1. Motivation
   1. Hoje em dia, a maior parte das empresas de software desenvolvem código a um ritmo alucinante, dada a necessidade de rápida adaptação ao ambiente onde atuam. O que exige que haja sistemas que se certifiquem que todas as alterações efetuadas não obstruem as funcionalidades existentes no código, através da **aplicação de testes. (Integração Contínua)**
   2. Estes sistemas têm a capacidade de recordar todas as versões que alguma vez existiram. estando assentes numa infraestrutura fundamental que é o **repositório**.
   3. Num repositório, podem existir vários “caminhos” de implementação, que se chamam **ramos**. Por definição ao ramo principal chamamos “**master branch**”, que corresponde à linha principal de desenvolvimento.
   4. Por exemplo, numa empresa como a Google, são feitas mais de 5000 alterações ao código, diariamente, num repositório com milhões de linhas de código.
   5. **Problema**: Devido à complexidade das relações e interdependências que existem entre ficheiros, a **probabilidade** de uma dessas alterações originar uma falha ou um conflito é **significativa**, implicando uma **regressão.**
   6. Detectá-las e corrigi-las, manualmente, é uma tarefa árdua e propensa a erros, por isso existe uma procura grande em desenvolver técnicas automáticas que possam ajudar a manter a **mainline verde** (ou seja, sem falhas), minimizando intervenção humana.
   7. Pelo contrário, uma **mainline vermelha** é problemática no sentido em que, se uma falha não for detetada o mais brevemente possível, corre-se o risco de ocorrer um “efeito bola de neve” e eventualmente ter de reverter o código até ao último ponto estável, resultando numa **quebra de produtividade**.
   8. Por outro lado, à medida que a dimensão dos projetos e das equipas aumenta, testar **todas** as alterações rapidamente faz com que o **custo** de usar estas metodologias se torne **inibitivo**.
   9. **Solução**: Encontrar uma estratégia **escalável**, em termos de tempo e de recursos computacionais, para manter, com confiança, um repositório de código eficiente e fiável.
   10. Então, para aumentar a confiança na correção de um sistema de software são explorados dois aspetos: a maneira como se gere a aplicação de testes e a maneira como se configuram sistemas de integração contínua.
2. Continuous Integration
3. Regression Testing
   1. Minimisation
   2. Selection
   3. Prioritisation
4. System Configuration Solutions
   1. Naive
   2. Cumulative
   3. Change-List
   4. Always Green Master
5. Comparative Table and End. (o que queremos é fazer melhor do que a estratégia cumulativa, com a informação disponível)