

Mestrado em Engenharia Informática
Estágio
Relatório Final

Implementação de Subsistema de Logging

Gonçalo Forte
gforter@student.dei.uc.pt

Orientador:
Teresa Mendes

Data: 12 de Julho de 2012



FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

A criação de registos sobre a execução de aplicações é uma prática comum no desenvolvimento de software conhecida pelo termo em inglês “Logging”, que tem um papel fundamental na manutenção de uma aplicação. Não obstante a sua importância, o logging e a sua implementação são muitas vezes menosprezados pelos programadores, o que poderá ter na performance das aplicações e na qualidade dos registos.

Neste trabalho abordamos conceitos que clarificam a função do logging no desenvolvimento de software e sugerimos directrizes de utilização de logging que potenciem a qualidade dos registos de log e minimizem o seu impacto na performance das aplicações. Adicionalmente, aplicámos este conhecimento na especificação, implementação e expansão de um subsistema de logging em .Net, que tem como função servir de "façade" para a utilização de diferentes ferramentas de logging.

Palavras-Chave

Benchmarks, Enterprise Library, Guidelines, Log, log4net, Logging, Logging Application Block, Nlog, Normalização, Performance

Índice

Capítulo 1 Introdução	1
1.1. Contextualização do Logging no âmbito do trabalho	2
Capítulo 2 Objectivos da Investigação	3
2.1. Documento de <i>Guidelines</i>	3
Performance	3
Normalização	3
2.2. Implementação e Expansão do Subsistema de Logging	4
Capítulo 3 Estado da Arte.....	5
3.1. Conceitos	5
Logging.....	5
Níveis de Log e Verbosidade.....	6
Categorias de Log e Severidade.....	8
3.2. Performance do Logging.....	9
Escrita para ficheiros.....	9
Expressões Lambdas.....	9
Construção das Mensagem de Logging	10
Formatação do elemento de TimeStamp	11
3.3. Normalização do Logging	11
3.4. Ferramentas de Logging	13
3.5. <i>Façades</i> de Logging	15
3.6. Logging e “Aspect Oriented Programming”	16
3.7. Logging e “Caller Info Attributes”	17
Capítulo 4 Metodologias de Abordagem	18
4.1. Medições de Performance	18
4.2. Desenvolvimento de Software.....	19
4.3. Ferramentas de Apoio.....	20
WinCVS - Sistema de Controlo de Versões	20
WISE – Relatório de esforço	21
Enterprise Architect – Desenho e Documentação de Software	23
Visual Studio 2010 – Plataforma de Desenvolvimento	23
Capítulo 5 Trabalho Realizado e Resultados.....	24

5.1. Especificação do Sistema.....	24
Documento de Especificação dos Requisitos de Software	24
Documento de Especificação da Arquitectura do Software.....	32
5.2. Medições de Performance	37
Frameworks de Logging.....	37
Concatenação de Strings	38
Formatação do Timestamp.....	39
Uso de “Reflection” na construção das mensagens	40
5.3. Documento de <i>Guidelines</i>	41
Cenários de Logging	41
5.4. Implementação e Expansão do Subsistema de Logging.....	43
Capítulo 6 Plano de Trabalho e Implicações.....	45
Capítulo 7 Conclusões	49
Apêndices	54
Apêndice 1 – Análise de registos de execução de projectos da CSW.....	55
Apêndice 2 – Processo de Desenvolvimento de Software da CSW	57
Apêndice 2 – Benchmark de Concatenação de strings	68
Apêndice 3 – Benchmark de formatação de Timestamp.....	69
Apêndice 4 – Diagrama de Responsabilidades.....	70
Apêndice 5 – Frameworks de logging para .Net.....	71
Apêndice 6 – “Façades” de logging para .Net.....	74
Apêndice 7 – Exemplo de utilização de AOP.....	76
Apêndice 8 – Valores dos Benchmarks.....	78

Lista de Figuras

Figura 1 – Expansão do Subsistema de Logging	4
Figura 2 – Níveis de log do log4net	7
Figura 3 – Ficheiro de log (Figura 17.2 do livro “Release It!”)	12
Figura 4 – Ficheiro de log (Figura 17.3 do livro “Release It!”)	12
Figura 5 – Ficheiro de log (Figura 17.4 do livro “Release It!”)	13
Figura 6 – ASPerfBench (Ferramenta de Benchmarking)	19
Figura 7 – WinCVS: Sistema de Controlo de Versões	20
Figura 8 – Wise: Registo de esforço	22
Figura 9 – Estrutura de desenho do Subsistema de Logging	23
Figura 10 – SRS: Features Arquitecturais	25
Figura 11 – SRS: Features Funcionais	26
Figura 12 – SRS: Features para as guidelines (Conceitos, Performance e Qualidade)	27
Figura 13 - SRS: Requisitos Arquitecturais	29
Figura 14 – SRS: Requisitos Funcionais	30
Figura 15 – Contexto do sistema	32
Figura 16 – Diagrama de Componentes	35
Figura 17 – Diagrama de Sequência	36
Figura 18 – Logging Frameworks Performance	37
Figura 19 – Benchmark para concatenação de strings	38
Figura 20 – Benchmark para formatação de timestamp	39
Figura 21 – Custo de “Reflection” na construção das mensagens de log	40
Figura 22 – Planeamento Inicial	46
Figura 23 – Planeamento final	47
Figura 24 – Ciclo de fase de vida de um software	57
Figura 25 - Elementos do processo de desenvolvimento de software	58
Figura 26 – Rational Unified Process	59
Figura 27 – Ciclo de fases de vida de um software (detalhado)	60
Figura 28 – Fase de Requisitos de Engenharia	61
Figura 29 – Fase de Desenho de Engenharia	62
Figura 30 – Fase de Validação	63
Figura 31 – Fase de Aceitação	64
Figura 32 – Fase de Manutenção	65
Figura 33 – Processo de desenvolvimento de software simplificado	66
Figura 34 – Responsabilidades	67
Figura 35 – Diagrama de Responsabilidades	70
Figura 36 – Simple Logging Façade	75

Lista de Tabelas

Tabela 1 – Comparação de frameworks de logging para .Net	14
Tabela 2 – Cenário de Logging (Informação vs Performance).....	42
Tabela 3 – Benchmark de Frameworks de Logging	78
Tabela 4 – Benchmark de uso de “reflection” na construção de mensagens de log	78

Lista de Abreviaturas

AOP	Aspect Oriented Programming
API	Application Programming Interface
CASE	Computer-Aided Software Engineering
CSW	Critical Software S.A.
CVS	Concurrent Versioning System
DFD	Data Flow Diagram
DLL	Dynamik-Link Library
EA	Enterprise Architect
ID	Identificator
IDE	Integrated Development Environment
ISO	International Organization for Standardization
LAB	Logging Application Block
MSDN	Microsoft Developer Network
SAS	Software Architecture Specification
SDP	Software Development Process
SLF	Simple Logging Façade
SRS	Software Requirements Specification
VS	Visual Studio
XML	Extensible Markup Language

Capítulo 1

Introdução

O presente trabalho, subordinado ao tema “Implementação de um Subsistema de Logging”, surge no âmbito da realização de um estágio curricular que visa cumprir as exigências para a obtenção do grau de Mestre em Engenharia Informática pela Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Foi desenvolvido na Critical Software, uma empresa de desenvolvimento de software, especializada no desenvolvimento de soluções, serviços e tecnologias para os sistemas de informação críticos de empresas e organizações de diversos sectores.

No desenvolvimento de software, a performance e a qualidade dos registos de execução são requisitos fundamentais, que podem ser comprometidos por uma utilização incorrecta do logging. Assim, a CSW decidiu investir neste projecto/estágio, com o intuito de melhorar o modo como o logging é aplicado na empresa. Isto implica reduzir o impacto dos módulos de logging na execução das aplicações e produzir registos de execução com uma estrutura homogénea e consistente, que facilite a sua legibilidade e a manutenção das aplicações.

Para alcançar os resultados desejados, a CSW definiu dois objectivos principais: a expansão de um subsistema de logging, já existente na framework de desenvolvimento de software em .Net da empresa, que será responsável por otimizar, uniformizar e automatizar a construção de mensagens de logging; e a criação de um documento com directrizes (*guidelines*) de implementação e utilização logging, que servirá de apoio aos programadores durante a fase de desenvolvimento das aplicações.

Aquando da criação do subsistema de logging original, os mecanismos de optimização, uniformização e automatização já haviam sido considerados. No entanto, foram detectadas necessidades de melhoria para este sistema, pelo que os objectivos específicos para este subsistema são: expandi-lo por forma a suportar a framework de logging da Microsoft, o Enterprise Library - Logging Application Block, visto que actualmente o sistema apenas suporta a framework de logging da Apache, o log4net; dotá-lo de métodos de construção de mensagens mais intuitivos e menos intrusivos; e fazer as alterações necessárias para que o sistema respeite ao máximo princípios básicos e fundamentais da programação orientada a objectos.

No que diz respeito ao documento de *guidelines* de logging, os objectivos passam por definir e clarificar conceitos essenciais, tais como níveis de verbosidade, cenários e categorias de logging, e apresentar exemplos e soluções que ajudem os programadores a tirar o melhor partido do logging.

1.1. Contextualização do Logging no âmbito do trabalho

A criação de registos sobre a execução de um software (logging) é indispensável para a sua manutenção. Sem a existência destes registos seria praticamente impossível ao programador monitorizar o software e reagir adequadamente perante um erro.

Apesar de o logging poder ser usado em diferentes contextos, como o desenvolvimento de software ou a administração de sistemas, neste trabalho iremos focar-nos apenas no âmbito do desenvolvimento de software. Dentro do âmbito do desenvolvimento de software o logging pode ser igualmente útil para outros propósitos que não a manutenção de software, tais como a depuração (*debugging*) ou auditoria. No entanto, para estes efeitos existem ferramentas mais apropriadas, como os depuradores dos IDEs ou as ferramentas de rastreamento (*tracing*).

Aproveitando a referência ao conceito de *tracing*, é importante esclarecer que *tracing* e logging são conceitos diferentes e devem ser usados de maneiras diferentes. O *tracing* deve ser usado para efeitos de *debugging* e ajustes de performance (*performance tuning*) durante a fase de desenvolvimento do software, enquanto o logging deve ser usado durante a fase de produção do software, para efeitos de monitorização operacional e resolução de problemas.

Capítulo 2

Objectivos da Investigação

Neste capítulo iremos apresentar e descrever os principais objectivos deste trabalho. Para este projecto existem dois grandes objectivos: a criação de um documento de guidelines que ajude o programador a produzir registos de log com maior qualidade e menor impacto na performance das aplicações; e a implementação/expansão de um subsistema de logging que irá servir de interface para a utilização de diferentes frameworks de logging e que irá ser ao mesmo tempo ser responsável por implementar mecanismos que normalizem a produção de registos de log.

2.1. Documento de *Guidelines*

O documento de *guidelines* tem como principal objectivo orientar o programador, durante a fase de implementação de instruções de logging, para as melhores práticas. Através deste documento pretende-se que o programador seja capaz de identificar os vários cenários de logging e posteriormente que saiba adoptar as melhores estratégias para obter registo de execução de aplicações que forneçam a melhor relação entre o impacto na performance e a quantidade e qualidade da informação.

Performance

A existência de logging numa aplicação acarreta forçosamente custos adicionais de performance para a aplicação e esta é uma particularidade que deve ser tida em conta durante a implementação do logging numa aplicação.¹

Um dos principais objectivos deste trabalho é encontrar soluções que permitam reduzir o impacto do logging nas aplicações desenvolvidas na CSW. Posto isto, é necessário ter em consideração factores, como a localização das instruções de logging no código, as abordagens de programação utilizadas, quantidade de informação registada, entre outros, pelo que se pretende analisar todas estas variáveis e chegar a uma conclusão sobre quais as melhores soluções a utilizar em cada caso.

Normalização

A principal função do logging é produzir registos da execução das aplicações para efeitos de manutenção. A pessoa responsável pela monitorização e manutenção de um software necessita de perceber o que está a acontecer durante a execução da aplicação, pelo que é necessário que os registos de log sejam legíveis e que contenham a informação suficiente e objectiva. Na realidade isto nem sempre acontece e o resultado são registos de execução ilegíveis, com informação escassa e irrelevante e estruturas de mensagens completamente diferentes, por vezes mesmo dentro do mesmo projecto.²

Tendo em conta estas necessidades, definimos como objectivo estudar e sugerir novas estruturas e formatos de mensagens de log que permitam uniformizar e normalizar a forma como é feito o log pelos diferentes programadores no mesmo projecto e especificamente a

¹ Esta preocupação está patente nos objectivos de design e documentação de frameworks como LAB da Microsoft, Nlog e log4net.

² Tal como podemos constatar pela análise de registos de execução de projectos da CSW, presente no Apêndice 1.

forma como o log é feito em todos os projectos da CSW. Só através da normalização dos processos de utilização do logging nos projectos é que será possível, por exemplo, fazer “merge” de logs vindos de fontes diferentes.

2.2. Implementação e Expansão do Subsistema de Logging

Tal como já referido, a CSW possui um subsistema de logging para a plataforma de desenvolvimento de software em .Net, que tem como função servir de “façade” para a utilização de diferentes ferramentas de logging. No início deste trabalho o subsistema de logging apenas suportava a framework de logging externa Apache log4net, pelo que um dos objectivos passa naturalmente pela expansão do subsistema de logging por forma a suportar mais uma framework de logging externa, neste caso, o Logging Application Block que vem com a distribuição Enterprise Library da Microsoft.

Para além da referida expansão, pretende-se que o subsistema actual seja avaliado por forma a implementar as alterações necessárias para que o sistema passe a adoptar as directrizes especificadas no documento de *guidelines*.

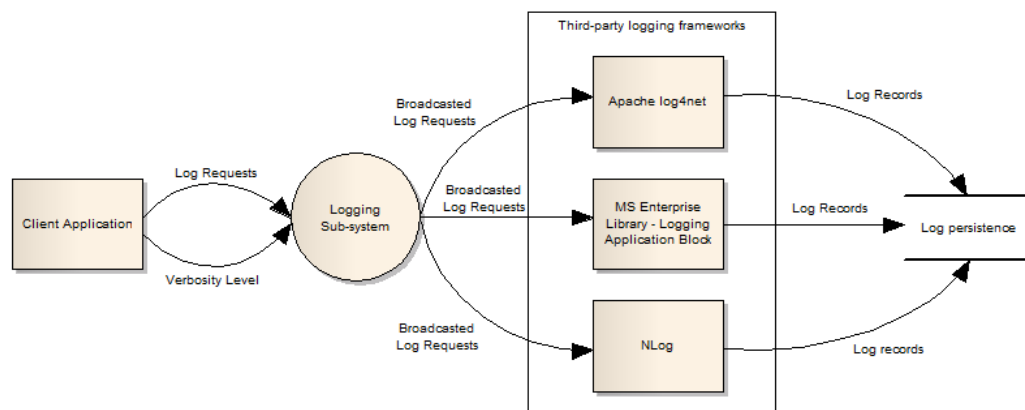


Figura 1 – Expansão do Subsistema de Logging

A imagem anterior representa o contexto inicial do sistema e da expansão do mesmo, através da inclusão de duas novas frameworks de logging externas. Como podemos ver, as “aplicações cliente” (que aqui se referem a aplicações que usam o subsistema de logging e não a “aplicações cliente” de acordo com o modelo cliente/servidor) interagem com o subsistema através de pedidos de log e da definição do nível de verbosidade. Por sua vez o subsistema de log reencaminha o processamento e persistência dos registos para as frameworks de logging suportadas pelo sistema. Inicialmente o subsistema suportava apenas a framework log4net, mas após a expansão passou a suportar também o LAB da Microsoft e o NLog.

Convém ainda salientar que para além das alterações necessárias ao código do subsistema de logging para que este pudesse albergar as novas frameworks de logging, o subsistema sofreu igualmente alterações de modo a implementar as directrizes de performance e qualidade dos registos de log, definidas no documento de *guidelines*.

Capítulo 3

Estado da Arte

Neste capítulo iremos construir um conhecimento sólido sobre a temática do logging, abordando e clarificando os principais conceitos e apresentando algumas das principais técnicas e abordagens utilizadas neste contexto. O capítulo começa exactamente definir e clarificar os principais conceitos relacionados com o logging, como o próprio conceito de logging, os níveis de log ou verbosidade, as categorias de log ou severidade. De seguida apresentamos e descrevemos algumas das principais técnicas e abordagem relacionadas com a performance e a normalização do logging, as frameworks de logging abordadas neste trabalho, algumas das “façades” de logging existentes no mercado, a utilização de AOP e de “Caller Info Attributes” na implementação de mecanismos de logging.

3.1. Conceitos

Para melhor entender o logging é necessário definir o seu conceito e circunscrever o seu âmbito. No início deste trabalho fizemos uma introdução à temática do logging e à sua contextualização no âmbito deste trabalho, no entanto neste capítulo iremos dar uma definição completa e fechada para o conceito de logging.

No contexto do logging, os conceitos de verbosidade e severidade são dois conceitos essenciais para a compreensão do funcionamento do logging, mas que geram confusão entre si, pelo facto de utilizarem nomenclaturas idênticas para definir os seus níveis de acção. Decidimos então reunir algumas citações e definições para os dois conceitos, compará-las, analisá-las e no final apresentar a nossa interpretação destes conceitos e como devem ser inseridos no contexto do desenvolvimento do software e mais concretamente no contexto do nosso projecto.

Logging

O logging é um conceito conhecido por muitos mas que poucos conseguem definir precisamente. É esta falta de definição causa confusão com conceitos similares (tracing) e utilizações indevidas de outras ferramentas (debugging, auditoria) para os mesmo propósitos. Pelo que é importante definir exactamente o que é o logging.

Depois de consultarmos alguns autores, reunimos algumas definições para o conceito e propósito do logging:

“Software logging is a conventional programming practice. As its efficacy is often important for users and developers to understand what have happened in production run for failure diagnosis... Logs are particularly beneficial to diagnosing production-run failures, which often require immediate resolutions as they directly impact the end users. Log messages printed during the production run are often the only data source available for developers to diagnose such failures. This is because it is often challenging to reproduce production failures... Consequently, software engineers mostly rely on log messages for trouble-shooting production failures. Besides developers, logs are also used by system administrators to resolve failures caused by security attacks, hardware errors, and misconfigurations.” (Characterizing Logging Practices in Open-source Software, 2012)

“Despite millions of R&D dollars on “enterprise application management” suites and spiffy operations centers with giant plasma monitors showing color-coded network maps, good old log files are still the most reliable, versatile information vehicle... Most developers implement logging as though they are the primary

consumer of the log files. In fact, administrators and engineers in operations will spend far more time with these log files than developers will. Logging should be aimed at production operations rather than development or testing.” (Nygard, 2007)

“Logging is an essential tool in every developer's arsenal. It helps the developer to identify problems faster by showing the state of an application at any given point. It is important after deployment, when all that the poor system admins have are the logs that are generated by your application.” (Legheri, 2003)

Estas citações confirmam os nossos pressupostos em relação ao logging, nomeadamente no que diz respeito ao facto de que o logging deve ser usado durante a fase de produção do software. Os autores referem, e muito bem, que o logging deve ser usado para monitorizar a execução da aplicação (hardware erros, misconfigurations, etc) e não para efeitos de debugging ou de auditoria. Estas considerações são muito importantes na definição do conceito de logging, pois permitem à partida identificar a natureza do logging e desta forma afastar conceitos idênticos ao log (como são os casos do tracing e da auditoria).

A principal dificuldade na distinção dos conceitos de logging, tracing e auditoria deve-se ao facto de algumas tecnologias serem utilizadas para os três conceitos e ao facto de muitos dos critérios que os distinguem serem contíguos e não discretos. Tal como podemos constatar pelas citações anteriores, o logging deve ser usado para registar eventos sobre a execução das aplicações por forma a auxiliar o administrador durante a fase de manutenção do software, enquanto o tracing e a auditoria têm propósitos diferentes. O tracing deve ser usado durante a fase de desenvolvimento para auxiliar o programador, e a auditoria deve ser usada para registar contexto de negócio da aplicação³. Para cada um dos casos existem técnicas e ferramentas diferentes que se adequam melhor às necessidades de cada um.

Posto isto, somos capazes de avançar com uma definição para o conceito de logging:

“Criação otimizada (em termos de performance) de registos estruturados sobre a execução de um software, com o intuito de auxiliar a fase de manutenção do software”.

Níveis de Log e Verbosidade

Sendo o logging uma ferramenta de manutenção de software, é necessário que este seja capaz de se adaptar às diferentes necessidades de performance e de volume de informação da fase de manutenção. Se o sistema se encontra em produção, queremos que o logging tenha o menor impacto possível e que apenas registre eventos importantes. Caso o sistema esteja a manifestar algum comportamento incorrecto, podemos querer aumentar a quantidade de informação registada pelo log e sacrificar um pouco a performance da aplicação para conseguir perceber e resolver o problema. Neste contexto, o volume de informação gerada tem o nome de “verbosidade” e a sua gestão é feita através da definição de níveis de log. São atribuídos níveis de log a cada mensagem, que posteriormente serão comparados ao nível de log configurado no sistema. Se o nível de verbosidade da mensagem for igual ou inferior ao nível de log configurado no sistema, a mensagem será registada, caso contrário a mensagem será ignorada. No entanto, a sua utilização não é consensual, essencialmente porque estes conceitos tendem a ser confundidos com os conceitos de categorias de log e severidade, que serão abordados de seguida. À semelhança do que acontecia com os conceitos de logging e tracing, esta confusão deve-se ao facto de estes conceitos serem utilizados, erradamente, para os mesmos objectivos, ou adoptarem as mesmas nomenclaturas.

³ “Auditing and Logging”, .Net Application Architecture Guide, 2nd Edition – Chapter 4: A Technique for Architecture and Design (Microsoft, 2009).

Mais uma vez, decidimos recorrer à literatura existente sobre o assunto para comprovar os nossos pressupostos e nos ajudar a esclarecer as confusões:

*“You establish how much data each provider reports by setting the verbosity level in the provider file. The verbosity level is a number in the provider file, ranging from 0 to 5, where 0 is the least information and 5 is the most information.”*⁴

No nosso entender a citação anterior indica o propósito essencial dos níveis de verbosidade num sistema de logging: estabelecer o volume de informação gerado pela aplicação.

“Not only is this great for ranking the importance of a particular entry, it can also be used to control the amount of logging making its way through to your log repository of choice.” (Twist, 2007)

Nesta citação está patente a confusão a que nos referíamos, pois se por um lado é reforçada a ideia de que os níveis de log devem ser usados para controlar o volume de registos criados, por outro lado é atribuída aos níveis de log, a responsabilidade de classificar a importância ou severidade de cada registo, função que é da responsabilidade das categorias de log, como iremos ver adiante. Neste caso, a confusão não se deve a um equívoco do autor, mas sim ao facto de o autor querer explicar a utilização do logging no contexto de ferramentas como o log4net e o NLog, que misturam os dois conceitos ao utilizarem os níveis de log para classificar as entradas de log e filtrar o volume de log gerado.

Regra geral as ferramentas de logging dividem os níveis de log em cinco ou mais níveis^{5 6}. No entanto esta divisão não traduz o conceito de níveis de verbosidade, mas sim os conceitos de categorias de log e severidade, uma vez que a filtragem do volume de log é feita de acordo com a categoria do log.

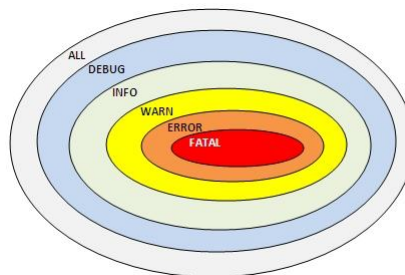


Figura 2 – Níveis de log do log4net⁷

Esta abordagem torna-se pouco flexível, pois quando um sistema está definido o nível de verbosidade mais baixo estamos limitados a mensagens do tipo “Fatal” sendo completamente impossível obter por exemplo mensagens do tipo “Info” ou do tipo “Warn”.

Em resumo, a literatura e documentação existentes confirmaram que os níveis de verbosidade servem para filtrar a produção de log e consequentemente controlar o seu impacto na performance das aplicações. No entanto, consideramos que os exemplos práticos não respeitam este conceito, pelo que sugerimos uma abordagem que utiliza apenas quatro níveis de log, com nomenclaturas mais adequadas ao conceito:

- “High” – Nível de log de maior verbosidade, com maior impacto na performance das aplicações. Deve ser utilizado quando ocorrer algum erro e a informação

⁴ Microsoft.com | TechNet – Verbosity levels (IIS 6.0).

⁵ Apache log4net Manual: Introduction.

⁶ NLog Tutorial: Log Levels.

⁷ Codeproject.com: log4net and SQLite

fornecida pelos restantes níveis não for suficiente para o detectar ou resolver. Preferencialmente, este nível de log não deve ser usado em produção;

- “Medium” – Nível de log de verbosidade média e com menor impacto na performance, aconselhado para fases de validação e testes de software. Pode ser utilizado em produção/exploração se a aplicação não tiver grandes requisitos de performance;
- “Low” – Nível de log de verbosidade baixa e com o menor impacto na performance. Este é o nível aconselhado para a fase de produção/exploração das aplicações;
- “Off” – Nível de log que desliga a produção de registos de log.

Categorias de Log e Severidade

Durante a execução de um software é necessário fazer log de diferentes tipos de eventos com diferentes características, pelo que é necessário classificar cada um desses eventos para que se torne mais fácil ao administrador do sistema identificá-los e compreendê-los. Esta classificação é realizada através de definição de categorias de log. Em algumas frameworks de logging estas categorias traduzem literalmente a natureza do evento em termos da severidade desse evento dentro da execução do software, pelo que estas categorias são igualmente conhecidas como categorias de severidade.

Tal como foi referido na secção anterior este conceito é confundido com o conceito de verbosidade, pois na realidade algumas frameworks de logging fazem um paralelismo entre categorias de log e níveis de log, utilizando estes dois conceitos em simultâneo para classificar eventos de log e filtrar a produção de log. Mas como já vimos essa abordagem torna os sistemas confusos e pouco flexíveis.

A principal função das categorias de log é classificar os diferentes eventos para que possam ser tomadas acções de acordo com cada categoria. Como já referimos a classificação das mensagens de log ajuda na identificação e compreensão dos eventos registados, mas serve também para definir comportamentos e acções diferentes de acordo com cada categoria, como criar alertas ou definir diferentes outputs.

Após a análise de documentação sobre o tema⁸, reunimos diferentes listagens para as categorias de log. No entanto detectámos que apesar de ligeiras diferenças (nas nomenclaturas), existe um consenso em torno de cinco categorias de log, que descrevemos de seguida:

- Debug – Fornece informação extra sobre um determinado evento ou execução da aplicação. Por norma, este tipo de mensagem é mais detalhada do que as mensagens do tipo “Info”;
- Info – Contém informação útil sobre a aplicação e que ajuda o administrador a perceber qual o estado da aplicação. Por norma é mais usada durante a inicialização e encerramento das aplicações;
- Warn – Avisa o administrador sobre situações não desejadas e potencialmente prejudiciais para a execução da aplicação. Falta de memória/espço ou falhas de configuração são exemplos de situações ideais para fazer log de um aviso;

⁸ A análise focou-se essencialmente nos artigos “Syslog#Severity_levels” e “Log4j#Log level” no site Wikipedia.org, no artigo “Logging Levels and how to use them”, no site Thejoyofcode.com e no secção “Logging levels” do capítulo “10. Logging Conventions” da documentação do jBoss.

- **Error** – Alerta o administrador para erros de execução da aplicação, mas que não comprometem necessariamente a execução da aplicação. Estes erros são por norma situações previstas, mas que indicam que existe algum problema na execução da aplicação. As mensagens do tipo “Error” são tipicamente implementadas dentro de *catch blocks*;
- **Fatal** – Alerta o administrador para erros críticos na aplicação, que comprometem a sua normal execução e que necessitam de intervenção imediata. Uma mensagem do tipo “Fatal” indica que o sistema a partir desse momento já não se encontra estável e como tal não garante o seu normal funcionamento. Em alguns casos a intervenção ideal pode passar por desligar o sistema.

Posto isto, resta-nos explicar como é que os conceitos de verbosidade e categorias de logging se relacionam. O ideal será que os eventos de log do tipo “Fatal” e “Error” sejam registados independentemente do nível de log definido, enquanto que os eventos do tipo “Debug” apenas devem ser registados em níveis de verbosidade alta. Os eventos do tipo “Info” e “Warning” devem surgir consoante o cenário de execução. No entanto, é possível registar eventos do tipo “Debug” em níveis de verbosidade baixa, se as necessidades do projecto assim obrigarem.

3.2. Performance do Logging

A questão da performance é uma preocupação transversal no contexto do logging e vários melhoramentos têm sido feitos desde que surgiram as primeiras versões das frameworks de logging. De seguida, iremos apresentar algumas das considerações de performance, incluindo algumas que já são inclusivamente usadas pelas frameworks de logging actuais.

Escrita para ficheiros

A versão inicial do LAB (2005) abria e fechava os ficheiros de log após cada escrita. Naturalmente isto originava fracos resultados de performance, na ordem dos 400 Logs/s. Para a versão 2.0 do LAB a abordagem anterior foi substituída por uma abordagem onde os registos de log eram *flushed* (através do uso do método `Flush()`, nativo da linguagem C#) após cada escrita de log aumentando a frequência de registos gerados para 3000 Logs/s.

Expressões Lambdas

A avaliação do nível de log das mensagens foi durante algum tempo um dos aspectos que aumentava o impacto do logging na performance das aplicações. Isto porque independentemente de as mensagens serem ou não escritas para log, eram gastos recursos para processar todos os parâmetros da mensagem. Inicialmente a solução adoptada para evitar esta situação foi a inclusão de uma verificação do nível de log antes da chamada de log propriamente dita.⁹

```
if(log.IsDebugEnabled)
{
    log.Debug("Entry number: " + i + " is " + entry[i]);
}
```

Esta abordagem realmente evita que a chamada de log seja realizada caso o nível de log do sistema seja inferior ao nível da chamada. Trata-se de uma abordagem que não privilegia em

⁹ “What is the fastest way of (not) logging?” em Apache log4net Frequently Asked Questions.

nada a manutenção e legibilidade do código e que continua a ter custos de performance que podem ser significativos se a instrução se replicar várias vezes no código ou se estiver inserida num ciclo de execução.

Felizmente existe uma solução que poupa o processamento adicional de construção da mensagem caso a mensagem seja ignorada e que passa pela utilização de expressões lambda.

As expressões lambda, também conhecidas por funções anónimas, permitem que o processamento do nível de log seja feito antes do processamento dos parâmetros da mensagem, pelo que se uma determinada mensagem for ignorada devido ao seu nível de log, já não iremos gastar recursos de processamento com os seus parâmetros. Por outro lado, quando comparada com a abordagem anterior, a utilização de expressões lambda aumenta significativamente a legibilidade do código.

Esta é uma solução sugerida em várias fontes¹⁰ e que já foi inclusivamente adoptada pela framework de logging para .Net, o NLog¹¹ na sua versão 2.0.

```
logger.Info(() => "message" + i + ", " + j + ", " + k);
```

Construção das Mensagem de Logging

Este é um dos aspectos onde se verificam grandes problemas de performance para o logging, mais concretamente na forma como é feita a concatenação/construção da mensagem de log. Existem algumas tentativas de melhoramento deste aspecto, como a utilização de diferentes abordagens de concatenação de *strings* ou a formatação da mensagem de uma forma assíncrona. No entanto nem todas as alternativas se provaram válidas.

Por exemplo, o uso de assincronismo para a construção da mensagem de log foi uma das alternativas implementadas em algumas frameworks de logging que não teve grande sucesso pois a abordagem acarreta alguns problemas¹²:

- No LAB é possível utilizar *listeners* não nativos que usam o objecto “TraceEventCache” para obter contexto, incluindo informação específica da *thread*, que não pode ser controlada. Isto significa que a realização de uma operação de logging numa *thread* diferente da de origem pode resultar num registo de log cuja informação de contexto esteja errada;
- Adicionalmente, o objecto “TraceEventCache” fornece também a *call stack*, que pode ser pré-inicializada na *thread* original. No entanto, esta é uma operação dispendiosa e muito provavelmente um desperdício uma vez que nem todos os *listeners* necessitam de tal inicialização;
- Com esta abordagem não há garantias de que todas as entradas de log sejam registadas. Por exemplo, se uma entrada de log for enviada para uma *thread* poucos instantes antes de a aplicação terminar, é possível que essa *thread* de logging nunca chegue a terminar o seu processamento.

No que diz respeito à construção das mensagens existem diferentes abordagens utilizadas pelas diferentes frameworks de logging. O LAB, por exemplo, antes da versão 4.0 utilizava o *StringBuilder* em conjunto com o *StringBuilder.Replace* para a construção das suas

¹⁰ “Making Logging Cheaper with Lambda Expressions”, de Elton Stoneman em geekswithblogs.net

¹¹ “NLog 2.0 Release Notes”

¹² Informação retirada do artigo: “Thoughts on improving performance of the Logging Application Block”

mensagens de log. No entanto, após a sugestão de Alois Kraus¹³, esta abordagem foi substituída por uma abordagem que usava o `StringBuilder.Append` em detrimento do `StringBuilder.Replace`, tornando assim a construção das mensagens 13 vezes mais rápida do que anteriormente.

Formatação do elemento de `TimeStamp`

Tal como referimos anteriormente a construção das mensagens de log pode ter grande impacto na performance do logging e consequentemente na performance das aplicações. Sendo assim, é necessario considerar todas as possíveis melhorias neste aspecto. Uma das possíveis melhorias detectadas é a formatação do elemento de timestamp. Este elemento é fundamental no corpo de uma mensagem de log pois permite ordenar e identificar uma entrada de log num ficheiro com inúmeras linhas de log. Mas será que a abordagem utilizada pelas frameworks de logging para formatar este elemento é a melhor? As abordagens que utilizam o `String.Format` e o `StringBuilder` são duas das mais utilizadas no desenvolvimento de software e pelas próprias frameworks de logging, mas será que estas abordagens apresentam os melhores valores de performance? A resposta a estas perguntas será dada mais à frente na secção de medições de performance deste trabalho.

Alois Kraus desafiou¹⁴ programadores a “apresentarem” novas abordagens de formatação do timestamp que fossem mais eficientes que as abordagens mais comuns como o `String.Format` ou o `StringBuilder` e o resultado foram algumas abordagens alternativas com performances superiores às referidas anteriormente. Infelizmente nem todas as abordagens foram consideradas válidas pois utilizam código não considerado seguro.

A abordagem “`FormatFastSafe`” revelou ser a que melhor cumpria os requisitos de eficiência e segurança, pelo que foi adoptada neste trabalho, nomeadamente para a realização de medições de performance, tal como podemos ver pelo Apêndice 3 – Benchmark de formatação de `Timestamp`.

3.3. Normalização do Logging

A normalização e homogeneização dos registos de log é um aspecto muitas vezes menosprezado pelos programadores mas muito valorizado pelos responsáveis pela manutenção do software. A normalização do logging é, a par da performance, um dos aspectos mais importantes do logging, pois tal como já referido, o objectivo do logging é produzir registos sobre a execução das aplicações que sejam úteis e eficazes durante a manutenção de um software.

Sendo assim, que aspectos devem ser tidos em conta para potenciar a eficácia de um registo de log? Após a consulta de alguma documentação encontramos algumas referências que nos podem ajudar a responder a esta pergunta a atingir os objectivos de normalização e homogeneidade.

O livro “*Release It! Design and Deploy Production-Ready Software*”¹⁵ diz-nos o seguinte: *“Above all else, log files are human-readable. That means they constitute a human-computer interface and should be examined in terms of human factors. This might sound trivial—even laughable—but in a stressful situation, such as a Severity 1 incident, human misinterpretation of status information can prolong or*

¹³ No artigo: “Really Fast Formatting with Enterprise Library”.

¹⁴ No seu blog, nomeadamente no post: “Performance Quiz: Fastest way to format time”.

¹⁵ Capítulo IV – Operations, secção 17.4 Logging.

aggravate the problem. Therefore, it behooves us to ensure that log files convey clear, accurate, and actionable information to the humans who read them.”

O autor diz-nos que os registos de log devem ser acima de tudo legíveis. O principal destinatário destes registos são ser humanos, responsáveis pela manutenção dos softwares, pelo que os registos de log devem ser claros, precisos e conter informação suficiente e relevante para quem os lê.

Neste livro, são referidos também alguns dos principais erros na construção de registos de log e apresenta algumas soluções para o problema.

“Figure 17.2, shows an example of a log format that was made for computers or perhaps Martians to read—not humans. This is from the start-up log of WebLogic 9.2. How rapidly can you spot the warning message?”

```
<Aug 13, 2006 7:24:53 PM CDT> <Notice> <Log Management> <BEA-170027> <The server
<Aug 13, 2006 7:24:54 PM CDT> <Notice> <WebLogicServer> <BEA-000365> <Server stat
<Aug 13, 2006 7:24:54 PM CDT> <Notice> <WebLogicServer> <BEA-000365> <Server stat
<Aug 13, 2006 7:24:57 PM CDT> <Notice> <Security> <BEA-090171> <Loading the ident
A-000331> <Started WebLogic Admin Server "examplesServer" for domain "wl_server"
<Aug 13, 2006 7:25:00 PM CDT> <Warning> <WorkManager> <BEA-002919> <Unable to fin
map to the default WorkManager for the application bea_wls9_async_response>
<Aug 13, 2006 7:25:00 PM CDT> <Warning> <EJB> <BEA-014014> <The message driven be
chPolicy" that refers to an unknown work manager. The default work manager will b
Could not invoke browser, command=netscape -remote openURL(http://192.168.1.98:70
can open from the cmd-line.
java.io.IOException: java.io.IOException: netscape: not found
<Aug 13, 2006 7:25:00 PM CDT> <Notice> <WebLogicServer> <BEA-000365> <Server stat
<Aug 13, 2006 7:25:00 PM CDT> <Notice> <WebLogicServer> <BEA-000360> <Server star
```

Figura 3 – Ficheiro de log (Figura 17.2 do livro “Release It!”)

Aqui o autor apresenta um mau exemplo de um ficheiro de log que dificultaria imenso a leitura dos registos de log e consequentemente o processo de manutenção de um software.

Como alternativa, o autor apresenta um exemplo de estrutura para registos de log, significativamente mais legível que o anterior.

“Figure 17.3, on the next page comes from WebSphere 6.1. It uses the same java.util.logging API that the previous example used, but it replaces the awful default format with a much more helpful one. This is a format that the human eye can scan. Once you know that I, W, and A indicate different severity levels (“information,” “warning,” and “audit”), then scanning for warnings and errors becomes trivial. The space-padded, columnar format helps humans read the file. Note the message code field, which aids in automated parsing of the file. This log format helps both humans and computers.”

```
[8/14/06 8:22:14:653 CDT] 0000000a SSLComponentI I CWPKI0001I: SSL service is
[8/14/06 8:22:14:813 CDT] 0000000a WSKeyStore W CWPKI0041W: One or more key
[8/14/06 8:22:14:848 CDT] 0000000a SSLConfigMana I CWPKI0027I: Disabling defau
[8/14/06 8:22:24:639 CDT] 0000000a WorkspaceMana A WKSPO500I: Workspace config
[8/14/06 8:22:25:508 CDT] 0000000a FileRepositor A ADMR0010I: Document cells/ti
[8/14/06 8:22:25:961 CDT] 0000000a SSLDiagnostic I CWPKI0014I: The SSL componer
[8/14/06 8:22:26:325 CDT] 0000000a FileRepositor A ADMR0010I: Document cells/ti
[8/14/06 8:22:26:670 CDT] 0000000a SSLComponentI I CWPKI0002I: SSL service ini
```

Figura 4 – Ficheiro de log (Figura 17.3 do livro “Release It!”)

Esta estrutura denota claramente preocupação com a legibilidade deste registo de log. Aspectos que à primeira vista podem parecer simples, como é o caso do espaçamento entre elementos da mensagem, fazem toda a diferença na legibilidade do registo e na facilidade em encontrar a informação desejada.

Outro exemplo de pequenos aspectos que podem dificultar a legibilidade de um ficheiro de logging é a criação de mensagens de log com mais do que uma linha, tal como refere o mesmo autor.

“Figure 17.4, on the following page, shows the same output as from WebSphere’s start-up sequence but using the default format for JDK’s java.util.logging package. I don’t know who thought up this two-line format,

but it makes scanning through logs utterly impossible. For that matter, the two-line format makes parsing the logs with other programs difficult, too. grep has no idea how to deal with two-line formats. You'll have to dig up an old-school UNIX hacker to do some awk trickery. This format defeats man and machine."

```
Aug 19, 2006 7:13:25 PM com.example.server.SSLComponentIdentity emit
INFO: SSL service is initializing the configuration
Aug 19, 2006 7:13:25 PM com.example.server.WSKeyStore emit
WARNING: One or more key stores are using the default password.
Aug 19, 2006 7:13:25 PM com.example.server.SSLConfigManager emit
INFO: Disabling default hostname verification for HTTPS URL connections.
Aug 19, 2006 7:13:25 PM com.example.server.WorkSpaceManager emit
INFO: Workspace configuration consistency check is false.
Aug 19, 2006 7:13:25 PM com.example.server.FileRepository emit
INFO: Document cells/trozNode01Cell/security.xml is modified.
Aug 19, 2006 7:13:25 PM com.example.server.SSLDiagnostic emit
INFO: The SSL component's FFDC Diagnostic Module com.ibm.ws.ssl.core.SSLDiagnosti
Aug 19, 2006 7:13:25 PM com.example.server.FileRepository emit
INFO: Document cells/trozNode01Cell/security.xml is modified.
Aug 19, 2006 7:13:25 PM com.example.server.SSLComponentIdentity emit
INFO: SSL service initialization completed successfully
```

Figura 5 – Ficheiro de log (Figura 17.4 do livro “Release It!”)

De acordo com esta citação, podemos constatar que a utilização de duas ou mais linhas por mensagem de log, dificulta igualmente a legibilidade para humanos e para máquinas.

Para finalizar esta secção de sugestões de melhoria das estruturas dos registos de log, o autor sugere ainda a inclusão de um elemento identificador (comumente referido como “correlation ID”) na estrutura das mensagens, que ajuda a correlacionar determinadas mensagens de log mais facilmente.

“Messages should include an identifier that can be used to trace the steps of a transaction. This might be a user’s ID, a session ID, a transaction ID, or even an arbitrary number assigned when the request comes in. When it’s time to read 10,000 lines of a log file (after an outage, for example), having a string to grep will save tons of time.”

Resumindo, a qualidade de um registo de log deve medir-se pela relação entre legibilidade, quantidade e precisão da informação, e estas devem ser as principais preocupações do programador aquando da implementação de registos de log.

3.4. Ferramentas de Logging

Nesta secção iremos apresentar as três frameworks de logging para .Net abordadas neste trabalho (log4net, LAB e Nlog) e que são igualmente as frameworks mais referenciadas na Internet¹⁶ e possivelmente as mais usadas pela programadores de .Net. O objectivo principal deste capítulo não passa por descrever exaustivamente cada uma das frameworks e as suas funcionalidades (essa descrição é feita no Apêndice 5 – Frameworks de logging para .Net deste trabalho), mas fazer um estudo comparativo das três frameworks por forma a destacar as principais características, pontos fortes e pontos fracos. Após consulta, encontramos alguns estudos comparativos para estas e outras frameworks de logging¹⁷, que apesar de estarem desactualizados em alguns aspectos ajudam a ter uma maior visão sobre as potencialidades de cada framework. No entanto, nesta secção estamos mais preocupados em apresentar as características mais distintas e que de alguma forma têm correlação com as actividades realizadas neste trabalho.

¹⁶ Segundo as páginas “Google insights for Search” e “Google Trends”.

¹⁷ Os estudos encontram-se nos artigos “Essential Diagnostics” em Codeplex.com, “Comparison of .Net Logging Frameworks and Libraries” em Dotnetlogging.com e “.Net Logging Library Competition Comparison” em Kellermansoftware.com

	log4net	LAB	NLog
Disponibilização	DLL	Enterprise Library	DLL
Expresões Lambda	Não	Não	Sim
Configuração	Programática, XML	Programática, XML	Dinâmica, XML
Consola de configuração	Não	Sim	Não
Suporta Event ID	Necessita extensão	Sim	Necessita extensão
Documentação	Site	MSDN	Site, Fórum
Comunidade	Inactiva	Activa	Activa
Última release	16-07-2003	05-2011	17-07-2011
Log para ficheiros (Performance)	58 ticks	73 ticks	1156 ticks
Log para Event Viewer (Performance)	2078 ticks	281 ticks	2061 ticks

Tabela 1 – Comparação de frameworks de logging para .Net

Através da análise desta tabela é possível concluir que apesar de terem sido desenvolvidas com o mesmo intuito estas frameworks têm algumas características distintas. Vejamos por exemplo que existe uma diferença na forma como são disponibilizadas as três frameworks de logging. Enquanto o log4net e o Nlog são disponibilizadas através de uma única DLL sem quais quer dependências, o LAB necessita que o Enterprise Library esteja instalado na máquina. Este pode ser um aspecto que comprometa a independência do próprio subsistema de logging, uma vez que para adicionar uma funcionalidade de logging temos de instalar um pacote inteiro de outras funcionalidades que podem não ser necessárias.

A utilização de expressões lambda, que como já vimos traz melhorias de performance, só está disponível na framework Nlog. No entanto como podemos ver pelas medições de performance realizadas, essa melhoria não se evidencia nos resultados do Nlog. De acordo com as nossas medições, cujas metodologias e resultados são detalhados mais à frente neste documento, o log4net e LAB apresentam resultados bastante melhores que o Nlog na escrita para ficheiros, enquanto na escrita para o Event Viewer o LAB revela ter muito menos custos de performance que as outras duas frameworks de logging.

As restantes características, não têm impacto relevante na utilização ou performance das frameworks, mas importa salientar que as três frameworks permitem que a configuração seja feita directamente no código da aplicação ou através de um ficheiro de configuração em XML, sendo que para o caso da configuração em XML, o LAB disponibiliza uma consola de configuração que facilita a tarefa. O LAB é ainda a única das três frameworks que suporta nativamente a inclusão de a propriedade “Event ID” (utiliza no Event Viewer) nas suas mensagens de log.

Para finalizar, existem ainda as questões da documentação e da comunidade de cada uma das frameworks. O log4net, apesar de ser uma das frameworks mais usadas, é um projecto praticamente “morto” uma vez que a última versão data de 2003. A única documentação oficial está disponível no seu site, no entanto é possível encontrar muitas referências não oficiais na Internet. O LAB, sendo uma ferramenta da Microsoft, está bastante activo e tem uma comunidade enorme a apoiar o projecto. A sua documentação encontra-se essencialmente no MSDN. No que diz respeito ao Nlog, é uma ferramenta popular na

comunidade open source e tem evoluído graças a este apoio. A sua documentação encontra-se essencialmente no seu site e fórum.

3.5. *Façades* de Logging

Façade deriva do termo “fachada” utilizado em arquitectura para referir a vista principal de um edifício. No caso do desenvolvimento de software, “Façade” é um objecto que fornece uma interface simplificada para um bloco de código maior e mais complexo.¹⁸ No nosso caso, uma “Façade” de Logging é um sistema que engloba e simplifica a utilização de diferentes frameworks de logging.

Mas será que precisamos mesmo de uma *façade*? Porque não usar simplesmente a framework de logging desejada? A resposta não é consensual, mas no nosso entender as vantagens de usar uma *façade* superam as desvantagens. Em primeiro lugar, a utilização de uma *façade* dá-nos flexibilidade. As frameworks de logging têm implementações distintas, pelo que é necessário “aprender” a fazer logging a cada mudança de framework. Por outro lado, com uma *façade* de logging, o esforço em aprender a lidar com cada framework é gasto apenas uma vez. A partir desse momento, a implementação do logging é feita sempre da mesma forma, independentemente da framework de logging a ser usada. Isto traz igualmente vantagens no que diz respeito à manutenção do código, uma vez que não é necessário fazer alterações de cada vez que se troca de framework de logging. Por último a utilização de uma *façade* possibilita inclusivamente que se testem diferentes frameworks de logging, por forma a averiguar qual a framework que melhor se adapta às necessidades do projecto.

Um dos problemas apontados à utilização da uma *façade* é o facto de a abstracção limitar o acesso às funcionalidades das frameworks de logging. No entanto para o nosso caso este até pode ser um aspecto vantajoso. Um dos objectivos deste projecto passa por orientar os programadores para as melhores práticas de logging. Então, porque não implementar essas práticas directamente na *façade* de logging? Na *façade* de logging, podemos definir métodos, estruturas de mensagens e propriedades que não só facilitem a implementação do logging mas que ao mesmo tempo o tornem homogéneo e eficaz. Por outro lado, se realmente for detectada uma funcionalidade não disponível na *façade*, existe sempre a possibilidade de expandir o sistema.

Outra questão várias vezes abordada é a da própria mudança de framework de logging. Será que existe algum programador ou projecto que esteja constantemente a trocar de framework? Ou será que depois de usar uma determinada framework de logging durante algum tempo alguém se vai lembrar de mudar de framework sem razão aparente? Realmente estas questões têm fundamento, porque na maioria dos projectos a escolha da framework de logging é feita no início, muitas vezes por imposição do cliente, e mantém-se até o final do projecto. Mas no contexto de uma empresa, com diferentes projectos e vários programadores é desejável que a implementação e produção do log seja o mais homogénea possível, pelo que a existência de um sistema que abstraia e simplifique a utilização de diferentes frameworks de logging facilitará essa tarefa.¹⁹

No nosso ponto de vista, estas são algumas das razões que justificam a utilização de uma *façade* de logging. Actualmente existem algumas soluções disponíveis no mercado, que descrevemos no Apêndice 6 – “Façades” de logging para .Net, contudo a nossa decisão por implementar um sistema de raiz em vez de utilizar um dos sistemas existentes, prende-se

¹⁸ Informação retirada do artigo “Facade pattern” em Wikipedia.org.

¹⁹ Argumentação baseada nos artigos “What’s the point of a logging facade” na página StackOverflow e “Unit Testing a Logging Wrapper” na página TheJoyofCode.

com o facto de considerarmos que o esforço despendido na implementação ser compensado pelo maior controlo do sistema e das funcionalidades disponibilizadas.

3.6. Logging e “Aspect Oriented Programming”

AOP ou programação orientada a aspectos é uma abordagem de programação que tem como objectivo aumentar a modularidade de um software através do isolamento de questões de programação transversais a todo o projecto. Existem várias técnicas e abordagens de programação que permitem agrupar funcionalidades e aumentar a modularidade de um software, no entanto existem algumas questões que ultrapassam este paradigma e o logging é um exemplo perfeito desse paradigma. Nós podemos agrupar todas as funcionalidades relativas ao logging numa classe, framework ou biblioteca, no entanto o código utilizado para produzir registos de logging estará disperso por todo o código. Através da utilização de AOP nós podemos definir blocos de código (aspectos) para diferentes situações de logging e depois apenas necessitamos de referencia-los na zona do código desejada. Isto não só torna o código mais legível, mas também facilita a edição de blocos de código relativos ao logging, uma vez que estes estarão centralizados num único local do código.²⁰

Mas o que é exactamente um “Aspecto”? Quando pensamos em objectos e nos seus relacionamentos normalmente pensamos em relações herança, no entanto esta abordagem tem algumas limitações. Vejamos o seguinte exemplo: uma classe do tipo “Cão” pode ser estendida (através de herança) para subclasses do tipo “Caniche” ou “Dálmata” que têm características diferentes. Até aqui tudo bem, mas imaginemos que identificamos uma nova característica do tipo “CãoObediente”, certamente nem todos os cães são obedientes, por isso a classe “Cão” não pode conter este comportamento. Neste caso podemos olhar para a obediência como um aspecto que nós aplicamos a qualquer tipo de cão que seja considerado obediente. No que diz respeito ao software, o AOP permite-nos aplicar aspectos que alteram o comportamento dos objectos ou classes independentemente da hierarquia de heranças e estes aspectos podem ser aplicados durante a compilação ou durante a execução das aplicações.

De forma a melhor compreender o conceito de AOP e a sua utilização convém esclarecer os seguintes conceitos:²¹

- Joinpoint – É um ponto ou uma zona do código facilmente detectável. Um bom exemplo de um *joinpoint* é uma zona no código onde é invocado um método;
- Pointcut – Refere-se à forma de identificar um *joinpoint*, seja ela por configuração ou programática.
- Advice – Cada *pointcut*, contém um *advice*, ou seja, contém uma acção que deve ser despoletada quando é atingido um *joinpoint*. Por norma um *advice* traduz-se na invocação de um novo método;
- Mixin – Refere-se à inclusão de uma instância de uma nova classe na classe original por forma a introduzir um novo comportamento.

O nosso objectivo aqui não é aprofundar muito sobre a temática do AOP, mas sim demonstrar que esta é uma técnica que pode ser bastante útil na implementação de mecanismos de logging. No Apêndice 7 – Exemplo de utilização de AOP deste trabalho

²⁰ Informação retirada dos artigos “Aspect-Oriented programming” em Wikipedia.com e “A look at aspect-oriented programming” em Ibm.com.

²¹ Informação retirada do artigo “Aspect-Oriented programming” no MSDN da Microsoft.

apresentamos um exemplo de como AOP pode ser útil na implementação de um simples mecanismo de logging antes e após a invocação de um método.

3.7. Logging e “Caller Info Attributes”

Um dos aspectos importantes para o logging é perceber onde ocorreu um determinado evento e quanto mais detalhada e precisa for a informação relativa à localização do evento melhor. Este aspecto não é propriamente uma novidade para quem está familiarizado com o logging, no entanto este é um aspecto muitas vezes menosprezado pelo programador, que ou não inclui informação fiável sobre a localização do evento ou utiliza técnicas para a obtenção dessas informação que têm impacto desnecessário na performance das aplicações (como é o caso do uso de *reflection* ²² na definição dos parâmetros da mensagem de log).

Durante o nosso estudo do estado da arte depara-mo-nos com um funcionalidade da versão 4.5 do .Net que permite aceder programaticamente a informação sobre *caller* de um determinado método, mais especificamente, o caminho completo para o ficheiro, o nome do caller e até a linha do código onde o método foi chamado. Esta funcionalidade chama-se “Caller Information”, é implementada tirando partido dos parâmetros opcionais da versão 4.0 do C# e é uma funcionalidade que é executada durante a compilação.²³

O cenário ideal para aplicação desta funcionalidade é exactamente o logging e inclusão de informação sobre a localização dos eventos nas mensagens de log. Infelizmente, esta funcionalidade não pode ser utilizada no nosso projecto, uma vez que devido à natureza da maioria dos projectos da CSW, o subsistema de logging foi implementado para a versão 3.5 do .Net.

²² Mais informações sobre reflection nos artigos “Reflection” no MSDN e “Reflection in C# Tutorial” em [Codeproject.com](http://codeproject.com).

²³ Mais informação sobre este assunto nos artigos “Extended Logging with Caller Info Attributes” em [Exceptionalcode.wordpress.com](http://exceptionalcode.wordpress.com) e “Caller Information” no MSDN.

Capítulo 4

Metodologias de Abordagem

Neste capítulo iremos apresentar e detalhar os métodos de abordagem adoptados para a realização do nosso trabalho. Nomeadamente, medições de performance de técnicas e abordagens de desenho e implementação, com o intuito de determinar o impacto destas na produção de log; a aplicação dos processos de desenvolvimento de software da CSW, que incluem a criação dos documentos de especificação de requisitos e arquitectura de software; e ferramentas de apoio ao desenvolvimento de software e gestão de projectos.

4.1. Medições de Performance

De modo a detectar possíveis melhorias nos processos de implementação de mecanismos de logging, foram feitas medições sobre algumas das operações mais comuns relacionadas com logging. Entre as operações seleccionadas encontram-se as operações de concatenação de *strings* (indispensáveis para a construção das mensagens de log) e as operações de obtenção e formatação do timestamp a ser apresentado nas mensagens de log.

Para efectuar as medições foi utilizada uma ferramenta de benchmarking de testes de performance para a linguagem .Net, chamada ASPerfBench²⁴, desenvolvida por André Dias Alves de Carvalho, “Senior Engineer” na Critical Software. Esta ferramenta permite importar uma DLL e executar métodos públicos, sem parâmetros, marcados com o atributo [TestMethod] nela contidos. Cada ficheiro possui um método de controlo vazio, que irá servir de comparação para os outros métodos medidos. Foi utilizada a seguinte metodologia:

- Cada método foi executado 100.000 vezes de forma sequencial;
- Após a chamada de cada método a cache do processador é limpa para que cada medição fosse o mais precisa possível;
- Foi atribuída prioridade máxima às *threads* lançadas para a execução das medições;
- Para evitar carregamentos externos, como importação de DLLs, foi realizado um primeiro teste com menos interações e que não fez parte das medições.
- Durante a execução das medições foram fechadas todas as aplicações não necessárias, foi desligado o acesso à Internet, de forma a assegurar que a execução das medições não sofresse interferências externas.

A imagem seguinte mostra-nos o aspecto da ferramenta de benchmarking, com as opções de configuração de medição no canto superior direito.

²⁴ Mais informação sobre esta ferramenta na página “ASPerfBench” em Tiddlyspace.com.

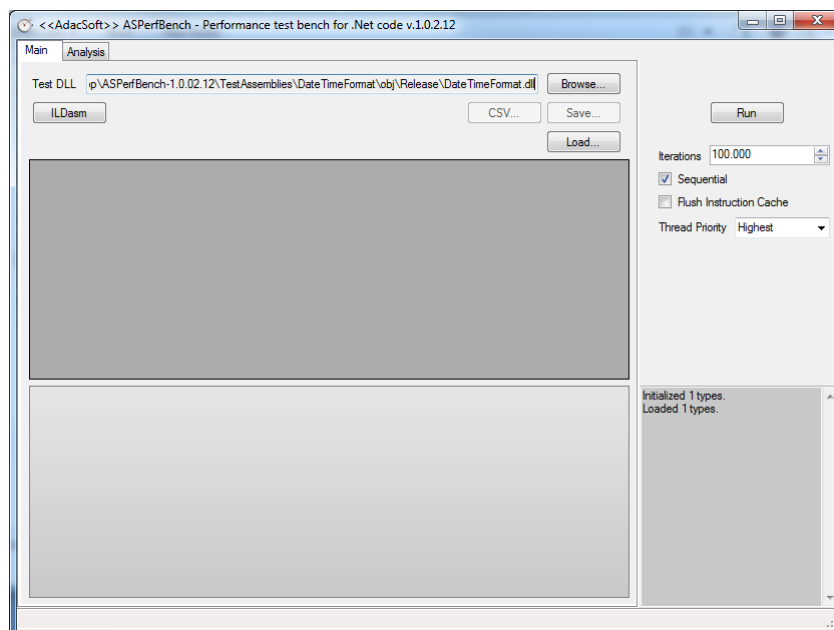


Figura 6 – ASPerfBench (Ferramenta de Benchmarking)

As medições foram realizadas num ambiente (computador) com as seguintes características:

- Marca: DELL Vostro
- Processador: Intel(R) Core(TM)2 Duo CPU T6570 @ 2.10GHz
- RAM: 4,00 GB (3,46 GB utilizáveis)
- Sistema Operativo: 32-bit Windows 7

Importa ainda salientar que o principal objectivo destas medições é fazer um estudo comparativo que nos permita identificar quais as abordagens com melhor ou pior performance. Neste estudo não é vital a precisão dos valores, mas sim na consistência das medições.

4.2. Desenvolvimento de Software

Para a fase de desenvolvimento de software deste projecto foi usado o Processo de Desenvolvimento de Software (SDP) da CSW²⁵ como metodologia de abordagem. A adopção do SDP garante a qualidade do projecto uma vez que se trata de um processo comprovado e utilizado em todos os projectos da empresa com pequenos ajustes conforme a natureza dos projectos. O SDP descreve as fases de desenvolvimento, processos e organização do ciclo de vida do projecto, fornecendo todas as informações relevantes para a equipa do projecto. Entre as várias tarefas que constituem este processo e para além da implementação propriamente dita convém salientar a elaboração dois documentos de extrema importância para o projecto (SRS e SAS), uma vez que servem de base para todo o projecto e vão influenciar todas as tarefas relacionadas com a implementação e desenvolvimento de software.

²⁵ Descrito no Apêndice 2 deste trabalho.

4.3. Ferramentas de Apoio

WinCVS - Sistema de Controlo de Versões

No desenvolvimento de software, bem como em outras áreas onde a edição de ficheiros é constante, é importante manter um registo das várias versões para que seja mais fácil recuperar informação.²⁶ ²⁷ Já imaginou estar a escrever um documento e chegar à conclusão que afinal o parágrafo que escreveu anteriormente e que entretanto já apagou afinal estava correcto? Agora imagine que o seu editor de texto não disponibiliza a função “Undo”? É aqui que entram os sistemas de controlo de versões (CVS), e as funcionalidades de “Undo” e “Redo” presentes hoje em dia na maioria das aplicações não são mais do que simples implementações deste conceito. No âmbito do desenvolvimento de software, a principal função de um sistema de controlo de versões é possibilitar a reposição de um momento do desenvolvimento, o que implica um contexto coerente de vários ficheiros.

O CVS permite não só controlar todas as modificações num conjunto de arquivos, mas permite também que vários programadores (potencialmente separados no espaço e/ou tempo) possam trabalhar sobre os mesmos arquivos sem conflitos. No entanto para que a colaboração concorrential entre programadores funcione correctamente é necessário obedecer a algumas regras e seguir algumas boas práticas, que serão descritas mais à frente.

A nossa intenção neste capítulo passa por explicar a utilização do CVS do ponto de vista do programador (iremos deixar de parte todas as considerações relacionadas com a instalação e configuração de sistemas de CVS, utilizando a ferramenta WinCVS com exemplo).²⁸

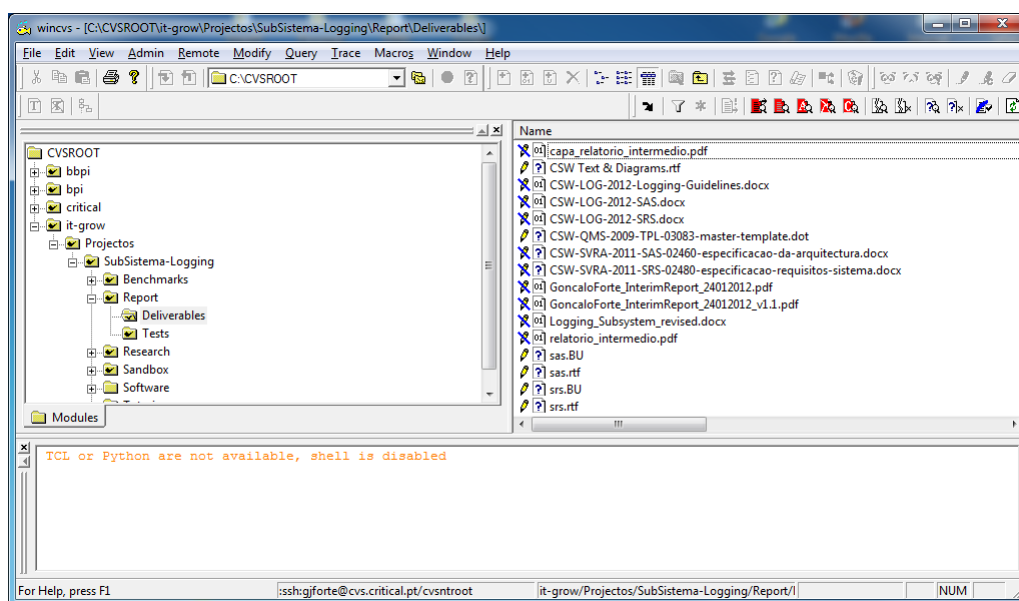


Figura 7 – WinCVS: Sistema de Controlo de Versões

O WinCVS dá-nos acesso ao repositório de ficheiros do projecto e disponibiliza acesso a uma vasta lista de operações. Entre as operações mais comuns para o programador encontram-se as seguintes:

²⁶ “Pragmatic Version Control using CVS” – The Pragmatic Programmers, de Thomas D. e Hunt A.

²⁷ “Concurrent Versions System” – en.wikipedia.org

²⁸ “WinCVS instruction” – sourceforge.net

- Adicionar – Esta é a operação mais comum do CVS. Como o próprio nome indica trata de adicionar novos ficheiros a um repositório de CVS. Só porque um ficheiro se encontra na directoria do projecto não significa que faça parte do repositório, é necessário adicioná-lo e confirmar essa adição através da operação “commit”;
- “Commit” – A operação “commit” serve para confirmar/guardar qualquer iteração com o CVS. Após adicionar um ficheiro é necessário confirmar, após fazer uma alteração é necessário confirmar. Um aspecto importante na confirmação de qualquer operação é a atribuição de uma descrição detalhada sobre a alteração a confirmar. Só desta forma será possível navegar pelas várias versões dos ficheiros e perceber qual a versão que nos interessa. No contexto específico do desenvolvimento de software a regra é fazer “commits” sempre que possível, ou seja, sempre que tenhamos um bloco de código correcto e coeso e não acumular blocos e código, por forma a simplificar a rastreabilidade das alterações;
- Reservar – É uma das operações mais importantes no que diz respeito à colaboração concorrential entre programadores. Reservar um ficheiro significa que apenas esse utilizador deverá editar o ficheiro, eliminando assim potenciais conflitos entre programadores. No entanto, o programador deve libertar o ficheiro a partir do momento em que já não esteja a trabalhar sobre ele. Na realidade, mesmo que um programador não reserve o ficheiro em que vai trabalhar e estejam dois ou mais programadores a trabalhar sobre o mesmo ficheiro ao mesmo tempo, o CVS apresenta as alterações inseridas por cada programador. O problema será fazer o “merge” de todas as alterações sem comprometer a integridade do ficheiro. Esta operação é essencial quando temos ficheiros não “mergeable” como binários, ficheiros auto-gerados ou ficheiros de persistência de ferramentas CASE²⁹;
- Libertar – Tal como foi referido, o programador deve libertar a reserva que fez sobre o ficheiro a partir do momento em que já não esteja a trabalhar sobre ele;
- “Update” – Operação responsável por actualizar ficheiros ou conjuntos de ficheiros. Quando trabalhamos em equipa convém certificarmo-nos que estamos sempre a trabalhar com as versões mais actualizadas de todos os ficheiros. Por norma, é após um *update* que são detectados conflitos em ficheiros que não foram reservados e que foram editados por mais do que um programador;
- “Checkout” – Significa descarregar todo o conteúdo do repositório do CVS para uma determinada pasta.

Para este trabalho, todos os documentos e ficheiros foram adicionados a um directório de CVS criado especificamente para este projecto e foram tratados de acordo com as orientações definidas em cima.

WISE – Relatório de esforço

A avaliação do estado de um projecto de software é uma das formas usadas pela gestão para adquirir conhecimento sobre o projecto, conhecimento esse que terá impacto sobre os cálculos financeiros de diversas actividades, projectos e, no final, nas finanças da própria empresa. Esta avaliação é feita através da análise de parâmetros específicos do projecto como o trabalho desenvolvido, as tarefas atribuídas, o esforço despendido, os recursos utilizados e os custos despendidos, pelo que é de extrema importância que se tenha um registo preciso e detalhado de cada um destes parâmetros. No caso particular do registo de

²⁹ Ferramentas CASE (Computer-Aided Software Engineering), referem-se a ferramentas de auxílio à engenharia de software. Mais informação no artigo “Computer-Aided Software Engineering” em Wikipedia.org.

esforço, trata-se de um artefacto indispensável para a avaliação dos custos do projecto. No entanto, pode ser igualmente útil como ferramenta de controlo do progresso de tarefas. A análise dos registos de esforço de um projecto permite à gestão avaliar, de uma forma mais precisa, não só o custo total do projecto, mas também os custos específicos de cada etapa ou módulo do projecto e estes dados são extremamente úteis durante a fase de planeamento de outros projectos. Do ponto de vista do programador, tal como referido anteriormente, o registo do esforço despendido em cada tarefa é igualmente útil para controlar o progresso das suas tarefas, pois este registo pode ser confrontado com o planeamento do projecto por forma a averiguar se está dentro dos prazos.

Na Critical Software o registo de esforço é feito através da plataforma WISE (Enterprise Information System), uma plataforma desenvolvida internamente com o objetivo de contribuir de forma significativa para a consistência e melhoria da gestão de informação da empresa. O módulo “Tasks” (Tarefas) do WISE é responsável por registar o esforço relacionado com as tarefas realizadas por cada trabalhador (também conhecido como o relatório 15-5³⁰), e consiste na atribuição de esforço para Projectos (através de *workpackages*), actividades, propostas ou outras acções importantes o suficiente para terem um registo de esforço detalhado. A pessoa responsável por um pacote de trabalho (o Gestor do Projecto ou o proprietário de um Centro de Custo) determina a granularidade das tarefas.

Tal como referido anteriormente, o esforço atribuído a cada tarefa deve ser o mais real possível, uma vez que este registo terá impacto directo em toda a actividade da empresa.

Tarefas

Quarta-feira, 27 Junho 2012

Figura 8 – Wise: Registo de esforço

A imagem anterior mostra-nos a consola de registo de esforço do WISE. Esta consola é constituída por três áreas distintas: a área de definição de esforço, onde são indicadas as tarefas, a descrição do trabalho realizado, o tempo despendido e a actividade a que pertence; o calendário, onde o utilizador pode navegar pelas datas e ter uma noção geral sobre os dias de trabalho, feriados e férias atribuídas; e por fim a área de estatística, onde são apresentados

³⁰ Os relatórios 15-5 foram inventados por Yvon Chouinard, CEO da empresa Patagonia, e descritos por Paul Hawken no seu livro “Growing a Business”. A ideia geral é a de que todas as semanas os colaboradores devem preencher um relatório que não demore mais de 15 minutos a preencher ou 5 minutos a ler. O processo é descrito por Hawken como sendo constituído por três partes:

- Descrição simples sobre o que o colaborador fez durante a semana;
- Descrição clara sobre o estado de espírito do colaborador e do ambiente que o rodeia;
- Uma ideia que possa melhorar o trabalho do colaborador, do departamento ou da empresa.

dados sobre o esforço realizado pela trabalhador. Podemos ver os valores mínimos esperados pela empresa e os valores alcançados pelo trabalhador.

Durante a realização deste trabalho o registo de esforço foi preenchido ao final de cada dia de trabalho, não incluindo para este efeito os esforços despendido fora do horário laboral estabelecido para o estágio curricular em que este trabalho se insere.

Enterprise Architect – Desenho e Documentação de Software

O Enterprise Architect é uma ferramenta rápida e intuitiva de desenho de UML e análise de negócio para modelação, documentação, “reverse-engineering”, construção e manutenção de sistemas de software orientados a objectos.

Esta foi a ferramenta utilizada para desenhar, modelar e especificar todo o subsistema de logging e foi a partir desta que foram gerados os documentos de especificação de requisitos e arquitectura de software, incluindo todo o texto de contextualização, especificação e descrição do projecto bem como a maioria das figuras apresentadas neste trabalho.

A figura seguinte apresenta a estrutura criada dentro do EA para suportar o desenho do subsistema de logging. Este é apenas um exemplo de uma possível estrutura para o desenho de um projecto de software, baseada em estruturas de outros projectos da CSW.

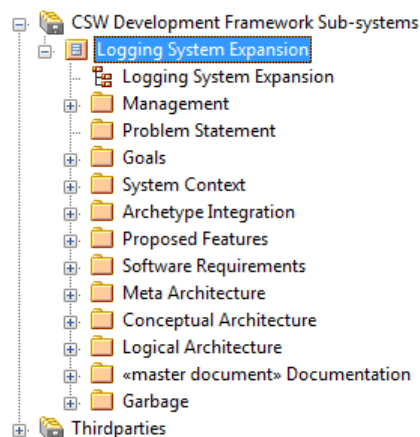


Figura 9 – Estrutura de desenho do Subsistema de Logging

No nosso entender trata-se de uma ferramenta essencial para o desenvolvimento de software que, para além de facilitar a modelação de esquemas e diagramas que suportam a especificação de um projecto de software, facilita ainda a edição de conteúdos, a reutilização de elementos e a verificação e validação de especificações através de mecanismos de rastreabilidade.

Visual Studio 2010 – Plataforma de Desenvolvimento

O Microsoft Visual Studio é uma plataforma de desenvolvimento de software direccionada para o desenvolvimento de aplicações para frameworks como o Windows, o .Net ou o Silverlight. É um ambiente integrado desenvolvimento (IDE) que vem equipado com um conjunto de ferramentas muito úteis para o programador, como é o caso do IntelliSense ou do debugger e suporta uma variedade enorme de linguagens de programação desde o C#, ao Visual Basic, apassando pelo ASP.Net entre outros.

Para a implementação do subsistema de logging foi criado no VS um projecto do tipo “Class Library para a versão 3.5 do .Net, que após compilado disponibilizará funcionalidades de logging a outros projectos através da importação de uma única DLL.

Capítulo 5

Trabalho Realizado e Resultados

Neste capítulo apresentamos o trabalho realizado e os resultados obtidos durante a realização deste trabalho. Começamos por apresentar e descrever os documentos de especificação de requisitos e arquitectura de software, de seguida apresentamos as medições realizadas e que serviram de suporte para a especificação e implementação do sistema, apresentamos as *guidelines* que ainda não foram referidas nos capítulos anteriores e terminamos com uma descrição da implementação e expansão do subsistema de logging.

5.1. Especificação do Sistema

A elaboração da especificação técnica tem como objectivo dar "resposta" às necessidades dos clientes. Esta contém uma definição precisa e coerente de funções, desempenhos, calendarizações, esforço e plano de implementação para todos os níveis do software a ser desenvolvido. Os requisitos e especificações nela contidos devem ser viáveis, completos e consistentes, garantindo que o cliente e a equipa de desenvolvimento os interpretam da mesma forma.

Tendo em conta os objectivos principais do projecto, também a especificação deve ser reutilizável e expansível, pelo que serão tidos em conta alguns aspectos na elaboração dos documentos de especificação, tais como, a nomenclatura de *features* e requisitos, que irão conter o termo “LOG”. Desta forma, em caso de reutilização destes elementos será mais fácil de identificar e evitar “colisões” entre elementos que pertençam ao projecto do subsistema de logging. As features e requisitos do projecto serão igualmente numeradas por forma a facilitar a rastreabilidade deste elementos ao longo dos vários documentos e fases do projecto.

De acordo com o processo de desenvolvimento de software de CSW, a especificação técnica será dividida em dois documentos: o documento de especificação de requisitos de software e o documento de especificação da arquitectura do software.

Documento de Especificação dos Requisitos de Software

O documento de especificação de requisitos de software é uma descrição completa do sistema a ser desenvolvido. O SRS contém um conjunto de requisitos arquitecturais, que impõem restrições de design e implementação, e requisitos funcionais que descrevem as funcionalidades do sistema e a forma como este interage com o utilizador.

Existem diferentes abordagens para a organização de um SRS³¹, mas para este projecto usámos uma abordagem baseada na versão simplificada do processo de desenvolvimento de software da CSW, descrita no Apêndice 2 – Processo de Desenvolvimento de Software da CSW deste trabalho. Esta abordagem foca a análise das “features” do projecto e dos requisitos de software necessários para satisfazer as necessidades do projecto.

Features são uma descrição de nível mais abstracto dos requisitos do projecto e são normalmente fornecidas pelo cliente, quando este descreve as características que deseja para

³¹ Como a que é sugerida por Andrew Stellman e Jennifer Greene, no capítulo 6.3 Software Requirements Specification, do livro “Applied Software Project Management”.

o software. Neste caso, tratando-se de um projecto interno da CSW, não existia um cliente, pelo que o levantamento das features do sistema foi feito através da consulta e interacção com os responsáveis pelo projecto. Esta opção na análise das features de um projecto é aconselhável de forma a reduzir a complexidade e o detalhe da análise, servindo igualmente de contexto para a elaboração da análise dos requisitos do sistema, que por sua vez deve ser muito mais profunda. As features são descritas no presente (tempo verbal), uma vez que tal facilita a verificação da implementação das mesmas.

A secção sobre a análise das features do sistema foi dividida em três partes: features arquitecturais, onde são descritas as características do sistema que terão impacto na arquitectura do software; features funcionais, que descrevem as funcionalidades e comportamentos do sistema; e features para as guidelines onde se descrevem os conceitos e os pressupostos relacionados com a temática do logging e que servem de suporte para a elaboração deste trabalho.

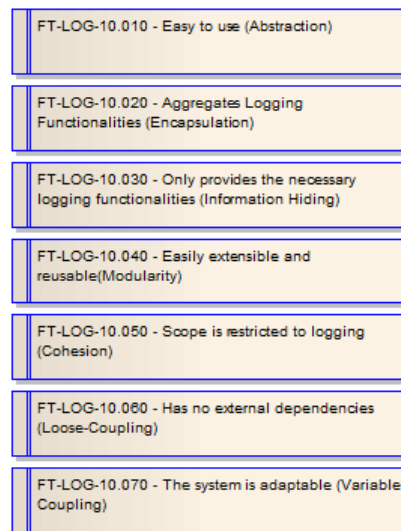


Figura 10 – SRS: Features Arquitecturais

Features Arquitecturais:

- Easy to use (Abstraction) – No âmbito da engenharia de software abstrair significa distinguir a informação necessária,^{32 33} ou seja, este sistema facilita o uso de diferentes frameworks de logging, através da abstracção das suas APIs;
- Aggregates logging functionalities (Encapsulation) – Encapsulamento diz respeito ao acto de englobar um ou mais itens dentro de um mesmo recipiente.^{19 20} Isto significa que o sistema engloba as uma ou mais frameworks de logging;
- Only provides the necessary logging functionalities (Information Hiding) – Este conceito é muitas vezes confundido com o de abstracção. No entanto abstracção é uma técnica utilizada para identificar que informação deve ser abstraída, enquanto “Information Hiding” uma técnica utilizada para abstrair ou esconder essas informação.¹⁹ Desta forma podemos dizer que o sistema apenas disponibiliza as funcionalidades estritamente necessárias para fazer logging;
- Easily extensible and reusable (Modularity) – Modularidade é uma propriedade de sistemas decompostos em módulos coesos e sem dependência.^{20 34} Com isto

³² “Abstraction, Encapsulation, and Information Hiding” de Edward V. Berard.

³³ “Object-Oriented Analysis and Design with Applications - 2nd Edition” de Grady Booch

³⁴ “Object-Oriented Software Construction – 2nd Edition” de Bertrand Myer.

queremos dizer que se torna fácil adicionar novas frameworks de logging ao sistema e que a reutilização do sistema em diferentes projectos é igualmente simples;

- Scope is restricted to logging (Cohesion) – A coesão mede o grau de familiaridade entre os elementos do mesmo módulo.^{20 35} Tendo isto em consideração, é possível afirmar que este sistema tem um nível de coesão muito alto pois apenas lida com logging.
- Has no external dependencies (Loose-Coupling) – Por “Coupling” entendemos o grau de associação e dependência entre dois módulos.^{20 22} Portanto um sistema “loose-coupled” é um sistema sem dependências de sistemas externos. No nosso caso, o sistema não tem qualquer dependência externa para além das respectivas frameworks de logging;
- The system is adaptable (Variable-Coupling) – Trata-se de uma extensão do conceito de coupling, que nos diz que as dependências do sistema podem ser variáveis. Isto significa que, apesar de o sistema lidar com várias frameworks de logging externas, este deve apenas depender das funcionalidades da framework de logging escolhida.



Figura 11 – SRS: Feautures Funcionais

Features Funcionais:

- Is capable of dealing with different external logging frameworks – O sistema está desenhado e preparado para utilizar diferentes frameworks de logging externas;
- Works with Apache Log4net – Significa que o sistema está preparado para usar a framework de logging Apache Log4net;
- Works with Microsoft Enterprise Library Logging Application Block – Suporta a utilização da framework de logging Microsoft Enterprise Library Logging Application Block;
- Works with NLog - Está igualmente preparado para usar a framework de logging NLog;
- Has low impact on applications performance – Esta feature garante que o sistema utiliza as melhores abordagens de programação, com vista à redução do impacto do logging na performance das aplicações;
- Standardizes logging production – Esta feature garante que as mensagens de log produzidas pelo sistema têm estruturas e formatos normalizados;
- Gives na alternative to the standardized logging production – Uma vez que os projectos podem ter necessidades diferentes das que nós considerámos, o sistema permite ao programador criar as suas próprias estruturas para as mensagens de log;

³⁵ “Cohesion and Coupling” de Jeremy Miller em MSDN Magazine – Patterns in Practice.

- Is suitable for various levels of operation – Isto significa que o sistema disponibiliza mecanismos para gerir a verbosidade do log;
- Allows logging events categorization – O sistema permite classificar as mensagens de log de acordo com a sua natureza, tipo ou severidade;
- Is compatible with Microsoft Event Viewer – O sistema é compatível com a utilização do Microsoft Event Viewer, independentemente da framework de logging externa que esteja a ser usada.

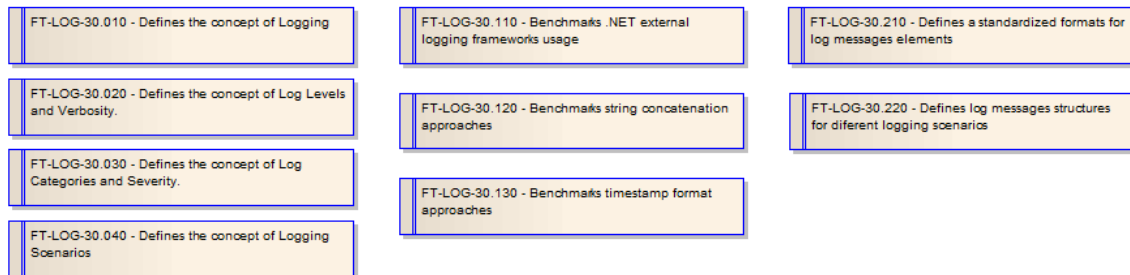


Figura 12 – SRS: Features para as guidelines (Conceitos, Performance e Qualidade)

Features para as Guidelines:

- Defines the concept of Logging – O documento de guidelines define o conceito de logging, identifica as suas principais características e distingue este conceito dos conceitos de tracing e logging de auditoria;
- Defines the concept of Log Levels and Verbosity – Dentro do contexto do logging é definido o conceito de níveis de log e verbosidade, salientando a sua importância e utilização;
- Defines the concept of Log Categories and Severity – É definido o conceito de categorias de log e severidade, explicando quando devem ser usados e quais as estruturas para as mensagens de log aconselhadas;
- Defines de concept of Logging Scenarios – Neste documento é igualmente definido o conceito de cenários de logging, que ajudam o programador a identificar as diferentes necessidades de logging para cada cenário;
- Benchmarks external logging frameworks usage – São realizadas medições de performance sobre as diferentes frameworks de logging externas utilizadas, de forma a saber quais as frameworks e as suas opções de utilização mais dispendiosas em termos de performance;
- Benchmarks string concatenation approaches – A concatenação de strings é um processo essencial na escrita de mensagens de log. Como tal, são realizadas medições de performance sobre diferentes abordagens de concatenação de strings;
- Benchmarks timestamp format approaches – À semelhança da concatenação de strings, também a formatação do elemento de timestamp é um processo essencial, dada a importância da existência de um elemento de timestamp na estrutura de uma mensagem de log. Desta forma, são igualmente realizadas medições de performance sobre diferentes abordagens de formatação do elemento de timestamp;
- Defines a standardized formats for log messages elements – O documento de guidelines define formatos normalizados para os elementos que compõem as mensagens de log;

- Defines log messages structures for different logging scenarios – O documento define também estruturas para as mensagens de log, de acordo com o cenário e a categoria da mensagem de log.

A análise das features do sistema dá-nos então um ponto de partida e um contexto para a fase seguinte do documento de especificação de requisitos de software, a fase de análise dos requisitos de software.

Requisitos de software são o elemento da engenharia de software responsável pela apresentação, análise, especificação e validação das necessidades de um software. Um requisito de software é uma propriedade que deve ser exibida por forma a resolver algum problema do mundo real.³⁶

Mas como é que nós garantimos que a nossa análise dos requisitos de software está em conformidade com as necessidades do cliente e que serve correctamente o propósito de servir de base para as fases seguintes do projecto?³⁷ Mike Mannion e Barry Keepence sugerem uma abordagem inspirada na técnica de definição de objectivos em Gestão de Psicologia, chamada “SMART Requirements”.³⁸ De acordo com estes autores, para garantirmos o sucesso da nossa análise de requisitos de software, devemos garantir que cada requisito levantado é específico (S), mensurável (M), atingível (A), realizável (R) e rastreável (T). Esta é igualmente uma das principais distinções entre features e requisitos.

A especificidade da descrição de um requisito é fundamental. Deve ser simples, deve dizer exactamente o que é suposto fazer, não deve ser ambíguo e deve usar as mesmas terminologias em toda a especificação.

Por mensurável os autores referem-se à possibilidade de verificar se um requisito foi satisfeito. Em algumas metodologias de engenharia de software é comum a descrição dos requisitos incluírem a instruções de teste para verificar se o requisito foi satisfeito.

Consequentemente, ser atingível significa ser fisicamente possível satisfazer as necessidades do requisito de acordo com os conhecimentos ou tecnologias existentes. Alguns requisitos podem estar além dos limites do conhecimento humano, ou terem apenas soluções teóricas, completamente irrealistas.

Ser realizável significa ser possível satisfazer o requisito tendo em conta as restrições sob as quais o sistema e o projecto deve ser desenvolvido, sejam estas físicas, de integração ou de disponibilidade de recursos.

Finalmente, rastreabilidade é a capacidade de rastrear ou seguir um requisito durante a concepção, especificação, design, implementação e teste. Esta característica é importante para que possamos entender a razão da existência do requisito, verificar se o requisito foi satisfeito e garantir que possíveis modificações sejam feitas de uma forma simples, completa e consistente.

De seguida apresentamos a nossa listagem e análise dos requisitos de software para o subsistema de logging. Os requisitos seguem a mesma estrutura de nomenclatura para facilitar a rastreabilidade com as features do sistema.

³⁶ Definição sugerida por Pierre Bourque and Robert Dupuis no capítulo 2 Software Requirements do livro “Guide to the Software Engineering Body of Knowledge”, versão 2004.

³⁷ Recomendações para a elaboração de um SRS: “IEEE Recommended Practice for Software Requirements Specifications”.

³⁸ Artigo de Mannion e Keepence sobre análise de requisitos de software: “SMART Requirements”.

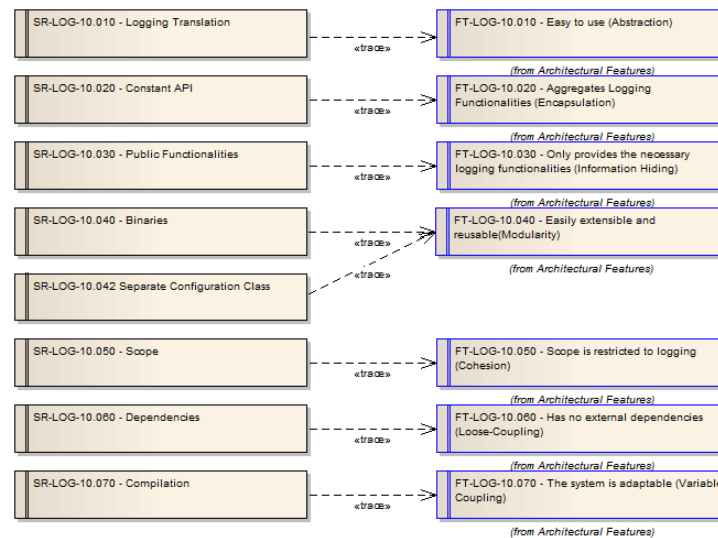


Figura 13 - SRS: Requisitos Arquitecturais

Requisitos Arquitecturais:

- Logging Translation – O sistema deve “traduzir” a utilização das frameworks de logging externas para um conjunto de cinco métodos (um por cada categoria de logging). Cada um destes métodos terá quatro overloads: um para escrever apenas a mensagem, um para escrever a mensagem e enviar o “Event Id”, um para escrever a mensagem e a excepção e por último um que escreva a mensagem, a excepção e envie o “Event Id”;
- Constant API – O sistema será exposto através de uma única API, constante independentemente da framework the logging usada;
- Public Functionalities – Apenas as funcionalidades estritamente necessárias, como a definição do nível de log, os métodos de log e os métodos de concatenação de strings, serão expostos para o utilizador;
- Binaries – Para ser utilizado em outros projectos, o subsistema de logging deve ser compilado numa única DLL;
- Separate Configuration Class – O sistema necessita de obter a configuração para as frameworks de logging através de um ficheiro de configuração. O código para a obtenção destas configurações deve ser colocado numa classe à parte de modo a ser independente da framework de logging externa a ser utilizada;
- Scope – O sistema deve lidar exclusivamente com funcionalidades de logging e as únicas funcionalidades não relacionadas com o logging que serão permitidas serão as funcionalidades de concatenação de strings e formatação de timestamp. Dentro do próprio sistema, cada classe terá as suas responsabilidades, sendo que a classe Logger será responsável pela tradução e disponibilização da API, as classes wrapper serão responsáveis por lidar com as funcionalidades específicas de cada framework de logging externa, e a classe de configuração será responsável por obter a configuração do ficheiro para as frameworks de logging externas;
- Dependencies – O sistema não terá dependências externas para além das dependências das frameworks de logging. Isto significa que para a utilização deste subsistema noutros projectos, apenas serão necessárias duas DLLs, uma para o próprio subsistema de logging e outra referente à framework de logging externa para a qual o sistema foi compilado;
- Compilation – O sistema deve disponibilizar apenas as funcionalidades da API relativas à framework de logging escolhida. Isto será obtido através da utilização de compilação condicional.

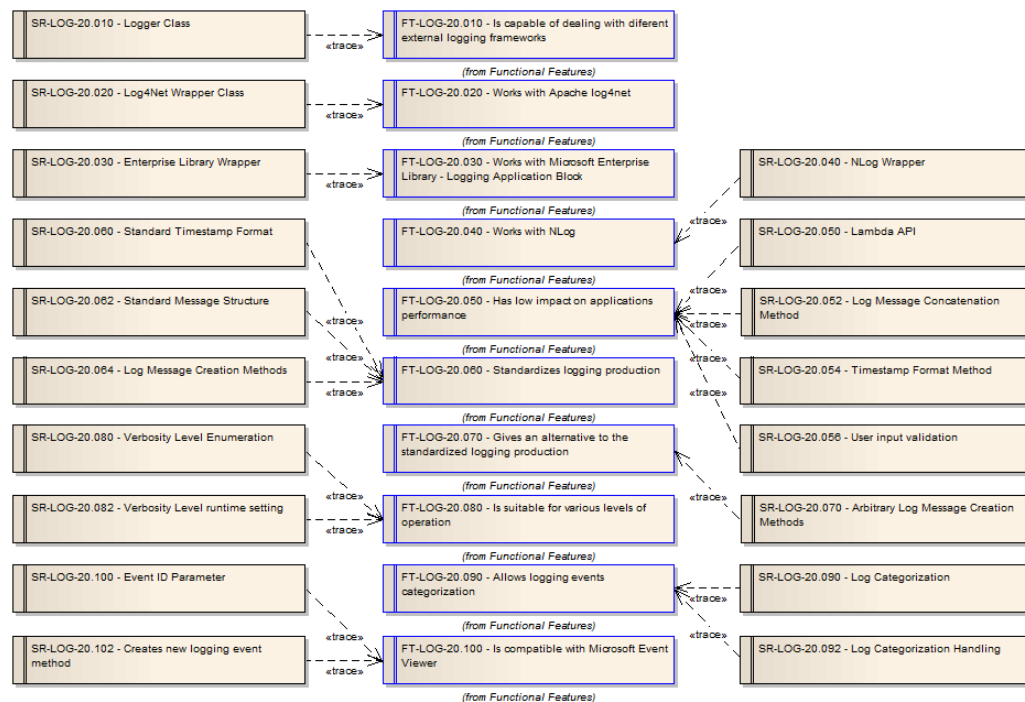


Figura 14 – SRS: Requisitos Funcionais

Requisitos Funcionais:

- Logger Class – O sistema deve implementar uma classe para ser usada como interface para multiplas frameworks de logging, através do encaminhamento de pedidos de log para a respectiva classe wrapper;
- Log4Net Wrapper Class – O sistema deve disponibilizar uma classe que seja responsável pela comunicação com o log4net, versão 1.2.10.0. Esta classe deverá conter funcionalidades para a definição do nível de log e cinco métodos de log (um por categoria de log) com quatro overloads cada um, de acordo com a API do subsistema;
- Enterprise Library Wrapper Class – O sistema deve fazer o mesmo que para o requisito anterior, mas neste caso para a framework de logging externa, Enterprise Library – Logging Application Block, versão 5.0.414.0;
- Nlog Wrapper Class – Este requisito é idêntico aos dois anteriores mas é aplicável à framework de logging externa, Nlog, versão 2.0;
- Lambda API – O sistema deve utilizar expressões Lambda na sua API de modo a torná-la mais eficiente. Mais concretamente, serão usadas expressões lambda para realizar a concatenação dos parâmetros das mensagens de log. Desta forma serão poupados processamentos desnecessários caso a respectiva mensagem de log não seja realizada devido ao seu nível de log;
- Log Message Concatenation – O sistema deverá usar as melhores abordagens para a concatenação dos parâmetros das mensagens de log. Neste caso, de acordo com as medições realizadas, a melhor abordagem será a concatenação de strings através da sua “soma”, ou seja, através da utilização do caracter ‘+’;
- Timestamp Format Method – O sistema deverá usar a abordagem para a formatação do elemento timestamp que tenha menor impacto na performance. De acordo com as nossas medições, a melhor abordagem é a abordagem retirada do artigo de Alois Kraus “Performance Quis: Fastest way to format time”, descrita no Apêndice 3 – Benchmark de formatação de Timestamp deste trabalho;

- User Input Validation – Um a vez que a validação dos parâmetros fornecidos pelo utilizador obrigaria a gastos adicionais de performance, o sistema nunca deve validar estes parâmetros;
- Standard Timestamp Format – De forma a normalizar a produção de registos de log o sistema deve possuir um formato ‘standard’ para o elemento timestamp das mensagens de log. Tendo em conta a norma ISO 8601 o elemento timestamp deverá ter o seguinte formato: “yyyy-MM-dd HH:mm:ss.fff”, sendo que ‘yyyy’ representa o ano, ‘MM’ o mês, ‘dd’ o dia, ‘HH’ a hora, ‘mm’ os minutos, ‘ss’ os segundos e ‘fff’ os milissegundos. Caso o projecto onde o subsistema de logging esteja inserido possua grandes requisitos de performance, este formato pode ser substituído por um formato mais compacto (com a utilização de menos separadores): “yyyyMMdd HHmmss.fff”;
- Standard Message Structures – O sistema deve possuir estruturas de mensagens normalizadas de acordo com o cenário e categoria de logging. As estruturas sugeridas encontram-se no documento de “Logging Guidelines”;
- Log Message Creation Methods – O sistema deve disponibilizar métodos de criação de mensagens de log que implementem a formatação de timestamp, a concatenação de strings e incluam as estruturas de mensagens normalizadas;
- Arbitrary Log Message Creation – Para além dos métodos de criação de mensagens de log predefinidos e normalizados o sistema deve ainda dar a liberdade ao utilizador de escolher a estrutura e os elementos para a sua mensagem de log;
- Verbosity Level Enumeration – O sistema deve possuir uma enumeração para os níveis de log com os seguintes níveis e valores: “High” (100), “Medium” (50), “Low” (1) e “Off” (0). Esta enumeração permitirá definir o nível de log para cada mensagem e o próprio nível de verbosidade do sistema;
- Verbosity Level Runtime Setting – O sistema deve permitir a definição e modificação do nível de verbosidade do sistema durante a execução das aplicações. Esta funcionalidade será disponibilizada através de uma variável pública para o efeito;
- Log Categorization – O sistema deve permitir a categorização das mensagens de log de acordo com a sua importância ou severidade. Para tal o sistema deve implementar métodos de criação de mensagens para cada uma das categorias de log. As categorias são: “Debug”, “Info”, “Warn”, “Error” e “Fatal”;
- Log Categorization Handling – Na eventualidade de as frameworks de logging não possuírem uma correspondência directa para as categorias de logging apontadas, o sistema deve inseri-las nas mensagens de log como sendo um parâmetro de texto normal;
- Event Id Parameter – Algumas frameworks de logging como o log4net e o Nlog, não permitem a definição do parâmetro “Event Id” nas suas APIs, com tal devem ser feitos overloads aos métodos das APIs destas frameworks de logging por forma a permitirem a inclusão deste parâmetro;
- Logging Event Method – Para permitir o overload dos métodos de criação de mensagens de log das frameworks que não suportam o parâmetro “Event Id”, é necessário criar um novo método responsável por enviar os pedidos de log para a framework. Este método será idêntico ao método existente, mas passará a incluir o parâmetro “Event Id” nas suas propriedades.

Documento de Especificação da Arquitectura do Software

O objectivo principal de um documento de especificação de arquitectura de software é garantir que um software é desenhado com qualidade e que serve de suporte para a fase de construção do software, respondendo a todos os requisitos de software e promovendo a reutilização e a rastreabilidade. O SAS deve projectar um software que implemente os requisitos de software e que possa ser testado de acordo com estes. Este documento deve descrever, entre vários aspectos, os principais componentes que irão implementar os requisitos de software, as interfaces internas e externas desses componentes ou as estratégias e recursos de reutilização.³⁹

Para este projecto decidimos focar a especificação da arquitectura do software em quatro capítulos: “Contexto do Sistema”, que irá proporcionar uma base de contexto para melhor compreender os requisitos de software e as decisões de arquitectura; “Meta-Arquitectura” onde serão descritas e fundamentadas as principais decisões que irão influenciar o desenho da arquitectura e a estratégia de implementação; “Arquitectura Conceptual” que irá focar a decomposição do sistema identificando os principais elementos, relacionamentos e responsabilidades; e por fim “Arquitectura Lógica” que descreve a organização e integração dos componentes.

Relativamente ao contexto do sistema, decidimos criar diagrama de fluxos de dados para nos dar uma representação gráfica do “fluxo” de dados através do sistema e modelar os principais aspectos do seu funcionamento. Esta representação mostra-nos que tipo de dados temos à entrada e à saída do sistema, a origem e o destino desse dados e onde serão armazenados. Este tipo de DFD é conhecido por “context-level DFD” e é normalmente usado para apresentar o contexto do sistema, pois permite apresentar as interacções entre o sistema e os agentes externos e desta forma dar uma visão geral sobre o sistema. Este diagrama pode depois ser expandido para um DFD de nível 1, fornecendo informação mais detalhada sobre os processos, fluxos de dados, funcionalidades e o armazenamento de dados do sistema.⁴⁰

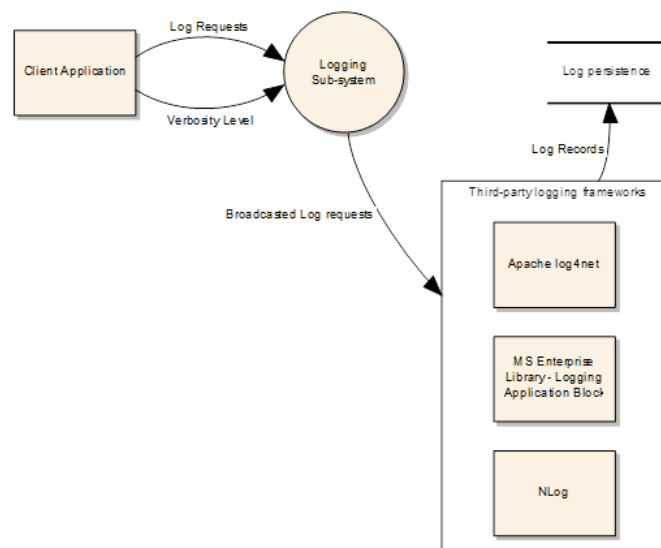


Figura 15 – Contexto do sistema

³⁹ Definição retirada do “Software Design Process” da CSW.

⁴⁰ Informação sobre DFD retirada do capítulo “4. Data Flow Diagrams” do livro “Structured Analysis and System Specification” de Tom Demarco e do artigo “The semantics of Data Flow Diagrams” de Bruza e Van der Weide.

A figura anterior dá-nos uma visão geral do contexto do sistema, mais concretamente, da forma como as “aplicações cliente” interagem com o subsistema de logging e este com as frameworks de logging externas. As aplicações cliente interagem com o subsistema através de pedidos de log ou através da definição do nível de verbosidade do sistema. Por sua vez o subsistema de logging encaminha os pedidos de log para a framework de logging desejada que será responsável por enviar as mensagens de log para os destinos escolhidos (ficheiro de texto, Event Viewer, etc).

O capítulo sobre a meta-arquitectura é um dos mais importantes pois reúne as principais decisões relacionadas com a estratégia de desenho da arquitectura do sistema. Na meta-arquitectura é definido o caminho a seguir, com decisões de alto nível que irão moldar a arquitectura e orientar os programadores. Em primeiro lugar é formulada uma visão arquitectónica baseada numa pesquisa de estilos, padrões e modelos devidamente documentados, em arquitecturas de referência, sejam elas da própria empresa, de competidores ou parceiros, e em literatura sobre o assunto. Com base neste estudo e na experiência da equipa é então formulada a meta-arquitectura. Isto inclui entre muitos aspectos, estilos arquitecturais, conceitos, mecanismos e princípios que irão guiar a equipa durante as próximas fases do projecto.⁴¹

De seguida apresentamos e explicamos as principais decisões relativas à arquitectura do subsistema do logging:

- “Logger” Class – A classe “Logger”, responsável pela interacção/interface entre as aplicações cliente e o subsistema de logging, através da sua API, deverá ser uma classe pública e estática. A utilização de classes estáticas torna a implementação simples e o sistema mais rápido, uma vez que não é necessário criar um objecto para fazer chamadas aos métodos. Posto isto, os métodos responsáveis pela tradução das APIs das frameworks de logging externa devem ser igualmente públicos e estáticos;⁴²
- Log Wrappers – As classes “wrapper” responsáveis por implementar as funcionalidades específicas de cada framework de logging, devem ser públicas e seladas (*sealed*). A propriedade “sealed” do C# impede a derivação destas classes, o que significa que as funcionalidades das classes não serão usadas de forma incorrecta;⁴³ Estas classes devem ainda ter uma instância privada, estática e “readonly” que servirá de *façade* única da classe, um tipo (SystemType) para servir de identificação das classes, e por fim um construtor estático que diga explicitamente ao compilador do C# para não marcar o tipo do construtor como “beforefieldinit”⁴⁴;
- Verbosity Level Runtime Setting – Para a definição do nível de verbosidade do sistema em *runtime* será necessário a implementação do padrão *singleton*⁴⁵. Isto implica criar uma classe privada e selada para a verbosidade, que possuirá uma instância pública, estática e “readonly”, e uma referência pública e volátil⁴⁶ da enumeração do nível de verbosidade que será usada para definir o nível de log do sistema. A utilização de um *singleton* não é *thread-safe*, mas a utilização da propriedade “volatile” assegura essa segurança;

⁴¹ Definição retirada do artigo “Meta-Architecture” de Malan e Bredemeyer e do SDP da CSW.

⁴² Mais informação no artigo “Static Classes and Static Class Members” no MSDN da Microsoft.

⁴³ Mais informação no artigo “Abstract and Sealed Classes and Class Members” no MSDN da Microsoft.

⁴⁴ Esta opção traz ganhos de performance, tal como está explicado no artigo “Get a Charge From Statics with Seven Essential Programming Tips”.

⁴⁵ O padrão singleton é explicado neste artigo “Singleton pattern” da Wikipedia.

⁴⁶ Mais informações no artigo “Volatile (C# Reference)” no MSDN da Microsoft.

- String Concatenation Methods – De acordo com as medições realizadas, o sistema deve usar a abordagem de soma de *strings* através do carácter ‘+’ para concatenar os elementos das mensagens de log, pois esta é a abordagem que apresenta melhores valores de desempenho;
- Timestamp Format – Por forma a ter o menor impacto na performance do subsistema de logging, deve ser usada a abordagem “FastDateTimeStr” (descrita no Apêndice 3 – Benchmark de formatação de Timestamp) para formatar o elemento de *timestamp* das mensagens de log. Em caso de necessidade de melhores valores de performance pode ser usada a mesma abordagem, mas com um formato mais simplificado através da remoção de alguns separadores;
- Lambda Functions – Devem ser usadas funções lambda na construção das mensagens de log para evitar gastos de processamento desnecessários com parâmetros de mensagens que não irão ser registadas devido ao seu nível de log. Desta forma a validação do nível de log será realizada antes do processamento dos parâmetros da mensagem, o que significa que esse processamento só será realizado se o nível de log da mensagem for igual ou inferior ao nível de log do sistema;
- Microsoft Event Viewer Compatibility – Deve ser criado um método privado nas classes “wrapper” referentes a frameworks de logging que não suportem o registo de mensagens de log no Event Viewer, que receba como parâmetros a mensagem, o “Event ID”, o nível de verbosidade da mensagem e a excepção, e que envie um pedido de log para a framework de logging respectiva;
- Conditional Compilation – As directivas de compilação condicional são usadas para incluir ou excluir partes do código num determinado projecto. O ambiente de compilação é definido nas propriedades do projecto, mais especificamente na opção “Build”. O programador deve definir como símbolo de compilação condicional uma das seguintes hipóteses, de acordo com a framework de logging que deseja utilizar: “LOG4NET”, “ENTLIBRARY” ou “NLOG”.

Feita a análise da meta-arquitectura podemos avançar para a arquitectura conceptual do subsistema de logging. A intenção da arquitectura conceptual é dirigir atenção para uma decomposição adequada do sistema, sem aprofundar os detalhes de implementação do interface. Nesta fase são identificados elementos chave da arquitectura do sistema como componentes, relacionamentos e mecanismos arquitecturais. Mecanismos arquitecturais são desenhados para resolver questões transversais que não podem ser localizadas num único componente.⁴⁷ A arquitectura conceptual é ainda uma forma útil de comunicar e explicar a arquitectura de um sistema para audiências não-especializadas, como por exemplo quadros de gestão ou marketing.

⁴⁷ Definição retirada do artigo “Conceptual Architecture Action Guide” da Bredemeyer Consulting.

Para começar a definição da arquitectura conceptual optámos por criar um diagrama de componentes que nos permitisse identificar os principais componentes deste sistema. Os diagramas de componentes são usados para ilustrar estruturas de sistema complexos pois ajudam a descrever os seus componentes e a forma como estes estão ligados entre si.

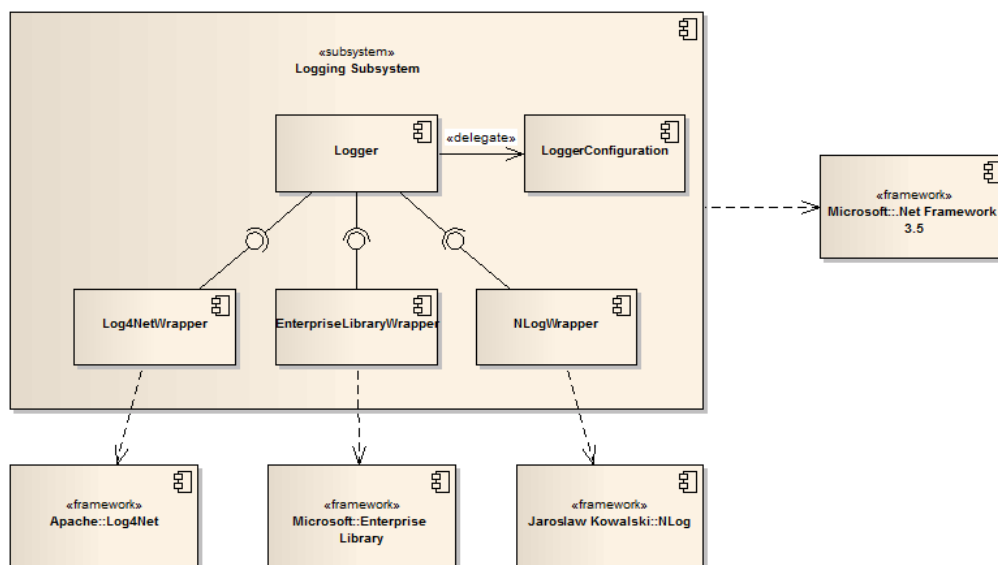


Figura 16 – Diagrama de Componentes

Através deste diagrama conseguimos perceber que o subsistema de logging tem uma dependência directa da framework de desenvolvimento de software .Net 3.5 da Microsoft e que é constituída por vários subcomponentes. O componente “Logger” representa a classe com o mesmo nome e como podemos ver delega a configuração do sistema ao componente/classe “LoggerConfiguration”. Por outro lado podemos ver também que os componentes “Log4NetWrapper”, “EnterpriseLibraryWrapper” e “NLogWrapper” dependem das respectivas frameworks de logging e que disponibilizam as suas funcionalidades ao componente “Logger”.

Outro aspecto importante da análise da arquitectura conceptual é a atribuição das responsabilidades a cada componente. Isto significa que os requisitos devem ser relacionados com os componentes que os irão implementar. Esta análise permite não só orientar os programadores durante a fase de desenvolvimento mas também verificar a realização dos requisitos. Como podemos ver pelo Apêndice 4 – Diagrama de Responsabilidades a atribuição das responsabilidades de cada componente é feita através de conexões do tipo “Realização” entre os componentes relacionados com os requisitos do sistema.

A definição da arquitectura conceptual de um sistema é o ponto de partida para a definição da sua arquitectura lógica, mas também é possível que a arquitectura conceptual sofra modificações durante a criação da arquitectura lógica, uma vez que modelar o comportamento dinâmico de um sistema é uma técnica bastante útil para refinar as responsabilidades e interfaces dos componentes. Para este projecto decidimos incluir na definição da arquitectura lógica do sistema uma descrição sobre o ciclo de operações realizadas pelo subsistema de logging para realizar uma registo de log, recorrendo a um diagrama de sequência. Um diagrama de sequência descreve os objectos de um sistema, e a sequência de mensagens trocadas entre os objectos para realizar uma determinada funcionalidade.⁴⁸

⁴⁸ Mais informação no artigo “Sequence Diagram” da Wikipedia.

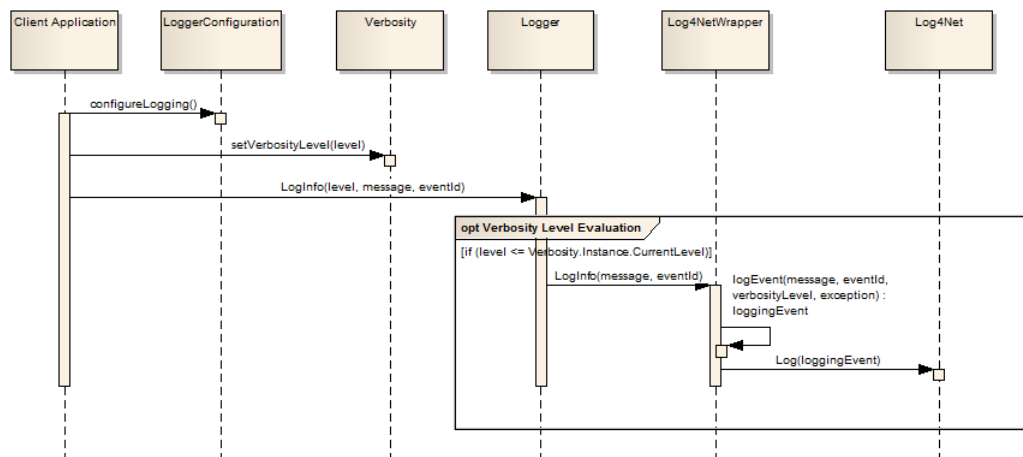


Figura 17 – Diagrama de Sequência

O ciclo de operações necessárias para realizar um registro de log começa naturalmente do lado da aplicação cliente. Neste caso a aplicação cliente começa por comunicar com a classe “LoggerConfiguration” a fim de definir configuração da framework de logging externa. Posto isto e apesar de o subsistema de logging possuir um nível padrão para a sua verbosidade, convém ao programador definir o nível de verbosidade desejado para o sistema através da chamada ao método *setVerbosityLevel(level)* da classe “Verbosity”. A partir deste momento o subsistema de logging está pronto para receber pedidos de log da aplicação cliente. A aplicação cliente faz um pedido de log através de um dos métodos da API da classe “Logger” (neste caso trata-se do método *LogInfo(level, message, eventId)*), pedido este que será filtrado antes de ser propagado no sistema. Esta validação é feita através da comparação do nível de log do sistema com o nível de log do pedido de log, o que significa que o pedido de log só será propagado no sistema se o seu nível de log for inferior ou igual ao nível de log do sistema. Uma vez propagado, o pedido de log é encaminhado para a classe “wrapper” respectiva, mais concretamente para o método correspondente nessa classe, neste caso o método *LogInfo(message, eventId)*. O passo seguinte seria a propagação do pedido para a respectiva framework de logging. No entanto, decidimos neste exemplo apresentar um caso onde a API nativa da framework de logging não suporta a inclusão do parâmetro “eventId”, o que obriga a que o pedido de log seja encaminhado para a função *logEvent(message, eventId, verbosityLevel, exception)* que será responsável por construir um evento de log para a framework de logging compatível com a inclusão do parâmetro “eventId”. Uma vez concluído este processo o subsistema estará preparado para propagar o pedido de log da aplicação cliente para a framework de logging escolhida.

5.2. Medições de Performance

Frameworks de Logging

Como o nosso subsistema suporta a integração das três frameworks de logging adoptadas neste trabalho, faz sentido medir o desempenho destas ferramentas, por forma a quantificar o possível impacto que estas possam ter no desempenho das aplicações em que se inserem.

Para este caso o objectivo é medir quanto custa fazer log de uma mensagem simples (apenas uma string) para ficheiro e para o Event Viewer para cada uma das três frameworks. As medições para cada um dos cenários foram feitas separadamente, para evitar possíveis conflitos, e as frameworks partilham a mesma configuração (adaptada a cada uma), onde não existem quaisquer restrições ou filtrações, as frameworks limitam-se a registar uma simples mensagem de log. A ferramenta utilizada para realizar as medições foi o já referido ASPerfBench e as medições são expressas em *ticks*.⁴⁹

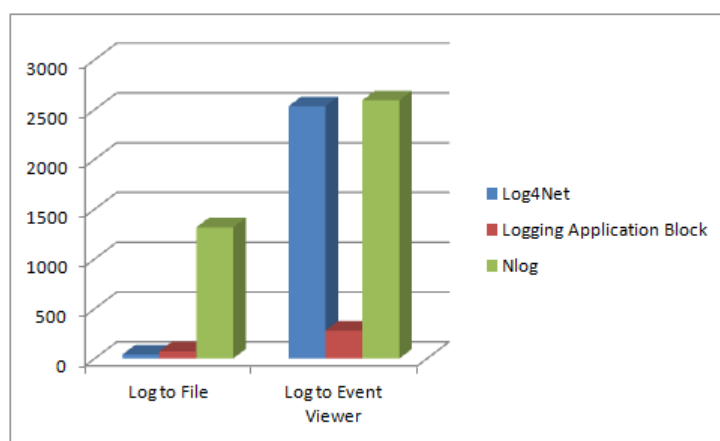


Figura 18 – Logging Frameworks Performance

Como podemos ver pelo gráfico o log4net tem o melhor desempenho na escrita de logs para ficheiros, logo seguido pelo LAB. O Nlog apresentou resultados para escrita de ficheiros que não estávamos à espera, especialmente tendo em consideração algumas referências encontradas na internet. No entanto, como foi referido, as configurações são idênticas para as três frameworks e foram realizados vários testes para despistar este comportamento e tal não aconteceu. Isto significa que o log4net é a escolha ideal para um projecto de software necessite de registar eventos de execução para ficheiros.

Em contrapartida, na escrita de *logs* para o Event Viewer, o LAB é a framework que apresenta melhores resultados, com uma diferença muito significativa em relação às outras duas frameworks que apresentam resultados muito próximos. Neste caso a escolha ideal seria sem dúvida a framework da Microsoft.

Resumindo, o log4net apresenta os melhores resultados para escrita de *logs* para ficheiros e o LAB apresenta os melhores resultados para a escrita de *logs* para o Event Viewer. Na eventualidade de um projecto necessitar de registar eventos para ambos os cenários, provavelmente a melhor escolha (a que teria menor impacto) seria o LAB.⁵⁰

⁴⁹ Esta é uma unidade directamente relacionada com o processamento do sistema, pelo que se torna mais fácil traduzir as medições obtidas no nosso ambiente para outros ambientes, nomeadamente com capacidades de processamento diferentes.

⁵⁰ Os valores obtidos para esta medição podem ser consultados no Apêndice 8 – Valores dos Benchmarks.

Concatenação de Strings

A imagem seguinte mostra-nos a média dos resultados de dez medições realizadas para as operações de concatenação de *strings* (as medições individuais encontram-se no Apêndice 2 – Benchmark de Concatenação de strings).

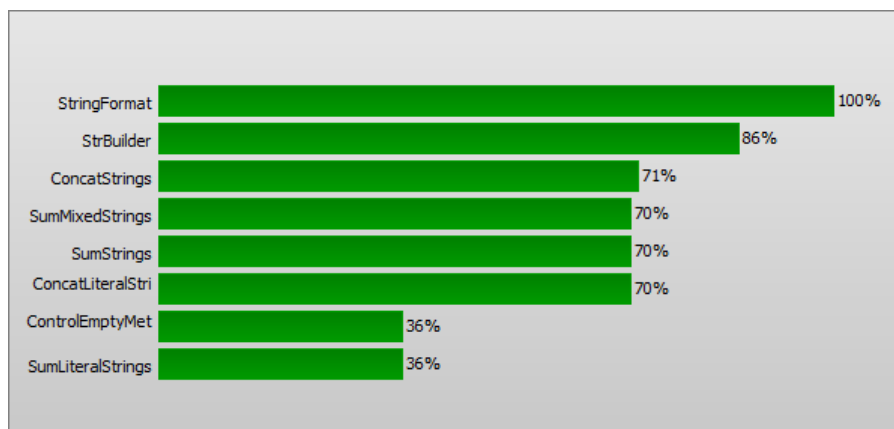


Figura 19 – Benchmark para concatenação de strings

Após a análise destas medições conseguimos tirar algumas conclusões importantes:

- O método `ControlEmptyMethod()`, que serve de base de comparação para todos os outros métodos, apresenta custos de performance na ordem dos 36%, pelo que podemos concluir que à exceção da abordagem de concatenação de strings literais usada no método `SumLiteralStrings()`, todas as outras abordagens têm custos muito superiores;
- Apesar de as diferenças de valores não serem muito significativas quando falamos da escrita de uma única mensagem de log, tal já não se verifica falamos de um cenário de escrita de mensagens de log dentro de um ciclo que é executado centenas ou milhares de vezes, numa aplicação que funciona vinte e quatro horas por dia. Nestes casos, a mais pequena melhoria de performance pode trazer grandes benefícios;
- Verificámos que a abordagem que usa a função `“String.Format()”` apresenta os piores valores de performance, sendo cerca de 64% mais dispendiosa que a abordagem usada no método `SumLiteralStrings()`, logo seguida da abordagem que usa a classe `“StringBuilder”`, com um custo superior na ordem dos 50%, enquanto as restantes abordagens mostraram ter custos de performance muito semelhantes, cerca de 34% mais dispendiosas que a base de comparação. Isto permite-nos concluir que na fase de construções das mensagens de log as abordagens usadas nos métodos `“String.Format()”` e `“StringBuilder”` devem ser preteridas, o que no contexto da concatenação de *strings* pode significar melhorias de performance entre os 16% e os 30%.

Formatação do Timestamp

À semelhança da secção anterior, esta imagem mostra-nos a média dos resultados de dez medições realizadas para as operações de formatação de *timestamp* (as medições individuais encontram-se no Apêndice 3 – Benchmark de formatação de Timestamp).

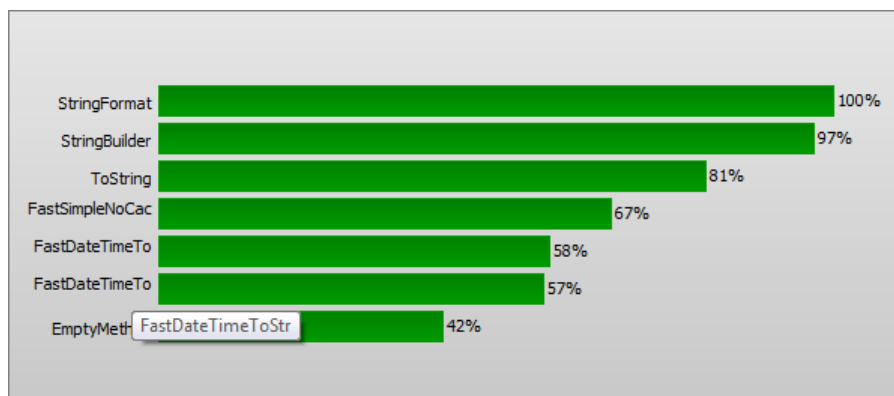


Figura 20 – Benchmark para formatação de timestamp

Depois de analisarmos os resultados constatamos o seguinte:

- As abordagens que usam a função `String.Format()` e a classe `StringBuilder` têm os piores valores de performance: Estes métodos custam aproximadamente o dobro dos métodos com os melhores resultados de performance, nomeadamente os métodos que usam a abordagem menos convencional inspirada no artigo “Performance Quiz: Fastest way to format time”, que por sua vez custam apenas mais 15% que o método vazio usado como referência;
- As restantes abordagens têm também resultados distintos, sendo que a abordagem usada no método `FastSimpleNoCache` custa aproximadamente mais 10% que as melhores abordagens, enquanto a abordagem que usa a função `ToString` supera esta última em 14%;
- Isto significa que é aconselhável usar a abordagem menos convencional na formatação do elemento de *timestamp* da mensagem de log, pois pode reduzir os custos de performance entre 10% a 43% em relação às outras abordagens.

É importante realçar ainda que os resultados obtidos nas medições relativas às abordagens de concatenação de strings e formatação de timestamp, podem não ser muito significativos quando comparados entre si. No entanto, se assumirmos que estas operações podem ser realizadas centenas ou milhares de vezes, o somatório destes valores pode então já ter um impacto significativo na performance das aplicações.

Uso de “Reflection” na construção das mensagens

“Reflection” é uma funcionalidade, neste caso do C#, que fornece informação sobre métodos e construtores. Esta pode ser uma funcionalidade útil para o desenvolvimento de software e muitas vezes é usada na definição de parâmetros de mensagens de log. O problema reside no facto de esta ser uma funcionalidade relativamente dispendiosa, o que pode significar custos adicionais para a performance do logging. Tendo isto em consideração decidimos comparar os custos de performance da utilização de “reflection” directamente na construção da mensagem do log, com os custos de performance da utilização de “reflection” através do armazenamento da sua informação num variável (por exemplo durante o carregamento da classe) que posteriormente será utilizada na construção da mensagem de log.

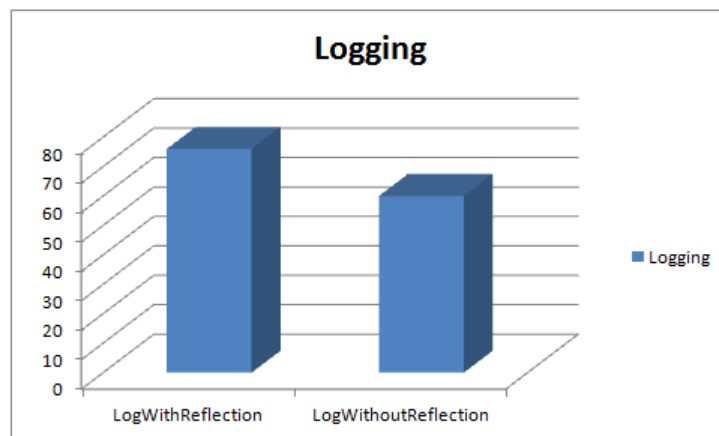


Figura 21 – Custo de “Reflection” na construção das mensagens de log

Como podemos constatar pelo gráfico, a abordagem de utilização de “Reflection” na construção das mensagens de log é mais dispendiosa que a abordagem onde a operação de “reflection” é realizada e armazenada durante o carregamento da classe e utilizada posteriormente na construção das mensagens de log.⁵¹

⁵¹ Os resultados desta medição bem como algumas informações extra sobre o ambiente de teste podem ser encontradas no Apêndice 8 – Valores dos Benchmarks

5.3. Documento de *Guidelines*

O documento de *guidelines* é uma das componentes mais importantes deste trabalho, uma vez que servirá de base de consulta, estudo e aprendizagem para os programadores que no futuro queiram implementar um sistema de logging nos seus projectos. Na realidade este documento é direccionado para o contexto do logging aplicacional para o desenvolvimento de software em .Net, e concretamente, através da utilização do subsistema de logging da CSW (que por sua vez suporta as frameworks de logging log4net, Enterprise Library – Logging Application Block e Nlog). No entanto muitos dos conceitos abordados neste documento são transversais e independentes do contexto.

Neste documento abordamos alguns conceitos que já foram discutidos em capítulos anteriores, como são os casos dos níveis e categorias de log, pelo que não faz sentido voltar a referi-los neste capítulo. Assim sendo, abordaremos questões que ainda não o foram até aqui e que servirão de orientação para os programadores durante a implementação e utilização do logging.

Cenários de Logging

Cenários de logging são situações ou fases de execução de uma aplicação, que têm ponderações diferentes a nível do impacto na performance das aplicações e no detalhe e tipo de informação disponibilizada. A correcta identificação do cenário de log em que o programador está inserido pode assim ter um enorme impacto no desempenho da aplicação. Tendo isto em consideração e após a consulta de alguns profissionais da CSW com vários anos de experiência em desenvolvimento de software, identificámos e descrevemos os seguintes cenários de log:

- “Inicialização e/ou Instanciação” – Logging feito em zonas onde o código é executado apenas uma vez e onde o *trade-off* entre performance e quantidade de informação tende para a quantidade de informação. Neste cenário é importante verificar configurações, inicializações de componentes e que a informação seja fiável e detalhada. Assim, neste cenário, uma mensagem de log deve indicar os seguintes elementos: os parâmetros de configuração; os valores atribuído aos parâmetros; a gama de valores permitidos, caso se trate de um erro; e a versão do ficheiro de configuração e de todas as dependências. Para este cenário devem ser utilizadas categorias de mensagens do tipo "Info" para apresentar o progresso da inicialização e/ou instanciação, mensagens do tipo "Warn" quando são detectados problemas com a inicialização e/ou instanciação com os quais o sistema consegue lidar (por exemplo, se for definido um valor errado para um parâmetro e o sistema tiver um valor *default*, o sistema deve assumir o valor *default* e lançar uma mensagem de log do tipo "Warn" a alertar para essa situação). Caso sejam detectados erros na inicialização e/ou instanciação com os quais o sistema não consegue lidar ou resolver, devem ser lançadas mensagens de log do tipo "Error" ou "Fatal", consoante a severidade do erro e as suas implicações para a estabilidade do sistema;
- “Encerramento” – Este cenário é semelhante ao anterior em termos de necessidades de logging, uma vez que numa fase de encerramento a maior preocupação recai sobre com a quantidade de informação disponível e não sobre o impacto que o logging possa ter nesta fase. Neste cenário o principal objectivo é assegurar que foi feito um *clean shutdown*, ou seja, que todos os dados existentes em memória foram guardados ou que todas as ligações foram fechadas;

- “Execução Cíclica” – Neste caso, a importância da performance sobrepõe-se à importância da quantidade de informação, pois estamos a falar de instruções que podem ser executadas milhares de vezes durante um curto espaço de tempo. A inclusão de mensagens de log nestes cenários deve ser restringida ao máximo a mensagens do tipo "Error" ou "Fatal", uma vez que nestes casos estamos dispostos a sacrificar a performance da aplicação em favor de informação suficiente sobre os erros. Caso seja necessário obter mais informação sobre o sistema durante esta fase (por exemplo, apresentar valores de carga do sistema), devem ser usadas estruturas de mensagens mais compactas e devem ser definidos *triggers* para despoletar chamadas de log de x em x tempo ou de x em x instruções ao invés de despoletar chamadas de log a cada iteração do ciclo.

De forma a auxiliar a identificação das necessidades de logging dos diferentes cenários em relação ao tipo de eventos registamos, criámos a seguinte tabela. Através desta tabela podemos facilmente constatar que, para os três cenários de logging descritos, as necessidades de informação se sobrepõem quase sempre às necessidades de performance. Ou seja, para estes cenários estamos interessados em obter o maior volume de informação, o mais detalhado possível. A exceção é o cenário de execução cíclica, onde para eventos do tipo “Debug”, “Info” e “Warn” estamos dispostos a sacrificar o volume de informação por melhores valores de performance.

	Inicialização e/ou Instanciação	Encerramento	Execução Cíclica
Debug	 Informação	 Informação	 Performance
Info	 Informação	 Informação	 Performance
Warn	 Informação	 Informação	 Performance
Error	 Informação	 Informação	 Informação
Fatal	 Informação	 Informação	 Informação

Tabela 2 – Cenário de Logging (Informação vs Performance)

5.4. Implementação e Expansão do Subsistema de Logging

O intuito deste capítulo não passa por descrever detalhadamente o processo de implementação e expansão do subsistema de logging, até porque essa descrição é feita nos documentos de especificação do sistema. Os documentos de especificação são na realidade o verdadeiro trabalho de engenharia de software, sendo a implementação uma mera interpretação e tradução das especificações para código fonte.

Neste capítulo iremos referir e descrever alguns dos módulos mais importantes que foram implementados no subsistema de logging e apontar algumas estratégias utilizadas para satisfazer as necessidades do projecto. Iremos começar a descrição da implementação e expansão por uma visão macro do sistema, avançando progressivamente para descrições mais específicas e pormenorizadas de aspectos relacionados com o projecto e com as boas práticas de desenvolvimento de software.

Durante a fase de implementação nós pegamos num sistema que continha apenas duas classes, a classe “Logger” e a classe “Log4NetWrapper”, responsáveis pela interface comum do sistema e pela comunicação com o log4net, e expandi-mo-lo para um sistema composto por cinco classes, as duas classes já referidas e mais três novas classes, a classe “EntLibWarpper” responsável pela comunicação com o LAB, a classe “NLogWrapper” responsável pela comunicação com o Nlog e por fim a classe “LoggerConfiguration” responsável pelo carregamento do ficheiro de configurações do sistema e que na versão anterior do subsistema de logging se encontrava inserida na classe “Log4NetWrapper”. Tendo em conta que esta classe é partilhada pelas três classes wrapper e que o carregamento da configuração é realizado uma única vez durante o arranque da aplicação, faz todo o sentido criar uma nova classe que sirva exclusivamente para este propósito.

Um aspecto importante a ter em consideração num projecto reutilizável e expansível é manter um registo das alterações efectuadas em cada classe ou objecto. Dessa forma todas as classes implementadas ou expandidas incluem um historial no cabeçalho, onde é indicada a data da alteração realizada, o nome do programador responsável pela alteração, um carácter que indique a natureza da alteração (por exemplo [+] se se tratar de uma nova funcionalidade, ou [*] se se tratar de uma alteração a uma funcionalidade existente) e por fim uma descrição da alteração efectuada.

A classe “Logger” foi sujeita a algumas alterações por forma a dar resposta ao requisitos do sistema, sendo a tradução das interfaces específicas de cada framework de logging a alteração com maior impacto nesta classe. Mas para além desta alteração foram realizadas outras alterações de relevância para o projecto, como por exemplo a reestruturação do objecto responsável definição do nível de verbosidade do sistema, que anteriormente usava a nomenclatura “Level” e os seus níveis eram divididos em “Debug”, “Development”, “Production” e “None” e que após a aplicação dos conhecimentos reunidos neste trabalho passou a usar a nomenclatura “VerbosityLevel” e os seus níveis passaram a estar divididos em “High”, “Medium”, “Low” e “None”. Esta alteração traduz de uma forma mais fiel os conceitos do logging e facilita a sua compreensão e utilização durante a implementação. Ainda em relação à classe “Logger” convém salientar a inclusão dos métodos referentes à concatenação de strings e formatação do elemento de timestamp, que de acordo com as nossas medições utilizam as abordagens mais eficazes e dos métodos que sugerem estruturas normalizadas para as mensagens de log.

No que diz respeito às classes *wrapper*, convém referir que as frameworks de logging log4net e Nlog não suportavam nativamente a inclusão do elemento “event ID” (utilizado pelo Event Viewer) nas mensagens de log, pelo que foi necessário incluir nestas classes um método específico por implementar esta funcionalidade. E uma vez que nós queremos que as frameworks de logging sejam apenas responsáveis por persistir os logs, ou seja, não queremos que estas sejam responsáveis pela inclusão e concatenação de nenhum parâmetro, foi necessário incluir manualmente a categoria do log nos parâmetros das mensagens, de modo a que as chamadas de log possa ser realizadas passando apenas uma única string para a framework de logging. Desta forma evitamos também qualquer tipo de processamento adicional para traduzir categorias de log com nomenclaturas diferentes em algumas frameworks de logging, como era o caso da LAB.

Capítulo 6

Plano de Trabalho e Implicações

O objectivo deste capítulo é apresentar o plano definido para a realização deste trabalho, comparando as previsões iniciais com o percurso efectivo do projecto e apontando as implicações que de alguma tiveram um impacto positivo ou negativo no projecto.

O tema do estágio foi um dos aspectos que condicionou a realização deste trabalho, pois apesar de o logging ser uma ferramenta muito utilizada no desenvolvimento de software, a sua utilização continua a ser menosprezada pelos programadores. Este “desprezo” está patente no facto de haver muita documentação sobre logging, mas muito pouco documentação sobre a aplicação de boas práticas de logging que favoreçam a qualidade dos registos produzidos e que reduzam o impacto do logging na performance das aplicações.⁵² No entanto, podemos afirmar que este tema nos deu a oportunidade de desenvolver um trabalho realmente útil e inovador.

Como já foi referido, este projecto foi realizado no contexto de um estágio curricular na empresa Critical Software, SA. A equipa do projecto era composta por três elementos, um gestor de projecto, um gestor técnico e o estagiário. Na realidade, o projecto tinha características diferentes de outros projectos, uma vez que para este caso o gestor de projecto assumiu o papel de “cliente” e ao estagiário foi atribuída a responsabilidade de desenvolver todas as fases do projecto, incluindo o planeamento, gestão, especificação e implementação, sob a supervisão e orientação do gestor técnico. Isto demonstra confiança por parte da empresa nas capacidades do estagiário e enquanto lhe proporciona a oportunidade de realizar um trabalho realmente diferente e completo, uma vez que irá participar de todas as fases inerentes à engenharia de software.

Posto isto, podemos avançar para a descrição do planeamento feito para este trabalho. Em primeiro lugar apresentamos e descrevemos o plano estabelecido inicial para a realização do trabalho, que por sua vez servirá de base de comparação para o plano final, onde estará traduzido o verdadeiro percurso do trabalho incluindo os principais marcos e linhas temporais de execução. O objectivo passa também para avaliar o que correu mal com o planeamento inicial e justificar decisões e possíveis desvios.

Os planos são expressos graficamente na forma de diagramas de Gantt e são acompanhados pela respectiva descrição.

⁵² De acordo com os autores do artigo “Characterizing Logging Practices in Open-source Software” de 2012, esse artigo é mesmo a primeira tentativa de criar um documento do género.

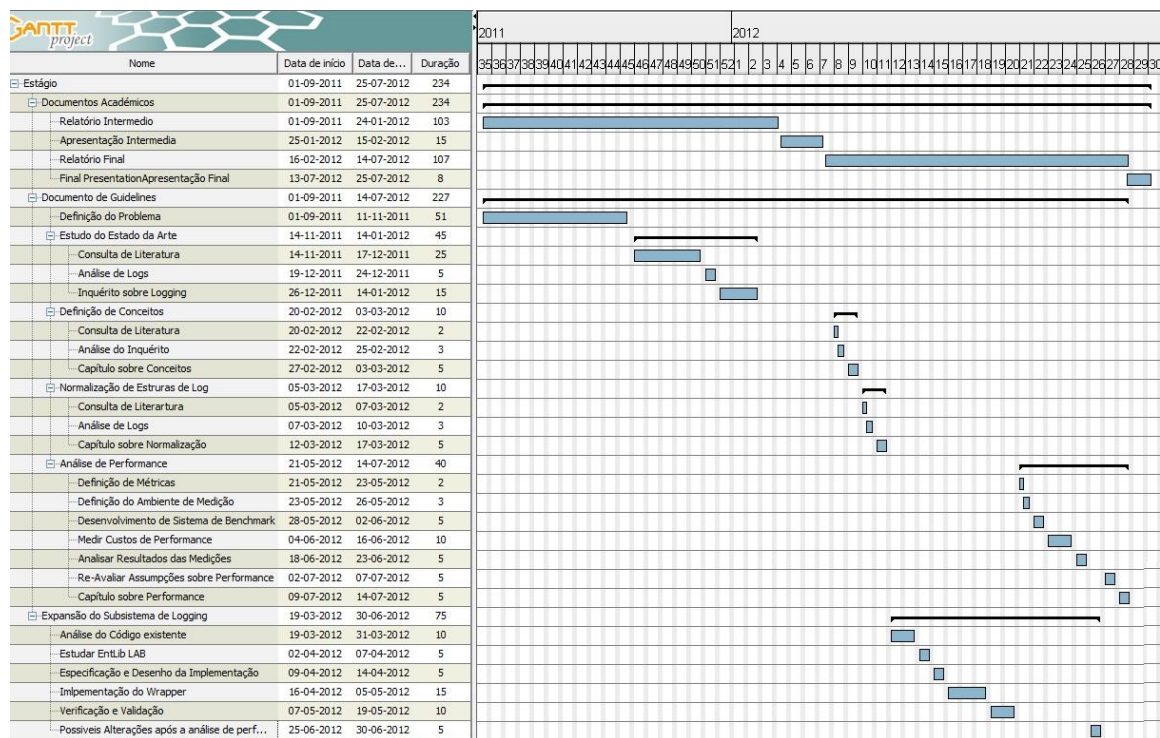


Figura 22 – Planeamento Inicial

Como podemos constatar por esta figura, o planeamento inicial era pouco detalhado, revelando alguma falta de experiencias e conhecimento sobre a temática do logging, naturais da fase inicial do projecto. O número de tarefas apresentadas era substancialmente reduzido e a sua descrição e divisão não traduziam as necessidades reais de um projecto desta natureza.

Este plano mostra uma produção muito baixa nos primeiros meses do trabalho, justificada por alguma dificuldade na definição clara dos objectivos do projecto, o que originou inclusive uma discrepância entre os objectivos esperados, pela faculdade, para o estágio e os objectivos desejados pela empresa para o trabalho. Esta situação acabou por ser resolvida e os objectivos foram definitivamente clarificados, permitindo ao estagiário definir um novo plano para o projecto que será apresentado mais à frente.

No plano inicial é possível identificar algumas tarefas importantes para a realização do trabalho, como a definição de conceitos, a normalização de estruturas logging, a medição de custos de performance e a expansão do subsistema de logging. No entanto este planeamento não traduz um conhecimento real das implicações inerentes a cada uma dessas tarefas. Pelo que após a defesa intermédia foi necessário definir um novo plano para a realização deste trabalho.

Este novo plano apresenta a lista de tarefas e respectivos tempos de execução apenas para a segunda metade deste trabalho, pois tal como foi referido a primeira metade do trabalho foi ocupada essencialmente com tarefas relacionadas com a definição dos objectivos e com o estudo e análise de tema do estágio.

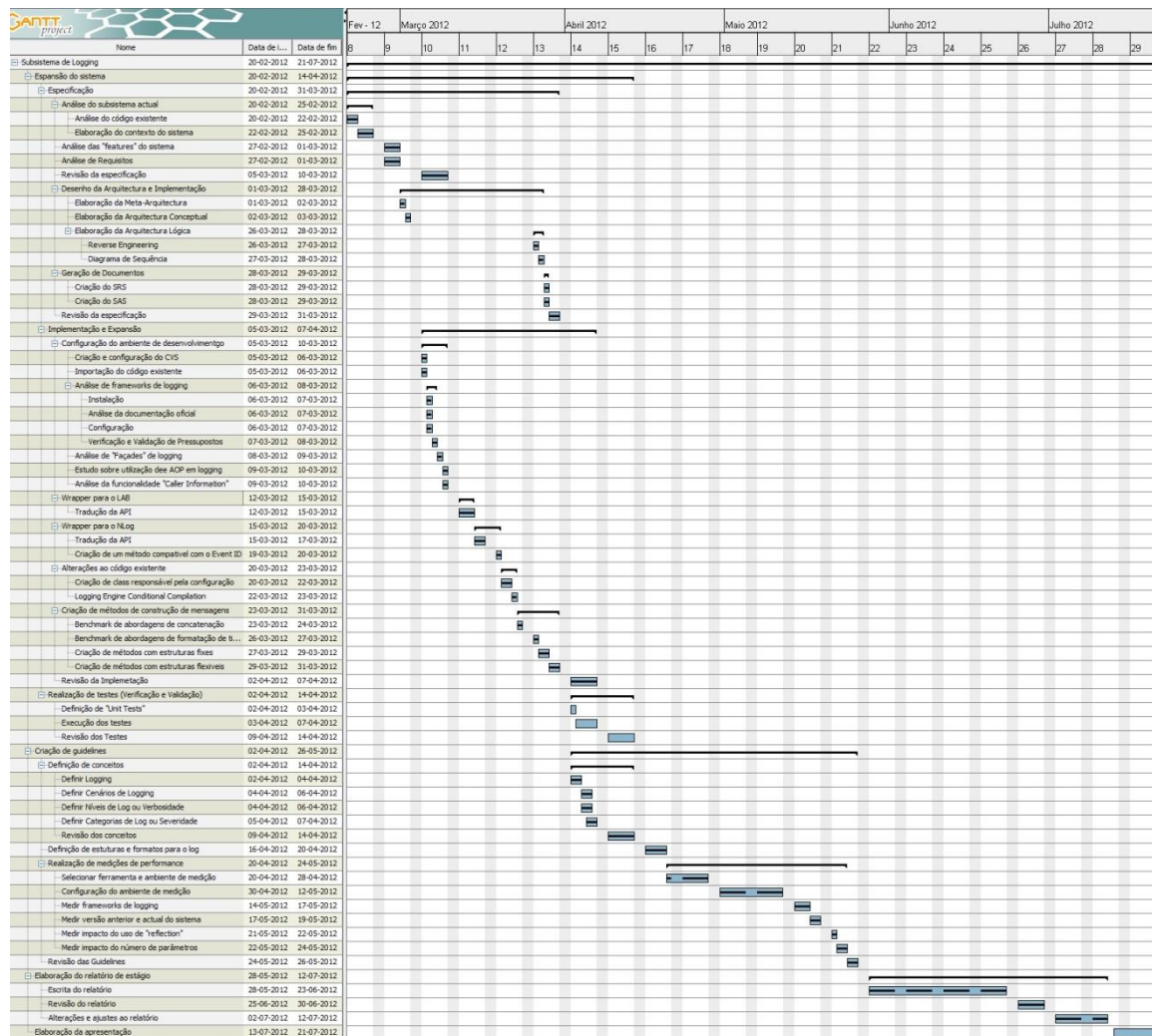


Figura 23 – Planeamento final

Através desta figura é notória a diferença entre os dois planeamentos efectuados durante a realização deste trabalho. Este é um plano muito mais detalhado e que refere muito ao pormenor todas as implicações inerentes à concretização dos objectivos do estágio.

Começamos por destacar a importância que é atribuída à especificação e desenho do subsistema de logging. Esta foi no nosso entender um dos factores críticos de sucesso deste trabalho visto que o verdadeiro trabalho de engenharia neste tipo de projectos reside exactamente nestes factores. A elaboração da especificação e desenho do sistema, de acordo com os processos de desenvolvimento de software definidos pela CSW, são uma garantia de concretização e qualidade do produto final.

Aproveitando a referência aos processos de desenvolvimento, convém destacar o facto de terem sido realizadas revisões progressivas do trabalho realizado. Como se pode ver no plano, após a conclusão de cada uma das tarefas principais foi realizada uma revisão por parte da equipa do projecto e pela orientadora do estágio.

No que diz respeito à concretização do plano, existem alguns aspectos a destacar. Em primeiro lugar devido à falta de tempo não foi possível realizar uma fase oficial de testes ao subsistema implementado. Isto não significa que não tenham sido realizados testes, pois durante a implementação todas as suposições implementadas foram verificadas e validadas através de testes unitários realizados pelo estagiário. Esta falta de tempo deveu-se

essencialmente a alguns atrasos relacionados com a seleção e configuração das ferramentas e ambientes de medição. Houve alguma dificuldade em identificar uma ferramenta que servisse as nossas intenções, ora porque as ferramentas não eram “open-source”, ora porque simplesmente não disponibilizavam as funcionalidades necessárias. A escolha acabou por recair na ferramenta ASPerfBench, uma ferramenta de *benchmark* para .Net, desenvolvida pelo gestor técnico da equipa. No entanto, tivemos igualmente problemas com a configuração dos ambientes de testes nesta ferramenta, uma vez que esta não conseguia carregar os ficheiros de configuração do subsistema quando este estava compilado para usar as frameworks log4net e LAB. A solução passou por introduzir as configurações programaticamente no código testado, mas fora das zonas de medição.

No que diz respeito à elaboração das guidelines de logging, é importante explicar que as estas se encontram ao longo de todo o trabalho e não apenas num capítulo específico. As guidelines podem ser encontradas, na definição do estudo do estado da arte, nos documentos de especificação e nas medições de performance realizadas.

Para finalizar, podemos igualmente constatar que a tarefa que exigiu mais tempo de execução foi a elaboração do relatório de estágio. Esta é uma decisão perfeitamente compreensível e aceite pela equipa do projecto, uma vez que é neste documento que está reunido o conhecimento adquirido durante a realização deste trabalho.

Capítulo 7

Conclusões

A realização deste trabalho foi baseada na assumpção de que o logging é uma ferramenta importante no desenvolvimento de software e que desta forma era necessário assegurar que as melhores práticas de logging seriam aplicadas no desenvolvimento de projectos da Critical Software. Os objectivos deste trabalho passavam essencialmente por construir e partilhar conhecimento sobre a temática do logging que permitisse orientar os programadores para produzirem registos de execução (logs) com maior qualidade (legibilidade e detalhe de informação) e menor impacto na performance das aplicações. E posteriormente aplicar esses conhecimentos na implementação e expansão de um subsistema de logging que servisse de interface única para a utilização de diferentes frameworks de logging.

Neste trabalho definimos e clarificamos um conjunto de conceitos, como “O que é o logging?”, “Para que serve o logging?”, “O que são níveis, categorias e cenários de log?”, que ainda hoje geram confusão que nos permitiram não só comprovar a importância do logging mas também construir uma base de conhecimento sólido sobre o logging, essencial para podermos definir as melhores abordagens para tirar o melhor partido desta ferramenta.

Definir e clarificar que logging é uma “Criação otimizada (em termos de performance) de registos estruturados sobre a execução de um software, com o intuito de auxiliar a fase de manutenção do software” foi essencial para distinguir o conceito do logging de conceitos como *tracing* ou auditoria, que têm objectivos e necessidades completamente diferentes. Distinguir os conceitos de verbosidade e categorias de logging foi essencial para compreender o funcionamento dos mecanismos do logging e perceber como tirar o melhor partido das potencialidades que as frameworks de logging disponibilizam. A verbosidade, tal como o próprio nome indica, serve para filtrar o volume de log gerado pela aplicação e de alguma forma poder controlar o impacto do logging na performance das aplicações de acordo com as necessidades. As categorias de logging servem para classificar eventos de log, pois existem tipos de eventos diferentes que necessitam de tratamento e acções de mitigação diferentes. Da mesma forma a definição do conceito de cenários de logging e a descrição das características de cada um dos cenários permitiu-nos compreender que o mesmo software pode ter necessidades de logging diferentes de acordo com a zona ou o tipo do código onde o logging está inserido e que como tal o logging não pode ser feito sempre da mesma maneira. Existem cenários onde as necessidades de qualidade e quantidade de informação se sobrepõem às necessidades de performance, como são os casos da inicialização da aplicação, da instanciação de módulos ou do encerramento da aplicação. Por outro lado existem cenários onde a performance é crítica e o logging deve ter o menor impacto possível no desempenho da aplicação, como é por exemplo o caso de execuções cíclicas. A menos que se tratem de eventos anormais, que possam de alguma forma comprometer a estabilidade da aplicação. Nesse caso, a necessidade de fornecer informação fiável e suficientemente detalhada sobrepõem-se a qualquer necessidade de performance.

Através deste trabalho conseguimos identificar um conjunto de técnicas e abordagens que melhoram substancialmente a qualidade dos registos de log produzidos e reduzem o impacto que este tem na performance das aplicações. Destacamos as boas práticas de geração de registos de log referidas em artigos e livros especializados, relacionadas com as estruturas e formatos das mensagens de log, tais como a inclusão de um elemento de *timestamp*, formatado de acordo com as normas ISO, a utilização de apenas uma linha de log por evento para facilitar a leitura, ou o simples facto de os registos de log terem uma estrutura homogénia. Em relação às técnicas e abordagens que reduzem o impacto do log na

performance das aplicações, destacamos as abordagens de concatenação de strings e formatação de *timestamp* implementadas neste trabalho, a utilização de expressões lambda na construção das mensagens, por forma evitar custos desnecessários de processamento mensagens filtradas pelo nível de verbosidade do sistema, ou a sugestão de evitar usar “reflection” directamente na definição dos parâmetros da mensagem de log.

Comprovamos a utilidade de uma “façade” de logging para simplificar a implementação e utilização de mecanismos de logging. A *façade*, referida neste trabalho como subsistema de logging, permite ao programador usar uma interface (API) constante independentemente da framework de logging utilizada. Para além das naturais vantagens de flexibilidade, a *façade* torna-se igualmente uma ferramenta essencial para garantir a normalização e homogeneidade dos registos de log produzidos.

Por outro lado, este trabalho deu-nos ainda a possibilidade expandir os nossos conhecimentos académicos e fazer parte de um projecto real de engenharia de software, inserido numa das empresas mais conceituadas na área. Através da qual tivemos contacto com as melhores práticas, processos e metodologias relacionados com o desenvolvimento e gestão de projectos de software. Este aspecto teve naturalmente um impacto na realização e resultados finais do trabalho.

Entre os processos e metodologias a que tivemos acesso, o processo de desenvolvimento de software da critical software foi o que teve um maior impacto neste trabalho uma vez que serviu de base e orientação para todo o projecto. A correcta utilização deste processo quase que garante por si só a qualidade e concretização do projecto. Foi com base neste processo que foram elaborados os documentos de especificação de requisitos e arquitectura do sistema. De acordo com o nosso ponto de vista, o verdadeiro valor de um projecto e o verdadeiro trabalho de engenharia de software encontra-se na fase de desenho e modelação do sistema.

A especificação do sistema foi suportada por validações de suposições através de consulta de literatura técnica e específica sobre os temas abordados, mas também através da realização de medições de performance de algumas técnicas e abordagens sugeridas. Entre as medições realizadas convém destacar as medições dos custos de performance de cada framework na escrita para ficheiros e para o “Event Viewer”, do desempenho do subsistema de logging em relação à frameworks de logging e da versão antiga do subsistema em relação a nova versão, ou os custos de performance das diferentes abordagens de concatenação de strings e formatação do elemento de timestamp. De acordo com estas medições podemos afirmar que o log4net tem um melhor desempenho na escrita de logs para ficheiro, enquanto na escrita de logs para o “Event Viewer” é o LAB que apresenta os melhores resultados. Para a concatenação de strings deve ser usada uma abordagem de concatenação simples através do uso do carácter ‘+’, enquanto que para a formatação deve ser usada a abordagem referida neste trabalho como “FastDateTimeToStr”.

Resumindo, o balanço é positivo para a empresa e para o estagiário. A empresa viu os objectivos cumpridos e em alguns casos até excedidos, como foi o caso da inclusão da framework de logging NLog no subsistema de logging. A versão final do subsistema cumpre o requisitos de performance e de normalização definidos. E as guidelines de utilização do logging foram apresentadas ao longo de todo o trabalho. Do ponto de vista do estagiário, foi uma experiência muito enriquecedora ao nível da construção de conhecimentos relacionados com a engenharia de software e a temática do logging. No final do trabalho foi inclusivamente feita uma validação dos conhecimentos de logging adquiridos junto de profissionais mais experientes da empresa e o feedback foi muito positivo.

Bibliografia

- Apache.** Apache log4net™ Frequently Asked Questions. *Apache Logging Services*. [Online] <http://logging.apache.org/log4net/release/faq.html>.
- . Apache log4net™ Manual - Introduction. *Apache Logging Services*. [Online] <http://logging.apache.org/log4net/release/manual/introduction.html>.
- Berard, Edward V. 1993.** Abstraction, Encapsulation and Information Hiding. *Information Technology Management Web*. [Online] 1993. <http://www.itmweb.com/essay550.htm>.
- Booch, Grady. 1998.** *Object-Oriented Analysis and Design with Applications - 2nd Edition*. s.l. : ADDISON-WESLEY, 1998.
- Bourque, Pierre and Dupuis, Robert. 2004.** Software Requirements. *Guide to the Software Engineering Body of Knowledge*. s.l. : IEEE, 2004.
- Buschmann, Frank, et al. 1996.** Pattern-Oriented Software Architecture Volume 1: A System of Patterns. s.l. : John Wiley & Sons, 1996, pp. 397-404.
- Characterizing Logging Practices in Open-source Software.* **Yuan, Ding, Park, Soyeon and Zhou, Yuanyuan. 2012.** Zurich, Switzerland : s.n., 2012. 34th International Conference on Software Engineering (ICSE'12).
- CodePlex. 2012.** Comparison of logging frameworks. *Essential Diagnostics*. [Online] Janeiro 3, 2012. <http://essentialdiagnostics.codeplex.com/wikipage?title=Comparison>.
- Google.** Google Insights for Search. [Online] <http://www.google.com/insights/search/#q=log4net%2Clogging%20application%20block%2CNLog%2Csmart%20inspect%2CObject%20Guy%20Logging%20Framework&cmpt=q>.
- . Google Trends. [Online] <http://www.google.com/trends/?q=log4net,+nlog,+logging+application+block,+smart+inspect,+Object+Guy+Logging+Framework&ctab=0&geo=all&date=all&sort=0>.
- Gurock Software.** Comparison of .NET Logging Frameworks and Libraries. *.NET Logging Tools and Libraries*. [Online] <http://www.dotnetlogging.com/comparison/>.
- IEEE. 2002.** Recommended Practice for Software Requirements Specifications. *IEEE Xplore*. [Online] Agosto 06, 2002.
- In, Sign. 2009.** Thoughts on improving performance of the Logging Application Block. *Thoughts on Agile Software Engineering and Beyond*. [Online] Junho 19, 2009. <http://blogs.msdn.com/b/agile/archive/2009/06/19/thoughts-on-improving-performance-of-the-logging-application-block.aspx>.
- JBoss.** Logging Conventions. *JBoss Community*. [Online] <http://docs.jboss.org/process-guide/en/html/logging.html>.
- Kellerman Software.** .NET LOGGING LIBRARY COMPETITION COMPARISON. *Kellerman Software*. [Online] <http://www.kellermansoftware.com/p-14-net-logging-library.aspx?mode=Competition&>.

- Kowalski, Jaroslaw.** NLog 2.0 Release Notes. *NLog Advanced .Net Logging*. [Online] <http://nlog-project.org/nlog-2-0-release-notes>.
- . Tutorial. *NLog Advanced .Net Logging*. [Online] http://nlog-project.org/wiki/Tutorial#Log_levels.
- Kraus, Alois.** 2006. Performance Quiz: Fastest way to format time. *Geeks with Blogs*. [Online] Abril 23, 2006. <http://geekswithblogs.net/akraus1/archive/2006/04/23/76146.aspx>.
- . 2008. Really Fast Formatting with Enterprise Library. *Geek With Blogs*. [Online] Janeiro 14, 2008. <http://geekswithblogs.net/akraus1/archive/2008/01/14/118543.aspx>.
- Legheri, Nauman.** 2003. Using log4net. *O'Reilly Windows DevCenter*. [Online] Junho 16, 2003. <http://ondotnet.com/pub/a/dotnet/2003/06/16/log4net.html>.
- Malan, Ruth and Bredemeyer, Dana.** 2004. Conceptual Architecture. *Bredemeyer Consulting - Enterprise Architecture, Software Architecture, Architects, and Architecting*. [Online] Setembro 2004. <http://www.bredemeyer.com/ArchitectingProcess/ConceptualArchitecture.htm>.
- . 2004. Meta-Architecture. *Bredemeyer Consulting - Enterprise Architecture, Software Architecture, Architects, and Architecting*. [Online] Abril 2004. <http://www.bredemeyer.com/ArchitectingProcess/MetaArchitecture.htm>.
- Mannion, Mike and Keepence, Barry.** 1995. SMART Requirements. *Software Engineering Notes vol 20 no 2*. April 2, 1995.
- . 1995. SMART Requirements. Abril 02, 1995.
- Meyer, Bertrand.** 1997. *Object-Oriented Software Construction - 2nd Edition*. s.l. : Prentice Hall, 1997.
- Microsoft.** 2009. A Technique for Architecture and Design - Auditing and Logging. *Application Architecture Guide*. 2009.
- Microsoft MSDN.** Design of the Logging Application Block. *Microsoft Developer Network*. [Online] [http://msdn.microsoft.com/en-us/library/ff664735\(v=pandp.50\).aspx](http://msdn.microsoft.com/en-us/library/ff664735(v=pandp.50).aspx).
- Microsoft.** Verbosity levels (IIS 6.0). *Microsoft | TechNet*. [Online] <http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/d87ac473-eddb-4f7e-9a91-ce9f168b5635.mspx?mfr=true>.
- Miller, Jeremy.** 2008. Cohesion And Coupling. *MSDN Patterns in Practice*. [Online] Outubro 2008. <http://msdn.microsoft.com/en-us/magazine/cc947917.aspx>.
- Nygard, Micheal T.** 2007. Logging. *Release It! Design and Deploy Production-Ready Software (Pragmatic Programmers)*. s.l. : Pragmatic Bookshelf, 2007, pp. 276-283.
- Source Fourge.** WinCvs Instructions. [Online] <http://sourceforge.net/apps/trac/sourceforge/wiki/WinCvs%20instructions>.
- Stellman, Andrew and Greene, Jennifer.** 2005. Software Requirements Specification. *Applied Software Project Management*. s.l. : O'Reilly, 2005.
- Stoneman, Elton.** 2009. Making Logging Cheaper with Lambda Expressions. *Geeks with Blogs*. [Online] Abril 15, 2009.

<http://geekswithblogs.net/EltonStoneman/archive/2009/04/15/making-logging-cheaper-with-lambda-expressions.aspx>.

Thomas, Dave and Hunt, Andy. 2003. *Pragmatic Version Control using CVS*. s.l. : The Pragmatic Programmers, 2003.

Twist, Josh. 2007. Logging Levels and how to use them. *The Joy of Code*. [Online] Abril 13, 2007. http://thejoyofcode.com/Logging_Levels_and_how_to_use_them.aspx.

Wikipedia. 2012. Concurrent Versions System. *Wikipedia*. [Online] Junho 14, 2012. http://en.wikipedia.org/wiki/Concurrent_Versions_System.

—. 2012. log4j. *Wikipedia*. [Online] Maio 31, 2012. <http://en.wikipedia.org/wiki/Log4j>.

—. 2012. Syslog. *Wikipedia*. [Online] Maio 05, 2012. <http://en.wikipedia.org/wiki/Syslog>.

Wycoff, Joyce. 2001. 5-15 Reports: Communication for Dispersed Organizations. *Innovation Network*. [Online] 2001. http://innovationnetwork.biz/articles/515_reports.html.

Apêndices

Apêndice 1 – Análise de registos de execução de projectos da CSW

Projecto	Ficheiro	Ferramenta	Estrutura
Projecto #1	LAfotocop.elf	EurekaLog	<p>EurekaLog 6.0.18</p> <p>Application:</p> <pre> ----- 1.1 Start Date : Mon, 22 Nov 2010 16:07:34 +0000 1.2 Name/Description: LAfotocop.exe - (Plataformas) 1.3 Version Number : 1.0.0.10 1.5 Compilation Date: Wed, 8 Jul 2009 11:36:43 +0000 1.6 Up Time : 22 seconds </pre>
Projecto #1	PPPLANNING.err	Desconhecido	<p>GroupBox2 (Owner: FORMEPattern; Owner: ; Owner: ; Owner:), planrps LogMessage=(PM_TurnsInt) You can only enter integer values in that FormEdit_ControlValues=Number: 50582.FL4; Reel Width: 1860; Number: Screen_ActiveForm_ControlValues=Number: 50582.FL4; Reel Width: 1860</p> <hr/> <p>PNButtonControl (Owner: FORMEFolioFinOrder; Owner: ; Owner: ; Owner: LogMessage=(PM_NullOrd) You must enter at least one Mill Order for Screen_ActiveForm_ControlValues=Finishing Order;; FL1: Y; Int. Stoc</p>
Projecto #1	server.log	Desconhecido	<pre> 2011-12-20 00:00:00,981 INFO [PDF_BGREPORT_TASK] Job 39 running 2011-12-20 00:00:01,002 INFO [com.criticalsoftware.camadacomum.aut 2011-12-20 00:00:01,839 INFO [LoggingAdvice] [null] [PrimaryLos 2011-12-20 00:00:02,030 ERROR [STDERR] Dec 20, 2011 12:00:02 AM orç INFO: Data Engine starts up 2011-12-20 00:00:02,330 INFO [STDOUT] 231 checkLogin demorou 0 ms 2011-12-20 00:00:02,330 INFO [com.criticalsoftware.camadacomum.bas 2011-12-20 00:00:02,330 INFO [LoggingAdvice] [itrnf] [PrimaryLos 2011-12-20 00:00:08,668 INFO [com.criticalsoftware.camadacomum.bas 2011-12-20 00:00:37,741 ERROR [STDERR] Dec 20, 2011 12:00:37 AM orç </pre>
Projecto #1	SHcontentor.ERR	Desconhecido	<p>PNButtonControl (Owner: FORMEOrdemRecepcao; Owner: ; Owner: ; Owner: LogMessage=Master has detail records. Cannot delete or modify. Key FormEdit_ControlValues=Status: INPROC; Carrier: 046035-LISCONTI-OPEF Screen_ActiveForm_ControlValues=Status: INPROC; Carrier: 046035-LIS</p> <hr/> <p>(Owner: ; Owner: ; Owner: ; Owner:), , "(LOCAL)" 2007/01/04 -"(LC LogMessage=SILENT EXCEPTION: Error registering program enter in fg_</p>
Projecto #2	Errors_SincCS.log	Desconhecido	<pre> 18-10-2011 1:03:22;id = ef9ec2a9-75b3-41b9-9a66-a1675a6c6d41 An exception of type 'System.Net.Mail.SmtpException' occurred and v dateTime =18-10-2011 1:03:22 exceptionType =System.Net.Mail.SmtpException, System, Version=2.0.0 exceptionMessage =Failure sending mail. source = System TargetSite =285041db-4a1a-4c49-bdb6-ad2aed9d54a9 name = Send callingConvention = Standard, HasThis declaringType = System.Net.Mail.SmtpClient </pre>
Projecto #2	Profiling_SincCS.log	Desconhecido	<pre> 08-12-2011 6:52:45;SincCSServer.get_Client 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#xnc 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#vnc 0 08-12-2011 6:52:45;SincCSService.#Pad 0 08-12-2011 6:52:45;SincCSService.#FU 30 08-12-2011 6:53:45;SincCSService.#Pad 0 08-12-2011 6:53:45;SincCSServer.get_Client 0 </pre>
Projecto #2	Trace_SincStats.log	Desconhecido	<pre> 20-12-2011 12:40:48;ITIJ.Citius.Sync.SincStats.Service.SincStatsSer 20-12-2011 12:40:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt 20-12-2011 12:40:48;ITIJ.Citius.Sync.SincStats.Service.SincStatsSer 20-12-2011 12:40:48;ITIJ.Citius.Sync.SincStats.Service.SincStatsSer 20-12-2011 12:41:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt 20-12-2011 12:41:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt 20-12-2011 12:41:48;ITIJ.Citius.Sync.SincStats.Service.SincStatsSer 20-12-2011 12:41:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt 20-12-2011 12:41:48;ITIJ.Citius.Sync.SincStats.Service.SincStatsSer 20-12-2011 12:41:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt 20-12-2011 12:41:48;Enter ITIJ.Citius.Sync.SincStats.Service.SincSt </pre>
Projecto #3	ActionProcessor.log	Desconhecido	<pre> 2011-08-12 15:57:23.6366[Warn]* ExecuteBusinessProcess Started * 2011-08-12 15:57:23.6366[Debug]Obtaining lock... 2011-08-12 15:57:23.6366[Debug]Lock obtained... 2011-08-12 15:57:23.6366[Debug]Creating FRW proxy... 2011-08-12 15:57:23.7148[Fatal]Could not load file or assembly 'CSW.F 2011-08-12 15:57:23.7304[Warn]Releasing lock 2011-08-12 15:57:23.7304[Warn]Lock released 2011-08-12 15:57:23.7304[Warn]* ExecuteBusinessProcess Ended * </pre>

Implementação de um Subsistema de Logging

Projecto #3	MsmqListener.log	Desconhecido	<pre> 2011-08-12 15:57:19.8399 Info MsmqListener.OnReceiveCompleted(): Be 2011-08-12 15:57:19.8868 Info MsmqListener.OnReceiveCompleted(): MS 2011-08-12 15:57:19.8868 Info MsmqListener.OnReceiveCompleted(): De 2011-08-12 15:57:20.5274 Info MsmqListener.OnReceiveCompleted(): Ri 2011-08-12 15:57:24.1210 Info MsmqListener.OnReceiveCompleted(): .. 2011-08-12 15:57:24.1210 Info MsmqListener.OnReceiveCompleted(): Be 2011-08-12 15:57:24.1210 Info MsmqListener.OnReceiveCompleted(): MS 2011-08-12 15:57:24.1210 Info MsmqListener.OnReceiveCompleted(): De 2011-08-12 15:57:24.1366 Info MsmqListener.OnReceiveCompleted(): Ri 2011-08-12 15:57:24.6991 Info MsmqListener.OnReceiveCompleted(): .. </pre>
Projecto #4	ccomum.log	Desconhecido	<pre> 2011-12-16 05:10:52,135 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:52,531 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:53,012 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:53,277 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:53,583 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:53,904 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:54,035 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:54,243 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:55,663 INFO [org.codehaus.xfire.spring.ServiceBes 2011-12-16 05:10:55,939 INFO [org.codehaus.xfire.spring.ServiceBes </pre>
Projecto #5	error-2011-12-19	Desconhecido	<pre> <error errorId="26904beb-da75-43a6-98f0-35ceec15615c" host="VM-EM <serverVariables> <item name="ALL_HTTP"> <value string="HTTP_CONNECTION:Keep-Alive&#xD;&#xA;HTTP_ACC </item> <item name="ALL_RAW"> <value string="Connection: Keep-Alive&#xD;&#xA;Accept: text </item> <item name="APPL_MD_PATH"> <value string="/LM/W3SVC/87257621/Root" /> </pre>

Apêndice 2 – Processo de Desenvolvimento de Software da CSW

As empresas de software, tal como todas as outras empresas, exigem eficácia e eficiência, sendo assim, gestores e programadores devem entender desde o início quais os métodos e práticas que são aplicáveis para resolver o problema que enfrentam. Assumindo este desafio, o documento do Processo de Desenvolvimento de Software descreve as fases de desenvolvimento, processos e organização do ciclo de vida do projecto, fornecendo todas as informações relevantes para a equipa do projecto. O desenvolvimento de software é na sua essência um processo⁵³, onde os inputs são as necessidades do cliente e os outputs o produto final. O processos podem ser visto como uma receita que estabelece um conjunto de passos sequenciais e os recursos necessário para executar determinadas tarefas e assegurar a sua qualidade. No entanto quanto mais complexo for o software, mais complexo será também o seu processo, pelo que uma das formas de reduzir a complexidade de um processo é particionar o processo em processos de âmbito mais contido, obedecendo a uma sequência imposta por necessidades de precedência, tal como podemos constatar pela vista em cascata da sequência de processos de desenvolvimento de software da CSW, presente na figura seguinte.

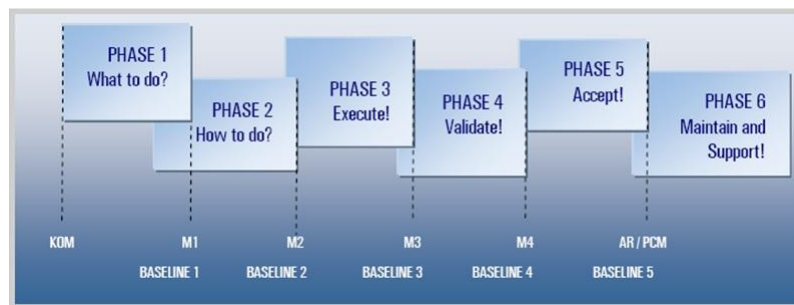


Figura 24 – Ciclo de fase de vida de um software

Nesta imagem conseguimos identificar alguns elementos essenciais do ciclo de desenvolvimento de software, mas para além de fases e processos existem outros elementos importantes como actividades e tarefas. Sendo assim, como é que todos estes elementos se relacionam entre si durante o ciclo de desenvolvimento de software? De uma forma geral o ciclo de desenvolvimento de software está dividido em fases, que por sua vez possuem processos distintos que são compostos por actividades e tarefas específicas. A imagem seguinte ajuda-nos a ter uma melhor visão do ciclo de desenvolvimento de software e dos elementos inerentes.

⁵³ O artigo ISO - ISO 9008:2000 – “Guidance on the concept and use of the process approach for management systems” define um processo da seguinte forma: “A ‘Process’ can be defined as a “set of interrelated or interacting activities, which transforms inputs into outputs”.

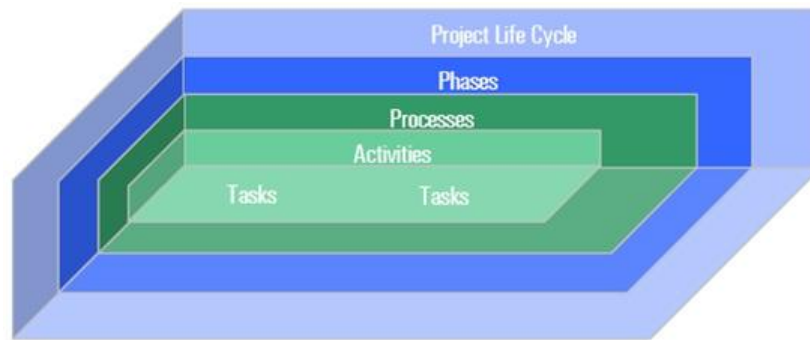


Figura 25 - Elementos do processo de desenvolvimento de software

- **Fases** - Etapas distintas na execução de um projecto que juntas constituem o ciclo de vida do projecto. As fases são um indicador do focus do projecto num determinado momento no tempo e como tal são um elemento essencial de control e gestão;
- **Processos** - Conjunto de recursos e actividades interrelacionados que transformam inputs em outputs;
- **Actividades** - Práticas de engenharia ou gestão que contribuem para a criação de artefactos do processo, ou para a evolução do próprio processo. Uma actividade pode ser descrita como um documento, um procedimento ou um conjunto de tarefas;
- **Tarefas** - Parte específica do trabalho a realizar para concluir uma actividade. Uma actividade pode ser implementada por uma ou mais tarefas.

Um dos processos de desenvolvimento de software mais conhecido é o Rational Unified Process (RUP), um processo de desenvolvimento de software iterativo proprietário da IBM⁵⁴. Mas ao contrário do que se possa pensar, o RUP não é uma abordagem fechada ao desenvolvimento de software, mas sim um conjunto flexível de processos, destinado a ser adaptado pelas organizações e equipas de desenvolvimento de software, consoante as suas necessidades.

O RUP é baseado num conjunto de elementos de conteúdo que descrevem o que deve ser produzido, os conhecimentos necessários e explicações detalhadas sobre como os objectivos deve ser atingidos.

Esse elementos são os seguintes:

- **Roles** (Quem?) – Define competências e responsabilidades;
- **Work Products** (O quê?) – Representa o resultado de uma tarefa, incluindo todos os documentos e artefactos produzidos durante a realização do processo;
- **Tasks** (how) – Descreve uma unidade de trabalho atribuída uma “Role”;

⁵⁴ RUP foi criado pela Rational Software Corporation, adquirida pela IBM, ganhando um novo nome IRUP que agora é uma abreviação de IBM Rational Unified Process e tornando-se uma brand na área de Software, fornecendo técnicas a serem seguidas pelos membros da equipe de desenvolvimento de software com o objetivo de aumentar a sua produtividade no processo de desenvolvimento.

Um projecto baseado em processos RUP avança em incrementos chamados iterações, onde o objectivo de cada iteração é produzir artefactos que possam validados pelos stakeholders e que traga valor acrescentado ao projecto. Os artefactos produzidos em cada iteração devem abranger todos, ou pelo menos a maioria, os subsistemas do projecto.

Dentro de cada iteração as tarefas são divididas em disciplinas, nomeadamente disciplinas de suporte:

- **Configuration and Change Management** - Registrar e controlar todas as versões de todos os artefactos do projecto;
- **Project Management** - Gerir calendários e recursos;
- **Environment** - Configuração e manutenção do ambiente de desenvolvimento.

e disciplinas de engenharia:

- **Business Modeling** - Definir necessidades do negócio;
- **Requirements** - Traduzir de necessidades de negócio em comportamentos do sistema;
- **Analysis and Design** - Traduzir de requisitos em arquitectura de software;
- **Implementation** - Criar software que se adequa à arquitectura e que possua os comportamentos necessários;
- **Test** - Garantir que todos comportamentos estão correctos e implementados;
- **Deployment** - Reunir todos os artefactos necessário para executar o projecto.

Estas disciplinas não estão separadas no tempo, pelo contrário, estas são executadas simultâneamente, com graus de intensidade diferentes, durante toda a vida do projecto, tal como podemos ver pela figura seguinte.

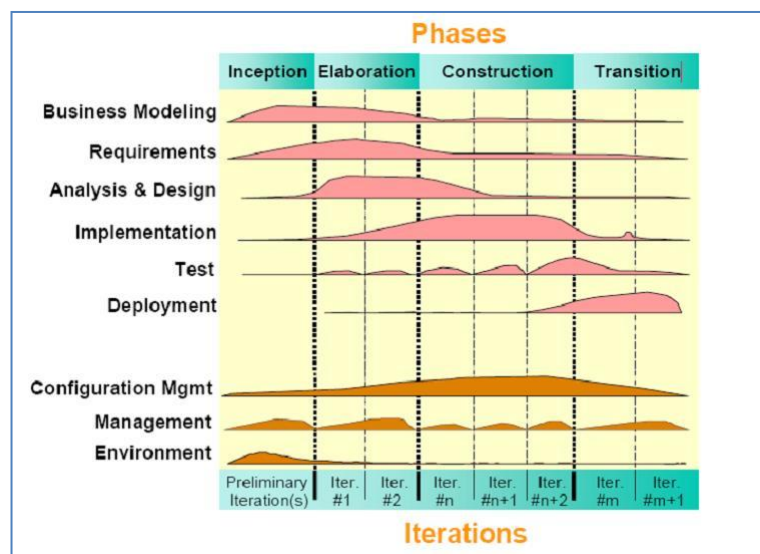


Figura 26 – Rational Unified Process

Isto significa, por exemplo, que não é escrito muito código durante as primeiras fases do projecto, o que não quer dizer que não se escreva código nenhum. E em contrapartida, na fase final do projecto a maioria dos requisitos já são conhecidos, o que não quer dizer que novos requisitos não possam ser identificados nesta fase.

Tendo este conceitos em consideração passamos de seguida para uma visão mais detalhada do ciclo de vida de desenvolvimento de software da CSW, onde é possível identificar as fases, as milestones e o estado do projecto.

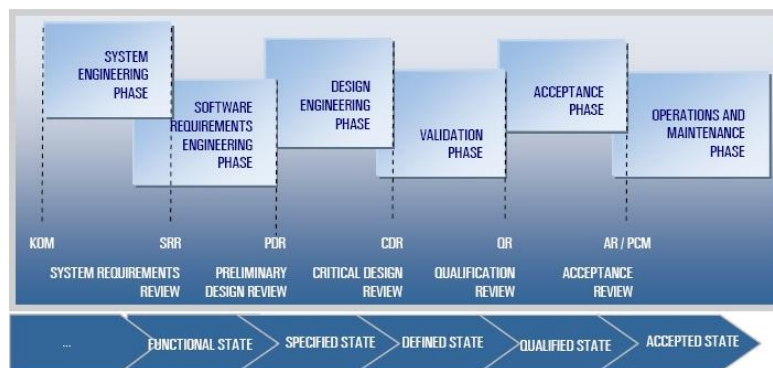


Figura 27 – Ciclo de fases de vida de um software (detalhado)

Esta imagem permite-nos perceber que as fases são delimitadas pelas milestones, onde revisões de qualidade devem ser realizadas, e que a realização das milestones determina o estado do projecto. Mais uma vez trata-se de uma vista em cascata, o que nos leva a concluir que abordagens em espiral podem ser representadas como cascatas sequenciais.

De seguida iremos descrever mais detalhadamente cada uma das fases do ciclo de vida do processo de desenvolvimento de software, dando especial ênfase aos objectivos de cada fase, o tipo de artefactos que é suposto gerarem e as suas milestones.

- **Fase de Engenharia do Systema** - Diz respeito à fase de elaboração e definição do problema. Tem início na Kick-Off-Meeting e no final deve ser realizada uma Revisão dos Requisitos do Sistema (SRR);
- **Fase de Requisitos de Engenharia** - Consiste na elaboração da especificação técnica, como "resposta" às necessidades dos clientes. A especificação técnica contém uma definição precisa e coerente de funções, desempenhos, calendarizações, custos e planos de implementação para todos os níveis do software a ser desenvolvido. Esta fase deve garantir também que os requisitos e especificações são viáveis, completos e consistentes, e que o cliente e a que a equipa desenvolvimento os compreendem da mesma forma;



Figura 28 – Fase de Requisitos de Engenharia

- **Fase de Desenho de Engenharia** - Produz o desenho detalhado do sistema, o código do software e dos Testes Unitários de cada elemento do software, em resposta às exigências contidas na especificação técnica. Durante esta fase, produzir o desenho detalhado do sistema paralelamente com o código fonte, permite que o desenho seja gerado através do código usando “ferramentas de reengenharia”. Da mesma forma, produzir Testes Unitários em paralelo com a criação de código permite identificar e corrigir erros antecipadamente;



Figura 29 – Fase de Desenho de Engenharia

- **Fase de Validação** - O objectivo da fase de validação é verificar que as funcionalidades do sistema satisfazem todos os requisitos e especificações. Durante esta fase, a equipa de testes do sistema executa os testes especificados no sistema, os resultados obtidos são documentados e a equipa de desenvolvimento é notificada sobre quaisquer discrepâncias. As correcções de software são realizadas pela equipa de desenvolvimento, em conformidade com procedimentos rigorosos de gestão de configuração. Depois de corrigido o software é retestado pela equipa de testes e são executados testes de regressão para assegurar que as funções anteriormente testadas não foram afectadas.



Figura 30 – Fase de Validação

- Fase de Aceitação** - O objectivo da fase de aceitação é demonstrar que o sistema atinge os seus requisitos no ambiente operacional. Os testes realizados durante esta fase são realizados sob a supervisão de uma equipa independente de teste de aceitação (em nome do cliente) e segue os procedimentos previstos no plano de teste de aceitação. Por norma a equipa de desenvolvimento dá formação e apoio à equipa independente de testes e posteriormente é também responsável por corrigir os erros detectados e actualizar a documentação sempre que necessário. Nesta fase são ainda realizados inquéritos de qualidade e o Questionário de Satisfação do Cliente é produzido, onde a meta é obter um grau de satisfação de 8 em 10.



Figura 31 – Fase de Aceitação

- **Fase de Manutenção e Operações** - Esta fase é iniciada após a conclusão da revisão aceitação do software. É uma fase muito importante uma vez que o sucesso do software está dependente do nível de operações que este pode proporcionar ao cliente. O processo de manutenção é activado quando o software sofre alguma alteração de código ou documentação associada, como resultado de correção erros ou da implementação de uma melhoria ou adaptação. As operações e manutenção podem ser invocadas durante o período de garantia ou através de um contrato de manutenção, e terminam quando o software for descontinuado, quanto terminar a garantia ou quando terminar o contrato de manutenção.



Figura 32 – Fase de Manutenção

Tal como referimos anteriormente, a abordagem descrita anteriormente retrata uma visão muito madura e certificada dos processos de desenvolvimento de software, no entanto nem todos os problemas e clientes são iguais, pelo que, é natural que se façam adaptações consoante as necessidades do projecto. A Critical Software criou uma visão mais simplificada dos processos de desenvolvimento de software que se adequa melhor a alguns dos seus projectos, pois facilita a sua compreensão e contextualização, como se pode ver pelo seguinte diagrama.

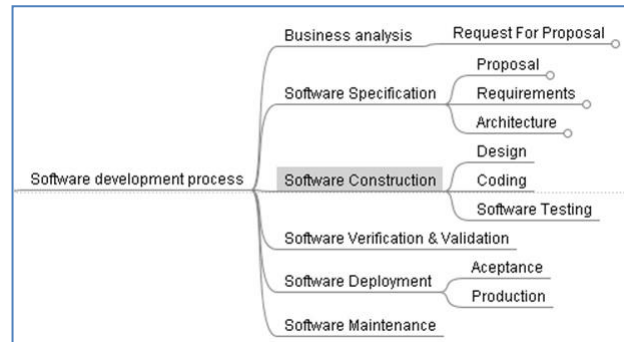


Figura 33 – Processo de desenvolvimento de software simplificado

O diagrama anterior retrata algumas particularidades, como a inclusão da “Request for Proposal” na fase de análise do negócio. Isto acontece porque é normal a CSW ser chamada pelo cliente para fazer parte da definição do problema. Outra particularidade que é necessário reconhecer é o facto de a proposta fazer parte da especificação do sistema, pois a experiência da empresa já revelou que este artefacto pode determinar o sucesso ou fracasso do projecto. A realização de testes⁵⁵ é uma das actividades que por vezes é subestimada na fase de construção, no entanto é de extrema importância para o desenvolvimento de software pois são os testes que garantem que o projecto está a ir no bom caminho. Esperar que o projecto esteja concluído para realizar os primeiros testes pode ser um erro fatal.

Mas como já referimos, todos os projectos têm contextos diferentes como maturidade técnica, criticalidade da solução, janela temporal, orçamento, etc, no entanto existe um ponto que é fundamental e que é transversal a todos os projectos: Se a informação é necessária é preciso tê-la! E quanto mais tarde a tivermos maior é risco! A ambiguidade é nossa inimiga e pode afectar o resultado do projecto em termos de âmbito, validação, qualidade, planeamento, coesão de arquitectura, reutilização, etc. A solução é adaptar-mo-nos às circunstâncias e se não podermos ter toda a informação necessária numa fase, temos de a ter obrigatoriamente na fase seguinte⁵⁶, sob pena de comprometermos o resultado do projecto.

Quando falamos do RUP, falamos do conceito de “Roles”, que define competências e responsabilidades dentro de um projecto. A definição de das diferentes “Roles” e maneira como estas interagem também faz partes dos processos de desenvolvimento de software. Nos projectos da CSW as “Roles” estão normalmente divididas da seguinte forma.

⁵⁵ Testes de integração, performance, segurança, etc.

⁵⁶ Se não tivermos toda a informação na fase de RFP, temos de a ter na fase da Proposta; se não a tivermos na fase da Proposta, temos de ter na fase de Levantamento Requisitos; se não estiver especificada na fase de Levantamento de Requisitos e Desenho da Arquitectura, temos de o fazer na fase de Implementação.

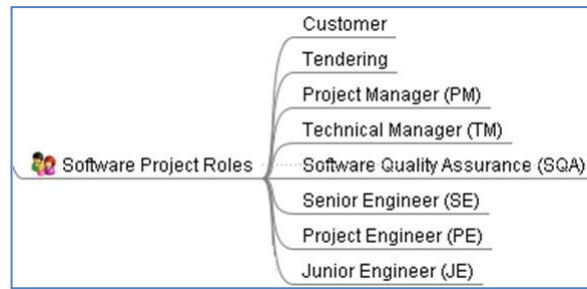


Figura 34 – Responsabilidades

Neste caso não nos queremos focar na descrição das competências e responsabilidades de cada uma das “Roles” até porque estas podem mudar de projecto para projecto, o que é importante salientar aqui é que o software profissional é desenvolvido por equipas e não por indivíduos e que uma equipa precisa de uniformização processos para conseguir trabalhar em conjunto e produzir software de qualidade.

Resumindo, porque é que usamos processos? A resposta é simples:

‘Porque temos medo! Porque temos medo que:

- *O projecto produza o produto errado;*
- *O projecto produza um produto de qualidade inferior;*
- *O projecto se atrase;*
- *Tenhamos de trabalhar 80 horas por semana;*
- *Tenhamos de quebrar compromissos;*
- *Não nos vamos divertir.*

*Os nossos medos motivam-nos a descrever processos que restrinjam as nossas actividades e que exijam determinados resultados. Estas restrições e resultados são retirados de experiências anteriores, através da escolha the solução que tenham funcionado bem noutros projectos. A esperança é que essas soluções voltem a funcionar e que afastem os nossos medos.”*⁵⁷

De outro ponto de vista, é preciso ter consciência que o individuo tem um papel extremamente importante na execução e aperfeiçoamento de um processo, pois os processos não prevêm todos os cenários, os processos apenas dizem o que fazer e quando fazer, o individuo precisa de saber como fazer, sem esquecer que os processos não são “sagrados”, é necessário ser assegurar os objectivos do projecto;

Resumindo, o processo de desenvolvimento de software tem como principais objectivos fornecer orientação adequada às equipas dos projectos, e confiança aos clientes que o resultado final satisfaz as expectativas e necessidades do projecto. Como tal, é um componente chave do Sistema de Gestão de Qualidade da CSW, e a sua correcta aplicação contribui directamente para que a empresa cumpra os requisitos da norma ISO9001 para Gestão de Qualidade e para a certificação CMMI de nível 5. Os processos de desenvolvimento software descritos são uma interpretação da CSW para a norma ISO12207 para o ciclo de vida de um software.

⁵⁷ Esta é a justificação dada por Grady Booch, Robert C. Martin e James Newkirk no livro “Object Oriented Analysis and Design with Applications 2d. ed.”.

Apêndice 2 – Benchmark de Concatenação de strings

A concatenação de *strings* é uma das operações essenciais na construção das mensagens de logging, pois estas mensagens são constituídas por vários elementos como o *timestamp* ou a descrição do evento. Para a medição das operações de concatenação de *strings* foram selecionadas algumas das abordagens mais comuns e que são descritas em cada um dos seguintes métodos:

- **ControlEmptyMethod** - É um método vazio que serve de comparação para os outros métodos. Isto permite-nos ignorar custos externos não relacionados com operações de concatenação.
- **SumLiteralStrings** - Método onde a concatenação é feita através da soma de *strings* literais, ou seja, não há recurso ao uso de variáveis. Esta abordagem serve também como base de comparação visto ser impossível aplicá-la num cenário real de construção de mensagens de log, onde os seus elementos são dinâmicos.
- **SumStrings** - Ao contrário do método “SumLiteral Strings”, este método realiza a concatenação através da soma de variáveis (previamente instanciadas).
- **SumMixedStrings** - Este método junta duas abordagens, a concatenação de *strings* literais com a concatenação de variáveis, para realizar a concatenação das *strings*.
- **ConcatLiteralStrings** - Método que recorre ao uso da função nativa da linguagem .Net “String.Concat()” para realizar a concatenação de strings literais.
- **ConcatStrings** - Também este método usa a função nativa da linguagem .Net “String.Concat()” para realizar a concatenação, mas neste caso a concatenação é realizada sobre variáveis.
- **StringFormat** - Método que usa a função “String.Format()”, nativa da linguagem .Net para realizar a concatenação de variáveis.
- **StringBuilder** - Neste caso a concatenação de *strings* é feita através do uso da classe `StringBuilder`, nativa da linguagem .Net.

Para cada medição são apresentados valores para os seguintes parâmetros: Média de *ticks* que cada método gastou, número total de *ticks* para cada abordagem, o número de *ticks* da primeira chamada a cada método, o máximo e o mínimo de *ticks* gastos em cada uma das diferentes abordagens. Estes valores são apresentados numa tabela e são acompanhados por um gráfico de barras horizontais que nos permite entender melhor os custos de performance de cada abordagem.

Apêndice 3 – Benchmark de formatação de Timestamp

Um dos elementos mais importantes das mensagens de log é o elemento de *timestamp*, pois é este elemento que ajuda a ordenar os eventos num registo de log. Sem o elemento de *timestamp* ou com um elemento de *timestamp* mal definido seria praticamente impossível consultar um registo de log. Sendo um elemento importante e usado recorrentemente convém saber se a abordagem de formatação de *timestamp* usada pode ou não ter influência significativa na performance de escrita de uma mensagem de log. Assim sendo foram seleccionadas algumas das abordagens mais comuns na formatação do elemento de timestamp em conjunto com algumas abordagens menos convencionais, baseadas em conceitos matemáticos:

- **EmptyMethod** - Método vazio que serve de comparação para os outros métodos. Isto permite-nos ignorar custos externos não relacionados com operações de concatenação;
- **StringBuilder** - Método que usa a classe “StringBuilder” nativa do .Net, através da função “AppendFormat()” e que usa uma variável externa do tipo “DateTime.Now”;
- **StringFormat** - Método que usa a função “String.Format” nativa do .Net, em conjunto com uma variável externa do tipo “DateTime.Now”;
- **ToString** - Este método aplica a função “ToString” nativa do .Net à variável externa do tipo “DateTime.Now”;
- **FastDateTimeToStr** - Este método utiliza um método externo estático para obter o formato para o timestamp. Este método externo utiliza uma abordagem menos convencional inspirada no artigo “*Performance Quiz: Fastest way to format time*”;
- **FastDateTimeToStrLocal** - Método que utiliza uma instância local de um método que usa a abordagem menos convencional, inspirada no artigo “*Performance Quiz: Fastest way to format time*”, referida previamente;
- **FastSimpleNoCache** - Método que utiliza a mesma abordagem usada nos métodos anteriores, mas sem cache para o acesso às propriedades da variável do tipo “DateTime.Now”.

Apêndice 4 – Diagrama de Responsabilidades

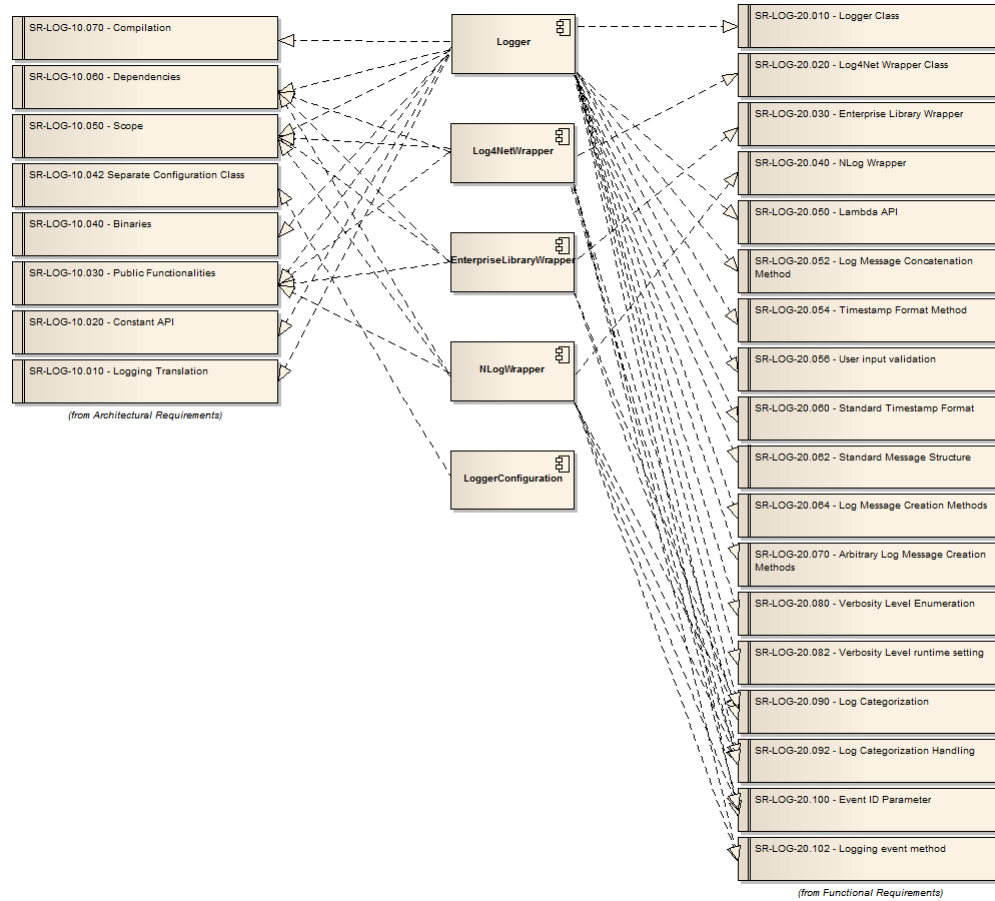


Figura 35 – Diagrama de Responsabilidades

Apêndice 5 – Frameworks de logging para .Net

Apache log4net

O log4net é uma ferramenta que ajuda o programador a criar registos de execução de software e a guardá-los em diferentes formatos. O log4net surgiu como uma alternativa ao Apache log4j, com provas dadas na linguagem Java, para a framework .Net da Microsoft, mantendo a mesma ideologia do log4j, mas aproveitando as vantagens oferecidas pelo .NET.

O log4net permite controlar o volume de log gerado e classificar cada registo de log de acordo com o nível de severidade da mensagem a ser registada. Mas naturalmente esta framework suporta uma vasta gama de características, que passaremos a elencar e descrever de seguida.

Características:

- Suporta várias frameworks - Incluindo todas as versões da *framework* .Net até à versão 4.0, as versões 1.0 e 2.0 da *framework* .Net Compact ou as versões 1.0 e 2.0 da framework Mono⁵⁸;
- Persistência dos registos em múltiplos formatos - A persistência é realizada através de “appenders” que suportam diferentes formatos, entre os quais, ficheiros, bases de dados, email ou o Windows Event Log;
- Configuração através de XML - A informação de configuração pode ser embutida nos ficheiros de configuração da própria aplicação, em ficheiros separados ou programaticamente;
- Configuração dinâmica - As configurações podem ser alteradas em *runtime*;
- Contexto de Logging - O log4net pode ser usado para guardar contexto sobre a aplicação, de forma a ser inserido automaticamente nas mensagens de log;

Arquitectura Comprovada - Baseado na framework log4j, desenvolvida em 1996, testada exaustivamente e com resultados comprovados.

Microsoft Enterprise Library – Logging Application Block

O LAB é uma ferramenta de logging distribuída pela Microsoft juntamente com a framework Enterprise Library. O Enterprise Library é um conjunto de blocos aplicacionais, desenvolvidos para auxiliar os programadores em alguns dos desafios mais comuns do desenvolvimento de software, como por exemplo o logging.

O LAB simplifica a implementação de operações de logging, desde a criação das próprias mensagens, à própria configuração do sistema de logging através da sua Consola de Configuração. À semelhança das outras frameworks de logging o LAB permite escrever registos de log para uma variedade de “targets” diferentes.

Apesar de o log aplicacional não dever conter informação de “negócio” (isso é função da auditoria), uma das limitações apontadas para esta framework é o facto de esta não encriptar a informação de log, o que pode trazer alguns problemas de segurança. No entanto esta

⁵⁸ Uma framework open-source e multi-plataforma para a linguagem C# compatível com a framework .Net da Microsoft.

limitação pode ser ultrapassada de duas maneiras, através da utilização de listas de controlo de acessos (ACL) para ficheiros ou através da criação de um formatador de mensagens próprio que encripte a informação.

Jaroslav Kowalski's NLog

NLog é uma framework de logging “open-source” desenvolvida para as plataformas .Net, Silverlight e Windows Phone. À semelhança das restantes frameworks de logging descritas neste trabalho, esta *framework* permite criar registos de logging com vários níveis de contextualização e enviar esses mesmos registos para diferentes tipos de “targets”. Para melhor descrever esta framework, iremos apresentar algumas das suas principais características.

“Targets” suportados:

- Consola - Semelhante ao comando Console.WriteLine(), incluindo codificação de cores;
- Ficheiros - Um ou vários ficheiros em simultâneo;
- Event Log - Local ou remoto;
- Bases de Dados - Armazena logs em Bases de Dados suportadas pelo .Net;
- Rede - Através dos protocolos TCP, UDP, SOAP e MSMQ;
- Email - Alertas para erros através de email;
- ... entre outros#.

Outras características:

- Configuração através de um ficheiro de configuração ou programaticamente;
- Utilização do *logger pattern* inspirado nas soluções log4xxx;
- “Routing” com recurso a “buffering”, assincronismo, balanceamento de carga e failover;
- Suporta as frameworks .Net, .Net Compact e Mono (Windows e Unix).

Esta framework é usada em diversos softwares comerciais e “Open Source”, de seguida apresentamos uma lista de alguns desses softwares.

Open Source Software:

- Castle Project/Windsor Container;
- Ninject;
- Common Infrastructure Libraries for .Net;
- TeamCity Configuration Monitor.

Commercial Software:

- Gibraltar;
- TradeBullet;
- LiteStock;
- Silverlight Map Server.

Os logs produzidos pelo NLog podem ser analisados por várias ferramentas, desde de *viewers* específicos para o NLog até *viewers* para as soluções log4xxx.

Ferramentas específicas para o NLog:

- Gibraltar (usando o adaptador para o NLog);
- Sentinel;
- NLogViewer.

Ferramentas para as soluções log4xxx compatíveis com o NLog:

- Apache Chainsaw;
- DevIntelligence Log4NetViewer;
- C# WPF Log4Net Viewer
- Log4Net Dashboard;
- Log4View;

Log2Console.

Apêndice 6 – “Façades” de logging para .Net

Common Logging

O projecto “Common Infrastructures Libraries for .Net”, tinha como principal objectivo fornecer bibliotecas de leve infraestrutura, que pudessem ser usadas em vários projectos. A primeira dessas bibliotecas a ser desenvolvida, e única até ao momento⁵⁹, foi uma abstracção de logging chamada Common.Logging. Esta biblioteca surgiu, à semelhança do subsistema de logging da CSW, como forma de solucionar o problema de não haver um interface comum entre as diferentes frameworks de logging. Neste caso, o Common.Logging introduz um nível de abstracção que permite escolher a framework de logging em *runtime*, ou aquando da compilação, usando sempre o mesmo interface. Novas frameworks de logging podem ser adicionadas à biblioteca através do uso de *wrappers*. A biblioteca é baseada no trabalho feito pelos programadores da framework IBatis.NET⁶⁰, e a sua utilização é inspirada na framework log4net. A primeira versão foi lançada em Março de 2007 e a última versão (v2.0.0) data de Fevereiro de 2010.

De acordo com algumas referências, Esta ferramenta é inclusivamente utilizada em projectos reais, como é o caso da framework Spring .Net⁶¹. Uma das desvantagens apontadas pela comunidade é o facto de não ser compatível a ferramenta Silverligh da Microsoft.

Simple Logging Façade

A SLF é, de acordo com os seus autores, uma framework com uma missão simples mas ambiciosa: fornecer ao programador, a capacidade de integrar facilmente funcionalidades de logging nas suas aplicações. Como o próprio nome indica, esta ferramenta procura ser simples, uma vez que permite adicionar funcionalidades de logging comprovadas através da implementação de uma única linha de código, enquanto proporciona a possibilidade de adaptar a qualquer cenário de logging. Por outro lado procura ser igualmente flexível, pois disponibiliza uma interface comum que abstrai a utilização das frameworks de logging escolhidas. Esta abordagem elimina quaisquer dependências externas, pelo que permite aternar entre as três frameworks de logging suportadas (log4net, LAB e Nlog) a qualquer altura.⁶²

Entre as várias funcionalidades das ferramenta destacamos as seguintes:

- É configurável através de código ou de ficheiros de configuração;
- Suporta “named loggers” com mecanismos hierárquicos de fallback;
- Suporta as frameworks de logging log4net, LAB e Nlog;
- Facilita a inclusão de novas frameworks de logging;
- Possibilita alternar entre framework de logging a qualquer altura (mesmo em runtime);

⁵⁹ Outras bibliotecas estão previstas, mas ainda não existe um planeamento. Sugestões vão desde uma biblioteca de validação até utilitários para armazenamento local através de threads.

⁶⁰ IBATIS é uma framework de persistência que automatiza o mapeamento entre bases de dados SQL e objetos .NET.

⁶¹ De acordo com a documentação oficial do projecto: “Spring Framework Reference – Chapter 13: Common Logging”

⁶² Informação retirada dos artigos “Simple Logging Façade” em Codeplex.com, “The Art of Logging” em Codeproject.com e “SLF Hands-on Tutorial” em Hardcodet.net.

- Uma arquitectura modular e extensível permite expandir o sistema para cobrir cenários de log não previstos;
- Em caso de configuração inválida o sistema envia um alerta em vez de quebrar a execução da aplicação;
- Possui vários exemplos prontos a utilizar;
- É compatível com a versão 2.0 do .Net.

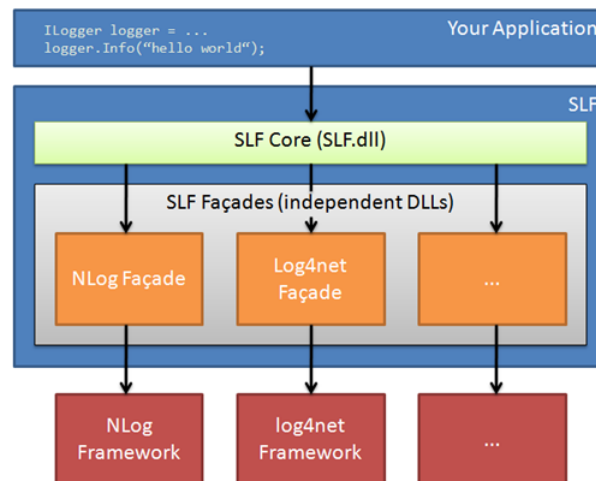


Figura 36 – Simple Logging Façade⁶³

Castle Logging

Esta ferramenta é uma ferramenta desenvolvida pelo projecto Castle⁶⁴, distribuída juntamente com o Windsor (outra ferramenta desenvolvida por este projecto) que tem como objectivo fornecer uma abstração para as frameworks de logging desejadas. Actualmente suporta a utilização de apenas duas frameworks de logging externas, o log4net da Apache e o Nlog desenvolvido por Jaroslaw Kowalski. À semelhança das restantes frameworks e *façades* de logging descritas neste trabalho, o Castle Logging permite a configuração do sistema através de um ficheiro de configuração (XML) ou através de código.⁶⁵

⁶³ Figura retirada do artigo “An Introduction to SLF, the Simple Logging Façade” em Hardcodet.net.

⁶⁴ Mais informação sobre este projecto no site Castleproject.org.

⁶⁵ A escassa informação aqui disponível foi retirada da documentação oficial do projecto, mais concretamente do artigo “Logging Facility”.

Apêndice 7 – Exemplo de utilização de AOP

Este apêndice apresenta um exemplo de utilização do AOP e demonstra as suas vantagens em termos de legibilidade e manutenção de código.

Sem AOP a nossa abordagem seria possivelmente a seguinte:

```
Console.WriteLine("Estamos a entrar no métodoA");
MetodoA();
Console.WriteLine("Estamos a sair do métodoA");

Console.WriteLine("Estamos a entrar no métodoB");
MetodoB();
Console.WriteLine("Estamos a sair do métodoB");
```

Como podemos constatar, esta abordagem “polui” imenso o código e torna a sua manutenção e legibilidade bastante difíceis. Já no caso do AOP a abordagem seria bastante diferente!

Começemos pelas configurações necessárias:

```
[Serializable]
public sealed class TraceAttribute : OnMethodBoundaryAspect
{
    private readonly string _category;

    public TraceAttribute(string category)
    {
        _category = category;
    }

    public override void OnEntry(MethodExecutionArgs args)
    {
        Trace.WriteLine(string.Format("Estamos a entrar no ", _category));
    }

    public override void OnExit(MethodExecutionArgs args)
    {
        Trace.WriteLine(string.Format("Estamos a sair do ", _category));
    }
}
```

Aqui é criado um aspecto do tipo “Trace” (que tem de ser Serializable) através da extensão do objecto *OnMethodBoundaryAspect* e que pode ser aplicado a qualquer método. Este atributo possui um construtor que irá receber uma categoria do tipo *string* e irá guarda-la numa variável privada. De seguida são criados *overrides* dos métodos *OnEntry* e *OnExit* que serão executados, respectivamente, antes e depois da invocação do método.

```
[Trace("metodoA")]
Private static void MetodoA()
{
    ...
}

[Trace("metodoB")]
Private static void MetodoB()
{
    ...
}
```

O atributo é depois aplicado aos métodos juntamente com a categoria para o construtor. E finalmente para obter o mesmo output da abordagem sem AOP, precisamos apenas de fazer o seguinte:

```
MetodoA() ;  
MetodoB() ;
```

É fácil de perceber que esta abordagem melhora imenso a legibilidade e manutenção do código, uma vez que não precisamos de replicar o mesmo código várias vezes e se for necessário modificar algum comportamento, essa modificação será feita apenas num determinado local.

Apêndice 8 – Valores dos Benchmarks

Este apêndice apresenta os valores obtidos nas medições realizadas durante este trabalho. De maneira a isolar as medições, as configurações foram carregadas previamente e foi realizada uma chamada de log para assegurar que os custos iniciais de instânciação do sistema ficassem fora das medições. Os resultados são expressos em *ticks*.

Frameworks de Logging

Para esta medição foi utilizada uma *string* simples ("Log4Net Info Message Log!") que não requeria processamento adicional para concatenar parâmetros.

	Log to File	Log to Event Viewer
log4net	58	2078
Logging Application Block	73	281
Nlog	1156	2061

Tabela 3 – Benchmark de Frameworks de Logging

Uso de “Reflection” na construção de mensagens de log

Para esta medição o método de “reflection” utilizado foi o seguinte:

MethodBase.GetCurrentMethod().DeclaringType

	LogWithReflection	LogWithoutReflection
Logging	76	60

Tabela 4 – Benchmark de uso de “reflection” na construção de mensagens de log