Universidade Tecnológica Federal do Paraná
Campus Campo Mourão

Disciplina: Engenharia de Software 02
Professor: Prof. Dr. Marco Aurélio Graciotto
Título: Relatório Atividade 07
Aluno: Renan Kodama Rodrigues 1602098

Relatório:

- **Tempos planejados e efetivos (considerando todos os projetos).**

Programa	Tempos Planejados	Tempos Efetivos
01	2:00	1:03
02	9:30	3:30
03	0:43	1:49
04	1:42	1:47
05	1:59	1:28
06	1:07	0:54
07	1:37	1:17

- **Tamanhos planejados e efetivos (considerando todos os projetos).**

Programa	Tamanho Planejados	Tamanho Efetivos
01	-	125
02	600	105
03	125	250
04	287	167
05	250	128
06	162	140
07	277	350

- **Análise da evolução da diferença entre os tempos planejados e efetivos. Justifique essa diferença, como erros encontrados no processo de planejamento poderiam ser corrigidos e evitados e como o método de estimativa de tempo poderia ser melhorado.**

R: Como observado, grande parte dos tempos efetivos acabaram se ajustando dentro do limite de tempo planejado, isso graças aos dados que foram capturados no decorrer das atividades, melhorando assim o histórico de dados e dando maior precisão para a estimativa, assim como também o desenvolvedor.

Erros no processo de planejamento poderiam ser evitados através da análises dos projetos anteriores, para encontrar soluções já criadas que atendam ao problema atual, além de compreender corretamente os requisitos solicitados, em muitas das vezes, o melhor jeito de entender/coletar os requisitos seria trazer o cliente para o processo de planejamento. O método de estimativa de tempo poderia ser melhorado se pudesse contar com a experiência do programador, se ele é de um nível mais baixo, intermediário ou avançado, consequentemente impactando no tempo de desenvolvimento.

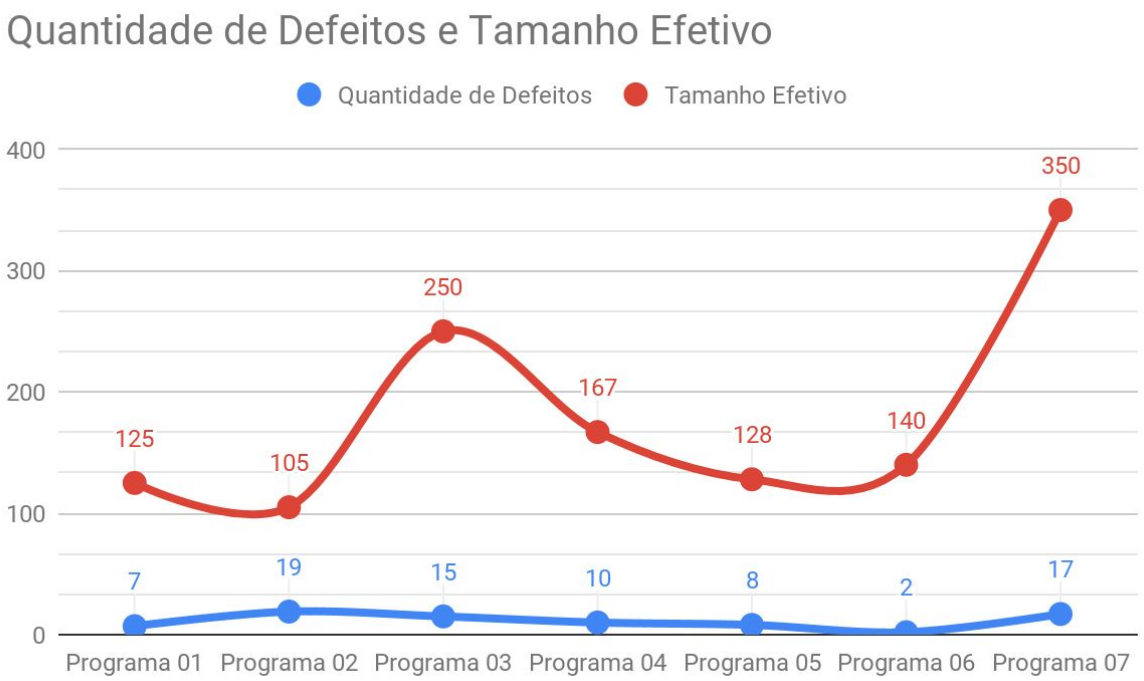
- **Análise da evolução da diferença entre os tamanhos planejados e efetivos. Justifique essa diferença, como erros encontrados no processo de planejamento poderiam ser corrigidos e evitados e como o método de estimativa de tamanho poderia ser melhorado.**

R: Com o decorrer dos projetos os valores para tamanho estimado e efetivo, acabaram se tornando mais próximos, isso, devido ao conhecimento e experiências adquiridas ao longo dos projetos desenvolvidos, que dão uma melhor noção para uma estimativa. O método de estimativa de tamanho poderia ser melhorado, se ele analisa-se também o tipo de padrão de código e a linguagem que o programador estaria utilizando que interfere diretamente na quantidade de linhas em um código.

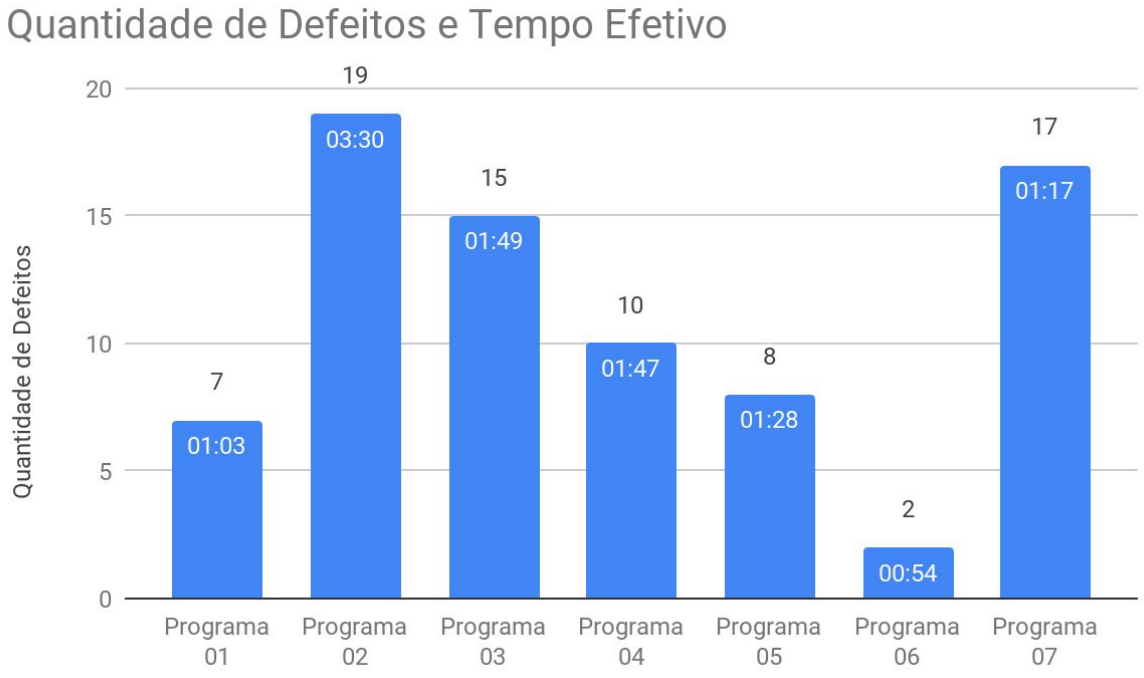
- **Quantidade de defeitos encontrados por projeto.**

Programa	Quantidade de Defeitos
01	7
02	19
03	15
04	10
05	8
06	2
07	17

- **Relação de defeitos encontrados e tamanho efetivo (densidade de defeitos) do programa (para cada projeto).**



- **Relação de defeitos encontrados e tempo efetivo para desenvolvimento do programa (para cada projeto)**



- **Quantidade de defeitos para cada tipo de defeito.**

Tipo de Defeito	Quantidade de Defeitos
Documentation	17
Syntax	19
Build Package	1
Assignment	2
Interface	19
Checking	0
Data	9
Function	6
System	0
Environment	5

- **Tempo médio para correção de cada tipo defeito.**

Tipo de Defeito	Tempo Médio para as Correções
Documentation	1.32352941176
Syntax	1.05263157894
Build Package	0.6
Assignment	0.35
Interface	0.68421052631
Checking	0
Data	1.16667
Function	1.51668
System	0
Environment	2.04

- **Quantidade de defeitos ocorridos (inseridos) por fase (considerando todos os projetos).**

Fases de Desenvolvimento	Quantidade de Erros
Planning	3
Design	5
Code	70
Compile	0
Test	0

- **Quantidade de defeitos encontrados/corrigidos por fase (considerando todos os projetos).**

Fases de Desenvolvimento	Quantidade de Erros/Corrigidos
Planning	0
Design	0
Code	52
Code Review	6
Compile	9
Test	11

- **Análise sobre os tipos de defeito e o esforço para corrigi-los.**

R: Como observado, através dos dados obtidos, grande parte dos defeitos encontrados foram de syntax, interface, documentação. Na qual, para os erros de syntax, a maioria ocorreu devido à falta de atenção e pouca experiência com a linguagem python. Para os erros de documentação, estes ocorram devido ao formato adotado para a padronização do código de acordo com o modelo PEP8. Os erros de Interface, podem ser explicado devido aos erros no formato dos parâmetros, valores incorretos passados como argumento e chamadas de funções incorretas. O esforço para corrigir os problemas se deu ao analisar o código, realizar comentários referentes a documentação das funções informando os parâmetros de entrada, procurar entender sobre a sintaxe da linguagem e o padrão de código, encontrar trechos que poderiam ser otimizados e por fim, executar e analisar os resultados obtidos com os casos testes.

- **Análise sobre os momentos em que os defeitos ocorreram, quando foram encontrados e como as atividade de garantia de qualidade em prática contribuíram para isso,**

justificando os dados, como os defeitos poderiam ser detectados antes da fase em que foram encontrados (se possível) e como os defeitos poderiam ser evitados.

R: Grande parte dos erros foram encontrados na fase de código. Em maior parte, os erros só foram encontrados após a compilação ou quando a API reconhecia automaticamente os problemas no código. Mas, para melhorar a precisão e validar os valores obtidos através dos cálculos matemáticos, foram utilizados casos de testes para testar as funções do código, inserindo diferentes valores de entrada, mas com o valores de saída já determinados. Por meio destes testes, foi possível analisar se o comportamento das funções e os resultados estavam corretos. No geral, grande parte dos defeitos encontrados, poderiam ser evitados se houvesse maior domínio na documentação da linguagem utilizada, ter realizado um planejamento mais preciso, flexível e modular. Tivesse tido mais cautela no planejamento do projeto tentando compreender melhor os requisitos. Ter dado maior atenção à funções que já haviam sido desenvolvidas tornando-as reutilizáveis. E por último, ter criado mais casos de testes dando uma maior cobertura ao programa, não se limitando apenas aos casos mencionados nas especificações.

- **Em relação às fases e atividades específicas para melhoria de qualidade, informe o tempo gasto e a quantidade de defeitos encontrados em cada uma delas.**

Programa	Fases de Desenvolvimento	Tempo Gasto	Quantidade de Defeitos
01	Design Review	-	-
	Code Review	-	-
	Test	0.09	2
02	Design Review	-	-
	Code Review	-	-
	Test	0.56	4
03	Design Review	-	-
	Code Review	-	-
	Test	0.11	2
04	Design Review	-	-
	Code Review	-	-
	Test	0.13	1

Programa	Fases de Desenvolvimento	Tempo Gasto	Quantidade de Defeitos
05	Design Review	0.01	0
	Code Review	0.10	2
	Test	0.14	1
06	Design Review	0.01	0
	Code Review	0.02	0
	Test	0.17	0
07	Design Review	0.01	0
	Code Review	0.04	4
	Test	0.21	1

- **Tamanhos reutilizados e Tamanhos reutilizáveis.**

Programa	Tamanhos Reutilizados	Tamanhos Reutilizáveis
01	0	0
02	0	0
03	3	0
04	38	18
05	46	23
06	78	0
07	141	10

- **Análise sobre o impacto de reutilização e de criar código reutilizável, considerando um projeto corrente e o cenário de vários projetos na perspectiva do custo de engenharia e na qualidade do produto.**

R: Quando a boa prática de reutilização de código é bem aplicada, os custos de desenvolvimento acabam se tornando menores, como por exemplo, o tempo. Na qual, evita o desenvolvedor a criar novas funções que uma vez já foram criadas em outros projetos e até mesmo de testá-las. Como as funções reaproveitadas já foram utilizadas em alguma ocasião, é esperado que elas tenham passado por testes, garantindo assim a confiabilidade do código, em

outros casos, até otimizações já foram feitas. Uma vez que o código a ser reutilizado esteja conciso e modular, os resultados irão afetar positivamente na qualidade do produto, assim como, na redução do tempo de desenvolvimento gerando menos custos financeiros para a empresa ou pesquisa.