5.3 PSP1.1

Para que o planejamento de uma atividade possa ser executado com sucesso é necessário obter duas informações importantes: o tamanho da atividade e a produtividade de quem vai desenvolvê-la. A primeira, já foi discutida e apresentada no nível PSP1, através do método PROBE. A segunda, deve ser obtida a partir do histórico do desenvolvedor para projetos similares. De posse destas duas informações, será possível estabelecer a quantidade de tempo necessária para completar o trabalho.

Para auxiliar o engenheiro de *software* nas atividades de planejamento e acompanhamento, o PSP oferece, no nível PSP1.1, os formulários *C47 – Task Planning Template* e *C49 – Schedule Planning Template*. Em conjunto, estas duas ferramentas são a base segura para firmar compromissos.

Os objetivos do PSP1.1 são fazer planos de recursos e cronograma, acompanhar o desempenho do desenvolvedor contra estes planos e julgar as prováveis datas de término do projeto.

5.3.1 O planejamento das tarefas

O primeiro passo para fazer um bom plano é dividir o trabalho em tarefas. Ao final de cada tarefa deve ser previsto um ponto de controle, com um produto que possa assegurar sua completeza. Isto facilita o acompanhamento e o gerenciamento do trabalho, permitindo identificar quando as coisas não vão indo bem e quando é necessário rever os planos e os recursos.

Por exemplo, a primeira tarefa para o desenvolvimento de um programa é documentar os requisitos do usuário. O produto típico desta fase é o documento de requisitos aprovado pelo usuário. Uma vez o documento aprovado, pode-se considerar que a tarefa de documentar os requisitos do usuário esteja concluída.

Task		Plan					Actual		
#	Name	Hours	Planned Value	Cumulative Hours	Cumulative Planned Value	Date Monday	Date	Earned Value	Cumulative Earned Value
1	Plano	15	10.35	15	10.35	31/07	01/08	10.35	10.35
2	Introdução	8	5.52	23	15.87	07/08			
3	Capítulo 1	21	14.48	44	30.35	14/08			
4	Capítulo 2	45	31.03	89	61.38	21/08			
5	Capítulo 3	56	38.62	145	100.00	11/09			
	Totals	145	100.00	145	100.00				

Figura 15. Planejamento das tarefas.

Para o planejamento das tarefas o PSP utiliza o formulário *C47 – Task Planning Template*, ilustrado na Figura 15. Conforme o exemplo da figura, o trabalho para escrever um manual do usuário composto de 3 capítulos é inicialmente dividido em tarefas (planejamento, introdução, capítulo 1, capítulo 2 e capítulo 3). Em seguida, é estimado o tempo (em horas) para cada uma das tarefas. Este número é derivado da estimativa de tamanho (usando o PROBE aplicado ao histórico de desenvolvimento de manuais do usuário do próprio desenvolvedor) e da produtividade.

O PSP utiliza a técnica do valor agregado (do original, em inglês, *earned value*). Ou seja, a cada uma das tarefas é atribuído um valor (percentual da tarefa em relação ao todo), registrado na coluna *Planned Value* e um ponto de controle claro e preciso, que possa identificar que a tarefa foi concluída e que o valor a ela atribuído pode ser creditado ao projeto. Desta forma, ao final de cada tarefa, o desenvolvedor terá a noção clara de quanto já tem de valor agregado.

No exemplo da Figura 15, a primeira atividade, cuja duração é de 15 horas, corresponde a 10.35% do esforço total. Isto significa que, uma vez que a tarefa de planejamento do manual do usuário esteja pronta, poderá ser creditado um valor de 10.35 ao projeto. E assim sucessivamente até o seu término.

As colunas dos valores acumulados representam os valores acumulados planejados após o término de cada atividade. Por exemplo, ao final do capítulo 1, o desenvolvedor terá creditado o valor total acumulado de 15.87.

As datas constantes da coluna *Date Monday* representam as segundas-feiras das semanas nas quais cada uma das atividades será concluída. Esta coluna só poderá ser preenchida em conjunto com o formulário C49.

As colunas que conterão os dados reais serão preenchidas à medida que as tarefas forem sendo concluídas. Por exemplo, na Figura 15, a tarefa 1 foi concluída no dia 01/08 e agregou ao projeto o valor de 10.35.

5.3.2 O planejamento do cronograma

Uma vez determinado o tempo necessário para cada tarefa será preciso conhecer o tempo que estará realmente disponível para a execução do projeto, de modo a estabelecer o cronograma. É comum que o desenvolvedor esteja envolvido em outras atividades e que não tenha 100% do seu tempo disponível para o projeto. Também é preciso pensar nas paradas naturais para descanso, interrupções etc. Uma medida considerada em muitas empresas é que o tempo líquido de trabalho esteja em torno de 75% do tempo total.

O PSP auxilia o planejamento e acompanhamento do cronograma através do formulário *C49 – Schedule Planning Template*, ilustrado na Figura 16.

Na fase de planejamento do projeto o desenvolvedor deverá entrar com as datas de início de cada semana, ou seja, as datas correspondentes às segundas-feiras. Em seguida, para cada semana, deverá estimar o total de horas que estará disponível para o projeto (diretas e acumuladas).

Na coluna *Cumulative Planned Value* o desenvolvedor deverá colocar o valor agregado conforme a previsão de conclusão das tarefas. Por exemplo, a primeira tarefa tem um tempo previsto de 15 horas. Como o desenvolvedor prevê que disporá de 15 horas para o projeto na primeira semana, significa que a primeira atividade será concluída nesta semana e que o valor agregado correspondente, 10.35, poderá ser creditado ao final da primeira semana.

Note que a atividade 4, escrever o capítulo 3 do manual do usuário, vai requerer 45 horas de trabalho. Isto significa que não poderá ser concluída em uma semana de 15 horas. Isto se reflete no cronograma pelo fato de não ser creditado nenhum valor agregado adicional na semana 6 e apenas na semana 7.

Na fase de desenvolvimento do projeto, à medida que as atividades vão sendo terminadas, os dados reais vão sendo registrados nas colunas *Actual*.

		Plan			Actual			
Week No.	Date Monday	Direct Hours	Cumulative Hours	Cumulative Planned Value	Direct Hours	Cumulative Hours	Cumulative Planned Value	Adjusted Earned Value
1	31/07	15	15	10.35	17	17	10.35	8.57
2	07/08	15	30	15.87	16	33	15.87	13.40
3	14/08	15	45	30.35				
4	21/08	15	60	30.35				
5	28/08	15	75	61.35				
6	04/09	15	100	61.35				
7	11/09	15	115	100				

Figura 16. Planejamento do cronograma.

A última coluna *Adjusted Earned Value* será utilizada para fazer pequenos ajustes no projeto. Por exemplo, vamos supor que ao terminar a primeira atividade o desenvolvedor descobriu que teria que incluir mais um capítulo (capítulo 4) e que levaria 30 horas para concluí-lo. Para não precisar refazer todo o plano, basta reduzir o valor agregado das atividades, de modo a compensar a inclusão de mais esta atividade. Ou seja, 100% não representará mais 145 horas mais sim 145 + 30 = 175 horas. Portanto, um reajuste de 0,8286 em todos os valores agregados será necessário. Para a atividade 1, conforme ilustra a Figura 16, ao invés de um valor agregado inicialmente proposto de 10.35, teremos 8.57.

O autor sugere que se use o bom senso na hora de definir os pontos de controle. Acompanhamento diário pode parecer excessivo para um trabalho de várias semanas. Porém, para trabalhos mais curtos, pode ser necessário este acompanhamento mais freqüente. De

qualquer forma, o acompanhamento regular do cronograma através do conceito do valor agregado permite ao desenvolvedor saber exatamente em que ponto do projeto ele se encontra e, consequentemente, perceber em tempo quando está se desviando do plano.

5.3.3 O Sumário de Projeto PSP1.1

O PSP1.1 introduz poucas mudanças no *Project Plan Summary*, conforme ilustra a parte em itálico da Figura 17.

Em relação ao nível anterior, é incluído o campo referente ao CPI - *Cost-Performance Index*. O *CPI* é dado pela razão entre o tempo total planejado acumulado e o tempo total real acumulado. O CPI expressa o quanto o desenvolvedor está atingindo seus compromissos de custo. O ideal é que o CPI seja próximo a 1, refletindo que sua capacidade de planejamento está melhorando, uma vez que o tempo planejado está próximo ao tempo real gasto. Números inferiores a 1 indicam que se está gastando mais tempo do que o planejado, ou seja, está se subestimando as atividades. Números muito superiores a 1 indicam que se está superestimando as atividades.

Além disto, são introduzidos campos para dois tipos de percentuais: percentual de utilização de código a partir da biblioteca de reuso (*% Reused*) e percentual de código novo a ser desenvolvido e incorporado à biblioteca de reuso (*% New Reused*).

Summary	**Plan**	**Actual**	**To Date**
LOC/Hour			
Planned Time			
Actual Time			
CPI(Cost-Performance Index)			(Planned/Actual)
% Reused			
% New Reused			

Figura 17. ***C45 - PSP1.1 Project Plan Summary*** **– modificações.**

5.4 Exercício 3

Objetivo	Aprender a planejar atividades segundo o PSP1.1 e a analisar o impacto de situações não previstas.
Descrição	Uma escola está contratando sua empresa para o desenvolvimento de um sistema para a emissão dos bloquetos de cobrança das mensalidades e estabelece as seguintes condições para o contrato: - R$ 5.000,00 pelo produto entregue; - R$ 500,00 de multa por dia corrido de atraso; - início do projeto 15/01/2001; - prazo para entrega do produto: 02/02/2001 às 18:00h. Você sabe que já tem os seguintes compromissos agendados: - 25 % do tempo diário alocado a outras atividades; - 16/01/2001 das 9:00 às 12:00h: reunião inadiável com um cliente; - 24/01/2001 das 14:00 às 16:00h: abertura de um edital de licitação; - 01/02/2001: viagem a negócios (ida e volta no mesmo dia). Seu histórico de atividades o levou às seguintes tarefas e tempos: - Levantamento dos Requisitos (4:00h) - Planejamento (2:00h) - Projeto (12:00h) - Codificação (24:00h) - Compilação (2:00h) - Testes (16:00h) - Manual do Usuário (12:00h) - Testes de Aceitação (6:00h) Responda as seguintes questões: - será possível realizar o projeto no prazo estipulado pela escola? - o que acontecerá se a atividade de compilação demorar o dobro do tempo e quanto receberá neste caso? - o que acontecerá se você só retornar da viagem de negócios no dia seguinte na hora do almoço e quanto receberá neste caso? Considerar que você tem umajornada diária de 8:00h.
Formulários	*C47 – Task Planning Template* *C49 – Schedule Planning Template*

Passo a Passo	**Planejamento das Tarefas:** ▪ Entre com todas as tarefas e os respectivos tempos no formulário *C47 – Task Planning Template* ▪ Calcule os valores das colunas *Planned Value, Cumulative Hours, Cumulative Planned Value* **Planejamento do Cronograma:** ▪ Entre com as datas das segundas-feiras no formulário *C49 – Schedule Planning Template* ▪ Calcule as horas que estarão disponíveis para o projeto em cada uma das semanas. Preencha o acumulado destas horas por semana ▪ Identifique o valor que será agregado ao projeto a cada semana, utilizando como base as informações do formulário *C47 – Task Planning Template* sobre as atividades que poderão ser completadas naquela semana ▪ Faça as análises solicitadas **Desenvolvimento (não será realizado para este exercício)** **Autópsia (não será realizada para este exercício)**

Table C47 - Task Planning Template

Student _____ Date _____

Project _____ Instructor _____

Task		Plan					Actual		
#	Name	Hours	Planned Value	Cumulative Hours	Cumulative Planned Value	Date Monday	Date	Earned Value	Cumulative Earned Value
Totals									

Table C47 - Task Planning Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Project:* nome do programa // *Instructor:* nome do professor

Task:

- *Number:* número da tarefa (entrar as tarefas na ordem em que se pretende desenvolvê-las).
- *Name:* nome da tarefa. Selecionar tarefas que tenham critérios de completeza bem definidos. Ex.: Planejamento completado, programa compilado e todos os defeitos corrigidos, testes completados e defeitos corrigidos, etc.

Plan:

- *Hours:* quantidade planejada de horas para completar cada tarefa (antes do desenvolvimento).
- *Planned Value:* % que expressa o total de horas planejado para a tarefa em relação ao total de horas planejado para o projeto; a soma de todos os percentuais planejados deve ser igual a 100% (antes do desenvolvimento).
- *Cumulative Hours:* para cada tarefa, some o tempo planejado para ela com o tempo total das tarefas anteriores (antes do desenvolvimento).
- *Cumulative Planned Value:* para cada tarefa, some o valor planejado para ela com o valor total das tarefas anteriores; a última tarefa deverá conter o valor 100% (antes do desenvolvimento).
- *Date Monday:* preencha com a data da segunda-feira da semana na qual se pretende completar a tarefa. Para isto, leve em consideração o tempo acumulado para a tarefa (ou seja, a soma de todos os tempos desta e das atividades anteriores a ela – *Cumulative Hours*) e o tempo acumulado que se terá disponível para o projeto (*Planned Cumulative Hours* do formulário *C49 – Schedule Planning Template*).

Actual:

- *Date:* data em que a tarefa foi efetivamente completada.
- *Earned Value:* para cada tarefa completada, entre com o valor planejado (*Planned Value)*, tornando-o valor agregado. Apenas tarefas 100% executadas devem constituir valor agregado. Não se deve somar valores para tarefas parcialmente completadas.
- *Cumulative Earned Value:* para cada tarefa completada, entre com a soma dos valores agregados até o momento (soma de todos os valores planejados já executados até o momento), tornando-os valores agregados acumulados.

Table C49 - Schedule Planning Template

Student _____ Date _____
Project _____ Instructor _____

		Plan			Actual			
Week No.	Date Monday	Direct Hours	Cumulative Hours	Cumulative Planned Value	Direct Hours	Cumulative Hours	Cumulative Earned Value	Adjusted Earned Value

Table C49 – Schedule Planning Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Project:* nome do programa // *Instructor:* nome do professor

Week:

- *Week Number:* número da semana (iniciando em 1).
- *Date Monday:* primeiro dia útil de cada semana (Segunda-feira).

Plan:

- *Direct Hours:* quantidade de horas que se planeja gastar diretamente com o projeto durante a semana. Levar em consideração: férias, feriados, tarefas em outros projetos, etc.
- *Cumulative Hours:* para cada semana, some o tempo planejado para ela com o tempo total planejado das semanas anteriores, obtendo um valor acumulado.
- *Cumulative Planned Value:* verifique a coluna *Cumulative Hours – Plan* do formulário *C47 – Task Planning Template*. Se for menor que o valor da coluna *Cumulative Hour – Plan* deste formulário, significa que a tarefa será concluída nesta semana e que, portanto, este campo deverá ser preenchido com o valor constante no *Cumulative Planned Value – Plan* do formulário C47. Caso contrário, esta tarefa não será concluída nesta semana e, portanto, deverá ser repetido o valor da linha anterior, significando que não será agregado valor ao projeto nesta semana.

Actual:

- *Direct Hours:* ao final de cada semana, entre com o total de horas realmente gasto no projeto durante aquela semana.
- *Cumulative Hours:* ao final de cada semana, entre com o total de horas acumulado realmente gasto com o projeto até aquela semana.
- *Cumulative Earned Value:* ao final de cada semana, entre com o total de valor agregado.
- *Adjusted Earned Value*: utilize este espaço para ajustar o valor agregado à medida que novas tarefas entrem no planejamento ou alguma tarefa seja retirada, evitando reajustes maiores no plano.

6 PERSONAL QUALITY MANAGEMENT - PSP2 E PSP2.1

"É melhor prevenir do que remediar."

6.1 PSP2

Como vimos anteriormente, os níveis iniciais PSP0 e PSP0.1 têm o objetivo de construir a base para as melhorias, através da incorporação das primeiras métricas ao processo do desenvolvedor. Os níveis seguintes, PSP1 e PSP1.1, desenvolvem a habilidade do planejamento, através da introdução do método de estimativas PROBE e dos guias para a elaboração e acompanhamento do cronograma. Nos níveis PSP2 e PSP2.1, é hora de pensar em qualidade pessoal.

Qualidade pessoal, no PSP2, é traduzida pela incorporação das revisões ao processo do desenvolvedor. Conforme Humphrey, em [HUM 95]:

"Fazer revisões é o passo mais importante que se pode dar para melhorar o desempenho em engenharia de *software*."

Escrever um programa pode ser comparado a escrever um livro. Apesar dos livros começarem a ser escritos através de um rascunho, não é o rascunho que é publicado como versão final. Na verdade ele começa como um rascunho, é inúmeras vezes revisado e, por vezes até escrito novamente, para, finalmente, ser publicado e distribuído. O problema com os programas é que muitas vezes o desenvolvedor não acha necessário revisá-lo. Ele pensa que pode escrever um

programa de boa qualidade já da primeira vez. Como isto nunca acontece, ele entra em um ciclo interminável de compilar-alterar-testar.

Revisar programas traz economia de tempo (e consequentemente de dinheiro), torna o processo mais estável e previsível e, certamente, resulta em um produto de melhor qualidade. Nas revisões se encontram diretamente os problemas, através dos testes encontram-se apenas os sintomas dos problemas.

Números provenientes dos cursos de PSP mostram que, em média, quando os desenvolvedores começam a usar o método, 30% do tempo é gasto nas fases de compilação e testes. No final do curso, quando já incorporaram o processo de revisões, apenas 10% do tempo total é gasto nas fases de compilação e testes.

Há três métodos principais de revisões: inspeções, *walk-throughs* e revisões pessoais. Os dois primeiros envolvem a revisão em equipes e o terceiro, como o próprio nome diz, o desenvolvedor efetua sozinho. Inspeções são métodos mais formais, com uma estrutura e papéis bem definidos. *Walk-throughs* são revisões menos formais, no estilo de apresentação. As revisões pessoais são aquelas que o desenvolvedor efetua sobre o seu próprio programa. É nestas que o PSP se concentra.

Os princípios básicos para se ter uma boa revisão são:

- estabelecer objetivos para a revisão: o objetivo do PSP é encontrar e corrigir todos os defeitos antes da primeira compilação.
- seguir um processo de revisão definido: o processo de revisão é um processo como outro qualquer e requer portanto, critérios de entrada, atividades e critérios de saída.
- medir e melhorar o processo de revisão: o ideal é remover o máximo de erros em um mínimo de tempo.

Ao efetuar as revisões é importante coletar métricas a respeito delas, de modo a poder avaliar o processo utilizado. As quatro métricas básicas das revisões são:

- o tamanho do programa sendo revisado (na fase de projeto ainda não temos LOC, mas podemos nos basear nas LOC estimadas);
- tempo de revisão em minutos;
- o número de defeitos encontrados;
- o número de defeitos encontrados posteriormente no programa.

A partir destas, podemos obter outras métricas derivadas que nos permitirão avaliar a eficiência do processo de revisão:

- o percentual de defeitos do programa que foi encontrado durante a revisão;
- a quantidade de defeitos / KLOC encontrada na revisão;
- a quantidade de defeitos encontrado por hora de revisão;
- a quantidade de LOC revisadas por hora.

É importante separar as revisões de projeto *(Design Reviews)* das revisões de código *(Code Reviews)*. Isto permite tornar o projeto mais compreensível, economizar tempo de implementação, evitar deixar passar algum defeito importante e propiciar a descoberta de pontos de melhoria.

6.1.1 As revisões de projeto

Que revisar é importante, não há dúvida. Porém, para o sucesso de uma Revisão de Projeto é importante produzir projetos que possam ser revisados. Projetos desorganizados e confusos são difíceis de revisar e tornam as revisões pouco eficientes. Portanto, o objetivo da fase de projeto deve ser, além de produzir um projeto sem defeitos, também produzir um projeto que possa ser revisado.

Seguir uma estratégia de revisão e revisar o projeto em etapas também auxilia nesta tarefa. As Revisões de Projeto no PSP são orientadas pelo formulário *C57 – C++ PSP2 Design Review Checklist*. Este formulário será alterado no nível seguinte para incorporar novos elementos.

É importante observar que este é um formulário que auxilia a revisar projetos de programas em C++. Se o desenvolvedor estiver utilizando outra linguagem de programação, o *checklist* tanto de Revisão de Projeto, quanto de Revisão de Código, deverá ser alterado para contemplar as particularidades da linguagem.

Special Cases	Check all special cases: - empty, full, minimum, maximum, negative, zero - out of limits, overflow, underflow - ensure "impossible" conditions are absolutely impossible - handle all incorrect input conditions	 X X X X		

Figura 18. *Checklist* para Revisão de Projeto – exemplo.

Este formulário está dividido em seções, agrupando os tópicos por assunto. A Figura 18 ilustra a seção dos Casos Especiais. Neste momento, o desenvolvedor deverá checar se foi previsto o tratamento adequado para: vazio, cheio, mínimo, máximo, negativo, zero; fora dos limites: *overflow*, *underflow*; assegure que condições impossíveis realmente sejam impossíveis; trate todas as exceções dos dados de entrada. Os quadrinhos em branco ao lado das informações indicam os espaços para marcar que aquele item foi checado.

6.1.2 As revisões de código

As Revisões de Código no PSP são orientadas pelo formulário *C58 – C++ PSP2 Code Review Checklist*. Este formulário serve de guia para uma Revisão de Código em C++. O desenvolvedor deverá desenvolver seu próprio *checklist* para outras linguagens, incorporando aspectos que sejam relevantes para a linguagem em uso.

Assim como o anterior, este formulário também está dividido em seções, agrupando os tópicos por assunto. O desenvolvedor deve checar em seu programa as condições apresentadas e assinalar no formulário. A Figura 19 ilustra a seção de Inicialização. Nela deve ser checado se as variáveis foram devidamente inicializadas: no início do programa, a cada início de loop, na entrada de funções/*procedures*.

Initialization	Check variable and parameter initialization: - at program initiation - at start of every loop - at function/procedure entry	 X X X			

Figura 19. *Checklist* para Revisão de Código – exemplo.

Uma questão polêmica no que se refere ao PSP é o momento de fazer a Revisão de Código. Há muitos argumentos dos desenvolvedores para que ela não seja feita antes da compilação. Afinal, os compiladores estão preparados para pegar a maioria dos erros de sintaxe. Porém, segundo o autor, ao buscar consertar os defeitos encontrados pelo compilador, o desenvolvedor perde o foco da qualidade do programa e fica fazendo apenas tentativas para "fazer o programa rodar". Este é um aspecto muito importante, porque uma revisão bem feita não apenas detecta os erros, como também identifica oportunidades de melhorar a lógica e a clareza do programa.

6.1.3 Como prevenir defeitos?

As revisões podem funcionar como uma forma de prevenir defeitos nos estágios posteriores. Porém, para ser realmente eficiente, é importante observar algumas questões:

- usar os melhores métodos para cada fase do ciclo: não apenas os melhores métodos como também aqueles que mais se adaptam ao perfil do projeto e do desenvolvedor;
- analisar constantemente o processo: identificar os pontos onde ele não parece ser eficiente e promover as mudanças necessárias;
- experimentar novos métodos e ferramentas: nem todos os métodos são apropriados para todos os desenvolvedores ou para todos os tipos de produtos;

- incorporar novos métodos e ferramentas que melhorem o desempenho;
- iniciar um programa de prevenção de defeitos: ao identificar quais são os erros que o desenvolvedor mais comete, é possível estabelecer prioridade sobre eles.

6.1.4 O Sumário de Projeto PSP2

O formulário de projeto do nível PSP2 passa a incorporar as duas novas fases inseridas ao processo: a Revisão de Projeto e a Revisão de Código. Elas passam a fazer parte das seções relativas à sumarização dos tempos por fase e também dos defeitos inseridos e removidos por fase.

Além disto, o formulário passa a incorporar, na seção inicial de sumário, dados sobre a densidade dos defeitos encontrados na fase de teste em relação ao total de LOC novas e modificadas e a densidade total de defeitos. Uma seção específica para abrigar os dados provenientes das revisões, denominada *Defect Removal Efficiency*, é acrescentada ao final do formulário. Esta seção, como o próprio nome sugere, destina-se possibilitar a análise da eficiência das revisões, conforme ilustra a Figura 20.

Um novo conceito surge que é o de *yield*. No contexto do PSP ele significa o percentual de defeitos que foi inserido e removido antes da primeira compilação. Na prática, ele expressa a eficiência dos processos de revisão.

	Plan	*Actual*	*To Date*
Test Defects/KLOC			
Total Defects/KLOC			
Yield %			
Defect Removal Efficiency	*Plan*	*Actual*	*To Date*
Defects/Hour - Design review			
Defects/Hour - Code review			
Defects/Hour - Compile			
Defects/Hour - Test			
DRL(DLDR/UT)			
DRL(CodeReview/UT)			
DRL(Compile/UT)			

Figura 20. *C55 – PSP2 Project Plan Summary* **– modificações.**

6.2 PSP2.1

No nível PSP2 o grande foco era a realização das revisões. Já no PSP2.1, o foco é melhorar a qualidade da fase de projeto do programa, através da introdução de quatro *templates*:

- *C66 – Operational Scenario Template*
- *C68 – Functional Specification Template*
- *C70 – State Specification Template*
- *C72 – Logic Specification Template*

Estes *templates* são baseados em métodos orientados a objetos. Caso a empresa não utilize métodos orientados a objetos, então o desenvolvedor deverá promover as adaptações necessárias para contemplar as ferramentas de especificação de projetos utilizadas em seu ambiente.

Os objetivos do nível PSP2.1 são: auxiliar o desenvolvedor a reduzir a quantidade de defeitos da fase de projeto, prover critérios objetivos para julgar se um projeto está completo e prover um *framework* consistente para verificar a qualidade do projeto.

6.2.1 O cenário operacional

Uma das ferramentas utilizadas para auxiliar o desenvolvedor a melhor especificar seu programa é o formulário *C66 – Operational Scenario Template*, ilustrado na Figura 21. Através dele é possível documentar os cenários possíveis de utilização do programa, mostrando como ele deverá reagir a cada uma destas condições.

Scenario # 1		**User Objective:** criar um novo registro no cadastro de clientes	
Scenario Objective: identificar possíveis erros provenientes do usuário			
Source:	**Step**	**Action:**	**Comments**
Usuário	1	Inicia o programa	
Programa	2	Exibe o menu principal Solicita seleção de opção	
Usuário	3	Entra com a opção	Opção inválida
Programa	4	Exibe msg: "Opção inválida" Exibe o Menu Principal Solicita seleção de opção	
Usuário	5	Entra com a opção de inclusão de novo cliente	Opção válida
Programa	6	Exibe tela de entrada de cliente Solicita preenchimento dos dados	
Usuário	7	Entra com a idade negativa	Opção inválida
Programa	8	Exibe msg: "Idade não pode ser negativa." Posiciona o cursor sobre o campo de idade	

Figura 21. Cenário operacional.

Sabe-se que não será possível prever todas as situações de uso, porém, com um pouco de experiência o desenvolvedor conseguirá prever as principais condições. Este *template* auxiliará também no planejamento dos casos de teste. Uma vez que as condições tenham sido estudadas na fase de projeto, revisadas pela Revisão de Projeto, implementadas na fase de codificação e revisadas na Revisão de Código, chegarão com muito menos erros à fase de testes.

Na Figura 21 temos ilustrado um programa para a entrada de um novo cliente no cadastro de clientes. Como se pode observar, o objetivo deste cenário é documentar as situações em que o usuário poderá entrar com informações inválidas e como o sistema deverá tratar estes casos.

Por exemplo, no passo 7, o usuário tenta cadastrar uma idade negativa. A resposta do sistema deverá ser exibir uma mensagem orientando sobre o problema e posicionar o cursor sobre o campo de idade, para que o usuário possa efetuar a correção.

Este é um dos formulários que auxiliará o desenvolvedor a reduzir a quantidade de erros descobertos na fase de teste. Se o projeto do programa for bem feito neste momento, significa que as chances de problemas nas fases posteriores do ciclo de vida serão bem menores.

6.2.2 A especificação funcional

O objetivo da especificação funcional é apresentar um panorama dos objetos, seus atributos e métodos. O formulário utilizado é o *C68 – Functional Specification Template*, ilustrado na Figura 22. Como se pode observar, o objetivo é identificar o objeto (ou classe), as condições de herança, os atributos e os métodos que os manipulam.

Para a descrição da funcionalidade do método, Humphrey utiliza como padrão a lógica preposicional, de modo a evitar ambigüidades. Pode ser que o padrão usado na organização do desenvolvedor seja outro, ou até mesmo que ele não tenha muita familiaridade com uma especificação mais formal. Neste caso, o padrão da empresa ou do próprio desenvolvedor deverá ser adotado. No entanto, é importante lembrar que a representação usando conceitos de lógica preposicional auxilia a prevenir eventuais dúvidas de interpretação.

Object/class name Cdata	**Parent Classes** Classe base	**Attributes** EstadoDaLista (0 a 4) PosiçãoDaLista (0 a n)
Method declaration	**Method external specification**	
Int Vazio()	EstadoDaLista == 0 :: return(true) V EstadoDaLista ≠ 0 :: return (false)	
Int Limpar()	:: setar os ponteiros de Cdata para nulo Λ EstadoDaLista = 0 Λ PosiçãoDaLista = 0 Λ return (true)	
...		

Figura 22. Especificação funcional.

Acompanhando o exemplo da Figura 22, temos que o método Vazio() da classe Cdata avalia se a lista está vazia. Ele faz isto através do atributo EstadoDaLista. Se este atributo for igual a zero, a lista está vazia e o método retorna verdadeiro. Caso, contrário, se a lista não estiver vazia, retorna falso. Na relação de atributos temos que o atributo EstadoDaLista pode assumir os valores 0, 1, , 3 ou 4. Como podemos observar, todos os principais elementos necessários à especificação funcional foram considerados.

6.2.3 A máquina de estados

Outra ferramenta que pode ser útil na fase de projeto do programa é a máquina de estados. No PSP ela é representada de forma descritiva através do formulário *C70 – State Specification Template*, ilustrado na Figura 23. Este formulário é utilizado para prever e mapear o comportamento do estado do objeto.

Note que, assim como os demais *templates* utilizados na fase de projeto, este também poderá ser substituído por uma representação gráfica. Isto pode ocorrer especialmente se houver uma ferramenta CASE disponível.

A Figura 23 ilustra a especificação de estados para analisar a situação de uma lista de nomes. O estado que está sendo representado é o EstadoVazio. Ou seja, para que a lista de nomes continue vazia, deverá ocorrer uma operação de retirada da lista (Pop ou Subtract). Para transitar do EstadoVazio e chegar a um EstadoDoMembro, basta executa uma operação de adição de elementos à lista (Push ou Add).

State #1 EstadoVazio	**Description** A lista não tem nomes	**Attributes** N == 0
EstadoVazio	Pop(&D) V Subtract (D)	
EstadoDoMembro	Push(D) V Add(D)	
. . .	...	
. . .	...	
Próximo estado # n	...	

Figura 23. Especificação de estados.

Alguns pontos são importantes quando da construção da máquina de estados de um objeto. Estes tópicos passam a fazer parte do *checklist* que será utilizado na Revisão de Projeto:

- verificar se a estrutura não contém possíveis loops;
- verificar se a estrutura está completa, ou seja, se todos os estados foram identificados;
- verificar se a estrutura é ortogonal, ou seja, para todo o conjunto de condições existe um e somente um próximo estado possível;

- verificar se as transições a parir de cada estado são completas e ortogonais, ou seja, a partir de todos os estados um único próximo estado é definido para uma dada combinação de dados de entrada.

6.2.4 A especificação lógica

A especificação lógica nada mais é do que a forma tradicional de se especificar programas/métodos: o pseudocódigo ou português estruturado.

No PSP isto é feito através do formulário *C72 – Logic Specification Template*, ilustrado na Figura 24. Note que, assim como os demais *templates*, este também poderá ser substituído pela ferramenta CASE em uso na empresa, se houver.

Da mesma forma algumas informações podem não ser pertinentes à linguagem em uso, como por exemplo a seção *Includes*. Neste caso, o desenvolvedor deverá promover as adaptações necessárias a refletir suas necessidades.

No exemplo da Figura 24, temos o exemplo de um trecho de especificação lógica.

INCLUDES: <stdio.h> <string.h>	
TYPE DEFINITIONS: char data[ARRAYSIZE][STRINGLENGTH + 1]	
Declaration:	Int Aset::SubtractSet(data D)
	....
Reference:	
logic reference numbers	**Program logic, in pseudocode**
	Done = 0
1	If Empty
2	If D == first item
	Delete first item
	Reset set pointer and next item pointer
	Done ++
3	.....
4	.....

Figura 24. Especificação lógica.

6.2.5 O Sumário de Projeto PSP2.1

Conforme ilustra a Figura 25, o sumário do projeto é alterado para abrigar o conceito de custo da qualidade (a seção dos tempos por fase aparece apenas para ilustrar os cálculos). No PSP este conceito de custo é expresso através de 3 indicadores:

- *% Appraisal COQ* : relação entre o tempo gasto nas Revisões (de Código e de Projeto) e o tempo total de desenvolvimento;
- *% Failure COQ:* relação entre o tempo gasto em compilação e testes e o tempo total de desenvolvimento;
- *COQ A/F Ratio – Cost of Qaulity Appraisal to Failure Ratio*: relação entre o tempo gasto nas revisões (de código e de projeto) e o tempo gasto em compilação e testes.

Summary	**Plan**	**Actual**	**To Date**	
% Appraisal COQ	11.8	6.7	1.8	
% Failure COQ	22.1	8.5	26.2	
COQ A/F Ratio	0.53	0.79	0.07	
Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date%**
Planning	27	33	227	8.5
Design	82	55	666	24.8
Design review	0	0	0	0.0
Code	92	21	822	30.6
Code review	40	11	48	1.8
Compile	17	6	202	7.5
Test	58	8	500	18.6
Postmortem	24	31	219	8.2
Total	340	165	2684	100.0

Figura 25. *C63 - PSP2.1 Project Plan Summary* – modificações.

Acompanhando a Figura 25, na coluna *Actual*, estes indicadores são calculados da seguinte forma:

- *% Appraisal COQ – Actual* = 100 * (0+11) / 165 = 6.7%
- *%Failure COQ – Actual* = 100 * (6+8) / 165 = 8.5%
- *COQ A/F Ratio* = (0+11) / (6+8) = 0.79

Isto significa que o desenvolvedor gastou 6.7% do tempo total do projeto fazendo as Revisões de Código e de Projeto e gastou 8.5% compilando e testando. Como se pode observar, neste programa ele não realizou as revisões de projeto.

Esta é uma das formas que o PSP utiliza para avaliar os custos da qualidade. Na realidade, o mais correto seria dizer: para avaliar os custos da falta de qualidade. Afinal estes tempos nada mais são do que tempos utilizados na remoção de defeitos.

6.3 Exercício 4

Objetivo	Aprender a utilizar os *templates* de projeto do PSP2.1 para auxiliar na fase de projeto do programa.
Descrição	Utilizando os *templates* do PSP2.1, gerar o projeto para um programa que deverá calcular a média, o desvio padrão e a variância de uma série de números inteiros, não negativos e diferentes de zero. O usuário deverá entrar com os dados, que serão armazenados em uma lista. O usuário deverá poder escolher qual dos valores deseja calcular.
Formulários	*C66 – Operational Scenario Template* *C68 – Functional Specification Template* *C70 – State Specification Template* *C72 – Logic Specification Template*
Passo a Passo	**Planejamento (não será realizado para este exercício)** **Desenvolvimento (consistirá apenas da fase de projeto para este exercício):** ▪ Desenvolva os cenários de uso utilizando o formulário *C66 – Operational Scenario Template* ▪ Desenvolva a especificação funcional utilizando o formulário *C68 – Functional Specification Template* ▪ Desenvolva a máquina de estados para a lista que irá armazenar os valores entrados pelo usuário, utilizando o formulário *C70 – State Specification Template* ▪ Desenvolva a especificação lógica para o programa, utilizando o formulário *C72 – Logic Specification Template* **Autópsia (não será realizada para este exercício)**
Dicas	▪ Utilizar as fórmulas dos Exercícios 1 e 2.

Table C66 - Operational Scenario Template

Student ______________________ Date __________
Program ______________________ Program # __________
Instructor ______________________ Language __________

Scenario #		User Objective	
Scenario Objective			
Source	**Step**	**Action**	**Comments**

Table C66 - Operational Scenario Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Program:* nome do programa // *Program #:* número do programa (Ex.: 2A)
- *Instructor:* nome do professor // *Language:* linguagem utilizada

Cenário:

- *Scenario Number:* número de referência do cenário.
- *User Objective:* listar os objetivos do usuário para o cenário (Ex.: iniciar o sistema e selecionar o modo de operação).
- *Source:* fonte da ação (Ex.: usuário, programa, sistema.).
- *Step:* Número seqüencial do passo do cenário. É importante quando ocorrerem loops.
- *Action:* Descreve a ação que a fonte deve executar. (Ex.: entrar com o modo de seleção correto, prover mensagem de erro, etc.).
- *Comments:* Comentários relevantes para a ação (Ex.: o usuário entra com um caracter ao invés de um numérico, no modo de seleção).

Table C68 - Functional Specification Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Program:* nome do programa // *Program #:* número do programa (Ex.: 2A)
- *Instructor:* nome do professor // *Language:* linguagem utilizada

Especificação Funcional:

- *Object/Class Name:* nome do objeto/classe
- *Parent Classes:* classe da qual o objeto/classe é herdado. Utilizar toda a árvore de hierarquia, se for possível.
- *Attributes:* entre com os atributos relevantes e que impactam no comportamento do objeto.
- *Method Declaration:* declaração do objeto, incluindo: todas as variáveis e parâmetros requeridos, designações de tipos requeridas, formato para declarar o método no programa.
- *Method External Specification:* descrição da operação executada pelo método. Quando possível, utilizar notação formal ou matemática. Incluir todos os retornos e condições de exceção.

Table C68 - Functional Specification Template

Student _____ Date _____

Program _____ Program # _____

Instructor _____ Language _____

Object/class name	Parent Classes	Attributes
Method declaration	**Method external specification**	

Object/class name	Parent Classes	Attributes
Method declaration	**Method external specification**	

Object/class name	Parent Classes	Attributes
Method declaration	**Method external specification**	

Object/class name	Parent Classes	Attributes
Method declaration	**Method external specification**	

Table C70 - State Specification Template

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____
Object _____ Routine _____

State #1	Description	Attributes
State #2	**Description**	**Attributes**
State #3	**Description**	**Attributes**
State #4	**Description**	**Attributes**

Table C70 - State Specification Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Program:* nome do programa // *Program #:* número do programa
- *Instructor:* nome do professor // *Language:* linguagem utilizada

Especificação de Estados (repetir este conjunto para cada estado):

- *State #n:* nome do estado (Ex.: cheio, enchendo e vazio).
- *Description:* descrição textual para o estado.
- *Attributes:* valores das variáveis que caracterizam o estado do objeto. (Ex.: Estado cheio quando K=10 e n=3).
- *Next State:* nome do próximo estado possível. (Ex.: se estiver descrevendo o estado "Vazio", o próximo estado possível será "Enchendo").
- *Transitions Conditions:* condições sob as quais a máquina sai do estado corrente (Ex.: "Vazio") e entra no próximo estado (Ex.: "Enchendo"). Se a transição for impossível, escrever "impossível".

Table C72 - Logic Specification Template (instruções)

Cabeçalho:

- *Student:* nome do aluno // *Date:* data do início do registro
- *Program:* nome do programa // *Program #:* número do programa (Ex.: 2A)
- *Instructor:* nome do professor // *Language:* linguagem utilizada
- *Object/Function:* nome da função que está sendo especificada juntamente com o nome do objeto ao qual ela pertence.

Outras informações:

- *Includes:* todos os "includes" novos ou não usuais que são requeridos por este programa. Os "includes" padrão para o projeto não precisam ser listados separadamente. Se apenas os "includes" padrão forem utilizados, escrever a expressão "Includes padrão do projeto".
- *Type Definitions:* todas as definições de tipos não usuais ou especiais. Para poupar tempo de implementação, todas as definições de tipo devem ser especificadas.
- *Declaration:* declaração da função exatamente como ela deverá ser escrita.
- *Reference:* se houver necessidade de informações adicionais que não estejam especificadas no *design*, escrever onde elas poderão ser encontradas.

Especificação Lógica:

- *Logic Reference Numbers:* número para cada comando lógico relevante (Ex.: chamada de funções, loops, comandos condicionais, etc.).
- *Program Logic:* pseudocódigo do programa, com uma linha para cada comando relevante. Utilizar linguagem comum ou notação matemática, conforme necessário. Incluir comentários para explicar a lógica, conforme necessário.

Table C72 - Logic Specification Template

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____
Object _____ Function _____

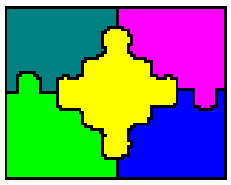
INCLUDES:

TYPE DEFINITIONS:

Declaration: _____
Reference: _____

logic reference numbers	**Program logic, in pseudocode**

7 CYCLIC PERSONAL PROCESS – PSP3



"Estratégia de Guerra: Dividir para Conquistar"

O objetivo do último nível do PSP, o PSP3 – *Cyclic Personal Process*, é o de auxiliar o desenvolvedor a planejar e acompanhar o desenvolvimento de programas maiores. Como o próprio nome sugere, o princípio básico é dividir o projeto em partes menores, às quais se possa aplicar um ciclo completo do PSP2.1.

7.1 PSP3

O processo PSP3 inicia com o planejamento e a concepção de um projeto de alto nível (*HLD - High Level Design)*. Através do HLD são identificadas partes menores que serão ciclicamente desenvolvidas utilizando o processo PSP2.1 (100 a 300 LOC).

Apesar do desenvolvimento ser feito em partes, as fases de Planejamento e Autópsia são feitas uma úncia vez para todo o projeto. A fase de Teste, bem como a Revisão de Projeto e a Revisão de Código, deverá ser o mais precisa possível em cada ciclo, para não carregar defeitos para os ciclos seguintes.

7.1.1 O registro das pendências

Em projetos maiores é comum haver questões que ficam abertas e que são deixadas de lado para serem resolvidas em outro momento. Se não aramzenadas e controladas de forma

adequada, a tendência é que sejam esquecidas e que possam resultar em prejuízos nas fases posteriores.

Com o objetivo de registrar e permitir o acompanhamento das pendências, o PSP3 oferece o formulário *C85 – Issue Tracking Log*, ilustrado na Figura 26. Nele é feito o registro da pendência, a data em que ela foi identificada, a data e a descrição da solução adotada.

Issue #:	001	Date:	11/08/2000	Phase:	Design
Description:	Domínio do campo situação do cadastro não definido.				
Resolution:	Definido pelo usuário: ativo, pendente e inativo.				
Date:	12/08/2000				

Figura 26. Registro de pendências.

7.1.2 O registro do ciclo

Para o registro dos dados acumulados dos diversos ciclos, o PSP3 utilzia um formulário específico, denominado *C82 – Cycle Summary*. Nele ficam sumarizados os dados relacionados ao tamanho dos módulos, tempo em cada fase e defeitos inseridos e removidos. Podem ser registrados tanto os dados planejados (estimados) como os dados reais (coletados após a conclusão do ciclo).

A Figura 27 ilustra uma parte do formulário *C82 – Cycle Summary*. Na parte apresentada são sumarizados os tempos por fase, por ciclo. As demais seções são similares a esta.

Cycles	**To Date**	**1**	**2**	**3**	**4**	**5**	**Total**
Time in Phase (min.)							
Design		66	47	11			124
Design Review		13	13	4			30
Code		43	43	14			100
Code Review		10	40	7			57
Compile		3	5	2			10
Test		38	22	5			65
Total		173	170	43			386

Figura 27. Registro dos ciclos do PSP3.

8 VISÃO GERAL DO TSP

O TSP - *Team Software Process* é um *framework* desenvolvido por Watts S. Humphrey para guiar o desenvolvimento e a manutenção de grandes projetos de *software*. Por destinar-se a equipes com mais de 20 pessoas e a projetos que levam até anos para serem completados, o TSP é muito complexo para ser aplicado em pequenos grupos [HUM 98]. Para tornar viável seu uso em pequenas equipes de projeto e para que pudesse ser ensinado em universidades, Humphrey desenvolveu o TSPi [HUM 99], uma versão simplificada do TSP.

O TSPi é baseado nos princípios que guiaram o desenvolvimento do PSP, consequentemente, os mesmos princípios do SW-CMM. Para aplicar o método, é necessário ter conhecimento e, de preferência, ser um praticante do PSP.

Algumas premissas que embasaram seu projeto foram:

- prover um *framework* simples, baseado nos fundamentos do PSP;
- utilizar o modelo de ciclo de vida incremental (ciclos);
- estabelecer medidas padrão para qualidade e performance;
- prover métricas precisas para equipes e estudantes;
- usar avaliações de papéis e de equipe;
- requer disciplina de processo;
- prover orientações para solução de problemas com equipes.

Por ter sido descrito principalmente para servir como um livro de texto em universidades, *Introduction to the Team Software Process* [HUM 99] faz diversas referências ao papel do instrutor. O instrutor resolve toda a sorte de problemas: desde decidir sobre requisitos do projeto até solucionar questões relativas ao desempenho de determinado membro da equipe. Informações adicionais seriam necessárias para a aplicação do TSP em escala industrial, conforme citado pelo próprio autor.

O modelo de ciclo de vida usado pelo TSPi é o incremental ou cíclico. Ou seja, o produto vai sendo desenvolvido em partes, através de ciclos completos, conforme a Figura 28. Ao final de cada ciclo, um produto testável é produzido.

Cada ciclo é composto por 7 fases: Lançamento, Estratégia, Planejamento, Requisitos, Projeto, Implementação, Teste e Autópsia. A saída de um ciclo representa a entrada para o ciclo seguinte, até que o produto chegue ao ciclo final, no qual é avaliado e entregue.

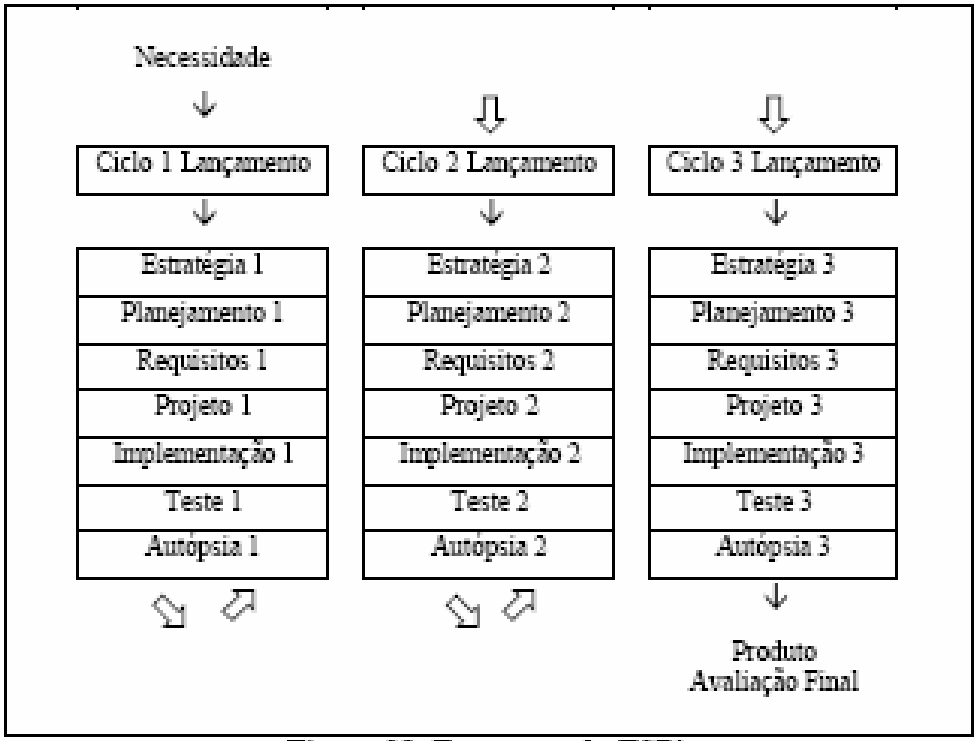


Figura 28. Estrutura do TSPi.

8.1 Conceitos fundamentais do TSPi

8.1.1 Roteiros

O TSPi, a exemplo do PSP, está totalmente estruturado em forma de roteiros. Cada um dos 21 roteiros que compõem o TSPi é apresentado como um conjunto estruturado de: propósito, critérios de entrada, descrição geral, atividades a serem desenvolvidas e critérios de saída.

O primeiro roteiro, denotado pela sigla DEV, descreve sucintamente os sete passos do desenvolvimento cíclico. Os demais, descrevem cada um destes sete passos detalhadamente para o primeiro ciclo (LAU1, STRA1, etc.) e para os demais ciclos (LAUn, STRAn, etc.).

O roteiro, além de funcionar como um guia para o desempenho das atividades do projeto, funciona também como um *checklist* para as entradas e saídas do processo.

8.1.2 Formulários

Assim como o PSP, o TSPi também funciona baseado em formulários. São ao todo 21 formulários que auxiliam o registro de informações e o desenvolvimento das atividades. Por exemplo, o formulário de registro de tempos (LOGT) é o mesmo *C16 - Time Recording Log* do PSP, o formulário de registro de defeitos (LOGD) é o mesmo *C18 – Defect Recording Log* do PSP.

8.1.3 Papéis

Para o bom andamento das atividades do trabalho em equipe, o TSPi estabelece cinco papéis a serem desempenhados pelos membros:

- Líder de Projeto: lidera a equipe e assegura que os engenheiros reportam seus dados de processo e completam seu trabalho conforme o planejado ;
- Gerente de Planejamento: apoia e guia os membros da equipe no planejamento e acompanhamento do projeto;
- Gerente de Desenvolvimento: lidera e guia a equipe na definição, projeto, desenvolvimento e teste do produto;
- Gerente de Processo/Qualidade: apoia a equipe na definição das necessidades do processo, na definição do plano de qualidade, e no acompanhamento da qualidade do processo e do produto;
- Gerente de Suporte: apoia a equipe na determinação, obtenção e gerenciamento das ferramentas necessárias para atingir as necessidades da equipe em termos de tecnologia e apoio administrativo.

Cada papel é descrito através de dois tipos de roteiros. Um deles descreve o papel de forma geral, contendo: objetivo, características, metas/métricas e principais atividades. O outro, descreve em detalhes cada uma das atividades que deverão ser desempenhadas semanalmente pelo papel: fase/semana, descrição geral da atividade, descrição detalhada da atividade e referência (capítulo do livro no qual a atividade é descrita).

Ao final do projeto, não apenas os membros do time terão sido avaliados segundo os critérios estabelecidos, como também os papéis por eles desempenhados.

8.1.4 Padrões

O TSPi também estabelece três tipos de padrão para orientar o trabalho das equipes de projeto:

- padrão de tipos de defeito (originário do PSP – *Defect Type Standard*);
- padrão para composição das anotações do projeto (*Project Notebook);*
- padrão para estabelecimento de metas e medidas de qualidade para o projeto.

8.1.5 Material de Apoio

Uma das principais críticas ao modelo PSP é a quantidade de informações que o engenheiro de software tem que registrar e a ausência de uma ferramenta automatizada para realizar as atividades de cálculo e análise, gerando distorções na interpretação dos resultados. Isto foi resolvido no TSPi com a disponibilização de uma ferramenta desenvolvida em Excel, por James W. Over, do SEI. A versão beta pode ser obtida no site da editora Adisson-Wesley.

A ferramenta de apoio auxilia a equipe nas atividades de planejamento, entrada de dados e acompanhamento. Ao usá-la para registrar os dados das atividades, os formulários de sumário serão automaticamente preenchidos, de forma completa e precisa. Além disto, o guia de instruções da ferramenta funciona também como um guia para o processo em si.

Além da ferramenta, também é possível obter no site da editora um guia para o instrutor e uma cópia eletrônica dos formulários do modelo.

8.2 Fases do Ciclo

Conforme citado anteriormente e ilustrado na Figura 28, cada ciclo de desenvolvimento está estruturado em 7 fases: Lançamento, Estratégia, Planejamento, Requisitos, Projeto, Implementação, Testes e Autópsia.

8.2.1 Lançamento

O objetivo da fase de lançamento do projeto é estabelecer as bases para o trabalho em equipe. Isto significa descrever o produto que será desenvolvido, distribuir papéis, estabelecer metas para os papéis e para os membros da equipe, estabelecer datas para encontros semanais e datas para a entrega de relatórios semanais.

Devido ao fato de que a maioria das equipes provavelmente não dispõe de dados históricos, Humphrey propõe em [HUM 99] números para os objetivos. Por exemplo: percentual de defeitos encontrados antes da primeira compilação: 80%.

8.2.2 Estratégia

Nesta fase é estabelecida a estratégia para o desenvolvimento do produto. O TSPi prevê o uso do desenvolvimento incremental, porém cabe à equipe definir o que será desenvolvido em cada um dos ciclos. A idéia é que no primeiro ciclo seja desenvolvido um produto testável, porém sem todas as funcionalidades requeridas. Nos ciclos seguintes, o produto vai sendo melhorado para contemplar todos os requisitos.

Nesta fase é feito um projeto conceitual do produto e são estimados o tamanho e o tempo de desenvolvimento. Também é estabelecido um plano para o gerenciamento de configuração.

Embora haja pouca informação detalhada nesta etapa, o tamanho das funções deve ser estimado em LOC (linhas de código) e o tempo, utilizando a produtividade em termos de

LOC/hora. O autor sugere que se use a experiência adquirida com o uso do PSP para produzir estas estimativas. Porém, se os desenvolvedores não forem usuários do PSP, esta informação, pelo menos no primeiro momento, terá que ser "chutada".

8.2.3 Planejamento

O objetivo desta etapa é o de produzir um planejamento das tarefas individuais e da equipe, o cronograma e o plano de qualidade.

Na fase de Estratégia foi produzido o projeto conceitual do produto e o tamanho das funções foi estimado em termos LOC. Na fase de Planejamento devem ser previstos também os outros tipos de produtos e seus tamanhos: documento de requisitos (páginas), projeto de alto nível – HLD (páginas), projeto detalhado – DLD (linhas), documentação do usuário (páginas), materiais de teste, etc.

Para estabelecer o plano de qualidade é fornecido um padrão bastante objetivo de critérios de qualidade que deve, em princípio, ser seguido. Por exemplo: o total de defeitos injetados/LOC deve estar entre 75 e 150.

8.2.4 Requisitos

Nesta etapa é produzido o documento de requisitos (SRS – *Software Requirements Statement*). O SRS deve estabelecer claramente o que deve fazer o produto e também os critérios segundo os quais se poderá avaliar, ao final do projeto, se o produto efetivamente faz o que deveria fazer. É nesta etapa também que será produzido o plano de testes.

8.2.5 Projeto

O método usa a denominação de projeto para todas as etapas tradicionalmente conhecidas como análise e projeto. A idéia é, a partir do HLD dividir as tarefas de detalhamento entre os membros da equipe e então combiná-las ao final. No TSPi são usados os quatro *templates* definidos no PSP2.1: *operational scenarios, functional specification, state specification* e *logic specification.*

8.2.6 Implementação

A implementação consiste não apenas da codificação em si, mas também do planejamento da implementação, detalhamento do projeto, revisão e inspeção do projeto detalhado, inspeção de código, teste unitário, revisão da qualidade do componente e, finalmente, entrega do componente.

O autor desencoraja o uso de outros tipos de métricas além de LOC e quantidade de páginas, ao menos para a realização dos projetos previstos pelo livro. Há algumas considerações superficiais sobre como usar estimativas baseadas em *proxies* para telas ou outros elementos.

8.2.7 Teste

O propósito dos testes no TSPi não é o de corrigir o produto, mas sim de verificar a sua qualidade. Nesta fase são desenvolvidos os planos de teste, é feita a montagem do produto final com base nos componentes desenvolvidos, são executados os testes de integração das partes e os testes de sistema e também é elaborada e revisada a documentação do usuário.

Nos ciclos seguintes ao primeiro, são feitos também os testes de regressão para certificar-se de que funcionalidades testadas em ciclos anteriores permanecem funcionando.

8.2.8 Autópsia

O propósito da fase Autópsia ao final de cada ciclo, é o de analisar os resultados do ciclo e identificar as oportunidades de melhoria do processo. O autor sugere que se foque pequenas, porém contínuas, melhorias no processo de produção de *software* em equipe. A base para estas melhorias são os registros que o engenheiro de *software* vai fazendo no PIP - *Process Improvement Proposal* (o mesmo doPSP).

Nesta fase também são conduzidas as avaliações dos engenheiros de *software* e dos papéis por eles desempenhados.

ANEXO A – FORMULÁRIOS DO PSP

Baseline Personal Process	PSP0	C14	PSP0 Project Plan Summary
		C16	Time Recording Log
		C18	Defect Recording Log
	PSP0.1	C25	PSP0.1 Project Plan Summary
		C27	PIP – Process Improvement Proposal
Personal Planning Management	PSP1	C34	PSP1 Project Plan Summary
		C37	Test Report Template
		C39	Size Estimating Template
	PSP1.1	C45	PSP1.1 Project Plan Summary
		C47	Task Planning Template
		C49	Schedule Planning Template
Personal Quality Management	PSP2	C55	PSP2 Project Plan Summary
		C57	C++ PSP2 Design Review Checklist
		C58	C++ Code Review Checklist
	PSP2.1	C63	PSP2.1 Project Plan Summary
		C65	PSP2.1 Design Review Checklist
		C66	Operational Scenario Template
		C68	Functional Specification Template
		C70	State Specification Template
		C72	Logic Specification Template
Cyclic Personal Process	PSP3	C80	PSP3 Project Plan Summary
		C82	Cycle Summary
		C84	PSP3 Design Review Checklist
		C85	Issue Tracking Log