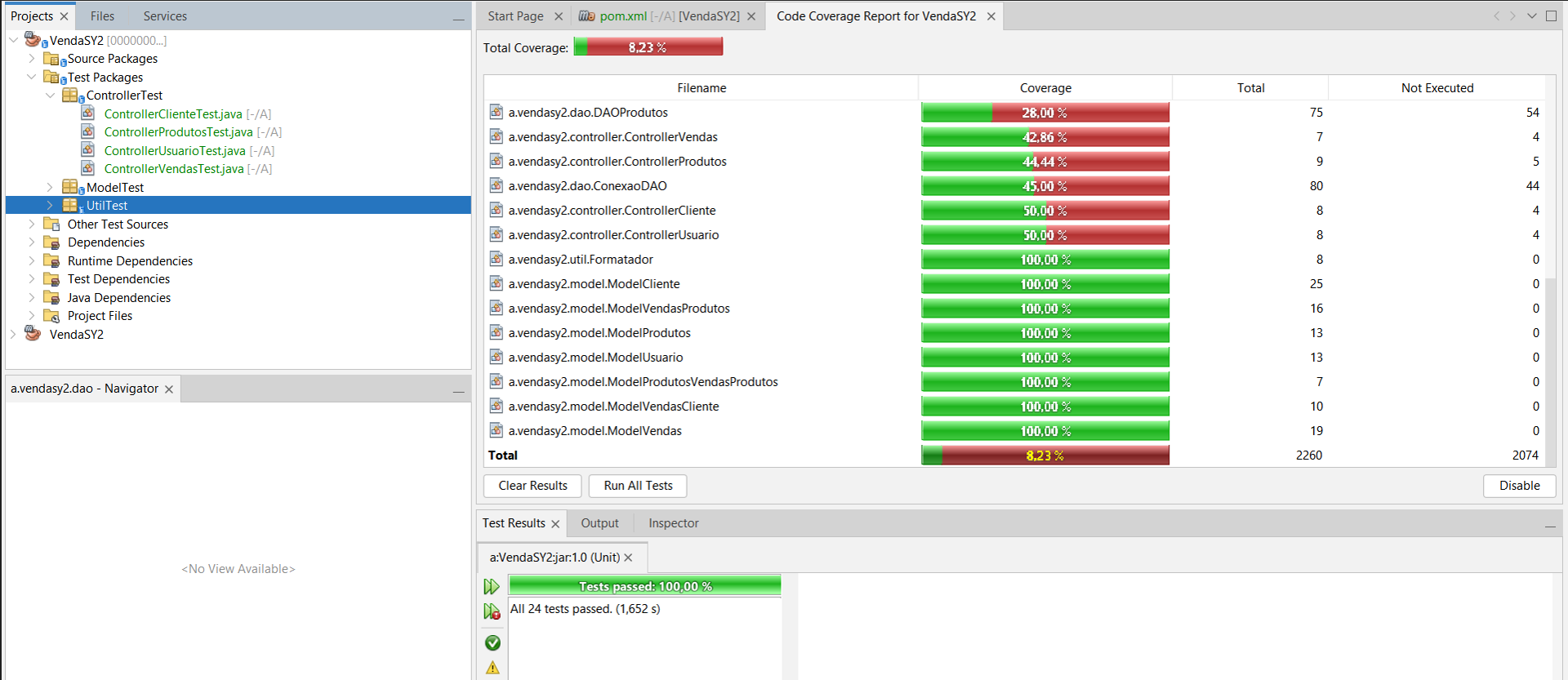
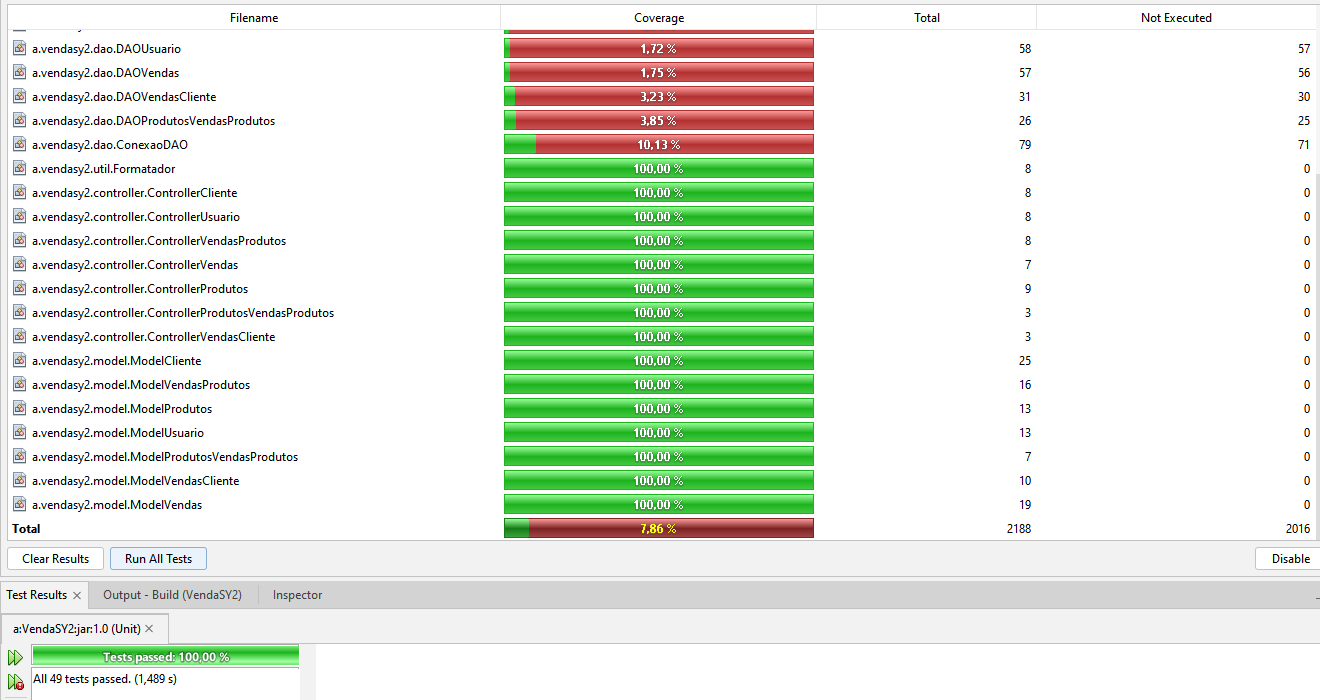
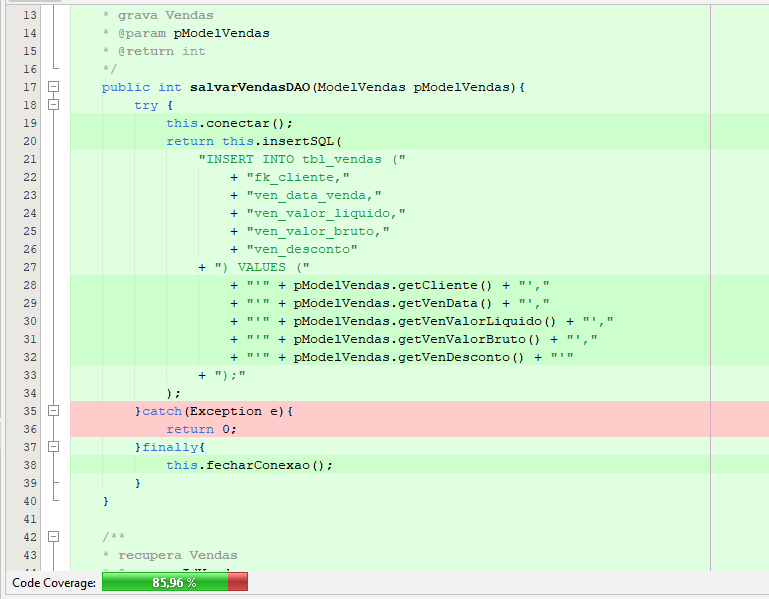
**Entrega 4**

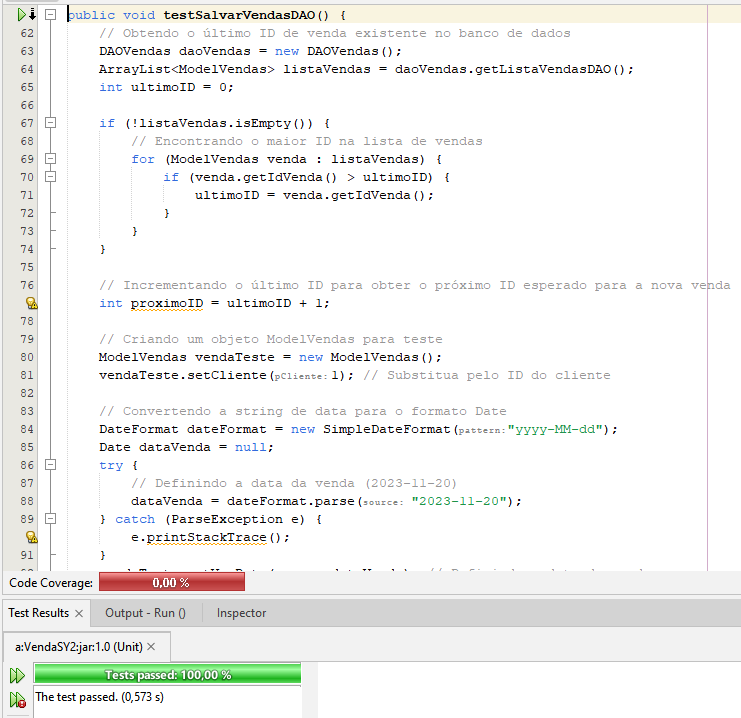
Na fase anterior, concluímos um total de 24 testes, os quais proporcionaram uma cobertura de código de 8,23%. Estes testes foram executados em todas as classes model, bem como em alguns métodos das classes controller e util.  
  
  
  
  
Para atender à cobertura de código exigida, identificamos a necessidade de realizar mais testes. Inicialmente, concentramo-nos na conclusão dos testes de todos os métodos das classes controller. Nesse estágio, realizamos um total de 49 testes, obtendo 100% de cobertura nas classes model e view. No entanto, nosso percentual global diminuiu para 7,86%. Prosseguimos, então, com os testes adicionais:  
  




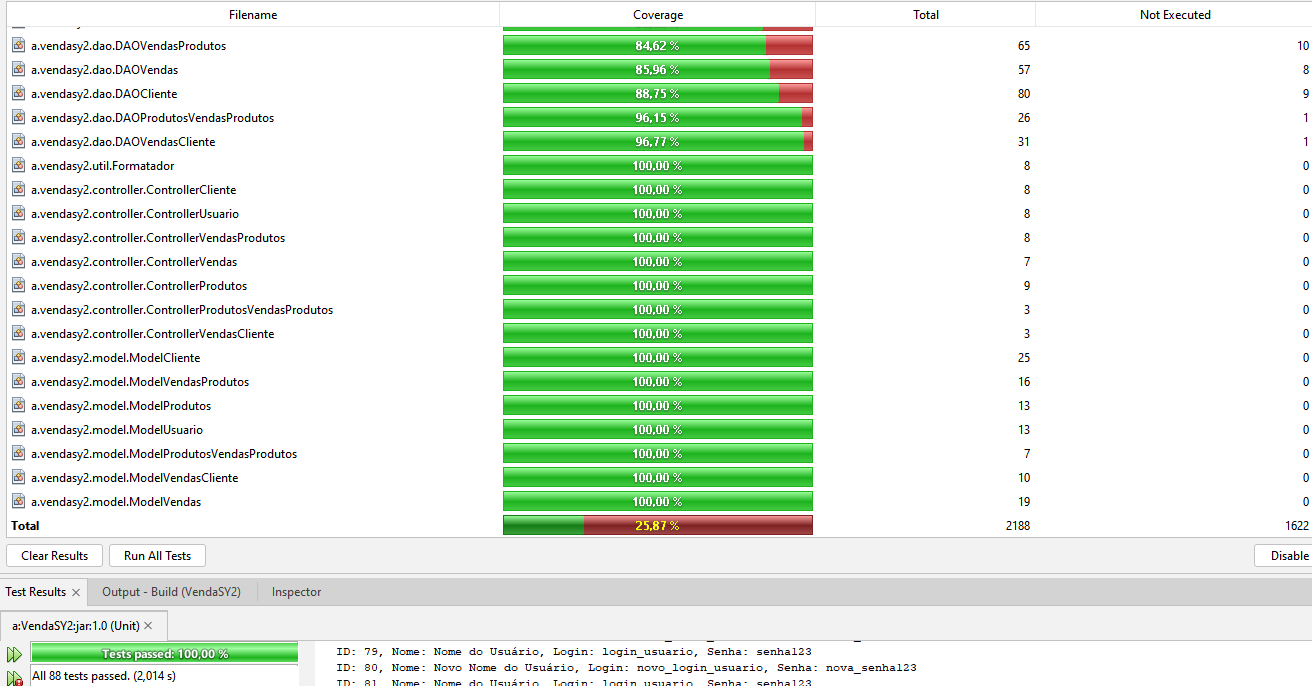
Após a conclusão das classes Model e Controller, avançamos para os testes nas classes DAO, uma tarefa mais desafiadora, dada a interação direta com o banco de dados. Apesar das complexidades encontradas, conseguimos testar todos os métodos das classes DAO, totalizando 81 testes e alcançando 24,59% de cobertura.

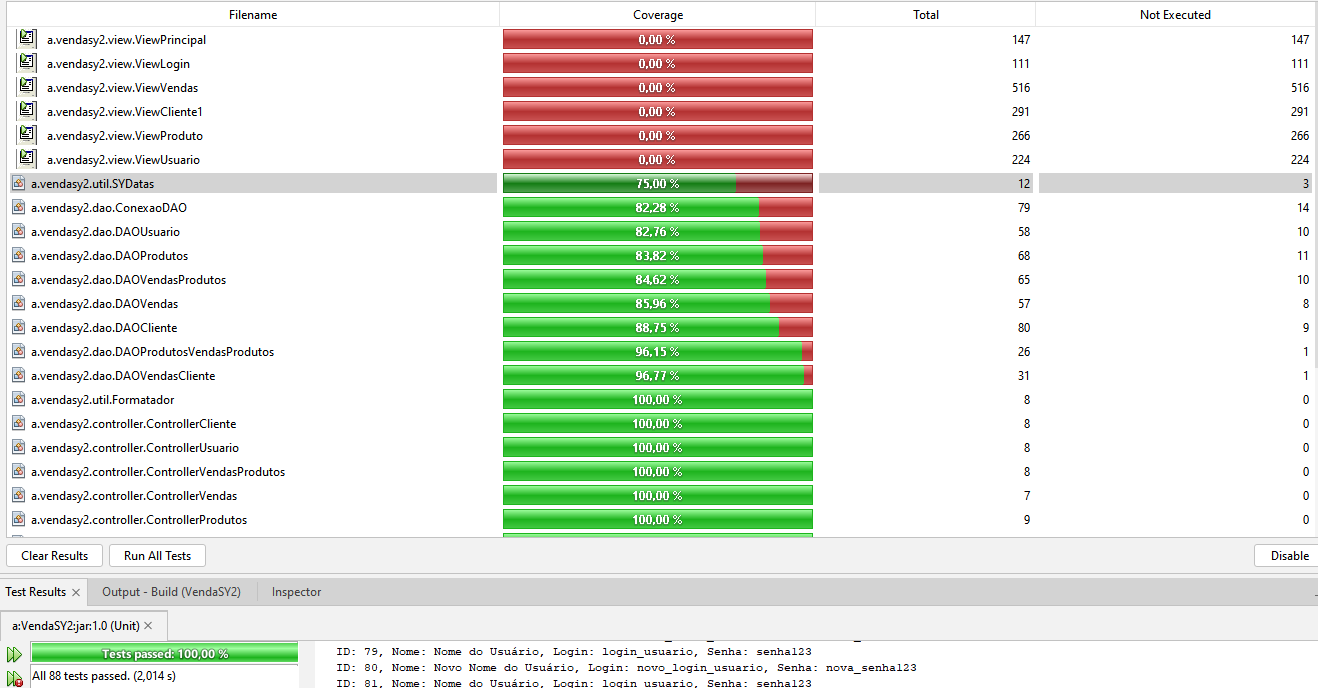
Embora todos os métodos tenham passado com êxito (100%), não alcançamos 100% nas classes devido a algumas exceções nos métodos:



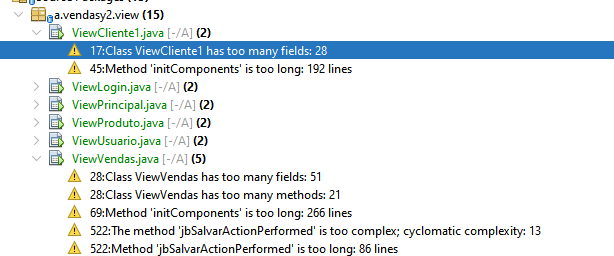


Para finalizar e cumprir a meta estabelecida, realizamos testes na classe util restante e em alguns métodos da classe de conexão. Ao final, acumulamos 88 testes, atingindo uma cobertura de 25,87% do código.



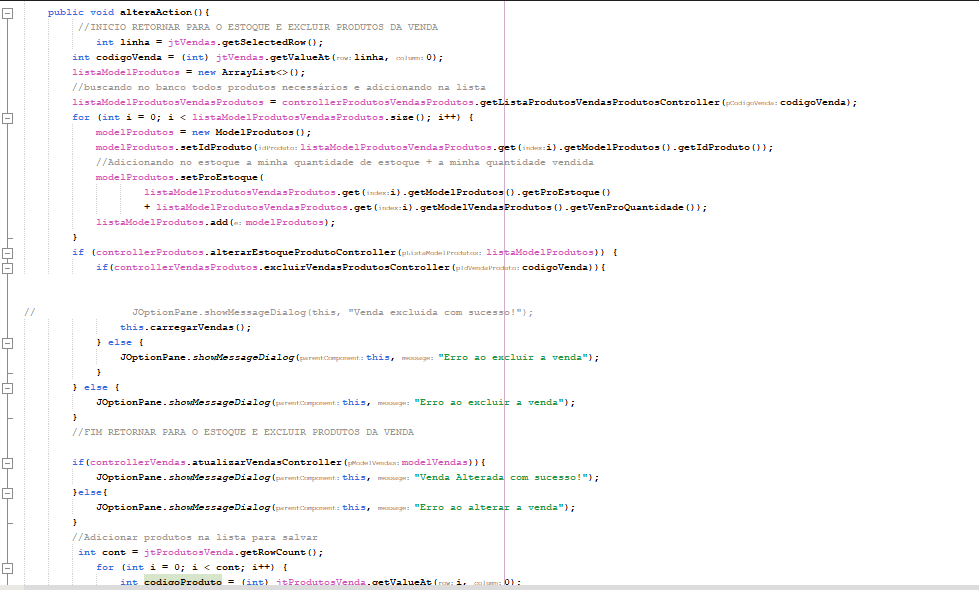


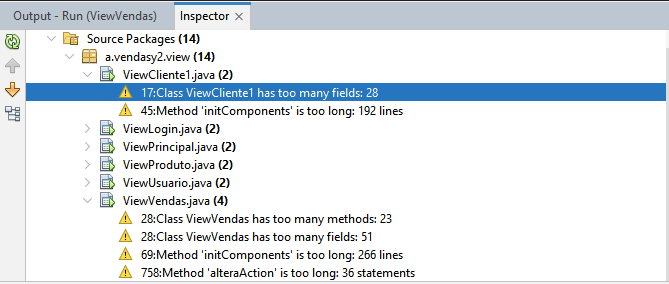
Reconhecemos que, devido ao tamanho das classes view, a porcentagem de cobertura foi menor, apesar dos inúmeros testes realizados. Não alcançamos a meta nesta sprint, mas estamos empenhados em encontrar estratégias para atingi-la até a entrega final do produto.  
  
**Plugin do code metrics**

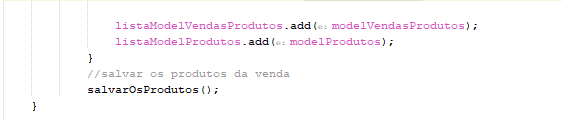
Utilizamos o plugin de métricas de código para inspecionar o código antes da entrega da sprint, resultando no seguinte diagnóstico:  
  


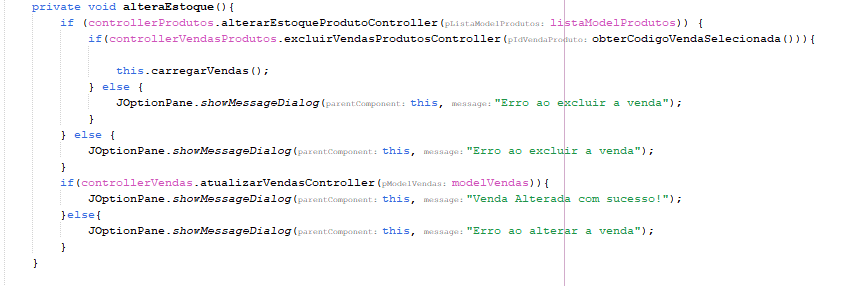
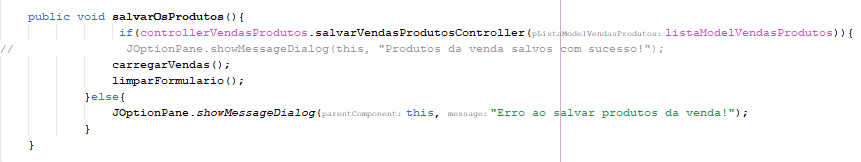
Identificamos problemas de extensão em todas as classes view. No entanto, observamos que os métodos "initComponents" são gerados automaticamente e não podem ser alterados, impossibilitando uma solução imediata. Na classe ViewVendas, o método "jbSalvarActionPerformed" que desenvolvemos foi identificado como longo e complexo. Para resolver essa questão, optamos por refatorá-lo e simplificá-lo. Criamos os métodos salvarAction e alteraAction para invocá-los no método a ser otimizado, reduzindo seu tamanho:





Após esta intervenção, reinspecionamos o código e resolvemos o problema inicialmente identificado no "jbSalvarActionPerformed". Contudo, nos deparamos com um dos métodos criados para mitigar o problema anterior, que também estava extenso.  
  
  
  
  
alteraAction()

Como solução, decidimos dividi-lo também, criando os métodos salvarOsProdutos e alteraEstoque.  
  
  


Com estas ações, resolvemos o problema de extensão identificado durante a inspeção do plugin de métricas de código.