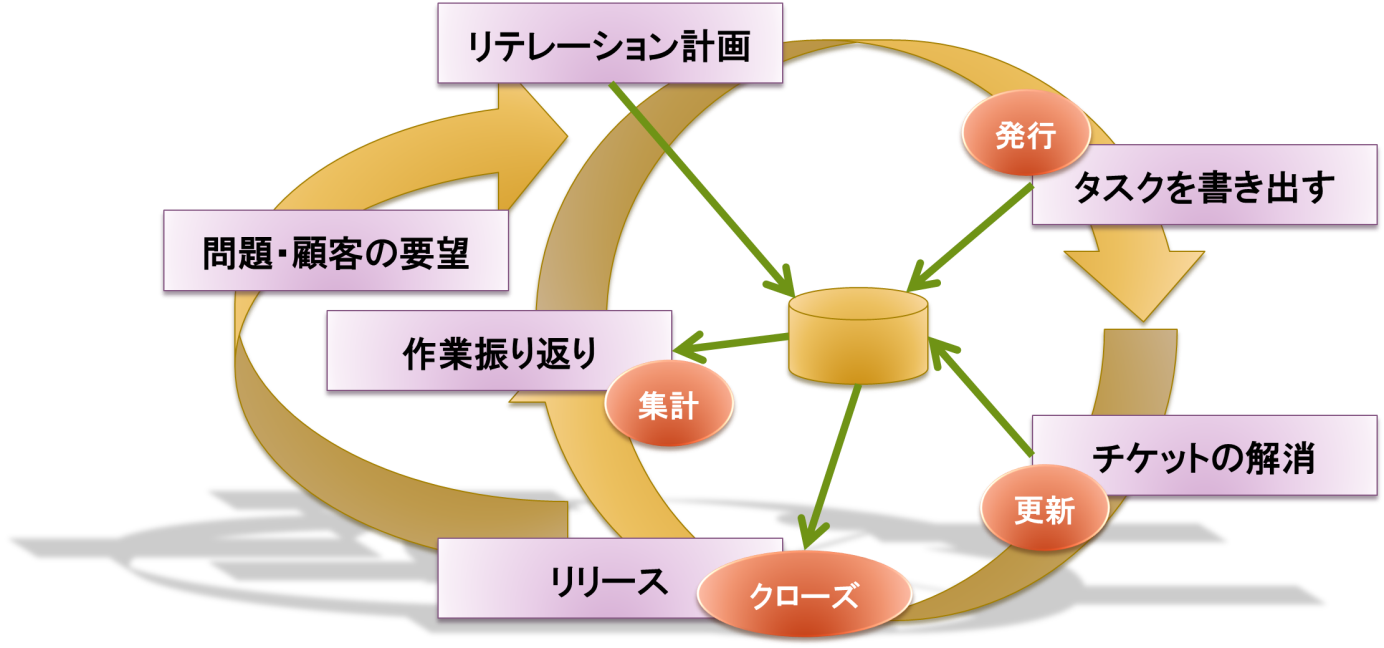


図 2‑4チケット駆動開発の開発サイクル

* + 1. チケット駆動開発と開発プロセス

ウォーターフォール型の開発は段階的詳細化を組織的に行うトップダウンの開発法であり，全工程で定めたことを前提に次工程を行わなければならなく，そのプロセスは機能的で，構造化分析・設計のように全体が1つのシステムとして実現されている．統率のとれた組織が実現できるため大規模開発が可能であるが，仕様変更の多い開発にはあまり向いていない．

アジャイル型の開発は実行可能なソフトウェアを順次開発し，常に顧客に価値を提供する開発方法であり，オブジェクト指向的で，独立したオブジェクトの組み合わせにより全体を構成する．細かい1度の開発の単位であるイテレーション単位に開発を行い，各イテレーションを始める前に新しく計画を立てる．そのため，柔軟に計画の変更に対応することが可能となっている．またタスクボードやミーティングなどコミュニケーションを重視し．開発者の能力を引き出し，自律的な組織みよって進められる．基本的には小規模開発に向いている．

これらの開発とチケット駆動開発との関係は以下の図のようになる．

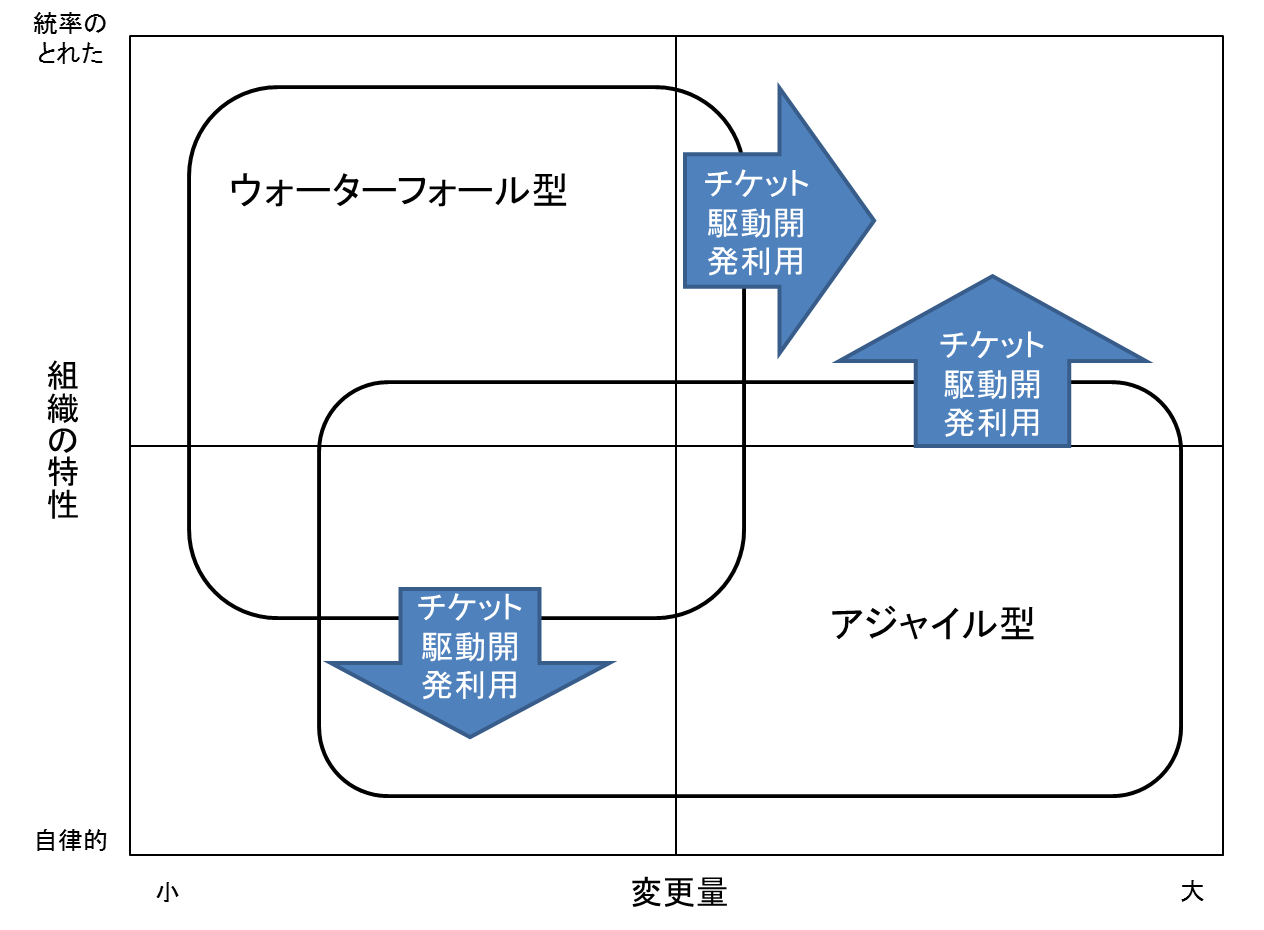


図 2‑5開発プロセスとチケット駆動開発の関係

ウォーターフォール型の開発では変更が多いと計画の変更が間に合わなくなり管理が困難になってしまう．このような状況においてチケット駆動開発を利用することで，変更管理をデジタル化し，効率化することが出来る．そしてリアルタイムなコミュニケーションと可視化によって自律的な組織運営を可能にすることが出来る．

　アジャイル型の開発は規模が大きくなるとアナログ型の管理に限界が出てくる．タスクボードに貼ることが出来る物理的なスペースにも限界があるほか，規模が大きくなるほど不具合が多くなってしまい，リリース後に不具合が生じた際にリリース済みのコード修正と，開発中のコードのそれぞれに修正を実施する対応が難しくなってしまう．そのためタスクカードをチケットに置き換えたチケット駆動開発を利用することで，タスクボードの限界を取り払い，ワークフローを利用してコード修正が的確に行われることを確認できるようになる．

　これらのようにチケット駆動開発を従来の開発に導入することで様々な問題を解決でき，今までの開発プロセスを洗練することが出来るのである．

* + 1. チケット駆動開発のメリット

チケットを利用することで，メンバーが気付いたこと，やること，やらなければならないこと，今まで潜在してきたタスクを洗い出すことが出来，可視化することが出来メンバー全体が把握できるようになる．また，いつでもだれでもチケットを参照でき，メンバー間のコミュニケーションにも利用できるため，誰がどのような状況なのか，何をやるのかを理解でき，作業の効率化を図ることが出来，プロジェクト効率を上げることが出来る．

* + 1. チケット駆動開発のデメリット

開発が大きくなるとチケットの数が膨大な量になりチケットのステータスを更新したり，終了させたりする作業に手間がかかる．また，チケットの量が増えすぎて，開発が追い付かなくなってしまうことがある為，チケットを含めた全体の開発を取り仕切る専門的なプロジェクトマネージャのような立場の人間が必要になってしまう．割り振りに偏りが生じてしまい個人が膨大な量のチケットを抱えてしまう恐れもある．

参考文献

[1] 小川明彦, 酒井誠. チケット駆動開発. 翔泳社, 2012-8-23.

[2] Project Management Institute, Inc. プロジェクトマネジメント知識体系ガイド(第4版). PMI, 2009-12.

[3] Jonathan Rasmusson. アジャイルサムライ―達人開発者への道. オーム社, 2011-11-25.

[4] あきぴー. 第3回 脱Excel! Redmineでアジャイル管理, 自分戦略＠IT研究所, <http://jibun.atmarkit.co.jp/lskill01/rensai/tool10/03/01.html>, 2013-10-10.

第3章

GitHubについて

# GitHubについて

* 1. 本章の構成

　本章では本研究でチケットを調査するにあたり，利用するバージョン管理システムについて，それを利用することが出来るGitHubについての基本知識，GitHubのAPIの解説，GitHubにおいてチケットと同様の機能を持つIssueについて，またそれらの機能について記述していく．

3.2　バージョン管理システムについて

バージョン管理システムは，ファイルの履歴を管理するシステムであり，修正や追加など作業によって生成されたファイルについての複数の履歴を記録し，後から古い履歴の取り出しや，差分の参照が可能になるシステムである．これらのファイルの更新履歴をリポジトリと呼び，自分が作業したファイルの更新をリポジトリに反映させることをコミットすると言う．またバージョン管理システムソフトウェアによっては，ファイルのDELETEや移動の履歴を確認する機能や，特定の利用者がファイルの管理する権限を獲得するロック機能，複数の変更を統合するマージ機能がある．

開発プロジェクトにおいて複数人で同一のファイルを編集する必要があるとき，バージョン管理システムの機能が役立つのである．バージョン管理システムを利用すると，更新者や変更点，変更日時が確認できるため，誰がいつどこの編集を行ったのかすぐに理解することが出来，混乱や時間の無駄遣いを避けることが出来，とても効率的に作業が進められるのである．

ソフトウェア開発においては，バージョン管理システムは特に有効的である．ソースコードなど長い文を編集した際など変更点を容易に把握でき管理できるので，障害がいつ発生したのか，どの時点から問題となっていたのか，いつ修正されたのかなどを容易に調べることが出来，早期の問題解決が可能になる．

また，バージョン管理システムは，管理方法の違いにより3つに分類される．ファイル単位で個別バージョン管理を行う「個別バージョン管理システム」，リポジトリをサーバーで一元管理し，コミットなどの操作はメンバーが行う「集中型バージョン管理システム」，リポジトリをメンバーで管理でき，そのメンバー間でリポジトリの連携が可能である「分散型バージョン管理システム」である．

* 1. 分散型バージョン管理システム

3つの種類のバージョン管理システムが存在すると前述したが，ここでは本論文の調査環境であるGitHubに用いられるバージョン管理システムである，分散型バージョン管理システムについて述べる．

分散型のバージョン管理システムではリポジトリをサーバー上ではなく，クライアントであるメンバーのコンピュータ上にも作成し，コミットの参照や差分を所得する場合に手元にあるリポジトリにアクセスするため，常にネットワークに接続している必要がないのである．またローカルファイルにリポジトリが存在するため，高速に動作することが可能であり，変更点を送受信する仕組みがあるため，複数のリポジトリ間で連携することが出来る．この分散型バージョン管理システムを行えるのがGitである．以下に分散型バージョン管理システムを図で表す．

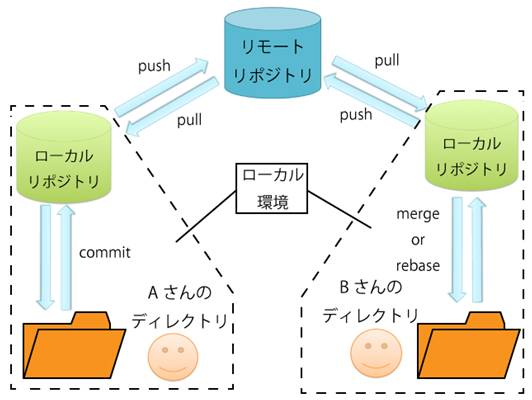


図 3‑1分散型バージョン管理システム

* 1. Gitについて

　分散型バージョン管理システムであるGitは2005年，Linuxカーネル開発現場での必要性から開発が始まった．それまで，Linuxカーネル開発のソース管理にはBitKeeperというバージョン管理システムが用いられていたのである．これはBitMover社製の商用のバージョン管理システムである．Bitkeeperは，先進の分散型バージョン管理システムで，カーネルプロジェクトが採用した当時，オープンソースの世界にはこれに匹敵する分散型バージョン管理システムの使用が不可欠であった．そのため，LinuxはGitを開発したのである．

Gitは分散型バージョン管理システムであるため，サーバーを必要としない．またユーザーそれぞれのコンピュータ上にリポジトリを持ち，それぞれが互いに連携しあうことができる．さらに，基本的なそれぞれのリポジトリにすべての履歴が保存されるため，差分やログの表示などを高速に行えるのである．

　リポジトリ間連携はネットワーク通信やメールを経由して行う．ほかにもリポジトリを共有リポジトリとして公開する仕組みや，ユーザー管理と組み合わせる方法があり，集中型バージョン管理システムのような利用形態もとれる．

　また，他のバージョン管理システムとのデータ交換も可能であり，既存の他のバージョン管理システムのリポジトリをGitリポジトリへ変更することや，中央リポジトリに他のバージョン管理システムのリポジトリを利用し，手元ではGitを利用する，といった形態をとることもできるのである．

オープンソースのバージョン管理システムとしては，CVSやSubversionが有名で，今でもこれらの集中型システムはよく使われている．しかし，近年になって，Linuxカーネル，X.org，Ruby on Rails，Perlといった有名なプロジェクトがGitに乗りかえて成功裡に使用しているのをみて，Gitを使用し始めとする分散型バージョン管理システムを使用するプロジェクトは飛躍的に増加してきている．

* + 1. Gitの特徴

Gitは主にファイル自身，ファイルの集合としてのツリー，そしてコミット情報という3つの情報を管理している．それぞれの情報はハッシュ値をもとに管理され，このハッシュ値はファイルが同一かどうかの判断にも用いられる．Gitはコミット情報を差分管理ではなく，ファイルそのままを保持しているという特徴を持っている．さらに時間的な変遷を管理する仕組みや，コミットをメールで受信する仕組みなどがある．また，ローカルコンピュータ上にリポジトリを持つため，場所や時間，あるいはネットワーク接続の状態に関わらずコミットすることが出来る．

* 1. GitHubとは

GitHubとは，GitHub.comにより運営されているGitホスティングサイトである．Gitリポジトリを利用してプロジェクトやソースコードの管理を行うバージョン管理システムを提供しているサービスである．GitHubはGitホスティングサイトとしては最も多く利用されているサービスであり，170万を超える人が利用している．

またGitHubはソーシャルな機能が特徴で，プログラマー同士がコードの共有を行ったり，コードの公開をし合ったりしている．そして，GitHubではソースコードだけでなく，画像やドキュメントなど，どんなファイルでもアップロードすることが出来，管理することが出来る．

* + 1. GitHubの基本用語

　GitHubの基本用語について以下に記述する．

1. Gitリポジトリ

GitHubで提供するデータベースのようなものである．論文に記述してあるリポジトリはGitリポジトリのことである．リポジトリの管理ユーザーが公開か非公開かに選択することができる．公開の場合は，誰でも閲覧することができる．だが，非公開に選択すると特定のユーザーのみしか閲覧することはできない．

1. コミット（commit）

ファイルの変更履歴情報を閲覧したり，ファイルの変更を保存したりすることである．

1. 共有リポジトリ

共有リポジトリとは，チームメンバーで共有するリポジトリで，ソースコードのメインバージョンが保存されている．

1. ローカルリポジトリ

作業者のコンピュータ上にあるリポジトリである．

1. インデックス

ローカルリポジトリへ反映する変更を一時的に保存しておく場所である．インデックスの内容は，commitによりローカルリポジトリへ反映される．

1. アドオン（add）

ソフトウェアに追加される拡張機能のことである．

1. 作業ツリー

ローカルリポジトリ上にある現在の作業ファイルである．作業ツリーの変更点はaddによるインデックスに追加される．

1. ディレクトリ

フォルダのことである．ファイルを分類・整理するための保管場所である．

1. フォロー

特定のユーザーをフォローすることが出来る．

1. フォロワー

ユーザーをフォローしているユーザー

1. スター

「いいね！」のようなもの．つけられたスターはカウントされ，スターのカウントが多いとほかのユーザーからの注目をされていると認識できる．

1. リビジョン

ある期間内までの過去のプロジェクトデータやある程度まとまったプロジェクトデータを記録したものである．

1. ウィキ（Wiki）：

その場にページが表示され，ドキュメントやコードがかける場所である．

1. Oragnization

Oragnizationとは普段，個人的にGitHutを使用している人が仕事などでGitHubを仕事用に使用したい場合にアカウントをもう1つ作成するのではなく，同じアカウントで会社用に使用するアカウントとして使用するのがOragnizationである．

* + 1. GitHubの機能

GitHubはGitをサポートするためにさまざまな機能を備えている．以下には機能について記述する．

* フォーク（forking）：

1つのプロジェクトが複数に分岐していくことである．だれかのリポジトリをほかの人がコピーし改変していくことである．

* ライト（write）：

　フォークしたときにコピーしたオリジナルのリポジトリのデータに書き込みをすることである．

* プルリクエスト：

　フォークを行い，コピーしたリポジトリのデータにライトしたことをオリジナルのリポジトリのユーザーに通知をPOSTすることである．

* マージ（merge）:

プルリクエストした人がその人のリポジトリに対して行われた変更を自分のリポジトリにも取り入れることである．

* イシュー（Issue）:

Issueとは，1つのタスクを1つのIssueに割り当てて，データの監視や管理を行えるようにするための機能である．1つの機能変更や修正などに対して1つのIssueが割り当てられるため，Issueを見れば，そのタスクの変更や修正に関することがすべてわかるよう管理できるのである．また，イシューにタグやマイルストーンをつけることも可能である．タグ機能は初期の設定の場合では，「バグ」，「重複」，「強化」，「無効」，「質問」が設定されている．タグの種類は増やすことも可能である．また，Issueには，「Open」と「Close」機能があり，Issueを受信した人は受信したIssueを拝見したらOpenをクリックし，拝見し終わったら，Closeをクリックすることで，Issueを発行したユーザーに拝見し終わったことを通知することができる．

* ウォッチ機能：

　他の人のデータを見ることができる．他の人の進捗状況やプロジェクト内容を閲覧できる．

* グループ機能：

特定のアカウントユーザーで構成し，特定のユーザー同士でリポジトリを共有したりすることができる．グループ機能を公開の状態にしておけば，グループ以外のユーザーも閲覧はできるが，グループ機能を非公開の状態にするとグループで決められたユーザー以外は閲覧やリポジトリを操作することができないように設定できる．

* 検索機能

　検索機能は「検索ウィンドウ」に検索ワード（ユーザー名やプロジェクト名，コードなど）を入力するとそれに関連した情報を表示することができる．

* 1. APIについて

　API（Application Programming Interface，アプリケーション・プログラミング・インターフェース）とは，アプリケーションから利用できる，オペレーティングシステムやプログラミング言語で用意されたライブラリなどの機能の入り口となるものである．主に，ファイル制御，ウィンドウ制御，画像処理，文字制御などのための関数として提供されることが多いのである．

　つまり，アプリケーションをプログラムするにあたって，プログラムの手間を省くために，非常に簡潔にプログラムができるように設定されたインタフェースのことである．

* + 1. APIを公開する・公開されるメリット

　Web製作者にとっては，他社の膨大なデータベースや機能を無料で利用できるため，Webサイトの開発のコストを大幅に削減できるので，効率的に制作できる．このため，個人や小資本の会社でも，人気のWebサイトを低コストで効率的に作成されることが可能になったのである．

　また，API提供会社にとっては，自社のみでは考え付かないようなWebサービスや自社のみでは実現できないようなWebサービスを外部の人間が作成してくれるため，API利用サイトから自社のサイトへユーザーが流入してくるのを期待できる．さらに，自社のデータベースや機能を間接的に自社のサイトを利用してくれるユーザーが増加し，それが結果としてデータを集積するスピードが速まり，競争優位に立つことを期待できるのである．

* 1. GitHubのAPIの種類

　GitHubのAPIとはGitHubの機能を抽出する時に必要なコードである．GitHubのAPIは機能ごとに設定されている．IssueにはIssueのAPIがあり，マイルストーンには，マイルストーンのAPIがある．以下に機能ごとのAPIを記述する．

・Issue（イシュー）のAPIは以下に記述する．

表 3‑1 IssueのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/issues/{id} | 選択されたIssueを表示する． |
| GET | /repos/{userName}/{repoName}/issues | リポジトリのIssueの一覧を表示する． |
| GET | /issues | ユーザーのIssueの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/issues/{id} | 指定されたリポジトリのIssueを編集する． |
| GET | /repos/{userName}/{repoName}/issues | 指定されたリポジトリにIssueを作成する． |

・Issue Events（イシューイベント）のAPIは以下に記述する．

表 3‑2 Issue EventsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/issues/events/{id} | リポジトリの指定したIssueの変化を表示する． |
| GET | /repos/{userName}/{repoName}/issues/events | リポジトリのIssueの変化を表示する． |
| GET | /repos/{userName}/{repoName}/issues/{issueId}/events | リポジトリの指定したIssueの変化を表示する． |

・Issue Comments（イシューコメント）のAPIは以下に記述する．

表 3‑3 Issue CommentsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueのコメントを削除する． |
| GET | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueのコメントを表示する． |
| GET | /repos/{userName}/{repoName}/issues/{id}/comments | リポジトリのIssueの返信コメントを表示する． |
| POST | /repos/{userName}/{repoName}/issues/{id}/comments | リポジトリのIssueにコメントを作成する． |
| POST | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueにコメントを編集する． |

・Events（イベント）のAPIは以下に記述する．

表 3‑4 EventsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /users/{userName}/events | ユーザーが実行したイベントの一覧を表示する． |
| GET | /networks/{userName}/{repoName}/events | ネットワーク内でユーザーが実行したリポジトリの共有リポジトリの一覧を表示する． |
| GET | /orgs/{org}/events | 組織の共有イベントの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/events | ユーザーが実行したリポジトリのイベントを表示する． |
| GET | /events | 共有イベント名を表示する． |
| GET | /users/{userName}/events/orgs/{org} | 組織でユーザーが実行したイベントの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/issues/events | リポジトリでユーザーが実行したイベントの一覧を表示する． |
| GET | /users/{userName}/received\_events | ユーザーが受信したイベントの一覧を表示する． |
| GET | /users/{userName}/events/public | ユーザーが共有イベント名の一覧を表示する． |
| GET | /users/{userName}/received\_events/public | ユーザーが受信した共有イベントの一覧を表示する． |

・Label（ラベル）のAPIは以下に記述する．

表 3‑5 LabelのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/issues/{id}/labels | 指定したリポジトリのIssueから全てのラベルを削除する． |
| DELETE | /repos/{userName}/{repoName}/labels/{id} | 指定したリポジトリから指定ラベルを削除する． |
| DELETE | /repos/{userName}/{repoName}/issues/{id}/labels/{id} | 指定したリポジトリのIssueからラベルを削除する． |
| GET | /repos/{userName}/{repoName}/milestones/{id}/labels | すべてのマイルストーンを表示する． |
| GET | /repos/{userName}/{repoName}/labels | リポジトリ内のラベルがついたものの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/labels/{id} | リポジトリ内の選択されたラベルを表示する． |
| POST | /repos/{userName}/{repoName}/issues/{id}/labels | リポジトリ内のIssueにラベルを追加する． |
| POST | /repos/{userName}/{repoName}/issues/{id}/labels | リポジトリ内のラベルを更新する． |
| POST | /repos/{userName}/{repoName}/labels/{id} | リポジトリ内にラベルを作成する． |
| PUT | /repos/{userName}/{repoName}/issues/{id}/labels | Issueのすべてのラベルを交換する． |

・Gists（ギスト）のAPIは以下に記述する．

表 3‑6 GistsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /gists/{id}/star | ギストのスターを解除する． |
| DELETE | /gists/{id} | ギストのスターを削除する． |
| GET | /gists | 認証されたユーザーのギストを一覧表示するか，匿名で呼び出された場合はすべての共有ギストを表示する． |
| GET | /users/{userName}/gists | ユーザー一覧のギストを表示する． |
| GET | /gists/public | 共有ギストの一覧を表示する． |
| GET | /gists/{id}/star | 指定したギストにスターがついているかを表示する． |
| GET | /gists/starred | ユーザーがスターをつけたギストの一覧を表示する． |
| GET | /gists/{id} | 指定したギストの情報を表示する． |
| POST | /users/{userName}/gists | 新しいギストを作成する． |
| POST | /gists/{id}/fork | ギストをフォークする． |
| POST | /gists/{id} | ギストを編集する． |
| 更新 | /gists/{id}/star | ギストのスターを更新する． |

・Gists Comments（ギストコメント）のAPIは以下に記述する．

表 3‑7 Gists CommentsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /gists/comments/{id} | ギストコメントを削除する． |
| GET | /gists/{gistId}/comments | ギストコメントの一覧を表示する． |
| GET | /gists/comments/{id} | ギストコメントの返信を表示する． |
| POST | /gists/comments/{id} | ギストコメントを編集する |
| POST | /gists/{gistId}/comments | ギストコメントを作成する． |

・GitData（ギットデータ）のAPIは以下に記述する．

表 3‑8 GitDataのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/git/commits/{sha} | 使用ユーザーはGitリポジトリからコミットを表示する． |
| GET | /repos/{userName}/{repoName}/git/trees/{sha} | 使用ユーザーはGitリポジトリからツリー形式を表示する． |
| GET | /repos/{userName}/{repoName}/git/tags/{sha} | 使用ユーザーはGitリポジトリのタグを表示する． |
| GET | /repos/{userName}/{repoName}/git/refs/tags | 使用ユーザーのGitリポジトリのすべての証明書タグを表示する． |
| GET | /repos/{userName}/{repoName}/git/refs | 使用ユーザーのGitリポジトリのすべての証明書を表示する． |
| GET | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーはGitリポジトリの証明書を表示する． |
| GET | /repos/{userName}/{repoName}/git/blobs/{sha} | 使用ユーザーはGitリポジトリのブロブを表示する． |
| GET | /repos/{userName}/{repoName}/git/trees/{sha}?recursive=1 | 使用ユーザーはGitリポジトリの1つ前のツリーを入手する． |
| POST | /repos/{userName}/{repoName}/git/blobs | 使用ユーザーのGitリポジトリにブロブを作成する． |
| POST | /repos/{userName}/{repoName}/git/trees | 使用ユーザーのGitリポジトリにツリーを作成する． |
| POST | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーのGitリポジトリ内の証明書を更新する． |
| POST | /repos/{userName}/{repoName}/git/tags | 使用ユーザーのGitリポジトリ内にタグオブジェクトを作成する． |
| POST | /repos/{userName}/{repoName}/git/commits | 使用ユーザーはGitリポジトリ内にコミットを作成する． |
| POST | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーのGitリポジトリの証明書を作成する． |

・Repo Download（ダウンロード・リポジトリ）のAPIは以下に記述する．

表 3‑9 Repo DownloadのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/downloads/{id} | リポジトリのダウンロードデータを削除する． |
| GET | /repos/{userName}/{repoName}/ downloads/{id} | リポジトリのダウンロードデータを表示する． |
| GET | /repos/{userName}/{repoName}/downloads | リポジトリのダウンロードデータの一覧を表示する． |
| POST | /repos/{userName}/{repoName}/downloads | リポジトリから新しくダウンロードする． |

・Repo Deploy Key（デプロイ・リポジトリ）のAPIは以下に記述する．

表 3‑10 Repo Deploy KeyのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を削除する． |
| GET | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を表示する． |
| GET | /repos/{userName}/{repoName}/keys | リポジトリの暗号鍵の一覧を表示する． |
| POST | /repos/{userName}/{repoName}/keys | リポジトリの暗号鍵を作成する． |
| POST | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を編集する． |

・Repo Fork（フォーク・リポジトリ）のAPIは以下に記述する．

表 3‑11 Repo ForkのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/forks | フォークデータの一覧を表示する． |
| POST | /repos/{userName}/{repoName}/forks | フォークを作成する． |

・Repo Hook（フック・リポジトリ）のAPIは以下に記述する．

表 3‑12 Repo HookのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/hooks/{id} | フックを削除する． |
| GET | /repos/{userName}/{repoName}/hooks | フックの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/hooks/{id} | 指定したフックを表示する． |
| POST | /repos/{userName}/{repoName}/hooks | フックを作成する． |
| POST | /repos/{userName}/{repoName}/hooks/{id} | フックを編集する． |
| POST | /repos/{userName}/{repoName}/hooks/{id}/test | フックのテスト（検証）を行う． |

・PullRequest（プルリクエスト）のAPIは以下に記述する．

表 3‑13 PullRequestのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/pulls | リポジトリ内のプルリクエストの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/pulls/{id}/commits | リポジトリ内のコミットへのプルリクエストの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/pulls/{id}/files | リポジトリからファイルへのプルリクエストの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/pulls/{id} | リポジトリから指定したプルリクエストを表示する． |
| GET | /repos/{userName}/{repoName}/pulls/{id}/merge | リポジトリからマージへのプルリクエストを表示する． |
| POST | /repos/{userName}/{repoName}/pulls/{id} | プルリクエストを更新する． |
| POST | /repos/{userName}/{repoName}/pulls | プルリクエストを作成する． |
| 更新 | /repos/{userName}/{repoName}/pulls/{id}/merge | マージをプルリクエストする． |

・Pull Request Review Comment（プルリクエスト・レビューコメント）のAPIは以下に記述する．

表 3‑14 Pull Request Review CommentのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/pulls/comments/{id} | プルリクエストのコメントを削除する． |
| GET | /repos/{userName}/{repoName}/pulls/{id}/comments | プルリクエストへのコメントの一覧をする． |
| GET | /repos/{userName}/{repoName}/pulls/{id} | プルリクエストへの指定されたコメントを表示する． |
| POST | /repos/{userName}/{repoName}/pulls/comments/{id} | コメントを編集する． |
| POST | /repos/{userName}/{repoName}/pulls/{id}/comments | コメントを作成する． |

・Milestone（マイルストーン）のAPIは以下に記述する．

表 3‑15 MilestoneのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/milestones/{id} | マイルストーンを削除する． |
| GET | /repos/{userName}/{repoName}/milestones | マイルストーンの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/milestones/{id} | 指定したマイルストーンの表示する． |
| POST | /repos/{userName}/{repoName}/milestones | マイルストーンを作成する． |
| POST | /repos/{userName}/{repoName}/milestones/{id} | マイルストーンを更新する． |

・User（ユーザー）のAPIは以下に記述する．

表 3‑16 UserのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /user | 認証されたユーザーを表示する． |
| GET | /users/{userName} | 指定したユーザーを表示する． |
| POST | /user | 認証されたユーザーを更新する． |

・User Email（ユーザー・メール）のAPIは以下に記述する．

表 3‑17 User EmailのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /user/emails | 電子メールアドレスを削除する． |
| GET | /user/emails | ユーザーの電子メールアドレスの一覧を表示する． |
| POST | /user/emails | 電子メールアドレスを追加する． |

・User Follower（ユーザー・フォロワー）のAPIは以下に記述する．

表 3‑18 User FollowerのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /user/following/{userName} | フォローを削除する． |
| GET | /user/following | 認証しているユーザーのフォローしている一覧を表示する． |
| GET | /users/{userName}/followers | フォロワーの一覧を表示する． |
| GET | /user/followers | 認証されたユーザーのフォロワーの一覧を表示する． |
| GET | /users/{userName}/following | フォローしているユーザーの一覧を表示する． |
| GET | /user/following/{userName} | フォローしているかどうかを表示する． |
| POST | /user/following/{userName} | ユーザーをフォローする． |

・User Public Key（ユーザー暗証キー）のAPIは以下に記述する．

表 3‑19 User Public KeyのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /user/keys/{id} | 公開鍵を削除する． |
| GET | /user/keys | 公開鍵の一覧を表示する． |
| GET | /user/keys/{id} | 指定した公開鍵を表示する． |
| POST | /user/keys/{id} | 公開鍵を更新する． |
| POST | /user/keys | 公開鍵を作成する． |

・Repos（リポジトリ）のAPIは以下に記述する．

表 3‑20 ReposのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /repos/{userName}/{repoName}/branches | ブランチの一覧を表示する． |
| GET | /user/repos | 認証済みユーザーのリポジトリの一覧を表示する． |
| GET | /users/{userName}/repos | 指定したユーザーの共有リポジトリの一覧を表示する． |
| GET | /repos/{userName}/{repoName} | ユーザーはリポジトリを表示する． |
| GET | /orgs/{org}/repos | 指定した組織の共有リポジトリの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/ contributors | 貢献者の一覧を表示する． |
| GET | /repos/{userName}/{repoName}/languages | 言語の一覧を表示する． |
| GET | /repos/{userName}/{repoName}/teams | チームの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/tags | タグの一覧を表示する． |
| POST | /user/repos | 認証されたユーザーのためにリポジトリを作成する． |
| POST | /repos/{userName}/{repoName} | ユーザーのリポジトリを編集する． |
| POST | /orgs/{org}/repos | 組織のリポジトリを作成する． |

・Repo Commit（リポジトリ・コミット）のAPIは以下に記述する．

表 3‑21 Repo CommitのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/comments/{id} | コミットコメントを削除する． |
| GET | /repos/{userName}/{repoName}/commits | コミットの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/commits/{sha} | 指定したコミットを表示する． |
| GET | /repos/{userName}/{repoName}/comments | コミットへのコメントの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/commits/{sha}/comments | 指定したコミットへのコメントを表示する． |
| GET | /repos/{userName}/{repoName}/comments/{id} | 指定したコメントを表示する． |
| GET | /repos/{userName}/{repoName}/compare/{base}...{head} | 2つのコミットを比較する． |
| POST | /repos/{userName}/{repoName}/comments/{id} | コミットのコメントを更新する． |
| POST | /repos/{userName}/{repoName}/commits/{sha}/comments | コミットのコメントを作成する． |

・Repos Watching（鑑賞用リポジトリ）のAPIは以下に記述する．

表 3‑22 Repos WatchingのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /user/watched/{userName}/{repoName} | リポジトリのウォッチをやめる． |
| GET | /user/watched | ユーザーによってウォッチされているリポジトリの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/watchers | ユーザーがウォッチしているリポジトリの一覧を表示する． |
| GET | /users/{userName}/watched | ユーザーがウォッチしたリポジトリの一覧を表示する． |
| GET | /user/watched/{userName}/{repoName} | 自分のリポジトリウォッチしているユーザーの一覧を表示する． |
| PUT | /user/watched/{userName}/{repoName} | リポジトリをウォッチする． |

・Repo Collaborator（共有リポジトリ）のAPIは以下に記述する．

表 3‑23 Repo CollaboratorのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /repos/{userName}/{repoName}/collaborators/{user} | 共同リポジトリを削除する． |
| GET | /repos/{userName}/{repoName}/collaborators | 共同リポジトリの一覧を表示する． |
| GET | /repos/{userName}/{repoName}/collaborators/{user} | 共同リポジトリの協力者を表示する． |
| PUT | /repos/{userName}/{repoName}/ collaborators/{user} | 共同リポジトリのユーザーを追加する． |

・Organization Team（組織チーム）のAPIは以下に記述する．

表 3‑24 Organization TeamのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /teams/{id}/members/{userName} | チームメンバーをDELETEする． |
| DELETE | /teams/{id}/repos/{userName}/{repoName} | チームからリポジトリをDELETEする． |
| DELETE | /teams/{id} | チームをDELETEする． |
| GET | /teams/{id}/members/{userName} | チームメンバーを表示する． |
| GET | /teams/{id}/members | チームのメンバーの一覧を表示する． |
| GET | /teams/{id} | チーム詳細を表示する． |
| GET | /orgs/{org}/teams | 組織内のチームの一覧を表示する． |
| GET | /teams/{id}/repos/{userName}/{repoName} | チーム内のリポジトリを表示する． |
| GET | /teams/{id}/repos | チームのリポジトリの一覧を表示する． |
| POST | /orgs/{org}/teams | チームを作成する． |
| POST | /teams/{id} | チームを編集する． |
| PUT | /teams/{id}/members/{userName} | チームメンバーを追加する． |
| PUT | /teams/{id}/repos/{userName}/{repoName} | チームリポジトリを追加する． |

・Organization（組織）のAPIは以下に記述する．

表 3‑25 OrganizationのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| GET | /user/orgs | 組織に認証されたユーザーの一覧を表示する． |
| GET | /orgs/{org} | 組織を表示する． |
| GET | /users/{userName}/orgs | 組織のユーザーの一覧を表示する． |
| POST | /orgs/{org} | 組織を編集する． |

・Organization Members（組織メンバー）のAPIは以下に記述する．

表 3‑26 Organization MembersのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| DELETE | /orgs/{org}/public\_members/{userName} | ユーザーのメンバーシップを非公開にする． |
| DELETE | /orgs/{org}/members/{userName} | メンバーを組織からDELETEする． |
| GET | /orgs/{org}/members/{userName} | メンバーを表示する． |
| GET | /orgs/{org}/members | 組織のメンバーであるすべてのユーザー一覧を表示する． |
| GET | /orgs/{org}/public\_members/{userName} | パブリックメンバーを表示する． |
| GET | /orgs/{org}/public\_members | パブリックメンバーの一覧を表示する． |
| PUT | /teams/{id}/members/{user} | メンバーを追加する． |
| PUT | /orgs/{org}/public\_members/{userName} | ユーザーの役割を変更する． |

参考文献

[1] 濱野純. 入門Git, 秀和システム, 2009-9-25.

[2] 片岡巌. WEB+DB PRESS Vol.69, 技術評論社, 2012-7-25.

[3] **田村啓.** APIとは, プログラミング実習室, <http://www.program-study.com/report/api>, 2013-10-10.

[4] APIS.io. APIS.IO, <http://apis.io/GitHub>, 2013-10-10.

[5] GitHub. GitHub Developer, [http://developer.GitHub.com/v3/issues/](http://developer.github.com/v3/issues/), 2013-10-10.

第4章

開発・調査

# 調査・開発

* 1. 本章の構成

　本章ではチケットの調査を行うための調査対象の記述． GitHub内からIssueのデータを抽出するツールを開発するための設計．実際に開発されたものの解説を記述する．

* 1. 調査対象

　ここでは本研究において調査対象となるデータの記述，およびそれらのデータを抽出する対象のプロジェクト名およびリポジトリ名を記述する．

### 調査対象データ

　本研究ではIssueの使われ方を調査する為に，Issue発行数，Issueの発行された日時および数，Issueの終了された日時および数を調査するため，Issueに含まれる以下のデータをGithubから抽出することとする．

表 4‑1　調査対象データ

|  |  |
| --- | --- |
| 対象データ | 解説 |
| state | Issueがオープンであるかクローズであるか表す． |
| created\_at | Issueがいつ発行されたのかを表す． |
| closed\_at | Issueがいつ終了されたのかを表す． |

* + 1. 調査対象プロジェクト

　本研究ではGitHubにおいてIssueがどのように使われ，プロジェクトが進められているのかを調査する為に，以下の36個のプロジェクトに着目し，それらのプロジェクトの日付ごとのIssueの発行数，終了数のデータを収集し，時系列ごとのIssueの数を調査する．

表 4‑2　調査対象プロジェクト

|  |  |
| --- | --- |
| ユーザ名 | リポジトリ名 |
| adobe | brackets |
| angular | angular.js |
| ariya | phantomjs |
| blueimp | jQuery-File-Upload |
| bower | bower |
| caolan | async |
| defunkt | jquery-pjax |
| discourse | discourse |
| gruntjs | grunt |
| hakimel | reveal.js |
| janl | mustache.js |
| jashkenas | backbone |
| jashkenas | coffee-script |
| jashkenas | underscore |
| joyent | node |
| LearnBoost | socket.io |
| less | less.js |
| madrobby | zepto |

|  |  |
| --- | --- |
| ユーザ名 | リポジトリ名 |
| maker | ratchet |
| mbostock | d3 |
| Modernizr | Modernizr |
| mojombo | jekyll |
| moment | moment |
| mozilla | pdf.js |
| mrdoob | three.js |
| plataformatec | devise |
| Prinzhorn | skrollr |
| resque | resque |
| rstacruz | nprogress |
| scottjehl | Respond |
| Shopify | dashing |
| thoughtbot | paperclip |
| twitter | typeahead.js |
| visionmedia | express |
| visionmedia | jade |
| xing | wysihtml5 |

* 1. 環境構築・調査

　本調査に必要である調査環境の構築を記述し，本調査で目的とするIssueのデータを抽出するプログラムの解説，抽出したデータからグラフをかくために必要なプログラムの解説，それらを自動化したプログラムの解説記述を行う．

* + 1. 調査環境構築

1. curlをインストールする．

　sudo apt-get install curl

1. Githubのログイン情報を省略する為に，github.passwdに書いておく．

echo 'ユーザ名:パスワード' > github.passwd

chmod 600 github.passwd

確認するために以下を行う．

cat github.passwd

Githubのユーザ名とパスワードを表示させることが出来る．

1. Issueのデータを所得する．

curl -s -u ユーザ名:パスワード "https://api.github.com/repos/ユーザ名/リポジトリ名/issues"

1. ユーザ名，パスワードを省略したものを利用するためには以下のように実行する．

curl -s -u $(cat github.passwd) "https://api.github.com/repos/ユーザ名 /リポジトリ名/issues"

1. API呼び出し結果の内容を確認するには以下のようにパイプ｜を利用する．

curl -s -u $(cat github.passwd) "https://api.github.com/repos/ユーザ名/リポジトリ名/issues" | grep title

1. PythonとHTTPアクセスのためのrequestsをインストールする。

sudo apt-get install python python-setuptools

sudo easy\_install requests

1. api.pyを作業ディレクトリにダウンロードする．

api.pyとは以下のようなプログラムで，データを所得し，ファイルに保存するプログラムである．

#!/usr/bin/python

# coding: UTF-8

import sys, json, requests

tmp = open('github.passwd').readline().rstrip('\n').split(':');

username = tmp[0]

password = tmp[1]

#print >> sys.stderr, username,password

url = sys.argv[1]

count = 0

while (url is not None):

#print url

r = requests.get(url, auth=(username, password))

print >> sys.stderr, r.headers['status'],

for item in r.json():

count = count + 1

print json.dumps(item)

if (r.links.has\_key('next')):

url = r.links['next']['url']

else:

url = None

print >> sys.stderr, count

1. JSON形式のデータを扱う為にjqのバイナリを作業ディレクトリにコピーし，jqを実行できるようにする為，以下を行う．

chmod +x jq

1. コマンドでグラフを描くためにGnuplotをインストールする．

sudo apt-get install gnuplot-x11

　Gunplotとは2次元または3次元のグラフを作成するためのコマンドラインアプリケーションソフトウェアのことであり，入力した数式等をもとにグラフを作成することが出来る．

* + 1. 調査方法

　GithubからIssue数の時間変化を調査する方法を記述する．

1. Githubから調査対象のプロジェクトのopenissueとclosedissueのデータを所得し.txt形式で書き出す．わかりやすくするためにファイル名は“ユーザ名‐リポジトリ名‐open(closed)issues.txt”とする．

python api.py "https://api.github.com/repos/ユーザ名/リポジトリ名/issues?per\_page=100" > ユーザ名-リポジトリ名-openissues.txt

python api.py "https://api.github.com/repos/ユーザ名/リポジトリ名/issues?per\_page=100&state=closed" > ユーザ名-リポジトリ名-closedissues.txt

1. ①のopen(closed)issues.txtからオープン・クローズの情報を抜き出す．

./jq '.created\_at' ユーザ名-リポジトリ名-openissues.txt | awk '{ printf("%s open\n", $0); }' > issues.tmp

./jq '.created\_at,.closed\_at' ユーザ名-リポジトリ名-closedissues.txt | awk '{ if (NR % 2 == 1) printf("%s open\n", $0); else printf("%s close\n", $0); }' >> issues.tmp

1. ②のissues.tmpのデータを時間で並び替えて，issues.txtを作成する．

sort issues.tmp > issues.txt

1. Issueの累積数を求めissues.csvを作成する．

awk 'BEGIN { openissues=0; closedissues=0; } $2=="open" { openissues++; } $2=="close" { closedissues++; } { printf("%s,%d,%d\n", $1, openissues, closedissues) }' issues.txt > issues.csv

1. gunplotでグラフを描く．（横軸を日時，縦軸をIssue数とする．）

set datafile separator ","

set xdata time

set timefmt "%Y-%m-%d %H:%M:%S"

set xl "date time"

set yl "number of issues"

set terminal png size 1680,1050

set border 15 lw 2

set tics font "Times New Roman,15"

set out "issuesCountChart.png"

plot 'issues.csv' using 1:2 with lines lw 2 title 'open issues', 'issues.csv' u 1:3 with lines lw 2 title 'closed issues'

※gunplotスクリプト解説

表 4‑3　gunplot解説

|  |  |
| --- | --- |
| 言語 | 解説 |
| set datafile separator "," | データをカンマで区切る． |
| set xdata time | X軸が時間であると指定する． |
| set timefmt "%Y-%m-%d %H:%M:%S" | 時間のフォーマットを設定する． |
| set xl "date time" | X軸をdate timeとする． |
| set yl "number of issues" | Y軸をnumber of issuesとする． |
| set terminal png size 1680,1050 | 結果をサイズ1680×1050の.pngで出力する． |
| set border 15 lw 2 | 外枠全体の太さをサイズ2で書く． |
| set tics font "Times New Roman,15" | 目盛の文字をTimes New Roman，サイズを15にする． |
| set out "issuesCountChart.png" | issuesCountChart.pngという名前で出力する． |
| plot 'issues.csv' using 1:2 with lines lw 2 title 'open issues', 'issues.csv' using 1:3 with lines lw 2 title 'closed issues' | Issues.csvの1列目と2列目を使ってopen issuesを描き，issues.csvの1列目と3列目を使ってcolosed issuesを描く． |

表 4‑4　コマンド解説

|  |  |
| --- | --- |
| コマンド | 解説 |
| curl |  |
| cat |  |
| echo | 因数に与えられた文字列を表示する． |
| > | ファイルに出力する． |
| >> | ファイルの末尾に追記する |
| chmod | ファイルやディレクトリのアクセス権を変更する．  600　→所有者にのみ読み込み書き込みの権限がある．  +x　 →実行権限を付加する． |
| -s | 行分割だけを行い、行の結合は行わない． |
| -u | 単語間のスペースを1つに、文の間はスペース2つに減らす． |
| | | コマンドの出力を次のコマンドの入力として渡す． |
| grep | 文字列を検索する． |
| awk |  |
| less |  |
| per\_page=数字 |  |

* + 1. 調査

上記の調査方法を利用してそれらを自動化し，2度のコマンド入力でデータの収集を行えるようにして，Issueの時間ごとの変化の調査を行った．

・所有者とリポジトリ名を指定してデータを取得する．

bash getIssues.sh ユーザ名 リポジトリ名

・グラフの描画

bash issuesCountChart.sh ユーザ名 リポジトリ名

上記の2つのコマンドでデータ所得，グラフの描画を行えるようにするために下記のプログラムを開発した．

1. getissues.sh

#!/bin/bash

python api.py "https://api.github.com/repos/${1}/${2}/issues?per\_page=100" > ${1}-${2}-openissues.txt

python api.py "https://api.github.com/repos/${1}/${2}/issues?per\_page=100&state=closed" > ${1}-${2}-closedissues.txt

このプログラムを実行することで，APIを利用してIssueのデータを取り出し，openissues.txtとclosedissues.txtを作成する．

1. issuesCountChart.sh

#!/bin/bash

./jq '.created\_at' ${1}-${2}-openissues.txt | awk '{ printf("%s open\n", $0); }' > ${1}-${2}-issues.tmp

./jq '.created\_at,.closed\_at' ${1}-${2}-closedissues.txt | awk '{ if (NR % 2 == 1) printf("%s open\n", $0); else printf("%s close\n", $0); }' >> ${1}-${2}-issues.tmp

sort ${1}-${2}-issues.tmp > ${1}-${2}-issues.txt

awk 'BEGIN { openissues=0; closedissues=0; } $2=="open" { openissues++; } $2=="close" { closedissues++; } { printf("%s,%d,%d\n", $1, openissues, closedissues) }' ${1}-${2}-issues.txt > ${1}-${2}-issues.csv

rm -f issues.csv

cp ${1}-${2}-issues.csv issues.csv

gnuplot issuesCountChart.pl

rm -f ${1}-${2}-issuesCountChart.png

mv issuesCountChart.png ${1}-${2}-issuesCountChart.png

1. issuesCountChart.pl

set datafile separator ","

set xdata time

set timefmt "%Y-%m-%d %H:%M:%S"

set xl "date time"

set yl "number of issues"

set terminal png size 1680,1050

set border 15 lw 2

set tics font "Times New Roman,15"

set out "issuesCountChart.png"

plot 'issues.csv' using 1:2 with lines lw 2 title 'open issues', 'issues.csv' u 1:3 with lines lw 2 title 'closed issues'

* + 1. 調査プログラムの実行

上記のプログラムを作業ディレクトリに入れ，Ubuntuの端末で

bash getIssues.sh ユーザ名 リポジトリ名

を実行する．

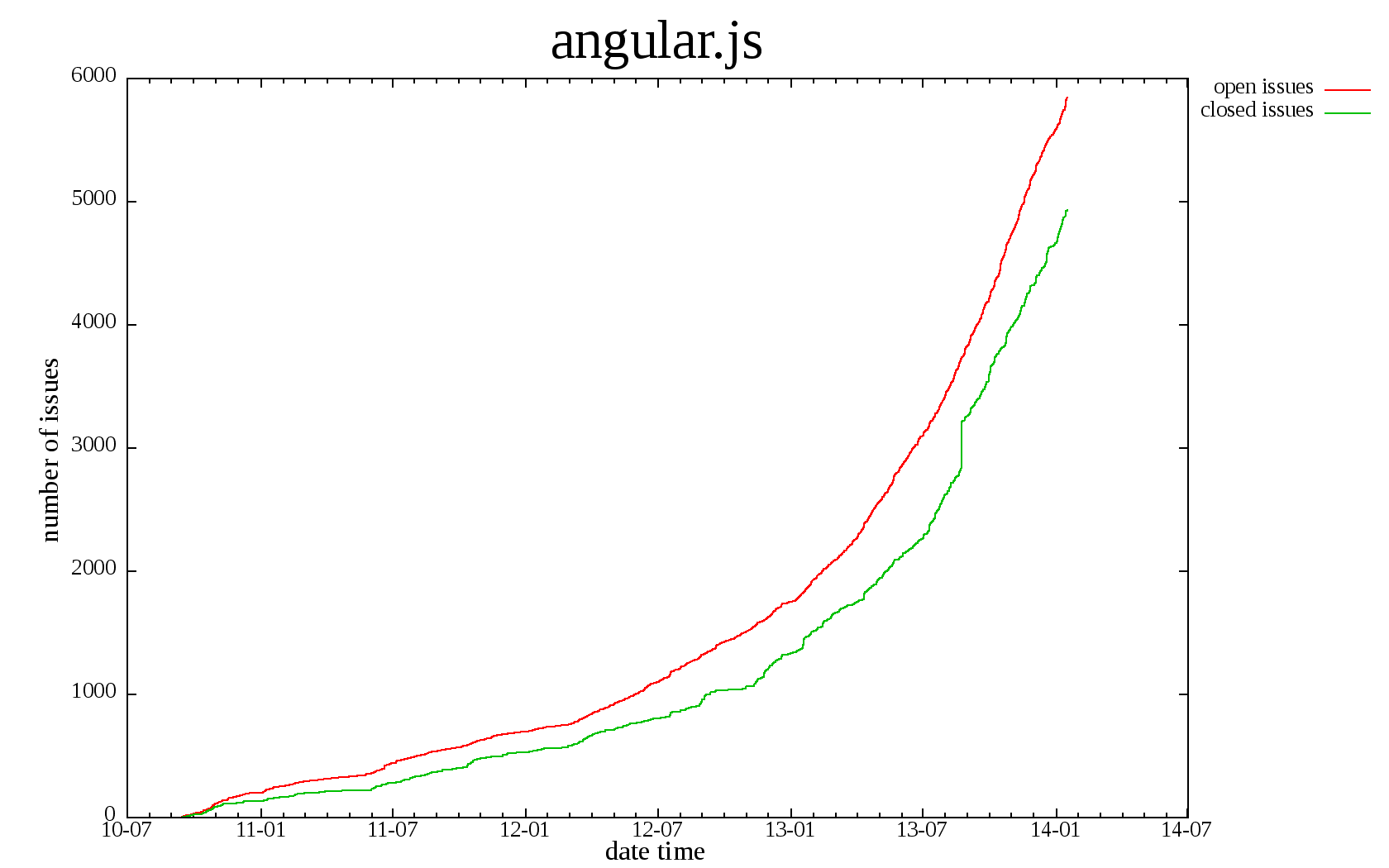


Issueのデータを取り出し，getissues.txt，closedissues.txtが作られる．

グラフを描くために

bash issuesCountChart.sh ユーザ名 リポジトリ名

を実行し，グラフを作成する．



以上の方法を用いてGithubのIssueのデータを引き出しIssue数の時間変化のグラフを描き，書かれたグラフを比べ，調査した．

第5章

調査結果・考察

# 調査結果考察

* 1. 本章の構成

　本章では前章で記述した調査方法を用いて，Github内のプロジェクトからIssueの時間変化の調査を行った結果を記述し，その結果に対する考察を記述していく．

* 1. 調査結果

　実際に調査を行い得られたグラフと解説を以下に記述していく．

* + 1. 調査結果グラフ

　調査結果のグラフを以下に示す．

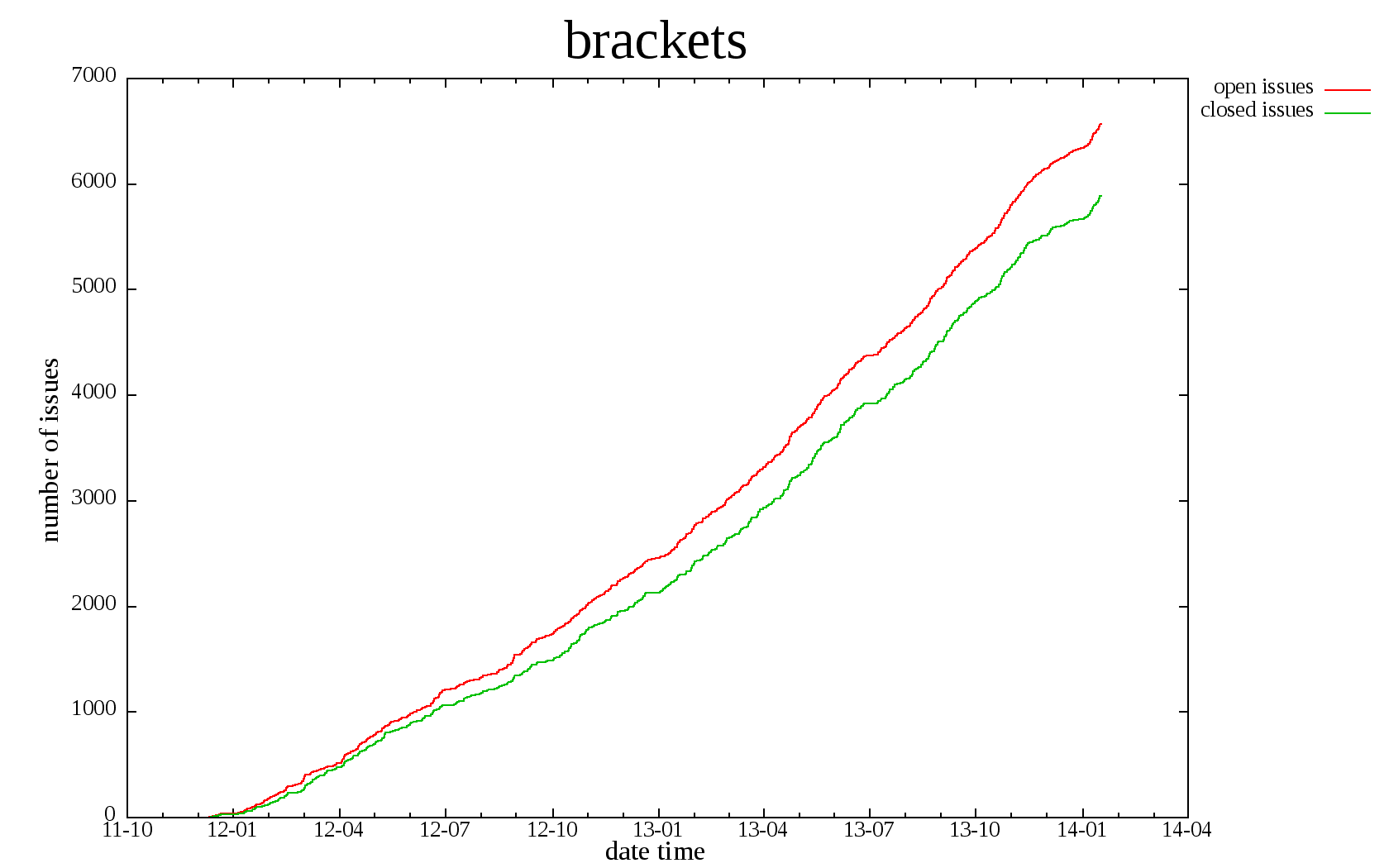


図 5‑1　Issueの時間変化グラフ/brackets

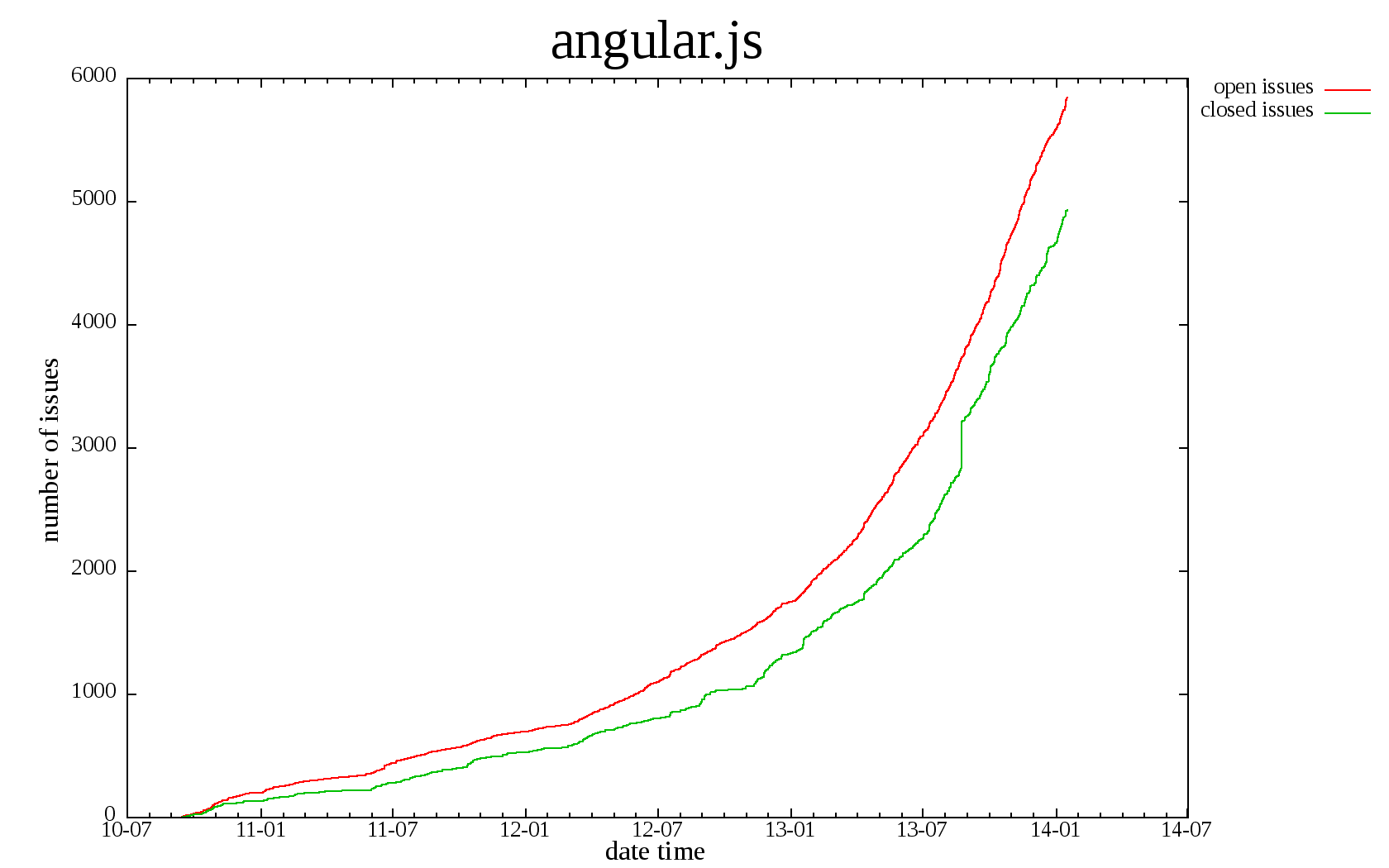


図 5‑2 　Issueの時間変化グラフ/angular.js

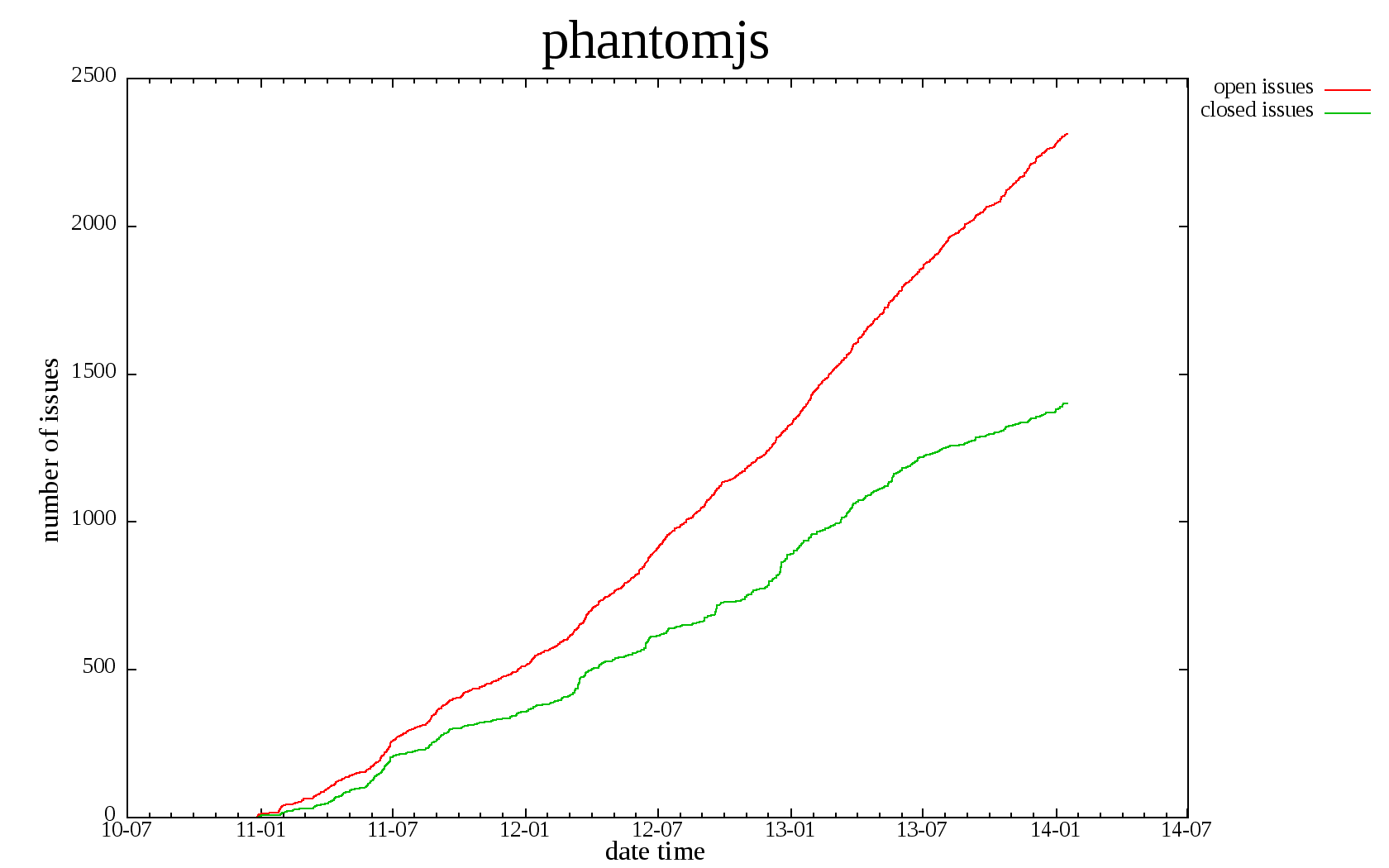
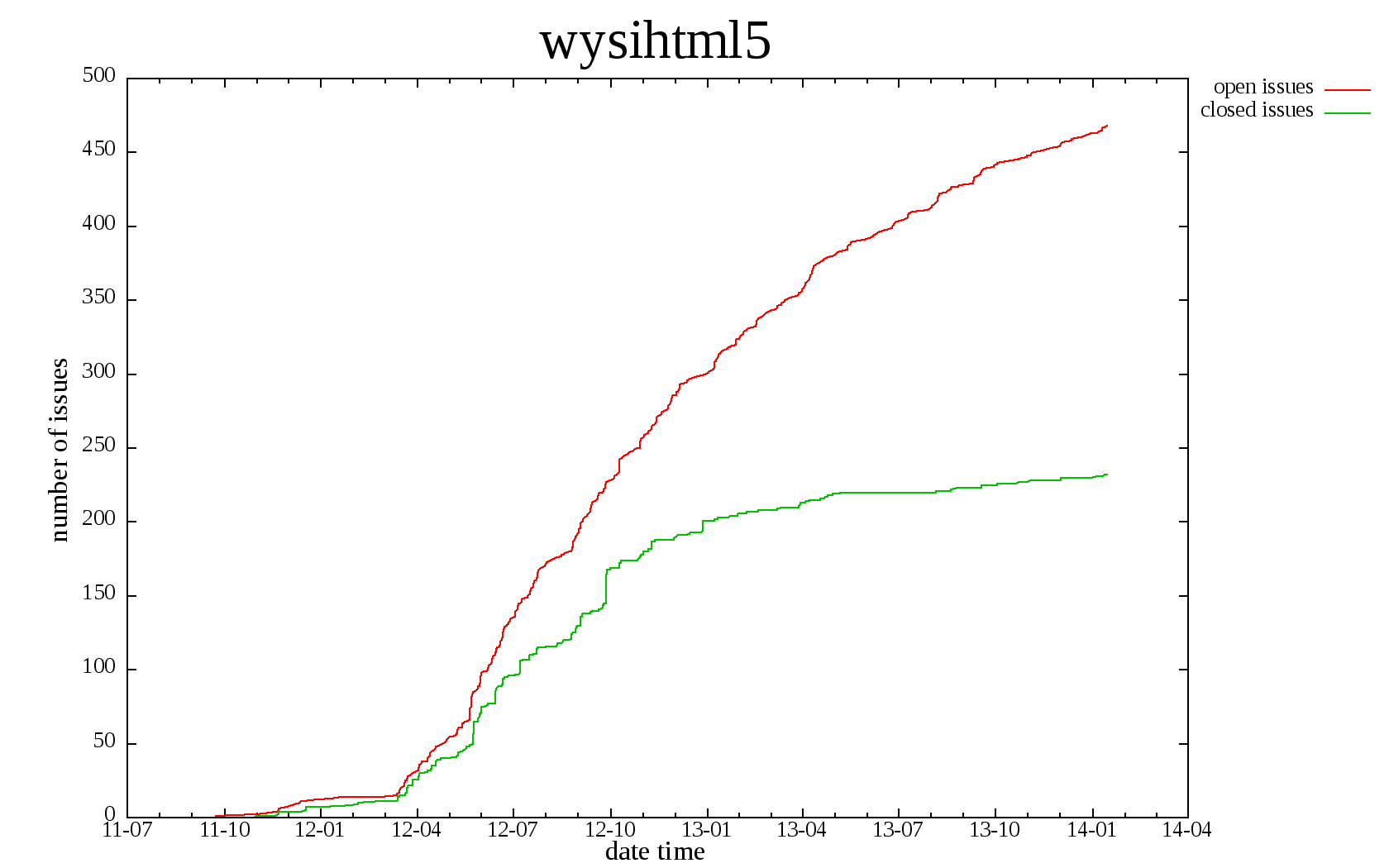
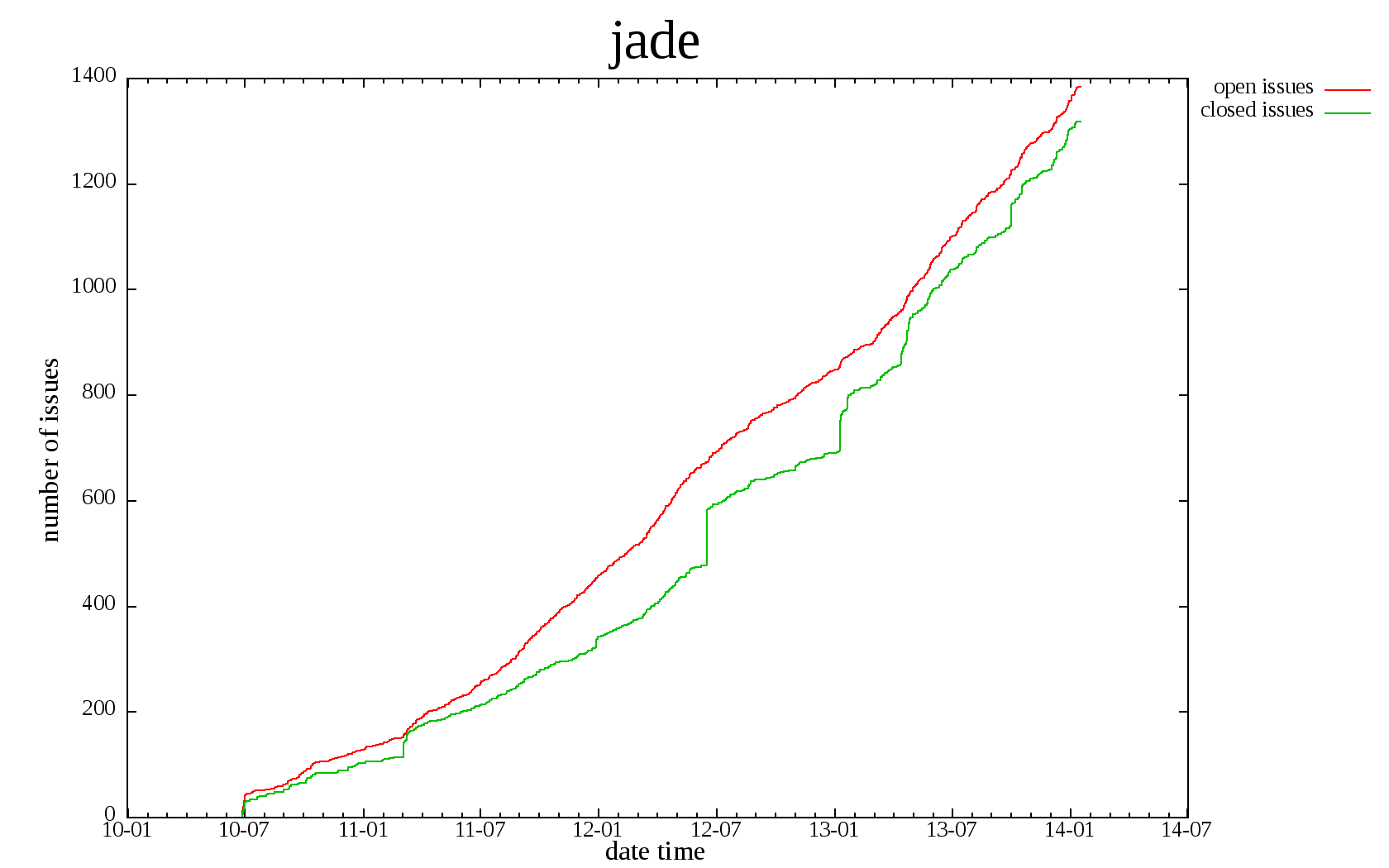
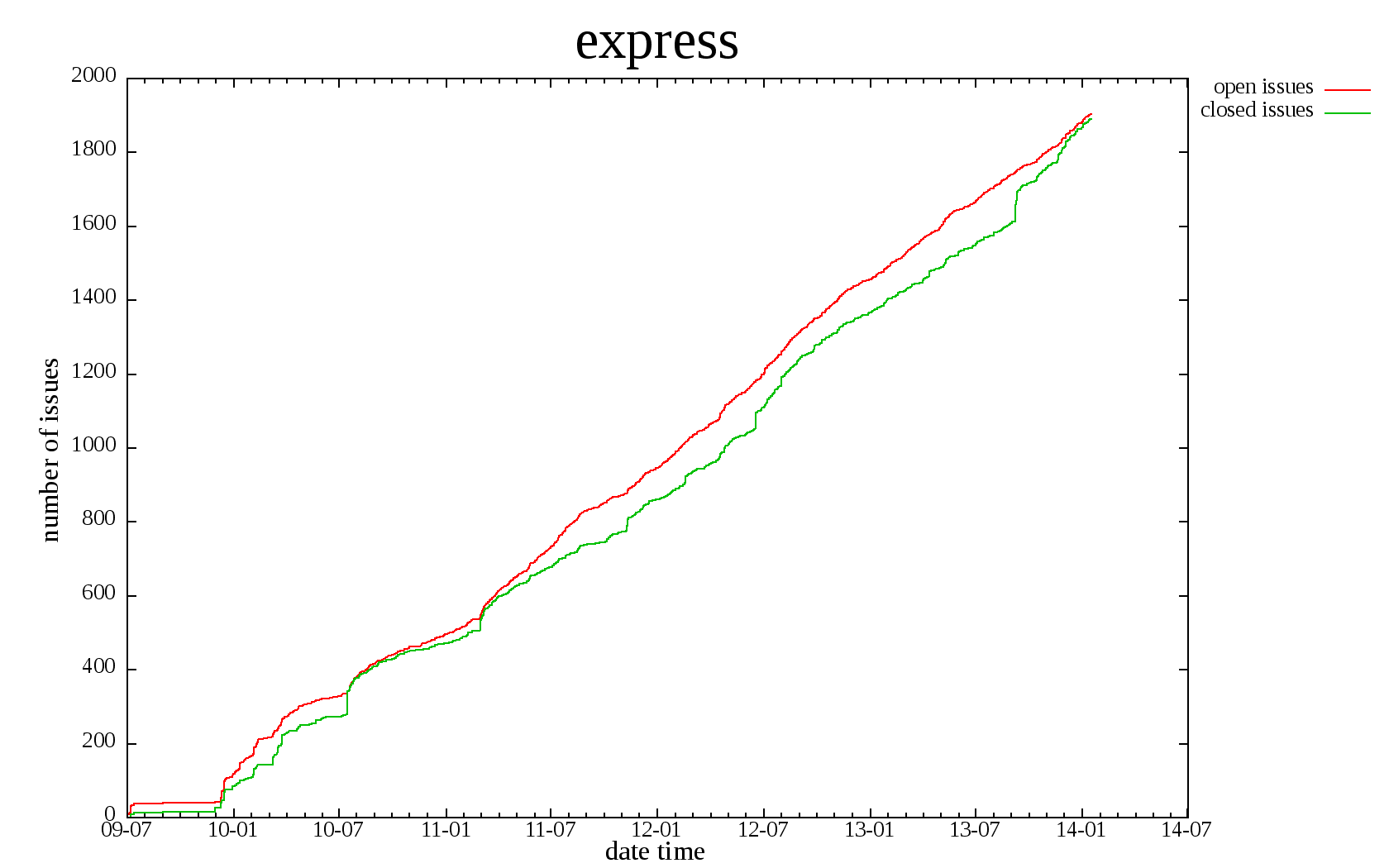
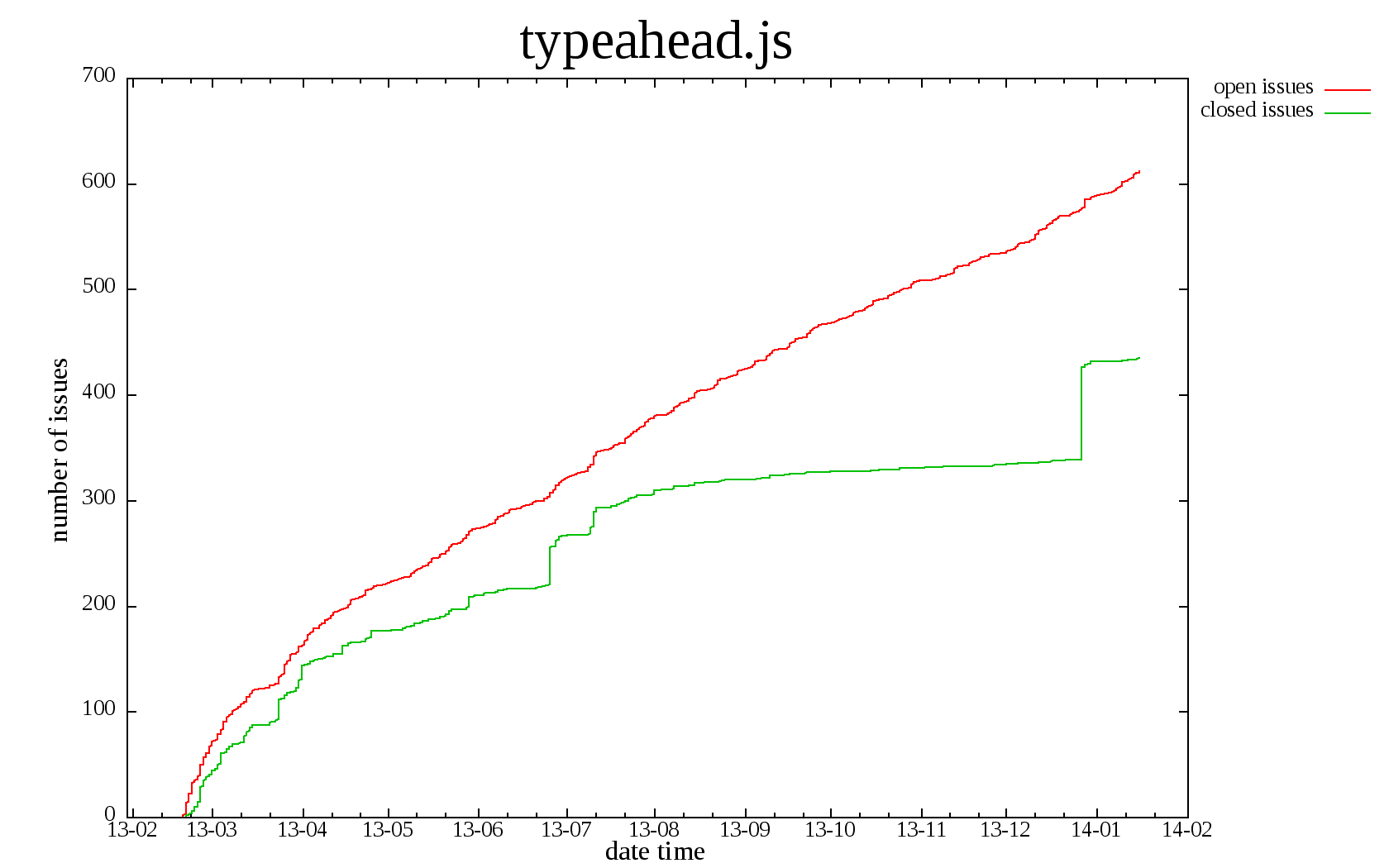
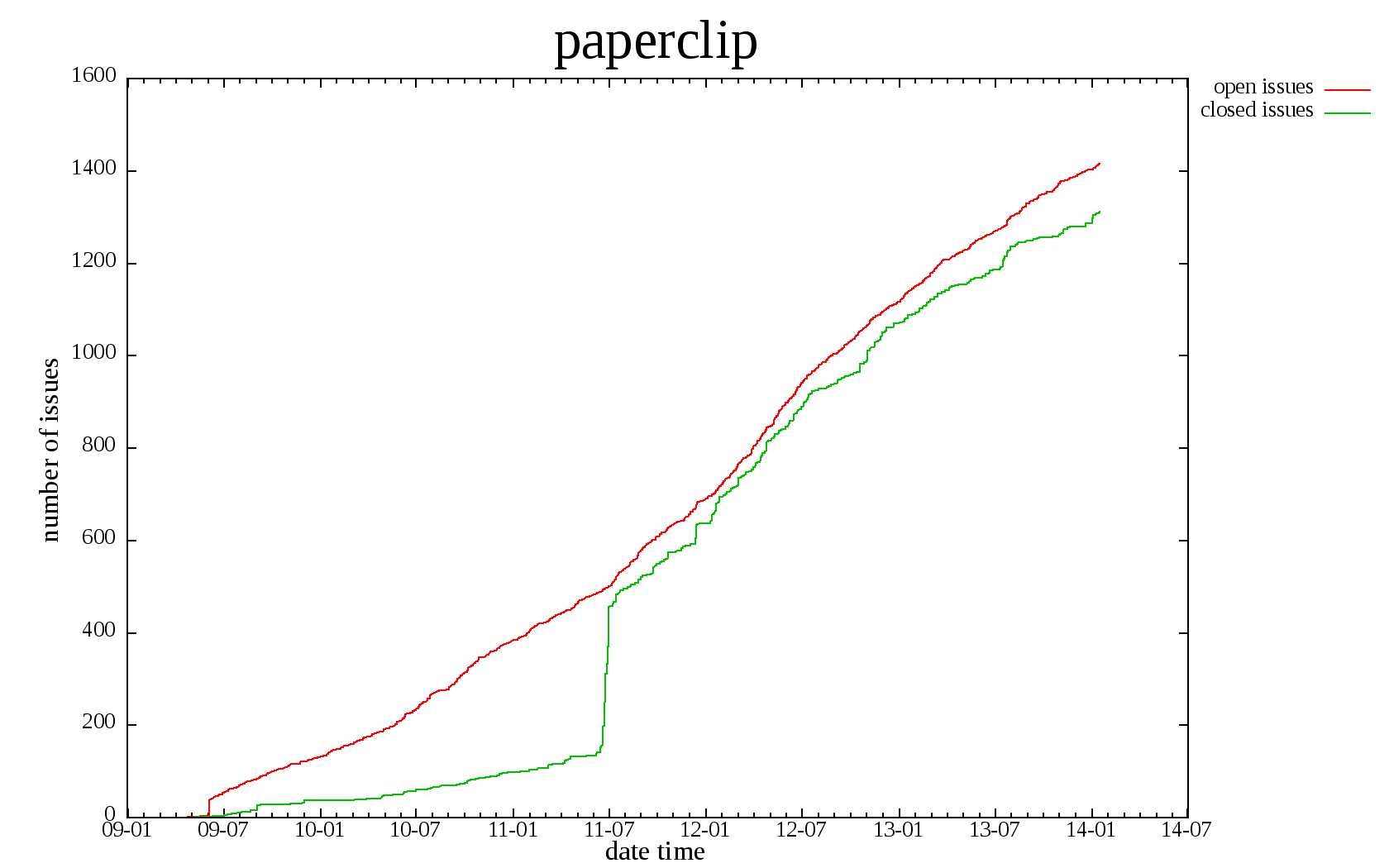
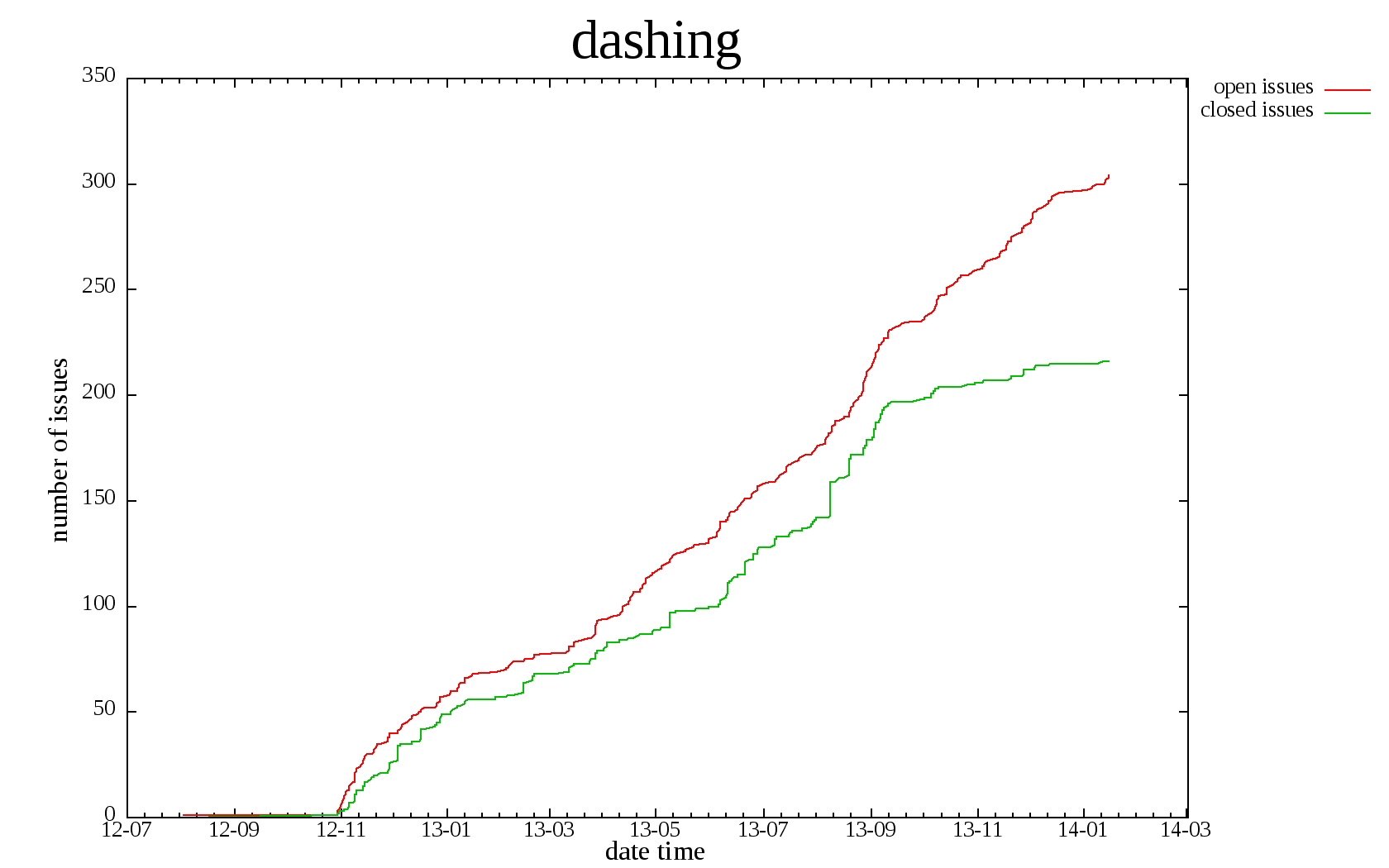
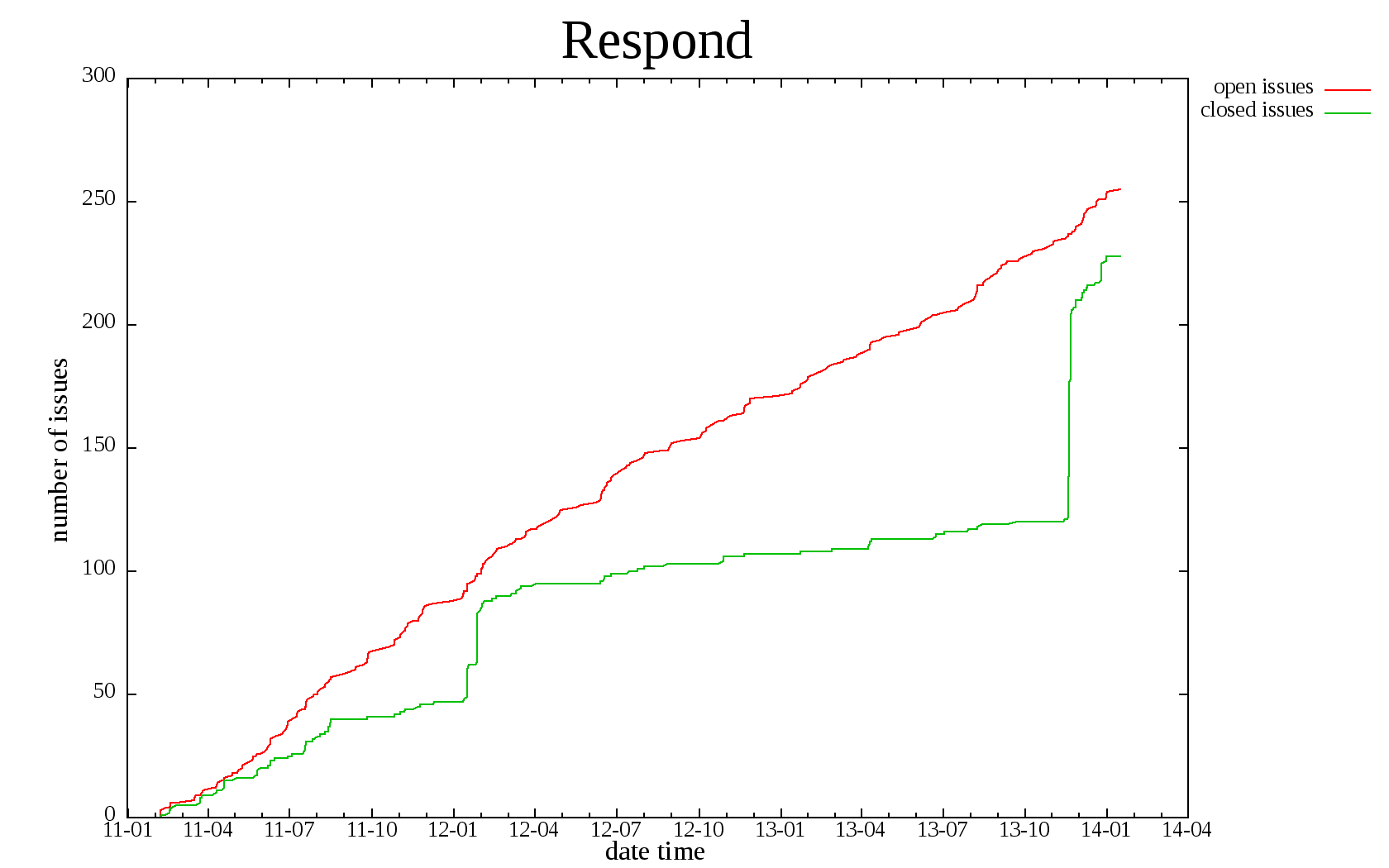
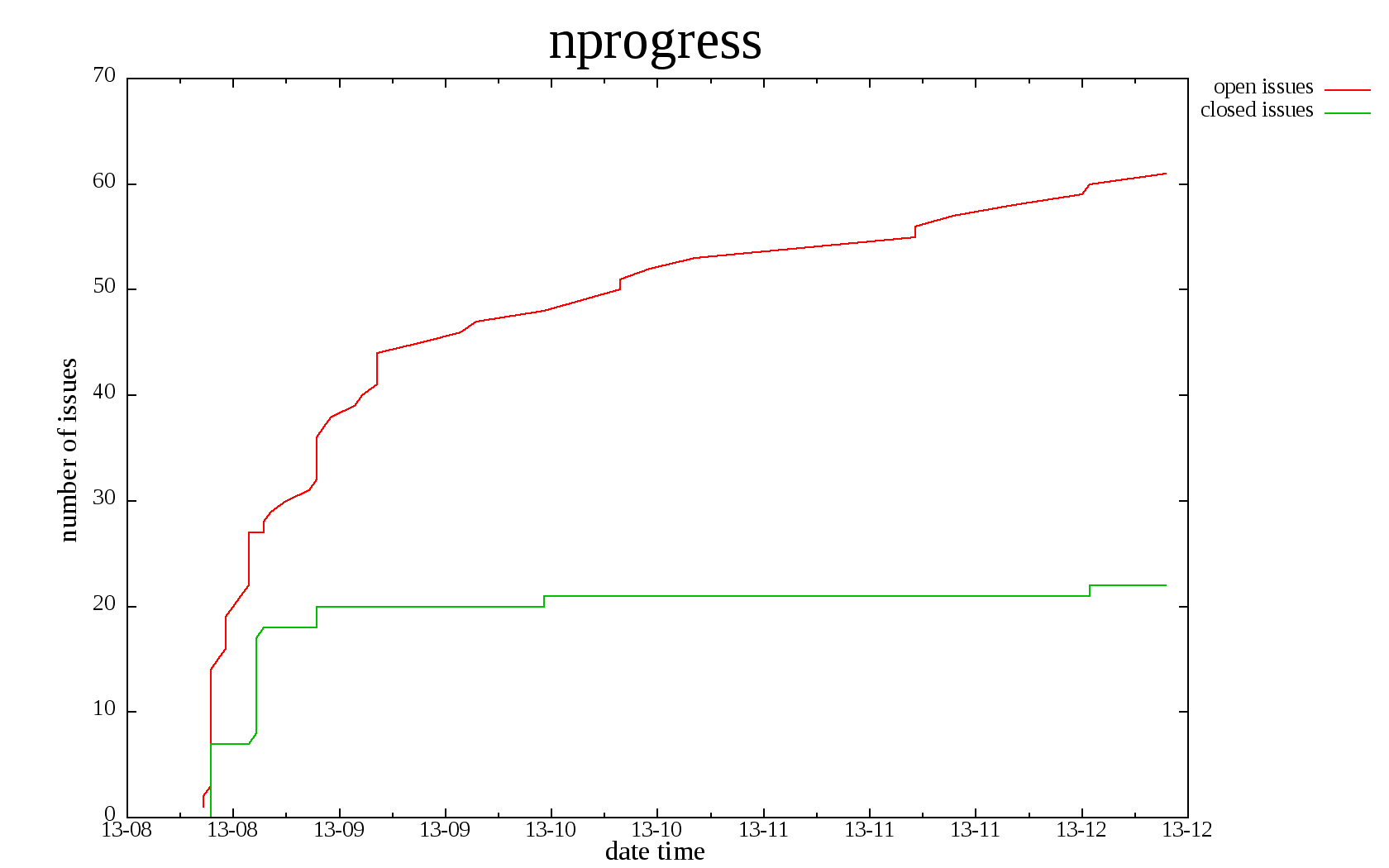
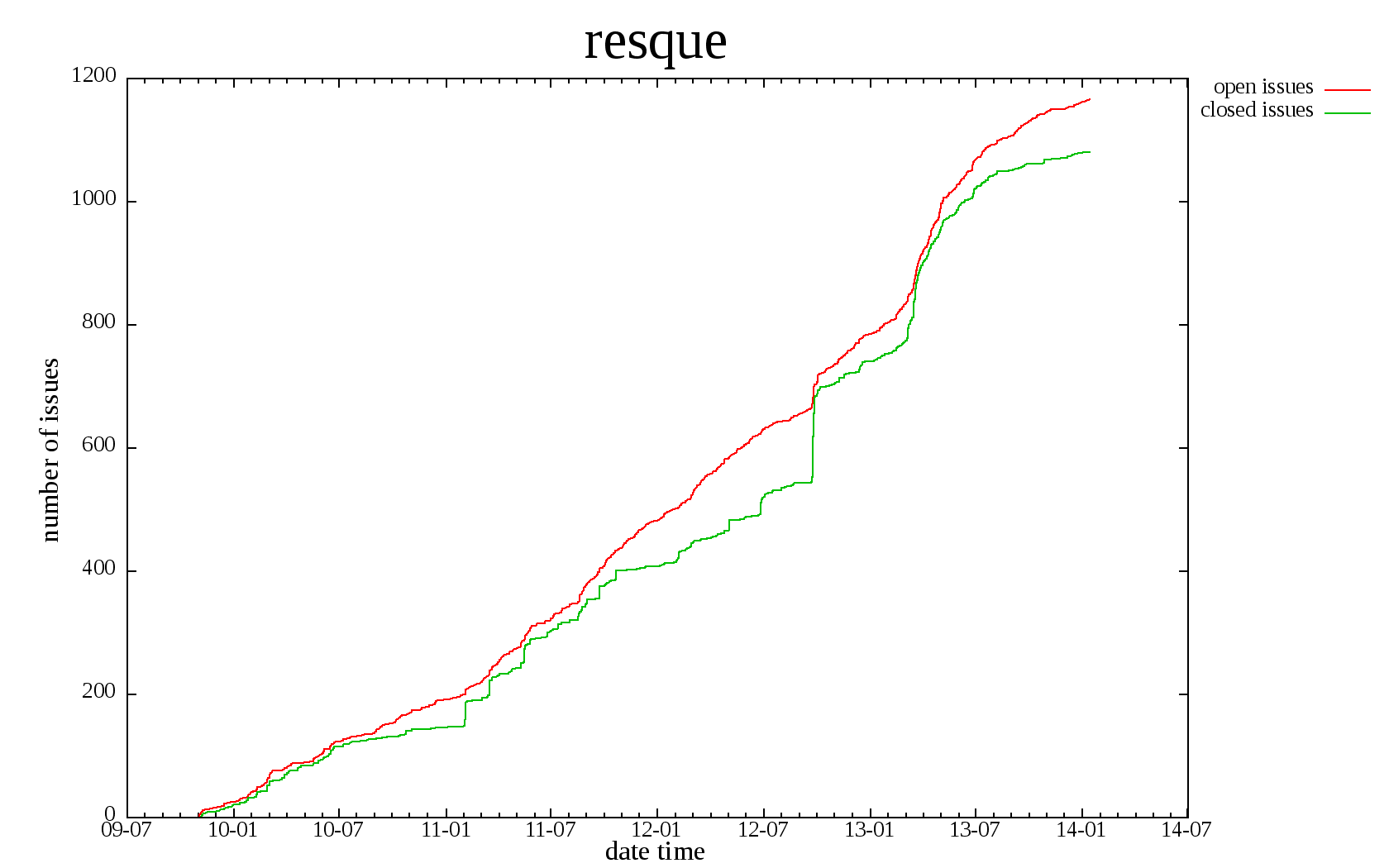
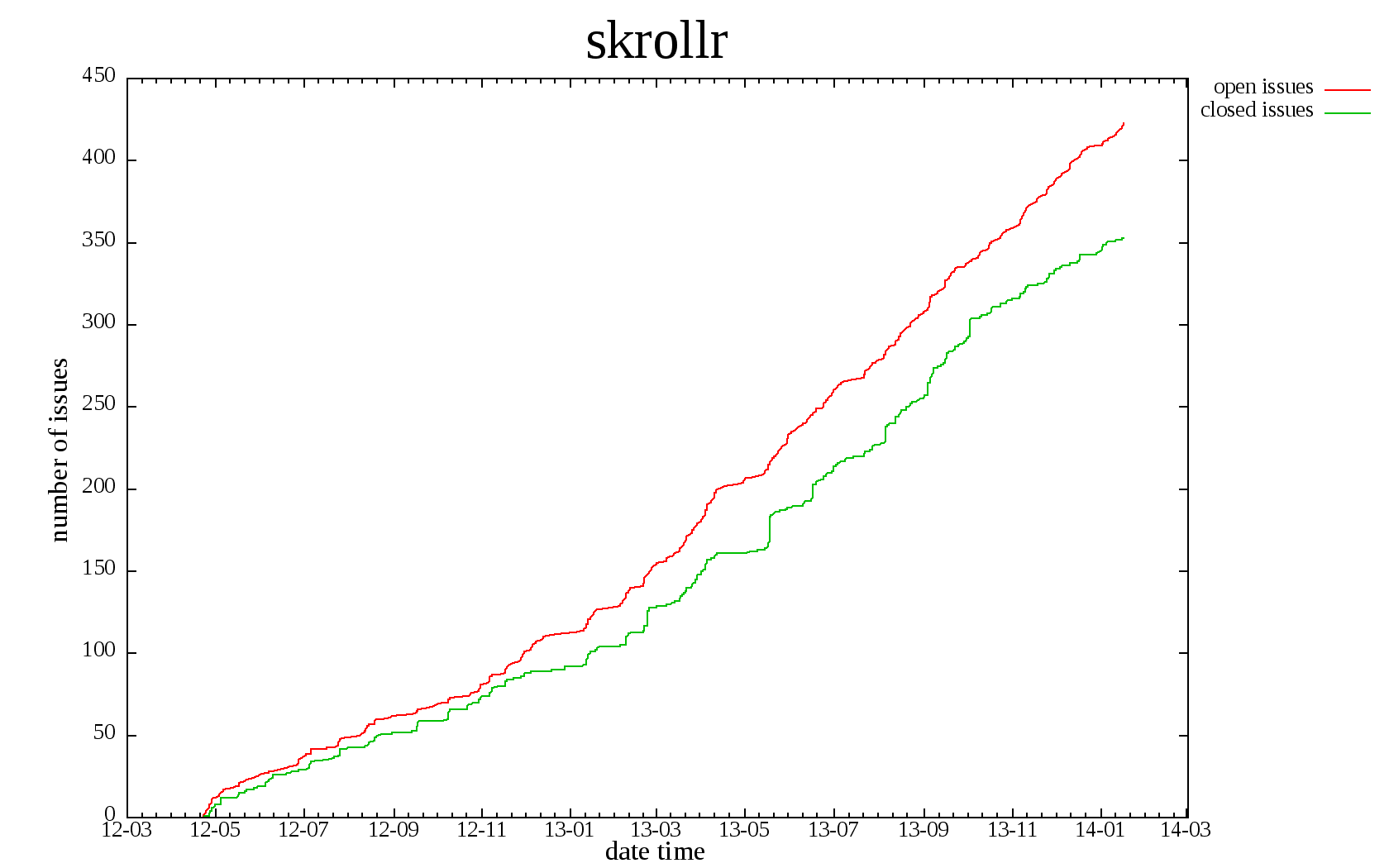
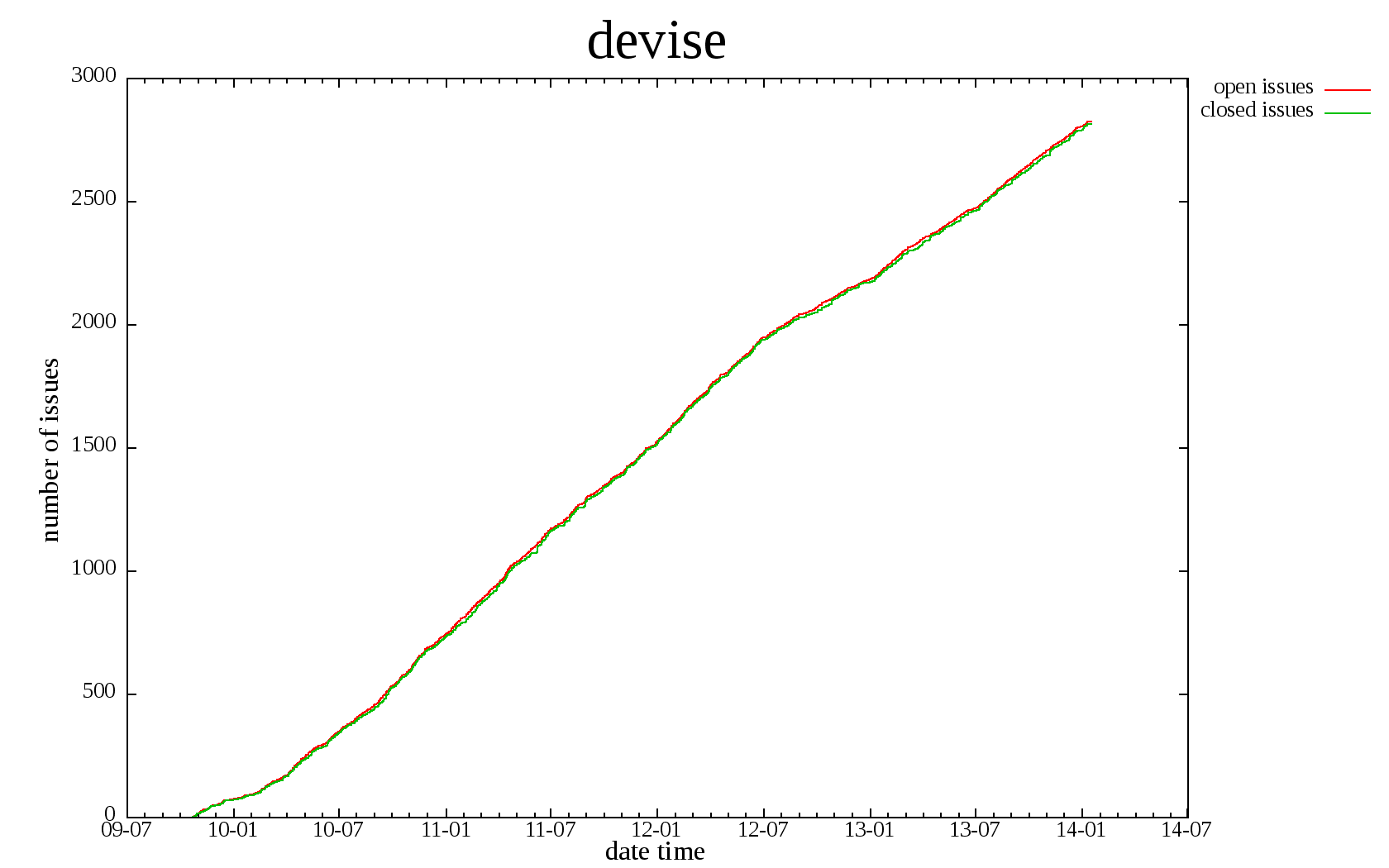
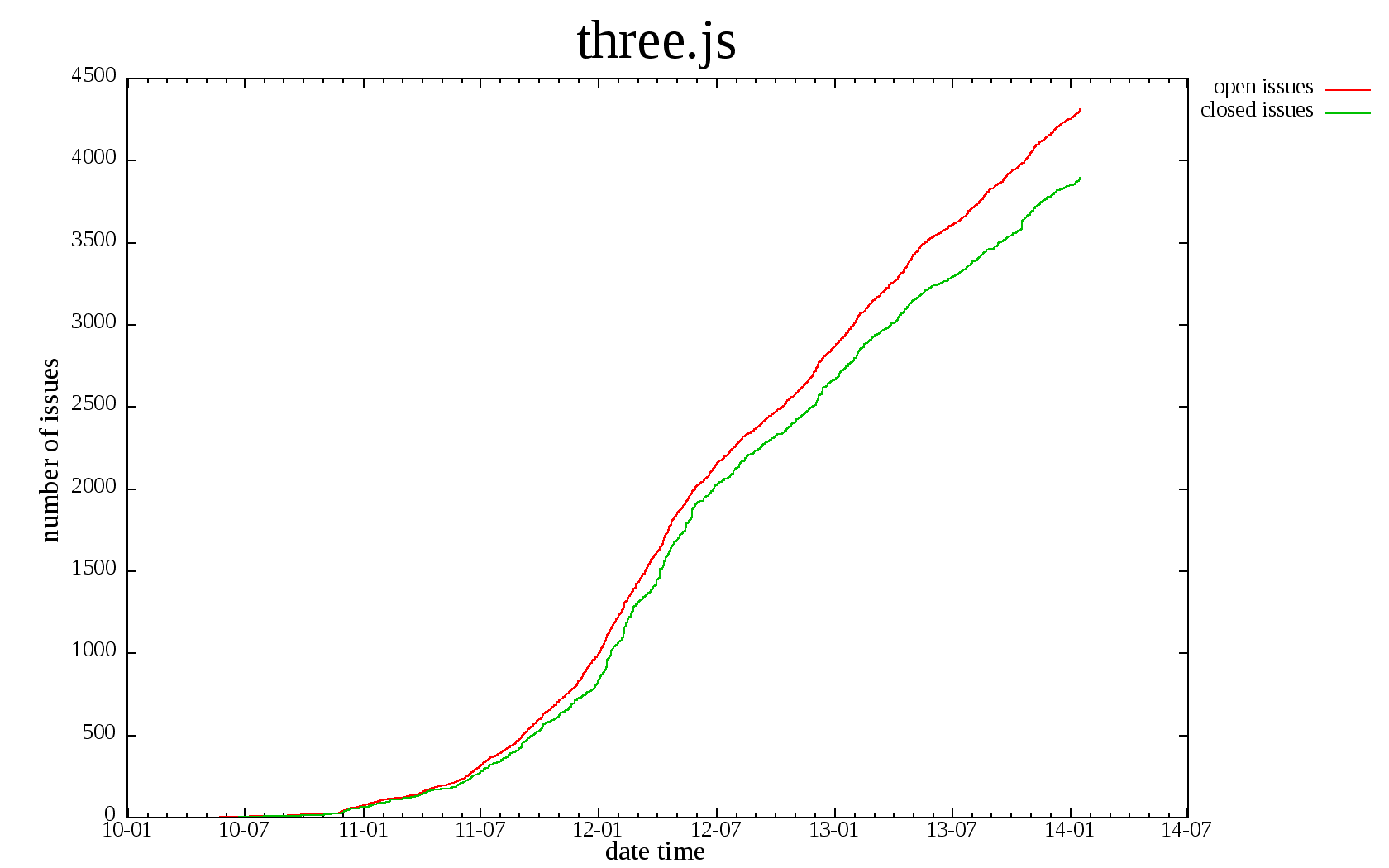
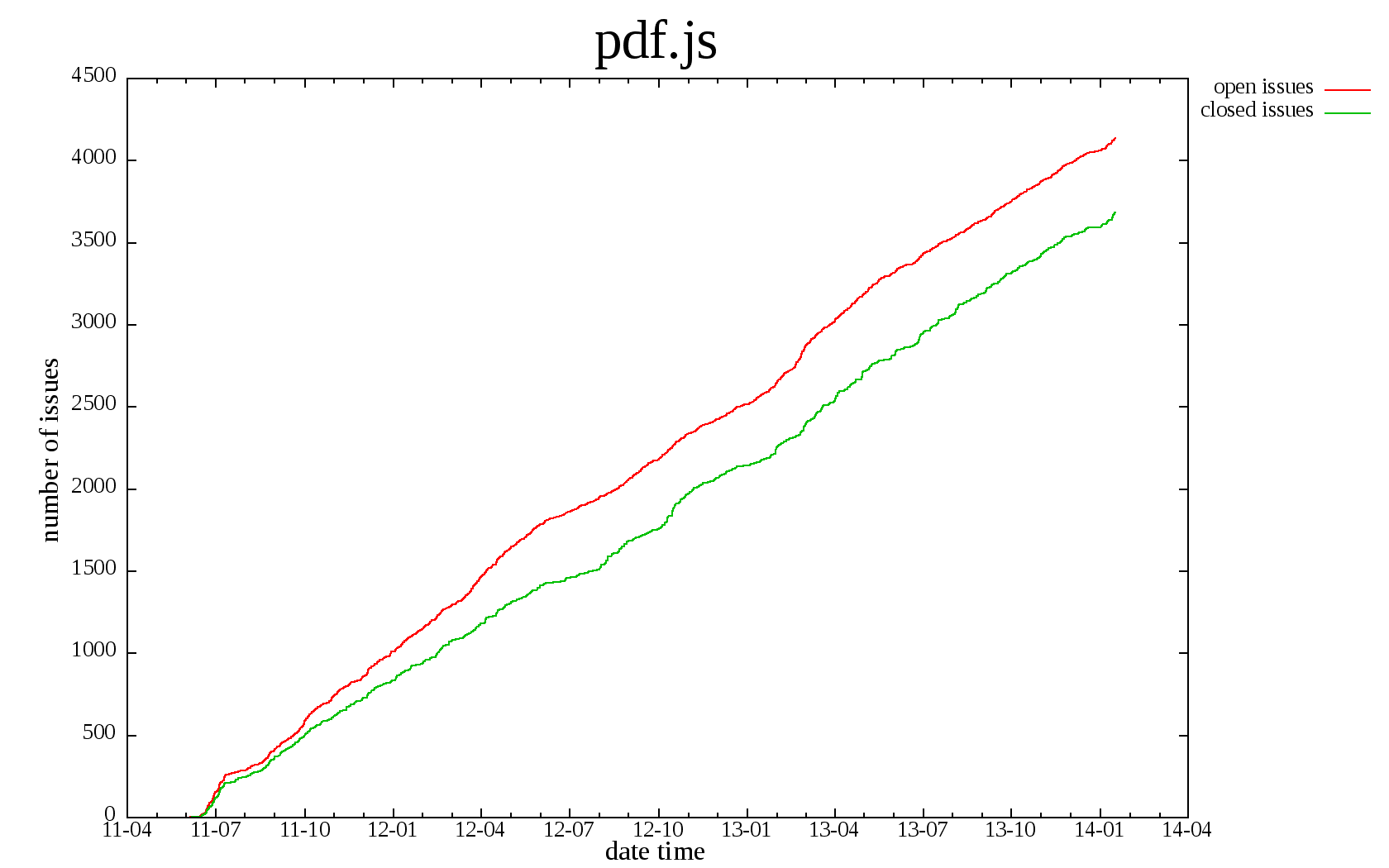
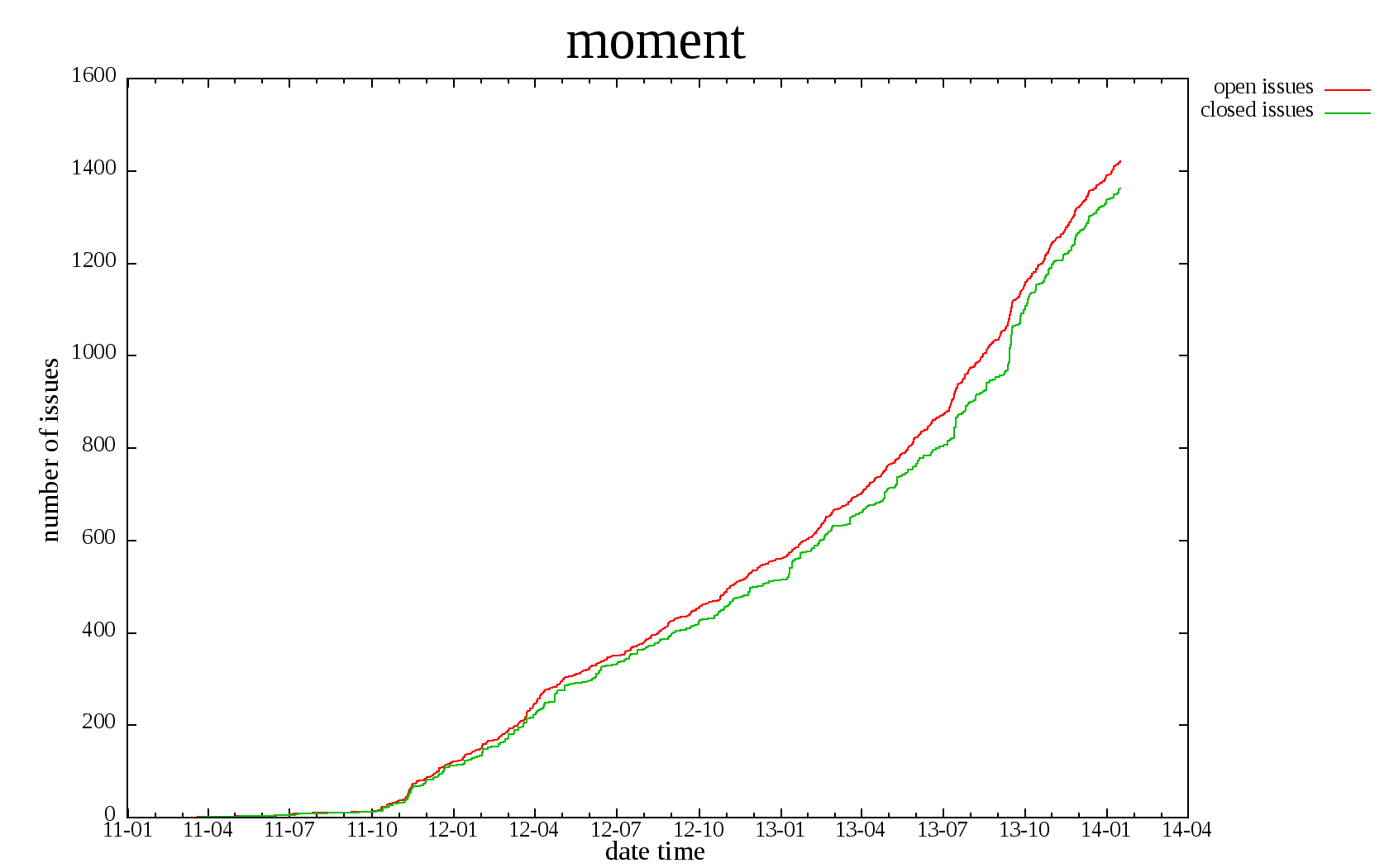
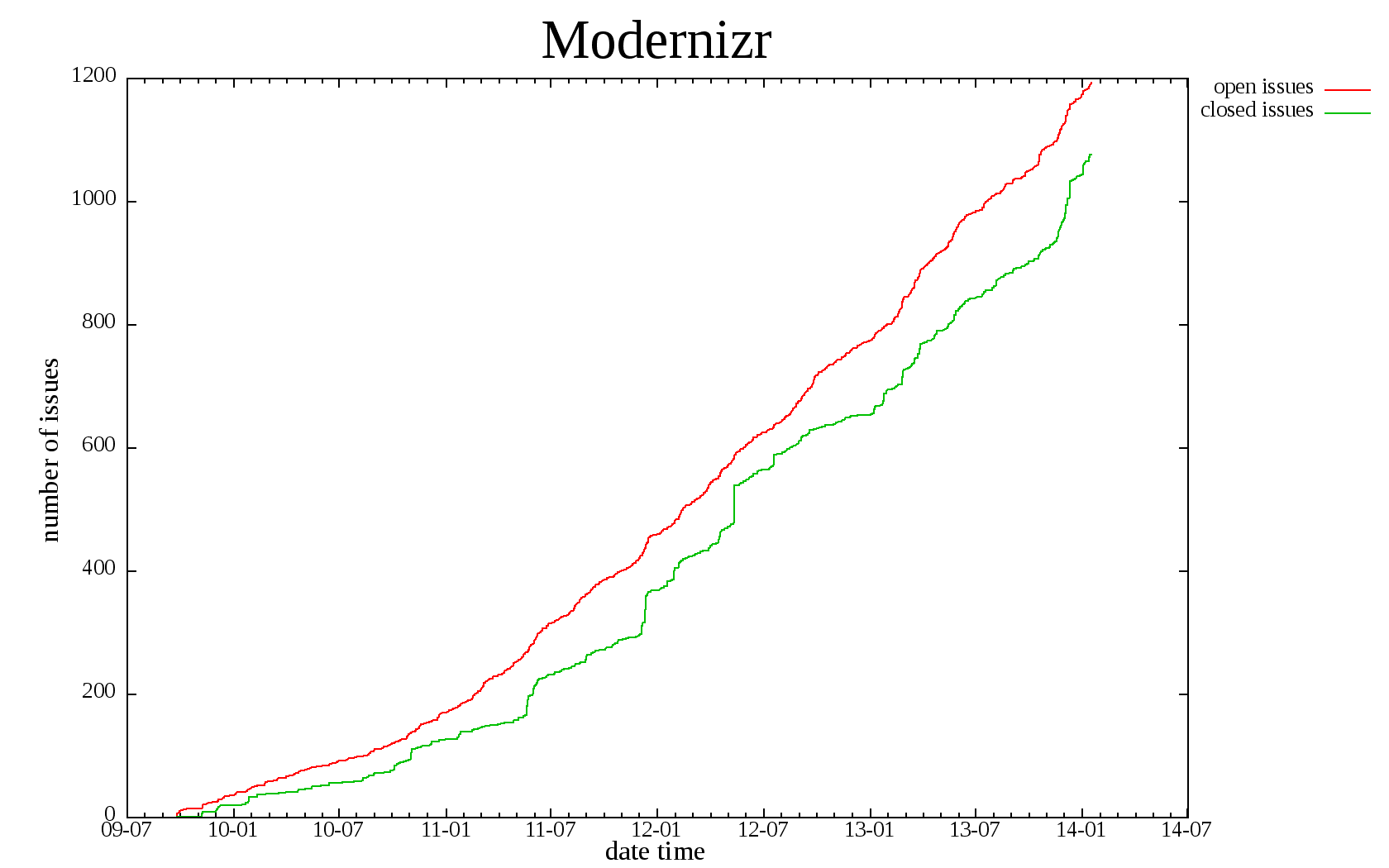
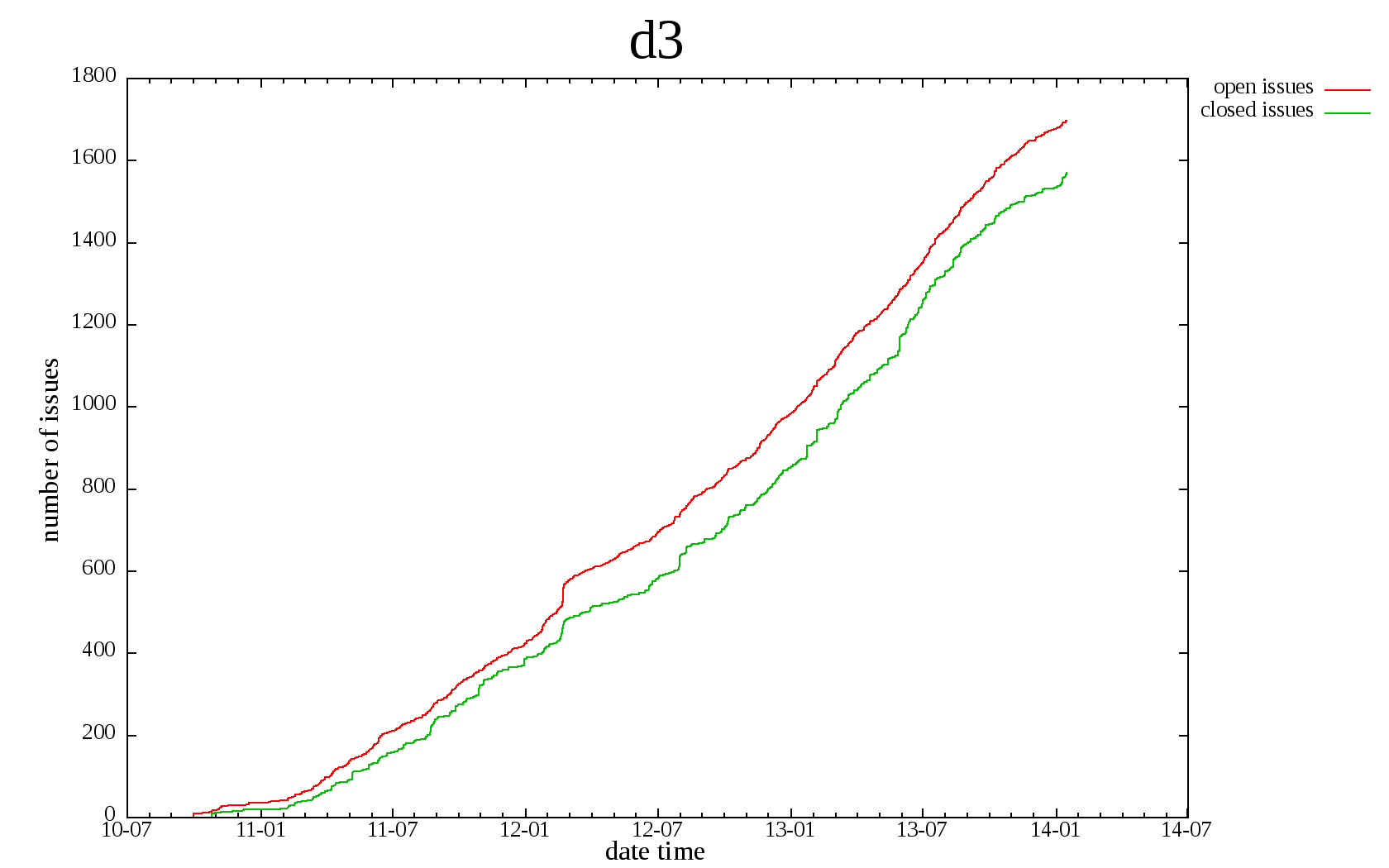
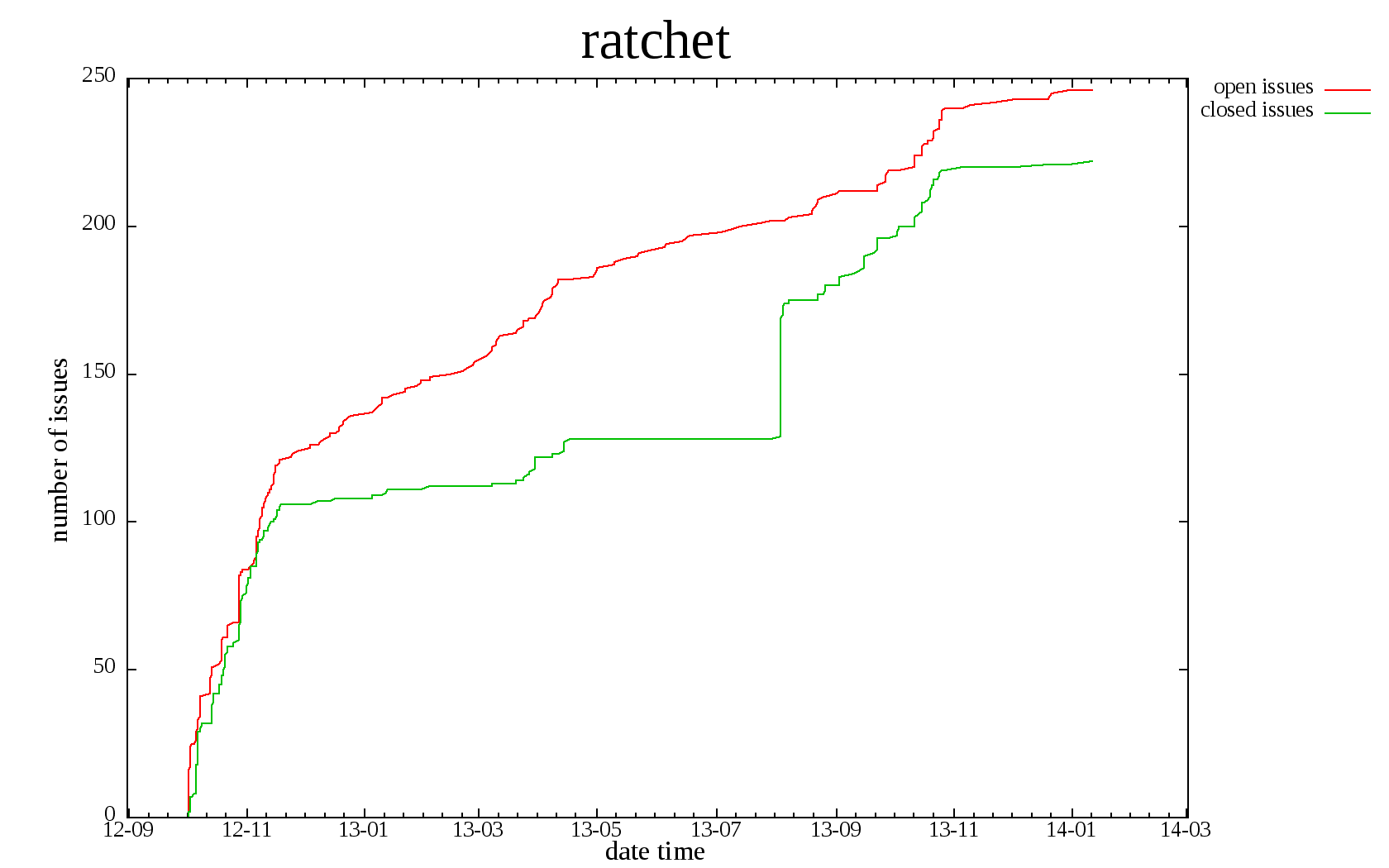
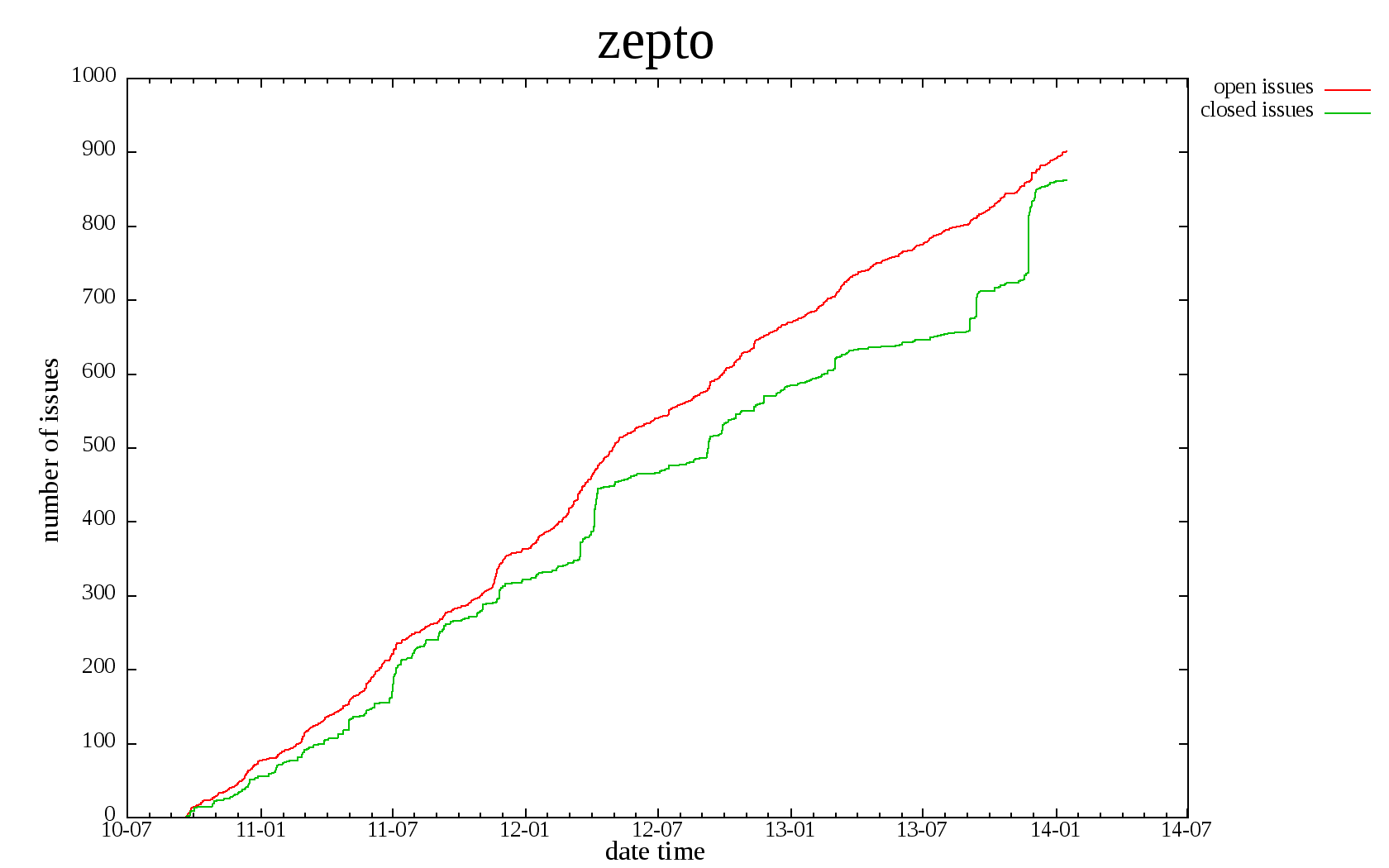
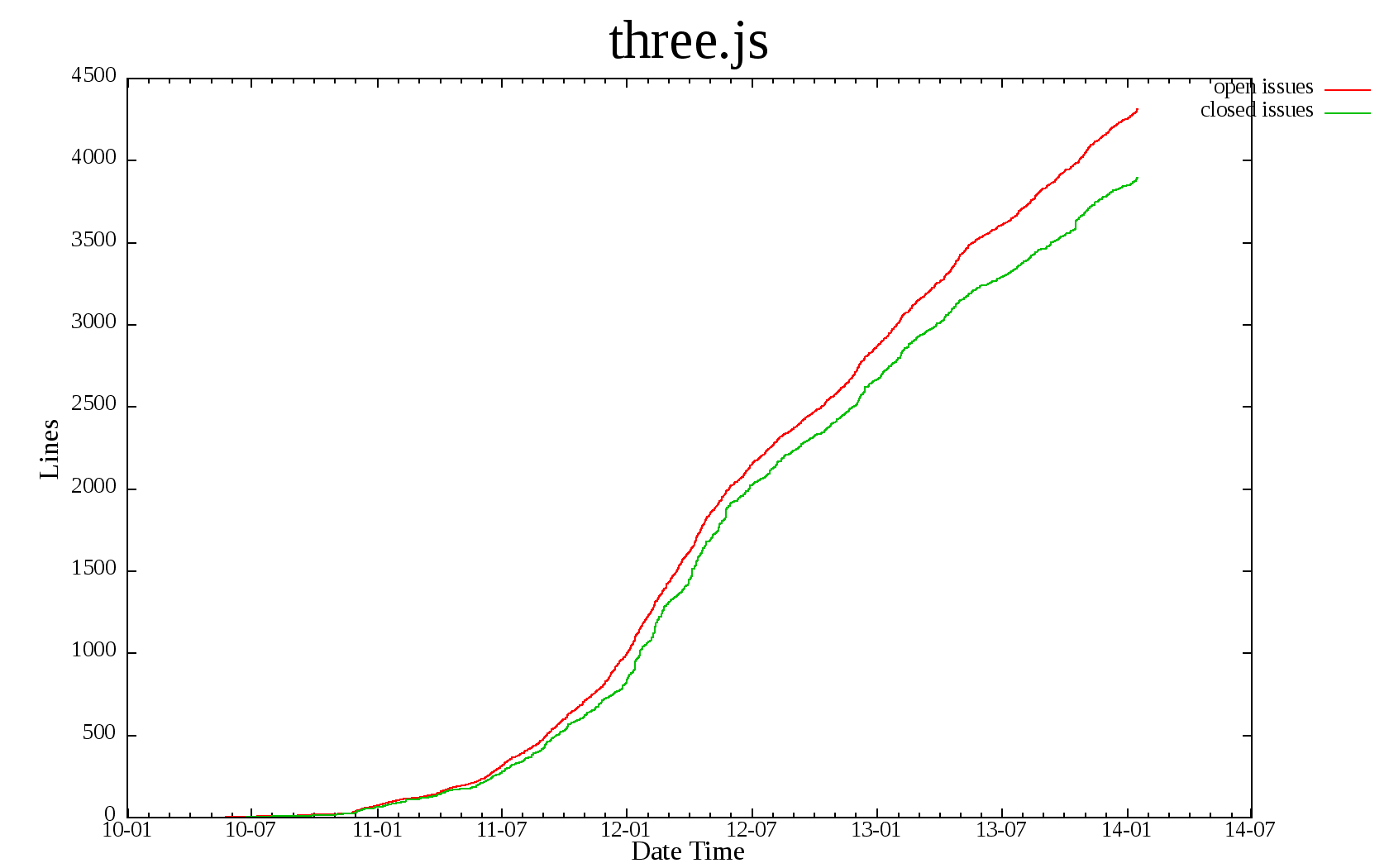
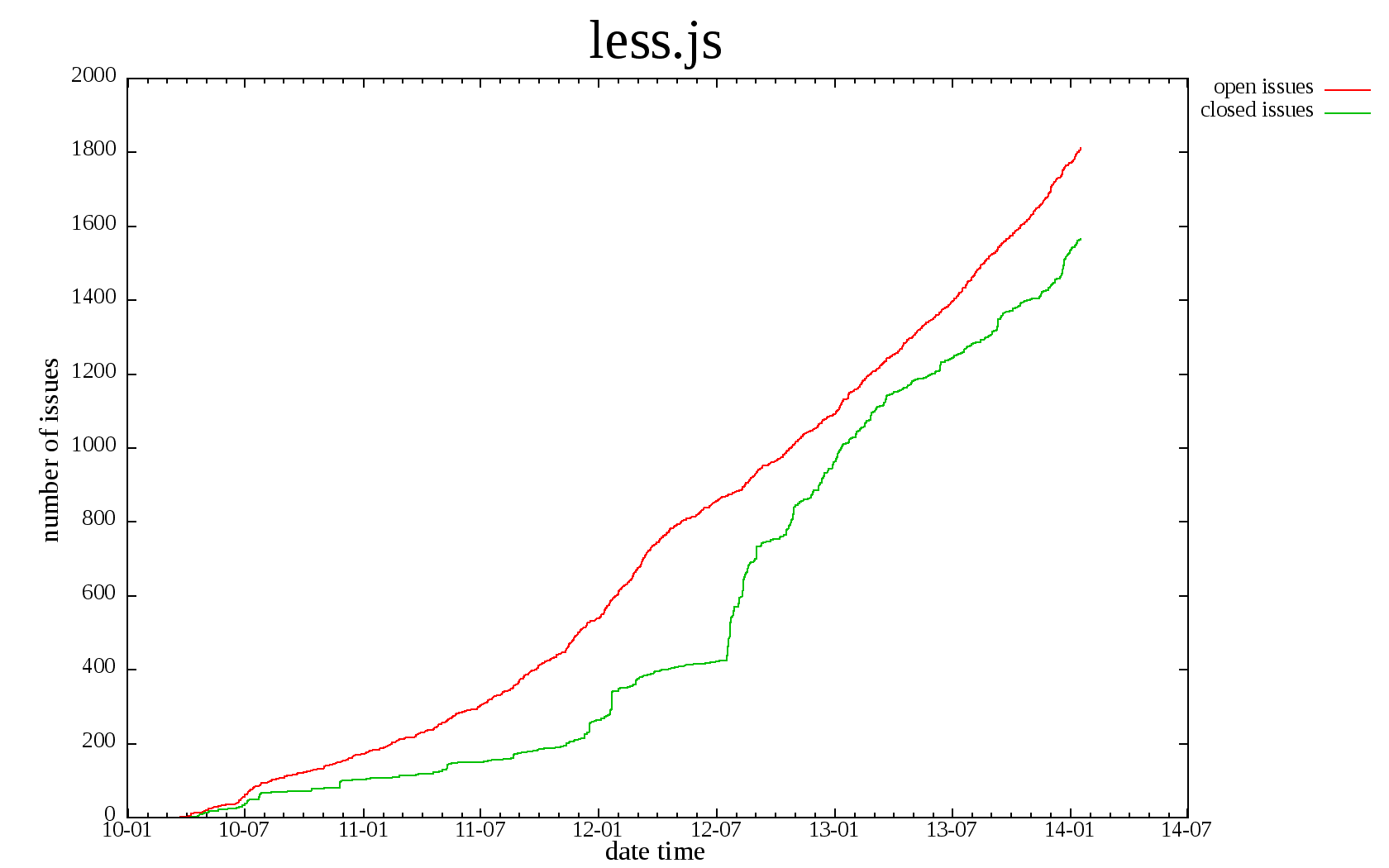
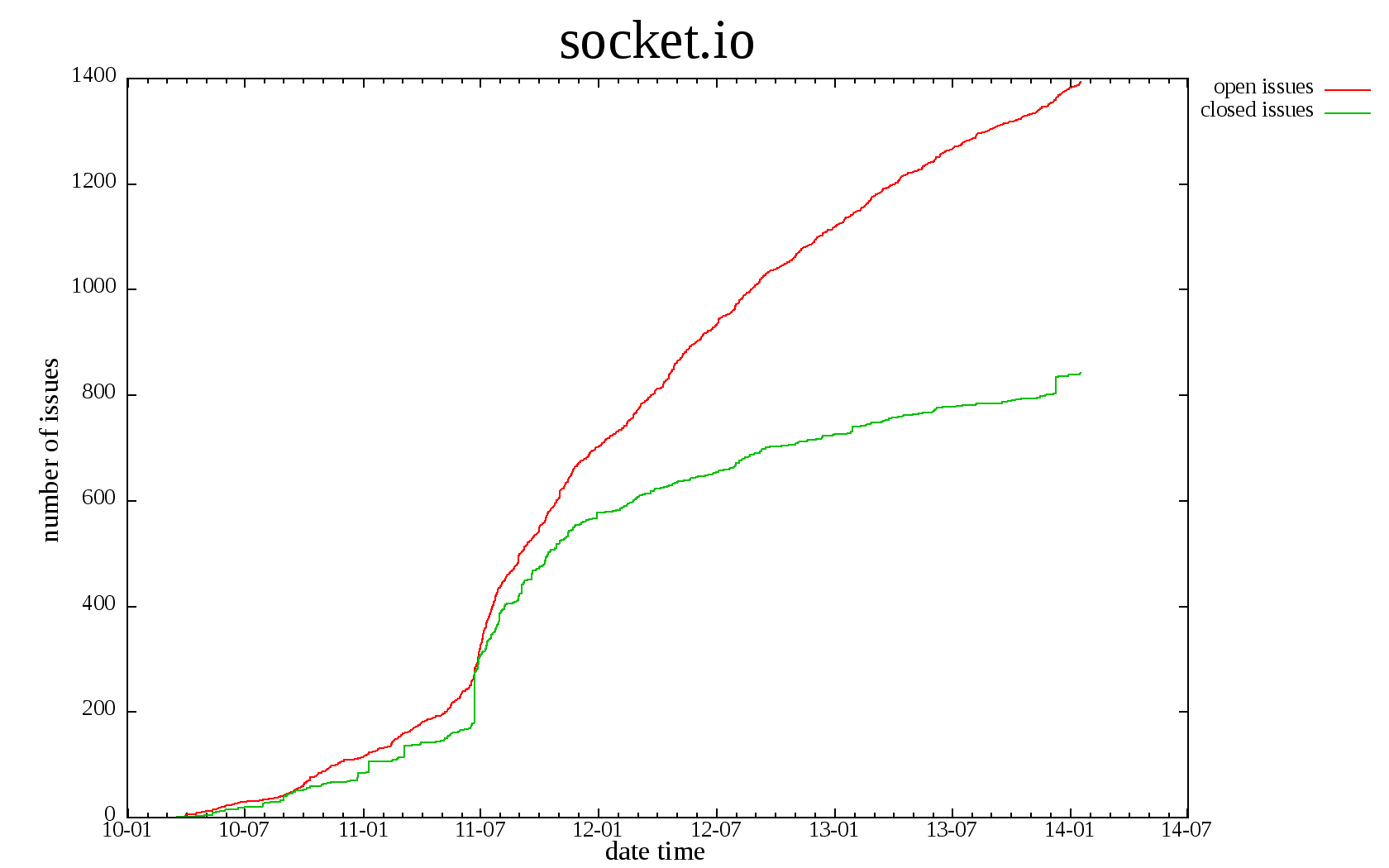
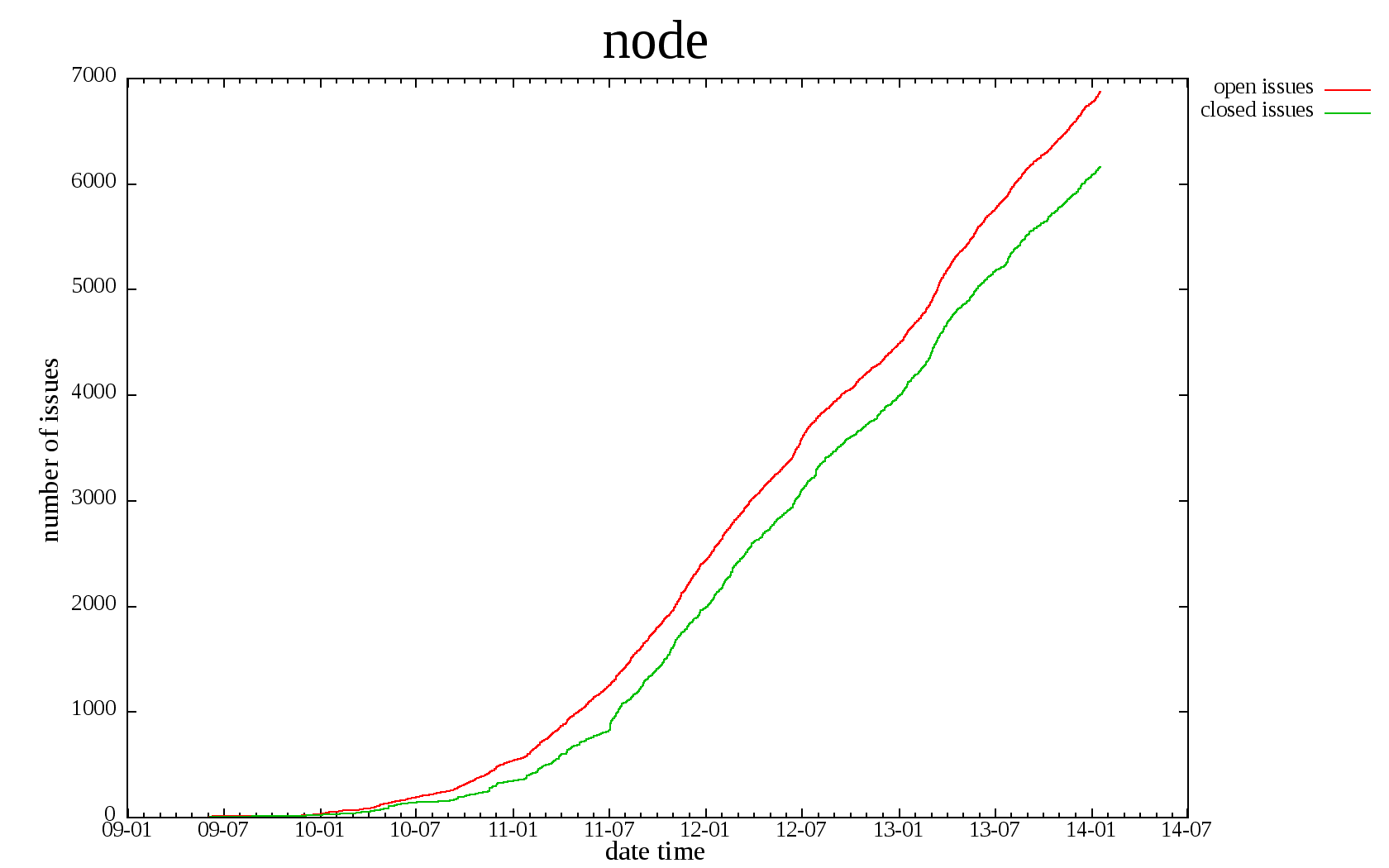
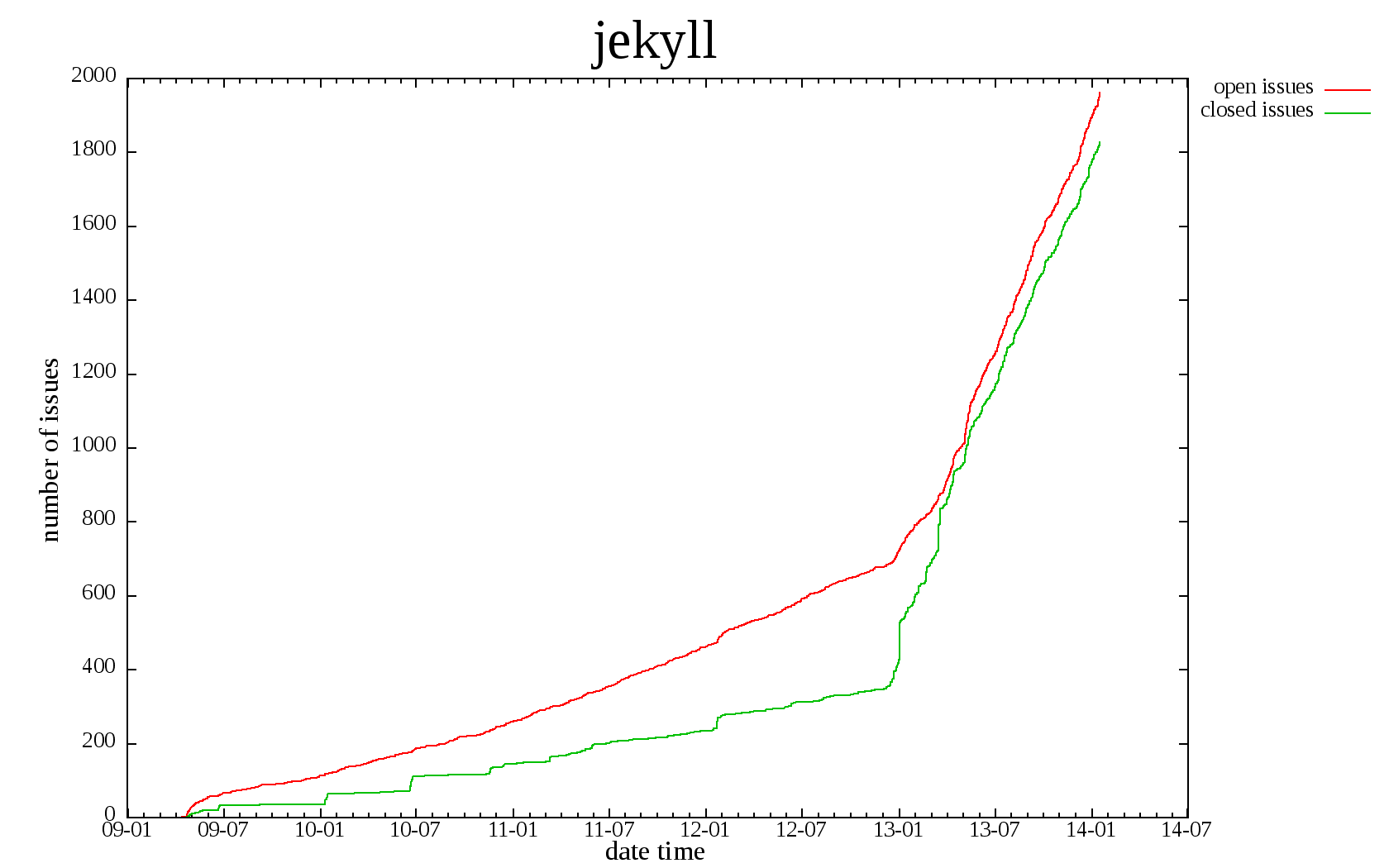
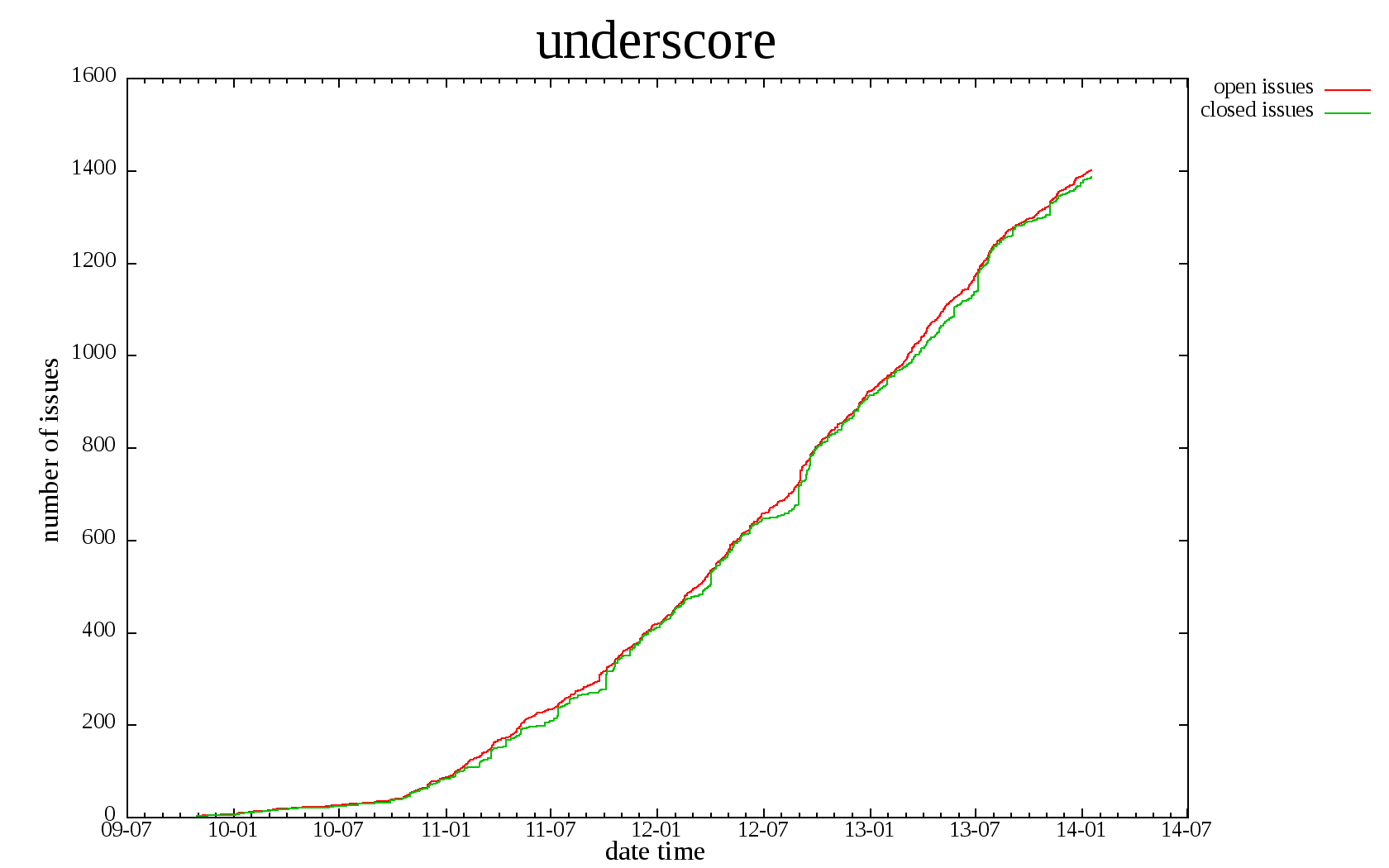
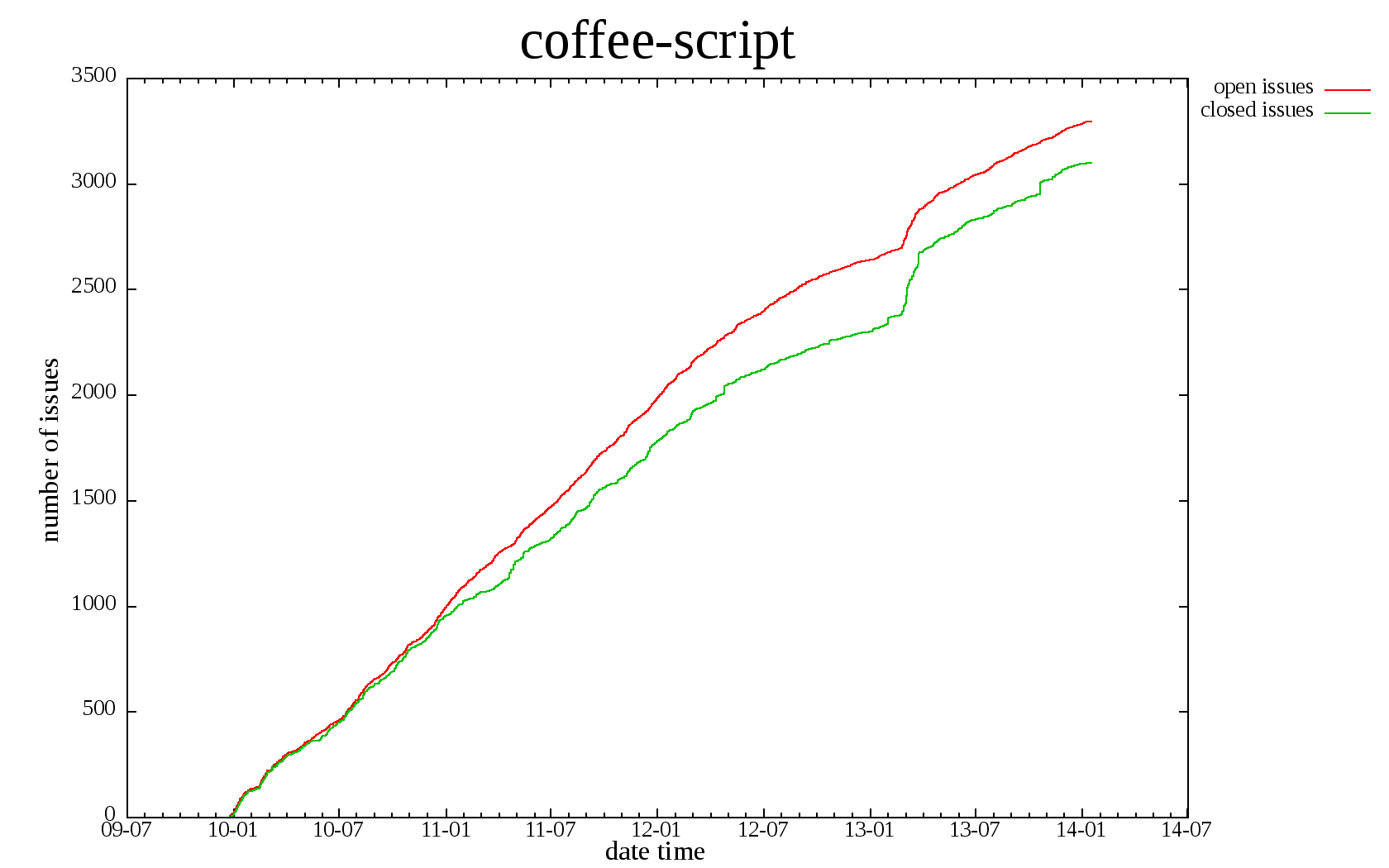
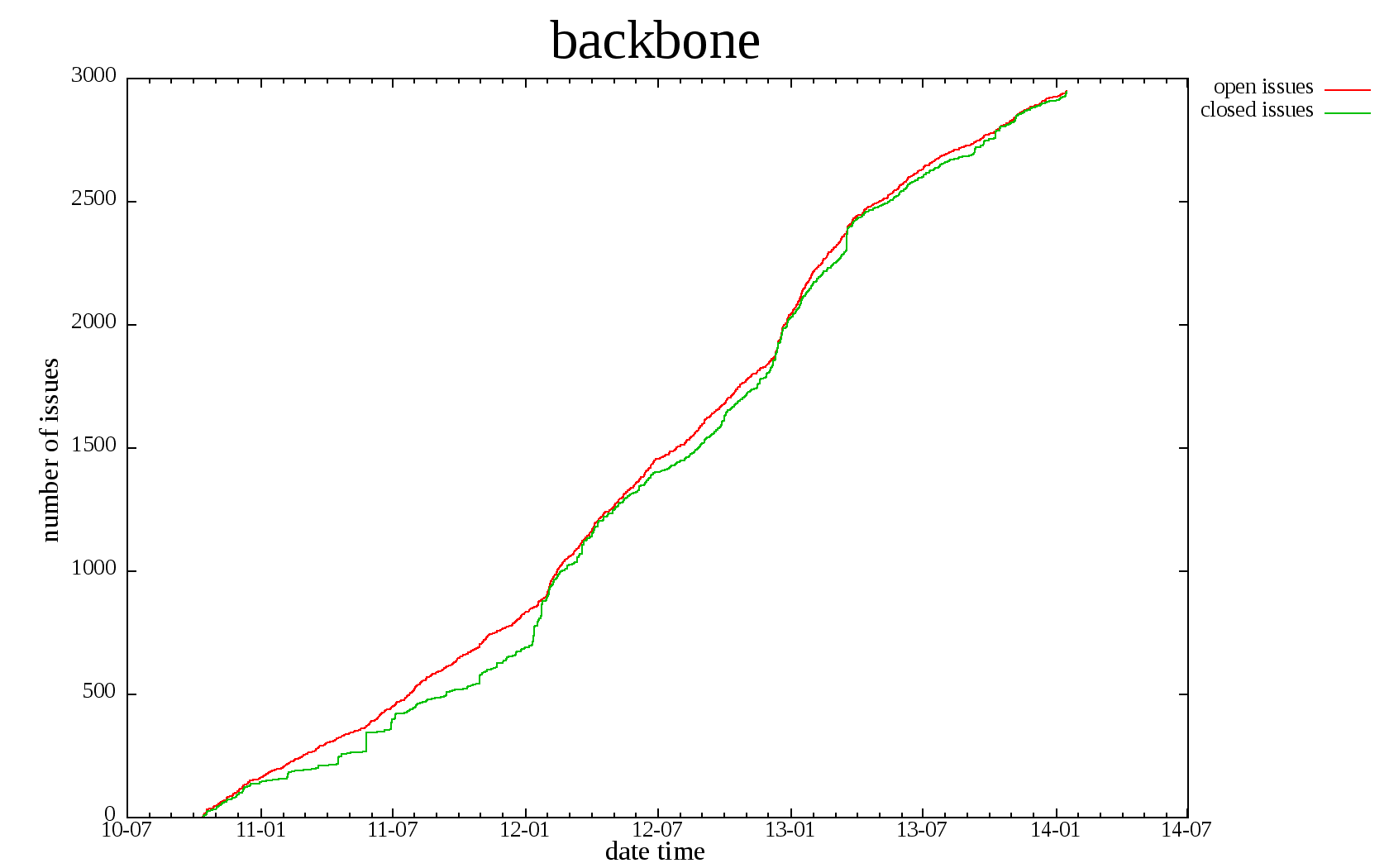
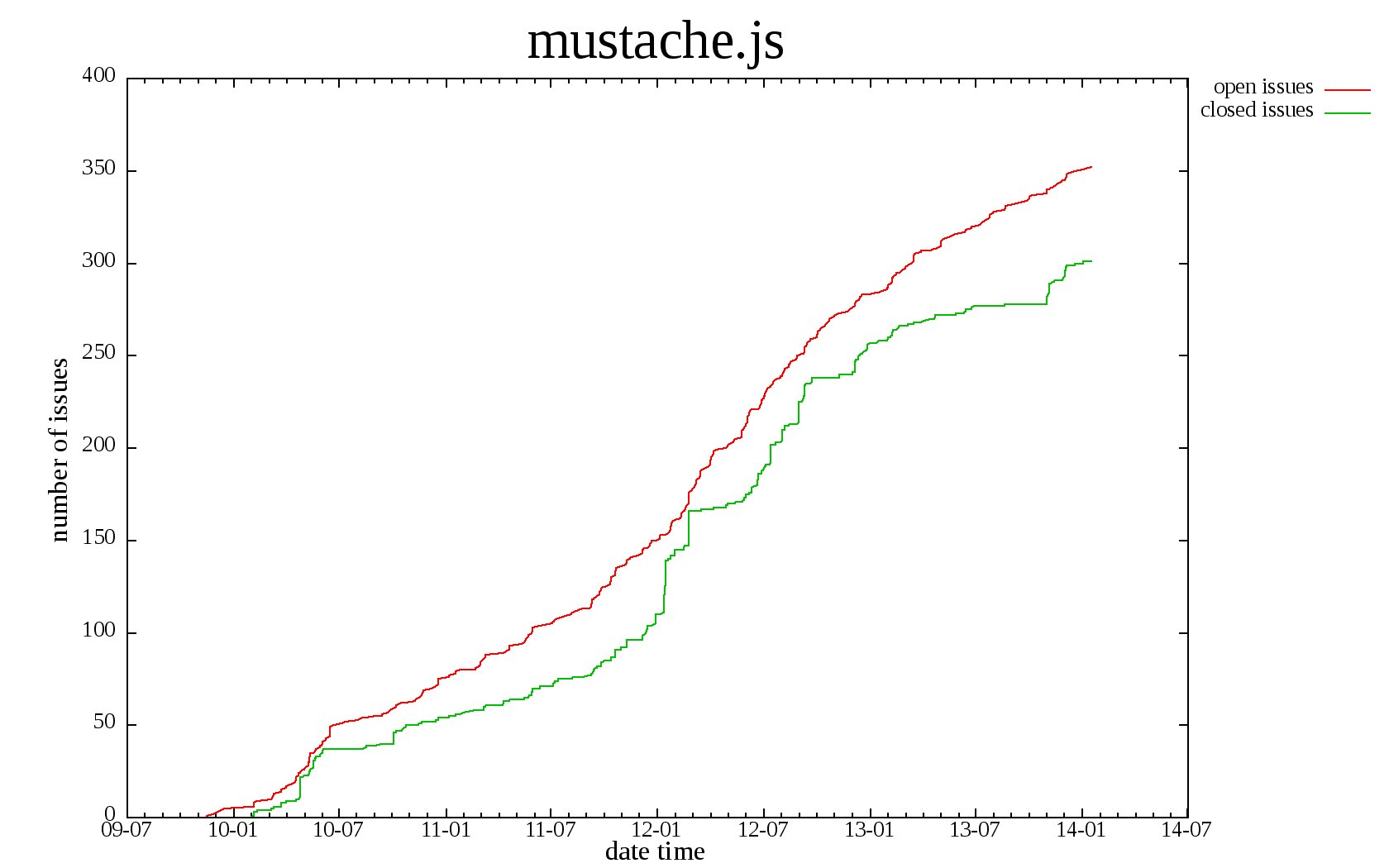
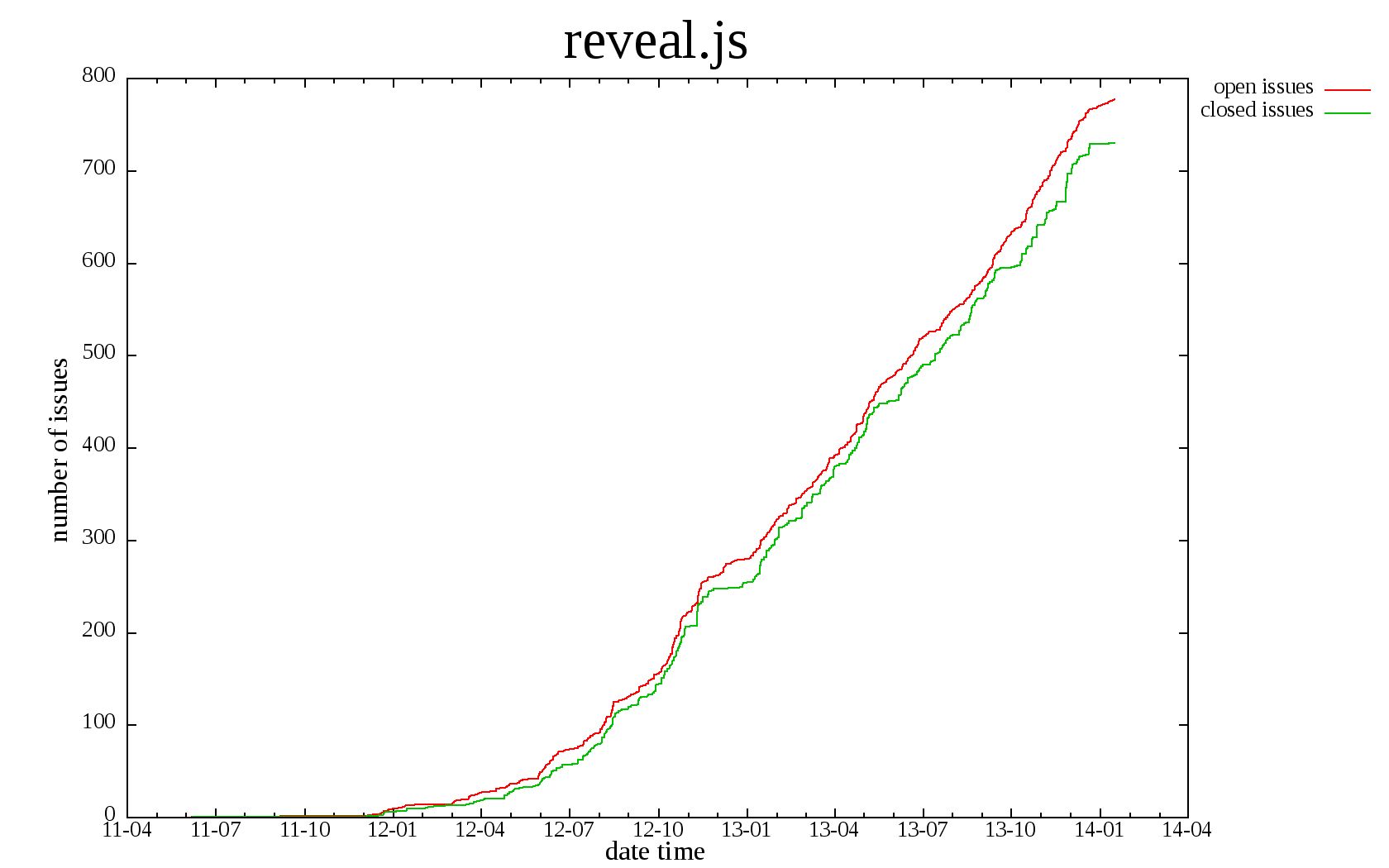
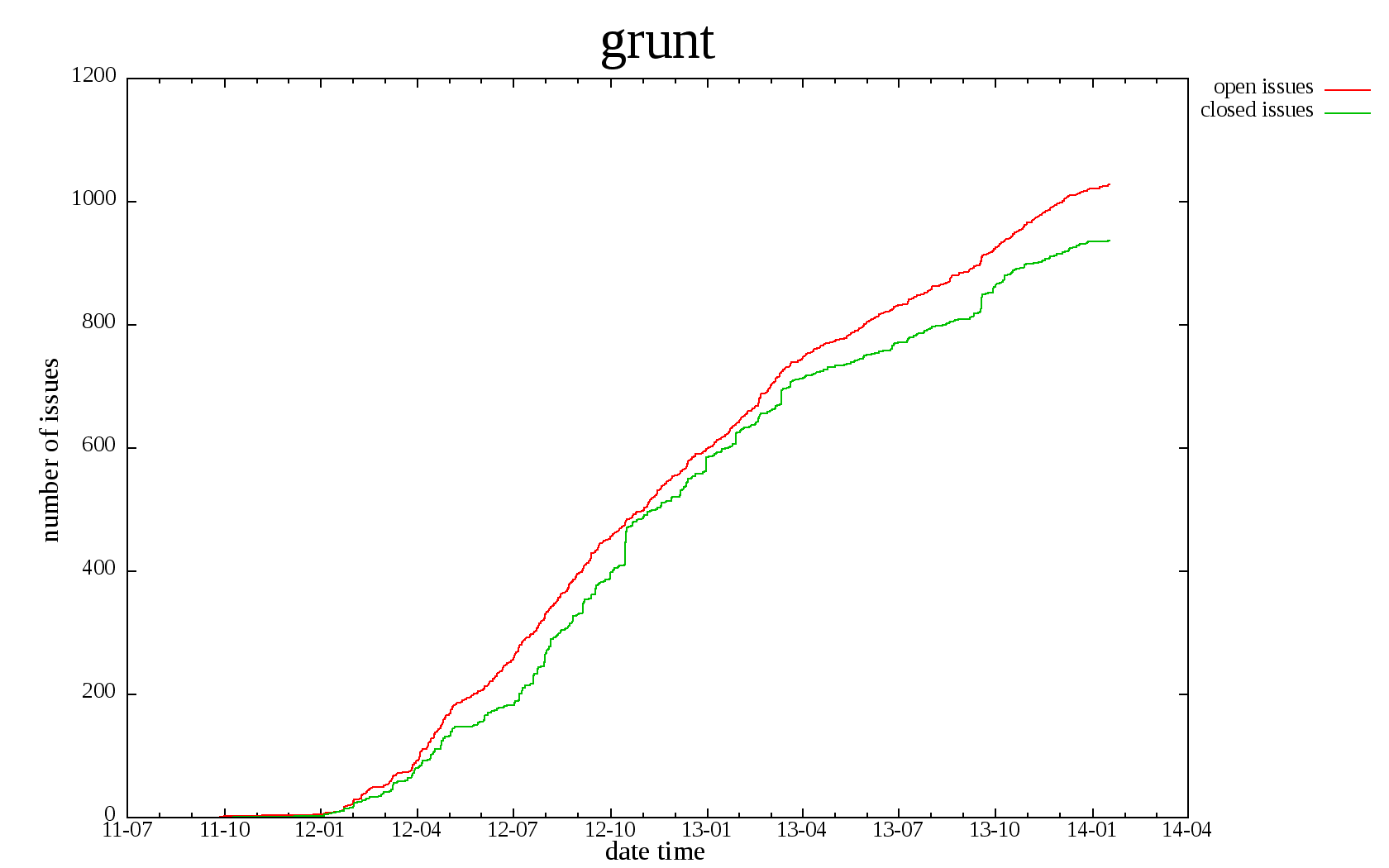
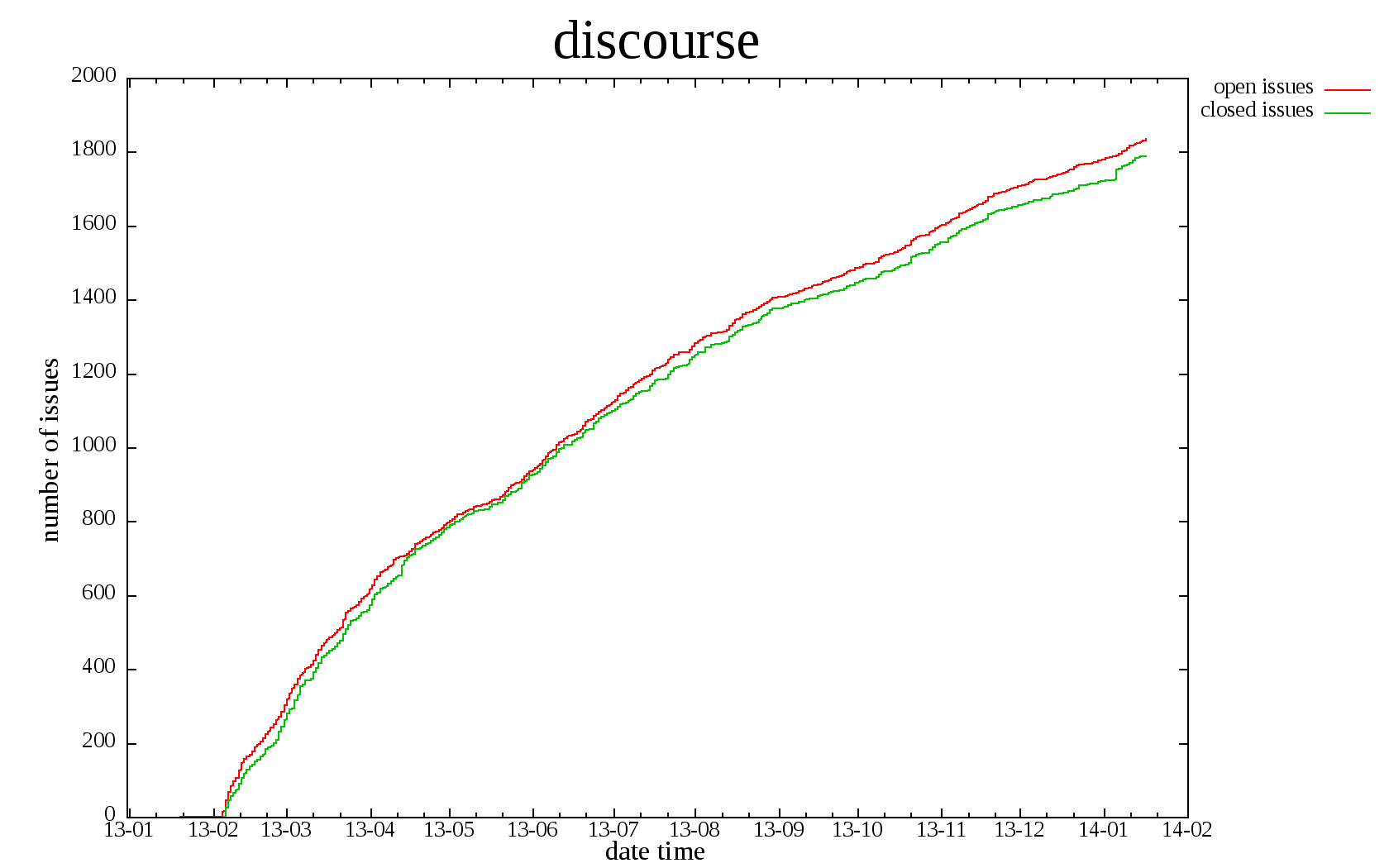
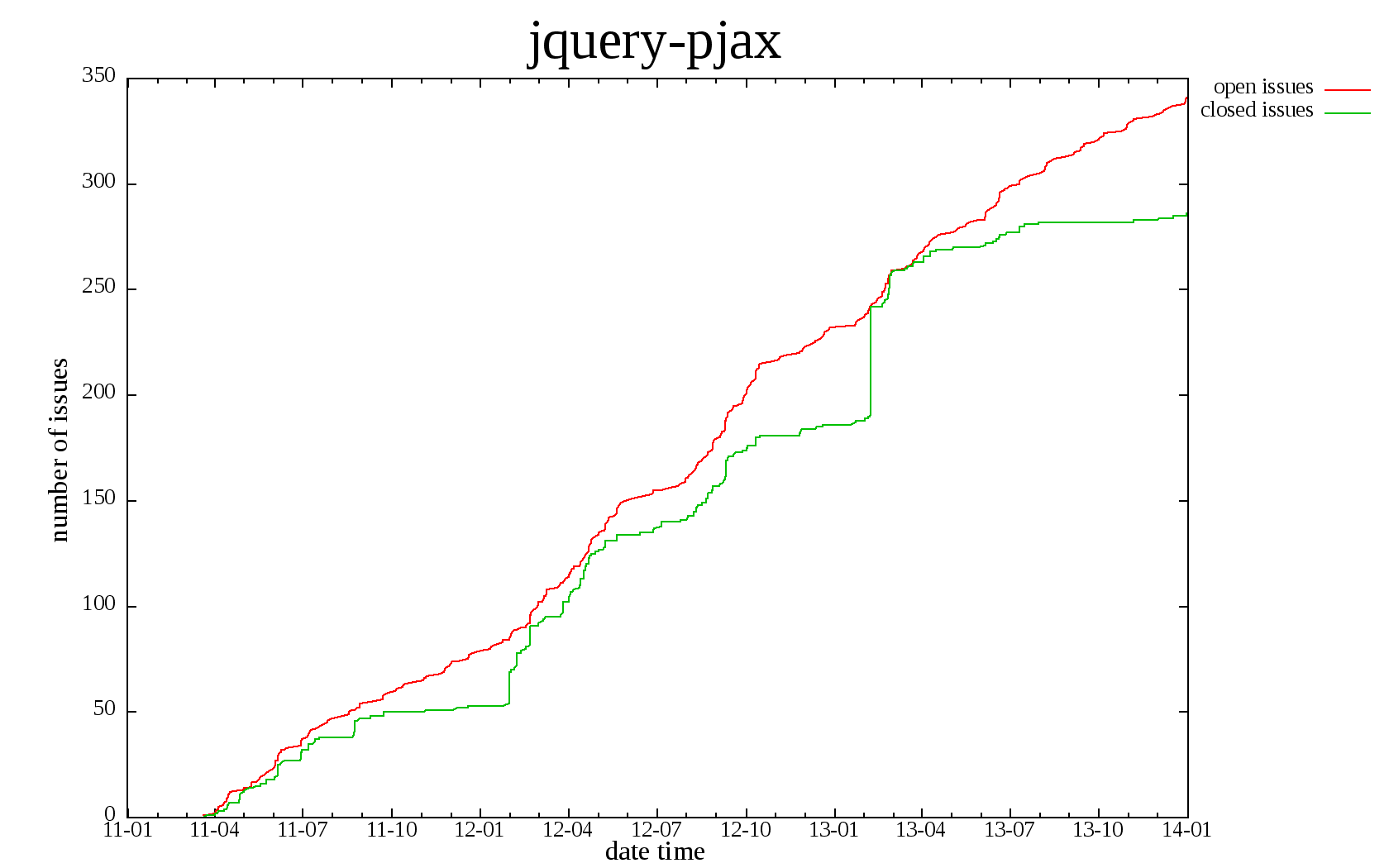
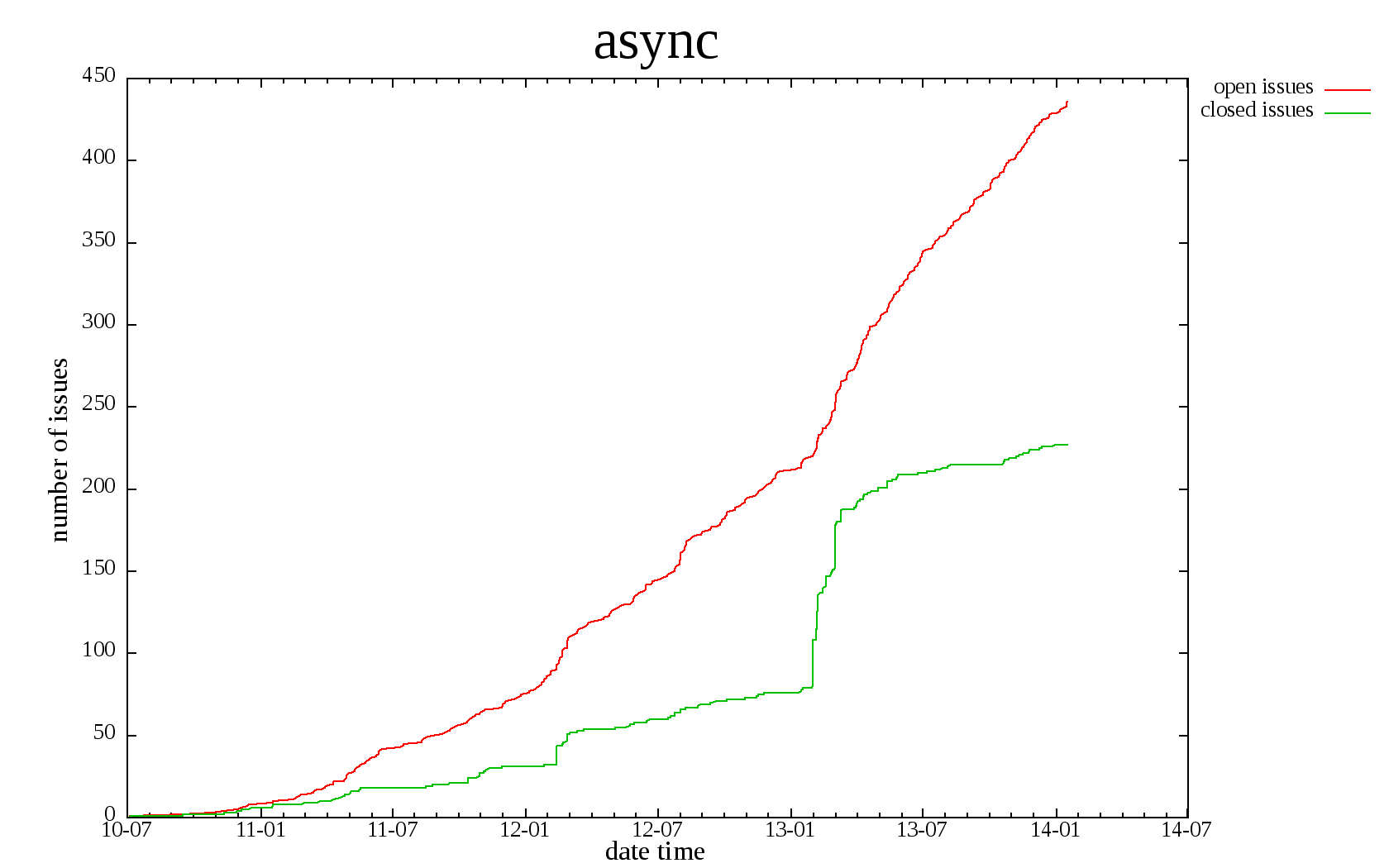
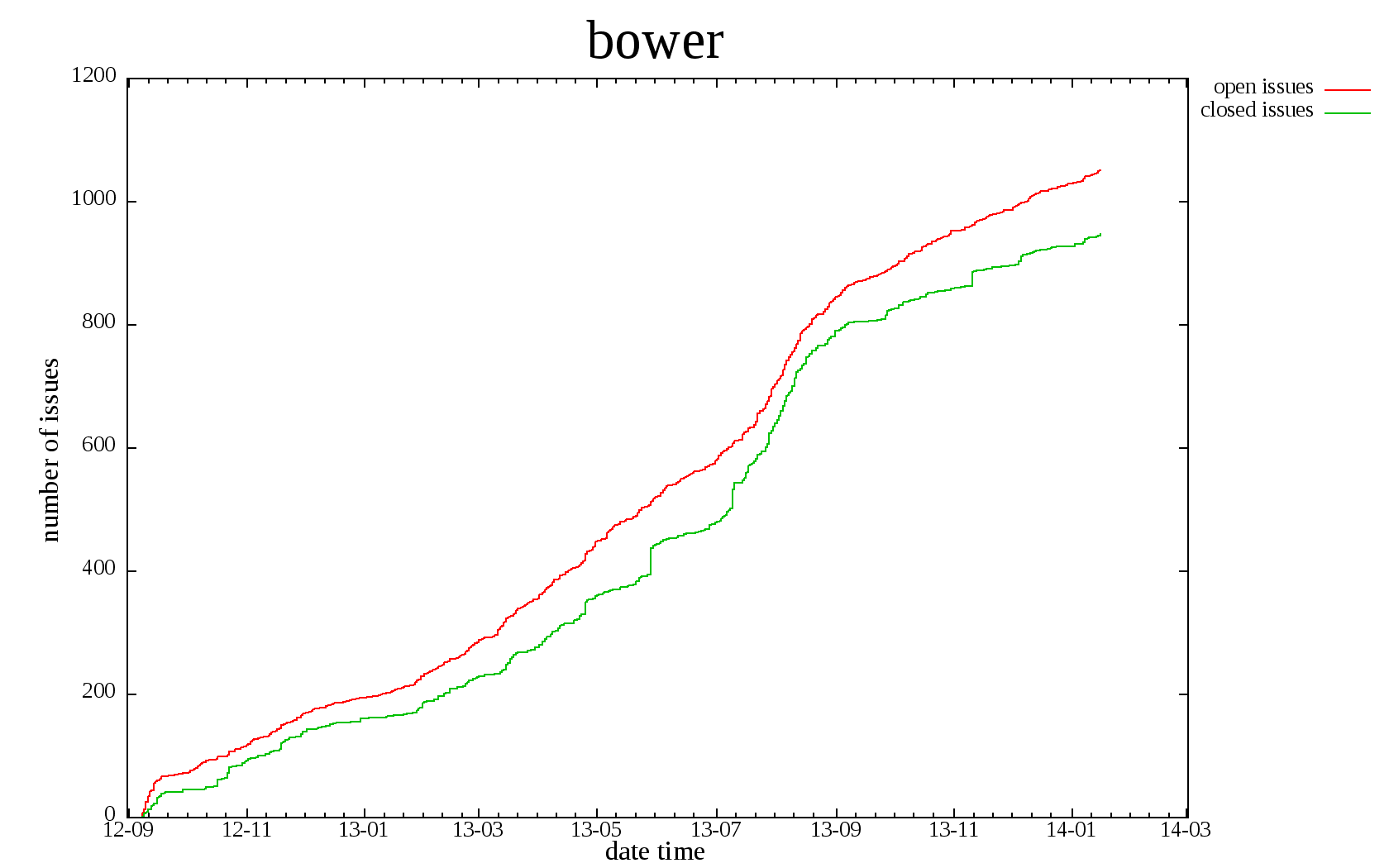
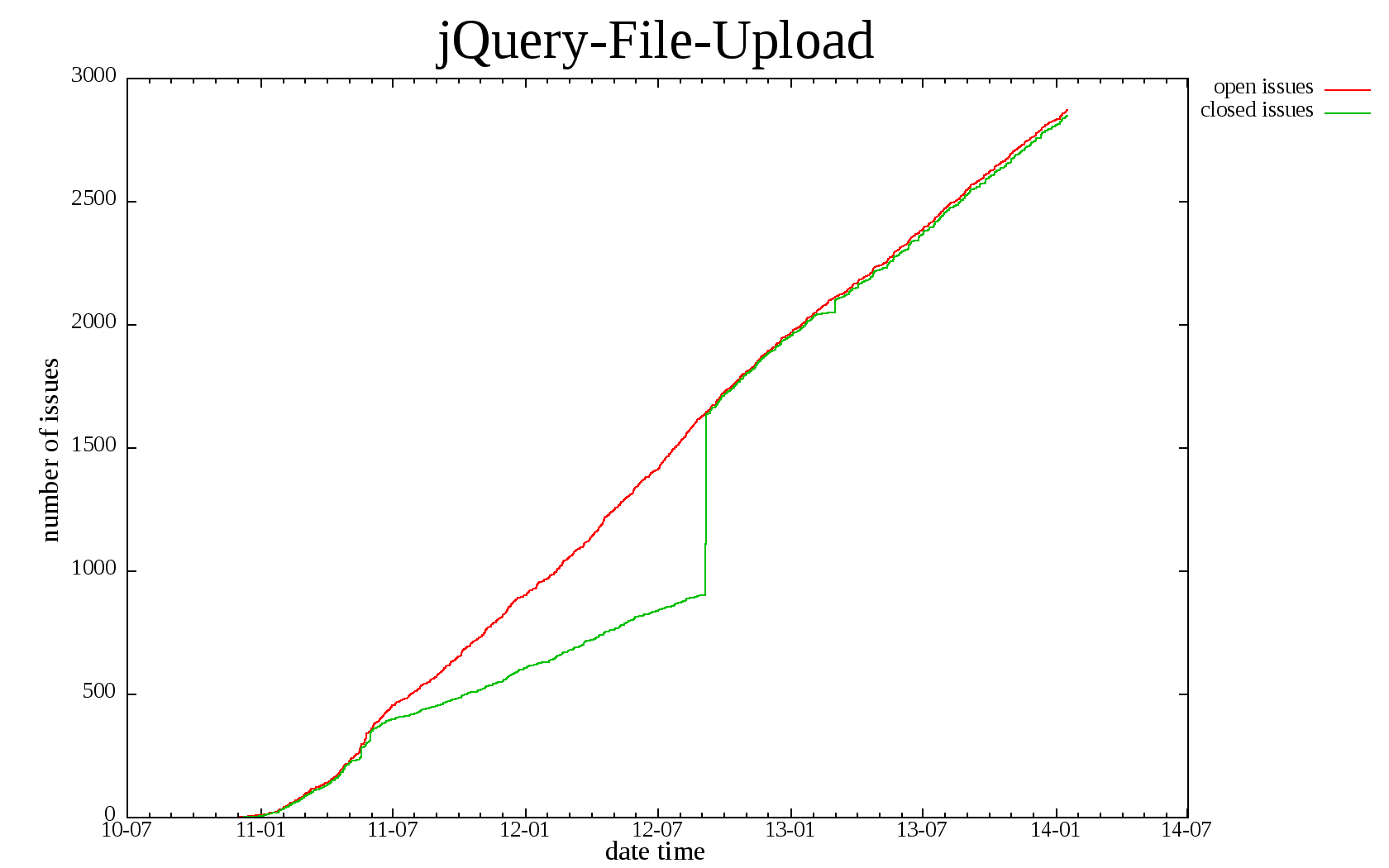


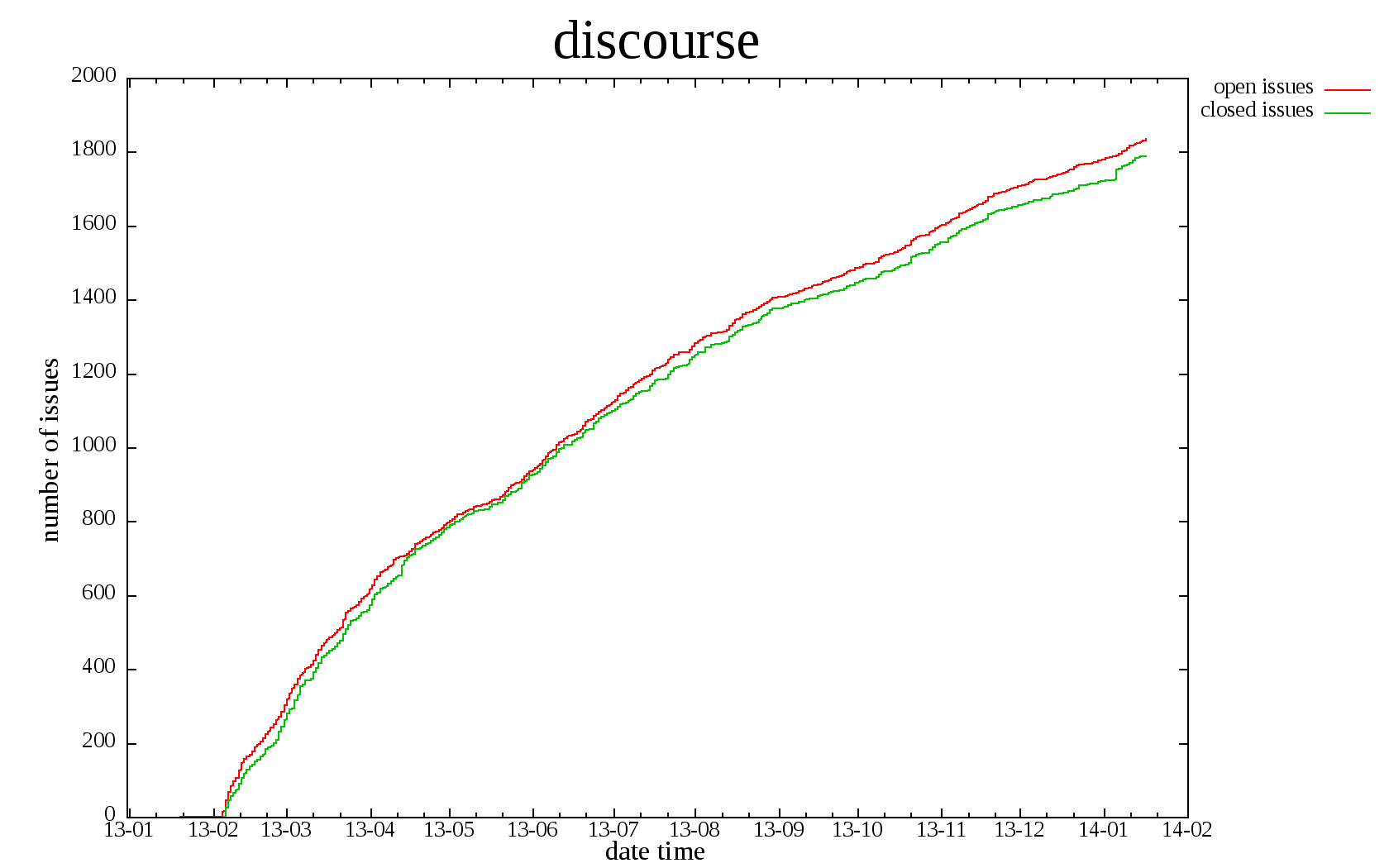
図 5‑3　 Issueの時間変化グラフ/phantoms



* + 1. グラフのグループ分け・解析

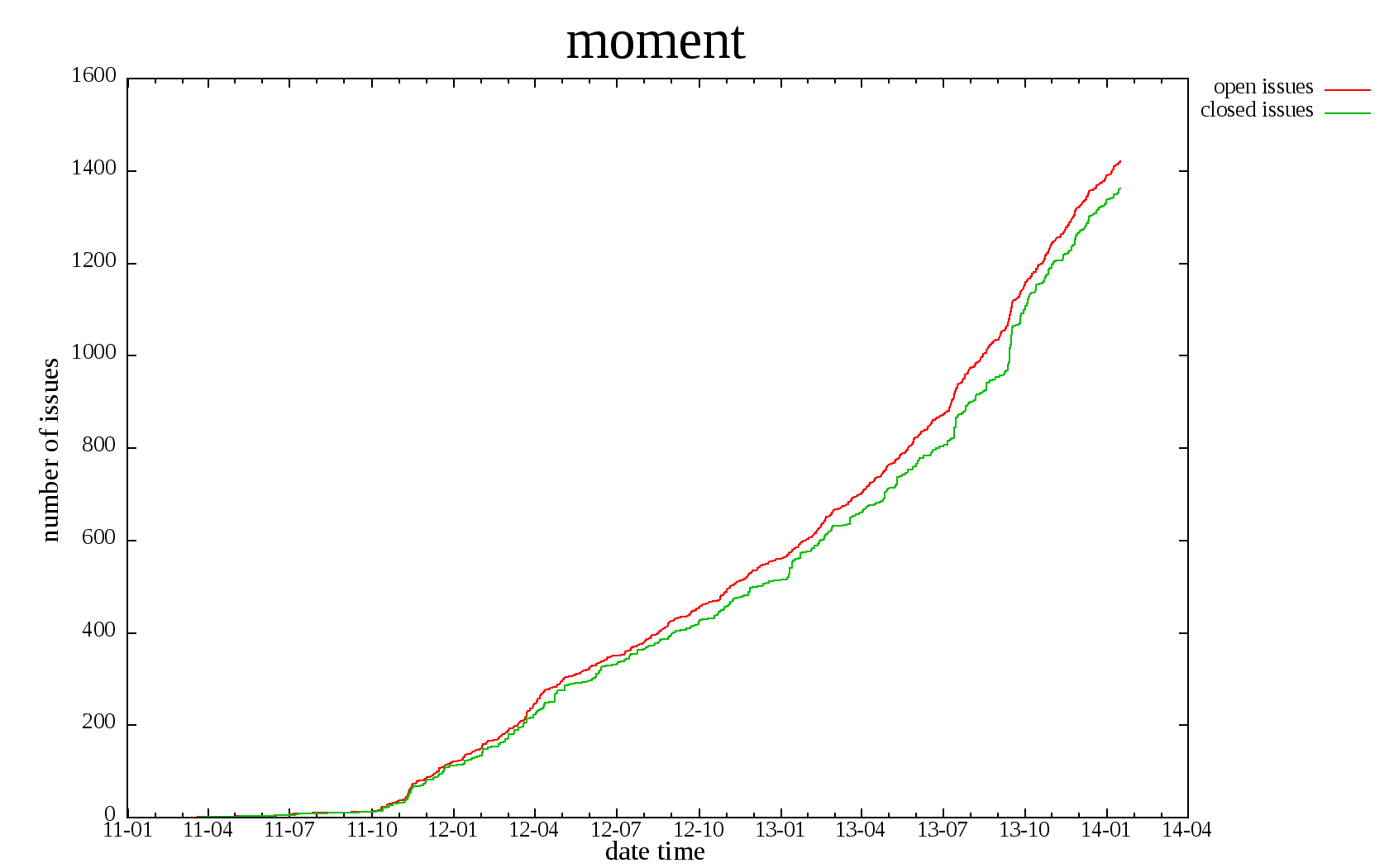
以上のグラフのような結果が得られた．これらのグラフからIssueの特徴を調査する為に，特徴別に4つのグループに分けることが出来た．

1. グラフのような緩やか山なりのグラフ

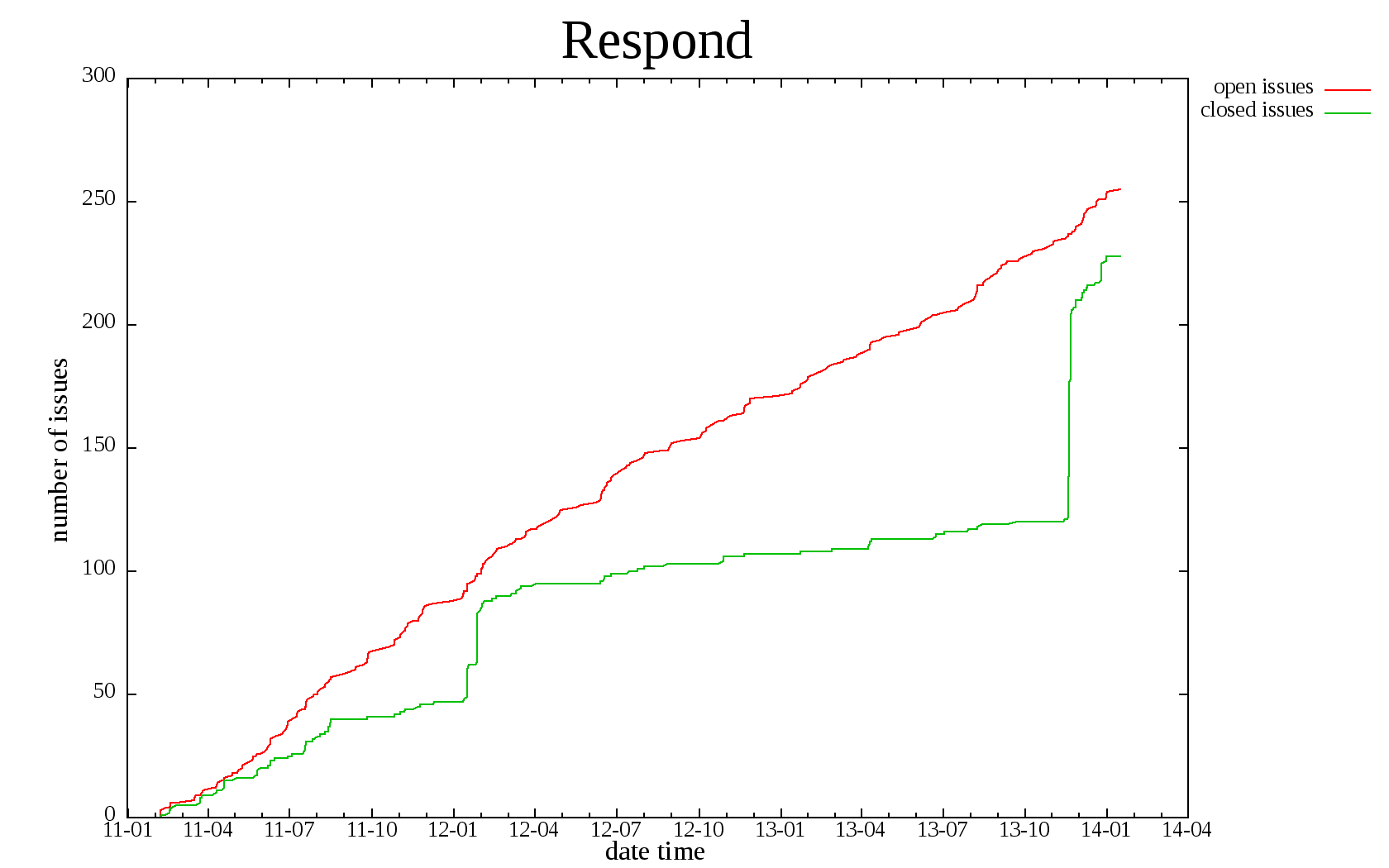


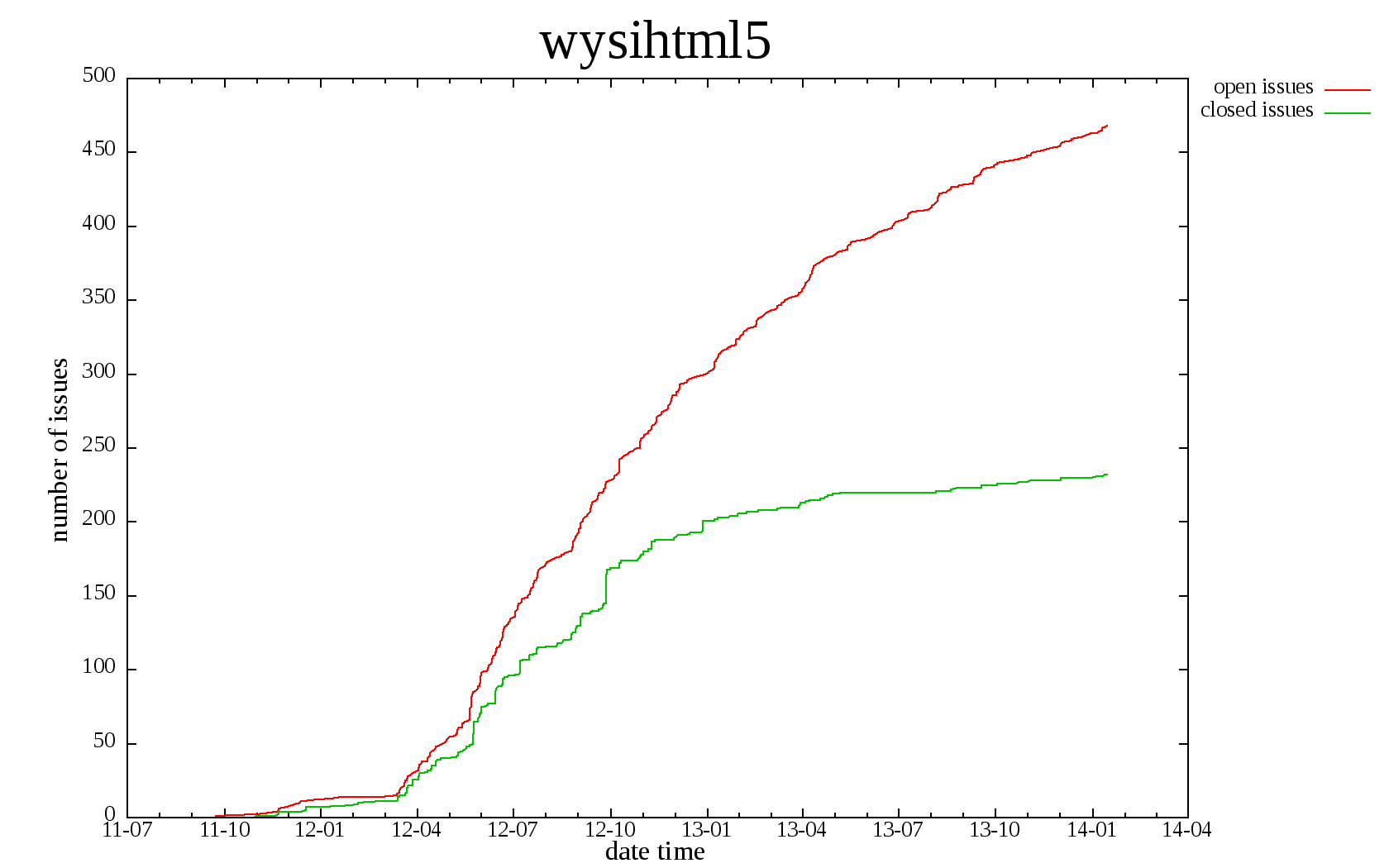
このグラフはIssue数の変化がプロジェクトの前半に多くそれ以降は緩やかにIssueが増えているという特徴がある．

1. ①とは逆に反ったグラフ．



このグラフはIssue数の変化が前半は緩やかで，後半にIssueの数が増えているという特徴がある．





結果

* 1. 考察

　以上の調査結果から考察を行った．