Relatório da utilização do Personal Software Process - PSP

Clodoaldo A. Basaglia da Fonseca

Maio 2019

1 Personal Software Process - PSP

Desenvolvido em 1993 por Watts S. Humprey, o PSP é uma aproximação estruturada e disciplinada no desenvolvimento de um software. Ao utilizar os conceitos e métodos no trabalho, os envolvidos no projeto conseguem melhorar suas estimativas e planejamentos, indpendendo da área que trabalhem, além de conseguirem melhorar sua produção e a qualidade do seu trabalho, resultando na diminuição dos defeitos em seus produtos [1]

A eficácia da metodologia do PSP tanto na area academica quanto na industria é documentada em inúmeros relatórios técnicos, já que o PSP depende da analise de informações pessoais para ajudar nas predições, os relatórios tem como base dados confiaveis.

2 Analise dos dados captados durante o desenvolvimento

2.1 Tamanho e tempo

A tabela 1 representa compila os dados pertinentes ao tempo utilizado efetivamente no desenvolvimento do programa, bem como o tempo planejado para o desenvolvimento:

A diferença entre os tempos planejados e efetivos pode ser justificada com a não famililiaridade do programador com a linguagem Python, mostrando a tamanha inteferencia na produtividade que isso pode acarretar, contudo, com

	Tempo	
Programa	**Planejado**	**Efetivo**
Programa 1	1:00	0:23
Programa 2	3:00	1:26
Programa 3	2:05	1:02
Programa 4	1:00	0:48
Programa 5	1:52	0:58
Programa 6	0:30	0:20
Programa 7	0:42	0:42
Total:	10:09	05:39

Tabela 1: Tabela de tempo planejado e efetivo dos 7 programas em horas

o passar dos projetos, o conhecimento da linguaguem aumenta, bem como a produção. Isso deveria entrar nos parametros do método de planejamento pois influencia diretamente durante a produção de um produto, tanto na qualidade do código, bem como na velocidade de produçao.

A tabela 2 representa compila os dados pertinentes ao tempo utilizado efetivamente no desenvolvimento do programa, bem como o tempo planejado para o desenvolvimento: As diferenças dos tamanhos planejados e efetivos

	Tamanho	
Programa	**Planejado**	**Efetivo**
Programa 1	-	57
Programa 2	55	86
Programa 3	81	121
Programa 4	54,6	70
Programa 5	91,8	116
Programa 6	92,3	87
Programa 7	335	362
Total:	709,7	899

Tabela 2: Tabela de tamanhos planejados e efetivos dos 7 programas

não é tão grande no começo, isso ajuda ao método calcular de forma mais correta os proximos planejamentos. A capacidade do programador ser mais sucinto no código pode influenciar no seu tamanho, com o código realizando mais em menos linhas. O método consegue, baseando se nos dados prévios,

prever o tamanho com uma certa porcentagem de corretude.

2.2 Erros

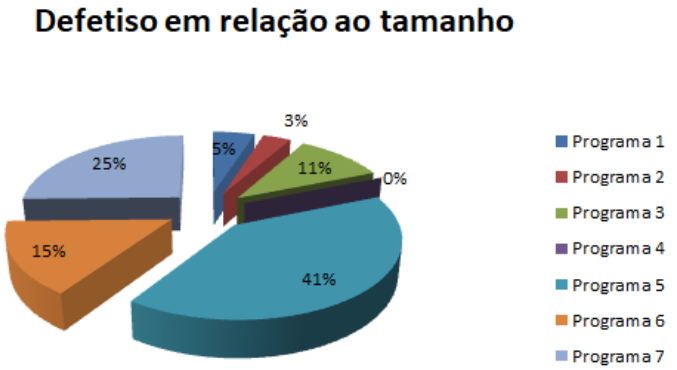
Como podemos observar na tabela 3, a maioria dos erros ocorreram nos prmeiros programas, dada a inexperiencia com a linguagem e possiveis imcompreensões do problema. Apesar dos problemas continuarem até o fim dos experiemtnos, seu número diminuiu se comparado com o tamanho dos projetos.

Programa	**Erros**
Programa 1	4
Programa 2	9
Programa 3	4
Programa 4	0
Programa 5	1
Programa 6	2
Programa 7	5
Total:	25

Tabela 3: Erros em todos os programas

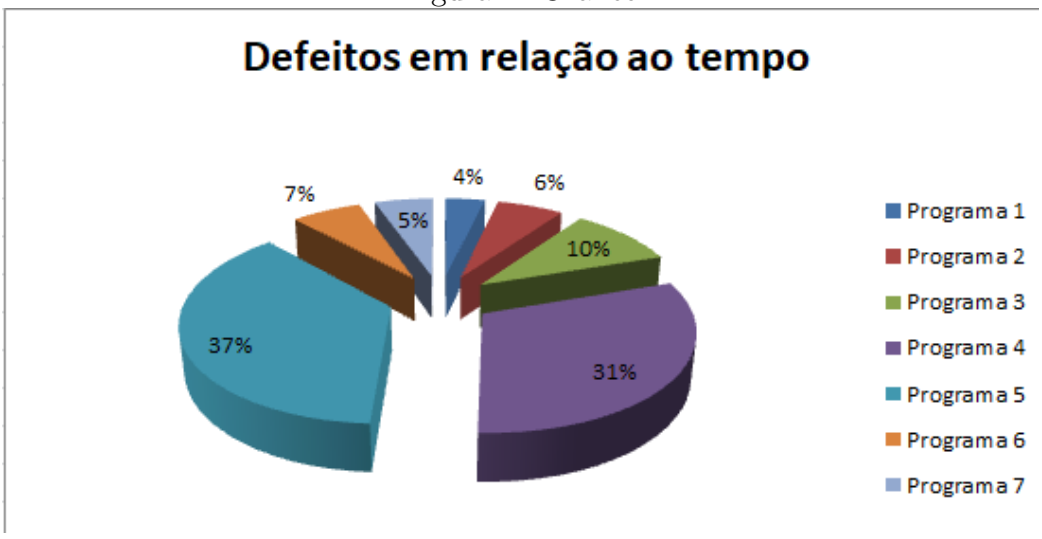
Como podemos observar no gráfico 1 abaixo, a relação entre tamanho e erros.

Figura 1: Grafico



Pode-se inferir que apesar dos programas maiores terem um número consideravel de erros, os programas menos complexos tem um número relativamente grande de erros.

Figura 2: Gráfico



Como pode ser observado na Figura 2, que os primeiros projetos tem maior tempo e número de erros compativel. Já os estimados pelo método,

com tempo menor, apresentam poucos erros, apesar de serem mais complexos e longos.

Erros inseridos por fase podem ser observados na tabela 4

Fase	**Erros**
Antes do desenvolvimento	0
Planejamento	0
Projetando	2
Revisão de projeto	1
Código	11
Revisão de código	0
Compilar	0
Teste	0
Total:	14

Tabela 4: Erros inseridos por fase

Erros corrigidos por fase podem ser vistos na tabela 5

Fase	**Erros**
Antes do desenvolvimento	0
Planejamento	0
Projetando	2
Revisão de projeto	0
Código	8
Revisão de código	1
Compilar	0
Teste	5
Total:	14

Tabela 5: Erros removidos por fase

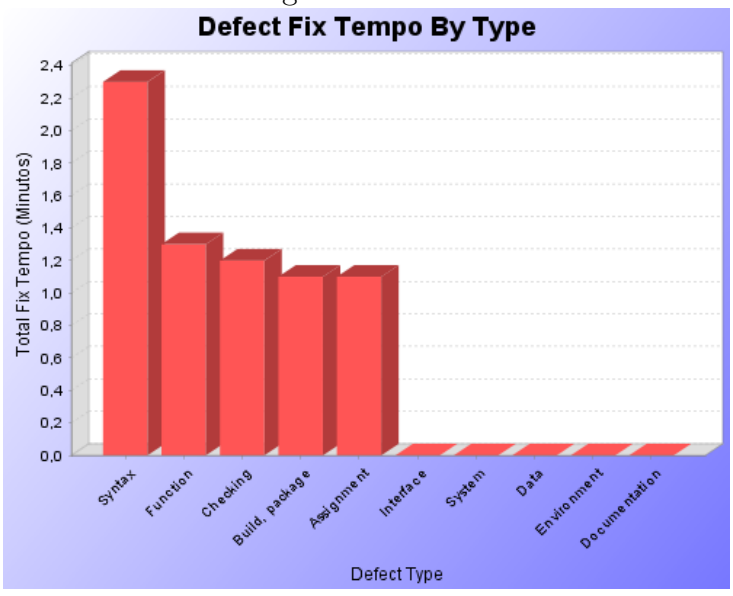
Na tabela abaixo encontram-se os erros por tipo:

Tipo de erro	Quantidade
Função	2
Checking	2
Syntax	4
Build, package	2
Assignment	2
Total:	12

Tabela 6: Erros por tipo

Os erros acima quantificados, foram em sua maioria por desconhecimento da linguagem, pois se tratam de conversões erradas ou utilização de forma erradas de métodos prontos. Nas de checagem, os erros aconteceram devido erros matemáticos, onde os numeros inseridos causavam excessões por divisão por zero ou números negativos. Tais correções não tomavam mais de 3 minutos para serem feitas, como podem ser observadas na figura 3.

Figura 3: Gráfico



2.3 Reutilização de código

A reutilização do código pode ajudar na produção mais rápida de um produto pois tradatasse de um código já testado e possivelmente livre de problemas. Contudo, caso certos cuidados não sejam tomados, a possibilidades de um novo projeto herdar problemas oriundos de outros projetos é grande, comprometendo a qualidade dos projetos a seguir. Levando em consideração que os problemas a serem solucionados nesse experimento, os quais tem natureza matemática, a reutilização de código é propicia dada a constante utilização de calculos, tais como média, somatórias e desvio padrão.

A tabela 7 demonstra a quantidade de código reutilizado nos sete programas:

Programa	**Codigo reutilizavel**	**Código Reutilizado**
Programa 1	0	0
Programa 2	0	0
Programa 3	5	5
Programa 4	70	65
Programa 5	0	0
Programa 6	70	65
Programa 7	230	212

Tabela 7: Código reutilizado

O programa 7 apresentou a maior quantidade de código reutilizavel devido a sua complexidade e a encorporação de várias funções já implementadas no anterior. É importante frisar que algumas partes desses códigos foram levemente modificadas a fim de acoplá-las as demais partes.

Referências

[1] Robert Cannon Mark Sebern Contributor Watts S. Humphrey Marsha Pomeroy-Huff, Julia L. Mullaney. *The Personal Software Process (PSP) Body of Knowledge, Version 1.0*. Software Engineering Institute, August 2005.