**1. Requisitos**

A seção de requisitos descreve os problemas que estão sendo resolvidos e também as soluções. Esta seção da documentação é de interesse dos usuários e dos implementadores. Deve conter detalhes sobre estratégias específicas de implementação. As outras partes da documentação do compilador não serão de interesse dos usuários; somente dos implementadores, mantenedores e relacionados.

1. **Visão geral (1 pág)**Uma explicação do propósito do sistema (compilador) e das funcionalidades que ele fornece.
2. **Especificação revisada**A especificação detalhada original do compilador pode estar com algumas partes sub-especificadas ou pouco claras. Nesta seção deve-se tornar claro quaisquer decisões e interpretações feitas sobre o significado dos requisitos, assim como extensões e modificações que foram introduzidas nos requisitos pela equipe.
3. **Manual do usuário (1 - 5 págs)**Uma descrição detalhada de como o usuário pode usar o compilador, quais operações o usuário pode realizar, procedimentos de instalação, opções de linha de comando, etc. Especificações detalhadas de formatos devem estar em apêndices, mas citadas onde necessário. Todas as definições de ambiente devem estar explicitadas aqui. Por exemplo, sobre quais plataformas o programa deve rodar, se há necessidade de uma hierarquia de diretório específica ou de algum tipo de aplicativo ou suporte de tempo real (JVM, por exemplo), etc. Juntamente com a visão geral, este manual deve fornecer todas as informações necessárias para o usuário do compilador.
4. **Desempenho (meia página)**Quais recursos o compilador necessita para funcionar normalmente e quanto espaço e tempo espera-se que ele consuma.
5. **Análise do problema (2 - 10 págs)**Uma descrição clara do problema. Isto inclui o modelo conceitual por trás do projeto (e possivelmente a interface com o usuário). A análise do problema geralmente inclui a discussão de questões complexas para implementar. A modelagem deve incluir tanto diagramas quanto descrições textuais essenciais, o que permitirá uma compreensão completa do problema. Esta parte também descreve as opções avaliadas e rejeitadas, com as devidas razões, questões ainda em aberto que deverão ser resolvidas posteriormente.

Também pode-se apresentar *casos de uso*, que são muito úteis para escrever a especificação revisada e/ou o manual do usuário. Um caso de uso é um objetivo específico e uma lista de ações que um usuário realiza para atingir aquele objetivo. Se a coleção de casos de uso cobrir todos os objetivos desejados pelo usuário, o cliente (neste caso, o professor) terá mais confiança no sistema (neste caso, compilador) e que ele atende os objetivos.

**2. Projeto**

A seção de projeto apresenta uma figura de alto nível da estratégia de implementação.

1. **Visão geral (1 - 3 págs)**Uma visão geral do projeto: organização top-down, questões de projeto particularmente interessantes, uso d e bibliotecas e outros módulos de terceiros e indicações para aspectos que não são definitivos ou que poderão vir a ser alterados. Também inclui problemas de projeto: decisões que podem ter sido equivocadas e compromissos entre desempenho e flexibilidade que podem afetar a avaliação geral.
2. **Estrutura de tempo de execução (1 - 5 págs)**Uma descrição da estrutura do programa, na forma de um modelo de objetos de código. Este modelo deve ocultar as representações de tipos abstratos de dados. O objetivo aqui é mostrar o relacionamento entre os objetos. A modelagem deve incluir tanto diagramas quanto descrições textuais essenciais, o que permitirá uma compreensão completa do problema. As representações de tipos de dados deve ser explicada se forem incomuns, particularmente complexas ou fundamentais para o projeto completo (cabe a vocês decidir isto - e esta decisão implicará em saber se vocês compreendem a estrutura do compilador ou não!).
3. **Estrutura dos módulos (1 - 5 págs)**Uma descrição da estrutura sintática do programa, expressado como um diagrama de dependência dos módulos. Por exemplo, em Java, deve-se incluir as estruturas de pacotes, as interfaces e as classes. Explique por que esta estrutura específica foi escolhida (por exemplo, por que determinadas interfaces foram definidas) e quais/como padrões de projetos (design patterns) foram usados.

Comente se as decisões de estrutura contribuiram para a simplicidade, extensibilidade (facilidade de incluir novos recursos), distribuição de tarefas (diferentes membros da equipe podem trabalhar em diferentes partes do sistema/compilador sem necessidade de comunicação constante) e outros objetivos louváveis estampados em qualquer manual de engenharia de software.

**3. Testes**

A seção de testes da documentação apresenta a abordagem tomada para verificar e validar o compilador. Em um sistema real, deve-se incluir testes com usuários para definir a adequação do sistema como solução ao problema descrito na seção de requisitos, além da execução de testes , para verificar a correção dos algoritmos. Mas a primeira parte será a apresentação do compilador ao professor; por isso, não é necessário documentar esta parte. Da mesma forma como não é adequado simplesmente listar o código como documentação do sistema, também não é adequado simplesmente listar os testes realizados. Por isso, discuta como os testes foram selecionados, porque eles são suficientes, porque um leitor deveria acreditar que nenhum teste importante foi omitido.

1. **Estratégia (1 - 2 págs)**Uma explicação da estratégia geral de teste: Black box, top-down e/ou bottom-up, os dados utilizados nos testes (no nosso caso, os programas em ZooA), ferramentas de teste (como jUnit), métricas utilizadas, verificações em tempo de execução, etc. Pode-se utilizar diferentes técnicas (ou combinações de técnicas) em diferentes partes do programa. Em cada caso, justifique suas decisões.

Explique quais tipos de erros esperava-se encontrar e quais não. Debata quais aspectos do projeto foram difíceis e fáceis de validar.

1. **Resultados dos testes (1 - 2 págs)**Resumo dos testes realizados: quais módulos foram testados e em que profundidade? Indique o grau de confiança no código: Quais tipos de falhas foram eliminadas? Quais ainda permanecem?

**4. Reflexão**

A seção de reflexão da documentação é onde há liberdade para expressar avaliações subjetivas a respeito do projeto como um todo, como por exemplo aprendizados, caminhos a não mais tomar ao desenvolver software, etc. O que mais surpreendeu a equipe? O que a equipe gostaria de saber antes de ter começado? Como poderiam ter sido evitados os problemas que foram encontrados durante o desenvolvimento?

1. **Avaliação (1 pág)**O que se considera sucessos e falhas do desenvolvimento: problemas de projetos não resolvidos, desempenho, etc. Identificar quais recursos do projeto são os mais importantes. Destacar técnicas de projeto ou de implementação que os deixaram mais orgulhosos. Discuta quais erros foram feitos em tempo de projeto e os problemas que eles ocasionaram na implementação ou no resultado final.
2. **Lições (1 pág)**Quais lições foram aprendidas desta experiência: o que fariam diferente em uma segunda oportunidade e sugestões sobre como as falhas de projeto e implementação poderiam ser resolvidas. Descrever os fatores que causaram problemas, tais como prazos perdidos, bugs e limitações conhecidas.
3. **Bugs e limitações conhecidas**Onde a implementação não adere à especificação? Sejam precisos. Embora bugs e recursos ausentes gerem perda de pontos na avaliação, os bugs identificados adequadamente não serão considerados como tal.

**5. Apêndices**

Os apêndices contêm detalhes de baixo nível sobre o compilador que não são necessários para compreendê-lo, mas sim para fazer a manutenção ou alterações.

1. **Formatos.**Uma descrição dos formatos assumidos ou garantidos pelo programa: para leitura/escrita de arquivos, opções de linha de comando, diálogos de usuário, etc. Pode-se dividir estes formatos em "visíveis aos usuários", que conceitualmente são parte dos requisitos de interface com o usuário e em "formatos internos", que são parte de componentes.
2. **Documentação do código.**Deve ser extraído do código e apresentado separadamente. Se a documentação do código for feito seguindo um estilo próprio (por exemplo, o estilo JavaDoc), a documentação do código será aquela que for extraída automaticamente do código. A especificação de um tipo abstrato de dados deve incluir sua visão geral, definição dos campos/atributos e operações elementares.
3. **Casos de teste.** Não é necessário incluir casos de testes muito grandes. Indique a função de cada teste: teste de stress, teste de entrada de dados, entrada com dados positivos, negativos e zero, detecção de erro léxico, erro sintático, etc.