consumos de energia sem ficheiro de logs, com 50 000 entradas e com 100 000 entradas para ser possível verificar diferenças. Para obter resultados mais estabilizados, foram executados 20 medições por cada teste e no fim foi calculada a média resultante dos valores.

Apesar do tempo de execução ter sido substancialmente maior, o consumo de energia diminuiu drasticamente em relação à versão original, tanto ao nível de atividade do CPU como ao nível do package.

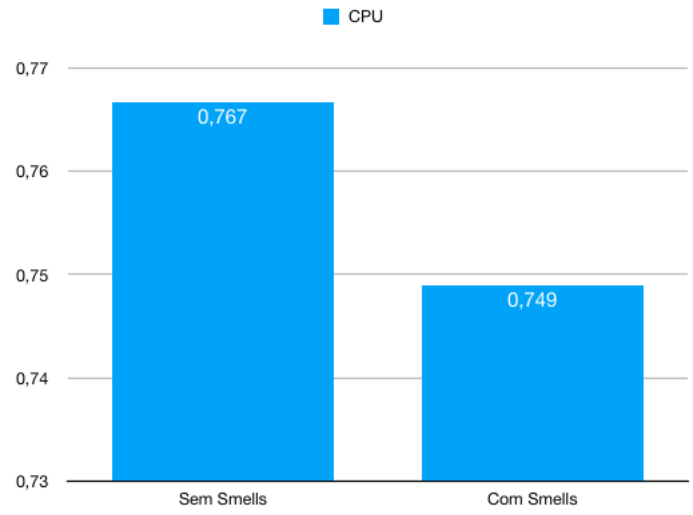

Figura 17: CPU: Sem Log

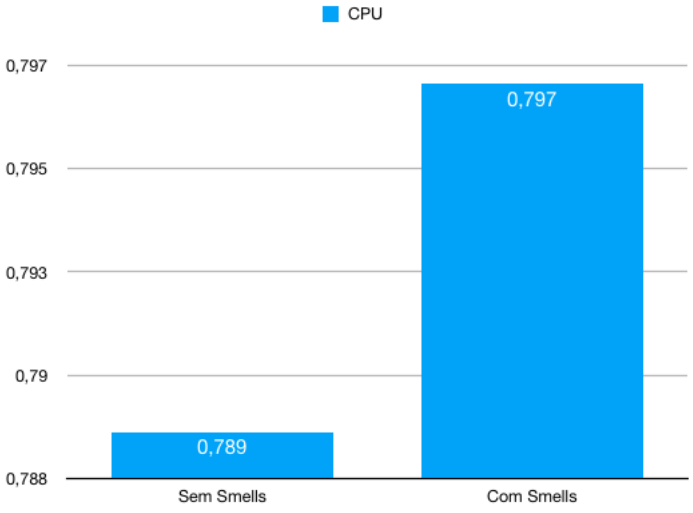

Figura 18: CPU: Com 50 000 Entradas no Log

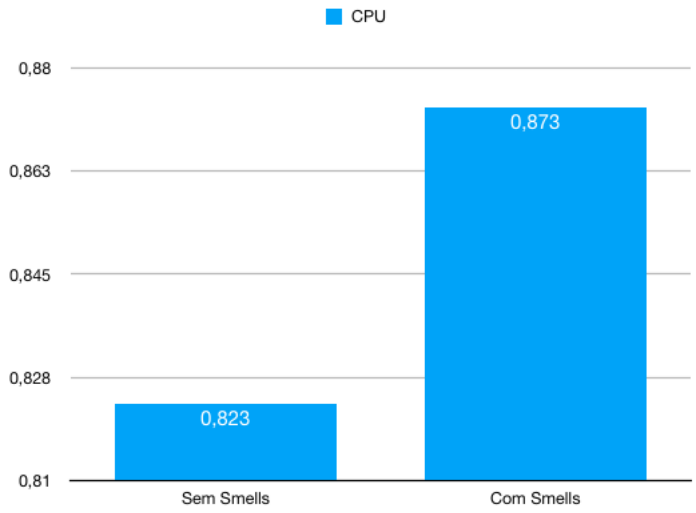

Figura 19: CPU: Com 100 000 Entradas no Log

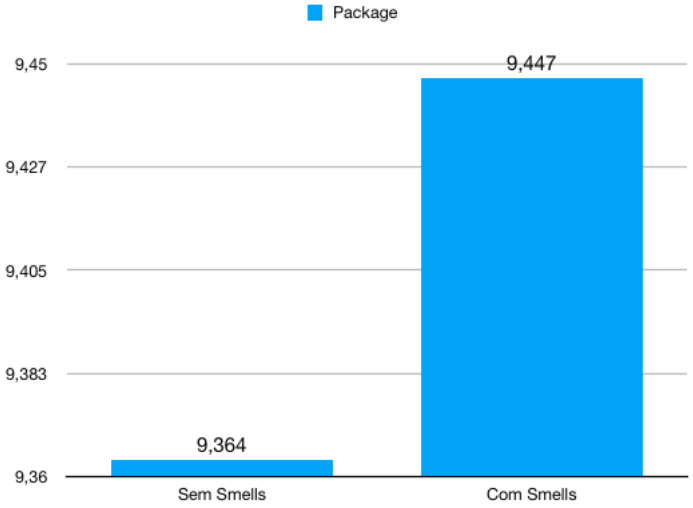

Figura 20: Package: Sem Log

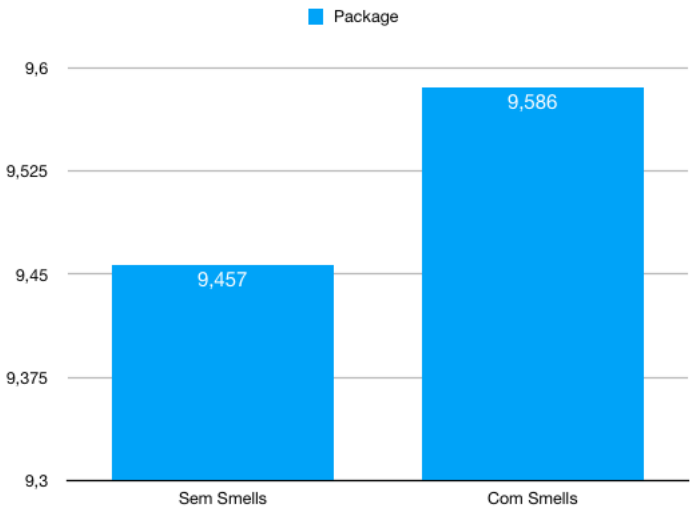

Figura 21: Package: Com 50 000 Entradas no Log

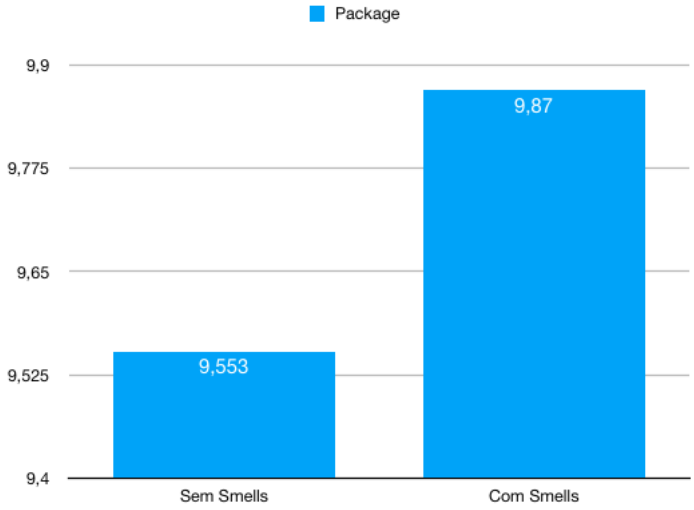

Figura 22: Package: Com 100 000 Entradas no Log

Conclusão

A conclusão de cada tarefa levou a um maior aprofundamento do conhecimento das ferramentas *OpenClover*, *Sonarqube*, *IntelliJ*, *EvoSuite* e *RAPL*.

A partir da conclusão da Tarefa 1, afirma-se que as métricas nem sempre são indicadoras fiéis da qualidade do código-fonte. Podem levar à descoberta de imperfeições, mas não as garantem. Posto isto, depois de concluída a Tarefa 2, foi possível identificar, em certas classes, *smells* em maior evidência e métricas que se destacavam em relação às das outras classes, nomeadamente a complexidade ciclomática. Isto permitiu estabelecer uma lista de prioridades, começando pela classe *Controller* que apresentava o maior risco de *smells* e *bugs*.

Depois de eliminar todos os *smells* relevantes, deixando a *Technical Debt* virtualmente 0, efetuaram-se testes de energia usando *RAPL* e daqui seria de esperar que os tempos de execução e custos energéticos medidos na aplicação sem *smells* melhorassem em relação aos valores medidos na aplicação original. O que aconteceu de facto foi que os custos energéticos melhoraram, mas o tempo de execução aumentou.

Logo, é preciso apontar que os resultados provenientes de uma ferramenta como o *RAPL* podem variar consoante o *hardware* instalado e a quantidade de *software* a correr em paralelo.

Com isto ficamos a saber que o *refactoring* em escala pode de facto melhorar fatores como a legibilidade e manutenibilidade do código-fonte, mas não significa necessariamente que a *performance* de um programa pode ser melhorada ou prejudicada.

É necessário manter sempre um equilíbrio entre manutenibilidade e *performance*, identificando onde os problemas podem surgir com mais probabilidade (tal como na Tarefa 1 com a ferramenta *OpenClover*) e focando-se mais nesses sítios, tendo sempre em conta que é necessária caução quanto à optimização prematura.