3 - Evolução de Software

Um projeto de software livre dificilmente chega a um final definitivo, já que mesmo quando o software atinge um estágio de desenvolvimento considerado satisfatório para os autores, este estágio pode não ser satisfatório para todos os demais usuários do projeto. Mesmo que o grupo de desenvolvedores originais abandone o projeto em algum momento, nada impede que outros desenvolvedores continuem o desenvolvimento do projeto em uma versão paralela ou mesmo assumam a administração da original (com a permissão dos autores).

Esse fato nos permite dizer que um projeto de software livre está em um contínuo processo evolutivo. Neste capítulo será mostrado uma introdução a evolução de software, a sua importância e como ela se aplica ao software livre.

3.1 - Introdução sobre evolução de software

Um software nada mais é do que uma seqüência lógica de algoritmos, o que nos leva a crer que uma vez que um software realize corretamente os requisitos estabelecidos para os quais ele foi construído, ele nunca mais precisará ser modificado, assumir isto é um erro, pois sistemas sofrem de um processo semelhante ao envelhecimento humano.

3.1.1 O processo de envelhecimento de software

O mundo real está em constante mudança, e sistemas são feitos para refletir comportamentos do mundo real [Gall97], logo é necessário que o software acompanhe as mudanças de requisitos impostas pelo ambiente na qual ele está inserido. Uma falha em acompanhar essas mudanças pode implicar em perda de qualidade por parte do software ou até mesmo no fim da sua vida útil.

O envelhecimento de um software é um processo inevitável, mas podemos tentar entender suas causas, tomar medidas para limitar seus efeitos,

temporariamente reverter os danos causados por ele e se preparar para o dia em que este software não seja mais viável [Lorge94].

Ainda segundo [Lorge94], existem dois tipos de envelhecimento de software: o primeiro ocorre quando os responsáveis por um software falham em adaptá-lo para os novos requisitos, e o segundo ocorre devido ao resultado provocado pela forma como as mudanças são realizadas no software, mudanças essas que tinham o objetivo de satisfazer esses requisitos que mudaram.

Um exemplo do primeiro caso seria um software que no passado funcionava perfeitamente, mas devido ao fato de seu sistema operacional ter caído em desuso e não existir uma nova versão do software para um sistema operacional mais recente foi esquecido por seus usuários. O segundo caso geralmente ocorre quando mudanças no software são feitas por desenvolvedores que não entendem a estrutura original deste, o que fatalmente implicará em algum dano para esta estrutura, que mesmo pequeno irá aumentando conforme novas atualizações forem sendo feitas, e com o passar do tempo cada nova mudança se tornará mais difícil e conseqüentemente cara. Caso não seja feita uma reestruturação do software, este chegará em um ponto onde novas atualizações ficarão inviáveis.

As desvantagens causadas pelo envelhecimento de um software são a perda de performance devido a modificações não adequadas na sua estrutura interna, número crescente de novos erros devidos a alterações indevidas no código e perda de usuários devido à falta de meios para concorrer com versões mais recentes de sistemas semelhantes.

Como pode ser observado, qualquer software que não tenha uma expectativa de vida curta, pode vir a sofrer os efeitos nocivos do envelhecimento. Apesar de inevitável, estes efeitos podem ser atrasados ou consideravelmente diminuídos, desde que sejam seguidos alguns cuidados no desenvolvimento e evolução do software em questão. Alguns destes cuidados mais importantes são segundo [Lorge94]:

1. Estruturar o software para a evolução

Sempre que um software tenha uma expectativa que não seja curta, deve-se fazer sua estrutura visando facilitar a evolução. Como não é possível se saber com exatidão quais mudanças serão feitas no futuro, deve-se, portanto avaliar as partes do software que estarão mais sujeitas a mudanças no decorrer de sua vida útil e desenvolvê-las de forma que estas mudanças ocorram mais facilmente. Apesar desta ser uma regra de programação aconselhável para todos os sistemas, normalmente ela é negligenciada devido aos prazos apertados da maioria dos projetos.

2. Documentar adequadamente

Nem sempre a documentação de um projeto é escrita pela pessoa mais qualificada para tanto, e mesmo que esta venha a ser escrita adequadamente, é sempre necessário que seja atualizada a contento à medida que novas mudanças forem sendo feitas no código. Deve-se ter em mente que esta documentação poderá ser usada para atualizar o sistema daqui a vários anos, e é bem possível que estas atualizações não sejam feitas pelos autores originais, logo deve-se fazer um esforço adicional para que a documentação seja clara e concisa com o software, e que seja compreendida por outros que não os seus autores.

3. Revisar a estrutura

Deve-se ter em mente que sempre que a estimativa de vida útil do software seja longa, revisões da estrutura são fundamentais. Caso exista uma equipe própria para a manutenção, é sempre uma boa política deixá-la participar da revisão. Vale lembrar que as revisões devem começar antes mesmo da codificação, tais revisões são baratas, rápidas e podem poupar muito tempo e recursos no futuro. Estranhamente estas revisões são muito pouco utilizadas na prática.

O processo de envelhecimento de um software é inevitável, o que gera uma necessidade constante de evolução por parte de todos os sistemas que esperam se manter ativos por um período grande de tempo. Para entender o processo evolutivo em questão, é necessário entender as oito leis da evolução de software, estas leis também são conhecidas pelo nome de “Leis de Lehman”.

3.1.2 As oito leis de Lehman

Estas leis começaram a ser formuladas no começo dos anos 70, com a análise do processo de programação da IBM, neste período foram formuladas as três primeiras leis, na década seguinte foram apresentadas as duas seguintes e as três restantes vieram na década de 90, sendo que a sexta lei foi apresentada em [lehman91] e as duas restantes em [lehman96]. Estas leis se aplicam a qualquer software que resolva um problema ou implemente uma solução computacional no mundo real; estes sistemas são denominados “sistemas do tipo E”.

A formulação das leis de Lehman foi baseada inicialmente na evolução de dois sistemas operacionais (IBM OS/360 e ICL VME Kernel), um sistema financeiro (Logica FW), um sistema de telecomunicações (Lucent) e um sistema de defesa (Matra BAE Dynamics), além de se basear como já foi dito, no processo de programação da IBM.

3.1.2.1 I - Mudança contínua

“Um sistema de informação que é usado deve ser continuamente adaptado, caso contrário se torna progressivamente menos satisfatório”.

Esta lei sugere que sistemas sofrem de um processo parecido com o envelhecimento humano; este envelhecimento é resultado de inconsistências do software e do domínio em que este está inserido, já que este domínio faz parte do mundo real e está sempre em contínua evolução. A evolução deve ser feita baseada no retorno dos usuários e seu nível de satisfação, caso haja resistência em

se evoluir o software, e conseqüentemente se adaptá-lo a realidade de seus usuários, seu nível de satisfação cairá com o tempo.

3.1.2.2 II - Complexidade crescente

"À medida que um programa é alterado, sua complexidade cresce a menos que um trabalho seja feito para mantê-la ou diminuí-la".

Uma vez que a necessidade de adaptação cresce e mudanças são sucessivamente implementadas, interações e dependências entre os elementos do sistema crescem em um padrão desestruturado e levam a um crescimento da entropia do sistema. A cada nova mudança, a estrutura original do software se tornará mais fragmentada, e o custo de novas mudanças aumentará gradativamente, até o momento em que estes custos não mais serão viáveis. Será então necessário um trabalho de reestruturação do software para que este tenha sua complexidade diminuída.

3.1.2.3 III - Auto-regulação

"O processo de evolução de software é auto-regulado próximo à distribuição normal com relação às medidas de produtos e atributos de processos".

A evolução de um software é implementada por um grupo de técnicos, que opera dentro de uma organização maior. Os interesses desta organização e seus objetivos se estendem bem acima do sistema em questão. Pontos de controle serão estabelecidos pela gerência para garantir que as normas operacionais serão seguidas e os objetivos organizacionais serão alcançados em todos os níveis.

Os controles de retorno positivos e negativos dos pontos de controle são um exemplo de mecanismos de estabilização. Existem vários outros, e juntos eles estabelecem uma dinâmica disciplinada cujos parâmetros são, pelo menos em parte, normalmente distribuídos. Depois de um tempo este grupo estabelecerá uma

dinâmica que fará com que o esforço incremental gasto em cada nova versão permaneça constante durante a vida do sistema.

3.1.2.4 IV - Conservação da estabilidade organizacional (taxa constante de trabalho)

"A taxa de atividade global efetiva média em um sistema em evolução é constante sobre o tempo de vida do produto".

De todas as oito leis, esta é sem dúvida a menos intuitiva, já que ainda se acredita que a taxa de atividade global gasta em um sistema em evolução é decidido pelos gerentes responsáveis; ao contrário, os projetos analisados mostram que essa taxa se estabiliza em um nível constante, e que na prática o nível de atividade de um projeto não é decidido exclusivamente pela gerência. Isso se deve ao fato de que o nível de atividade vai ser decidido pelas necessidades dos usuários e seus retornos, como mostrado na terceira lei, este nível após um período de tempo, tende a se manter constante, e o nível de pessoas trabalhando no software não pode crescer indefinidamente, já que um aumento muito grande acarreta em um aumento igualmente grande na entropia do sistema, o que pode até mesmo resultar em uma diminuição da taxa de atividade global.

3.1.2.5 V - Conservação da Familiaridade

"Durante a vida produtiva de um programa em evolução, o índice de alterações em versões sucessivas é estatisticamente invariante".

Um fator determinante na evolução de um software é a familiaridade de todos os membros da equipe com os objetivos desta, quanto mais mudanças forem necessárias, maior vai ser a dificuldade de que toda a equipe esteja ciente dos objetivos. A taxa e qualidade de progresso e outros parâmetros são influenciados, até mesmo limitados, pela taxa de aquisição da informação necessária pelos participantes coletivamente e individualmente.

Dados coletados sugerem que esta relação não é linear, mas uma na qual existem um ou mais tamanhos críticos, que se excedidos acarretam em mudanças comportamentais.

3.1.2.6 VI - Crescimento contínuo

"O conteúdo funcional de um programa deve ser continuamente aumentado para manter a satisfação do usuário durante seu tempo de vida".

Após o lançamento do software, mudanças serão necessárias para a contínua satisfação do usuário, estas mudanças podem ser correções de erros, adições de novas funcionalidades ou melhorias em funções pré-existentes. Muitas destas mudanças não foram planejadas pela equipe de desenvolvimento na época da primeira versão, ou foram causadas devido a alguma mudança no domínio operacional em que o software está inserido (podendo invalidar assim alguma suposições feitas anteriormente). Estas mudanças não planejadas inicialmente geram a necessidade de aplicações externas e módulos extras para o software, causando assim um inevitável aumento do conteúdo funcional deste programa.

3.1.2.7 VII - Qualidade decrescente

"Programas apresentarão qualidade decrescente a menos que sejam rigorosamente mantidos e adaptados às mudanças no ambiente operacional".

O fato de um software ser criado com um número de recursos e tempo limitados, somado ao fato deste estar inserido em um domínio suscetível a efeitos externos, causa uma certa imprevisibilidade a este software; Mesmo que este software funcione satisfatoriamente por muitos anos, isto não será um indicativo de que continuará funcionando a contento nos anos vindouros.

A sétima lei diz que esse nível de incerteza aumentará com o tempo, a não ser que seja feito um esforço para detectar e corrigir as causas desta incerteza. Este esforço evolutivo deve ser contínuo para todas as novas versões do software.

Esta lei também é conseqüência do fato de que conforme o tempo passa, a comunidade fica mais exigente com o software que usa e o critério de satisfação cresce. Produtos concorrentes surgem no mercado, novas tecnologias são criadas, novas funcionalidades passam a ser necessárias, logo o software que tinha uma qualidade satisfatória anos atrás não necessariamente terá a mesma qualidade anos depois.

3.1.2.8 VIII - Sistema de retorno

"Processos de programação de software constituem sistemas de *multi-loop, multi-level* e devem ser tratados como tais para serem modificados e melhorados com sucesso".

Esta lei só foi apresentada na década de 90, mas ela foi formulada a partir dos primeiros estudos nos anos 70. Estes primeiros estudos já mostravam que o sistema de evolução de um software do tipo E constitui um sistema complexo que é constantemente realimentado por retorno de seus usuários.

A vida de um software é um ciclo bem estabelecido de retorno positivo e negativo dos usuários entre cada versão do software. A longo prazo, a taxa de crescimento do sistema será estabelecida pela quantidade de retornos negativos e positivos, e controlado por fatores como quantidade de verba, número de usuários pedindo por uma nova funcionalidade ou reportando algum erro, interesses administrativos e tempo entre uma versão e outra.

Caso não seja possível evoluir um software, este estará fadado ao fracasso no futuro, já que o envelhecimento será inevitável e dentro de um certo período de tempo, o software em questão não mais será capaz de satisfazer as necessidades de seus usuários e será, portanto esquecido e substituído por outro mais atual. Estas leis nos ajudam a entender como ocorre o processo evolutivo de um software, e já que podemos considerar um software livre como um software em

um contínuo processo evolutivo, o estudo da evolução de software é um assunto bem pertinente quando apresentamos o tópico software livre.

3.2 - Evolução de software em software livre

Quando tentou-se examinar a evolução de software livres com base nestas leis, esperava-se que todas se aplicassem, já que o processo de evolução de um software livre normalmente é muito mais informal do que um processo estruturado de grandes empresa como as que desenvolveram os sistemas que ajudaram na formulação das leis, mas estudos como apresentados em [Godfrey00] com o Linux, mostram que nem sempre esse é o caso.

A partir dos dados obtidos com diversas versões do sistema, foram feitas medidas de crescimento do código do Linux durante essa pesquisa, tais como número de módulos, arquivos fonte, linhas de código, número de funções globais, variáveis e macros. Podemos observar alguns destes resultados nas figuras 3.1 e 3.2. A primeira mostra o crescimento em tamanho do arquivo do k*ernel* do Linux no formato compactado (*tar*), a segunda figura mostra o crescimento em número de linhas de código.

Como podemos observar em ambos os gráficos, o crescimento é superior a o de uma reta (crescimento linear), o que mostra que as versões estão crescendo de forma superlinear (acima do crescimento linear) com o tempo. Para confirmar este fato, [Godfrey00] também realizou medições de crescimento usando outros critérios, tais como número de arquivos fontes, número de funções globais, variáveis e macros, onde todas essas obtiveram resultados semelhantes aos gráficos mostrados.

Tais medições contrariam as métricas obtidas por Lehman em [Lehman98]. Neste artigo ele mostra que o crescimento evolutivo de um software diminui ao longo do tempo. Muito deste crescimento não esperado do *kernel* do Linux vem de adições de novas funcionalidades e suporte a novas arquiteturas, e não de simples correções de erros de versões anteriores. Um grande número destas

contribuições é devido à rápida popularização que o Linux vem sofrendo, e é comum que sejam feitas por empresas de grande porte como a IBM que faz uso do Linux em alguns de seus mainframes.

Também foi observado por Godfrey que o Linux não parece obedecer à terceira lei de Lehman, e que o esforço incremental gasto em cada versão não permanece constante durante a vida do sistema como diz a lei. Como conseqüência verifica-se a grande discrepância na variação de tamanho entre os arquivos fontes de uma versão para outra (variando de alguns bytes a alguns megabytes em cada versão) e o aparecimento de novos arquivos fontes adicionados ao software. Este comportamento pode ser observado na figura 3.3, que mostra o crescimento em linhas de código, dos subsistemas que compõe o Linux. Percebe-se que o subsistema responsável pelos *drivers* possuem um crescimento bem superior aos demais subsistemas, o que mostra que muito deste esforço incremental adicional vem do fato do surgimento de novos *drivers* para dar suporte a novos equipamentos.

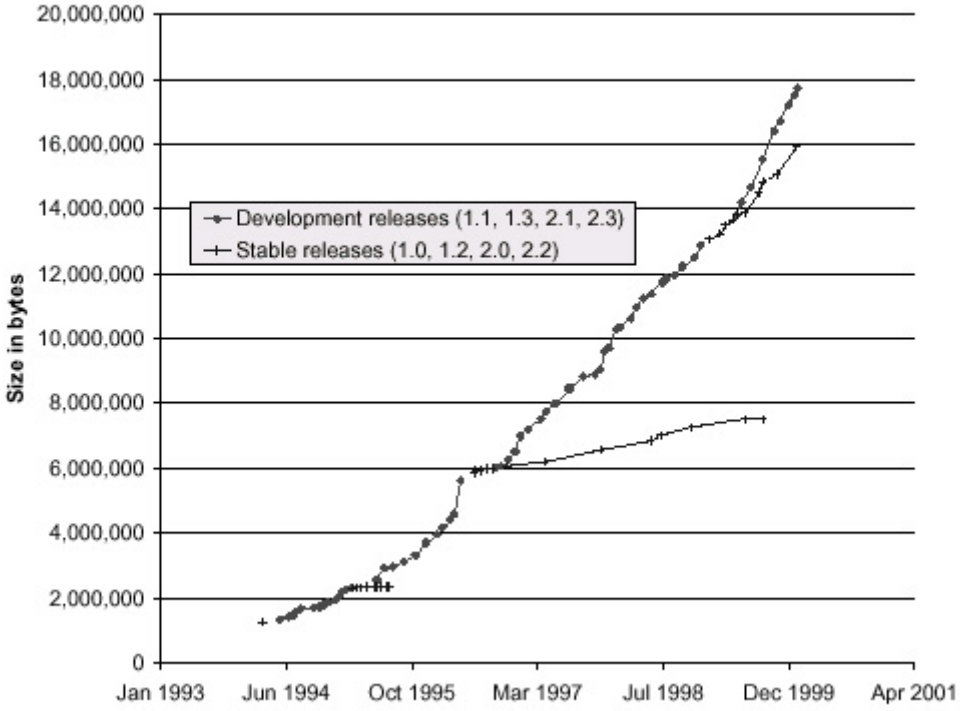

Figura 3.1 - Crescimento do arquivo *tar* com a versão completa do *kernel* do Linux segundo [Godfrey00].

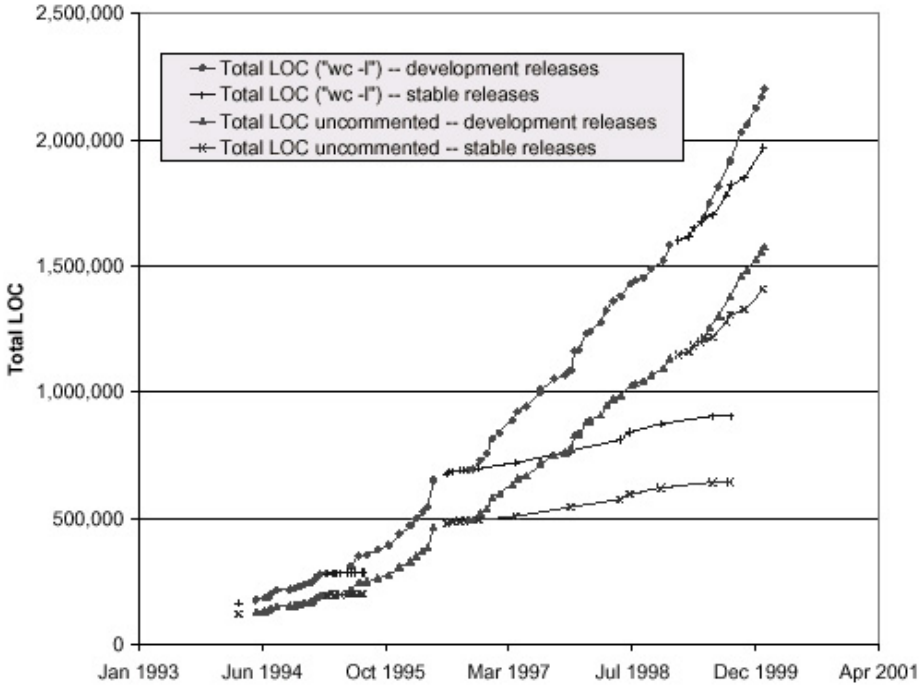

Figura 3.2 - Crescimento do número de linhas de código do Linux segundo [Godfrey00].

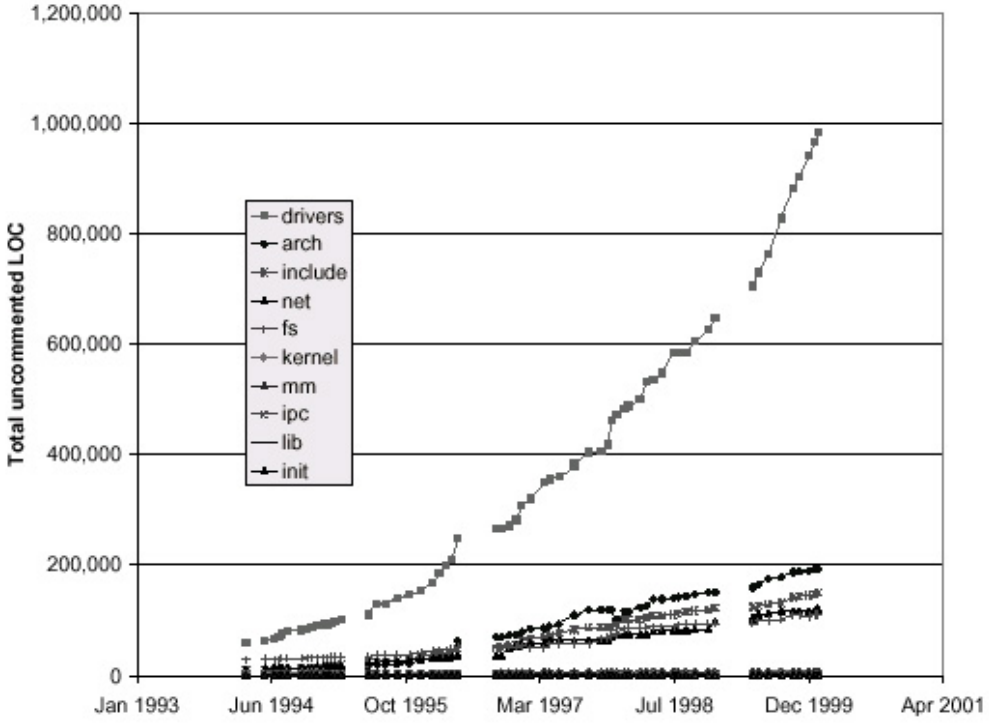

Figura 3.3 - Crescimento dos subsistemas do Linux segundo [Godfrey00].

Um estudo de caso sobre o desenvolvimento de um grande sistema de telecomunicações (o sistema em questão não é um software livre), apresentado em [Perry01], indica que caso o sistema possua um número muito grande de mudanças feitas paralelamente em cada versão do software, as interações podem se confundir com as atividades de manutenção, e dessa forma não são bem representadas pelas leis de Lehman. Esse estudo não contradiz as leis de Lehman, mas introduz um novo fator organizacional que deve ser levado em conta em futuras revisões da lei, mas mesmo assim tal estudo nos faz refletir sobre a aplicação das leis da evolução a um software livre, já que este faz uso intenso de desenvolvimento paralelo.

Alguns estudos parecem comprovar a idéia de que as leis necessitam de uma revisão para o caso de desenvolvimento de sistemas de software livres. Estes estudos apontam que o Linux não é o único software livre que não concorda com algumas das leis elaboradas por Lehman. Foi verificado que taxas de crescimento

entre as versões acima do previsto também ocorre em outros software livres, esse fato também foi observado em sistemas como o VIM, GNOME, Mono e o Debian GNU/Linux, mas isso não quer dizer de maneira alguma que todo software livre se comporta desta forma, já que outros estudos com os também software livres Fetchmail, X-Windows, Gcc e Pine comprovam que o nível de crescimento do código de uma versão para outra está dentro do previsto pelas leis de Lehman [Scacchi03].

[Scacchi03] levanta também a possibilidade de que as tecnologias e técnicas para desenvolvimento e manutenção de sistemas de software livre constituam um regime tecnológico distinto, e que este regime talvez não seja coberto adequadamente pelas atuais leis de evolução de software. Isto também pode ser verdade para tecnologias emergentes como sistemas de software baseados em componentes e aqueles compostos por arquiteturas dinâmicas.

É importante notar que todos estes estudos, por mais cuidadosos que sejam, não devem ser tidos como definitivos para definir se as leis da evolução se aplicam ou não aos software livres. Sistemas como o Linux, GNOME ou o Debian GNU/Linux são exemplos de software livres de grande porte e com uma grande base de usuários e desenvolvedores, mas não necessariamente representam a maioria dos software livres. Isso pode ser verificado em estudos recentes feito no portal de hospedagem de projetos de software livre Sourceforge [SourceForge03] e apresentados em [Robottom03]: 74% dos projetos pesquisados possuem de 1 a 5 membros e 32% do total de projetos pesquisados tinha apenas um membro, estes dados mostram uma realidade bem diferente de projetos de grande porte como o Linux. Entende-se por membros, pessoas que contribuem regularmente para o projeto (não necessariamente apenas com código fonte).

Apesar dos números significativos, deve-se lembrar que as leis de Lehman são elaboradas tendo em vista sistemas de software do tipo E, ou seja, sistemas que resolvem um problema ou implementam uma aplicação computacional no mundo real. O dado relativo à quantidade de projetos do tipo E pesquisadas não se encontra disponível em [Robottom03], mas como a maioria dos software livres se

encaixam neste perfil de se propor a resolver um problema no mundo real, acredita-se que o resultado final da pesquisa não deve ser muito diferente caso sejam excluídos sistemas que não sejam do tipo E.

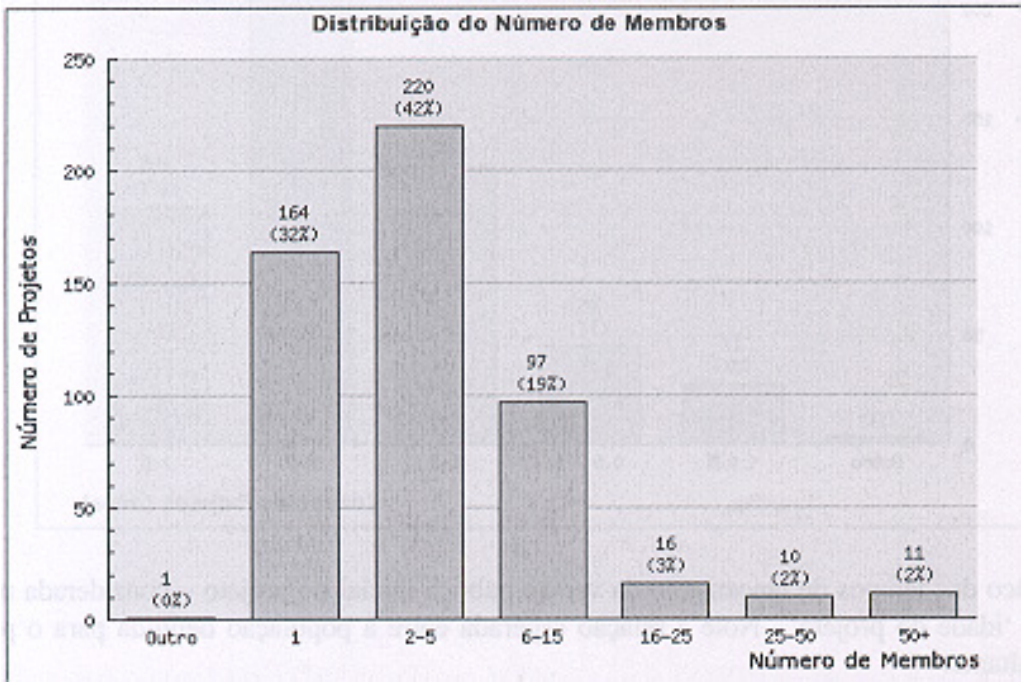

Figura 3.4 - Distribuição do número de membros em projetos de software livre segundo [Robottom03].

O tamanho médio das equipes dos projetos de software livre estudados a fundo (Linux, GNOME, Apache HTP Server e outros) é de mais de 50 pessoas, isso se deve a popularidade destes projetos na comunidade, esta popularidade foi muito provavelmente um dos motivos pelos quais os autores dos estudos de caso tiveram para escolhê-los como material de estudo. Todos os estudos feitos em sistemas de grande porte como estes são valiosas fontes de informação e conhecimento, mas deve-se ter cuidado para que não sejam tomados como regra para projetos de software livre em geral.

As leis da evolução de software são estudadas há 30 anos, e sua contribuição para a melhor compreensão da evolução de software e a engenharia de software como um todo são inestimáveis. No entanto devido ao avanço das técnicas, processos e práticas de desenvolvimento e manutenção de software nos últimos 10 anos, além do aumento de projetos de software livre no cenário mundial, verificou-se a partir de alguns estudos de casos que as leis da evolução de software necessitam de uma revisão profunda. Um dos fatos que talvez levem a esta revisão, é o de que os estudos que levaram as formulações das leis foram formuladas usando como base processos de desenvolvimento de sistemas baseados em sistemas centralizados e corporativos, usados para produzir sistemas com código fechado e de grande porte com poucos concorrentes no mercado e para uso de grandes corporações. Tais características em nada se assemelham a sistemas de software livre que são geralmente feitos e mantidos em sistemas descentralizados e coletivamente por uma comunidade que não tem que conviver com prazos apertados para o lançamento de versões, como é normalmente visto em sistemas corporativos [Scacchi03].