# 5
# Avaliação do Sistema de Críticas

Este capítulo descreve uma avaliação qualitativa, que foi realizada com usuários, sobre o NCL-Inspector. Está dividido em duas partes: a primeira, faz uma descrição detalhada do estudo e a segunda apresenta detalhes dos resultados obtidos com o estudo.

## 5.1
## Descrição do estudo

### 5.1.1
### Objetivos do estudo

O objetivo principal do estudo foi avaliar a facilidade de um desenvolvedor ao criar uma regra de inspeção utilizando o NCL-Inspector. Através das dificuldades encontradas pelos desenvolvedores ao utilizar a ferramenta, buscamos identificar oportunidades de melhorias do NCL-Inspector. Essas melhorias encontradas são descritas em detalhes no capítulo que aborda os trabalhos futuros (Capítulo 6).

Como objetivo secundário, gostaríamos de saber se o esforço de criação de regras para o NCL-Inspector é aceitável, se contrastarmos com o esforço de uma inspeção manual de um programa NCL.

### 5.1.2
### Perfil dos participantes

Para realização do estudo, o perfil buscado foi o de pessoas experientes com desenvolvimento de software. Os participantes deveriam ter sólidos conhecimentos de XML, conhecimentos básicos de NCL e bons conhecimentos nas tecnologias Java ou XSL. Além disso, devem ter uma noção do que é o ambiente de desenvolvimento Eclipse.

Para verificar o perfil de um determinado participante, um questionário pré-avaliação contendo seis perguntas foi feito antes de realizar a avaliação. Este questionário encontra-se no Apêndice C.

### 5.1.3 Cenário

Um cenário é uma narrativa textual ou pictórica, rica em detalhes contextuais, de uma situação de uso de uma aplicação, envolvendo usuários, processos e dados reais ou potenciais. Um cenário típico é uma história sobre pessoas realizando uma atividade (Rosson et al. 2002).

O ator do cenário utilizado é um estudante de Ciência da Computação, um entusiasta de novas tecnologias e com experiência em desenvolvimento para a TV Digital. A atividade que o estudante realiza é a de criar uma regra, que inspecione um determinado problema recorrente em suas aplicações, utilizando o NCL-Inspector. A seguir o cenário completo utilizado no experimento:

> Paulo é um universitário de uma das universidades mais bem conceituadas do país. Está próximo de finalizar o curso de Ciência da Computação. Ele é um aluno muito dedicado e apaixonado pelo que faz. Suas notas durante o curso são uma das mais altas entre os seus colegas.
>
> Amante de novas tecnologias, Paulo se interessa pelo desenvolvimento de aplicações em NCL para TV Digital. Apesar do grande interesse, Paulo sente falta de um maior suporte de ferramentas para o desenvolvimento de aplicações NCL. Foi então que ele descobriu que na PUC-Rio está sendo criada uma ferramenta com esse objetivo, o NCL-Inspector.
>
> O NCL-Inspector é uma ferramenta de análise estática de código para NCL. Um dos seus maiores objetivos é detectar problemas de codificação em programas escritos em NCL. Outro grande objetivo de projeto do NCL-Inspector é que ele seja facilmente extensível. Possui uma arquitetura de plugins que permite a inserção de novas regras sem a necessidade de recompilar todo o seu código. Permite que as regras sejam implementas através de Java ou Folhas de Estilo XML (XSL).
>
> O desenvolvedor do NCL-Inspector tem o desejo de que a ferramenta fosse facilmente estendida e que também favorecesse o desenvolvimento distribuído de regras de inspeção. Em anexo

é possível ver a descrição de um problema de código NCL muito recorrente nos programas escritos por Paulo. A ferramenta NCL-Inspector não faz esse tipo de validação, porém, Paulo gostaria de estendê-la de forma que recebesse uma notificação da ferramenta caso esse problema ocorra.

Se você fosse Paulo, como implementaria uma regra que verificasse a ocorrência desse problema? Prefere implementar a regra usando Java ou XSL?

### 5.1.4 Procedimento

Antes de iniciar a avaliação, em cumprimento à Resolução 196/96 do Conselho Nacional de Saúde, cada participante assinou um termo de consentimento. O termo esclarece como serão usados os dados coletados durante o experimento. Além disso, deixa claro que os dados serão usados mantendo o anonimato e a privacidade dos participantes.

Em linhas gerais, o experimento consistiu em apresentar um problema de código NCL a cada participante. Ao tomar conhecimento do problema, cada um deveria criar uma regra a fim de estender o NCL-Inspector, de modo que, agora, ele passe a detectar esse tipo de problema. O Apêndice D possui todos os problemas que foram usados na avaliação. Optamos por utilizar problemas ligeiramente modificados para cada participante. Embora não fossem iguais, os problemas possuiam o mesmo nível de dificuldade ou eram instâncias do mesmo tipo.

Após a leitura da descrição do problema, o participante deveria ler o Manual de Criação de Regras do NCL Inspector. A leitura desse manual é importante, pois explica passo-a-passo como criar regras utilizando Java e XSL.

Para a criação das regras, o participante poderia optar livremente por usar Java ou XSL. Acreditamos que não fixar uma tecnologia seria uma opção melhor por dois motivos: (1) não faz parte do objetivo desse experimento avaliar se é mais fácil implementar regras usando Java ou XSL; (2) cada participante poderia implementar usando a tecnologia que fosse mais familiar, ou a que ele acreditasse que lhe permitiria completar a tarefa mais rapidamente.

Para realização do teste, foi utilizado o ambiente Eclipse para escrever as regras. O participante iniciava o teste com o ambiente já configurado, pois isso não faz parte do escopo da avaliação, além de reduzir drasticamente o tempo do experimento.

Para avaliar se o participante completou corretamente a sua tarefa, foram elaborados testes automatizados utilizando a ferramenta JUnit e o Mini-*framework* de teste do NCL-Inspector. Os testes consistiam em fornecer um ou mais arquivos NCL contendo propositalmente as violações, que as regras criadas pelos participantes deveriam detectar. Através do mini-framework de testes do NCL-Inspector, os participantes verificavam se a regra implementada detectava todas as violações. Além disso, foi fornecido um arquivo correto para verificar se a regra não estava emitindo falsos positivos. Os arquivos NCL utilizados foram criados especificamente com o propósito de teste. Para ensinar ao participante como testar a regra, foi criado o Manual de Execução de Testes do NCL-Inspector.



Durante a realização da tarefa, o avaliador seguiu um roteiro, chamado de Ficha de Apoio à Observação de uso (Apêndice E). Esse roteiro continha questões-chave que deveriam ser observadas durante a avaliação. Após a realização do teste, o participante foi submetido a uma entrevista cujas perguntas estão disponíveis no Roteiro de Entrevista Pós-Avaliação (Apêndice F). Os objetivos principais dessa entrevista foram: (1) mapear através da experiência do usuário, as dificuldades, complexidades e limitações existentes na ferramenta; (2) saber os pontos fortes da ferramenta e sugestões de melhorias; (3) saber a opinião de usuários em relação ao esforço necessário para criação de uma regra. As informações coletadas utilizando ambos os roteiros foram usadas na análise.

### 5.1.5 Participantes

Foram selecionados cinco participantes e mais um participante que foi usado como piloto, de acordo com o perfil traçado. Porém, ao final de todos os testes, resolvemos aproveitar o piloto nas nossas análises, devido aos motivos que seguem:

1. **Não houve modificações significativas no material fornecido aos participantes.** As únicas duas modificações feitas foram sugeridas pelo próprio participante piloto. A primeira modificação foi sugerida de forma

que o cenário possuísse alguma informação mais detalhada a respeito da ferramenta sob avaliação. A outra modificação eram algumas pequenas correções no Manual de Criação de Regras do NCL-Inspector.

2. **O participante piloto possuía rigorosamente as mesmas informações que os outros participantes.** A única informação relevante que não havia sido passada ao piloto era o detalhamento da ferramenta no cenário. Porém o piloto já possuia essa informação, e inclusive, foi sugestão dele colocar essa informação no cenário, a fim de facilitar o entendimento por parte dos outros participantes.

Para manter o sigilo das informações, os participantes serão identificados através de códigos. Dessa forma, temos os seguites participantes: A0, A1, A2, A3, A4, A5. Esse código será usado para identificar cada um durante as suas respectivas descrições, que será feita nos parágrafos seguintes.

O participante A0 era Bacharel em Ciência da Computação e estava cursando Mestrado em Informática. Utilizava Java há 5 anos, XML há 4 anos e 6 meses, Eclipse há 4 anos, NCL há 1 ano e nunca havia trabalhado com XSL. Considerava seus conhecimentos acima da média em Java, XML e Eclipse, regular em NCL e novato em XSL. Em NCL nunca havia escrito um programa completo, apenas alterado os já existentes. Entretanto vinha trabalhando quase que diariamente com NCL no último ano.

O participante A1 era Mestre em Informática e cursava o Doutorado em Informática. Utilizava Java há 8 anos, XML há 4 anos, Eclipse há 6 anos, NCL há 1 ano e 2 meses e nunca havia trabalhado com XSL. Apesar de utilizar Java e XML há muito tempo, considerava seu conhecimento regular, pois não tinha tido uma continuidade no uso dessas tecnologias durante esse tempo. Considerava seu conhecimento de NCL um pouco abaixo da média, pois havia utilizado por apenas pouco tempo e nunca havia escrito um programa NCL completo.

O participante A2 era Bacharel em Ciência da Computação e cursava o Mestrado em Informática. Utilizava Java há 7 anos, XML há 6 anos, Eclipse há 6 anos e nunca havia trabalhado com XSL e NCL. Se considerava especialista em Java, XML e Eclipse. Em XSL, apesar de nunca ter usado, considerava seu conhecimento regular.

O participante A3 era Bacharel em Informática e cursava o Mestrado em Informática. Utilizava Java há 5 anos, XML há 5 anos, Eclipse há 1 anos, NCL há 2 anos e XSL há 3 anos. Se considerava um especialista em NCL e abaixo da média em relação ao Eclipse.

O participante A4 era Mestre em Informática e cursava o Doutorado em Informática. Utilizava Java há 5 anos, XML há 5 anos, Eclipse há 3 anos, NCL há 6 anos e XSL há 2 anos. Se considerava um especialista em NCL. Possuía um conhecimento regular sobre XSL, porém fazia muito tempo que não utilizava.

O participante A5 era Bacharel em Ciência da Computação e cursava o Mestrado em Informática. Utilizava Java há 4 anos, XML há 3 anos e 6 meses, Eclipse há 3 anos e 6 meses, NCL há 3 anos e nunca havia usado XSL. Se considerava um especialista em NCL e Eclipse. Considerava ter um conhecimento acima da média em Java e XML.

A tabela 5.1 apresenta um resumo do perfil de cada participante. As estrelas representam o nível de conhecimento de cada participante em relação a uma determinada tecnologia. Cinco estrelas é o valor máximo e significa que o participante se considera um especialista no assunto. Uma estrela é o valor mínimo e significa que o participante não tem conhecimento sobre a tecnologia e se considera um novato no assunto.

| Participante | Formação | Java | XML | XSL | NCL | Eclipse |
|---|---|---|---|---|---|---|
| A0 | Mestrando | ★★★★ | ★★★★ | ★ | ★★★ | ★★★★ |
| A1 | Doutorando | ★★★ | ★★★ | ★ | ★★ | ★★★★ |
| A2 | Mestrando | ★★★★★ | ★★★★★ | ★★★ | ★ | ★★★★★ |
| A3 | Mestre | ★★★★ | ★★★★ | ★★ | ★★★★★ | ★★ |
| A4 | Doutorando | ★★★ | ★★★★★ | ★★★ | ★★★★★ | ★★★ |
| A5 | Mestrando | ★★★★ | ★★★★ | ★ | ★★★★★ | ★★★★★ |

Tabela 5.1: Comparação resumida do perfil de cada participante

## 5.2 Resultados

Uma análise inter-participante foi feita com os dados coletados a partir do estudo, objetivando identificar recorrências nessas experiências. Essa análise será apresentada nesta seção, em que os dados coletados são agrupados em tópicos de interesse.

### 5.2.1 Tecnologias usadas: Java e XSL

Para implementar as regras, os participantes A0, A3 e A4 optaram por realizar a implementação utilizando XSL. Já os participantes A1, A2 e A5 optaram por fazer essa implementação utilizando Java. Ressaltando o que foi dito em seções anteriores, nenhuma imposição sobre a tecnologia de implementação foi feita. O fato ocorrido, que metade dos participantes optaram por implementar em Java e a outra metade em XSL, foi mera coincidência.

O participante A0, que optou por implementar a regra em XSL, justificou sua escolha dizendo: "Queria ver se realmente XSL era mais fácil. Achei fácil, mas tive problemas com a sintaxe." Vale a pena ressaltar que, mesmo sem ter pouco ou nenhum conhecimento de XSL, o participante completou a tarefa e ainda se demonstrou satisfeito. A satisfação do participante pode ser confirmada com mais um afirmação feita por ele: "Achei simples de implementar, não sabia nada de XSL e consegui".

O participante A1, a princípio, optou por implementar a regra em XSL. Entretanto mudou de ideia pois não sabia definir a verificação, utilizando a sintaxe. Então, resolveu implementar em Java, pois tinha mais domínio da linguagem. Mesmo assim, o participante informou que achou mais fácil expressar as regras em XSL, com exceção do caso que foi proposto a ele durante a avaliação.

O participante A2 implementou em Java, pois era a que tinha mais experiência. Demonstrou que não queria perder tempo aprendendo uma nova tecnologia, visto que Java já atenderia as necessidades. Isso é possível de ser comprovado, com a afirmação feita por ele: "Eu gosto de trabalhar orientado ao problema. Quero gastar o máximo de esforço resolvendo o problema. Não gosto de perder tempo com outras coisas."

O partcipante A3 também optou por implementar em XSL, mesmo tendo mais experiência com a linguagem Java. Ao ser perguntado sobre a sua escolha, ele respondeu: "Achei que para o problema proposto, é o (meio) mais rápido". Assim com o participante A0, teve alguns problemas com a sintaxe, mas foram solucionados com certa facilidade, através de consultas a mecanismos de busca.

O participante A4, através da leitura do manual, achou que seria mais simples desenvolver em XSL. Após a conclusão, manteve a opinião. Acrescentou dizendo que achou o mecanismo bem direto e que na maioria dos casos acredita que será mais fácil usar XSL. Fez algumas ressalvas, dizendo que acha o XPATH uma linguagem complicada, todavia, isso poderia não ser um problema, pois na maioria dos casos, era provável que fosse necessário usar apenas um subconjunto da linguagem XPATH.

Em alguns casos pontuais, como por exemplo, a verificação de um elemento `<media>` fazendo refer de outro elemento `<media>`, que por sua vez já faz refer de um terceiro, ele acredita que seria mais fácil de implementar utilizando Java. Além disso, acredita ser mais fácil validar atributos em Java, como por exemplo, verificar se os atributos `begin` e `end` de um elemento `<area>`, filho de um elemento `<media>` com o atributo `type` igual a `application/x-ginga-type`, é uma data UTC válida. Finalizou dizendo que: "Achei que a escolha de usar XPATH/XSL foi uma decisão acertada".

O participante A5 implementou em Java, pois, acredita que XSL é muito complexo para o que se propõe, o que contradiz a todos os três participantes que realizaram a implementação em XSL. Porém finalizou dizendo: "Não trocaria uma linguagem imperativa por XSL". Essa última afirmação pode indicar que o participante pode ter feito a escolha da tecnologia baseando-se mais em sua opinião pessoal do que nas complexidades do XSL.

A inclusão da tecnologia XSL para implementação das regras mostrou ser uma boa solução. Como visto, metade dos participantes escolheram livremente usar a tecnologia e mesmo tendo pouca ou nenhuma experiência, obtiveram bons resultados. Isso demonstra que as facilidades proporcionadas se sobrepõem ao fato dessa linguagem não ser tão bem conhecida, como são as linguagens imperativas, neste caso Java.

Em média, o tempo de implementação das pessoas que usaram XSL foi compatível com o tempo do participante A5, único que leu o manual dos que implementaram em Java. Isso sugere que é possível aprender XSL enquanto se está criando as regras, sem perda de produtividade.

A partir dos argumentos apresentados nos parágrafos anteriores, pode parecer que a utilização de Java para construção de regras possa ser dispensável. Porém, argumentos fornecidos pelo participante A5 reforçam a ideia de que é

necessária a disponibilização também de um ambiente imperativo para criação de regras, como já foi argumentado na Seção 2.2.

Através da discussão, podemos adotar como procedimento para criação das regras: (1) Avaliar se a regra é possui implementação trivial em XSL, se sim, criá-la em XSL; (2) caso contrário deverá ser implementada utilizando Java. Através dos estudos, podemos observar que esse procedimento vale a pena até mesmo para usuários que não conheçam XSL.

### 5.2.2 Tempo de aprendizado

Os participantes A1 e A2 demonstraram muita ansiedade em relação ao teste. Ambos não leram os dois manuais do NCL-Inspector adequadamente. Já os participantes A0, A3, A4 e A5 leram atentamente o manual e conseguiram finalizar o experimento dentro do tempo estimado. Excluindo os participantes A1 e A2, que não leram o manual, cada participante levou em média trinta minutos para ler os manuais e aprender sobre a ferramenta. Os participantes que observaram corretamente as instruções fornecidas pelo manual levaram em média uma hora para finalizar o teste, isto é, dentro do prazo previamente estipulado de duas horas.

O participante A1, que não leu as instruções presentes no manual, levou três horas e quarenta minutos para completar a tarefa, excedendo portanto, o tempo máximo estipulado. O motivo do participante A1 ter ido além do prazo limite, foi ele ter comunicado que gostaria de continuar até conseguir, mesmo tendo seu tempo excedido.

Em relação ao participante A2, ele não leu o material explicativo e tentou implementar a regra a partir do método de tentativa-erro. Obteve diversos erros os quais não conseguia identificar a origem. Após diversas intervenções do avaliador, obteve algum progresso, porém em um determinado momento desistiu de tentar cumprir a tarefa.

Os fatos apresentados indicam que o sistema possui uma tempo de aprendizado inicial baixo, pois bastam trinta minutos para que os usuários consigam realizar a tarefa no prazo.

### 5.2.3 Criação de múltiplos arquivos

Todos os participantes, com exceção do participante A0, afirmaram que não gostaram de ter que criar múltiplos arquivos para a criação de uma regra. Porém o participante A0 afirmou: "Achei chato colocar os arquivos no local correto".

O participante A1 afirmou que: "Não gostaria de ter vários arquivos para gerenciar". Certamente essa dificuldade tem relação com o fato de ter que criar um Arquivo de Declaração do Inspetor, um Arquivo de Implementação do Inspetor e um Arquivo de Mensagens para cada idioma que deseja prover internacionalização.

De fato, os Arquivos de Mensagens para Internacionalização são indispensáveis. Eles possibilitam que as mensagens de violação possam ser expressas em diversos idiomas. No caso do XML de Declaração do Inspetor, este poderia ser suprimido, como será discutido no capítulo de trabalhos futuros (Capítulo 6).

### 5.2.4 API do XML Beans

Os participantes A1, A2 e A5, que implementaram as regras utilizando Java, não conseguiram completar a atividade sem a ajuda do avaliador. Todos encontraram dificuldades na utilização da API do XML Beans. Apesar dos participantes A1 e A2 não terem lido o material instrucional, a dificuldade persistiu com o participante A5, que leu o material cuidadosamente.

Alguns fatos ocorridos com o participante A5 comprovam que a API do XML Beans gerou certa confusão. Esses fatos são:

1. Não perceber que o objeto passado como parâmetro no método `visit(XmlObject)` era um objeto de uma classe que representa um elemento da linguagem NCL, especificamente no caso do participante A5, o objeto `MediaType`;

2. A exclamação feita por esse participante: "Ah! Eu posso recuperar as `<area>` de um elemento `<media>` através desse método aqui (`MediaType.getAreaArray()`)". Isso demonstra que ele não entendeu, a priori, que um elemento `MediaType` era uma representação orientada a

objetos de um elemento `<media>`. Sendo assim, esse método retornaria a representação orientada a objetos dos elementos `<area>` contidos dentro do elemento `<media>` em questão.

Ainda sobre o participante A5, ele mencionou: "Achei a API do XML Beans meio esquisita". Porém ao ser indagado em relação ao motivo desta afirmação, ele respondeu: "Achei isso pois não conhecia direito". Isso reforça que é necessário uma melhoria nos manuais do sistema, em relação a como o NCL-Inspector faz uso da API do XML Beans.

### 5.2.5 Arquivo de Declaração do Inspetor

Os participantes A1, A3 e A4 mencionaram que, se pudessem, gostariam de eliminar a necessidade de ter que escrever o Arquivo de Declaração do Inspetor. O participante A5 também em um determinado ponto da entrevista chegou a mencionar o mesmo, porém voltou atrás. Ele achou que talvez no futuro, esse arquivo poderia ser útil para aumentar a flexibilidade do sistema.

### 5.2.6 Integração com o Ambiente de Desenvolvimento

Todos os participantes foram unânimes sobre a necessidade de haver um assistente (*Wizard*) para criação dos arquivos. O participante A0 e A1 foram mais detalhistas, e complementaram listando os seus desejos mais claramente. Eles gostariam que a ferramenta tivesse uma integração com o Eclipse, e que fornecesse templates para os arquivos XSL, Java, Arquivo de Declaração do Inspetor e a classe de teste. Além disso, esse assistente poderia fornecer arquétipos de projetos com os arquivos já posicionados no local certo, baseado no identificador da regra de inspeção.

### 5.2.7 Melhorias na notificação de erros em XSL

O participante A4 sugeriu que se a linha e coluna do XML que reporta erro, em uma regra XSL, fossem omitidas, o sistema poderia considerar como padrão a linha e coluna do elemento que foi capturado, através do `<template match="">`. Essa sugestão foi baseada na crença que, na maioria das vezes, o desenvolvedor terá a intenção de retornar a linha e coluna de

erro do elemento capturado pelo `<template match="">`. Isso economizaria um certo esforço na escrita da regra, por parte do usuário.

### 5.2.8 Linguagem de Domínio Específico

Uma linguagem de domínio específico foi sugerida pelo participante A4. Isso facilitaria ou tornaria possível a implementação de regras por usuários não-programadores. O participante fez duas sugestões sobre a forma como essas linguagens poderiam ser implementadas. A primeira poderia ser através de uma linguagem visual, onde o usuário não-programador iria compondo elementos gráficos até conseguir obter a regra de inspeção que deseja. A outra forma mencionada foi de realizar a inserção das regras através de programação por exemplo.

Entretanto, é preciso avaliar se vale a pena um usuário não-programador ter acesso a esse tipo de ferramental. Provavelmente, esse perfil de usuário não produziria um programa utilizando diretamente NCL. Em vez disso, usaria provavelmente um ambiente de autoria gráfico ou templates (dos Santos et al. 2009, Soares et al. 2009). Neste caso, um bom requisito seria estender a ferramenta para a avaliação dos templates, pois o criador dos templates possui um maior conhecimento técnico. Entretanto, criar uma ferramenta de inspeção de templates não faz parte do escopo desse trabalho e poderia ser uma sugestão de um trabalho futuro.

### 5.2.9 Esforço de implementação das regras

A terceira pergunta da entrevista pedia para o participante fazer uma comparação entre o esforço de implementar uma regra de inspeção, com o de fazer a inspeção manualmente, utilizando por exemplo, uma lista de verificação (*checklist*).

O participante A0 achou melhor implementar a regra de inspeção. Ele mencionou acha difícil lembrar de todas as regras. Também mencionou que fazer avaliação de aplicações grandes pode demandar muito esforço. Como exemplo de aplicação grande, ele mencionou a aplicação de Comercial da Proview disponível no Clube NCL (`http://clube.ncl.org.br/`). Na sua opinião, a implementação da regra levou pouco tempo.

O participante A1 disse que, se a complexidade do código NCL for grande, a verificação automática se torna mais relevante, mesmo para um número pequeno de mídias. Ele mencionou que um documento contendo um número a partir de cinco mídias, somado à esperança de um dia poder reutilizar aquela regra, já acreditaria que valeria mais a pena escrevê-la.

O participante A2 acha que sempre vale a pena implementar a regra. A justificativa é que independentemente do tamanho do programa, pode-se reaproveitar a regra em diversos outros, mesmo que todos esses sejam pequenos.

O participante A3 acredita que, em regras simples, é muito mais vantajoso implementar usando o NCL-Inspector.

Já o participante A4 demonstrou bastante entusiasmo quando essa pergunta foi feita, ele exclamou: "Ah, não! É muito mais simples aqui! (se referindo ao NCL-Inspector)." Também mencionou que as regras poderiam ser reusadas.

O entusiasmo também pode ser notado no participante A5, através da afirmação que foi feita ao também ser perguntado: "Não dá nem pra comparar!" – indicando que é muito melhor realizar a inspeção de forma automática, através do NCL-Inspector. Acrescentou ainda, afirmando que "a validação automática é indispensável".

Os participantes A2, A4 e A5 foram completamente a favor da validação automática. Já os participantes A0 e A1 estabeleceram critérios mínimos onde acham vantajoso a inspeção automática. Porém, os dois estabeleceram critérios mínimos tão baixos, que se encaixam virtualmente em todas as aplicações NCL existentes. Considerando esse fato, todos os participantes com exceção do A3, concordaram que utilizar uma inspeção automática é sempre mais vantajosa.