André Kenji Horie
Paulo Muggler Moreira
Ricardo Henrique Gracini Guiraldelli
Virgílio Vettorazzo

Proposta de Metodologia para Análise da Complexidade de Sistemas Computacionais em um Âmbito Global e Evolutivo

São Paulo

2008

André Kenji Horie
Paulo Muggler Moreira
Ricardo Henrique Gracini Guiraldelli
Virgílio Vettorazzo

Proposta de Metodologia para Análise da Complexidade de Sistemas Computacionais em um Âmbito Global e Evolutivo

Monografia para o curso PCS 2058 - Engenharia de Informação.

Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais

São Paulo

2008

Abstract

This work proposes a model for assessing complexity of computer systems in a global and evolutive scope. This will be achieved by defining relevant metrics and the way they relate among themselves, providing a quantifiable measure for complexity. Moreover, the proposed model will be applied to available data, thus creating a valid complexity curve whose tendency will be analysed.

Keywords: Computer Systems Complexity

Sumário

1 Introdução

1.1 Objetivo

Este trabalho tem por objetivo analisar a complexidade em sistemas computacionais em um âmbito global e evolutivo através da definição de uma metodologia, identificando-se métricas relevantes para o cálculo da curva de complexidade. Aplicando-se o modelo proposto, espera-se verificar a tendência da complexidade em função do tempo.

1.2 Conceitos Teóricos

Para um estudo sobre a complexidade de sistemas computacionais, vê-se a necessidade de se definir o conceito de complexidade, para então reduzir esta definição ao escopo de sistemas computacionais e, assim, poder considerar os aspectos relevantes para a criação do modelo.

1.2.1 Complexidade

Warren Weaver, em seu artigo "*Science and Complexity*" (WEAVER, 1948), introduziu o conceito de complexidade na litetura científica como o grau de dificuldade de se prever as propriedades de um sistema se as propriedades de cada parte for dada. Classifica-se então a complexidade sistêmica em organizada e desorganizada. Os sistemas de complexidade desorganizada caracterizam-se pelo número elevado de variáveis e pelo seu comportamento caótico, embora as propriedades do sistema como um todo possam ser entendidas utilizando-se métodos probabilísticos e estatísticos. A complexidade organizada, por outro lado, refere-se a interações entre as partes constituintes do sistema, sendo o comportamente deste redutíveis às interações, e não às propriedades das partes elementares. A visão proposta por este artigo influenciou fortemente o pensamento contemporâneo acerca da complexidade.

Empiricamente, observa-se que uma proporção alta dos sistemas complexos encontrados na natureza possuem uma estrutura hierárquica. Em teoria, espera-se que qualquer sistema

complexo seja hierárquico, tendo como a decomposição uma propriedade de sua dinâmica (SIMON, 1962). Esta simplifica tanto o estudo do comportamento como a descrição desses sistemas.

Em 1988, Seth Lloyd afirma que a complexidade de uma propriedade física de um objeto é função processo ou conjunto de processos responsáveis por sua criação (LLOYD, 1988). Em outras palavras, a complexidade é uma propriedade da evolução de um estado, e não do estado em si. Consequentemente, uma medida da complexidade deve classificar sistemas em estados aleatórios como de baixa complexidade, e quantificar a evolução deste sistema para seu estado final.

1.2.2 Organização de Sistemas Computacionais

O termo "arquitetura" é amplamente utilizado para se referir à estrutura na qual um sistema é organizado. Em Tecnologia da Informação, sistemas computacionais são representados pela arquitetura de hardware, de software, de rede e de informação (GENTLEMAN, 2005). Em relação ao escopo coberto por cada uma delas, observa-se que a arquitetura de hardware é essencialmente local, sendo assim definido para apenas um nó do sistema, enquanto as arquiteturas de rede e de informação requerem necessariamente a interação entre diversos componentes. A arquitetura de software, no entanto, pode tanto indicar a organização do software em apenas um componente como também em diversos componentes distribuídos, quando aplicável. Esta decomposição vê-se necessária para melhor endereçar a complexidade de cada vertente de um sistema computacional.

2 Definição do Modelo

2.1 Variáveis Relevantes

A seguir serão apresentadas as variáveis relevantes para o modelo de complexidade em termos globais.

2.1.1 Arquitetura de Hardware

Existem diversas abordagens para a medição da complexidade da arquitetura de hardware, todas elas baseando-se na complexidade como uma função do processo de construção dos componentes. É possível dizer, a fins de simplificação, que o crescimento de complexidade entre os principais componentes deve ser proporcional. Caso contrário, o gargalo de um computador seria facilmente observável em pouco tempo, o que de fato não ocorre.

Existem diversos valores representativos para o cálculo de complexidade relacionada à arquitetura de hardware. O primeiro deriva da Lei de Moore (MOORE, 1965), que afirma que o número de transistores em circuitos integrados aumenta exponencialmente, conforme ilustra a figura 1. Outros valores incluem métricas normalmente utilizadas para *benchmarks*, como por exemplo o número de instruções por segundo do processador, a capacidade de armazenamento das memórias de acesso aleatório, cache e do disco rígido. De qualquer forma, todos os estes valores devem supostamente respeitar a mesma tendência.

Desta forma, a complexidade de hardware assume a forma exponencial:

$$\mathbb{C}_{HW,local} = \alpha \cdot e^{\beta t} + \gamma \tag{2.1}$$

onde α, β e γ são fatores de ajuste.

Vale lembrar também que o escopo da arquitetura de hardware é essencialmente local. Para transpor este valor para um âmbito global, parece coerente assumir que há uma proporção direta entre o quanto a complexidade aumenta com o quanto os custos de produção totais

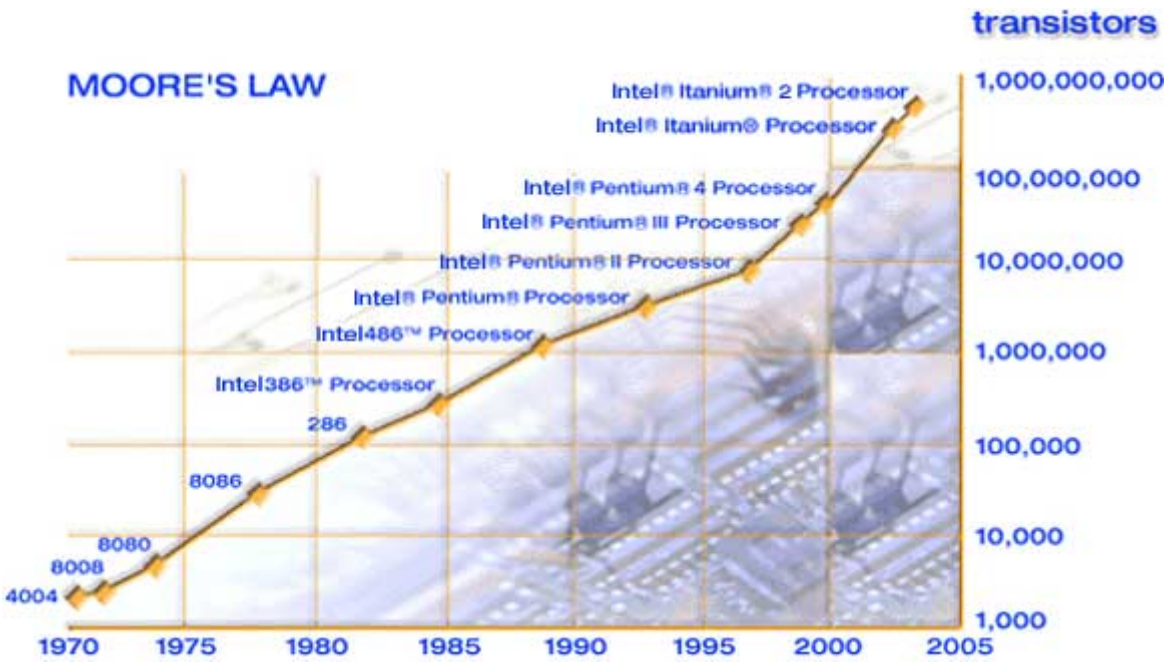

Figura 1: Lei de Moore

aumentam. Dado o custo total de produção como $C_T(Q) = C_F + C_V = C_F + Q \cdot C_{Mg}$, onde C_T é o custo total, C_F é o custo fixo, C_V é custo variável, Q é a quantidade produzida e C_{Mg} é o custo marginal. Portanto, a complexidade da arquitetura de hardware global é dada por $\mathbb{C}_{HW} = f(Q) \cdot \mathbb{C}_{HW,local}$, sendo $f(Q)$ um fator de aumento da complexidade em função da quantidade produzida. Este deve ser proporcional aos custos totais C_T e portanto, considerando-se um modelo simplificado, é linear.

2.1.2 Arquitetura de Software

(KEARNEY et al., 1986), (RANGANATHAN; CAMPBELL, 2007), (LEHMAN; RAMIL, 1998), (VASA; SCHNEIDER, 2003),(MCCABE; R, 1989).

Enquanto os aspectos computacionais de complexidade temporal e espacial de algoritmos têm sido estudados extensivamente, o aspecto humano da complexidade no desenvolvimento de software permanece relativamente imaturo. Este aspecto ainda não é bem compreendido, dificultando a obtenção de estimativas diversas, o que leva frequentemente a aumentos de prazo e/ou orçamento de projetos de software, ou até ao fracasso completo dos mesmos.

A complexidade de software pode ou não ser evitável. A complexidade evitável é aquela introduzida por fatores desnecessários ao processo de software porém frequentemente difíceis de se avaliar ou até mesmo detectar. Aspectos como inexperiênca dos desenvolvedores/projetistas, más práticas de projeto e desenvolvimento de software, eventos aleatórios

como falhas de hardware, todos contribuem desnecessariamente para o acréscimo de complexidade em um sistema. Este tipo de complexidade pode e deve sempre ser minimizado através de boas práticas de software.

A complexidade inevitável advém da complexidade inerente dos domínios de problemas resolvidos pelas arquiteturas de software, que vêm se tornando cada vez mais intrincados e interdependentes. O estudo deste tipo de complexidade nos permite avaliar as tendências para o crescimento da complexidade das arquiteturas de software.

É importante fazermos uma distinção entre complexidade do sistema e complexidade de uso do sistema. A complexidade que interessa para este estudo é a complexidade intrínseca, isto é, a complexidade geral do sistema, de seus componentes e da interação entre eles, entre outros aspectos. Um sistema de alta complexidade pode ser relativamente simples de se usar, escondendo os aspectos complexos de sua implementação e oferecendo uma interface simples para o seu uso.

(RANGANATHAN; CAMPBELL, 2007) identifica cinco aspectos da complexidade em sistemas computacionais. Em seu artigo, ele propõe meios de medir cada um destes aspectos e explicita como cada um dos aspectos afeta os envolvidos nos processos de software: desenvolvedores, administradores e usuários. Os aspectos de complexidade identificados por (RANGANATHAN; CAMPBELL, 2007) são:

- Complexidade estrutural;
- Imprevisibilidade;
- Complexidade de tamanho;
- Complexidade Caótica;
- Complexidade de Algoritmo;

A figura a seguir ilustra esta divisão:

2.1.2.1 Complexidade Estrutural

A complexidade estrutural avalia a dificuldade em se compreender ou realizar um procedimento específico. A tarefa é geralmente descrita por um grafo, que no caso de desenvolvedores de software pode representar a estrutura do programa, enquanto para outros usuários pode ser um fluxograma de processo ou similar. Uma métrica de complexidade utilizada para avaliar

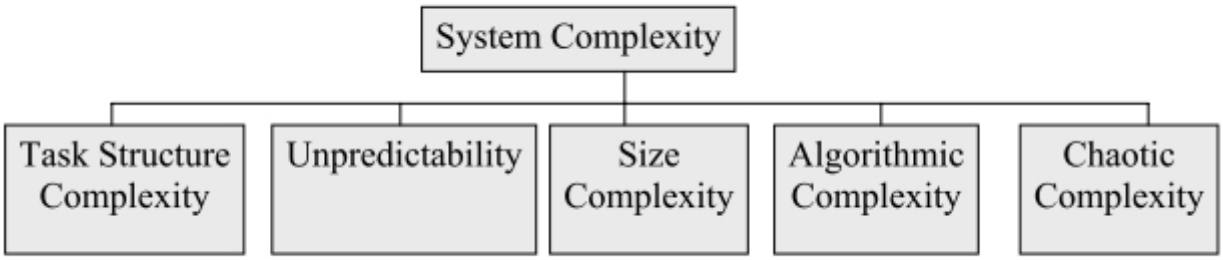

Figura 2: Aspectos de complexidade em uma arquitetura de software

a complexidade estrutural de um software é a complexidade ciclomática, ou complexidade condicional, introduzida por Thomas J. McCabe em 1976. Esta medida, como o nome sugere, é utilizada para determinar a complexidade de um programa estruturado (cíclico). É de conhecimento comum na área de desenvolvimento de software que o número de casos de teste necessários para exercitar um trecho de código é diretamente proporcional à sua árvore decisória. A complexidade ciclomática é uma estimativa do fator de ramificação desta árvore. A fórmula para a complexidade ciclomática de um grafo é:

$$CC = E - N + p, \text{ onde:}$$

E = número de arestas
N = número de vértices
p = número de componentes conectados

O termo 'p' é igual a 1 para um único processo, e pode ser maior que 1 quando múltiplos processos sejam executados concorrentemente para a realização de uma taefa. Esta fórmula tambe massume que um único estado final existe para o grafo, de forma que quaisquer ramificações no grafo levem eventualmente a este estado final. A medida de complexidade ciclomática permite avaliar o número de pontos de decisão em um programa. Um processo com um n´mero maior de pontos de decisão possui um nível de complexidade ciclomática maior do que um processo com uma sequência linear de ações. A figura 3 ilustra dois exemplos de grafos com diferentes níveis de complexidade ciclomática:

O grafo da esquerda possui uma complexidade ciclomática de $CC = 9 - 9 + 1 = 1$. O grafo da direita tem uma complexidade ciclomática de $CC = 24 - 20 + 1 = 5$. Para um desenvolvedor, o aumento da complexidade ciclomática significa maior esforço de desenvolvimento, teste e manutenção do software. A complexidade ciclomática é especialmente útil para

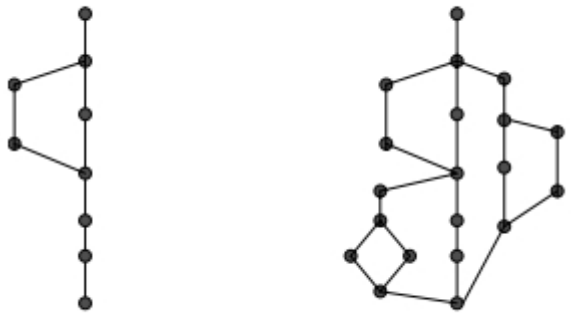

Figura 3: Exemplo de complexidade ciclomática em grafos

determinar o número de casos de teste necessários para um determinado programa ou trecho de código.

Para administradores e usuários finais, um aumento na complexidade ciclomática de uma tarefa implica maior complexidade cognitiva na compreensão e execução da mesma. As escolhas possíveis nem sempre estão claras, e as consequências de cada escolha ao longo do processo, podem estar menos claras ainda. Por exemplo, ao instalar um novo aplicativo, um usuário não terá grandes dificuldades se tudo o que tiver a fazer for clicar no botão 'Próximo' até o final da instalação. Por outro lado, o processo de instalação torna-se significativamente mais complexo quando existem diversas escolhas a serem feitas em cada etapa do processo e as consequências destas escolhas não estão claras no momento em que elas são feitas. Além disso, uma sequência linear de ações é facilmente automatizável, enquanto uma tarefa altamente ramificada, que exija interação e tomada de decisão em diversos pontos é mais difícil de ser automatizada. Portanto, o aumento do número de pontos de decisão, medido pela complexidade ciclomática do grafo, contribui para o aumento da complexidade de um sistema de software.

2.1.2.2 Imprevisibilidade

A imprevisibilidade fornece uma medida da dificuldade em se prever os efeitos de uma ação em um sistema. Um indicador importante do grau de previsibilidade de um sistema é a quantidade de entropia, ou não-determinismo, presente no mesmo. Quanto maior a entropia do sistema, mais difícil se torna prever o estado assumido por este após a execução de uma ação qualquer.

Uma boa medida para a entropia de um sistema é obtida utilizando-se a distribuição de porbabilidade dos estados possíveis do sistema. Se ao executar uma ação, um sistema passa

para um de k diferentes estados, com probabilidades $p1, p2, ..pk,$, então a entropia H do sistema é dada por:

$$H = \sum_{i=1}^{k} p_i \log_2 \left(\frac{1}{p_i}\right)$$

O termo $log_2 \left(\frac{1}{p_i}\right)$ é denominado o *fator surpresa*, que indica o grau em que se pode ser surpreendido por um resultado inesperado. Se a probabilidade de um dado evento é igual a 1, não há surpresa ao se deparar com seu resultado. Conforme esta probabilidade diminui, existe uma expectativa menor de se encontrar em um destes estados. Logo, um sistema que possa assumir um número limitado de estados, todos com probabilidade alta, causará poucos eventos imprevistos em seu funcionamento. Porém um sistema com alto número de estados raros terá uma entropia, ou imprevisibilidade, muito alta.

A imprevisibilidade de um sistema está frequentemente associada às ocndições dinâmicas do mesmo, falhas, e condições de corrida. Se uma ação executada por uma aplicação ou serviço for imprevisível, os testes e manutenção do sistema tornam-se mais difíceis. A automação de processos também pode introduzir aspectos de imprevisibilidade se for mal projetada e/ou implementada.

A imprevisibilidade em um sistema de software é sempre indesejada, e pode ser reduzida de duas maneiras: reduzindo-se o número de estado possíveis do sistema e/ou aumentando a probabilidade dos estados 'desejáveis'.

2.1.2.3 Complexidade de Tamanho

O tamanho de um sistema também pode ser utilizado como avaliação de sua complexidade. Existem diversas métricas para o tamanho de uma arquitetura de software, geralmente relacionadas ao seu código-fonte e/ou modelo estrutural. Algumas das métricas mais comumente utilizadas são: linhas de código, quantidade de métodos, quantidade de módulos, pontos de função, etc. Este tipo de complexidade é o mais comumente avaliado, dada a maior facilidade em se obter algumas das medidas citadas acima.

2.1.2.4 Complexidade Caótica

A complexidade caótica refere-se à propriedade dos sistemas de sofrerem grandes mudanças em seu comportamento geral devido a pequenas alterações em um certo componente do sistema. A complexidade caótica dificulta a compreensão dos sistemas, e resulta de uma falta de modularização do software e do excesso de inter-dependências entre componentes.

A complexidade caótica também se aplica aos domínios de problema, que podem requerer estruturas inerentemente compexas para sua modelagem e resoução.

Um exemplo de fator de complexidade caótica em sistemas de software é a instanciação de políticas, por exemplo políticas de controle de acesso. Estas políticas têm frequentemente o potencial de afetar o funcionamento de diversas partes do sistema (ao menos do ponto de vista do usuário), e a verificação de consistência de políticas é difícil de ser realizada em software, ficando geralmente a cargo do administrador. Logo, a introdução de uma política mal formulada no sistema pode causar acidentalmente a negação de acesso de uma determinada funcionalidade ou até mesmo do sistema todo para todos os usuários.

Um fator importante que contribui para a complexidade caótica de um sistema é o grau de acoplamento entre componentes. Os componentes podem ser acoplados de várias formas, e algumas destas são listadas em ordem crescente de complexidade:

1. Acoplamento de dados: passagem de parâmetros escalares ou vetoriais entre componentes
2. Acoplamento de controle: um componente passa valores que controlam a lógica interna de outro;
3. Acoplamento de dados comum: componentes que se referem aos mesmos dados globais;
4. Acoplamento de conteúdo: componentes que acessam e/ou alteram os estados internos uns dos outros.

A redução do acoplamento entre componentes é um aspecto desejável em sistemas de software, porém frequentemente inatingível. A redução no npumero de interconexões entre componentes reduzem a chance de que uma falha ou defeito em um componente cause uma falha geral no sistema ou em outros componentes. Além disso, a redução de complexidade caótica entre componentes melhora a reusabilidade destes. Por fim, a complexidade de compreensão do sistema também é reduzida, facilitando o trabalho de programadores e administradores do sistema.

Uma forma de medir o acoplamento entre componentes, e consequentemente um indicativo da complexidade caótica de um sistema, é a complexidade *fan-in fan-out* (KAFURA; HENRY, 1981). Esta métrica utiliza uma contagem dos fluxos de dados de entrada e saída em um componente, bem como o número de estruturas de dados globais atualizadas pelo mesmo. A complexidade *fan-in fan-out* é dada pela fórmula:

$$C = L * (Fan - in * Fan - out)^2,$$

onde:
C = Complexidade,
L = Comprimento, dado por uma medida de comprimento qualquer, como linhas de código, número de componentes, número de métodos, etc.,
$Fan - in$ = Contagem de fluxos de dados de entrada no componente,
$Fan - out$ = Contagem de fluxos de dados de saída do comonente.

O acoplamento é apenas uma das causas de complexidade caótica em um sistema, porém uma medida de acoplamento como a descrita acima pode servir como uma boa estimativa do grau de complexidade caótica do sistema em muitos casos.

Cabe aqui uma compração entre complexidade caótica e imprevisibilidade. A complexidade caótica é por definição determinística, enquanto a imprevisibilidade é essencialmente probabilística. Ou seja, dado que se possua conhecimentos suficientes sobre o funcionamento de um sistema, podemos dizer precisamente qual será o efeito de uma alteração em um componente independente do grau de caos inerente ao sistema. Já a imprevisibilidade advém de fatores que não podem ser previstos deterministicamente, como condições de corrida, falhas aleatórias, e etc.

É interessante notar que o grau de complexidade caótica e sua relação com a imprevisibilidade não são consequência só da estrutura do sistema, mas também do grau de conhecimento disponível sobre as interações do sistema e da escolha do modelo utilizado. A falta de conhecimento sobre as leis de interação de um sistema podem nos levar a crer que o seu comportamento é predominantemente probabilístico, aumentando sua imprevisibilidade. À medida que obtemos conhecimento sobre estas leis, o grau de imprevisibilidade do sistema diminuirá, porém a sua complexidade caótica poderá aumentar drasticamente. Além disso, frequentemente optamos por utilizar um modelo probabilístico mesmo que conheçamos as leis que regem o funcionamento de um sistema, simplesmente porque não é prático obter todos os dados ou programar todas as regras que regem um modelo determinístico.

2.1.2.5 Complexidade Algorítmica

A definição tradicional de complexidade algorítmica refere-se aos requisitos de tempo e espaço exigidos por algoritmos executados em máquinas de Turing. Um aspecto pouco explorado da complexidade de algoritmos diz respeito à sua complexidade cognitiva, isto é,

à dificuldade de compreensão do funcionamento de um algoritmo pelo desenvolvedor. Há frequentemente uma troca entre a performance e a complexidade cognitiva de um algoritmo.

Um conjunto de métricas interessante para a complexidade cognitiva de algoritmos, e que pode ser uma boa estimativa para projetos de software, são as métricas de Halstead (HALSTEAD, 1977). O uso destas métricas estima principalmente o esforço de programação, mas podem ser facilmente estendidas para medir a complexidade cognitiva de um algoritmo. Elas se baseiam em quatro variáveis escalares extarídas diretamente do código-fonte de um programa:

$n_1 =$ número de operadores distintos,
$n_2 =$ número de operandos distintos,
$N_1 =$ número total de operandos,
$N_2 =$ número total de operadores.

Estas variáveis permitem derivar várias métricas de complexidade:

Comprimento do programa: $N = N_1 + N_2$;
Vocabulário do programa: $n = n_1 + n_2$;

Volume do programa: $V = N * \log_2 n$. Esta métrica avalia a informação contida em um programa ou o tamanho da implementação de um algoritmo;

Dificuldade: $D = \frac{n_1}{2} * \frac{N_2}{n_2}$. A dificuldade de um programa também está associada à sua tendência para apresentar defeitos;

Esforço: $E = D * V$. O esforço de implementação ou compreensão de um sistema é proporcional ao volume ou grau de dificuldade deste.

Estas medidas de complexidade podem ser bastante relevantes para programadores e administradores para determinar o esforço de implementação de um determinado programa ou script. É importante notar que para utilizar estas medidas, os conceitos de operador e operando devem estar bem definidos para o sistema em uqestão. Por exemplo, um *loop* pode ser considerado como um único par operador-operando (iteração-lista), ou todas as variáveis envolvidas na iteração (e.g. o índice) podem ser consideradas. Para uma avaliação de esforço de desenvolvimento, esta escolha deve ser baseada principalmente no grau de experiência dos

desenvolvedores.

2.1.3 Arquitetura de Rede

Quando se fala em arquitetura de rede, é natural que se pense na *Internet*. Isso faz com que seja imprescindível uma análise da complexibilidade não de se usá-la, mas intrínseca à ela, ou seja, o quão complexa tal rede é e quais fatores alimentam tal condição. Para isso, basta analisar a evolução — desde sua criação — do número de usuários conectados.

Além disso, deve-se analisar o

2.1.4 Arquitetura de Informação

2.2 Modelo Matemático

.

3 Aplicação do Modelo

3.1 Dados Históricos

3.1.1 Arquitetura de Hardware

Através dos dados históricos relacionados à arquitetura de hardware, pretende-se verificar:

Processador	MIPS	Frequência (MHz)	Ano
4004	0,06	0,1	1971
8008	0,06	0,2	1972
8080	0,64	2,0	1974
8086	0,33	5,0	1978
80286	0,90	6,0	1982
386 DX	5,0	16,0	1985
486 DX	20,0	25,0	1989
80486 DX2	50,0	50,0	1992
Pentium	60,0	60,0	1993
Pentium Pro	200,0	200,0	1995
Pentium II	300,0	300,0	1997
Pentium II Xeon	400,0	400,0	1998
Pentium III	500,0	500,0	1999
Pentium III Xeon	550,0	550,0	1999
Mobile Pentium II Xeon	400,0	400,0	1999
Pentium 4	1.500	1.500	2000
Itanium	2.500	800	2001
Pentium 4 Northwood	10.000	3.200	2003
Pentium 4E Prescott	11.000	3.800	2004

Tabela 1: Evolução dos processadores da Intel

- Se está de fato correto assumir a proporção entre o crescimento de complexidade dos diversos componentes de hardware;
- Se a tendência de evolução da complexidade segue a Lei de Moore.

A partir de dados obtidos dos próprios processadores da Intel, observa-se o número de milhões de instruções por segundo e o de frequência do clock em função do ano de introdução do processador no mercado na tabela 1.

A partir destes dados, as figuras 4a e 5b são obtidas. Claramente, observa-se nelas a tendência exponencial conforme suposta anteriormente.

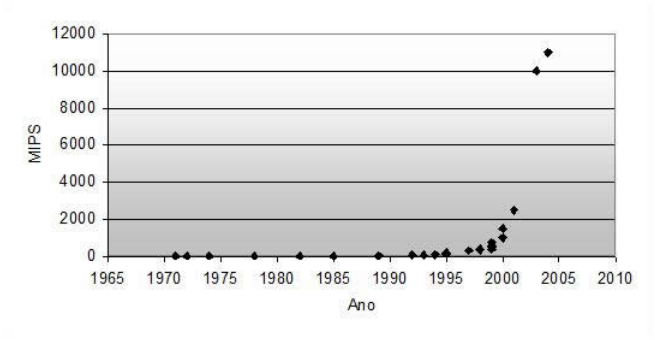

Figura 4: Tendência da evolução do MIPS para processadores Intel

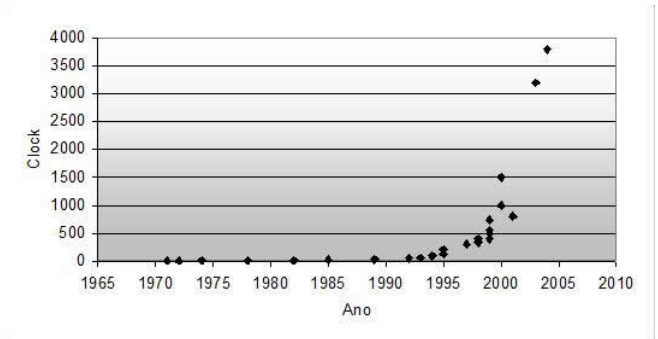

Figura 5: Tendência da evolução do clock para processadores Intel

Por fim, utilizando-se as capacidades de disco rígido e de memória de acesso aleatório como parâmetro, obtemos a mesma tendência (ver figuras 6 e 7).

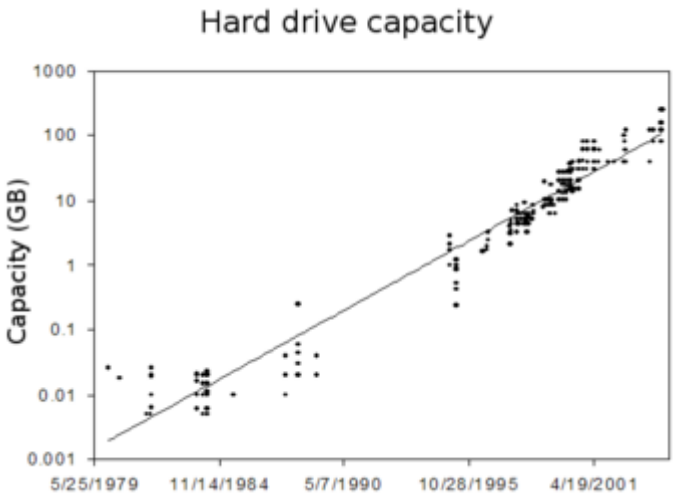

Figura 6: Tendência da evolução da capacidade de disco rígido

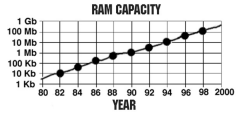

Figura 7: Tendência da evolução da capacidade de memória de acesso aleatório

3.1.2 Arquitetura de Software

.

3.1.3 Arquitetura de Rede

.

3.1.4 Arquitetura de Informação

3.2 Aplicação do Modelo de Complexidade

.

4 Conclusão

Com os dados obtidos, percebe-se as evoluções das complexidades relativas a cada uma das arquiteturas apresentadas são proporcionais, com tendência exponencial. Isso pode ser explicado pelo auto-ajuste que há entre elas: as novas necessidades observadas em uma delas causa novos requisitos para as outras, e evolução natural de uma delas permite novas oportunidades para as outras. Por exemplo, o aumento da capacidade dos hardwares é motivada por softwares que requerem cada vez mais processamento, e o aumento na complexidade do fluxo de informações implica maior complexidade da rede e dos softwares que proveem soluções. Da mesma forma, um aumento da capacidade de hardware permite que novas aplicações de software sejam criadas.

Contudo, deve-se notar que o período de tempo coberto pelos dados é relativamente curto, podendo não haver amostragem suficiente para que conclusões significativas sejam obtidas. É possível que o crescimento venha eventualmente a se estabilizar, embora essa tendência não seja observável com os dados atuais.

Referências

GENTLEMAN, W. M. *Dynamic Architecture*: Structuring for change. 2005.

HALSTEAD, M. Elements of software science, operating, and programming systems series. Elsevier, v. 7, 1977.

KAFURA, D.; HENRY, S. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, IEEE, v. 7, n. 5, p. 510–518, 1981.

KEARNEY, J. P. et al. Software complexity measurement. *Commun. ACM*, ACM, New York, NY, EUA, v. 29, n. 11, p. 1044–1050, 1986. ISSN 0001-0782.

LEHMAN, M. M.; RAMIL, J. F. Metrics and laws of software evolution - the nineties view. *Proceedings of the Fourth International IEEE Symposium, 1997*, IEEE, 1998.

LLOYD, S. *Black Holes, Demons and the Loss of Coherence*: How complex systems get information, and what they do with it. Tese (Doutorado em Fï¿½sica Teï¿½rica) — The Rockefeller University, Nova York, NY, EUA, 1988.

MCCABE, T. J.; R, C. W. B. Design complexity measurement and testing. *Communications of the ACM*, ACM, New York, NY, EUA, v. 32, n. 12, 1989.

MOORE, G. E. Cramming more components onto integrated circuits. *Electronics Magazine*, v. 38, n. 8, 1965.

RANGANATHAN, A.; CAMPBELL, R. H. What is the complexity of a distributed computing system? *Complex.*, John Wiley & Sons, Inc., New York, NY, USA, v. 12, n. 6, p. 37–45, 2007. ISSN 1076-2787.

SIMON, H. A. The architecture of complexity. *Proceedings of the American Philosophical Society*, v. 106, n. 6, p. 467–482, 1962.

VASA, R.; SCHNEIDER, J.-G. Evolution of cyclomatic complexity in object oriented software. *Proceedings of the 7TH WORKSHOP ON QUANTITATIVE APPROACHES IN OBJECT-ORIENTED SOFTWARE ENGINEERING*, 2003.

WEAVER, W. Science and complexity. *American Scientist*, v. 36, p. 536–544, 1948.