Revista de Ciências da Computação, 2014, nº9

Testes de *Software* na Redução do Consumo Energético dos Sistemas de Informação

Paulo José Matos [1], José Silva Coelho [2,3], Cristina Carapeto [3]

[1] Estudante Univ. Aberta, [2] INESC TEC, [3] Univ. Aberta-DCeT
Catalaopaulo@outlook.com, Jose.Coelho@uab.pt, Carapeto@uab.pt

Resumo

A mudança climática não pode ser desmentida. Desde que a humanidade adquiriu o conhecimento da eletricidade que tem transformado todo o seu modo de viver em seu redor. Para a produção de eletricidade recorre-se, em parte, à combustão de materiais que libertam CO_2 e que, pela quantidade emitida, potencialmente degrada o ambiente. Noutra perspetiva, em certas zonas geográficas, o acesso à energia elétrica é escasso. É neste contexto que surge a profissão de *Software Testing*. Neste artigo procura-se quantificar o contributo dos testes de *software* no desenvolvimento de aplicações que tenham em conta um consumo energético mais reduzido. Para atingir esse objetivo é proposta e aplicada uma metodologia para a medição de consumo e é definida uma fórmula matemática para apuramento da viabilidade económica dos testes. As consequências de um *software* que consuma menos energia serão de três ordens: a ambiental – na redução da emissão de CO_2; a humana – pela possibilidade de mais pessoas utilizarem a capacidade energética instalada; e a financeira – na redução direta do custo do consumo.

palavras-chave: Eficiência Energética, Mudança Climática, Testes de *Software*, Watt

Abstract

Climate change cannot be ignored any longer. Since humanity has developed the knowledge of electricity and has shaped his existence in a way that we are all dependent on it. The generation of electricity is linked to the combustion of products that release CO_2 into the atmosphere which has the potential to degrade the environment. On the other hand, some countries have a restricted access to electricity. It is in this context that a new profession emerges – the software testing. This article aims at quantifying the contribution of Software Testing in the energy consumption of software. To achieve this goal it is proposed and applied a simplified methodology for measuring consumption of software and it is defined a mathematical formula to calculate the economic viability of the tests. The consequences of an optimized software in its energy consumption will be: environmental – by reducing CO_2 emissions; human – by allowing more people to use the already installed power capacity; and financial – by reduction of economical expenditure.

keywords: Climate Change, Energy Efficiency, Software Testing, Watt

1. Introdução

O presente artigo procura ser um contributo para a promoção do aumento da eficiência energética do *software* e pretende analisar uma vertente onde os profissionais de testes podem contribuir para esse esforço.

Na secção 2 faz-se uma breve revisão sobre o tema "eficiência energética" na ótica do *software testing*. Esta parte subdivide-se em cinco componentes:

- Quais os contributos existentes ao nível do tema por parte de *testers*,
- Contributos da comunidade de sistemas de informação,
- Apresentação de uma ferramenta de simulação de consumo energético de *software* disponibilizada pela Microsoft® Research – o Joulemeter,
- Proposta de um modelo simplificado de Testes ao Consumo Energético do *Software* (TCES) e
- Demonstração da viabilidade económica para a realização de testes por via de uma fórmula matemática.

Na secção 3 são apresentados os resultados obtidos pelos ensaios realizados com base na metodologia proposta, incluindo-se aqui a aplicação direta da fórmula matemática. Finalmente, na secção 4, termina-se com as considerações finais.

2. Eficiência Energética nos Sistemas Computacionais

Ao longo da sua história, o Homem evoluiu muito mais do que apenas sob o ponto de vista biológico. De entre os muitos aspetos incluídos na evolução humana, pode mencionar-se a crescente utilização da tecnologia que gradualmente alterou o seu modo de vida, hábitos e costumes, formas de pensar e de agir. O domínio da ciência informática terá sido, até hoje, a última grande revolução na forma como encaramos a vida e como a moldamos. Tudo começou apenas no conceito de *hardware*. Este pequeno passo foi de tal modo decisivo que hoje já se acredita numa existência virtual comandada e intermediada por uma "entidade" intangível – o *software*.

2.1 *Software Testing* e Eficiência Energética do *Software*

A profissão de *Software Testing* é relativamente recente, contando com aproximadamente 5 décadas no mundo. Já a realização de testes de *software* para medir a eficiência energética não teve ainda contributos substanciais por parte destes profissionais com exceção de Brennan e Blatt, ambos no ano de 2008.

Brennan, na EuroSTAR Conference 2008, referiu que *Ineffective/Inefficient Testing = Resource Wastage / Global Impact* (Brennan, 2008). Porém as ideias apresentadas são essencialmente "formas de trabalhar", ou seja, boas práticas que devem existir nos projetos de *software* do ponto de vista macro, como, por exemplo, o facto de os ambientes de testes serem reutilizados. Blatt, por sua vez, destacou que os engenheiros/produtores de *hardware* ao terem implementado medidas de eficiência energética já conhecem os benefícios financeiros daí decorrentes, tendo mesmo uma métrica nesse sentido – *performance per watt* (Blatt, 2008). Blatt fez ainda a pergunta

essencial sobre a qual todos os engenheiros de *software* devem refletir: *highly efficient code will use less processing power, and therefore less energy; but is the extra development effort (and energy spent doing it) worth it?* (Blatt 2008).

2.1.1 O Impulso Alemão

Na Alemanha, em 2010, surgiu o projeto Green Software Engineering e a conferência anual Energy-Aware – High Performance Computing (EAHPC) que, desde então, têm sido os impulsionadores de muita investigação nesta área, dando origem a diversos artigos científicos sobre a eficiência energética do *software*.

O projeto Greensoft apresentou um modelo com o mesmo apelido (Tabela 1), em que Dick *et al.* (2010) olhando para o ciclo de vida do *software* globalmente, decompuseram-no em três momentos: o desenvolvimento, a utilização e o fim de vida.

Tabela 1 – Ciclo de vida do *software* proposto pelo modelo GreenSoft

	Desenvolvimento		Utilização	Fim de Vida	
	Desenvolvimento	**Distribuição**	**Utilização**	**Desativação**	**Alienação**
Efeitos de primeira ordem	• Viagens de negócios • Climatização do escritório • Iluminação do posto de trabalho • Condições de trabalho • …	• Embalagens • Manuais • Transporte • Tamanho do *download* • ….	• Consumo energético do *software* • Consumo de recursos pelo *software* • Requisitos de *hardware* • Acessibilidade • …	• Tamanho do *backup* • Tempo de conservação por imposição legal • Reconversão de dados para futura utilização • …	• Embalagens • Manuais • *Backups* • …
Efeitos de segunda ordem	• Teletrabalho • Distribuição de tarefas de desenvolvimento em rede • Motivação da equipa de desenvolvimento • …		• Desmaterialização • Logística • Métricas • …	• Interrupções de funcionamento • …	
Efeitos de terceira ordem	• Alteração dos métodos de desenvolvimento de *software* • Alteração do funcionamento das organizações • Alteração no estilo de vida • …		• Mudanças nos processos de negócio • Introdução de novas tecnologias mais eficientes • …	• Procura de novos produtos de *software* • …	

Esta abordagem holística do *software*, aqui decomposta nas suas vertentes de desenvolvimento (desenvolvimento e distribuição), utilização e fim de vida útil (desactivação e alienação), mostra a possibilidade dele ser sustentável, além de ultrapassar o debate específico do consumo energético *per si*. Porém, e numa adaptação da ISO 14756:1999, os autores propuseram pela primeira vez uma abordagem mais simples da medição do consumo do *software*, admitindo que ele não consome energia isoladamente. Segundo esta linha de raciocínio, dever-se-ia validar o consumo energético de todo o conjunto (*software* + *hardware*) e que denominaram como *System Under Test* (SUT). Aliás, esta interpretação – *para medir a influência do software no comportamento de um sistema de processamento de dados é necessário medir o*

comportamento do sistema no seu todo – está em conformidade com a definição da ISO/IEC 14756:1999 – *Measurement and Rating of Performance of Computer-based Software Systems* (Medição e Classificação da Performance de Sistemas de Software em Computadores) que define o normativo internacional de qualidade para efetuar medições a sistemas informáticos.

Fora do contexto dos profissionais de testes, aconteceu, em 2010, o primeiro EAHPC em que se destacou o trabalho de Minartz *et al.* que mediram o consumo de cada componente de computador para dois tipos de atividade: (i) atividade nula (0% utilização ou *idle*) e (ii) atividade com carga no sistema (ou stress), concluindo ser possível comparar diferentes programas em termos de eficiência energética. A partir dos dados obtidos, os autores acreditam ser possível extrapolar para uma avaliação em termos de benefícios económicos. Nos casos estudados, e verificadas diferentes estratégias, encontraram possíveis benefícios energéticos entre 10% a 32%.

Já em 2011, também na conferência EAHPC, Mämmelä *et al.* demonstraram que após terem programado um algoritmo (*Job Scheduler*) num conjunto de servidores, e tendo-o deixado correr por um período superior a 3 anos, se tinha verificado um aumento da eficiência energética nesses servidores em cerca de 39%.

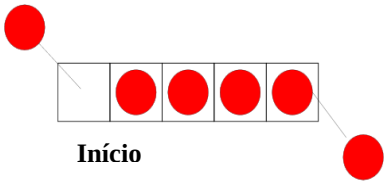

Figura 1 – Esquema de uma fila de "Job" (processos) (Mämmelä *et al.*, 2011)

2.1.2 Joulemeter – O Contributo Outsider de Ouro da Microsoft

O departamento da Microsoft Research deu a atenção que se impunha à eficiência energética do *software* e, em 2010, Kansal *et al.* disponibilizaram um *software* de simulação do gasto energético – o Joulemeter. Este *software*, para além de ser totalmente gratuito, apresenta características ímpares. Entre estas inclui-se a simulação do consumo energético de uma aplicação de *software* em concreto. Se esta simulação for aplicada a um dispositivo portátil, o nível de precisão é mais elevado do que quando aplicada a um computador inamovível, pois a calibração do Joulemeter é realizada com base no desgaste da energia existente na bateria.

2.2 Proposta de Metodologia de Testes ao Consumo Energético do *Software* (TCES)

Com o objetivo de massificar a realização de Testes ao Consumo Energético do *Software* (TCES) é apresentada uma metodologia faseada em seis (6) etapas:

1. Identificação das especificações do *System Under Test* (SUT)
   Nesta etapa são obtidas as especificações técnicas do *software* e *hardware*, tais como qual o sistema operativo, a arquitetura (32 ou 64 bits) e ou ainda o *thermal design power* (TDP). Estas informações são importantes para permitirem a possibilidade de replicação futura dos ensaios.

2. Identificação das variáveis referência
   Consideram-se variáveis de referência o CMaEG e o CMiEG (ver mais à frente designações), que nos indicam o consumo de energia máximo e mínimo, protagonizado por aquele SUT específico. Para calcular o CMaEG o SUT deve ser levado ao limite da sua capacidade de processamento, situação possível com recurso à execução dos programas do *software* HeavyLoad e/ou IntelBurnTestV2, e será o valor máximo medido no Joulemeter. No cálculo do CMiEG o computador deve estar ligado durante 5 minutos sem realizar qualquer operação e será o valor mais baixo apurado no Joulementer durante esse período.

3. Preparação do ambiente de testes
   Esta fase é importante pois é necessário reservar um período temporal para poder instalar os *softwares* adicionais de monitorização, como por exemplo o HWiNFO32 que irá medir a atividade de processamento em tempo real, ou ainda para o exercício de provas de conceito sobre a operacionalidade do SUT com os instrumentos de medição físicos ligados. Efetuando as provas de conceito é possível perceber se o ensaio de testes terá sucesso na medição, se será contaminado por erros ou se origina más medições.

4. Informações do caso de teste
   É importante perceber qual o objetivo do caso de teste e aquilo de que se necessita para que ele seja cumprido. Esta condição é imprescindível para que o *tester* possa encontrar a melhor forma de o executar interagindo o mínimo possível com o SUT. Por exemplo, se a funcionalidade a medir for o clicar num botão dentro de um *browser*, ou percorrer uma série de ecrãs, então poderá criar-se um *script* de automatização de ações no *browser*. Ou ainda, outro exemplo, se a funcionalidade que se pretende medir for a execução de um programa em MS-DOS poderá criar-se um *batch script*. Interagir o mínimo possível quando ocorre a medição é importante porque é difícil ao ser humano efetuar comportamentos repetíveis sem adicionar outros elementos (ou variáveis) à execução do caso de teste.

5. Execução do caso de teste
   Um caso de teste é uma operação funcional de negócio realizada via *software*, durante a qual se faz a medição do consumo energético do SUT. Se for utilizado o *software* medidor, como o Joulemeter, este deve ser iniciado antes do caso de teste decorrer, tal como deve ser iniciada a medição de um wattímetro ou pinça amperimétrica se for escolhido o método físico. A seguir deve-se iniciar o *software* de apoio HWinfo32 para recolha dos dados em tempo real do processamento do SUT. Para que o SUT não seja contaminado com o processamento do programa de monitorização HWinfo32 e/ou com o Joulemeter deve ser feita uma pausa de 120 segundos para estabilização do sistema. Depois inicia-se a execução do caso de teste até ao objetivo/ação ser cumprido. Resultam deste processo dois tipos de outputs: uma medição manual, fruto da observação do *tester* no medidor físico (ou um ficheiro de log com origem no Joulemeter) e um ficheiro de log (dados) do HWinfo32.

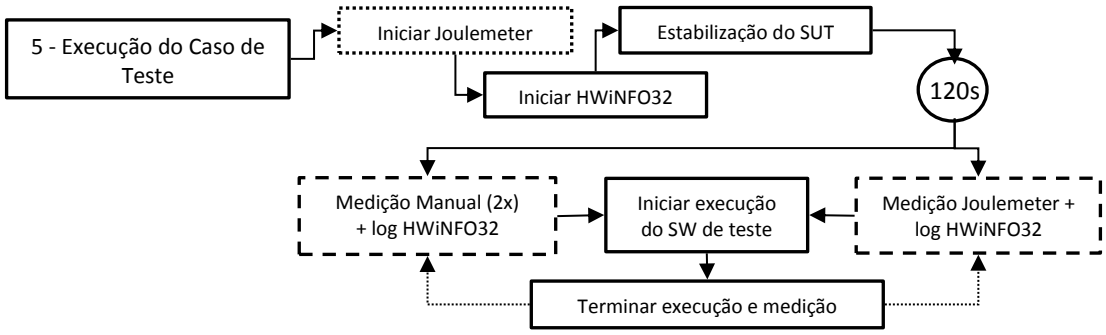

Figura 2 – Detalhe da etapa 5 da metodologia proposta no TCES

6. Resultados do caso de teste
   Terminada a execução do caso teste, os resultados dos diversos *logs* (incluindo o obtido manualmente) devem ser aglutinados e uniformizados numa única folha de cálculo. Entende-se por uniformização de dados, por exemplo, a normalização de datas-hora ou ainda efetuar arredondamentos a números inteiros, utilizando para isso as diversas fórmulas da folha de cálculo. No processo de aglutinação, se se encontrarem falhas no apuramento dos resultados é da competência do *tester* considerar se deve ou não haver uma repetição do passo anterior. Só com os dados uniformizados é possível tirar conclusões comparativas entre dois ensaios.

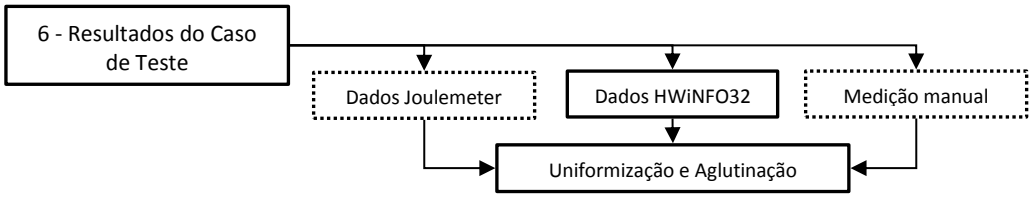

Figura 3 – Detalhe da etapa 6 da metodologia proposta no TCES

A proposta apresentada é menos formal, menos rígida e de mais fácil acesso do que, por exemplo, a preconizada na norma ISO 14756:1999, que assim tem impedido a massificação dos testes de *software* com o objetivo do aumento da eficiência energética. Por sua vez, os equipamentos físicos usados na medição são de preço acessível, bem como os *softwares* de apoio (que são livres de encargos ou bloqueios legais à sua utilização).

Considera-se terem sido incluídas todas as boas práticas para garantir a qualidade da operação (como por exemplo terem sido definidos tempos de estabilização do *System Under Test* (SUT), a vasta caraterização de todo o ensaio, permitindo assim a fácil reprodutibilidade por terceiros, entre outros).

A metodologia TCES foi flexibilizada pelo que não é obrigatória a existência de um medidor externo ao SUT (seja ele físico ou emulado), isto porque se considera que o processo de alimentação desse medidor, por via de um *software* cliente (dentro do SUT), acaba por ter um impacto equivalente a um *software* de recolha das variáveis em tempo real.

Ao nível das responsabilidades, estas são totalmente atribuídas ao *tester* em linha com a ISO 14756:1999, que indica que o *tester* é o condutor do processo de medição, ao ponto de este poder considerar uma medição válida ou descartá-la na totalidade, provocando a sua repetição.

De forma a balizar os consumos obtidos nos testes, a metodologia obriga a que seja feito o apuramento do Consumo Mínimo Energético Global (CMiEG) e do Consumo Máximo Energético Global (CMaEG), sendo que assim se saberão os limites possíveis do consumo energético do SUT.

2.3 Fórmulas do Ponto de Viabilidade Económica para Execução de Testes

São apresentadas duas fórmulas matemáticas para o cálculo da viabilidade da execução de testes de *software* no apuramento do consumo energético do mesmo. A diferença entre elas reside no facto de, na segunda, ao custo da energia ser somado o valor de aquisição de créditos de carbono.

1ª Fórmula custo do teste *vs* custo da energia:

$$h * x \leq y * z \qquad \text{Detalhando} \qquad h * \mathbf{x}_{(€/hr)} \leq \frac{(\mathbf{e1}_{(Ws)} - \mathbf{e2}_{(Ws)})}{3600 * 1000} * \mathbf{VAPD}_{(€/kWh)} * \mathbf{z}$$

2ª Fórmula custo do teste *vs* custo da energia *vs* custo da aquisição de créditos carbono:

$$h * x \leq y_2 * z \qquad \text{Detalhando} \qquad h * \mathbf{x}_{(€/hr} \leq \frac{(\mathbf{e1}_{(Ws)} - \mathbf{e2}_{(Ws)})}{3600 * 1000} * \left[\mathbf{VAPD}_{(€/kWh)} + \mathbf{VRPE}_{(tCO2/kWh)} * \mathbf{ACC}_{(€/tCO2)}\right] * \mathbf{z}$$

Legenda

h Número de horas de testes (hr)
x Custo da execução de testes, por hora (€/hr)
z Número de utilizações estimado (útil.)
y Consumo energético da aplicação acima do normal, por utilizações (€/útil)
y_2 Consumo energético da aplicação acima do normal deduzindo a poupança por não aquisição de crédito de carbono, por uma hora (€/hr)
e_n Consumo Energético de uma aplicação ou funcionalidade em teste (Ws)
VAPD Valor de Aquisição a um Produtor de Eletricidade (€/kWh)
VRPE Valor referência do produtor de eletricidade das toneladas de CO_2 libertadas por kWh produzido (tCO_2/kWh)
ACC Aquisição de Créditos de Carbono (€/tCO_2)

Nas fórmulas acima apresentadas existem três variáveis que serão mais voláteis, que as restantes, pois podem ser alteradas por motivos exógenos (por exemplo condições da oferta e procura do mercado de trabalho) e num curto espaço de tempo. São consideradas variáveis voláteis: o (i) Custo de execução de testes por hora (ii) Consumo energético da aplicação acima do normal e (iii) Número de utilizações. As restantes variáveis são mais robustas, pois permanecem mais constantes durante o tempo, como é o caso do custo de eletricidade junto do produtor.

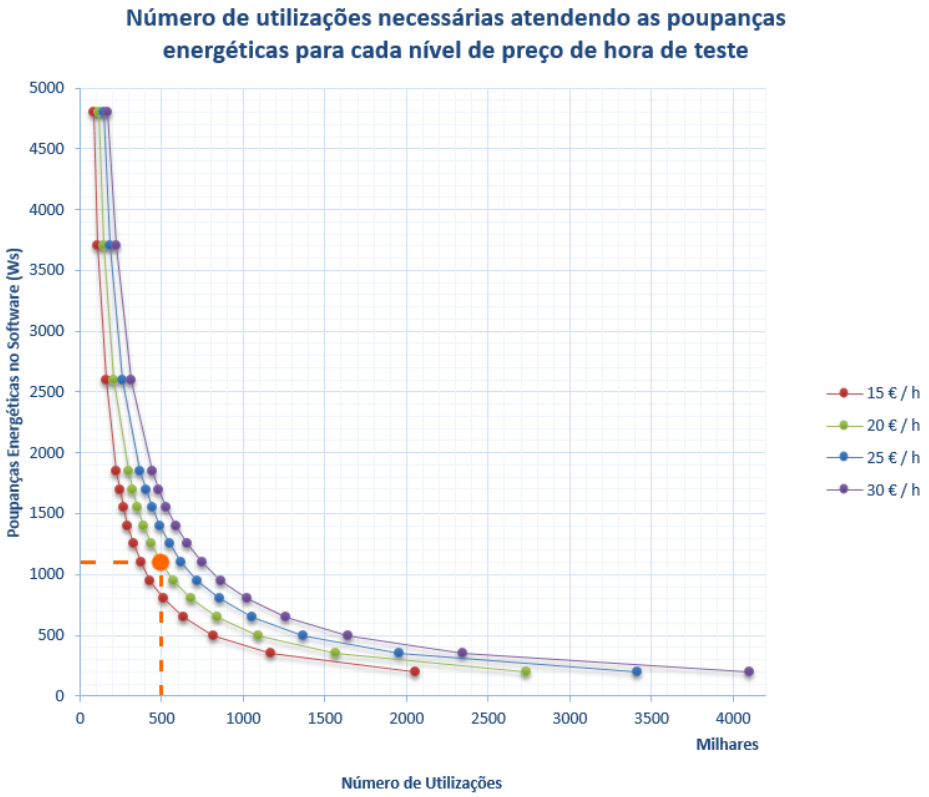

Gráfico 1 – Aplicação da fórmula "custo de teste *vs* custo da energia" na relação entre três variáveis: Preço Hora de Testes de *Software* / Ganhos de Eficiência Energética por Funcionalidade / Número de Utilizações Necessário para Rentabilização

O gráfico 1 permite aos *stakeholders* avaliarem, de uma forma visual, se valerá ou não a pena investirem na aquisição de testes de *software*. Tendo em consideração que 1 hora de testes de *software* custa cerca de 20€ e que esses testes permitem uma redução do consumo em 1100 Ws (prevendo-se uma utilização em cerca das 500.000 vezes no contexto da vida útil do *software*), não deverá ser difícil tomar a decisão mais vantajosa.

Poderão surgir críticas a este modelo matemático, pois apenas está a considerar o custo/hora do *tester* e não o custo do *refactoring* do código fonte por parte do programador. Esta assunção tem por base a realidade portuguesa, onde a presença do *tester* nos projetos de *software* ainda é entendida como facultativa, ainda que sempre desejável. Assim, a fórmula demonstra que o *tester* é necessário para a realização deste tipo de testes (orientados ao consumo energético) e que essa relação custo/benefício pode ser quantificada. Outra crítica aqui assumida é que o potencial de melhorias ao nível do consumo energético poderá ficar aquém do estimado, pelo que o consumo energético em excesso deve ser considerado uma perda potencial e não efetiva. Ainda que a crítica seja aceite, todos os estudos encontrados na bibliografia vão no sentido oposto, isto é, os algoritmos testados confirmaram sempre a possibilidade de redução de consumo energético (Mämmelä et al, 2011).

3. Resultados

Aplicou-se a metodologia de Testes ao Consumo Energético de Software proposta em vários casos de teste e em dois SUT com características diferentes: desktop clássico *versus* portátil.

3.1 Eficiência Energética no *Software*

Os *softwares* de apoio Joulemeter e HWiNFO32 mediram em tempo real, 35 ou 55 parâmetros, consoante o SUT. Após a análise de cada um dos parâmetros, atribuiu-se importância à sua medição no estudo do consumo energético aos seguintes: **a)** Tempo do Caso de Teste, **b)** Consumo Energético da Globalidade do SUT, **c)** Consumo Energético do Processador, **d)** Consumo Energético da Aplicação em Teste, **e)** Nível de Processamento Alcançado, **f)** Nível de Transações em Disco Rígido e **h)** Nível de Processamento da Placa Gráfica.

A análise dos resultados permite verificar que o Caso de Teste 1A está contaminado por pequenas falhas que, todavia, não impediram o sucesso do objetivo proposto. Este Caso de Teste 1A parece ter provocado uma distribuição das necessidades de processamento um tanto confusa (Gráfico 2). Da observação do gráfico podemos ver que, ao segundo 10, o "Core#1 Thread #1" quase atingiu 50% de utilização enquanto os restantes 3 Cores desceram. Contudo, estes últimos quase que imediatamente voltaram a subir, enquanto o primeiro desceu até ao zero de processamento. Também se pode observar que as amplitudes de processamento são na casa dos 55% ("core#1 Thread #0"). Se analisarmos o Caso de Teste 1B verificamos que as amplitudes são bastante menores atingindo no máximo 30% (Gráfico 3).

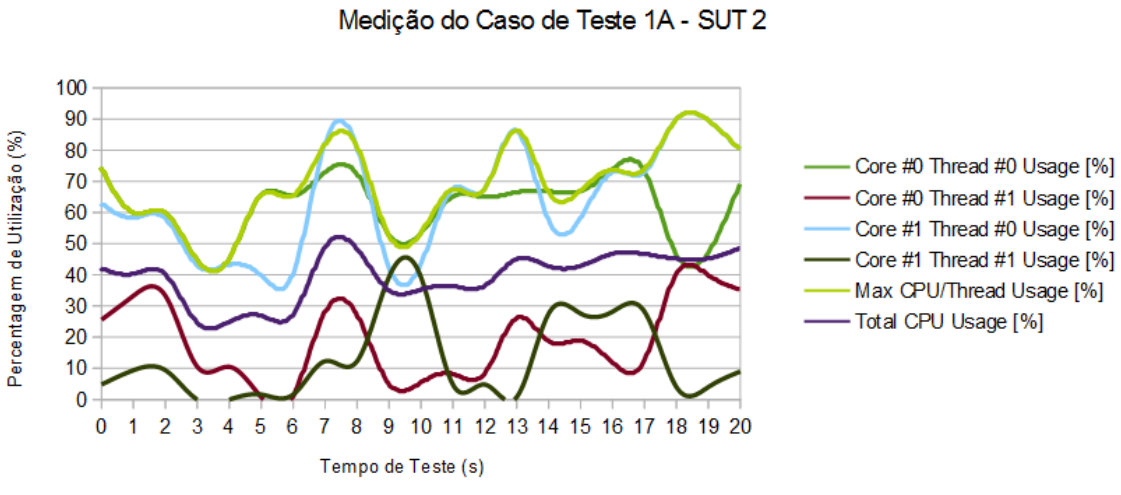

Gráfico 2 – Nível de processamento do SUT 2 no caso de teste 1 A

O gráfico 2 mostra amplitudes de processamento na casa dos 55% (exemplo o "core#1 Thread #0" que no segundo 6 estava com 35% e depois, no segundo 7, com 90%).

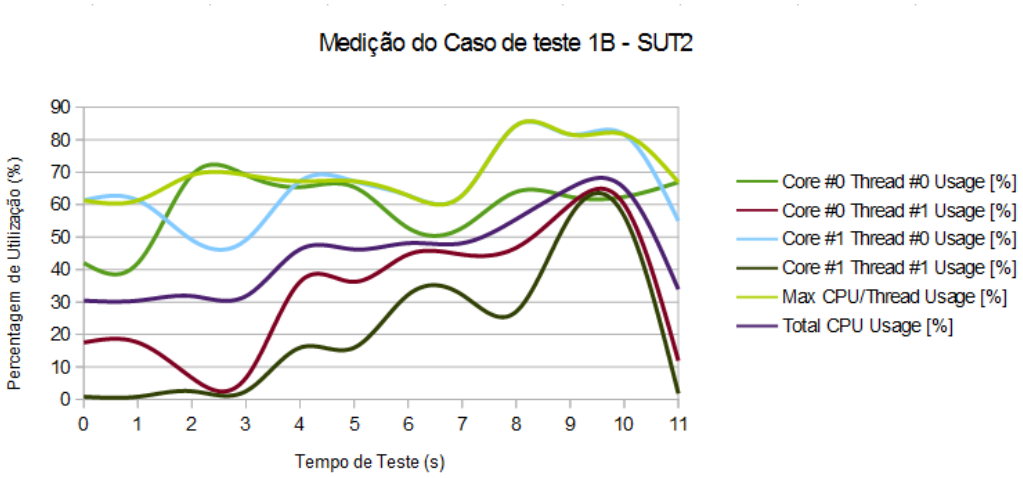

Gráfico 3 – Nível de processamento do SUT 2 no caso de teste 1 B

O gráfico 3 mostra amplitudes de processamento na casa dos 30% (exemplo o "Core #0 Thread #1" no segundo 3 estava nos 10% e no segundo 4 estava nos 40%).

Esta análise permitiu entender que, se um *software* for mais constante ao nível das requisições de capacidade de processamento, poderá este parâmetro ser um indício de um melhor desempenho energético. Neste caso concreto, tal situação fica demonstrada ao verificar-se que o Caso de Teste 1A teve um consumo 334 Ws e o Caso de Teste 1B teve um consumo de apenas 195Ws.

Outro aspeto que diferencia os casos teste 1A e 1B, medidos no SUT 2, foi o tempo despendido na tarefa. Para completar o Caso de Teste 1A demorou-se mais 9 segundos que para completar o Caso de Teste 1B. Estes 9 segundos adicionais representam um incremento de pelo menos 80%. Esta diferença de tempo por si só poderá não ser muito importante se a funcionalidade em teste não necessitar de ser calculada em tempo real. Assim se, teoricamente, o CT1A tivesse durado mais tempo mas consumido menos energia, no total do apuramento ele seria mais eficiente (ver Gráfico 4).

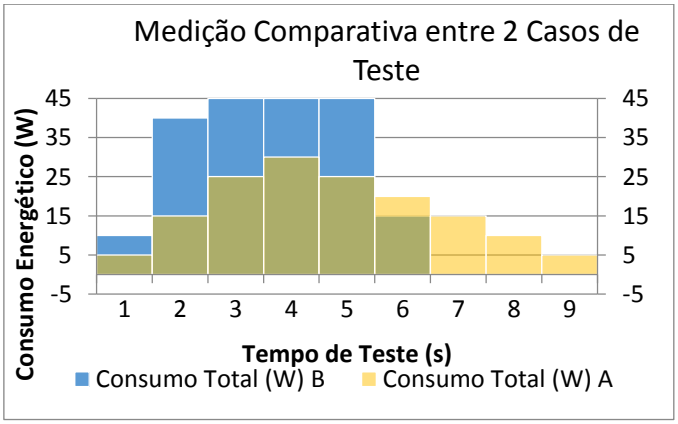

Gráfico 4 – Comparativo teórico entre duas aplicações em que o CT B consumiu 200 Ws e o CT A consumiu 150 Ws.

Conclui-se que, para uma análise de eficiência energética de um *software*, são de recolha obrigatória os parâmetros Tempo do Caso de Teste e Consumo Energético da Globalidade do SUT.

3.2 Análise dos Resultados dos Ensaios Realizados segundo a TCES

Os resultados alcançados, numa comparação intra SUT, demostram que em todos os ensaios realizados há possibilidade de implementar melhorias nos programas de *software* (Tabela 2).

Tabela 2 – Comparação entre os totais mínimos e máximos do consumo energético nas SUT 1 e 2 para cada caso de teste

		Consumo Energético Total			
SUT	Caso de Teste	Min (Ws)	Max (Ws)	Δ (Ws)	Δ %
1	**1**	1878	8278	6400	341%
2		195	334	139	71%
1	**2**	2127	13100	10973	516%
2		149	532	383	257%
1	**3**	1555	52462	50907	3274%
2		258	9712	9454	3664%
1	**4**	355	4134	3779	1065%
2		49	748	699	1427%

No caso de teste 1 e 2, pelo facto do ensaio utilizar o mesmo *software* em versões diferentes, em que há sempre uma versão que apresenta falhas não impeditivas, poderá este cenário traduzir as variações de consumo energético que mais facilmente irão surgir no mundo real. Ainda assim, encontramos variações que vão desde 71% a 516% e que representam, respetivamente, uma variação de poupança de 139 Ws a 10973 Ws.

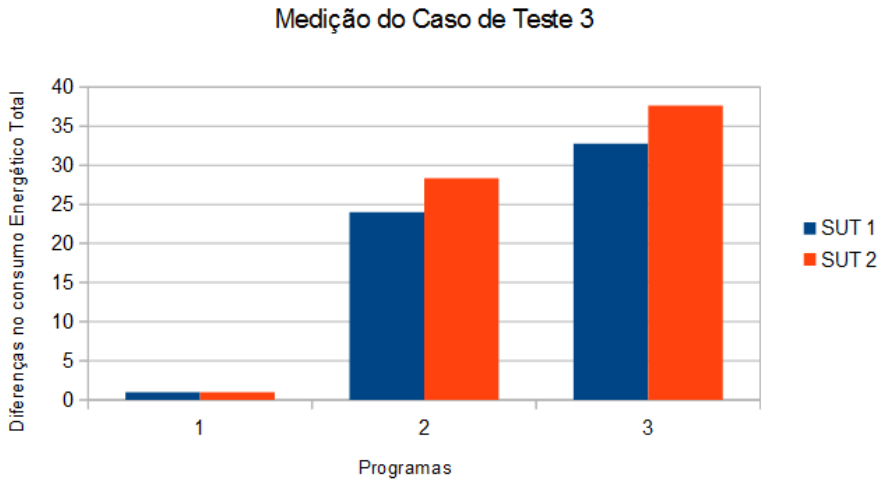

Gráfico 5 – Avaliação do Caso de Teste 3, em que o caso A, por ser o mais baixo, é usado como referência para o B e C

Noutra perspetiva, e avaliando os resultados do caso de teste 3 (Gráfico 5), em que se comparou o consumo energético de três programas construídos de forma autónoma entre si mas que visavam encontrar a mesma solução, ou seja, foram programados por pessoas diferentes mas orientadas para que o programa tivesse a mesma funcionalidade,

verificou-se que, entre o programa que consome menos e o que consome mais, a variação é muito elevada (em cerca de mais de 35 vezes para o pior caso), e isto independente do SUT onde se realizou o teste. Em termos absolutos, a pior diferença foi encontrada no SUT 1 no caso de teste 3C com uma diferença de 50907Ws. Porém, neste SUT 1 todos os testes executados gastaram mais que no SUT 2 (em média 1140%[1]). Em termos relativos, foi no SUT 2 que se gastou mais energia face ao programa de *software* considerado como padrão no processamento de dados, o que quer dizer que apesar do SUT 2 apresentar sempre um consumo inferior face ao SUT 1, o *software* estava tão pouco otimizado que foi efetivamente necessário consumir mais 3664% para que o programa conseguisse devolver um resultado válido.

Por fim, no caso de teste 4 (ver Tabela 2), a diferença do consumo situou-se entre os 1065% e os 1427% a mais, respetivamente para o SUT 2 e SUT 1. Acontece que, quando a situação de erro não devolve um resultado válido ou não devolve sequer nenhum resultado, sendo que o processamento e energia gasta foi para "o vazio", o custo é sempre considerado alto, independentemente do valor (absoluto) obtido.

Com os resultados acima apurados podem-se retirar 3 conclusões: a) o consumo energético de um *software* é muito influenciado pela forma como o código fonte foi construído/desenvolvido; b) num *software* pequenos erros provocam um aumento no consumo energético mas o seu impacto é menor que num *software* que não tenha o seu código fonte otimizado; c) um *software* que apresente erros que levem ao colapso do programa, ou que não devolvam resultados válidos, apresentam um duplo problema – não só consumem mais energia como esse consumo gasto no processamento é em vão.

3.3 Viabilidade Económica para Realização de Testes

Face aos resultados apresentados e utilizando o pior cenário[2] encontrado, isto é, onde existe um valor de diferença de 3664% (9712 Watts-segundo), fica demonstrado que compensaria investir 25€ em testes de *software* num cenário em que se estimasse, a título de exemplo, a utilização do programa por 1446 utilizadores e em que cada um deles usasse a funcionalidade 50 vezes.

Aplicamos a fórmula "custo de teste *vs* custo da energia" para no cálculo deste caso exemplo, que contempla a diferença de 9712 Ws :

$$1\text{h de testes} * 25\,€ \leq \frac{9712\ \text{Ws}}{3600 * 1000^{3}} * 0{,}1317\ €/\text{kWh} * (50\ \text{utilizações} * 1446\ \text{utilizadores})$$

$$25\,€ \leq 25{,}68\,€$$

[1] (2478% + 963% + 2462% + 1428% + 603% + 511% + 540% + 724% + 553%) / 9 Casos de teste = 1140%

[2] SUT 2 – caso de teste 3C

[3] Destina-se à normalização de unidades, de Ws para kWh, uma vez que o custo de energia é em kWh, enquanto a diferença de energia está em Ws

4. Conclusões

Este artigo demonstra a viabilidade económica da contribuição dos profissionais de testes para a construção de um *software* mais eficiente do ponto de vista energético, bastando para isso a conjugação das seguintes três variáveis: a) Preço Hora de Testes, b) Número de Utilizações Previsto do *Software* e c) Redução de Consumo Estimada.

Pese embora a ausência de estudos sobre o tema da eficiência energética do ponto de vista do *software testing*, as informações recolhidas permitiram enquadrar um conjunto de boas práticas que foram traduzidas numa metodologia de medição do consumo energético do *software*. A metodologia, por ser mais simples que as existentes, poderá ser mais facilmente massificada por estes profissionais, pelo que terá um impacto direto na indústria beneficiando todos. Do ponto de vista de negócio, a metodologia sugerida poderá abarcar todos os sectores da sociedade, em particular os *softwares* de grande consumo, assim como os distribuídos na internet (ou que funcionam como SaaS – *Software as Service*). Pode ser ainda aconselhável a utilização desta metodologia nos *softwares* que necessitem de ser auditados pela norma 14756:1999, funcionando como uma pré-inspeção. Do ponto de vista ambiental, uma vez que há uma menor emissão de CO_2, os resultados também se apresentam favoráveis. Por outro lado ainda, utilizando a energia de uma forma mais eficiente, evitando desperdícios que, como se viu são facilmente evitados, ela pode ser disponibilizada para outras funções.

De sublinhar o facto de este assunto ainda estar pouco estudado. A bibliografia disponível não é rica em experiências deste tipo, o que torna este estudo bastante inovador.

O trabalho aqui apresentado pode ter uma importância relevante. Por exemplo, o caso de teste 2 (ver tabela 2), em que se comparou a visualização de um vídeo no *Youtube* num *browser* sem problemas e num *browser* com problemas, demonstrou a existência de uma diferença de 10973Ws. Imaginando que o problema pode afetar todos os mil milhões de utilizadores mensais[4], e se 1 hora de testes fosse suficiente para identificar o problema, então o potencial ganho da aplicação dos testes de *software* para benefício coletivo seria muito grande: 16 057 157%. Aplicando a fórmula "custo de teste versus custo da energia" resulta que: 25 € < 401 428 916,7 €

Assim, cada projeto de *software* deve ser analisado caso a caso e em particular as dimensões que as principais três variáveis "voláteis" podem assumir: 1) custo dos testes, 2) % de poupanças no consumo, e 3) número de utilizações previsto.

[4]Fonte: https://www.youtube.com/yt/press/statistics.html

Bibliografia

1. Blatt, Michael (2008), "Green Computing and Performance Testing: A New Paradigm?", SQS Performance Testing Competency Centre, London, United Kingdom [Consultado em Outubro 2013]. Disponível em URL: http://www.sqs.com/en/se/Download/SQS_RI_Green_Computing.pdf

2. Brennan, John (2008), "Red Hot Testing in a Green World", AppLabs, The Hague, Netherlands, [Consultado a 27 Junho 2014]. Disponível em URL:http://www.slideshare.net/jsb1976/AppUK-Red-Hot-Testing-in-a-Green-World-V106-Animation

3. Dick, Markus; Naumann, Stefan; Kuhn, Nobert (2010) "A Model and Selected Instances of Green and Sustainable Software", Trier University of Applied Sciences Environmental Campus Birkenfeld – Institute for Software Systems, Germany.

4. GreenSoft – Green Software Engineering (2014) "Project – Green Software Engineering" [Consultado a 28 Junho 2014]. Trier University of Applied Sciences Environmental Campus Birkenfeld – Institute for Software Systems, Germany, Disponível em URL: http://www.green-software-engineering.de/en/project.html

5. ISO/IEC 14756: (1999), "Measurement and Rating of Performance of Computer-based Software Systems".

6. Kansal, Aman; Goraczko, Michel; Liu, Jie; Zhao, Feng (2010) "Joulemeter: Computational Energy Measurement and Optimization", Microsoft Research, Redmond, United States [Consultado a 1 Julho 2014]. Disponível em URL: http://research.microsoft.com/en-us/projects/joulemeter

7. Mämmelä, Olli, Majanen, Mikko, Basmadjian, Robert, De Meer, Hermann, Giesler, André, Homberg, Willi (2011) "Energy-aware job scheduler for high-performance computing" EnA-HPC Conference 2011, Hamburg, Germany, [Consultado a 30 Junho 2014]. Disponível em URL: http://www.ena-hpc.org/2011/talks/mammela-slides.pdf e URL: http://www.cs.ucc.ie/~gprovan/CS6404/2012/energy-aware-HPC-sim2011.pdf

8. Minartz, Timo, Kunkel, Julian, Ludwig, Thomas (2010) "Simulation of Power Consumption of Energy Efficient Cluster Hardware", EnA-HPC Conference 2010, Hamburg, Germany [Consultado a 29 Junho 2014]. Disponível em URL:http://www.ena-hpc.org/2010/talks/EnA-HPC2010-Minartz-Simulation_of_Power_Consumption_of_Energy_Efficient_Cluster_Hardware.pdf

Paulo José Estrela Vitoriano de Matos licenciou-se em 2007 pela Escola Superior de Tecnologia e Gestão de Portalegre em Gestão Empresarial e na atualidade é estudante do mestrado Cidadania Ambiental e Participação na Universidade Aberta. Profissionalmente, em 2008, escolheu seguir a carreira de Software Tester, tendo alcançado em 2014 o segundo lugar num concurso internacional de testers organizado pela Belgium Testing Days. Tem como interesses a Politica, o Ambiente e o Software Testing.

José Pedro Fernandes da Silva Coelho é Professor Auxiliar na Universidade Aberta desde 2004, no Departamento de Ciências e Tecnologia. Publicou 12 artigos em revistas internacionais e mais de 35 recursos de natureza variada, no repositório aberto. Nas suas atividades profissionais interagiu com 36 colaboradores em co-autorias de trabalhos científicos. Os seus principais interesses de investigação são na Investigação Operacional (Gestão de Projetos e Otimização Combinatória); Inteligência Artificial; e-Learning.

Cristina Maria Carapeto Pereira é Professora na Universidade Aberta desde 1993, no Departamento de Ciências e Tecnologia. É Professora Associada desde 2001. Autora de manuais de estudo universitário e diversos artigos em revistas internacionais tanto na área das Ciências do Ambiente como na área da Nutrição Humana / Saúde. As suas áreas científicas de docência e investigação abrangem as Ciências do Ambiente, Nutrição Humana, Saúde Ambiental e E-Learning.

(esta página par está propositadamente em branco)