Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

DENIS MENDONÇA LADEIRA

O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE FOCADO NA QUALIDADE PESSOAL

Assis
2012

DENIS MENDONÇA LADEIRA

O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE FOCADO NA QUALIDADE PESSOAL

Projeto apresentado ao Programa de Iniciação Científica (PIC) ao Instituto Municipal de Ensino Superior de Assis – IMESA e à Fundação Educacional do Município de Assis – FEMA.
Orientando: Denis Mendonça Ladeira
Orientador: Luiz Ricardo Begosso
Linha de Pesquisa: Ciências Exatas e da Terra

Assis
2012

FICHA CATALOGRÁFICA

LADEIRA, Denis Mendonça
O Processo de Desenvolvimento de Software Focado na Qualidade Pessoal / Denis Mendonça Ladeira. Fundação Educacional do Município de Assis – FEMA – Assis.

191p.

Orientador: Luiz Ricardo Begosso.

Programa de Iniciação Cientifica (PIC) – Instituto Municipal de Ensino Superior de Assis – IMESA

1. Qualidade de Software. 2. PSP

CDD: 001.61
Biblioteca da FEMA

RESUMO

Qualidade é um dos assuntos mais importantes na indústria de software da atualidade, e provavelmente será ainda mais importante num futuro próximo, pois há cada vez mais sistemas controlados por software, fazendo com que a economia de praticamente todos os países seja muito dependente da qualidade dos softwares por eles utilizados.

As aplicações de software aumentam em tamanho em uma ordem de magnitude a cada 5-10 anos, e assim ocorre também com o número de “bugs”. Numerosos métodos surgiram para resolver o problema de defeitos de software. O PSP (Processo de Software Pessoal) (HUMPHREY 95), um processo disciplinado de auto-melhoria, é um dos mais proeminentes. O Processo de Software Pessoal é um processo de desenvolvimento de software que permite ao indivíduo aplicar a disciplina de nível industrial na sua prática pessoal. O PSP provê aos engenheiros de software um modo para melhorar a qualidade, a previsibilidade e a produtividade do seu trabalho. É projetado para resolver as necessidades de melhoria de engenheiros individuais e de organizações de software pequenas.

O presente trabalho conclui que a filosofia de PSP é pertinente para o desenvolvedor de software e pode melhorar a qualidade de software por ele produzido. Porém, modos de reduzir os custos da implementação do PSP são necessários antes do método ser adotado amplamente. O autor do presente trabalho acredita que o custo para executar o PSP pode ser reduzido provendo apoio automatizado para as atividades do método. Também acredita que o PSP pode se tornar uma base para desenvolver um processo de software pessoal customizado.

Palavras-chave: qualidade, defeito, processo, engenharia de software, Processo Pessoal de Software, método, projeto.

ABSTRACT

Quality is one of the most important issues in the software industry today, and will probably be even more important in the near future, as more and more systems controlled by software, making the economy of virtually all countries are very dependent on the quality of software they use.

Software applications increase in size with an order of magnitude every 5-10 years, and so does the number of "bugs". Numerous methods have arisen to address the problem of software defects. The PSP (Personal Software Process) (HUMPHREY 95), a disciplined self-improvement process, is one of the most prominent. The Personal Software Process is a software development process that allows the individual to apply industrial level discipline to his or her practice. The PSP provides software engineers a way to improve the quality, predictability, and productivity of their work. It is designed to address the improvement needs of individual engineers and small software organizations.

This text concludes that the philosophy of PSP is relevant for software developers, and can improve software quality. However, ways of reducing the cost of implementing PSP are necessary before the method is likely to be widely adopted. It is believed that the cost to perform PSP can be reduced through providing automated support for PSP activities. It is also believed that the PSP can become a basis for developing a custom personal software process.

Key-words: quality, defect, process, software engineering, Personal Software Process, method, project.

LISTRA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

O processo de engenharia de software de uma organização é altamente influenciado pelas ferramentas adotadas, regras de negócio, estrutura organizacional, métodos de gerenciamento, políticas estruturais, ambientes de trabalho, formas de comunicação, interações sociais, e pessoas.

As organizações de desenvolvimento de software possuem uma série de mecanismos para auxiliá-las no gerenciamento e melhoria da qualidade dos seus produtos. Os aspectos relacionados à garantia da qualidade de software têm sido muito ressaltados nos últimos anos, especialmente pelo fato das empresas estarem preocupadas com a qualidade em todos os seus segmentos.

Para se alcançar a qualidade dos produtos de software é fundamental que o processo de desenvolvimento de software seja construído com parâmetros qualitativos. Para as grandes organizações, torna-se razoável a adoção de modelos de gestão com conceitos da Qualidade Total que reflitam na garantia da qualidade, tais como a metodologia Kaizen, 5S, Seis Sigma, ISO, CMMI, entre outros. Entretanto, para as pequenas empresas ou organizações individuais de desenvolvimento de software, a adoção de tais práticas é extremamente difícil.

A falta de adoção de práticas de qualidade pelas pequenas organizações de desenvolvimento de software contribui para a alta taxa de mortalidade destas empresas, já que parte de seus escassos recursos são desperdiçados por falta de planejamento adequado. Além disso, estima-se que uma quantidade elevada dos serviços realizados pode ser definida como retrabalho, comprometendo os custos e os prazos estabelecidos.

Uma das soluções propostas para a melhoria da qualidade do software produzido pelas pequenas empresas ou organizações individuais relaciona-se com a adoção formal de um processo de desenvolvimento de software. O PSP – Personal Software Process (HUMPHREY, 2005) é uma alternativa que tem se mostrado bastante eficiente neste sentido, sendo que sua adoção incorpora benefícios aos desenvolvedores individuais ou às pequenas equipes de desenvolvimento de software.

O presente trabalho está situado no contexto da Engenharia de Software, com o foco na qualidade no desenvolvimento dos produtos de software.

1.1. PROBLEMATIZAÇÃO

Segundo a atual norma brasileira sobre Sistemas de Gestão e Garantia da Qualidade (ABNT NBR ISO 9000:2005), o termo qualidade pode ser definido como: "*A totalidade das características de uma entidade que lhe confere a capacidade de satisfazer as necessidades explícitas e implícitas*". Nesta definição, o termo "entidade" refere-se a um serviço. O termo "necessidades explícitas" refere-se às próprias condições e objetivos propostos pelo produtor. E o termo "necessidades implícitas" inclui as diferenças entre os usuários, a evolução no tempo, as implicações éticas, as questões de segurança e outras visões subjetivas.

De acordo com PRESSMAN (2011), a qualidade na Engenharia de Software é composta por três aspectos: qualidade do produto, qualidade do processo de desenvolvimento e qualidade da equipe de desenvolvimento.

A qualidade de software consiste num conjunto mínimo de requisitos exigidos do software, que correspondem a fatores internos e externos. Com os requisitos definidos, o desenvolvedor de software conhece quantitativamente o nível de qualidade que se deseja alcançar no software e precisa estruturar métodos e técnicas para garantir e controlar esta qualidade desejada.

Diversos autores destacam que a qualidade de software está diretamente associada à qualidade do processo utilizado para seu desenvolvimento (PRESSMAN, 2011; SOMMERVILLE, 2007). Os processos de software não são fixos nem rotineiros, podem variar de organização para organização e até mesmo de projeto para projeto dentro de uma mesma organização. Por esta razão não é fácil defini-los. Se tal dificuldade é clara para uma organização de porte maior, sabe-se que ela é ainda mais preocupante para pequenas equipes ou profissionais individuais.

Uma vez definido o processo, há a necessidade de garantir que ele seja seguido e executado por toda a organização. Para isso é necessário disciplina da equipe de desenvolvimento para

seguir os padrões requeridos em cada fase, na supervisão e no acompanhamento do processo. Somente assim a qualidade de todo processo de software pode ser garantida.

Dentre os vários padrões de qualidade estabelecidos, o Software Engineering Institute (SEI) definiu o Personal Software Process (PSP), centrado no processo pessoal de trabalho do desenvolvedor de software, também com o objetivo de estabelecer padrões e disciplinas ao mesmo.

1.2. OBJETIVOS

Este trabalho tem o objetivo genérico de identificar os benefícios obtidos com a adoção de um processo pessoal de desenvolvimento de software para a melhoria da qualidade dos produtos de software desenvolvidos por pequenas equipes ou profissionais individuais.

Para se alcançar este objetivo genérico, os objetivos específicos deste projeto são: realizar um estudo sobre o Processo Pessoal de Desenvolvimento de Software PSP e, em seguida, aplicá-lo no desenvolvimento de um projeto piloto para avaliação dos resultados. Para isto, este trabalho será dividido em três etapas: revisão bibliográfica, modelagem da aplicação piloto e desenvolvimento da aplicação.

1.3. RELEVÂNCIA

Os aspectos sobre qualidade de software têm sido altamente valorizados pela indústria de desenvolvimento de software. O número de pequenas empresas de desenvolvimento de software é muito grande e justamente por isso torna-se fundamental o estudo de práticas de qualidade voltadas para este público alvo.

1.4. REVISÃO DA LITERATURA

O modelo PSP foi criado por HUMPHREY (2005) como um processo de qualidade para auxiliar no controle, gerenciamento e melhoria da forma pessoal de desenvolvimento de software, possibilitando a apropriação de dados históricos para o atendimento de requisitos e tornando os elementos de rotina do trabalho mais previsíveis e eficientes.

Assim, o PSP pode ser usado nos processos de planejamento, previsão (especialmente de prazos, custos e recursos), e medição da qualidade dos produtos de software. Conforme HILBURN; TOWHIDNEJAD (2002), o PSP é especialmente destinado aos engenheiros de software que desenvolvem trabalhos individuais ou em pequenas equipes, permitindo-lhes o desenvolvimento de processos pessoais disciplinados e bem definidos, que incluam planejamento realista e efetivo, e que enfatize uma metodologia de qualidade para a programação.

A filosofia pela qual o PSP foi desenvolvido está baseada no fato de que a habilidade de uma organização para construir sistemas de software em larga escala é altamente dependente da habilidade de seus engenheiros de software desenvolver programas em pequena escala, através de uma maneira disciplinada e efetiva. Ele auxilia os engenheiros a organizarem e planejarem seus trabalhos, acompanharem seus desempenhos, gerenciarem os defeitos do software, analisarem e melhorarem seus próprios processos pessoais de desenvolvimento de software.

O PSP é estruturado em quatro fases de melhoria do processo, divididas em sete níveis, e a evolução é feita de maneira incremental, com os níveis superiores adicionando características aos níveis já implantados, minimizando assim, o impacto na mudança do processo pessoal e proporcionando o amadurecimento do engenheiro de software no modelo.

O PSP é um modelo que pode ser aplicado em qualquer estrutura organizacional para a melhoria da qualidade pessoal de desenvolvimento, tornando-se fundamental para o sucesso de pequenos projetos e de grande auxílio para projetos maiores.

Pelo fato de ser voltado a um processo individual, o PSP torna-se bastante adequado para ser aplicado no ensino de graduação, tendo o objetivo de criar uma base sólida de habilidades para a formação teórica e prática de pequenas equipes. Existem várias aplicações

educacionais bem sucedidas (RONG, G. et al, 2011) que demonstram a aplicação do modelo junto ao ambiente de ensino.

1.5. METODOLOGIA

As metas a serem alcançadas durante o período de execução desse trabalho científico são:

- Realizar uma revisão bibliográfica sobre Qualidade de Software;
- Identificar as características e benefícios da adoção do PSP – Personal Software Process;
- Desenvolver um projeto prático aplicando o PSP durante a modelagem e o desenvolvimento de uma aplicação piloto;
- Avaliar os resultados obtidos.

1.6. ESTRUTURA DO TRABALHO

O Capítulo 1 introduz a necessidade que o mercado tem por produtos de software de qualidade, os objetivos e metas à serem alcançadas no término do trabalho com a revisão Literária sobre o modelo de desenvolvimento de software estudado.

O Capítulo 2 é feito uma Revisão bibliográfica sobre Qualidade de Software, Tratando-se primeiramente da conceituação do termo Qualidade que independe da área de desenvolvimento de Software; após isto, é feito a conceituação sobre a Engenharia de Software, descrevendo as principais características do processo de Engenharia de Software nas ultimas décadas; subseguindo é debatido o termo ‘Qualidade de Software’, suas principais interpretações partindo de grandes estudiosos, e como o termo é comumente associado na Sociedade, sobre os atributos fundamentais em termos de produto e de Software levando em consideração o ponto de vista de usuário; ao final é citada a economia relacionada à Qualidade de Software, algumas responsabilidades em questão do Gerenciamento de Qualidade, e uma breve introdução ao termo Gestão de Qualidade Total (TQM).

O Capítulo 3 introduz o PSP (Processo de Desenvolvimento de Software Pessoal), processo no qual é estudado neste trabalho; comenta-se sobre a importância de desenvolvimento de Software utilizando um modelo de Qualidade para pequenas Empresas e Engenheiros que trabalham individualmente ou em pequenas equipes; é descrito algumas características de um processo de desenvolvimento de Software e apresentadas algumas ressalvas e dicas para que irá iniciar um estudo sobre o PSP.

O Capitulo 4 faz-se uma apresentação metodológica de todo o PSP, com todos os seus níveis apresentados nas seções do capitulo; este conteúdo será apresentado de maneira técnica com intuito de auxiliar o estudo e pratica do processo.

O Capítulo 5 é realizado a conclusão do trabalho; apresentando os custos do PSP, não precisamente ao monetário, mas principalmente quanto ao tempo requerido para estudo do processo; seus benefícios para um indivíduo ou para uma organização; é levantado o questionamento sobre a automação do processo; estudos de impacto sobre o PSP, comentando seus resultados; e expectativas de futuros projetos.

O Apêndice A contem todo o conteúdo de tabelas, formulários e scripts utilizados no Processo de Software Pessoal.

2. QUALIDADE DE SOFTWARE

Este capítulo tem por objetivo apresentar o embasamento teórico obtido através da pesquisa bibliográfica realizada, constituindo um registro de alguns assuntos que foram considerados relevantes, e que são apresentados nesta ordem: I) Conceituação de Qualidade, onde são apresentadas a origem e as definições de Qualidade conforme a visão de diversos especialistas da área. II) Conceituação de Engenharia de Software, onde é apresentado um breve discurso de como se caracterizava o Desenvolvimento de Software ao longo da segunda metade do século XX. III) Qualidade de Software, Onde é apresentada a importância do assunto, os critérios de um bom produto, e o que o Software precisa ter como características fundamentais para ser considerado um produto de qualidade dadas as mais relevantes sobre o ponto de vista de diferentes tipos de clientes. Também é enfatizada a questão Econômica no desenvolvimento de Software, o quanto é importante à realização de testes no decorrer do período de desenvolvimento. E por ultimo o Gerenciamento de Qualidade sobre a responsabilidade dos Gerentes e Engenheiros de Software em uma Instituição. IV) TQM, é feita uma introdução ao termo (TQM) Gestão da Qualidade Total que consiste em uma estratégia de administração orientada a criar consciência da qualidade em todos os processos organizacionais, são apresentados algumas empresas que adotam esta filosofia e seus pontos-chaves na satisfação de seus colaboradores.

2.1. CONCEITUAÇÃO DE QUALIDADE

É importante antes da de abordar a Qualidade de Software que é o enfoque deste capitulo, fazer considerações sobre a sua origem, que tem como base o desenvolvimento e a produção de produtos e bens de qualidade na indústria de manufatura, cujos princípios estão descritos a seguir.

O conceito de produzir qualidade iniciou-se no Japão na década de 1950, e foi encarado pelas organizações ocidentais como um fator competitivo importante somente 30 anos depois, quando especialistas passaram a dar consultoria a estas empresas sobre métodos para a melhora a qualidade de seus produtos.

De acordo com BEGOSSO (2002), qualidade pode ser definida como “propriedade, atributo ou condição das coisas ou das pessoas capaz de distingui-las das outras e de lhes determinar a natureza”; “permite avaliar e, consequentemente, aprovar, aceitar ou recusar qualquer coisa”.

Entretanto para cada consumidor, pode - se encontrar definições diferentes para um produto de qualidade; alguns podem considerar apenas características como facilidade de uso ou aquisição, custo final, livre de defeitos, que atenda completamente as suas necessidades, que seja seguro quando este fator é critico ao produto desenvolvido, ou mesmo uma combinação entre essas e outras características.

Vários especialistas têm dedicado muitos anos de pesquisa sobre a melhoria da qualidade e, embora não sejam específicos da área de qualidade de software, estes estudos tem potencial em ser usados para a melhoria de qualidade de qualquer produto. Os parágrafos seguintes mostram as definições de qualidade, de acordo com a visão de alguns desses teóricos, reconhecidos mundialmente como grandes colaboradores para o processo de desenvolvimento e adoção da qualidade nas organizações modernas.

Para melhorar a qualidade, BEGOSSO (2002) defende a adoção dos seis C’s: compreensão, comprometimento, competência, comunicação, correção e continuidade. Ele define qualidade como "em conformidade com os requisitos", e tem a opinião de que "a qualidade é gratuita", argumentando que enquanto aumentam os custos voluntários da prevenção de defeitos, os custos involuntários do re-trabalho diminuem numa proporção muito maior, resultando em menor custo total.

O processo de controle da qualidade está todo baseado na educação e treinamento das pessoas envolvidas, começando e terminando com o ensino. Para introduzir a cultura da garantia da qualidade, ele sugere treinamento para todos os níveis da organização, ensino de longo prazo sobre o controle da qualidade, e ensino e treinamento permanente.

BEGOSSO (2002) declara que a qualidade tem um grau previsível de uniformidade e confiança a baixo custo, e deve visar o atendimento das necessidades do cliente. Ele diz que a melhoria da qualidade é de responsabilidade da alta administração da organização, e quando implantada, pode alcançar a redução de custos, o aumento da produtividade e da fatia de mercado ocupada pelo produto, além de gerar mais empregos. BEGOSSO (2002) identifica três processos para se alcançar o gerenciamento da qualidade, conhecidos por Trilogia de

Juran: (I) planejamento da qualidade: relaciona-se com a identificação dos clientes e suas necessidades, criação de um produto que atenda a estas necessidades, e o desenvolvimento de um processo para produzir o produto; (II) controle da qualidade: relaciona-se com a manutenção do estado atual e não deixar o processo piorar; e (III) melhoria da qualidade: relaciona-se com a melhoria dos processos de produção e a redução dos custos para aumentar a qualidade. A definição de qualidade baseada na teoria do TQM – Gerenciamento da Qualidade Total é feita dividindo a evolução do termo em quatro adequações: ao padrão (típica na década de 1950), ao uso (típica na década de1960), ao custo (típica na década de 1970), e à necessidade latente (a partir da década de1980), sendo que esta última enfoca a satisfação das necessidades dos clientes antes deles estarem conscientes delas. Neste último caso, quando uma organização consegue descobrir tal necessidade, ela pode ter o monopólio por algum período de tempo e ser bastante lucrativa. Para isso, os autores descrevem sete ferramentas do planejamento da qualidade que ajudam a identificar e traduzir as necessidades latentes em planos para produtos e processos de produção. Nesta abordagem, o cliente passa a ser o centro das atenções, e informações sobre seu estilo de vida e maneiras de melhorá-lo devem atingir toda a organização.

Pode-se dizer que a década de 1990 foi caracterizada como a era da qualidade, devido a grande ênfase dada a este assunto, especialmente sob a perspectiva da qualidade dos produtos de software, assunto que será discutido nas seções seguintes.

2.2. CONCEITUAÇÃO DE ENGENHARIA DE SOFTWARE

Olhando para a Engenharia de Software de uma perspectiva histórica, a década de 60 e anos anteriores podem ser vistos como uma era funcional, a década de 70 a era de programação, os anos 80 a era dos custos, a década de 90 a era da qualidade além da eficiência. Nos anos 60 aprendemos como explorar a tecnologia da informação e conhecer necessidades institucionais, começamos a vincular o software com as operações diárias das instituições. Nos anos 70, como as indústrias eram caracterizadas pelo massivo agendamento de atrasos e custos excessivos, o foco estava em planejar e controlar os projetos de software. Fases baseadas em modelos de ciclo de vida foram introduzidas, e analisadas, como o mítico homem-mês, surgiu. Nos anos 80 os custos de hardware continuavam a cair, e a tecnologia da

informação permeou a todos os aspectos das instituições que se tornaram disponíveis para os indivíduos. Como a concorrência nas indústrias tornaram-se fortes e as aplicações de baixo custo tornaram-se amplamente implementadas, a importância da produtividade no desenvolvimento de software aumentou significativamente. Vários modelos de custos de engenharia de software foram desenvolvido e utilizado. Na década de 1980, a importância da qualidade também foi reconhecida.

Os anos de 1990 e para além deles é certamente a era de qualidade. Com o estado-da-arte da tecnologia agora capaz de fornecer uma funcionalidade abundante, os clientes exigem alta qualidade. Exigência de qualidade é ainda mais intensificada pela dependência cada vez maior da sociedade sobre software. Erros de faturamento em grande escala de serviços telefônicos interrompidos, e até mesmo falhas de mísseis durante as guerras recentes podem ser rastreados para a questão da qualidade de software. Nesta época, a qualidade tem sido trazido para o centro do processo de desenvolvimento de software. Do ponto de vista dos fornecedores de software, a qualidade não é mais um fator de vantagem no mercado, tornou-se uma condição necessária, se uma empresa é competir com sucesso.

2.3. O QUE É QUALIDADE DE SOFTWARE?

A qualidade deve ser definida e medida se a melhoria é está a ser alcançada. No entanto, um grande problema em engenharia e gestão da qualidade é que o termo qualidade é ambíguo, de tal forma que é comumente mal interpretado. A confusão pode ser atribuída a várias razões. Primeiro, a qualidade não é uma idéia única, mas sim um conceito multidimensional. As dimensões da qualidade incluem a entidade de interesse, o ponto de vista sobre essa entidade, e os atributos de qualidade da entidade. Em segundo lugar, por qualquer conceito, existem níveis de abstração, quando as pessoas falam sobre a qualidade, uma parte poderia estar se referindo a ela em seu sentido mais amplo, enquanto outra pode estar se referindo ao seu significado específico. Em terceiro lugar, o termo qualidade é uma parte da nossa linguagem cotidiana e os usos populares e profissionais podem ser muito diferentes.

Atingir um alto nível de qualidade de produto ou serviço é o objetivo da maioria das organizações. Atualmente não é mais aceitável entregar produtos com baixa qualidade e reparar os problemas e as deficiências depois que o produto já foi entregue ao cliente.

O principal foco de qualquer definição de Qualidade de Software deveria ser o que os usuários necessitam. Crosby define qualidade como “o atendimento aos requisitos”. (HUMPHREY, 1995) Embora se possa debater a distinção entre requisitos, necessidades, e desejos, definições de qualidade devem considerar as perspectivas do usuário. As questões chaves são então, quem são os usuários, o que é importante para eles, e como as suas prioridades se relacionam com o modo como você constrói, desenvolve, e suporta os seus produtos?

2.3.1. Qualidade do Produto.

Para responder estas questões, é necessário reconhecer a natureza hierárquica da Qualidade de Software. Primeiramente, um produto de software deve fornecer funções de um tipo e no momento quando o usuário necessita delas. Se isso não acontecer, nada mais importa. Segundo, o produto deve funcionar. Se o sistema tem tantos defeitos que faz com que o produto não execute com uma consistencia razoável, os usuarios não irão usa-lô independentemente de seus outros atributos. Isso não significa que defeitos terão sempre a maior prioridade, mas podem ser de grande importância. Um minimo nível de defeitos deve ser corrigido. Para além deste princípio de qualidade, contudo, a importância dos defeitos, bem como de usabilidade, compatibilidade, funcionabilidade, e todos os outros “lidades”, dependem do tipo de usuários que irão utiliza-lôs, da aplicação, e do ambiente.

Em um amplo sentido, a qualidade do ponto de vista dos usuários deve lidar com a facilidade de instalação do produto, eficiência operacional, e ser convênient. Irá rodar no sistema no qual se destina, irá executar as aplicações, irá lidar eficientemente com os arquivos exigidos ? O produto é conveniente, o usuário pode lembrar como usa-lô, e pode facilmente descobrir o que eles ainda não conhecem? O produto é suscetível, pode surpreender o usuário, e protege-lôs de si mesmos e dos outros e pode isola-lôs da mecânica do funcionamento do sistema? Estas e uma série de perguntas semelhantes são importantes para os usuários. Enquanto as prioridades variam entre os usuários, a qualidade tem muitas camadas, e nenhuma definição universal será aplicada em cada caso. Se o seu software não está à altura em qualquer área que é importante para seus usuários, eles não vão julgar o seu produto como de alta qualidade.

Alguns poucos Engenheiros de Software irão debater esses pontos, pois suas ações não são coerentes com as prioridades. Ao invés de dedicar partes principais de seu processo de desenvolvimento para a instalabilidade, usabilidade e eficiência operacional do sistema, optam por gastá-los em testes, o elemento de maior custo na organização do software. Além disso, esses custos de testes são quase exclusivamente dedicados a encontrar e corrigir defeitos.

2.3.2. Qualidade de Software

Em software, o sentido mais restrito da qualidade do produto é comumente reconhecido com a falta de "bugs" no produto. É também o sentido mais básico de conformidade com os requisitos, porque se o software contém muitos defeitos funcionais, o requisito básico de fornecer a função desejada não é cumprido. Esta definição é normalmente expressa em duas formas: taxa de defeito (por exemplo, número de defeitos por milhão de linhas de código-fonte, por ponto de função, ou outra unidade) e confiabilidade (por exemplo, número de falhas por n horas de operação, tempo médio de falha , ou a probabilidade de operação livre de falha em um período de tempo especificado). A satisfação do cliente é geralmente medido por cento satisfeito ou insatisfeitos (neutro e insatisfeito) a partir de inquéritos de satisfação do cliente. Para reduzir o viés, técnicas tais como pesquisas de cegos (o entrevistador não conhece o cliente e o cliente não conhece o qual empresa representa o entrevistador) são normalmente utilizados. Além de satisfação geral do cliente com o produto de software, a satisfação para atributos específicos também são avaliada.

A qualidade de software deve envolver desde os aspectos da qualidade dos produtos de software até os aspectos da qualidade do processo de desenvolvimento desses produtos. Diversas normas de qualidade têm sido elaboradas para especificar e avaliar a qualidade dos produtos de software, entre elas a ISO/IEC 9126 (HUMPHREY, 1995), que é utilizada como referência para o processo de avaliação da qualidade de produtos de software. Tal norma define as seis principais características de um software, que podem ser utilizadas como critérios para a qualidade:

- Funcionalidade: refere-se a um conjunto de atributos que relacionam-se com a existência de funções que satisfazem as necessidades do usuário, tais como adequação, acurácia, interoperabilidade, segurança de acesso e conformidade.

- Confiabilidade: refere-se a um conjunto de atributos que relacionam-se com a capacidade do software manter seu nível de desempenho, sob condições estabelecidas, por um período de tempo, tais como maturidade, tolerância a falhas,capacidade de recuperação e conformidade.

- Usabilidade: refere-se a um conjunto de atributos que relacionam-se com o esforço necessário para utilizar o software, tais como inteligibilidade, apreensibilidade, operacionalidade, atratividade e conformidade.

- Eficiência: refere-se a um conjunto de atributos que relacionam-se entre o nível de desempenho do software e a quantidade dos recursos utilizados sob certas condições estabelecidas, tais como comportamento em relação ao tempo, comportamento em relação aos recursos e à conformidade.

- Manutenibilidade: refere-se a um conjunto de atributos que relacionam-se ao esforço necessário para fazer certas modificações no software, tais como analisabilidade, modificabilidade, estabilidade, testabilidade e conformidade.

- Portabilidade: A capacidade do sistema ser transferido de um ambiente para outro. Como "ambiente", devemos considerar todos os fatores de adaptação, tais como diferentes condições de infra-estrutura (sistemas operacionais, versões de bancos de dados, etc.), diferentes tipos e recursos de hardware (tal como aproveitar um número maior de processadores ou memória). Além destes, fatores como idioma ou a facilidade para se criar ambientes de testes devem ser considerados como características de portabilidade.

Por exemplo, a IBM monitora a satisfação com seus produtos de software em níveis de CUPRIMDSO. (capacidade [funcionalidade], usabilidade, desempenho, confiabilidade, instalabilidade, manutenibilidade, documentação / informação, serviço, em geral). A Hewlett-Packard se concentra em FURPS (funcionalidade, usabilidade, confiabilidade, desempenho, e

facilidade de manutenção). Outras empresas utilizam dimensões semelhantes de satisfação do cliente de software. Juran chama de tais atributo parâmetros de qualidade, ou parâmetros para a adequação ao uso.

Para aumentar a satisfação total do cliente, bem como a satisfação com diversos atributos de qualidade, os atributos de qualidade devem ser levados em conta no planeamento e design do software. No entanto, estes atributos de qualidade não são sempre congruente um com o outro. Por exemplo, quanto maior é a complexidade funcional do software, mais difícil se torna para alcançar a manutenção. Dependendo do tipo de software e os clientes, fatores de ponderação diferentes são necessários para os atributos de qualidade diferentes. Para os grandes clientes com redes sofisticadas e em tempo real de processamento, o desempenho e a confiabilidade podem ser os atributos mais importantes. Para os clientes com sistemas independentes e operações simples, por outro lado, a facilidade de utilização, instalabilidade, documentação e pode ser mais importante. Figura 1 mostra as possíveis relações de alguns atributos de qualidade. Alguns relacionamentos se apóiam mutuamente, alguns são negativos, e outros ainda não são claros, dependendo dos tipos de clientes e aplicações. Para o software com um conjunto de clientes diversificados deve-se, portanto, estabelecer metas para vários atributos de qualidade e para atender às exigências dos clientes não é fácil.

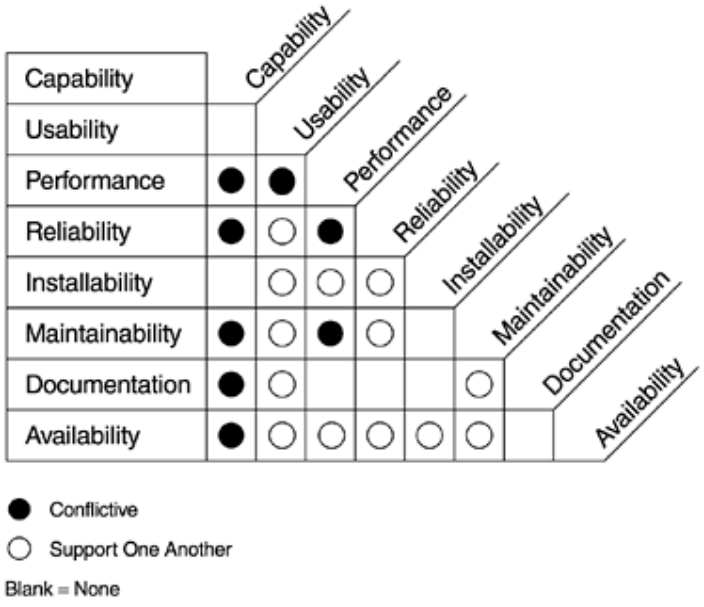

Figura 1: Inter-relacionamento de atributos de software - CUPRIMDA.

Em vista dessas discussões, a definição atualizada da qualidade (ou seja, a conformidade com os requisitos dos clientes) é especialmente relevante para a indústria de software. Não é surpreendente que os erros de requisitos constituem uma das categorias principais problemas no desenvolvimento de software. De acordo com [BEGOSSO 2002], 15% ou mais de todos os defeitos de software são erros de requisitos. Um processo de desenvolvimento que não trata de requisitos de qualidade é obrigado a produzir má qualidade de software.

No entanto, uma outra visão da qualidade do software é a da qualidade do processo versus qualidade do produto final. De necessidades do cliente para a entrega de produtos de software, o processo de desenvolvimento é complexo e muitas vezes envolve uma série de etapas, cada uma com caminhos de realimentaçao.

2.3.3. Economia de Qualidade de Software

Qualidade de software pode ser visto como uma questão econômica. Você sempre pode executar outro teste ou fazer uma nova inspecção. Em sistemas grandes, cada novo teste geralmente expõe uma série de novos defeitos. Assim, é difícil saber quando parar os testes. Embora seja importante para produzir um produto de qualidade, cada teste custa dinheiro e leva tempo. Economia é, portanto, uma questão de qualidade importante, não só por causa desta decisão de teste, mas também por causa da necessidade de otimizar os custos do ciclo de vida de qualidade. A chave para fazer isso é reconhecer que você deve colocar um produto de qualidade em teste antes de você poder esperar para receber um fora.

A economia da qualidade de software em grande parte diz respeito aos custos de falhas de detecção, prevenção e remoção de defeitos. O custo de encontrar e corrigir um defeito inclui cada um dos seguintes elementos:

- Determinar de que existem problemas;
- Isolar a origem do problema;
- Determinar exatamente o que há de errado com o produto;
- Corrigir os requerimentos conforme necessário;

- Corrigir o projeto conforme necessário;
- Corrigir a implementação conforme necessário;
- Inspecionar a correção para garantir que está correta;
- Testar a correção para garantir que o defeito identificado foi corrigido;
- Testar a correção para grantir que não foi cousado outro problema;
- Alterar a documentação conforme necessário para refletir a correção.

Enquanto cada correção não irá envolver cada elemento do custo, quanto maior o defeito é no produto, maior será o número de elementos que provavelmente serão envolvidos. Encontrar um problema de requisitos durante os testes pode, portanto, ser muito caro. No entanto encontrando um erro de codificação durante uma revisão de código geralmente custará muito menos. O Objetivo, portanto, devem ser de remover os defeitos dos requisitos, do designer, e do código o mais rapidamente possível. Revisar e inspecionar os programas logo depois que eles são produzidos minimiza o número de defeitos no produto em cada etapa. Isso também minimiza a quantidade de retrabalho e os custos de retrabalho. É também provávelmente reduz os custos de encontrar os defeitos em primeiro lugar.

2.3.4. Gerenciamento de Qualidade de Software

As responsabilidades dos gerentes e engenheiros de Software de uma organização é garantir que o nível garantido de qualidade do produto seja atingido. Eles encorajam as equipes de desenvolvimento a assumir a responsabilidade pela qualidade de seu trabalho e desenvolver novas abordagens para a melhoria dessa qualidade. Embora os padrões e os procedimentos sejam à base do gerenciamento de qualidade, os gerentes de qualidade experientes reconhecem que existem aspectos intangíveis na qualidade de software (adequação, legibilidade, etc), que não podem ser incorporados em padrões. Eles apóiam as pessoas interessadas nesses aspectos intangíveis de qualidade e encorajam o comportamento profissional de todos os membros da equipe.

O gerenciamento de qualidade de software pode ser estruturado em três atividades principais:

1. *Garantia de Qualidade* O estabelecimento de uma estrutura de procedimentos e de padrões organizacionais, que conduzam ao software de alta qualidade.
2. *Planejamento de Qualidade* A seleção de procedimentos e de padrões adequados a partir dessa estrutura e a adaptação destes para um projeto de software.
3. Controle de Qualidade A definição e aprovação de processos que assegurem que os procedimentos e os padrões de qualidade de projeto sejam seguidos pela equipe de desenvolvimento de software.

2.4. GESTÃO DE QUALIDADE TOTAL(TQM)

O termo de Gestão da Qualidade Total (TQM) foi originalmente criado em 1985 pela Naval Air Systems Command para descrever a sua abordagem de gerenciamento ao estilo japonês para melhoria da qualidade. O termo tem assumido uma série de significados, dependendo de quem está interpretando e de como é aplicado. Em geral, no entanto, representa um estilo de gestão que visa alcançar sucesso a longo prazo, associando qualidade e satisfação do cliente. Básico para a abordagem é a criação de uma cultura em que todos os membros da organização participam na melhoria dos processos, produtos e serviços. Vários métodos específicos para a implementação da filosofia TQM são encontrados na tese de doutorado de BEGOSSO (2002).

Desde a década de 80, muitas empresas norte-americanas adotaram a abordagem TQM para a gestão de qualidade. O Malcolm Baldrige National Quality Award (MBNQA), estabelecido pelo governo dos EUA em 1988, destaca a adoção de uma filosofia e estilo de gestão. A adoção da ISO 9000 como o padrão de gestão da qualidade pela Comunidade Europeia e a aceitação dessas normas pelo setor privado dos EUA nos últimos anos ilustra ainda mais a importância da filosofia da qualidade em ambientes de negócios de hoje. No computador e na indústria eletrônica, exemplos de aplicação do TQM com sucesso incluem Controle Hewlett-Packard (HP) de Qualidade Total (TQC), Estratégia Six Sigma da Motorola, e a Qualidade da IBM, impulsionada pelo mercado. Na verdade, a Motorola ganhou o primeiro prêmio MBNQA (em 1988) e o prêmio AS/400 da IBM em Rochester, Minnesota (1990).

O controle de qualidade total (TQC) da Hewlett-Packard se concentra em áreas-chave como compromisso de gestão, liderança, foco no cliente, a participação total, e de análise sistemática. Cada área tem estratégias e planos para impulsionar a melhoria da qualidade,

eficiência e capacidade de resposta, com o objetivo final de atingir o sucesso através da satisfação do cliente (BEGOSSO 2002). No desenvolvimento de software, o Programa de Qualidade de Software e Análise de Produtividade (SQPA) é uma das abordagens para aumentar a qualidade.

A estratégia Six Sigma da Motorola se concentra em alcançar níveis de qualidade rigorosos, a fim de obter a satisfação total do cliente. Redução do tempo de ciclo e uma gestão participativa estão entre as principais iniciativas da estratégia (Smith, 1989). Seis Sigmas não são apenas uma medida do nível de qualidade; inerente ao conceito são as melhorias de design de produtos e reduções em variações de processo. Seis Sigmas são aplicadas a qualidade do produto, bem como tudo o que pode ser suportado por dados e de medição. "O cliente é o árbitro final" é o tema chave da estratégia da IBM de qualidade impulsionada pelo mercado. A estratégia compreende quatro iniciativas: a eliminação de defeitos, redução do tempo de ciclo, satisfação de cliente e parceiros de negócios, e aderência à disciplina de avaliação Baldrige.

Apesar das variações na sua execução, os principais elementos de um sistema TQM podem ser resumidos da seguinte forma:

- Foco no cliente: O objetivo é alcançar a satisfação total do cliente. Foco no cliente inclui o estudo dos desejos e necessidades, reunindo as exigências dos clientes, medindo e gerenciar a satisfação dos clientes.

- Processo: O objetivo é reduzir as variações de processo e para alcançar melhoria contínua de processos. Esse elemento inclui tanto o processo de negócios e o processo de desenvolvimento do produto. Através do processo de melhoria, a qualidade do produto, será reforçada.

- Lado humano da qualidade: O objetivo é criar uma cultura de qualidade na empresa. As áreas de foco incluem liderança, compromisso de gestão, a participação total, capacitação dos funcionários, e outros fatores sociais, psicológicos e humanos.

- Medição e análise: O objetivo é conduzir a melhoria contínua em todos os parâmetros de qualidade do sistema de medição objetivo-orientado.

Além disso, uma organização que adota as práticas de TQM deve ter liderança executiva, se concentrar em infra-estrutura, treinamento e educação, e deve fazer o planejamento da qualidade estratégico.

A figura 2 é uma representação esquemática dos elementos chave do TQM. Claramente, a medição e análise são os elementos fundamentais para medir a melhoria contínua.

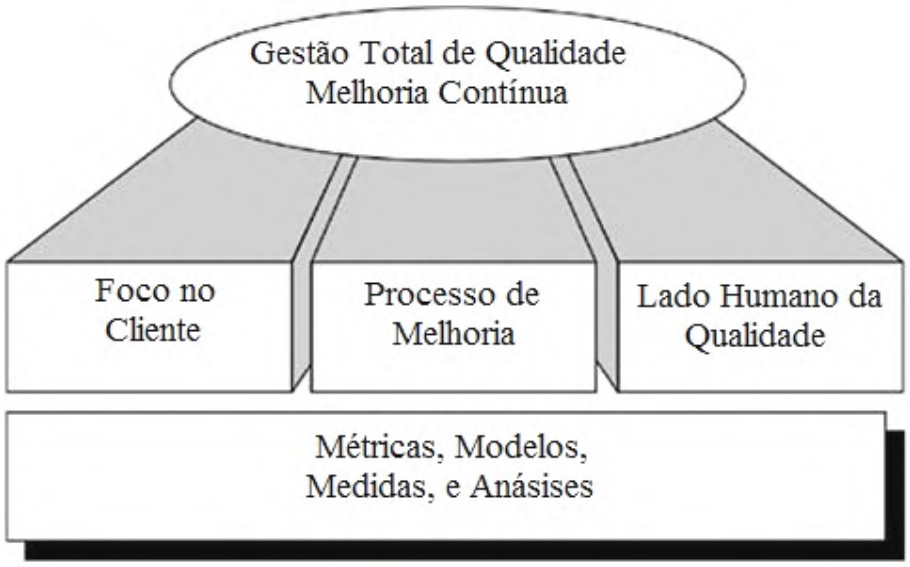

Figura 2: Elementos chaves da Gestão da Qualidade Total.

Vários quadros organizacionais têm sido propostos para melhorar a qualidade que pode ser usada para fundamentar a filosofia TQM. Exemplos específicos incluem o Modelo Integrado de Maturidade da Capacidade (HUMPHREY 1995) desenvolvido no Instituto de Engenharia de Software (SEI), assim como, o próprio PSP sendo o modelo principal da pesquisa, e que será destacado nos capítulos seguintes.

3. O PSP (PERSONAL SOFTWARE PROCESS)

Mesmo os melhores engenheiros de Software cometem muitos erros, e alguns desses defeitos que eles introduzem podem ser incrivelmente penosos para serem descobertos. Tais dificuldades pessoais, enquanto inconveniente para engenheiros individuais, pode ser um problema grave para a organização de um grupo de Engenheiros. A Qualidade do Software começa com o engenheiro individual. Se mesmo os seus pequenos programas não são de altíssima qualidade, eles serão difíceis de serem testados.

Um engenheiro de software pode observar que sua produtividade aumenta quando escreve programas muito pequenos. Sua produtividade, no entanto, cai acentuadamente quando desenvolve grandes sistemas, uma significante parte do problema é causada por defeitos. Quando se escreve sistemas grandes, a dificuldade de encontrar e consertar problemas aumenta exponencialmente. Se o Engenheiro de Software constantemente escreve pequenos programas de altíssima qualidade, não só poderia escrever melhores programas, como também melhorar sua produtividade e a sua organização.

3.1. ESTRATÉGIA DO PROCESSO DE SOFTWARE PESSOAL (PSP)

O PSP (Personal Software Process) é um processo de auto-melhoria projetado para ajudar o engenheiro de software a controlar, administrar e melhorar o modo como ele trabalha. Sendo uma estrutura de formulários, diretrizes e procedimentos para o desenvolvimento de software. O PSP fornece os dados históricos que o engenheiro de software precisa fazer e conhecer melhor seus compromissos e tornar os elementos rotineiros do seu trabalho mais previsíveis e mais eficientes.

Essencialmente o propósito do PSP é servir como uma base para cooperar e melhorar o trabalho do engenheiro de software. É uma poderosa ferramenta que pode ser usada de diferentes formas, por exemplo, para administrar o seu trabalho, avaliar o seu talento e construir suas habilidades. Pode ajudar a planejar melhor, descobrir precisamente o próprio desempenho e medir a qualidade de seus produtos.

Ao invés de usar uma abordagem para cada trabalho, engenheiros de software precisam de um conjunto sofisticado de ferramentas e métodos e o conhecimento para usar elas apropriadamente. O PSP fornece os dados e técnicas analizadas que o engenheiro de software pode usar para determinar quais tecnologias adotar e quais métodos auxiliam melhor seu trabalho.

O PSP não é uma resposta mágica para todos os problemas do engenheiro de software. Embora pode sugerir onde e como melhorar, porém, as melhorias devem ser feitas pelo próprio profissional.

3.2. A LÓGICA PARA UMA DISCIPLINA DE ENGENHARIA DE SOFTWARE

O Software tem se tornado uma questão critica na sociedade moderna. Todo mundo parece precisar de mais e melhores softwares rápidos e baratos. Muitos projetos desenvolvidos são agora tão grandes e complexos que alguns especialistas já não podem lidar com eles. Infelizmente, não há nenhum sinal de uma nova tecnologia mágica para resolver estes problemas. As alternativas são para melhorar o desenvolvimento das práticas dos Engenheiros de Software enquanto encorajam pessoas a entrar neste campo.

Os métodos intuitivos de desenvolvimento de software geralmente usados atualmente só são aceitáveis porque não existem alternativas. A prática atual está mais próxima de uma arte do que de uma disciplina de engenharia. Os profissionais geralmente desenvolvem seus próprios métodos e técnicas privadas. A maioria dos produtos de software já finalizados podem funcionar, mas normalmente só depois de extensos testes e reparos. De um ponto de vista científico, o processo é imprevisível.

Esta situação se torna crítica quando a contribuição de cada indivíduo é exclusivamente importante. Um projeto bem executado é composto por boas ferramentas, métodos, técnicas e engenheiros bem capacitados, cada engenheiro é um contribuinte altamente competente e disciplinado. Engenheiros individuais se salientam ocasionalmente, mas o projeto inteiro é muito mais do que a soma dessas partes, e uma única ação ruim executada de qualquer indivíduo pode danificar o projeto inteiro.

Na maioria das profissões, competência requer proficiência demonstrada com métodos estabelecidos. Não é uma questão de criatividade versus habilidade, porque frequentemente trabalho criativo simplesmente não é possível até que a pessoa domine as técnicas básicas. Disciplinas bem fundadas encapsulam anos de conhecimento e experiência. Profissionais iniciantes têm que demostrar proficiência em muitas técnicas antes de lhe permitirem executar os procedimentos mais rotineiros. A habilidade sem defeito, uma vez adquirida, aumenta a criatividade. Um profissional qualificado em um campo pode superar até mesmo o leigo mais brilhante, mas inexperiente.

Uma organização disciplinada de engenharia de software terá práticas bem definidas. Seus profissionais usarão essas práticas, irão se monitorar e se esforçar para melhorar seu desempenho e vão se sentir responsáveis pelo controle de qualidade. Terão a confiança e os dados requeridos para resistir a demandas desarrazoadas de compromissos.

3.3. PROCESSO DE SOFTWARE

O Processo de Software é uma sucessão de passos requeridos para o desenvolvimento ou evolução de um software. Uma definição do processo de software é uma descrição deste processo. Quando corretamente projetadas e apresentadas, a definição guia os engenheiros de software como eles trabalham.

Mais especificadamente, o processo de software estabelece uma estrutura técnica e administrativa para a aplicação de métodos, ferramentas, e pessoas para as tarefas de software. Enquanto a definição de processo identifica papéis e especifica tarefas. A definição ainda estabelece medidas e fornece critérios de entrada e saída para cada grande passo.

Processos definidos provêem os seguintes benefícios:

- habilitam a efetiva comunicação entre o processo entre usuários, desenvolvedores, gerentes, clientes e pesquisadores;

- aumentam o entendimento do Analista, estabelecem uma base precisa para a automação do processo, e facilitam a mobilidade do pessoal;

- facilitam o reuso de processos. O desenvolvimento de processos consome tempo e são caros. Poucos grupos de projeto podem dispor de tempo ou recursos para definir completamente o modo como eles trabalharão. Eles podem economizar ambos, usando os elementos reutilizáveis padrões que um processo definido estabelece;

- suportam a evolução de processo estabelecendo meios para a aprendizagem de processos e uma fundação sólida para a melhoria de processos;

- ajudam a administração dos processos. A Administração efetiva requer planos claros e um modo preciso e quantificado para medir a situação contra eles. Processos definidos estabelecem, assim, uma estrutura.

3.4. MATURIDADE DE SOFTWARE

Os Engenheiros de Software frequentemente encontram dificuldades em definir, compreender e introduzir processos para desenvolvimento de software em larga escala por geralmente serem grandes e complexos. Apenas definir um bom conjunto de métodos e praticá-los não é suficiente, os engenheiros devem estar cientes que precisam de mudanças, apresentar o que estas mudanças proporcionam e suportá-las enquanto aprendem e praticam um novo processo.

1 Inicial: O Processo de Software é caracterizado como um *ad hoc*[*] e até mesmo caótico ocasionalmente. Poucos processos estão fixados e o processo depende do esforço individual.

2 Repetível: são estabelecidos processos básicos de gerenciamento de projeto, para identificar custos, horários e funcionalidades. A ordem de processo necessária está em repetir sucessos em projetos com aplicações semelhantes.

3 Definição: o processo de software para administração e atividades de engenharia é documentado, normalizado, e integrado a um processo de software padrão para a organização.

[*] Em engenharia de software, a expressão *ad hoc* é utilizada para designar ciclos completos de construção de softwares que não foram devidamente projetados em razão da necessidade de atender a uma demanda específica do usuário, ligada a prazo, qualidade ou custo.

Todos os projetos usam uma versão aprovada e adaptada do projeto do processo de software padrão da organização, para desenvolver e prolongar a vida do software.

4 Gestão: detalhadas medidas do processo de software e qualidade do produto são coletadas. Tanto o processo de software quanto os produtos são quantitativamente entendidos e controlados.

5 Otimização: a melhoria continua do processo é habilitada através de uma avaliação continua do processo e da condução de ideias inovadoras e tecnologias.

O Instituto de Engenharia de Software (SEI), trabalhando com as principais organizações de software norte-americanas, refinou estes níveis de definições e as suas praticas no Modelo Integrado de Maturidade da Capacidade (CMMI) para Software (HUMPHREY 1995). A cada nível, *key process áreas* (KPAs) estabelecem metas e exemplo práticos. O CMM foi revisto e aprimorado por muitos especialistas e representa o seu melhor julgamento dos métodos mais eficazes para atingir os objetivos de cada nível de maturidade.

O PSP tem uma estrutura de maturidade como o CMMI. A Figura 3.1 mostra o CMMI com as KPAs que são resolvidas pelo menos parcialmente pelo PSP, mostradas com um asterisco. Alguns itens do CMMI são excluídos por não se adequarem à aplicação no nível individual.

3.5. RESPONSABILIDADES PESSOAIS

Todo Profissional de Software tem responsabilidades com as pessoas e com ele próprio. Os profissionais necessitam compreender suas habilidades, e aplicá-las nas tarefas que são a eles atribuídas, gerenciando as suas fraquezas e construindo seus pontos fortes. É dever do profissional fazer disto parte de seu trabalho diário, isto também é parte de sua responsabilidade. Somos abençoados com talentos únicos e oportunidades, é obrigação que o profissional decida o que fazer com elas.

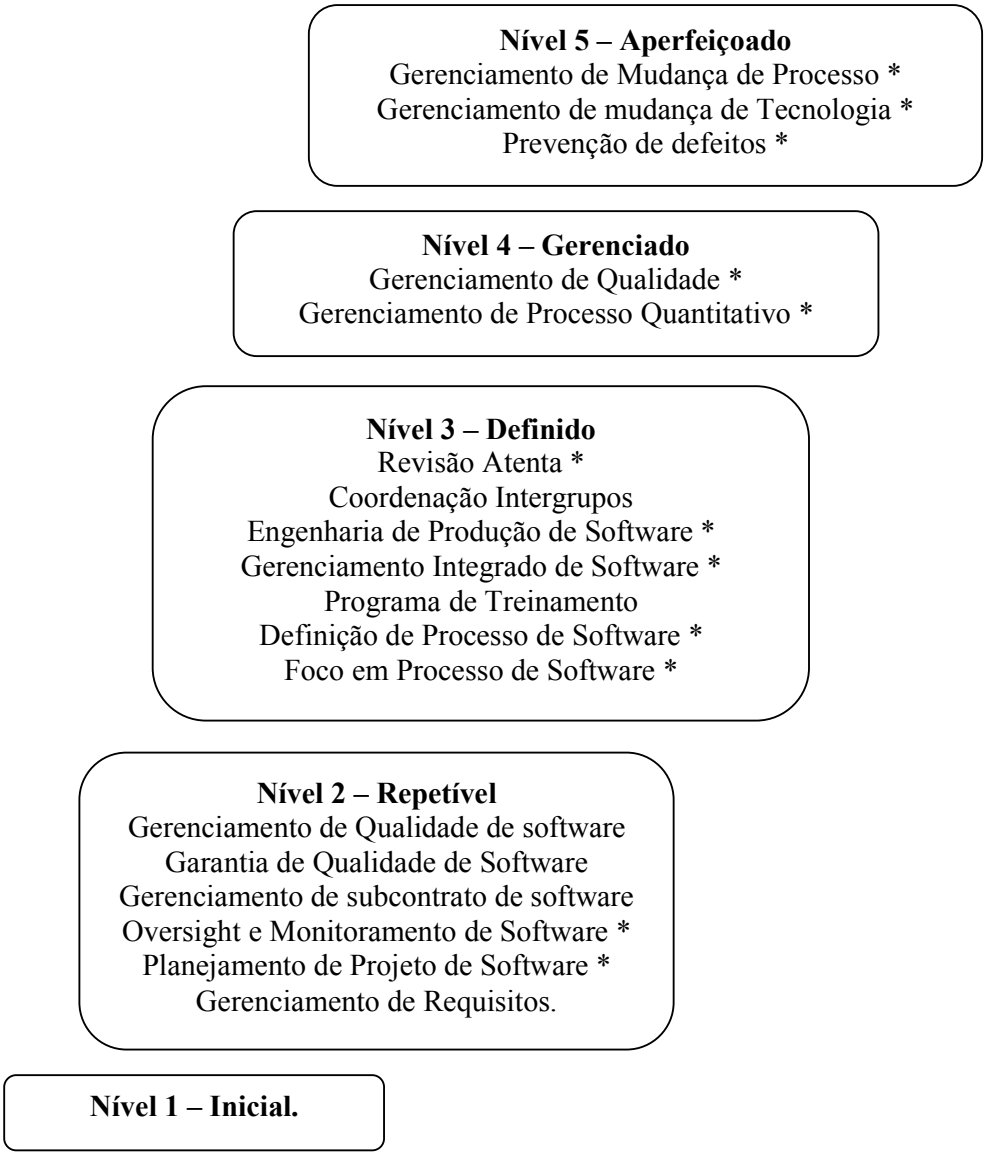

Figura 3.1 – O CMMI e o PSP

3.6. A LÓGICA PARA O PSP

O PSP ainda é novo e sem duvida vai crescer com o tempo, O PSP foi derivado de princípios provados em outros campos. A lógica para o PSP é a seguinte:

- Os Profissionais de software entenderão melhor o que eles fazem se eles definirem, medirem e monitorarem o seu trabalho.

- Eles terão uma estrutura de processo definida e critérios mensuráveis para avaliar e aprender através de suas próprias experiências e das dos outros.
- Com este conhecimento e experiência, eles podem selecionar quais métodos e praticas que melhor se adaptem as suas tarefas e habilidades particulares.
- Usando um conjunto padronizado ordenadamente, constantemente praticado, e praticas pessoais de alta qualidade, eles serão os membros mais efetivos dos seus grupos de desenvolvimento e projeto.

Essa lógica é baseada em cinco princípios. Esses princípios são resumidos abaixo:

1. Um processo definido e estruturado pode melhorar a eficiência do trabalho.
2. Processos pessoais definidos devem se ajustar às habilidades individuais e preferências de cada engenheiro de software.
3. Para os profissionais estarem confortáveis com o processo definido, eles devem estar envolvidos em sua definição.
4. Como as perícias e as habilidades dos profissionais evoluem, esses processos deveriam fazem o mesmo.
5. A melhoria continua de processo é alcançada através de rápido e explicito feedback.

3.7. PRODUTIVIDADE E O PSP

No inicio quando o estudante de PSP escrever seus pequenos programas, a sua produtividade deve recuar isso se deve porque o PSP envolve muitas tarefas que não são normalmente uma parte do desenvolvimento de pequenos programas. Ele apresenta etapas de planejamento, medição e analise para a construção das habilidades que você precisa para introduzir produtos de larga escala. À medida que você ganha experiência com o PSP, no entanto, você ira encontrar a sua melhoria de produtividade, possivelmente além de onde você começou.

3.8. CAVEATS (RESSALVAS)

Como você vai usar o PSP, tenha os seguintes pontos em mente:

O trabalho desenvolvido por (HUMPHREY 1995) concentra-se no projeto, codificação e fases de testes de desenvolvimento de software. Porém, o PSP não se aplica só a eles, mas também a quase qualquer outro aspecto do processo de software, inclusive especificação de requerimentos, manutenção de produto, planejamento de testes e desenvolvimento de documentação.

Cronograma, custos e problemas de qualidade de desenvolvimento de código tipicamente custam muito às organizações. Enquanto custos de testes são óbvios, custos de documentação e instalação não são claramente relacionados à qualidade do código. Para muitos produtos, muita documentação é requerida, uma vez que o produto é difícil de entender, instalar ou para consertos ou atualizações posteriores. Muito dessa documentação não seria necessária se o código fosse inicialmente de alta qualidade. Assim a organização de desenvolvimento e a organização do usuário seriam auxiliadas por uma melhor qualidade de código.

3.9. OS NÍVEIS DO PSP

A implantação do PSP é dividida em sete níveis, Essa implantação é feita de maneira incremental. Os níveis superiores adicionam características aos níveis já implantados. Isto minimiza o impacto da mudança no processo do engenheiro, no qual ele somente tem que adaptar novas técnicas as já existentes. Abaixo os níveis do PSP são descritos de forma resumida. Os níveis do PSP podem ser vistos em uma representação gráfica na Figura 3-2.

- PSP 0 – a base: o primeiro passo do PSP é estabelecer uma base (baseline) que inclui algumas medidas básicas e um relatório. Está *baseline* providencia uma base concreta para medir o progresso e uma fundação definida sobre à qual melhorar. O PSP 0 deve ser o processo normal que você usa para escrever software.
- PSP 0.1 – o PSP 0 é reforçada para o PSP 0.1, adicionando um padrão de código, medidas de tamanho e a proposta de melhoria de processo (PIP – Process Improvement Proposal). O

PIP é a forma que providencia uma maneira estruturada para gravar os problemas do processo, experiências e sugestões de melhorias.

- PSP 1 – O processo de planejamento pessoal: o PSP 1 adiciona o planejamento de passo ao PSP 0. O Incremento inicial adiciona um relatório de testes e estimativas de recursos e tamanhos.
- PSP 1.1 – No PSP 1.1 são introduzidos planejamento de horários e tarefas.

"*Uma vez que você conhece a sua própria taxa de desempenho, você consegue planejar o seu trabalho mais precisamente, fazer compromissos mais realisticamente, e cumprir com esses objetivos de forma mais consistente*".

(HUMPHREY 1995).

- PSP 2 – O processo de administração da qualidade pessoal: para administrar os seus defeitos, o desenvolvedor tem que saber quantos ele faz. O PSP 2 acrescenta técnicas de revisão ao PSP 1, para ajudar a achá-los no início, quando forem menos caros para resolver. Faz-se isto juntando e analisando os defeitos achados na compilação e nos testes dos primeiros programas. Com esses dados, o desenvolvedor pode estabelecer listas de conferência de revisão e fazer suas próprias avaliações de qualidade e processo.
- PSP 2.1 - O PSP 2.1 estabelece critérios de perfeição de projeto e examina varias técnicas de verificação e consistência de projeto.

- PSP 3 – Até agora, o PSP foi concentrado em um simples processo linear para construir pequenos programas. Um programa, programa de larga escala, por exemplo, com 10 mil linhas de código, é muito grande para e escrever, depurar e fazer revisão de código usando o PSP 2. A estratégia do PSP 3 é subdividir um programa maior em pedaços do tamanho requerido pelo PSP 2. A primeira construção é um modulo básico ou núcleo (KERNEL), que aumenta em ciclos de iteração. Em cada repetição, faz-se um PSP 2 completo, incluindo projeto, compilação e testes. Assim, o PSP 3 é satisfatório para programas de até varias mil LOC (KLOC) [*].

O processo PSP 3 cíclico, efetivamente escala programas grandes contanto que cada incremento sucessivo seja de alta qualidade.

---

[*] LOC – Linhas de Código (Lines of Code)
KLOC – Milhar(es) de Linhas de Código.

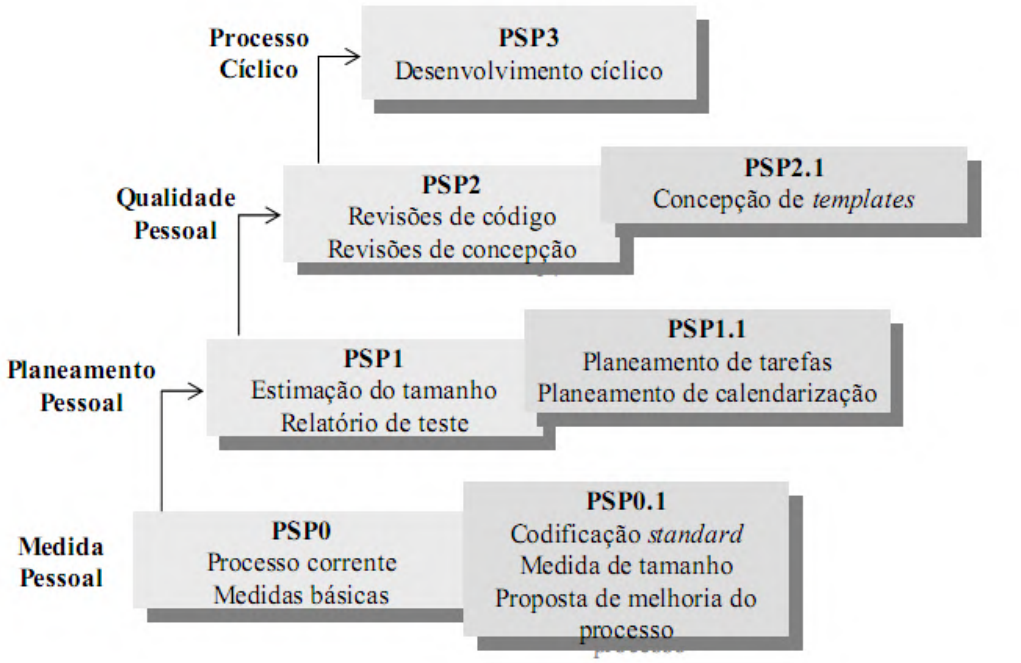

Figura 3-2 Os níveis do PSP.

- TSP – O processo de Software da equipe. Usando o PSP 3, você pode construir programas com mais 10 KLOC. Existem, no entanto, dois problemas típicos para os programas de grandes dimensões. Primeiro como o tamanho cresce, aumentam assim o tempo e os esforços necessários. Este pode ser um problema se você for o único engenheiro no projeto. Em segundo lugar, a maioria dos engenheiros tem dificuldade para visualizar todas as faces importantes do programa, mesmo as de tamanho moderado. Há tantos detalhes e inter-relações que podem passar despercebidas algumas dependências lógicas, as interações de tempo ou condições de exceção. Este problema é agravado pelo que é chamado habituação, ou auto-hipnose, o que faz com que eles não vêem o design real na página ou na tela, mas sim suas imagens mentais do design. Assim, eles tendem a perder erros "óbvios", não porque os erros são muito complexos, mas porque os engenheiros não conseguem vê-los.

Há tantas possíveis respostas para estas perguntas. Um das mais poderosas, porém, é o processo de Software da equipe (TSP), onde você chama com o apoio de seus pares. Quando várias pessoas cooperam em um projeto comum, analisando o trabalho um do outro. Esta avaliação é apenas parcialmente eficaz porque as equipas podem sofrer de habituação. Isto

pode ser combatida através com revisões periódicas do projeto por um estranho. O papel do estranho é fazer as perguntas "tolas". Uma porcentagem surpreendente dessas questões "tolas" vai identificar questões fundamentais que têm sido assumidas por tanto tempo que acabaram esquecidas.

4. METODOLOGIA DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

4.1. PSP 0 – A BASE:

Ao se definir um processo pessoal, começa-se a pensar em seus termos. Tarefas abstratas são estruturadas e sujeitas a análise racional. Adquire-se uma estrutura para medidas e um foco para melhorias. O Principal objetivo desta secção e fornecer um quadro para recolha dos dados iniciais do projeto.

O PSP 0 é a base para ampliações de processo introduzidas nos níveis seguintes. Ele é mostrado na figura 4-1. Os *scripts* guiam através dos passos do processo, os *logs* ajudam a registrar os dados dos processos e o resumo de plano provê um modo conveniente para registrar e informar os resultados. Este processo provê:

- uma estrutura conveniente para fazer tarefas em pequena escala: o que fazer primeiro? O que fazer depois? E assim por diante.
- uma estrutura para medir estas tarefas: um processo definido permite juntar dados do tempo gasto em cada tarefa de software e monitorar o numero de defeitos introduzidos e removidos em cada passo de processo. Estes dados ajudam a analisar o processo, entender erros e melhora-lo.
- uma base para a melhoria de processo: se não se sabe exatamente o que se esta fazendo, é difícil melhorar este trabalho.

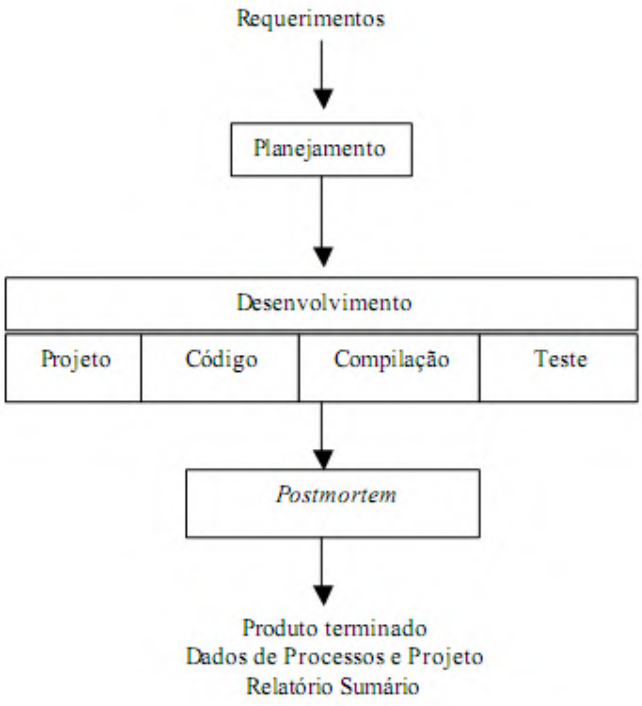

Figura 4-1 O PSP0

4.1.1. Os elementos do processo

Primeiro, no passo de planejamento é produzido uma plano para fazer o trabalho. Depois, é feito o desenvolvimento do software. No fim, no passo de Postmortem, é feita uma comparação do desempenho atual com o planejado, são registrados os dados de processo e é produzido um relatório sumario. Embora estes passos de planejamento e Postmortem possam parecer desnecessários ao se escrever pequenos programas, eles se tornam essenciais para se construir um processo pessoal disciplinado. Se o programa é tão simples que um plano parece insensato, o plano deve ser trivial de produzir.

4.1.2. O Processo

Um script guia o estudante de PSP através de um processo. Os elementos principais deste script são seus propósitos, os critérios de entrada, as fases (ou passos) a serem executados, e

os critérios de saída. O Script do processo PSP 0 é mostrado na Tabela A-1, no apêndice A. Ele descreve em palavras a estrutura de processo apresentada na Figura 4-1. Um segundo script do PSP 0, o Script de Planejamento, é mostrado na Tabela A-2. Ele resume brevemente os passos simples de planejamento requeridos no PSP 0. As fases de planejamento e Postmortem estão bastante claras nos scripts das Tabelas A-2 e A-4, mas a fase de desenvolvimento na Tabela A-3 tem quatro passos: projeto, codificação, compilação e teste. Até que estes passos tenham critérios explícitos de entrada e saída, não há nenhuma maneira de dizer quando começa ou termina cada um. Uma confusão comum nesses casos diz respeito à distinção entre codificação e compilação. Um modelo do fluxo de processo do PSP 0 é apresentado na figura 4-2.

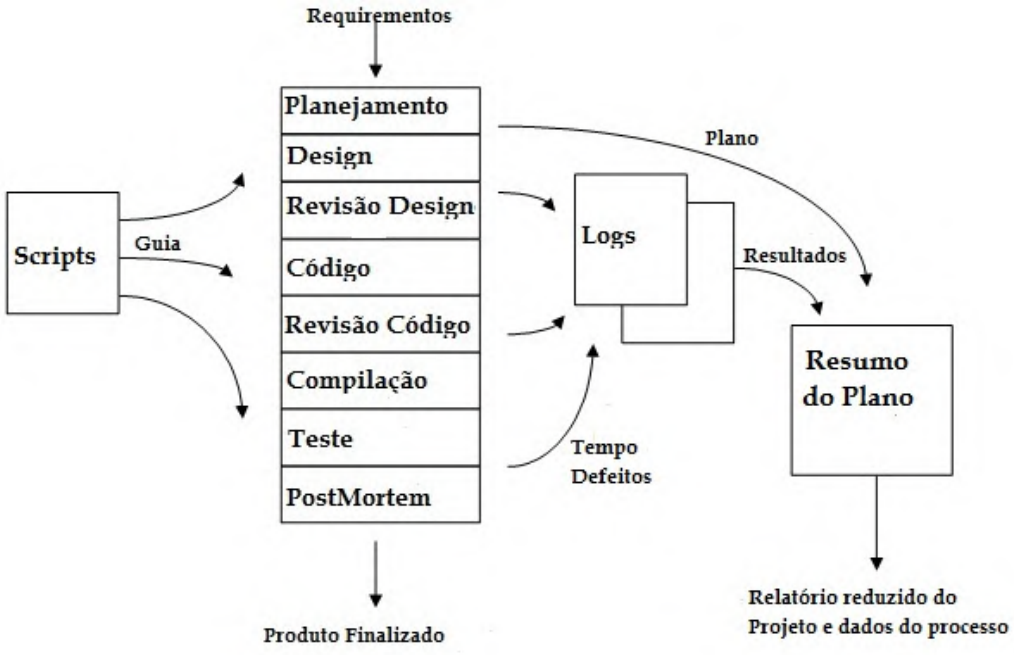

Figura 4-2 – Fluxo do Processo PSP 0

4.1.3. Medidas do PSP 0

O PSP0 tem duas medidas:

- o tempo gasto por fase: registro do tempo que se gasta em cada parte de processo do PSP. O objetivo é determinar onde o tempo é gasto e como esta distribuição muda à medida que o processo muda.
- os defeitos por fase: registro dos dados de cada defeito (mudança do programa) localizado durante a compilação e testes.

Junta-se os dados de tempo e defeitos para obter a base para planejar os projetos futuros. Eles dão uma base contra a qual medir o desempenho, mostram onde se passa mais tempo e indicam onde se faz e se acha a maioria dos defeitos. Eles também ajudarão a ver como estas distribuições mudam à medida que o processo evolui.

A Tabela A-7 mostra o Log de Registro de Tempo, e a Tabela A-8 contém as instruções para completá-lo.

O Log de Registro de Defeito e suas instruções são mostradas nas Tabelas A-9 e A-10. Quando o estudante terminar de corrigir o defeito, deve anotar o Tipo de Defeito Padrão mostrado na Tabela A-11. Este padrão foi modelado na IBM Research, e deve ser suficientemente geral para cobrir a maioria das necessidades.

O Sumário de Plano de Projeto do PSP0 e suas instruções são mostrados nas Tabelas A-5 e A-6.

Embora seja possível ajustar o modo particular de cada um de projetar software, Humphrey sugere que os processos do PSP não sejam modificados até a conclusão do curso de PSP, principalmente pelo grande trabalho extra que isso acarretaria.

4.1.4 PSP 0 - Conteúdos do Processo.

O PSP 0 apresenta a família de processos do PSP e seus formulários, scripts, e padrões. Provê uma estrutura ordenada para planejar o trabalho e relatar os resultados, estes elementos de processo podem economizar uma quantia significativa de tempo. Sem formulários padrão, por exemplo, haveria a necessidade de decidir como produzir um plano, quais dados juntar, e como registrá-los.

Como mostrado no script da Tabela A-1, é necessário,

1. assegurar-se que se têm todas as entradas requeridas,
2. assegurar-se que os requerimentos para o trabalho foram entendidos,
3. calcular o tempo em minutos que se espera levar para o desenvolvimento do programa,
4. registrar esse tempo no Resumo de Plano de Projeto, inclusive o tempo que o planejamento tomou,
5. fazer o desenvolvimento,
6. entrar com os dados atuais dos Logs de Registro de Tempo e Defeito na coluna Atual do Sumário de Plano de Projeto.

Ao fazer o desenvolvimento do programa, o projeto, a implementação, a compilação e os testes são efetuados usando os métodos de desenvolvimento atuais do engenheiro de software. Também, são registrados os defeitos no Log de Registro de Defeito e o tempo no Log de Registro de Tempo. Cada fase de processo é descrita nos Scripts de Planejamento, Desenvolvimento e Postmortem.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Os objetivos do PSP 0 são:

- incorporar medidas básicas no processo de desenvolvimento de software;
- requerer mudanças mínimas nas práticas pessoais;
- demonstrar o uso de um processo definido ao se escrever programas pequenos e

- usar o processo atual como uma estrutura de processo introdutório.

As condições prévias para o PSP 0 são que o desenvolvedor seja razoavelmente fluente em pelo menos uma linguagem de programação. O processo PSP0 e seus scripts, formulários, modelos, padrões e instruções são descritos em termos gerais nas seções anteriores. É necessário revisar esses elementos antes de começar a escrever o primeiro programa.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

Os scripts, formulários, modelos e padrões são os itens usados no processo PSP 0. Os scripts e o Resumo de Plano de Projeto mudam em cada processo, mas o Log de Registro de Tempo, o Log de Registro de Defeito e o Tipo de Defeito Padrão são usados sem mudanças em todas as versões de processos subseqüentes. Todos estes itens são incluídos nos números de tabela indicados.

PSP0 - Script de Processo	Tabela A-1
PSP0- Script de Planejamento	Tabela A-2
PSP0 - Script de Desenvolvimento	Tabela A-3
PSP0 - Script de Postmortem	Tabela A-4
PSP0 – Resumo do Plano de Projeto e Instruções	Tabelas A-5 e A-6
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

Os Elementos do PSP 0

Como o PSP0 é o primeiro processo do PSP, todos os seus elementos são novos. Os scripts, Log de Registro de Tempo, Log de Registro de Defeito e Tipo de Defeito Padrão foram descritos no início desta seção.

ESPECIFICAÇÃO DE RELATÓRIO DE PROCESSO

O objetivo principal das aplicações desenvolvidas com o PSP 0 é proporcionar experiência no uso de um processo disciplinado. Embora seja importante produzir um programa funcionando, os critérios para avaliar os exercícios do PSP 0 são:

- os dados de processo estão completos;
- os dados são precisos e auto consistentes;
- o relatório de processo é submetido na própria ordem e formato.

O uso de um formato padrão e ordenado torna mais fácil para o estudante assegurar-se de que os resultados estão completos e para o instrutor assegurar-se que eles estão corretos. Os itens a serem incluídos nos relatórios das aplicações do PSP 0 e a ordem nas quais eles serão submetidos são como segue:

- Resumo do Plano de Projeto do PSP 0
- Log de Registro de Tempo
- Log de Registro de Defeito
- Listagem do programa fonte

4.2. PSP 0.1 – MEDINDO O TAMANHO DO SOFTWARE

4.2.1. O Processo de Planejamento

O Planejamento é o primeiro passo do PSP por três razões. Primeiro sem bons planos não é possível administrar efetivamente nem mesmo projetos de Software de tamanho modesto. Segundo, planejar é uma habilidade que se pode aprender e melhorar com a prática. Terceiro, boas habilidades de planejamento ajudarão a fazer um melhor tamanho de Software.

Embora o planejamento pessoal seja importante para o planejamento de projetos, é só uma parte do processo. Muitos outros assuntos estão envolvidos na produção de um plano completo para um grande projeto. Porém, estes planos de projeto maiores serão

provavelmente mais realísticos quando eles estiverem compostos de planos pessoais múltiplos feitos pelos indivíduos ou grupos que farão o trabalho.

4.2.1.1. O que é um plano

O plano de projeto define o trabalho e como será feito. Provê uma definição de cada tarefa principal, uma estimativa do tempo e recursos requeridos e uma estrutura para revisão da administração e controle. O plano de projeto também é um poderoso veiculo e aprendizado. Quando corretamente documentado, é um ponto de referência para comparar com o desempenho atual. Esta comparação permite aos planejadores verem seus erros estimados e melhorar a precisão de estimativas [HUM 89, aput [HUM 95]] Assim, um plano é muitas coisas. Em organizações de Software maduras, planos são tipicamente usados como:

- Uma base para concordar sobre custos e horários para um trabalho,
- Uma estrutura organizada para fazer o trabalho,
- Uma estrutura para obter os recursos requeridos, e
- Um registro do que foi inicialmente comprometido.

Os planos do PSP têm dois usuários: o desenvolvedor e seus clientes. Para o trabalho do desenvolvedor, são necessárias quatro coisas gerais para um plano:

- Medir o tamanho do trabalho: qual o tamanho do trabalho, quanto tempo ele levará?
- Estrutura de trabalho: como será feito o trabalho? O que será feito primeiro, segundo e assim por diante?
- Estado do trabalho: como se sabe onde se está? O desenvolvedor vai terminar na hora certa e os custos estão sobre controle?
- Avaliação: qual a qualidade do plano? Houve algum erro óbvio, que enganos deveriam ser evitados no futuro, e como se pode fazer um trabalho melhor da próxima vez?

Os clientes poderiam ser o instrutor do curso de PSP, os colegas de trabalho, o gerente ou um usuário final. Estas pessoas também querem quatro coisas gerais do plano:

- Qual é o compromisso? Especificamente, o que será entregue, quando e a que custo?

- Qual será a qualidade deste produto? Alcançará a expectativa do Cliente? O trabalho está corretamente planejado para assegurar os ajustes do produto às suas necessidades? Há provisões para eles fazerem checagens na qualidade de produto? Há provisões para solucionar assuntos?
- Há algum modo para monitorar o progresso? Eles terão cedo advertências de custos, horários ou problemas de qualidade? Neste caso, quando poderão descobrir os problemas e o que podem fazer sobre isto?
- Serão capazes de avaliar a qualidade do trabalho terminado? Podem separar os problemas causados pelo planejamento ruim daqueles causados por mau gerenciamento? O impacto de mudanças de escopo estará claro e identificável?

Quando se examina o plano no contexto destas perguntas, várias coisas ficam claras:

1. O plano deve estar baseado em se fazer uma parte definida do trabalho.
2. O trabalho deve envolver passos múltiplos que estão claramente definidos e medidos. Isto provê uma estrutura para o plano e uma base para monitorar o progresso.
3. Será necessário algum modo de conferir o plano com o usuário antes de se começar o trabalho. Esta sempre é uma boa ideia, e é essencial para qualquer trabalho, mesmo para as menores tarefas.
4. Será necessário fazer declarações de progresso periódicas aos clientes.

4.2.1.2. Planejamento um projeto de Software.

Os passos seguintes ajudarão o desenvolvedor a construir uma estimativa de processo estável e efetiva:

- Começar com uma declaração explicita do trabalho a ser feito e conferir para se assegurar que é o que o cliente espera (os modos nos quais os projetos podem diferir são infinitos).
- Para projetos que levam mais do que alguns dias de trabalho, dividi-los em múltiplas tarefas menores e calcular cada tarefa separadamente. O detalhe adicionado cai melhorar a precisão do plano e provavelmente melhorará a acúracia do desenvolvedor.

- Basear estimativas, comparando o trabalho atual com os dados históricos dos trabalhos anteriores.
- Registrar as estimativas e depois compará-las com os resultados atuais.

4.2.1.3. Estrutura de planejamento

A estrutura de planejamento do PSP é mostrada na figura 4-3. As tarefas executadas são mostradas nos retângulos e os vários dados, relatórios e produtos são mostrados nas elipses. Aqui, começando com uma necessidade do cliente, são definidas as exigências (requerimentos). A seguir, é produzido um projeto conceitual que em troca ajuda a relacionar a estimativa de planejamento ao produto atual que se pretenda construir.

Com este projeto conceitual e dado históricos de produtos previamente construídos, é possível estimar o tamanho provável do novo produto. Com esta estimativa de trabalho, usa-se os dados históricos de produtividade para estimar quantas horas o trabalho levará. Finalmente, estas horas são alocadas em um calendário para se ter um horário de projeto. Com estes dados e com uma data presumida de inicio, é possível estimar agora a data que o trabalho terminará.

Com o plano em mãos, e assumindo que se tenha toda a informação e facilidades necessárias, o desenvolvimento é iniciado. Durante o desenvolvimento e a conclusão do projeto, são registrados o tempo gasto e o tamanho do produto produzido. Estes dados são usados para produzir relatórios periódicos e fazer análises de processo. Estas análises fornecem os dados de tamanho e produtividade para fazer planos futuros.

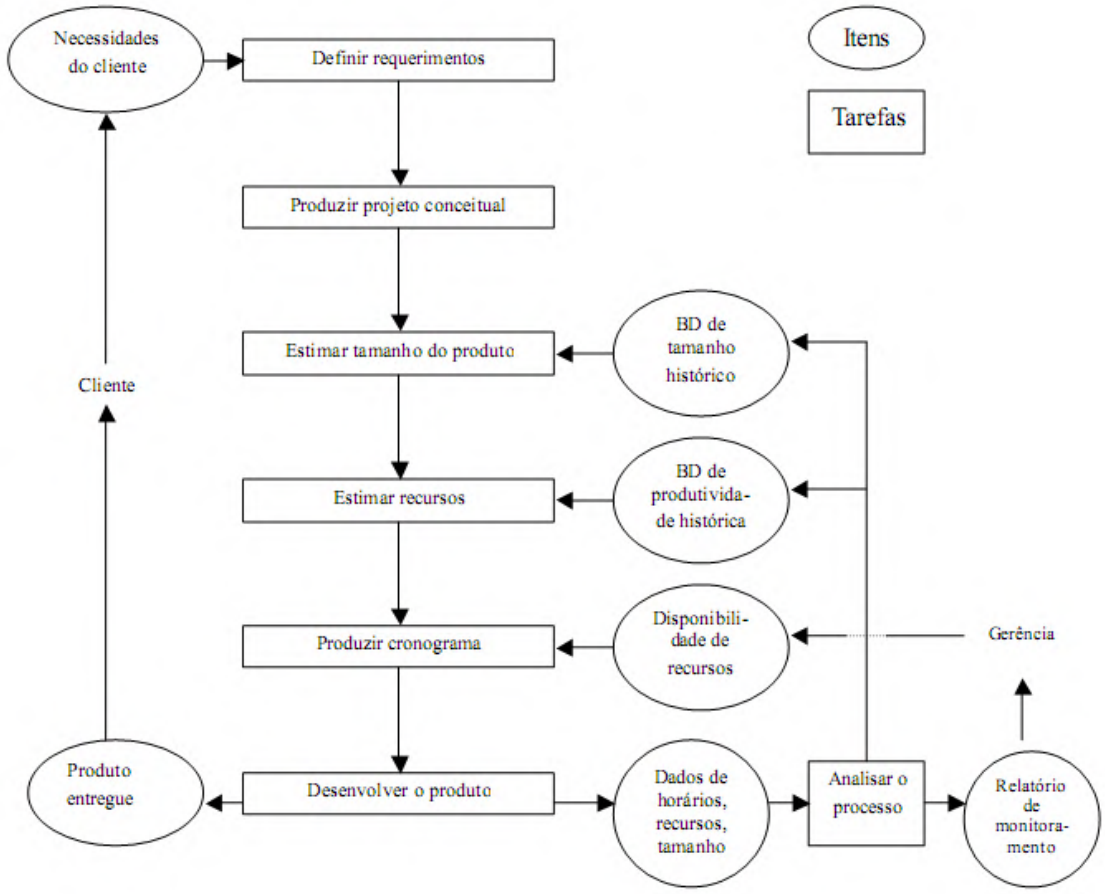

Figura 4-3. Estrutura de Planejamento de Processo.

4.2.1.4. Produzindo um Plano de Qualidade

À medida que o engenheiro de Software constrói suas habilidades de planejamento ele precisa pensar na qualidade do plano. O que constitui um bom plano? Qual a precisão desses planos e o que ele pode fazer para melhorar suas habilidades de planejamento? As seis questões chaves do plano são a seguinte.

1. Está completo?
2. É acessível?
3. Está claro?
4. É específico?
5. É preciso?
6. É acurado?

4.2.2. Medindo o tamanho do Software

O processo de planejamento de software começa com uma estimativa de tamanho do trabalho. Antes de poder estimar o tamanho do software, porém, é necessário um modo consistente e repetível para descrever o tamanho de um produto.

O Projeto de Medida de Processo de Software do SEI desenvolveu um trabalho para descrever medidas de tamanho de Software. Os dois critérios principais para este foram os seguintes:

- Comunicação: “Se alguém usar métodos para definir uma medida ou descrever um resultado de medida, outros saberão precisamente o que foi medido e o que foi incluído ou excluído?”.
- Repetibilidade: “Outra Pessoa poderia repetir a medida e receber o mesmo resultado?”.

Considerando que a contagem de LOC pode estar enganada, ela deve ser tratada com um pouco de cuidado. Devem-se usar definições precisa e anotar cuidadosamente os tipos de elementos de programas incluídos. Não é uma boa ideia usar contagem de LOC para comparar projetos ou organizações porque geralmente há muitas diferenças que não são detectáveis de simples dados de LOCs. Até mesmo ao nível do PSP, a contagem de LOCs não é uma base útil para se comparar a produtividade ou efetividade de indivíduos.

Ao medir a produtividade do desenvolvimento, os Engenheiros de Software normalmente contam o numero de declarações fonte por hora de desenvolvimento. Para este propósito, deve-se contar os recentes desenvolvimentos mais as declarações modificadas. Também é importante usar precisamente as mesmas definições ao se estimar a produtividade de desenvolvimento e no planejamento de projeto.

Quando compara a taxa de defeitos de programas, o Engenheiro, comumente, usa a taxa de defeitos por mil linhas de código adicionado e modificado. Ao estimar a carga provável de trabalho e manutenção para um programa, porém, será mais apropriado considerar as LOCs totais do produto terminado.

Um contador de LOCs pode ser projetado para contar linhas físicas ou linhas lógicas. Um terceiro método combina estes dois; conta linhas lógicas usando um padrão de codificação e um contador de LOCs físicas. Esta é a abordagem usada com o PSP.

Podem ser obtidas muitas estatísticas de LOCs úteis com ferramentas corretamente projetadas. Para cada modificação do programa, por exemplo, o engenheiro poderá saber quantas LOCs foram adicionadas e apagadas. Um modo prático de obter tais dados é com um programa que compare cada versão do programa com a versão anterior. Este comparador estão identifica e conta cada linha adicionada ou apagada.

A chave para medidas de tamanho de *Software* efetiva é assegurar que elas se ajustem às necessidades pessoais do desenvolvedor.

4.2.3. PSP 0.1 – Conteúdos do Processo

Ao PSP0 é acrescentado ao PSP 0.1, adicionando um padrão de codificação, medidas de tamanho e a proposta de melhoria de processo (PIP – Process Improvement Proposal). A PIP é um formulário que provê um modo estruturado para registrar problemas de processo, experiências e sugestões de melhoria.

Antes de usar o PSP 0.1 para desenvolver um programa, faz-se um cálculo do tamanho e uma estimativa de tempo completa. Na conclusão do desenvolvimento, mede-se o tamanho do

programa completado e conta-se ou calcula-se as LOCs reusadas, apagadas, modificadas, adicionadas, totais novas e alteradas e totais novas reusadas.

Objetivos e Condições Prévias

Em adição aos objetivos do PSP 0, este processo tem o objetivo adicional de ajudar a medir e calcular os tamanhos dos programas produzidos. As condições prévias para o PSP 0.1 é a presente seção.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES.

PSP0.1 - Script de Processo	Tabela A-12
PSP0.1- Script de Planejamento	Tabela A-13
PSP0.1 - Script de Desenvolvimento	Tabela A-14
PSP0.1 - Script de Postmortem	Tabela A-15
PSP0.1 – Resumo do Plano de Projeto e Instruções	Tabelas A-16 e A-17
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSOS NOVOS

Os elementos novos para o PSP 0.1 não só incluem scripts e relatório sumário mas também a proposta de melhoria de processo (PIP) e padrões de codificação. As seções seguintes dão uma descrição desses elementos.

Proposta de Melhoria de Processo. O formulário PIP é usado para registrar qualquer problema ou sugestão de melhoria que ocorram no uso de um processo. Muitos problemas de definição de processo concernem a detalhes aparentemente secundários. Porém, são tão detalhados que fazem a diferença entre um processo aborrecido e inconveniente e um processo confortável e eficiente. Embora seja possível lembrar-se posteriormente que algum

formulário ou passo foi um problema, provavelmente os detalhes serão esquecidos se não forem registrados prontamente. O formulário PIP em branco deve ser mantido sempre à mão para, prontamente, registra-se qualquer ideia de melhoria. Detalhes são importantes em um processo pessoa; e, para fixar os detalhes, é necessário completar os PIPs regularmente.

Os PIPs devem ser usados também, para registrar comentários ou fazer notas sobre o que foi aprendido em cada programa. Ao se fazer as tarefas, os PIPs devem ser submetidos com os relatórios de processo.

O formulário PIP é mostrado na Tabela A-18 e suas instruções estão na Tabela A-19. Deve-se reter uma copia de todos os PIPs completados, para uso posterior.

Padrões de Codificação. Além dos programas estarem corretos, o código fonte também deve ser compreensível. Escrever código legível ajudara a pensar mais claramente no projeto e nos testes, modificações e reuso. Comentários claros e precisos também ajudam outras pessoas que trabalham com esses programas.

O padrão de codificação de PSP, para C++, é mostrado na Tabela A.20.

Para ser útil, o padrão de codificação deve ser projetado para a linguagem e o ambiente de trabalho de uso. Assim, é possível alterar o padrão de codificação do PSP para as necessidades particulares, mas devem-se reter todos os itens quês este padrão inclui.

4.3. PSP1 – ESTIMANDO TAMANHO E TEMPO

4.3.1. Estimando o tamanho do software

Esta seção discute primeiramente o problema da estimativa de tamanho e então descreve o método de estimativa PROBE.

A razão principal para estimar o tamanho de um produto de software é ajudar a planejar o desenvolvimento do produto. A qualidade de um plano de desenvolvimento de software geralmente depende da qualidade da estimativa de tamanho. Um bom plano fornece a base para consolidar e prover de pessoal o trabalho, para saber o que tem que ser feito, quando e por quem. Uma causa freqüente de planos ruins é uma estimativa de tamanho ruim.

Uma prática aceita na engenharia, na manufatura e na construção civil é basear planos de desenvolvimento em estimativas de tamanho de produto. Engenheiros civis experientes podem freqüentemente calcular grandes projetos com uma margem de erro de um ou dois por cento do custo total atual.

O grau para o qual se pode acurada e precisamente planejar um trabalho depende do que se sabe sobre ele. No início, ou no estágio de pré-proposta, só se tem uma idéia geral dos requerimentos do produto. Praticamente a única maneira de fazer uma estimativa é por analogia a produtos prévios.

Depois da fase de pré-proposta, se sabe progressivamente mais sobre o produto planejado. Pode-se, então, fazer estimativas mais refinadas de tamanho do trabalho.

Estimativas para trabalhos de software grandes podem ser controladas desse modo. Para fazer uma estimativa precisa, começa-se com uma especificação de projeto.

Então cada parte do trabalho é examinada e estimada. Esta estimativa requer estimativas separadas para cada componente de software, cada documento principal, os casos de teste, planejamento da instalação, conversão de arquivos e treinamento de usuários. Os componentes de programa podem ter diferentes sub-elementos. Se há telas para desenvolver, relatórios para gerar ou lógica funcional para projetar, estes também devem ser calculados. Para grandes produtos de software, há potencialmente muitos detalhes úteis. Para fazer

estimativas precisas, é necessário definir todos estes detalhes e considerar cada um na estimativa.

4.3.1.1. Critérios de estimativa de tamanho

Um método de estimativa de tamanho utilizável deve satisfazer os seguintes critérios:

- deve usar métodos estruturados e treináveis. Um método estruturado facilita a melhoria do treinamento e processo. Também permite monitorar e melhorar o próprio método de estimativa.
- deve ser um método que se possa usar durante todas as fases de desenvolvimento de software e manutenção.
- deve ser utilizável por todos os elementos do software. Não só deve controlar o código, mas também arquivos, relatórios, telas e documentação.
- deve ser satisfatório para análise estatística. Um método de estimativa de tamanho estatisticamente baseado provê os meios para ajustar as estimativas de parâmetros baseadas em dados históricos.
- deve ser adaptável aos prováveis tipos de trabalho a serem feitos no futuro. À medida que se construa dados de estimativa e se ganhe experiência, se estará então construindo um recurso de valor contínuo.
- deve prover os meios para julgar a precisão das estimativas.

Deve-se sempre comparar cada estimativa com o tamanho do produto resultante atual. Revisando estas comparações, freqüentemente as causas dos erros serão conhecidas e o método de estimativa poderá ser mais bem ajustado e melhorado.

Métodos de estimativa populares

Boehm descreveu o método Wideband-Delphi para opiniões múltiplas de experts. Putnam definiu o método de lógica fuzzy e o método de componente-padrão para julgar os tamanhos de produtos novos baseado em dados de tamanho históricos. O método de ponto por função de Albrecht usa fatores standards para julgar a importância relativa de vários requerimentos funcionais. Estes métodos formam a fundação para o método PROBE usado com o PSP.

4.3.1.2. Proxy Based Estimating - PROBE

No planejamento de projetos, geralmente são requeridas estimativas antes de o desenvolvimento começar. Nesta fase inicial, os requerimentos podem ser entendidos, mas pouco é conhecido sobre o próprio produto. O problema da estimativa é, assim, predizer o tamanho provável terminado do produto. Como ninguém pode saber com antecedência o tamanho que um produto planejado, a estimativa de tamanho de software sempre será um processo incerto. Porém, a necessidade é fazer uma estimativa tão acurada quanto possível. Em geral, todos os métodos de estimativa usam dados de programas semelhantes previamente desenvolvidos para estabelecer alguma base para julgar o tamanho do novo programa.

4.3.1.3 Proxies

Exemplos de proxies são objetos, telas, arquivos, scripts ou pontos de função. Os critérios para uma boa proxy são como segue:

- A medida de tamanho da proxy deve se relacionar de perto ao esforço exigido para desenvolver o produto.
- O conteúdo da proxy de um produto deve ser automaticamente contável.
- A proxy deve ser fácil de visualizar no começo de um projeto.
- A proxy deve ser customizável às necessidades especiais das organizações.
- A proxy deve ser sensível a qualquer variação de implementação que impactem custos de desenvolvimento ou esforços.

Muitos tipos potenciais de proxies poderiam satisfazer os critérios previamente esboçados. O método de ponto por função é um candidato óbvio porque é extensamente usado. Muitas pessoas acham o método ponto por função útil para o recurso de estimativas. Outras possíveis proxies são objetos, telas, arquivos, scripts e capítulos de documentos.

4.3.2 O Método PROBE (Proxy-Based Estimating)

O PSP1 formaliza a estimativa de tamanho do programa. Introduz o PROBE, que é um método algorítmico para a estimativa de tamanho. O PROBE analisa o banco de dados histórico e olha para uma tendência na relação entre tamanho calculado e atual. Assume que o desempenho do passado é indicativo do desempenho futuro. Se os bancos de dados históricos mostram que o tamanho do projeto final é tradicionalmente mais alto que a estimativa original do programador por alguma porcentagem, o PROBE permite ao usuário aumentar a estimativa original dele por aquela porcentagem. Usa uma análise de regressão linear na relação entre o tamanho calculado e o tamanho atual. A equação da linha de melhor ajuste é usada para converter uma estimativa inicial em uma estimativa projetada. Pelo uso da distribuição *t-Student*, um salto no erro de estimativa pode ser colocado para um determinado nível de confiança. Se a relação linear entre os pontos de dados for fraca, o PROBE proporcionará para um resultado uma alta gama de erro.

Um fluxograma do procedimento de estimativa de tamanho PROBE é mostrado na Figura 4-4. Estes passos também são explicados no *script* PROBE na **Tabela A-27** e nas instruções do modelo de estimativa de tamanho **(Tabela A-30).**

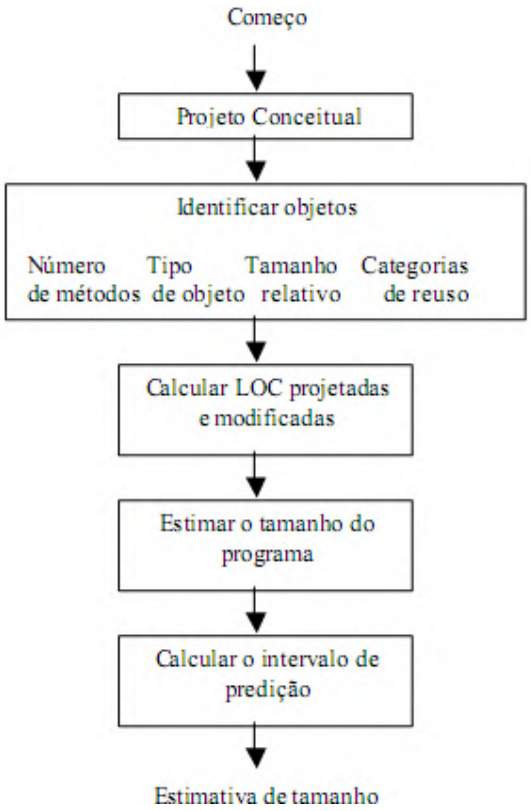

Figura 4-4 - O Método PROBE (Proxy-Based Estimating)

O projeto conceitual

Para a estimativa de tamanho refletir corretamente o produto a ser construído, é necessário começar com um projeto conceitual. Este projeto estabelece uma abordagem de projeto preliminar e nomeia os objetos esperados e suas funções. Sua intenção aqui não é fazer o projeto completo, mas postular os objetos que serão necessários e as funções que eles executarão. Este é um processo de abstração durante o qual o estimador diz, "Se eu tivesse objetos que fizessem as funções A, B, e C, eu saberia construir este produto".

Ao estimar produtos relativamente pequenos, é possível produzir um projeto conceitual diretamente. No projeto conceitual de grandes produtos, o objetivo é dividir cada uma das partes principais do produto em elementos que se assemelham a aqueles com os quais o desenvolvedor tem experiência e dados. Se um objeto está no nível certo e não pertence a quaisquer das categorias existentes, então seu tamanho é estimado como o primeiro de uma nova categoria.

Determinação do tipo de objeto e do tamanho

Quando já se tem o projeto conceitual, cada objeto foi nomeado e sua categoria já foi determinada, é necessário localizar objetos no banco de dados que se assemelhem a cada um destes objetos. Para cada novo objeto, é feito um julgamento de como seu tamanho se compara com os do banco de dados em sua categoria. Com base nesse julgamento, é feita uma estimativa superficial do tamanho do novo objeto.

Por exemplo, considerar a estimativa que um estudante do PSP fez para o programa l0A em C++, mostrada no formulário de estimativa na Tabela 4-1, abaixo:

Estudante	Estudante 12	Data	10/08/2012
Instrutor	HUMPHREY	Programa #	10A

PROGRAMA BASE LOC	ESTIMADO	REAL
TAMANHO BASE (B) => => => => => => => => =>	695	695
LOC DELETADAS (D) => => => => => => => => =>	0	0
LOC MODIFICADAS (M) => => => => => => => =>	5	18

LOC OBJETO

ADICÕES BASE	TIPO[1]	MÉTODOS	TAMANHO RELATIVO	LOC	LOC
TOTAL ADICÃO BASE (BA) => => => => =>					

NOVO OBJETO	TIPO*	MÉTODOS	TAMANHO RELATIVO	LOC (Novo	Reutizado*)
Matrix	Dados	13	Médio	115	136
Linear System	Calc.	8	Grande	197	226
Linked List	Dados	2	Grande	*49	54
Control	Lógico	2			48
TOTAL NOVOS OBJETOS (NO) => => => => => => => =>				361	464

OBJETOS REUSADOS

Linked List	73	73
Data Entry	96	96

*L-Lógico, I-E/S, C-Cálculo, T-Texto, D-Dados, S-Instalação

TOTAL REUSADOS (R) => => => => => => => => => =>		169	169
		TAMANHO	TEMPO
LOC Tamanho Estimado (E):	E = BA+NO+M	366	
Parâmetros de Regressão:	β_0 Tamanho e Tempo	62,0	108,0
Parâmetros de Regressão:	β_1 Tamanho e Tempo	1,3	2,95
Estimativa LOC Novas e Alteradas (N):	$N = \beta_0 + \beta_1 * E$	538	
Estimativa LOC Total:	T = N + B - D - M + R	1397	
Estimativa novas reusados Total (Soma das * LOC):		49	
Tempo Total Estimado de Desenvolvimento: Tempo = $\beta_0 + \beta_1 * E$			1186
Intervalo de Predição:	Intervalo	235	431
Intervalo de Predição Superior:	UPI = N + Range	773	1617
Intervalo de Predição Inferior:	LPI = N - Range	303	755
Porcentagem do Intervalo de Predição:		90%	90%

Usando o projeto conceitual, o estudante nomeou cada objeto novo e determinou o seu tipo. O primeiro objeto, Matrix, era do tipo de Dados. O estudante a seguir estimou quantos métodos ou procedimentos este objeto provavelmente conteria, neste exemplo, 13. A seguir, o estudante julgou o tamanho relativo deste objeto como médio e determinou do seu banco de dados histórico para objetos C++ que um objeto de dados médio teria 8.84 LOC por método. Multiplicando o número de métodos, resultou um 114.9 LOC. O estudante, então, repetiu este procedimento para cada objeto novo para dar um total de 361 LOC de objeto novas.

Programa base

Ao mesmo tempo em que se está estimando os objetos novos, também se determinam o tamanho do programa básico que se está aumentando e qualquer mudança dele. A Tabela 4-1 mostra que o Estudante 12 identificou 695 LOC de código básico ao mesmo tempo 5 das quais ele planejou modificar.

Objetos reusados

Se for possível achar objetos ou procedimentos disponíveis que poderiam prover as funções requeridas pelo projeto conceitual, é possível reusá-los. No exemplo da Tabela 4-1, o estudante 12 identificou dois objetos reusados com um total de 169 LOC.

O método PROBE considera dois tipos de reusados. Os primeiros são os objetos reusados tomados da biblioteca de reuso. Os segundos, os objetos reusados novos, são objetos novos que se planeja desenvolver. São identificados como objetos reusados quando são suficientemente gerais para serem postos dentro da biblioteca de reuso.

Cálculo de LOC objetos estimadas

Começando no topo do exemplo da Tabela 4-1, entrar com os vários totais. A base (B) é igual a 695 LOC, apagadas (D) é 0, e modificadas (M) é 5. Adições Básicas
(BA) é igual a 0. Há três Objetos Novos (NO) daquele total de 361 LOC, e 49 LOC destas são novas reusadas. Assim E = 0 + 361 + 5 = 366.

Regressão linear

Uma vez que se tenham as LOC objeto, é necessário um modo para calcular o tamanho total do programa. Um modo simples de fazer isto é olhar para o histórico de desenvolvimento pessoal. Se, por exemplo, os dados históricos dos programas acabados sempre são 25 por cento maiores que o total estimado, seria somado 25 por cento para se obter a estimativa para o programa finalizado. Tudo o que os cálculos de regressão linear fazem é calcular uma fórmula para fazer esta conversão. A fórmula de regressão linear para fazer este cálculo é:

Tamanho de programa = ß0 + LOC Objeto estimadas * ß1

Ou, mais geralmente:

$$y_k = ß\,0 + x_k\,ß\,1$$

Figura 4-5. Tamanho do Programa.

Os parâmetros de estimativa ß0 e ß1 são calculados dos dados históricos usando as seguintes equações:

$$ß1 = \left(\,?_{i=1}^{n}\, x_i y_i - n x_{avg} y_{avg}\right) / \left(\,?_{i=1}^{n}\, x_i^2 - n(x_{avg})^2\right)$$

Figura 4-6. Parâmetro de Estimativa ß1. *

$$ß0 = y_{avg} - ß1 x_{avg}$$

Figura 4-7. Parâmetro de Estimativa ß0.

Por exemplo, aqui x_1 seria o tamanho de objeto estimado originalmente para o programa 1 e y_1 seria o tamanho do programa terminado total do programa 1. Semelhantemente, x_2 e y_2 seriam os dados para o programa 2, e assim sucessivamente. Também, x_{avg} é a média de todos os termos x_i e y_{avg} é a média de todos os termos y_i.

Determinando o tamanho estimado do programa.

O produto acabado conterá mais que os objetos há pouco calculados. Por exemplo, as estimativas de objeto não incluem a rotina principal ou declarações de código de cabeçalho. Para estes, aplica-se algum fator baseado na experiência histórica do desenvolvedor. O script de estimativa PROBE, na **Tabela A-27,** descreve como obter os parâmetros de regressão linear ß0 e ß1 para fazer isto.

* " ? " Simboliza a soma de i = 1 para x1 * y1 até o valor n (? = ∑)

Intervalo de predição

Uma vez feita uma estimativa, é necessário avaliar a sua qualidade. Usando dados históricos e algo chamado t, ou distribuição t-Student, pode-se calcular o intervalo de predição. Este intervalo fornece a gama ao redor da estimativa, dentro da qual é provável que o tamanho do programa atual caia. A fórmula para o intervalo de predição é:

$$\text{Gama} = t(a/2, n\text{-}2)s \sqrt{\left(1 + 1/n + \left((x_k - x_{avg})^2 \Big/ \sum_{i=1}^{n}(x_i - x_{avg})^2\right)\right)}$$

Figura 4-8. Intervalo de Predição.

Aqui, os termos x_i são novamente os números de LOC objeto estimado em cada programa no banco de dados histórico. O termo x_{avg} é a média de LOC objeto estimado nestes mesmos programas. O termo x_k são as LOCs objeto estimadas no programa novo, e *n* é o número de programas no banco de dados. **a/2** refere-se à porcentagem usada para o intervalo de predição, como 70 ou 90 por cento. Note-se que tabelas estatísticas padrões dão a distribuição de *t* de um lado só, embora a fórmula gama use os valores dos dois lados da distribuição *t*. A distribuição de dois lados é indicada pelo parâmetro **a/2** na expressão *t*.

O termo **S** na equação é o desvio padrão dos dados ao redor da linha de regressão. É calculado como mostrado na equação abaixo, usando os parâmetros ß0 e ß1.

$$\text{Variância} = s^2 = (1/(n\text{-}2)) \; {}^{n}? \; {}_{i=1} (y_i - ß0 - ß1)^2$$

Figura 4-9. Variância. *

$$\text{Desvio padrão} = \sqrt{(\text{Variância})} = s$$

Figura 4-10. Desvio Padrão.

* " ? " Simboliza a soma de i = 1 para y1 até o valor n (? = ∑)

Agora é possível calcular o valor da Gama. O intervalo de predição é então o valor estimado calculado para LOCs do programa mais ou menos esta gama de intervalo.

Categorias de Objetos

O método de estimativa PROBE requer que se tenha dados históricos dos tamanhos dos objetos desenvolvidos e que estes dados sejam divididos em categorias. A seguir são descritos métodos para dividir esses dados.

São necessárias categorias de tamanho de objeto para se ter uma estrutura para julgar o tamanho dos objetos novos no produto planejado.

Como se está principalmente interessado no tamanho relativo dos objetos, com base no julgamento da sua complexidade funcional, é útil normalizar o tamanho dos objetos dividindo o total de LOC objeto pelo número de métodos em cada objeto. Agora um objeto complexo com um método pode ser distinguido de um objeto simples com muitos métodos.

Gamas de tamanho de objeto

Para julgar o tamanho relativo de objetos, usa-se o desvio padrão:

$$\text{Variância} = s^2 = (1/n) \; {}^{n}? \; {}_{i=1} (x_i - y_{avg})^2$$

* $$\text{Desvio padrão} = \sqrt{(\text{Variância})} = s$$

Onde n é igual ao número de itens, x_i é igual ao número de LOCs por método e x_{avg} é igual à média.

Exemplo:

* " ? " Simboliza a soma de i = 1 para x1 até o valor n (? = ∑)

Tabela 4-2. LOC por Método e Desvio Padrão de Objetos Texto em Pascal.

Nome do Objeto	Numero de Metodos	LOC Objeto	LOC por Método	$(LOC-LOC_{avg})^2$
each_line	3	31	10,333	93,494
each_char	3	18	6,000	196,072
list_clump	4	87	21,750	3,054
character	3	87	29,000	80,954
single_character	3	25	8,333	136,171
string_read	3	18	6,000	196,072
list_clp	4	89	22,250	5,051
char	3	85	28,333	69,402
single_char	3	37	12,333	58,817
converter	10	558	55,800	1,281,456
string_manager	4	82	20,500	0,247
string_builder	5	82	16,400	12,978
	10	230	23,000	8,985
Total			260,033	2.142,753
Média			20,003	
Variância = Total/n				164,827
Desvio Padrão = (Variância)^2				12,839

No caso acima, o desvio padrão é de 12,839, o que significa que os pontos centrais da gama de tamanho são os seguintes:

Muito Pequeno (VS) = -5.68 (=20 – 2*12,839)
Pequeno (S) = 7.16 (=20-12,839)
Médio (M) = 20.0 = 20
Grande (L) = 32.84 (=20+12,839)
Muito Grande (VL) = 45.68 (=20+2*12,839)

O método anterior pode resultar em número negativo de LOCs, então usa-se o método seguinte:

1. Calcular o logaritmo natural do valor de LOC por método para cada um dos 13 objetos. Por exemplo, o logaritmo natural do primeiro objeto, each_line, 10, 333, é 2,335.
2. Calcular a média dos valores logarítmicos. Neste caso, a média é 2.802.
3. Calcular a variância dos valores logarítmicos ao redor do valor do meio ou média. Faz-se isto como segue:

- Para cada termo, calcular o quadrado de sua distância do meio; por exemplo, para o primeiro termo:

$$[\ln 10{,}333 - (\ln x)_{avg}]^2 = (2{,}335 - 2{,}803)^2 = (0{,}467)^2 = 0{,}2173$$

Notar que 2,802 não é o log da média, pelo contrário a média dos logs.

- Somar todos os valores quadrados para dar 5,2348.
- Dividir pelo número de objetos, 13, para dar a variância de 0,4027.
- Tomar a raiz quadrada da variância, 0,4027, para obter o desvio padrão, 0.6346.

4. Usando estes valores logarítmicos para média e desvio padrão, calcular os logaritmos dos pontos centrais da gama de tamanho. Exemplo:

$\ln(VL) = avg_{\ln} + 2*$ DesvioPadrao $= 2{,}802 + 1{,}269 = 4{,}071$
$\ln(L) = avg_{\ln} + 1*$ DesvioPadrao $= 2{,}802 + 0{,}635 = 3{,}437$
$\ln(M) = 2{,}802$
...

5. Tomar os anti-logaritmos destes para obter a os pontos centrais da gama de tamanho de objeto em LOCs. Exemplo:

$e^{4,071} = 58{,}62$ LOC
$e^{3,437} = 31{,}09$ LOC
$e^{2,802} = 16{,}48$ LOC
..

Esse método funciona porque os valores dos logs dos tamanhos dos objetos são mais de perto distribuídos normalmente do que os valores de tamanhos de LOCs.

Inicialmente, com poucos dados, devem-se fazer esses cálculos para todos os objetos. Quando houver dados suficientes para uma distribuição por categorias, devem-se dividir os dados em grupos e fazer os cálculos para cada grupo.

4.3.2.1 Distribuição de tamanho de objeto

Deve-se balancear as estimativas de forma a que os tamanhos das categorias se conformem mais ou menos à distribuição normal, conforme a figura a seguir.

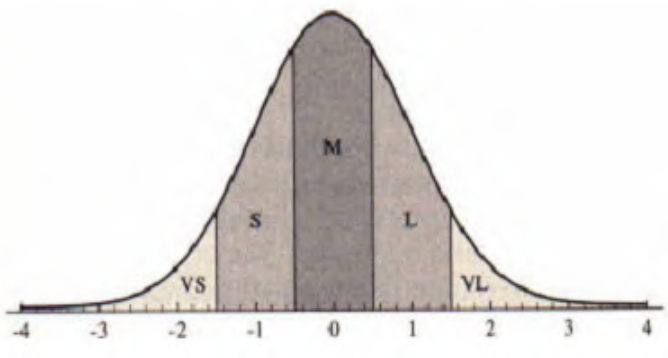

Figura 4-11. Distribuição normal com gama de tamanho

Os valores da gama são os seguintes:

- 6.68 por cento devem ser muito pequenos,
- 24.17 por cento devem ser pequenos,
- 38.3 por cento devem ser médios,
- 24.17 por cento devem ser grandes,
- 6.68 por cento devem ser muito grandes.

Se a estimativa tiver tal distribuição, seria típico dos programas do banco de dados de estimativa de tamanho do desenvolvedor.

Usar estes números apenas como guia geral. Se for muito diferente das estimativas atuais, reexaminar para ver se há algum erro.

4.3.2.2. Considerações sobre estimativas

Apesar de, geralmente, a taxa de erros do estudante ao fazer estimativas parecer não melhorar com os exercícios do PSP, é possível melhorar a habilidade de estimativa geral mesmo que cada estimativa esteja substancialmente errada.

À medida que se aprende a usar dados de tamanho históricos, a qualidade deve melhorar. Ao quantificar a experiência de estimar, é possível fazer melhores estimativas e compensar as tendências pessoais. Por exemplo, se o engenheiro sempre estima para menos, ele logo irá reconhecer esta tendência e começar a corrigi-la. Com o feedback de cada estimativa, é possível ajustar o processo de estimativa para reduzir gradualmente as tendências.

Quando os parâmetros ß0 e ß1 convergirem para valores estáveis, pode-se parar de recalculá-los a cada estimativa. À medida que o processo evoluir, contudo, é preciso atualizar os cálculos estatísticos para representar as práticas correntes.

Quando os parâmetros de regressão linear parecerem inaceitáveis, deve-se usar os valores médios nas estimativas. Por exemplo:

LOC (de 4 programas) estimadas: 427
LOC (atual desses 4 programas): 583
Taxa = 583/427 = 1,365
LOC estimadas (do programa atual): 137
LOC estimadas corrigido pela taxa: 187.

Antes de 3 conjuntos de dados históricos não de seve usar o PROBE. Nesses casos, calcular os parâmetros ß0 e ß1 com o método da média.

Só é possível estimar corretamente o tamanho do produto após completar o projeto.

Tendências

Em grandes projetos, deve-se adquirir o hábito de fazer novas estimativas de tamanho e recursos ao completar o projeto e ao completar o código, e comparar essas valores na fase de Postmortem.

Selecionando um nível de abstração

Quanto menor for a abstração, maior será o número de pontos de dados para cada projeto. É preciso cuidado ao selecionar o nível de abstração, pois se um o bjeto típico tem métodos com uma média de 10 a 15 LOCs e existem de 3 a 10 métodos por objeto, então os objetos variarão de 30 a 100 LOCs cada. Quando o PROBE for usado em um sistema de 50.000 LOC, será necessário categorizar, identificar e taxar a complexidade de cerca de 400 a 600 objetos. Isto pode dar muito trabalho.

Com o PROBE, um programa de 5.000 LOCs pode ser estimado em 1 ou 2 horas. Então, 50.000 LOCs podem ser estimadas em 100 horas, ou 13 dias, mais ou menos. Entretanto, um programa desse tamanho deve levar muitos meses de programação.

Grandes Sistemas: nestes casos, pode ser desejável usar construções de alto nível como proxies, agrupando grupos de objetos em classes. Entretanto, isso pode comprometer as vantagens estatísticas de usar rotinas suportadas por estimativas de dados, devido às individualidades (diferenças) dos objetos. Uma abordagem prática para resolver isso é dividir o programa em diversas partes principais, estimando cada parte e usando o nível de abstração de objetos.

4.3.3. PSP1 - Conteúdos do processo

O PSP1 introduz a estimativa de tamanho ao PSP. Antes de começar um desenvolvimento de software do PSP1, o método de estimativa de tamanho PROBE deve ser usado para estimar o tamanho das LOCs novas e alteradas no novo programa.
Também, devem ser estimadas as LOCs base, reusadas, apagadas, modificadas, adicionadas e total de novas e mudadas. Notar que quando o PSP é usado primeiramente, não existirão

dados históricos para fazer estimativas de tamanho. Assim, deve-se estimar as LOC objeto e deve-se usar o próprio julgamento para obter o total de LOCs novas e mudadas. Para o segundo e o terceiros programas escritos com o PSP1, é possível usar os dados do primeiro e do segundo para guiar as estimativas. O Script de Estimativa PROBE da Tabela A-27 guia esses passos.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Em adição aos objetivos do PSP0.1, o PSP1 é planejado para estabelecer um procedimento ordenado e repetível para estimativas de tamanho de desenvolvimento de software. À medida que este processo é usado para calcular o desenvolvimento de programas, será construída uma base crescente de dados estimados que devem ajudar a fazer estimativas de tamanho progressivamente mais precisas.

As condições prévias para o PSP1 são o PSP0.l e o programa 3A. É necessário, também, ter os dados de tamanho atuais de pelo menos três programas previamente desenvolvidos.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP1 - Script de Processo	Tabela A-21
PSP1- Script de Planejamento	Tabela A-22
PSP1 - Script de Desenvolvimento	Tabela A-23
PSP1 - Script de Postmortem	Tabela A-24
PSP1 – Resumo do Plano de Projeto e Instruções	Tabelas A-25 e A-26
Script de Estimativa PROBE	Tabelas A-27
Modelo de Relatório de Teste e Instruções	Tabela A-28 e A-29
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-30 e A-31
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSO NOVOS

Modelo de Relatório de Teste. O Modelo de Relatório de Teste é mostrado na Tabela

A-28 e suas instruções na Tabela A-29. É usado para registrar dados dos testes. Ao incorporar programas pequenos em programas maiores, fazer modificações, corrigir problemas, ou ao reusar os programas, será útil re-exibir os testes previamente completados. Para fazer isto, é necessário saber quais testes foram feitos, quais dados de teste foram usados e os resultados que foram obtidos.

Isso pode ser útil ao se modificar um programa para incorporar uma nova função. Como parte dos testes, o desenvolvedor poderá se assegurar de que as mudanças não causaram problemas com funções que já funcionavam previamente. Tais problemas são chamados de regressões. Problemas de regressão são comuns quando são feitas muitas mudanças em grandes programas com multi versões. Um teste de regressão é mais facilmente e efetivamente feito rodando de novo os testes que funcionavam previamente. Porém, isto é difícil se não foram salvos os dados de teste adequados.

4.4. PSP1.1 – PLANEJAMENTO DE TAREFAS E TEMPO

4.4.1. Estimando recursos e horários

É possível produzir planos melhores, por menos tempo, estruturando o processo de planejamento, juntando dados históricos e usando métodos efetivos. À medida que o desenvolvedor ganha experiência de planejamento, sua confiança em suas estimativas lhe dará a convicção para defender e vender seus planos. Esta melhora da habilidade de planejamento também melhorará as relações de trabalho com os colegas e gerentes e o tornará um engenheiro de software mais consistente.

O processo de planejamento inicia com uma estimativa de tamanho. Então, os recursos exigidos são estimados e um cronograma é produzido. Para fazer isso, relacionam-se as horas de desenvolvimento usadas em projetos anteriores às estimativas de tamanho atuais. Como todo projeto é diferente, se todas as atividades de projeto forem amontoadas em algum

número de produtividade global, muito do pensamento exigido para produzir um bom plano será eliminado. Assim, é importante refletir sobre o que é requerido exclusivamente para cada projeto. Abaixo se resume o procedimento de estimativa de tempo de desenvolvimento.

Estimando o tempo de desenvolvimento:

Basicamente depende da quantidade e qualidade dos dados históricos:

Produtividade:

- opção 1: usar o cálculo de regressão sobre os dados de LOC objeto estimadas e horas atuais (método PROBE).
- opção 2: usar LOCs atuais novas e alteradas e horas de desenvolvimento atuais para fazer o cálculo de regressão.
- opção 3: existem dados históricos, não existem dados de LOC objetos estimadas e não existem dados para regressão: calcular a produtividade em LOCs por hora.
- opção 4: julgamento.

Tempo:

Após calcular a produtividade, calcular o tempo requerido para o programa, dividindo as LOCs alteradas e novas estimadas pela taxa de produtividade.

Intervalo:

Na opção 3, calcular o maior e o menor tempo usando os dados históricos, ou seja, calcular a produtividade do maior tempo e do menor tempo, e aplicar essas duas taxas de produtividade ao tamanho estimado do programa atual. Nas opções 1 e 2 fazer os cálculos do intervalo pelo método do Intervalo de Predição (UPI e LPI).

Antes de se poder fazer um cronograma, é necessário um plano de recursos detalhado. Geralmente, esta seria uma estimativa de quantas horas diretas se espera gastar em cada tarefa. É preciso, também, projetar o tempo direto total que estará disponível. Esta projeção tem que permitir os outros compromissos e as muitas atividades subordinadas que preenchem um dia de trabalho normal. Os engenheiros freqüentemente acham que eles só gastam aproximadamente 50 por cento do seu tempo em seus trabalhos diretos de projeto.

Com uma estimativa do tempo direto disponível e um plano de recursos detalhado, é possível calcular quando cada tarefa começará e quando terminará. Então são fixadas as datas de planejamento para os marcos de projeto fundamentais. Estas datas provêem uma base sã para se fazer compromissos.

O monitoramento de valor merecido (ou ganho) é um modo para se avaliar o progresso do projeto. Com ele é estabelecido um valor relativo para cada tarefa e creditado esse valor (valor merecido) quando a tarefa é completada.

O sistema de valor merecido "porcentagem do tempo total estimado para o programa, atribuído à uma tarefa" provê uma escala de valor comum para cada tarefa, independentemente do tipo de trabalho envolvido. São estimadas as horas totais para fazer o trabalho inteiro e fornecido para cada tarefa um valor de planejamento, baseado em sua percentagem estimada deste total. O crédito de valor merecido é dado, então, a essa tarefa, quando completada. Tarefas parcialmente completadas não adquirem nenhum crédito.

4.4.2. PSP1.1 - Conteúdos do processo

O processo PSP1.1 introduz a estimativa e planejamento de recursos e horários.
Quando combinado com o método de estimativa de tamanho PROBE - introduzido com o PSP1 - e com suficientes dados de tamanho e custo, será possível, então, fazer melhores planos de desenvolvimento e julgar a precisão desses planos.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Em adição aos objetivos do PSP1, os objetivos do PSP1.l são introduzir e
praticar métodos para:

- fazer planos de recursos e horário,
- monitorar o desempenho pessoal contra estes planos, e
- julgar datas prováveis de conclusão de projeto.

A condição prévia para o processo PSP1 .1 é o PSP1 e a presente seção.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP1.1 - Script de Processo	Tabela A-32
PSP1.1 - Script de Planejamento	Tabela A-33
PSP1.1 - Script de Desenvolvimento	Tabela A-34
PSP1.1 - Script de Postmortem	Tabela A-35
PSP1.1 – Resumo do Plano de Projeto e Instruções	Tabelas A-36 e A-37
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-38 e A-39
Modelo de Planejamento de Horário e Instruções	Tabelas A-40 e A-41
Script de Estimativa PROBE	Tabela A-27
Modelo de Relatório de Teste e Instruções	Tabelas A-28 e A-29
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-30 e A-31
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

NOVOS ELEMENTOS DE PROCESSO

Os dois novos elementos de processo incluídos com o PSP1.1 são o Modelo de Planejamento de Tarefa e o Modelo de Planejamento de Horário. Estes são tipicamente usados para projetos que levarão vários dias ou semanas para terminar.

A seção sumária do Resumo de Plano de Projeto foi ampliada para incluir estatísticas de processo básicas para este programa e para o trabalho de desenvolvimento até a data. Conseqüentemente, o desenvolvedor pode comparar seu recente desempenho com sua média histórica e julgar a racionalidade dos seus planos.

4.4.3. Medidas no PSP

Para ajudar a entender e melhorar os processos, dados são coletados e analisados. Para poder fazer perguntas inteligentes sobre o seu processo, o engenheiro precisa de um modelo de processo. Uma vez que a definição do processo de software provê o modelo de obtenção de dados, ao menos uma definição de processo básica deve sempre preceder a coleta dos dados. Esta é a razão pela qual o processo de software é definido antes de se juntar dados extensos.

As medidas usadas para colher dados caem em uma de três categorias: produto, processo e recurso. Medidas de produto referem-se ao volume do produto produzido. Medidas de processo relacionam eventos às fases de processo, isto é, quantificam o comportamento do processo (contagem de eventos, medição de tempo, etc). Recurso mede principalmente o tempo do programador (horas de trabalho). Embora a medida comum de recursos seja de meses ou semanas, Humphrey reafirma aqui que o tempo pessoal deve ser medido em minutos (para o controle das distrações, como telefonemas, cafezinhos, entre outros).

Objetivos gerais da obtenção de dados no PSP:

- entender como o processo funciona;
- determinar os passos a serem tomados para melhorar a qualidade do produto;
- determinar o impacto da mudança de processo na produtividade pessoal;
- estabelecer *benchmarks* para medir a melhoria do processo;
- ajudar a fazer planos mais acurados.

Medidas apropriadas são importantes para melhorar a estimativa, prover objetivos e monitorar a informação. Para alcançar validade, as medidas do PSP foram definidas usando a abordagem GQM, que tenta definir medidas válidas identificando as metas da medida; as questões que investigam realização dessas metas; e medidas que possam responder a essas perguntas.

O paradigma Métrica-Questão-Objetivo (Goal-Question-Metric-GQM) pode ajudar a projetar e implementar um programa de medida. Dados são coletados para conhecer as metas

específicas e responder a questões explícitas. Estes dados também devem ser precisamente definidos, consistentemente obtidos e corretamente usados.

Exemplo de GQM:

- Objetivo: produzir programas sem defeitos.
- Questão: como produzir software de tal qualidade que não sejam encontrados defeitos em testes ou usos posteriores?

As ajudas requeridas para, manualmente, colher, analisar e usar dados do PSP são os formulários, bancos de dados, planilhas eletrônicas e relatórios sumários. Ferramentas automatizadas podem ajudar a fazer estas coletas de dados e análises mais convenientes, oportunas e precisas. Porém, como o julgamento é envolvido ao colher a maioria dos dados de processo, ferramentas para colher automaticamente estes dados não são prováveis no futuro próximo.

Por exemplo, quando o desenvolvedor pára de digitar, o que está acontecendo? Ao mudar o código, ele está corrigindo um defeito ou melhorando o programa? Onde o defeito foi injetado?

A obtenção de dados sobre o processo de software causa impacto no desempenho. Também pode ser uma tarefa consumidora de tempo e tediosa. Assim, se não se está comprometido a juntar e usar os dados, os dados não serão coletados ou serão incompletos e inexatos.

Ao começar a usar o processo pessoal, provavelmente haverá curiosidade para saber se o desempenho está melhorando. Embora esta seja uma preocupação válida, é quase impossível de resolver sem dados significativos. Somente é possível conhecer as melhorias ao se examinar estatisticamente um volume grande de dados.

Os dados de desempenho pessoal podem ser desencoraja-dores. Os resultados variarão, e, à medida que se esforce para melhorar, eles podem não demonstrar nenhuma melhora consistente. Para alcançar uma tendência de melhora consistente, é necessário fazer mudanças específicas no processo pessoal.

Um processo básico fornece uma base para analisar futuros resultados e determinar onde e como se está fazendo progresso. Fazer isto ajuda a planejar o trabalho e justifica os esforços de melhoria de processo.

4.5. PSP2 - MELHORANDO A QUALIDADE

4.5.1. Revisão de projeto e código

O propósito das revisões é assegurar que os programas produzidos sejam da mais alta qualidade. Dos muitos tipos de revisões que podem ser executadas, as principais são as inspeções, *walk-throughs* e revisões pessoais. Podem ser usadas revisões nos requerimentos, projeto, documentação ou qualquer outro elemento de produto. Esta seção trata de revisão de projeto e código.

Muitos projetos de software gastam quase a metade do seu tempo de desenvolvimento em testes. Isto é muito ineficiente. A revisão de projeto e código é um modo muito mais eficiente para achar e corrigir defeitos. Com as revisões, é possível achar os defeitos diretamente, enquanto que em testes obtém-se somente os sintomas. A diferença crucial entre estas duas abordagens é chamada depuração. Ao revisar um programa, o engenheiro sabe onde está e o que, supostamente, a lógica está fazendo. Assim, as correções serão, provavelmente, mais completas e corretas.

As revisões do PSP irão obter um retorno maior que qualquer outra coisa que se possa fazer. Cada programa deve ser lido, estudado e entendido. Devem ser corrigidos os defeitos de lógica, estrutura e clareza. Quando algumas áreas estiverem obscuras ou confusas, devem-se adicionar comentários ou, melhor ainda, fazer uma reescrita completa. O engenheiro deve fazer os programas fáceis de ler e entender.

Deve produzir algo que ele ficaria orgulhoso de publicar ou mostrar a seus amigos e colegas.

Os três princípios básicos das revisões pessoais são: estabelecer metas de revisão, seguir um processo disciplinado e medir e melhorar este processo. A decisão para fazer revisões é

dirigida pelo desejo de ser produtivo e produzir produtos de qualidade. Para saber se as revisões o estão ajudando a alcançar estas metas, o desenvolvedor precisa medi-las.

Aqui é onde se usa uma distribuição de PARETO para estabelecer prioridades de revisão e desenvolver uma lista de conferência de revisão. Seria sábio reexaminar esta distribuição de PARETO periodicamente, para assegurar-se que ainda se está focalizando corretamente os defeitos mais significativos do processo.

Princípios de revisão de Projeto:

- produzir projetos que possam ser revisados;
- seguir uma estratégia explícita de revisão;
- revisar o projeto em estágios;
- verificar se a lógica implementa corretamente os requerimentos.

Embora o rendimento dê a melhor medida da qualidade da revisão, não se pode calculá-lo antes do desenvolvimento estar completo. Assim, é essencial achar medidas atuais ou imediatas que se relacionem ao rendimento, como LOCs revisadas por hora.

4.5.2. Gerenciamento da qualidade de software

À medida que o engenheiro de software trabalha para melhorar a qualidade do software desenvolvido, ele deve se focalizar na habilidade do seu processo para produzir produtos de qualidade. Deve buscar os métodos mais efetivos para localizar os defeitos, como também os modos mais efetivos para os prevenir. Também reconhecer que, quanto mais tempo são deixados no produto, os custos de achar e corrigir defeitos aumentam rapidamente. A melhor estratégia é assegurar que o programa seja da mais alta qualidade ao produzi-lo inicialmente.

Bugs de software, defeitos e erros são só uma pequena faceta da qualidade do software; porém, este é o foco de qualidade do PSP. Os defeitos raramente são a prioridade de qualidade principal dos usuários, mas eles são um foco essencial do PSP porque os defeitos são mais bem controlados ao nível individual. Se os programas elementares em um sistema

tiverem muitos defeitos, o desenvolvimento do sistema inteiro será prejudicado pelo processo demorado e caro de achar e corrigir esses defeitos.

Essencialmente, a qualidade de software é um assunto de economia. Porém, poucas organizações têm os dados sobre os quais fundar bons planos de qualidade. A medida básica de qualidade do PSP é o rendimento - a porcentagem de defeitos removidos antes do primeiro teste ou compilação.

Os componentes principais do custo de qualidade são os custos de falhas, custos de avaliação e custos de prevenção. A medida do custo de qualidade é tipicamente usada para avaliar o desempenho da qualidade das organizações. Quando usada em projetos individuais, mostrará uma variação considerável. Uma medida de custo de qualidade também pode ser usada com o PSP.

Técnicas de *benchmarking* são úteis para comparar processos. Elas podem ser usadas para monitorar processos através do tempo ou para comparar processos. Para fazer isto, é necessário definir medidas de processo que são independentes do processo, mas que reflitam sua capacidade e robustez. Infelizmente, não há nenhum ponto de referência completamente adequado para medir o processo de software. O rendimento do processo, a relação de custo da avaliação para o fracasso (A/FR) e a produtividade são úteis para o benchmarking do processo de software, mas eles não satisfazem completamente os critérios para um benchmark de propósito geral. Porém, eles são os melhores existentes e devem ser usados até que melhores medidas sejam inventadas.

O processo de software pode ser visto como a combinação de dois processos competidores: injeção de defeito e remoção de defeito. O conteúdo de defeitos do produto acabado é então governado pela diferença entre os resultados destes dois processos. Como ao fazer grandes alterações, mudanças relativamente pequenas em qualquer processo podem fazer uma diferença proporcionalmente grande no resultado final. Para administrar efetivamente a qualidade de software, é necessário focalizar ambos os processos – o de remoção e o de injeção.

Embora descobrir e corrigir defeitos sejam extremamente importantes, é uma estratégia inerentemente defensiva. Para fazer melhorias de qualidade significativas, deve-se identificar as causas destes defeitos e dar os passos para eliminá-los. As ações iniciais de prevenção de

defeitos devem resolver a qualidade do projeto e do código e as ações que se podem tomar para assegurar-se que o processo definido é fielmente seguido.

4.5.3. PSP2 - Conteúdos do processo

O processo PSP2 introduz revisões de projeto e código. Ambos melhorarão a produtividade e a qualidade dos produtos. Com o PSP2, são iniciados os cálculos dos intervalos de predição para as estimativas de tamanho e tempo.

OBJETIVOS E PRÉ-REQUISITOS

Além dos objetivos para o PSP 1.1, os objetivos para o PSP2 são:

- introduzir revisões de projeto e código e
- introduzir métodos para avaliar e melhorar a qualidade das revisões.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP2 - Script de Processo	Tabela A-42
PSP2 - Script de Planejamento	Tabela A-43
PSP2 - Script de Desenvolvimento	Tabela A-44
PSP2 - Script de Postmortem	Tabela A-45
PSP2 – Resumo do Plano de Projeto e Instruções	Tabelas A-46 e A-47
PSP2 - Lista de Revisão de Projeto	Tabela A-48
Lista de Revisão de Código	Tabela A-49
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-38 e A-39
Modelo de Planejamento de Horário e Instruções	Tabelas A-40 e A-41
Script de Estimativa PROBE	Tabela A-27
Modelo de Relatório de Teste e Instruções	Tabelas A-28 e A-29
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-30 e A-31
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

4.6. PSP2.1 – MODELOS DE PROJETO

4.6.1. Projeto de Software

Muitos dos problemas históricos do desenvolvimento de software originaram-se de uma expectativa incorreta de que o desenvolvimento deveria começar com firmes e completos requerimentos. Porém, a história demonstra que, para um sistema de software novo, as exigências não serão completamente conhecidas até depois que o trabalho já tenha começado. Então, ao definir um processo de software, o engenheiro tem que reconhecer que os requerimentos provavelmente mudarão. Portanto, assim que possível, devem ser incluídos passos para identificar e solucionar incertezas de requerimentos.

Um das mais difíceis problemas de projeto refere-se ao *trade-off* entre a perfeição do projeto e os custos de desenvolvimento e cronograma. A especificação completa de um projeto de software requer muita informação. Isto inclui definições para as classes e objetos que definem as suas relações, identificação dos dados trocados entre eles, definição dos dados requeridos e das transformações de estado e especificação das entradas e saídas do sistema. A completa e inambígua definição de todo este material geralmente requer uma quantia significativa de documentação.

À medida que o trabalho de projeto de software é feito, deve-se considerar duas diferentes questões: como o projeto é feito? Como o projeto tem que se parecer quando está terminado? O PSP não especifica um método particular de projeto. Porém, ele trata dos critérios de saída das fases de projeto. Para especificar corretamente a perfeição do projeto, as necessidades daqueles que usarão o projeto devem ser consideradas. Então, é possível determinar o que deve ser o produto do projeto.

Os pontos iniciais e finais do processo de projeto podem ser descritos com a ajuda de modelos. Estes modelos asseguram que os dados exigidos estão claramente definidos e concisamente representados. O Modelo de Especificação Funcional descreve as funções executadas por um programa, um objeto ou um procedimento. O Modelo de Especificação de Estado descreve o modelo de estado do programa. Uma completa especificação de estados contém uma descrição de cada estado de objeto e das transições entre eles. O Modelo de Especificação de Lógica usa pseudo-código para descrever precisamente a lógica de

programa. O Modelo de Cenário Operacional (Operational Scenario Template) descreve o comportamento operacional do programa por um ou mais cenários (situações). Seu propósito é auxiliar a visualizar como o programa reage sob diversos cenários típicos do usuário.

Os modelos de projeto provêem, assim, uma base para o estabelecimento de critérios de saída de projeto. Os modelos podem ser usados para, progressivamente, refinar a especificação e o projeto de um software. Começando no nível mais alto, os modelos especificam as funções do programa principal, incluindo o seu relacionamento com o ambiente externo. Faz-se isso com os modelos de Especificação Funcional e Cenário Operacional. O comportamento interno de cada nível é definido com os Modelos de Especificação de Estado e de Lógica.

A figura abaixo mostra uma maneira de visualizar isso. Começando com uma declaração de requerimentos do programa, descreve-se as necessidades do usuário. A seguir esses requerimentos são traduzidos em uma especificação do programa. Como parte dessa especificação, pode-se completar um Modelo de Cenário Operacional e um Modelo de Especificação Funcional para o programa total. Este processo é repetido a cada nível sucessivo de projeto, completando um Modelo de Especificação de Estado e de Lógica para cada módulo ou objeto.

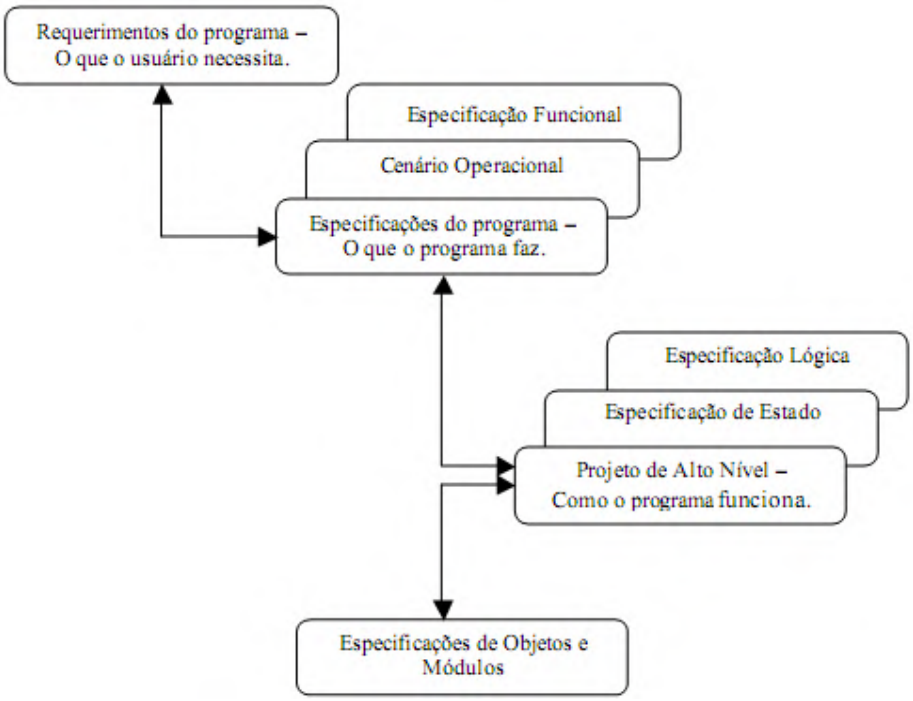

Figura 4-12. Hierarquia de Projeto

No final, como mostra a Figura 4 -12 especificam-se o projeto detalhado para os módulos de programa de nível mais baixo.

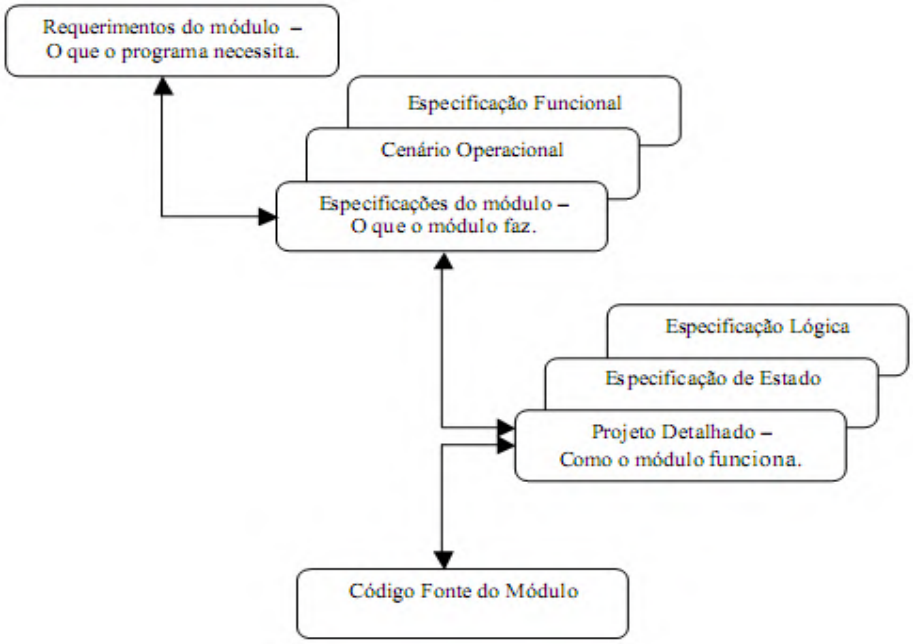

Figura 4-13. Hierarquia de Implementação

Em essência, o projeto por abstração consiste de sucessivas desintegrações do sistema em partes manejáveis. A implementação, então, consiste em construir e integrar estas partes em um todo coerente. Durante o projeto, freqüentemente não é possível definir precisamente todos os subsistemas até que se saiba mais sobre eles. Algumas especificações de alto nível poderiam então ficar incompletas até que seja feito algum projeto de baixo nível. A seguir, o projeto poderia ser feito como uma série desconecta de projetos parciais. O engenheiro pode começar no topo e proceder até que encontre alguma abstração pouco conhecida. Se a abstração se parecesse particularmente difícil ou crítica, ela poderia ser projetada antes de proceder. Aquela abstração, por sua vez, pode invocar outras que se parecem igualmente críticas, e estas podem chamar outras. Como resultado, o engenheiro poderia se achar pesquisando profundamente no sistema antes de haver completado o projeto de alto nível.

Haverá casos nos quais não se poderá especificar um objeto externamente antes de projetá-lo completamente ou, possivelmente, até mesmo construí-lo e testá-lo. Nestes casos, protótipos destes objetos devem ser desenvolvidos antes de se completar os projetos de alto-nível. Depois de completados os protótipos e resolvidas as incertezas, os projetos de alto-nível podem ser completados.

4.6.2. PSP2.1 - Conteúdos do processo

O processo PSP2.1 introduz quatro modelos de projeto que provêem uma estrutura e formato para registrar os projetos. Embora eles não digam como fazer o projeto, podem ajudar a registrar o projeto corretamente quando estiver terminado.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Além dos objetivos para o PSP2, os objetivos para o PSP2.1 são:

- ajudar a reduzir o número de defeitos nos projetos,
- prover critérios para determinar se um projeto está completo, e
- prover uma estrutura consistente para verificar a qualidade dos projetos.

As condições prévias para o PSP2.1 são o PSP2 e a presente seção.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES

PSP2.1 - Script de Processo	Tabela A-50
PSP2.1 - Script de Planejamento	Tabela A-51
PSP2.1 - Script de Desenvolvimento	Tabela A-52
PSP2.1 - Script de Postmortem	Tabela A-53
PSP2.1 – Resumo do Plano de Projeto e Instruções	Tabelas A-54 e A-55
Modelo de Cenário Operacional e Instruções	Tabelas A-56 e A-57
Modelo de Especificação Funcional e Instruções	Tabelas A-58 e A-59
Modelo de Especificação de Estado e Instruções	Tabelas A-60 e A-61
Modelo de Especificação de Lógica e Instruções	Tabelas A-62 e A-63
Lista de Conferência de Revisão de Projeto do PSP2.1	Tabela A-64
Lista de Conferência de Revisão de Código	Tabela A-49
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-38 e A-39
Modelo de Planejamento de Horário e Instruções	Tabelas A-40 e A-41
Script de Estimativa PROBE	Tabela A-27
Modelo de Relatório de Teste e Instruções	Tabelas A-28 e A-29
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-30 e A-31
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

ELEMENTOS DE PROCESSO NOVOS

A única mudança na Lista de Conferência de Revisão de projeto do PSP2.1 é uma adição que se refere aos modelos de projeto introduzidos com o PSP2.1, descritos anteriormente.

4.7. PSP3 – DESENVOLVIMENTO CÍCLICO

4.7.1. Escalando o Processo Pessoal de Software

O tamanho dos projetos de software aumentou rapidamente durante mais de 30 anos. Assim, os produtos desenvolvidos no futuro serão provavelmente muito maiores dos que os de hoje. Como um processo que é ótimo para um programa pequeno não o será, provavelmente, para um processo cinco a dez vezes maiores, provavelmente o processo de desenvolvimento terá que ser mudado.

Tipicamente, a fase de projeto de amplos sistemas de software começa com um esforço de projeto de sistema de alto nível, que divide o produto em componentes. Estes componentes são desenvolvidos separadamente e então o sistema é integrado. Como estes componentes são muito menores do que o sistema total, eles podem, presumivelmente, ser desenvolvidos com métodos de pequena escala.

Os tamanhos de projetos de desenvolvimento de software podem ser divididos em gamas gerais. Estes limites de escalabilidade são largamente determinados pela perícia e habilidade individual:

- Na Fase 0, se trabalha com os blocos de construção menores, como loops, if-then-else, case, etc.
- Quando se combinam as construções da Fase 0 em um programa, move-se para a Fase 1. Na Fase 1, são desenvolvidos os módulos do programa. Estes programas ou módulos são típicos dos cursos iniciais de programação e tem um tamanho que variam de poucas dúzias até alguma centenas de LOCs. Como características dos processos dessa fase podem ser citados: não são escaláveis, usam métodos intuitivos que não possibilitam a construção das habilidades e disciplinas necessárias para o desenvolvimento de trabalhos em larga escala e, normalmente, são usados onde não se aplicam.
- Na Fase 2, são definidos os componentes de programas que contêm vários destes módulos. Agora, em vez de se focalizar nos detalhes de projeto, o desenvolvedor

visualiza as interconexões destes módulos da Fase 1. Os programas são visualizados como abstrações.

- A Fase 3 trabalha com sistemas grandes, compostos de abstrações da Fase l e Fase 2, isto é, abstrai-se a essência, relegando grande parte da complexidade para os componentes de mais baixo nível. A complexidade do sistema é "mascarada", mantendo-se, porém, a qualidade dos componentes. Nesta fase, uma única pessoa não pode compreender os detalhes de todo o sistema.
- Sistemas de software verdadeiramente de larga escala na Fase 4 têm múltiplos subsistemas autônomos, cada qual manipulando responsabilidades do sistema inteiro.

O PSP3 é um exemplo de um processo pessoal de larga escala. Ele pode apoiar razoavelmente desenvolvimentos individuais grandes ou pode servir como uma base para o desenvolvimento de software de larga escala. Sua estratégia é usar um processo cíclico. Uma estratégia de desenvolvimento bem fundada é construída sobre a estrutura natural do produto planejado. Esta estratégia define conteúdos de ciclo como elementos de tamanho pequeno que são projetados separadamente e implementados em ciclos de desenvolvimento. Cada ciclo é progressivamente testado e integrado e, ao fim, se tem o programa integrado e completo, pronto para a integração ou teste de sistema. O processo cíclico usado pelo PSP3 é mostrado na figura abaixo.

4.7.2. PSP3 - Conteúdos do processo

O PSP3 introduz um processo cíclico projetado para ajudar a desenvolver programas maiores. O fluxo conceitual do processo PSP3 é mostrado na Fig. 4-14. O processo começa com planejamento e projeto de alto-nível, seguido por uma série de ciclos de desenvolvimento. As principais preocupações da fase de projeto de alto-nível são produzir um projeto global e uma estratégia de desenvolvimento. Esta estratégia resolve testes, reusos e a estruturação de desenvolvimento de produto em incrementos que são, cada um, satisfatórios para o desenvolvimento com o PSP2.1 - como ciclos de processo.

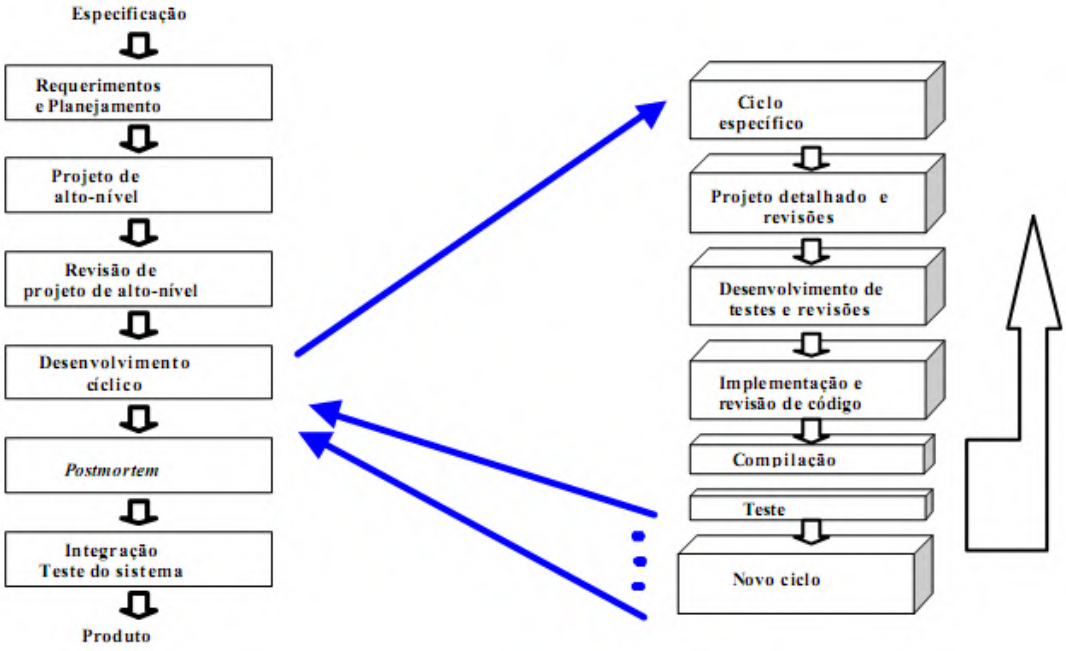

Figura 4-14 – O Processo PSP3

Requerimentos e Planejamento: produz um projeto conceitual do sistema como um todo, estima o tamanho e planeja o trabalho de desenvolvimento.

Projeto de alto nível: identifica as divisões naturais do produto e planeja uma estratégia cíclica. Cada ciclo deve ter entre 100 e 300 linhas de código fonte (novas e alteradas).

Desenvolvimento cíclico: primeiro, estabelece as especificações para o ciclo corrente. Cada ciclo é essencialmente um processo PSP2.1:

- Cada ciclo é a base para o seguinte. Para ser preservada a escalabilidade, cada desenvolvimento incremental deve ser autocontido e livre de erros.
- Os testes devem ser desenvolvidos antes de se escrever o código. A vantagem de desenvolver e planejar testes antes é que isso força a pensar sobre o produto à partir de uma perspectiva de teste.
- Ao reavaliar o trabalho, determina-se o status atual. Caso necessário pode-se fazer ajustes do plano.

OBJETIVOS E CONDIÇÕES PRÉVIAS

Os objetivos do PSP3 incluem os objetivos do PSP2.1 mais um outro: estender a capacidade do processo pessoal para o desenvolvimento de programas de até várias mil LOC. A condição prévia para PSP3 é o PSP2.1.

SCRIPTS, FORMULÁRIOS, MODELOS E PADRÕES.

PSP3- Script de Processo	Tabela A-65
PSP3- Script de Planejamento	Tabela A-66
PSP3- Script de Projeto de Alto Nível	Tabela A-67
PSP3- Script de Revisão de Projeto de Alto Nível	Tabela A-68
PSP3- Script de Desenvolvimento	Tabela A-69
PSP3 - Script de Postmortem	Tabela A-70
PSP3– Resumo do Plano de Projeto e Instruções	Tabelas A-71 e A-72
Resumo de Ciclos e Instruções	Tabelas A-73 e A-74
Log de Controle de Emissão e Instruções	Tabelas A-76 e A-77
Modelo de Cenário Operacional e Instruções	Tabelas A-56 e A-57
Modelo de Especificação Funcional e Instruções	Tabelas A-58 e A-59
Modelo de Especificação de Estado e Instruções	Tabelas A-60 e A-61
Modelo de Especificação de Lógica e Instruções	Tabelas A-62 e A-63
Lista de Revisão de Projeto do PSP3	Tabela A-75
Lista de Revisão de Código	Tabela A-49
Modelo de Planejamento de Tarefa e Instruções	Tabelas A-50 e A-51
Modelo de Planejamento de Horário e Instruções	Tabelas A-52 e A-53
Script de Estimativa PROBE	Tabela A-27
Modelo de Relatório de Teste e Instruções	Tabelas A-28 e A-29
Modelo de Estimativa de Tamanho e Instruções	Tabelas A-30 e A-31
Proposta de melhoria de Processo (PIP) e Instruções	Tabelas A-18 e A-19
Padrão de Codificação	Tabela A-20
Log de Registro de Tempo e Instruções	Tabelas A-7 e A-8
Log de Registro de Defeito e Instruções	Tabelas A-9 e A-10
Tipo de Defeito Padrão	Tabela A-11

NOVOS ELEMENTOS DO PROCESSO PSP3

Além dos scripts do PSP3, os novos elementos de processo introduzidos com o PSP3 são o Log de Monitoramento de Assunto e o Resumo de Ciclo.

Log de Monitoramento de Assunto. O Log de Monitoramento de Assunto (ITL – Issue Tracking Log) provê um conveniente lugar para registrar assuntos, problemas e questões em aberto. No meio de um projeto, por exemplo, um nome de variável poderia ser alterado para representar melhor seu propósito. Em lugar de parar a cada ponto para se assegurar que cada ocorrência deste nome seja mudada ao longo do programa, este item poderia ser anotado no ITL para atenção posterior.

Resumo de Ciclo. Ao usar o PSP3, o desenvolvedor deve planejar implementar programas maiores em módulos de cerca de 100 LOC (esta cifra é arbitrária). Usa-se uma cópia do Resumo de Ciclo para registrar os planos de ciclo e outra para entrar com os resultados de ciclo atuais. Os dados do ciclo são fáceis de obter ao término de cada ciclo, mas são geralmente difíceis de reconstruir depois. No registro do tamanho, por exemplo, pode-se contar as LOCs totais do programa ao término de cada ciclo e determinar quais linhas foram reusadas, adicionadas, apagadas ou modificadas. Mais tarde, isto pode ser muito mais difícil.

É preciso salvar o tamanho planejado e atual, o tempo e os dados de defeito porque cada ciclo do PSP3 é essencialmente um pequeno projeto. Juntando estes dados e monitorando o desempenho pessoal, o desenvolvedor pode planejar e administrar melhor o trabalho. Ao considerar cada passo cíclico como uma tarefa e usar o método de valor merecido para monitorar o progresso do desenvolvimento, o engenheiro pode adquirir um quadro surpreendentemente preciso de seu estado e o cronograma de conclusão provável.

5. CONCLUSÃO

Em seu artigo *The personal Software Process In The Internet Age*, (HUMPHREY 1995) manifesta-se de forma bastante otimista com relação ao PSP, qualificando-o como "o estado da arte na metodologia de engenharia de software". "Provavelmente o mais estruturado e medido processo ou curso de engenharia de software oferecido pelo SEI, livros, indústria ou universidades". Entretanto, como não poderia deixar de ser, o PSP tem qualidades e defeitos. Esta seção discute alguns prós e contras do PSP.

5.1. O CUSTO DO PSP

Aplicar o PSP a uma prática tem um custo significativo e um alto grau de compromisso. Primeiro, leva tempo para se aprender e aplicar os seus métodos. Aprender o PSP pode levar até quatro meses. O trabalho de curso para o PSP tomará uma média de 130 horas. Depois de aprendidos os métodos do processo, ainda levará tempo para se colecionar e analisar os dados. À medida que melhorias de processo são implementadas, elas também levarão tempo para se acostumar. O tempo gasto ao se aplicar o PSP variará, dependendo das ferramentas usadas para aplicar os métodos. No caso de se usar caneta e papel, pode demorar um minuto para anotar cada entrada de tempo e cada defeito. Quando o projeto estiver acabado, pode levar uma hora para se reunir todos os dados e computar as métricas de desempenho.

É importante também a conclusão a que chegou (HUMPHREY 1995): aprender os métodos do PSP toma muito tempo e não pode ser feito em paralelo com um curso de tempo integral.

Secundariamente, o PSP pode "desafiar" os sentimentos. Muitos esperam uma melhoria imediata do PSP, mas não funciona desta maneira. Enquanto que as pessoas que geralmente são novas no processo de software vêem a produtividade aumentada, o engenheiro com um método estável que aprende o PSP experimentará uma diminuição em sua produtividade.

Em tese, conclui-se que, embora o PSP tenha muitos benefícios, ainda há várias faltas que seu usuário terá que superar para receber esses benefícios. Em geral, não deve ser esperado que os usuários de PSP sigam os métodos exatamente como descritos. Ele salienta as vantagens

do uso de formulários de controle de tempo, defeitos, o modelo da PIP, o Modelo de Relatório de Testes, planejamento e revisões. Entretanto questiona alguns erros ou faltas nesses formulários, o método de contagem de linhas de código lógicas como medida padrão de tamanho, o conceito de código reusado e a redundância que alguns modelos propostos p elo PSP poderão ocasionar no trabalho de engenheiros de software com práticas já estabelecidas, entre outros.

O PSP exige uma devoção disciplinada ao dever. Conforme (HUMPHREY 1995) o processo de sete passos para a sua introdução "é uma série de ritos de iniciação crescentemente árduos, onde só os escolhidos podem sobreviver".

Finalmente, o PSP traz um risco de se estar em conflito com a auto-imagem. A pessoa tem que reconhecer as suas forças e nunca deve enfatizar as suas fraquezas.

5.1.1. Os benefícios do PSP

Embora o PSP tenha vários custos, Humphrey argumenta que um número grande de benefícios pode ser percebido quando o processo pessoal da pessoa foi estabilizado. Eles são listados a seguir:

- O programador ganha uma avaliação das suas forças e fraquezas, mostrada nos dados do PSP.
- O programador pode interpolar dados colecionados de tal forma que podem ser derivadas idéias para a melhoria de processos.
- O programador pode resistir a uma pressão irracional por discussão do tamanho antecipado do problema e relacionar isto a sua produtividade histórica.
- A conclusão organizada de um projeto fornece ao programador um senso de realização pessoal.
- Considerando que o programador tem um processo repetível, consistente e estável, ele vai ganhar mais confiança do seu grupo.

Benefícios do PSP para a Organização

O PSP introduz os seguintes benefícios para a organização:

- Dados do PSP melhoram o planejamento e o gerenciamento do cronograma de projetos de software.
- A remoção de defeitos introduzidos antes da compilação e testes resulta em um produto de melhor qualidade, reduz o custo dos testes e diminui o tempo de desenvolvimento.
- O PSP introduz um aprendizado e prática para a melhoria do processo. Pequenos ciclos de desenvolvimento e os dados pessoais tornam fácil o entendimento e aumenta a experiência.
- O PSP ajuda os engenheiros e seus gerentes a aprenderem a prática da quantificação do gerenciamento do processo. Eles aprendem o uso do processo definido e aprendem também a coletar dados para gerenciar, controlar e melhorar o trabalho.
- Finalmente, o PSP expõe os engenheiros a 12 áreas chaves do CMMI (KPAs). Com isso facilita a preparação dos engenheiros a participar na melhoria baseada no CMMI.

5.1.2. Automação: sim ou não?

Considera-se que uma das principais questões sobre o PSP refere-se à sua automação. Embora Humphrey não se mostre favorável ao uso de ferramentas totalmente automatizadas, é preciso atentar para o grande esforço exigido para completar e controlar todos os formulários, scripts e modelos propostos. Além disso, (HUMPHREY 1995) salienta quatro desvantagens importantes na abordagem manual do PSP:

- Usar lápis e papel para executar o PSP é tedioso.
- O ser humano é naturalmente propenso a erro.
- Formulários prontos de papel não permitem muito ao usuário divergir do processo prescrito.
- Uma correção em um formulário forçará o recálculo de muitos campos, que levam muito tempo à mão.

A questão é: a Engenharia de Software deve municiar o desenvolvedor com ferramentas e técnicas efetivas, que na prática signifiquem uma melhora na atividade de desenvolvimento, ou deve o desenvolvedor despender uma quantia significativa de seu tempo com o manuseio de um volume considerável de papéis? Acredita-se na efetividade da primeira questão.

Porém, é preciso atentar para o fato que uma ferramenta mal desenvolvida seria uma inconveniência a seus usuários. Uma ferramenta de apoio do PSP deve ser transparente com respeito à manipulação de tarefas, tal que os engenheiros possam se focalizar nos seus produtos em vez de nos seus processos.

O PSP foi projetado para aqueles desenvolvedores que querem fazer um trabalho de qualidade, seguindo um processo definido ao desenvolver programas. O uso de facilidades apenas incentivará o cumprimento das tarefas propostas.

5.2. ESTUDOS DO SEI SOBRE IMPACTO DO PSP

Em 1997 Will Hayes e James W. Over, fizeram um estudo com 298 engenheiros de software e o resultado deste estudo está disponível no relatório técnico "*An Empirical Study of Impact of PSP on Individual Engineers*" disponível no site do SEI (Instituto de Engenharia de Software).

O relatório descreve o efeito de PSP em dimensões de desempenho fundamentais destes engenheiros, incluindo a sua habilidade para calcular e planejar o trabalho, a qualidade do software que eles produziram, a qualidade do seu processo de trabalho e a sua produtividade. O relatório também discute como melhorias na capacidade pessoal também melhoram o desempenho organizacional em várias áreas: custo e administração de horários, qualidade de produto entregue e tempo de ciclo de produto.

Resultados do estudo:

No estudo foram examinadas cinco dimensões de melhoria de processo pessoais do PSP: acurácia da estimativa de tamanho e esforço, qualidade do produto, qualidade do processo e produtividade pessoal. Foi concluído que o PSP melhorou o desempenho nas primeiras quatro dimensões sem qualquer perda na quinta área, produtividade.

Conclusões gerais do estudo:

Os objetivos do estudo eram examinar o efeito do Processo de Software Pessoal no desempenho de engenheiros de software e considerar se os resultados observados poderiam ser generalizados além dos participantes de estudo. Como o PSP foi desenvolvido para melhorar o desempenho individual pela introdução gradual de práticas novas, o estudo tomou uma abordagem semelhante, examinando a mudança no desempenho individual à medida que estas práticas foram introduzidas.

Foi concluído que as melhorias em quatro dimensões, precisão de estimativa de tamanho, precisão de estimativa de esforço, qualidade de produto e qualidade de processo, eram estatisticamente significativas. Nenhuma mudança estatisticamente significativa na produtividade foi achada. Assim, o estudo declarou que as melhorias observadas nas outras dimensões de desempenho foram alcançadas sem qualquer perda de produtividade.

Concluindo, as análises informadas no relatório substanciam que as tendências no desempenho pessoal, observadas durante o treinamento do PSP, são significativas, e que as melhorias observadas representam real mudança no desempenho individual, não uma mudança no desempenho comum do grupo. Além disso, concluiu-se que as melhorias observadas ocorreram devido ao PSP e podem ser generalizadas.

5.3. FUTURAS PESQUISAS

Há a oportunidade de continuar este trabalho relacionado do Processo de Software Pessoal (PSP), desenvolvendo uma ferramenta para o processo que não seja totalmente automatizada, porém, que oculte alguns pontos fracos do processo e auxiliem em seus pontos fortes.

REFERÊNCIAS

ABNT NBR ISO 9000. Sistemas de Gestão e Garantia da Qualidade – Fundamentos e Vocabulário. Associação Brasileira de Normas Técnicas, 2005.

BEGOSSO, Luiz Ricardo. **AMBIENTE PARA O DESENVOLVIMENTO DE MATURIDADE EM ENGENHARIA DE SOFTWARE EM UM CURSO DE CIÊNCIA DA COMPUTAÇÃO**. 2002. 218p. Tese (Doutorado) – Área de Sistemas Digitais - Universidade de São Paulo, São Paulo, 2002.

HILBURN, T; TOWHIDNEJAD, M. **Software Quality Across the Curriculum**. In: Proceedings of the 15th Conference on Education and Training. IEEE Software Engineering Conference, EUA, 2002.

HUMPHREY, Watts. **PSP: A Self-Improvement Process for Software Engineers**. Editora Addison-Wesley, 2005.

HUMPHREY, Watts S. "**A Discipline for a Software Engennering**", 1ª edição – Addison Wesley: USA, 1995.

JUNIOR, José Barreto. Qualidade de Software – Disponível em: < http://www.barreto.com.br/qualidade >. Acesso em: 21 junho 2012.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. São Paulo: Editora McGraw-Hill, 2011.

RONG, G. et al. **Delivering PSP course in Tertiary Education Environment: Challenges and Solution**. In: Proceedings of the 24th Conference on Software Engineering Education and Training. IEEE Software Engineering Conference, EUA, 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 8 ed. São Paulo: Pearson Addison-Wesley, 2007.

APÊNDICE A – MATERIAIS DO PSP

Tabela A-1. PSP0 Script do Processo.

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP0 Formulário do Plano de Projeto resumida • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões • Cronômetro (opcional)
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Estimar o tempo requerido para o desenvolvimento • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Implementação do design • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
4	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Log de gravação de tempo e registro de defeitos completado.

Tabela A-2. PSP0 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP0 Formulário do Plano de Projeto resumido • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Recursos estimados	• Fazer sua melhor estimativa do tempo requerido para desenvolver este programa
4	Critérios de saída	• Uma documentada declaração de requisitos • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-3. PSP 0 Script de Desenvolvimento.

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tempo de desenvolvimento planejado. • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos
1	Design	• Revisar os requisitos e produzir um projeto para alcançá-los • Registrar tempo no registro de gravação de tempo
2	Código	• Implementar o projeto • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
3	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
4	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
	Critérios de saída	• Um Programa totalmente testado • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-4. PSP 0 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tempo de desenvolvimento planejado. • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 0.1 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 0.1 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 0.1 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado • Resumo do Plano de Projeto concluído. • Log de registro de tempo e defeitos concluídos

Tabela A-5. PSP 0 Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Tempo na Fase (min.)	Plano	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Código				
Compilação				
Teste				
Postmortem				
Total				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois do Desenvolvimento			

Tabela A-6. PSP 0 Resumo do Plano de Projeto Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.
Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação

Tabela A-7. PSP 0 Log de Gravação de Tempo

Estudante ______________________________ Data ______________
Instrutor Programa #

Data	**Inicio**	**Termino**	**Tempo de Interrupção**	**Tempo Delta**	**Fase**	**Comentários**

Tabela A-8. Instruções Log de Registro do Tempo

Proposta	Este formulário serve para registrar os dados gastos em cada fase do projeto. Estes dados são usados para completar o Resumo do Plano de Projeto
Geral	• Registra o tempo total usado no processo • Registrar o tempo em minutos • Ser mais preciso possível Se precisar de um espaço adicional, usar outra cópia deste formulário
Cabeçalho	Entre com o seguinte • Nome, Data Atual, nome do Instrutor e o número do programa Se estiver trabalhando com uma tarefa não programática, entre com a descrição do serviço no campo Programa #
Data	Entre com a data quando a entrada é feita (EX. 13/09/2012)
Inicio	Entre com o tempo de quando a tarefa teve inicio
Exemplo	9:20
Termino	Entre com o tempo de quando a tarefa foi finalizada
Exemplo	10:30
Tempo de Interrupção	Registre a duração de quaisquer interrupção não relacionada a tarefa e a razão para a interrupção Se tiver varias interrupções entre com o tempo total delas
Exemplo	37 – pausa
Tempo Delta	Entre com o tempo total gasto em cada tarefa menos o tempo de interrupção
Exemplo	Das 9:20 as 10:30, menos 37 minutos, ou 33 minutos
Fase	Entre com o nome da fase ou passo sendo trabalhado
Exemplo	Planejamento, codificação, teste, e assim por diante
Comentários	Entre com qualquer comentário pertinente ou quaisquer circunstâncias incomuns desta atividade
Exemple	Houve problema com o compilação, e foi necessário procurar ajuda
Importante	É importante registrar todo o tempo de trabalho. Se esquecer de registrar o inicio o termino ou o tempo de interrupção de uma tarefa, prontamente entre com sua melhor estimativa do tempo

Tipos de Defeitos	
10 Documentação	60 Controle
20 Sintaxe	70 Dados
30 Pacote	80 Função
40 Atributo	90 Sistema
50 Interface	100 Ambiente

Tabela A-9 – Log de Registro de Defeitos

Estudante ______________________ Data ____________

Instrutor ______________________ Programa # ____________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Data	Numero	Tipo	Inserido	Removido	Tempo de Conserto	Defeito Consertado

Descrição ______________________

Tabela A-10. Instruções Log de Registro de Defeitos

Proposta	Manter os dados de cada defeitos encontrada e corrigi-lo Usar estes dados para Completar o Resumo do Plano de Projeto
Geral	• Registrar todos os registro de defeitos encontrados na revisão, compilação e testes • Registrar cada defeito separadamente e completamente Se precisar de um espaço adicional, usar outra cópia deste formulário
Cabeçalho	Entre com os dados • Seu nome • Dia de hoje • O nome do instrutor • O numero do programa
Data	Entre com a data de quando o defeito foi encontrado
Número	Entre com o numero do defeito Para cada este deve ser um numero seqüencial iniciando por exemplo com 001 ou 1
Tipo	Entre com o tipo de defeito de acordo o Padrão de tipos de defeitos na Tabela A-11 Use seu melhor julgamento selecionando o melhor tipo de defeito no qual o mesmo se aplica
Inseridos	Entre com a fase no qual o defeito foi inserto Use o seu melhor julgamento
Removidos	Entre com a fase no qual o defeito foi removido Use o seu melhor julgamento
Tempo de conserto	Entre com seu melhor julgamento do tempo levado para consertar o defeito. Este tempo pode ser determinado usando um cronômetro ou seu julgamento
Defeito consertado	• Se este defeito foi inserido enquanto outro defeito era consertado, insira o numero do defeito impropriamente consertado • Se não conseguir identificar o numero do defeito, entre com um X na caixa de Defeito Consertado
Descrição	• Escrever uma sucinta descrição do defeito clara o suficiente, para que posteriormente seja relembrado o erro, e o porque o cometeu

Tabela A-11. Padrão de tipos de Defeitos

Número do Tipo	Nome do Tipo	Descrição
10	Documentação	Comentários, mensagens
20	Sintaxe	Ortografia, pontuação, erros de digitação
30	Construção, Pacote	Controle de mudanças, bibliotecas, controle da versão
40	Atribuição	Declaração, nomes duplicados, escopo, limites
50	Interface	Referencias e chamadas de procedimentos, E/S, formatos de usuário
60	Controle	Mensagens de erro, checagens inadequadas
70	Dados	Estrutura, conteúdo
80	Função	Lógico, ponteiros, loops, recursão, computação, defeitos de funções
90	Sistema	Configuração, cronometragem, memória
100	Ambiente	Design, compilação, teste, ou outros problemas de manutenção do sistema.

Tabela A-12. PSP 0.1 Script do Processo.

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP0.1 Formulário do Plano de Projeto resumido • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões • Cronômetro (opcional)
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Estimar o total de linhas de código novas e alteradas necessários. • Estimar o tempo requerido para o desenvolvimento • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Implementação do design • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
4	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-13. PSP 0.1 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP0.1 Formulário do Plano de Projeto resumido • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa
3	Recursos estimados	• Fazer sua melhor estimativa do tempo requerido para desenvolver este programa • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas.
	Critérios de saída	• Uma documentada declaração de requisitos • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-14. PSP 0.1 Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa estimado e tempo de desenvolvimento. • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos e produzir um projeto para alcançá-los • Registrar tempo no registro de gravação de tempo
2	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
3	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
4	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-15. PSP 0.1 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa planejado e o tempo de desenvolvimento. • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 0.1 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 0.1 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 0.1 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos

Tabela A-16. PSP0.1 Resumo do Plano de Projeto

Estudante		Data	
Time		Programa#	
Projeto		Linguagem	

Tamanho do Programa (LOC)	***Plano***	***Real***	***Datado***
Base(B)		*(Mensuradas)*	
Apagadas (AP)		*(Contadas)*	
Modificadas (M)		*(Contadas)*	
Adicionadas (A)		*(T-B+AP-R)*	
Reutilizadas (R)		*(Contadas)*	
Total Novas & modificadas (N)		*(A+M)*	
Total Linhas de Código (T)		*(Mensuradas)*	
Total Nova(s) Reutilizada(s)			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Código				
Compilação				
Teste				
Postmortem				
Total				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Tabela A-17. PSP 0.1 Resumo do Plano de Projeto Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Atual. • Usando o seu melhor julgamento, estime o numero de linhas de código novas e modificadas que esperas desenvolver. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R • Calcule o total de novas e reutilizadas linhas de código N = A+M
Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.
Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação

Tabela A-18. Proposta de Melhoria do Processo (PIP)

Estudante _____ Data _____
Instrutor _____ Programa # _____
Processo _____ Elementos _____

Número PIP	Descrição do Problema
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Numero de Proposta PIP	Descrição da Proposta
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Notas e Comentários

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Tabela A-19. Proposta de Melhoria do Processo (PIP) instruções

Propósito	• Para fornecer uma maneira de gravar os problemas do processo e idéias de melhoria • Para fornecer um registro ordenado de suas idéias de melhoria de processos para uso em processos de melhoria posteriores
Geral	Use o formulário PIP como se segue: • Para registrar as idéias de melhoria de processos como elas ocorrem para você. • Para estabelecer prioridades para seu plano de melhorias • Para registrar lições aprendidas e condições incomuns Mantenha o formulário do PIP em mãos enquanto usa o PSP • Registre problemas do processo mesmo sem soluções propostas • Reter os PIPs para uso no processo de melhoria.
Formulário de identificação PIP	• Usar a data e numero do programa para identificar o formulário do PIP
Cabeçalho	• Insira seu nome, a data, o nome do instrutor, e o numero do programa ou designação de outro projeto • Entre com o nome do processo que está usando, tal como PSP 0.1 • Se o PIP refere-se a um elemento do processo especifico, anote o nome do elemento, tal como PSP 0.1 Resumo do Plano
Problema	Descreva o problema mais claramente possível: • A dificuldade encontrada • O impacto no produto, o processo, e si mesmo Numero de problemas em cada forma na coluna da esquerda: • Use uma seqüência números conveniente • Inicie com 1 em cada PIP
Proposta	• Descreva seu processo de melhoria proposto mais explícito possível • Sempre que possível, referencie um elemento especifico do processo, as palavras ou entradas a serem modificadas. • Sempre que necessário, referencie os números de descrições de problemas na coluna da esquerda. • Se achar que uma proposta para melhoria é particularmente importante, descreva sua prioridade e explique porquê.
Notas e comentários	Para cada processo, complete pelo menos o formulário do PIP com comentários gerais sobre o processo • Registre as lições aprendidas do processo • Observe quaisquer condições que precisem ser registradas para posteriormente determinar porque o processo funcionou particularmente bem ou mal

Tabela A-20. C++ Padrão de Codificação

Propósito	Guiar o desenvolvimento de programas em C++
Cabeçalho de programas	Comece todos os programas com um cabeçalho descritivo
Formato de cabeçalho	Guiar o desenvolvimento de programas em C++
Conteúdo listado	Comece todos os programas com um cabeçalho descritivo
Exemplo de conteúdo	/********** */ /* Atribuição do Programa: O numero do programa */ /* Nome: seu nome */ /* Data: inicio do desenvolvimento */ /* Descrição: uma pequena descrição da função */ /* do programa */ /********** */
Instruções de reuso	Fornece um resumo do conteúdo listado
Exemplo de Reuso	/********** */ /* Conteúdo listado: */ /* Instruções de Reuso */ /* Instruções de Modificação */ /* Instruções de Compilação */ /* Inclui */ /* Declaração de Classe */ /* CData */ /* ASet */ /* Código fonte em c:\classes\CData.cpp */ /* CData */ /* CData() */ /* Empty() */ /********** */
Identificadores	• Descreva como o programa é usado. Forneça o padrão de declaração, tipos e valores do(s) parâmetro(s), e limites do(s) parâmetro(s) • Forneça avisos de valores ilegais, condições de overflow, ou outras condições que podem potencialmente resultar em uma operação imprópria
Exemplo de identificadores	/********** */ /* Instruções de Reuso: */ /* int ImprimirLinha(char *linha_de_caracteres) */ /* Propósito: imprimir uma string 'linha_de_caracteres', */ /* em uma linha de impressão */ /* Limitações: o tamanho máximo da linha é LINHA_TAMANHO */ /* Retorno: 0 se pode imprimir, e 1 caso contrario */ /********** */
Comentários	Use nomes descritivos para todas as variáveis, nomes de funções, constantes, e outros identificadores, evitar abreviações ou variáveis com uma letra
Bom	int numero_de_estudantes; /* Isso é BOM */

comentário	float x4, j, ftave; /* Isso é MAL */
Mal comentário	• Documentar o código para que o leitor possa entender o seu funcionamento • Comentários devem explicar tanto a finalidade e o comportamento do código • Comente as declarações de variáveis para indicar sua finalidade
Secções principais	if(registros_contador > limite) /* Todos os registros foram processados? Limite.
Exemplo	if(registros_contador > limite) /* checar se registros_contador maior que limite.
Espaços em branco	Preceder seções principais do programa por um bloco de comentário que descreva o processamento que é feito na próxima seção.
Recuo	• Recuar cada nível de chaves ({ }) do anterior • Chaves de abertura e fechamento devem estar sozinhas e alinhadas umas com as outras

Exemplo Recuo	while(perder_distancia > entrada)
	{
	codigo_sucesso = mover_robo(localização_ponto);
	if(código_sucesso == FALHA_MOVER)
	{
	printf(“O movimento do robo falhou\n”);
	}
	}

Escreva com letra maiúscula	• Escreva em letra de forma todas as constantes • Letra minúscula para outros identificadores e palavras reservadas Mensagens enviadas ao usuário podem um caso misto, de modo à fazer uma apresentação ao usuário limpa
Exemplo	#define NUMERO-ESTUDANTES-PADRAO 15 Int class-size = NUMERO-ESTUDANTES-PADRAO;

Tabela A-21. PSP1 Script do Processo

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP1 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Estimativa histórica de dados e tamanho real • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Use o método PROBE para estimar o total de linhas de código novas e alteradas necessários. • Completar o Modelo de Estimativa de Tamanho • Estimar o tempo requerido para o desenvolvimento • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Implementação do design • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
4	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Modelo de estimativa de tamanho completado • Modelo de Relatório de Teste completado • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-22. PSP1 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP1 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Estimativa histórica de dados e tamanho real • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Produza um projeto de um programa conceitual • Use o método PROBE para fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa • Estime linhas de código base, adicionadas, removidas e reusadas • Complete o Modelo de Estimativa de Tamanho e o Resumo do Plano de Projeto
3	Recursos estimados	• Baseado no tempo requerido por LOC do ultimo programa, faça sua melhor estimativa do tempo requerido para desenvolver este programa • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas.
	Critérios de saída	• Uma documentada declaração de requisitos • O projeto de um programa conceitual • Modelo de Estimativa de Tamanho completo. • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-23. PSP1 Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa estimado e tempo de desenvolvimento. • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos e produzir um projeto para alcançá-los • Registrar tempo no registro de gravação de tempo
2	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
3	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
4	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo • Complete o Modelo de Relatório de Teste com testes conduzidos e os resultados obtidos
	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Modelo de Relatório de Teste completo • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-24. PSP1 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa e o tempo de desenvolvimento. • Modelo de Relatório de Teste completo • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 1 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 1 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 1 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Um Modelo de Relatório de Teste completo. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos

Tabela A-25. PSP1 Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Resumo LOC/Hora	***Plano***	***Real***	***Datado***

Tamanho do Programa (LOC)	***Plano***	***Real***	***Datado***
Base(B)		*(Mensuradas)*	
Apagadas (AP)		*(Contadas)*	
Modificadas (M)		*(Contadas)*	
Adicionadas (A)		*(T-B+AP-R)*	
Reutilizadas (R)		*(Contadas)*	
Total Novas & modificadas (N)		*(A+M)*	
Total Linhas de Código (T)		*(Mensuradas)*	
Total Nova(s) Reutilizada(s)			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Código				
Compilação				
Teste				
Postmortem				
Total				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			

Continuação Tabela A-25			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Tabela A-26. PSP1 Resumo do Plano de Projeto – Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Resumo	Entre com o novo e alterado valor de LOC por hora planejado e real, e para todos os programa desenvolvidos insira no campo Datado
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Plano. • Entre com o valor de novas e modificadas LOC (N) do Modelo de Estimava de Tamanho. • Estimar o numero de LOC adicionadas (A) e modificadas(M), então faça N = A + M. • Estimar o numero de LOC deletadas (D) e reusadas (R) e combinar com o número de LOC da base (B), T = N + B –M –D + R. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R • Calcule o total de novas e reutilizadas linhas de código N = A+M
Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.

Continuação Tabela A-26

Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação

Tabela A-27. Script de Estimativa PROBE

Proposta	Para guiar o tamanho e processo de estimativa de tempo pelo método PROBE
Critérios de Entrada	- Requisitos declarados - Modelo do Tamanho de Estimativa e instruções - Tamanho por dados de item para tipos de membro - Log de Registro de Tempo

Numero Fase	**Atividades**	**Descrição**
1	Projeto Conceitual	Revise os requisitos e produza um projeto conceitual
2	Identificação do Objeto	• Identifique o nome dos objetos necessários para implementar o projeto • Julgue o numero de métodos em cada objeto • Determine o tipo que se enquadra cada objeto (L-Lógico, I-E/S, C-Cálculo, T-Texto, D-Dados, S-Instalação); • Estime o tamanho relativo dos métodos do objeto (S=pequeno, M-Médio, L-grande, VL-muito grande); • Encontre o LOC por método através do banco de dados do objeto para este tipo de objeto, • Calcule o tamanho estimado do objeto como o numero de LOC dos métodos por método • Julgue quais dos objetos provavelmente serão inseridos na biblioteca de reutilizáveis e destaque-os com um * na coluna LOC (Novo Reutilizável*)
3	LOC de outros programas	Siga o Modelo de Estimativa de Tamanho para estimar o LOC da base, removidas, modificadas, adicionadas, reutilizadas, e novas

		reutilizáveis.
4	Estimativa de Base	A – Se não tem dados históricos, use seu julgamento para estimar o numero de LOC novas e alteradas e o tempo de desenvolvimento por LOC novas e alteradas B – Se tem dados reais suficientes no tempo sobre o desenvolvimento e tamanho do programa, estime o numero de LOC novas e alteradas para o novo programa e multiplique isso com seu histórico de LOC novas e alteradas por a taxa de produtividade por hora. C - Se tem dados reais suficientes no tempo sobre o desenvolvimento e tamanho do programa, calcule os fatores de regressão β_0 e β_1 a partir dos dados reais de LOC do objeto e hora. D – Se tem dados suficientes da estimativa de LOC por objeto, calcule dois conjuntos de parâmetros de regressão β_0 e β_1. Um para o LOC do objeto estimado versus as LOC reais novas e alteradas e use estes parâmetros para calcular o numero estimado de novas e alteradas LOC no Modelo de Estimativa de Tamanho. Calcule o segundo conjunto de parâmetros de regressão β_0 e β_1 para estimar as LOC do objeto e horas reais. Use estes parâmetros para estimar o tempo de desenvolvimento planejado.
5	Dados Suficientes	• Para usar o método da media, irá precisar apenas de dados históricos de LOC novas e alteradas e horas atuais de desenvolvimento para um projeto • Para usar o método da regressão, precisa ter ao menos três conjuntos de dados de desenvolvimento históricos. Para ser útil para fins de estimativa, a correlação entre LOC(estimados e reais) e tempo de desenvolvimento deve dar um valor r como este $r^2 >= 0,50$
6	Método da Regressão	• Use o procedimento de regressão linear descrito na subseção 4.3.2. para calcular os parâmetros de regressão β_0 e o β_1 • Selecione os dados de tamanho como linhas de códigos de objetos reais (Caso C no passo 4 abaixo) ou linhas de código de objetos estimados (Caso D no passo 4 abaixo) Nota: Os parâmetros de regressão no modelo de estimativa de tamanho relacionam-se com as adições à base (BA), novos objetos (NO), e modificações à base do programa (M) para o total de novos e alteradas linhas de código (N). Seu conjunto de dados de regressão deve incluir estes mesmos itens.
7	Teste de Parâmetros de Regressão	• A magnitude absoluta de β_0 deve ser substancialmente menor que o tamanho estimado (para estimativas de tamanho) ou tempo de desenvolvimento (para horas estimadas) do programa que se está fazendo estimativas • Para tamanhos estimados, o valor de β_1 deve ser positivo e relativamente próximos de 1,0 • Para horas de desenvolvimento estimadas, o valor de $1/\beta_1$ deve ser relativamente próximo de LOC/Hora históricas.

8	Método da Média	• Atribua o valor de $\beta_0 = 0$ • Atribua o valor de β_1 como segue: $\beta_1 = {}^{n}\sum_{i=1}$ (tempo i) / $^{n}\sum_{i=1}$ (Tamanho i) Tempo: o tempo de desenvolvimento para cada programa no banco de dados Tamanho: linhas de código reais do objeto (caso B no passo 4) para cada programa do banco de dados Nota: Os dados de tamanho no banco de dados de tamanho estimado devem incluir os mesmos itens incluídos no novo programa estimado
9	Estimar	Estime as linhas de código projetadas no Modelo de Estimativa de Tamanho e calcule o tempo de desenvolvimento Horas = $\beta_0 + \beta_1$ * (Linhas de código projetadas mais modificadas)
10	Intervalo de Predição	Caso o método de regressão foi usado, calcular o intervalo de predição para a estimativa como descrito na subseção 4.3.2. Nota: O uso de LOC novas e alteradas e horas atuais de desenvolvimento nos cálculos de regressão são equivalentes a assumir que o tamanho estimado foi exatamente correto. Isto irá geralmente dar um intervalo de predição irrealisticamente pequeno.

Tabela A-28. Modelo de Relatório de Testes

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Nome Teste/Número	
Objetivo do Teste	
Descrição do Teste	
Condições de Teste	
Resultados Esperados	
Resultados Reais	
Nome Teste/Número	
Objetivo do Teste	
Descrição do Teste	
Condições de Teste	
Resultados Esperados	
Resultados Reais	

Tabela A-29. Modelo de Relatório de Testes - Instruções

Proposta	• Para manter um registro de testes executados e os resultados obtidos. • Para ser suficientemente completas para que você possa mais tarde repetir os mesmos testes e atingir os mesmos resultados. O uso adequado deste relatório irá simplificar os testes de regressão de programas modificados
Geral	• Aumente esta Tabela ou use múltiplas copias quando necessário • Registre cada teste realizado • Seja o mais breve e conciso possível
Cabeçalho	Entre com o que segue • Seu nome, • Data de Hoje, • Nome do Instrutor, • Nome do programa, ou comentário, • O número do programa, e qual linguagem utilizada.
Nome Teste/ Número	Identifique cada execução de Teste para cada programa • Mesmos testes com diferentes datas • Mesmas datas com diferentes testes • Execute novamente o teste com a correção aplicada.
Objetivo do Teste	Brevemente descreva o objetivo do teste
Descrição do Teste	Descreva cada teste com data e procedimentos com detalhes suficientes para permitir que depois serem reprisados.
Condições do Teste	• Liste qualquer configuração especial, registro de Tempo (Cronômetro), correções, e outras condições para o teste. • Quando diferentes testes estão executados com diferentes correções, liste cada um separadamente, por exemplo, Inicial, Correção 1, Correção 2, e assim por diante.
Resultados Esperados	Liste os resultados que o teste deve produzir, se isto executar corretamente.
Resultados Reais	• Liste os resultados que foram realmente alcançados • Quando o mesmo o teste é executado múltiplas vezes durante a correção de vários defeitos, destaque o resultado para cada teste, por exemplo, Resultado Inicial, Resultado A, Resultado B, e assim por diante.

Tabela A-30. Modelo de Estimativa de Tamanho

Estudante _____ Data _____
Instrutor _____ Programa # _____

PROGRAMA BASE LOC				ESTIMADO	REAL
TAMANHO BASE (B) => => => => => => => => =>					
LOC DELETADAS (D) => => => => => => => => =>					
LOC MODIFICADAS (M) => => => => => => => =>					
LOC OBJETO					
ADICÕES BASE	TIPO[1]	MÉTODOS	TAMANHO RELATIVO	LOC	LOC
TOTAL ADICÃO BASE (BA) => => => => =>					
NOVO OBJETO	TIPO*	MÉTODOS	TAMANHO RELATIVO	LOC (Novo	Reutizado*)
	.				
TOTAL NOVOS OBJETOS (NO) => => => => => => => =>					
OBJETOS REUSADOS					
TOTAL REUSADOS (R) => => => => => => => => =>					

		TAMANHO	TEMPO
LOC Tamanho Estimado (E):	E = BA+NO+M		
Parâmetros de Regressão:	β_0 Tamanho e Tempo		
Parâmetros de Regressão:	β_1 Tamanho e Tempo		
Estimativa LOC Novas e Alteradas (N):	N = β_0 + β_1*E		
Estimativa LOC Total:	T = N + B - D - M + R		
Estimativa novas reusados Total (Soma das * LOC):			
Tempo Total Estimado de Desenvolvimento:			
Tempo = β_0 + β_1*E			
Intervalo de Predição:	Intervalo		
Intervalo de Predição Superior:	IPS = N + Range		
Intervalo de Predição Inferior:	IPI = N - Range		
Porcentagem do Intervalo de Predição:			

*L-Lógico, I-E/S, C-Cálculo, T-Texto, D-Dados, S-Instalação

Tabela A-31. Modelo de Estimativa de Tamanho – Instruções

Proposta	Para guiá-lo no processo de estimativa e guardar os dados estimados
Cabeçalho	Entre com o que segue: • Seu Nome • Dia de Hoje • Nome do Instrutor • Número do programa
Nota de Aplicação	Enquanto o Método PROBE é descrito para linguagens Orientadas à Objetos, isto também pode ser usado em outras linguagens com as seguintes mudanças • Se a linguagem utiliza procedimentos, substitua a palavra procedimento para a palavra objeto • Se a linguagem utiliza função, substitua a palavra função para a palavra objeto • Com estas mudanças, use o método PROBE para estimar para sua linguagem, usando um método por procedimento ou função
Programa Base	Se este desenvolvimento é uma modificação ou aprimoramento de um programa existente, faça o seguinte: • Conte e entre com o tamanho da base do programa e entre em B • Entre com o número de linhas de código para serem excluídas em D • Entre com o número de linhas de código para serem modificadas em M
LOC Projetadas – Adições à Base (BA)	Se planejas adicionas Linhas de Código ao programa base, faça o seguinte: • Identifique as funções à serem adicionadas • Estime as linhas de código para cada função adicionada • Se apropriado estime está função como se ela fosse um novo objeto, usando o tipo de categoria que ela se aplica • Entre com o total de adições à Base em BA
LOC Projetadas – Novo Objeto (NO)	• Atribua nome para cada novo objeto planejado • Estime o tipo do objeto (veja no rodapé do modelo) • Estime o número de métodos que o objeto provavelmente conterá • Estime o tamanho relativo do objeto: Muito pequeno(MP), Pequeno(P), Médio(M), Grande(G), Muito Grande(MG) • Obtenha as linhas de código por método para o tipo de objeto e seu relativo tamanho de seu banco de dados • Para determinar o numero de linhas de códigos estimadas para cada tipo de objeto, multiplique as linhas de código do método por o estimado numero de métodos • Destaque cada objeto planejado ou para a biblioteca de reuso com um *. • Entre com o total de linhas de código para novos objetos estimados em NO

	• Entre com o total de linhas de código para novos objetos reutilizáveis (destacados com *) no Resumo do Plano de Projeto
Objetos Reusados	• Entre com o nome de cada objeto reusado sem modificações • Entre com as linhas de código para cada objeto reusado sem modificações abaixo de data • Some as linhas de código e entre no TOTAL REUSADAS (R) Se quais querem objetos reusados estão a ser modificados ou aprimorados, incluía as linhas de código do objeto ao longo dos números de tamanho base do programa com quais queres exclusões e modificações. Inclua as adições nas adições da base
Cálculos	• Some as linhas de código para adições a base (BA) e novos objetos (NO) • Entre com o total mais M em (E) Estimar as novas e alteradas linhas de código (N) • Calcule os parâmetros de regressão dos dados históricos usando os procedimentos descritos na subseção 4.3.2. Usando os parâmetros de regressão e o Tamanho estimado (E) para o novo programa calcule (N) Estimativa de LOC Total (T) • T = N + B – D – M + R Estimativa do total de Novas Reusadas • A soma de Novas Reusadas * itens A Faixa de Predição • Calcule a faixa usando os procedimentos descritos na subseção 4.3.2. O Superior (IPS) e inferior (IPI), intervalos de predição. • IPS = N + Faixa • IPI = N – Faixa Se IPI for negativo use 0 • Porcentagem do Intervalo de Predição: • Liste a percentagem de probabilidade utilizada para calcular o intervalo de predição (70 % ou 90 %)

Tabela A-32. PSP1.1 Script do Processo

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP1.1 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Histórico de dados estimados de tamanho reais. • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Use o método PROBE para estimar o total de linhas de código novas e alteradas necessários. • Completar o Modelo de Estimativa de Tamanho • Use o método PROBE para estimar o tempo requerido para o desenvolvimento • Complete um Modelo de Planejamento de Tarefas • Complete um Modelo de Planejamento de Horário • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Implementação do design • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Modelo de estimativa de tamanho completado • Modelo de Relatório de Teste completado • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-33. PSP1.1 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP1 Formulário do Plano de Projeto resumido • Modelos de Planejamento de Tarefas, Planejamento de Horário, e Estimativa de Tamanho. • Histórico de dados estimados de tamanho reais e dados de tempo • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Produza um projeto de um programa conceitual • Use o método PROBE para fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa • Estime linhas de código base, adicionadas, removidas e reusadas • Complete o Modelo de Estimativa de Tamanho e o Resumo do Plano de Projeto
3	Recursos estimados	• Use o método PROBE para estimar o tempo requerido para o desenvolvimento deste programa • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas.
4	Planejamento de Horários e Tarefas	Para projetos que requerem vários dias ou mais de trabalho, complete os Modelos de Planejamento de Horário e Tarefas.
	Critérios de saída	• Uma documentada declaração de requisitos • O projeto de um programa conceitual • Modelo de Estimativa de Tamanho completo. • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-34. PSP1.1 Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa estimado e tempo de desenvolvimento. • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos e produzir um projeto para alcançá-los • Registrar tempo no registro de gravação de tempo
2	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
3	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
4	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo • Complete o Modelo de Relatório de Teste com testes conduzidos e os resultados obtidos
	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Modelo de Relatório de Teste completo • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-35. PSP1.1 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa e o tempo de desenvolvimento. • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados. • Modelo de Relatório de Teste completo • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 1.1 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 1.1 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 1.1 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Um Modelo de Relatório de Teste completo. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos

Tabela A-36 – PSP1.1 – Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Resumo	**Plano**	**Real**	**Datado**
Tamanho/Hora			
Tempo Planejado			
Tempo Real			
CPI (Índice de Custo-Desempenho)			(Planejado/Atual)
% Reusado			
% Novo Reutilizável			

Tamanho do Programa (LOC)	***Plano***	***Real***	***Datado***
Base(B)	***(Mensuradas)***	***(Mensuradas)***	
Apagadas (AP)	***(Estimadas)***	***(Contadas)***	
Modificadas (M)	***(Estimadas)***	***(Contadas)***	
Adicionadas (A)	***(N – M)***	***(T-B+AP-R)***	
Reutilizadas (R)	***(Estimadas)***	***(Contadas)***	
Total Novas & modificadas (N)	***(Estimadas)***	***(A+M)***	
Total Linhas de Código (T)	***(N + B – M – D + R)***	***(Mensuradas)***	
Total Nova(s) Reutilizada(s)			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Código				
Compilação				
Teste				
Postmortem				
Total				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			

Total Desenvolvimento			
Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Tabela A-37. PSP1.1 Resumo do Plano de Projeto – Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Resumo	• Entre com o novo e alterado valor de LOC por hora planejado e real, e para todos os programa desenvolvidos insira no campo Datado • Entre com os tempos reais e planejados para este programa e a soma de todos os tempos planejados e reais de para todos os exercícios datados • CPI = (Tempo planejado datado) / (Tempo Real datado) • Entre com os dados de reusados para planejados, reais e datados.
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Plano. • Entre com o valor de novas e modificadas LOC (N) do Modelo de Estimava de Tamanho. • Estimar o numero de LOC adicionadas (A) e modificadas(M), então faça N = A + M. • Estimar o numero de LOC deletadas (D) e reusadas (R) e combinar com o número de LOC da base (B), T = N + B –M –D + R. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R

	• Calcule o total de novas e reutilizadas linhas de código N = A+M
Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.
Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação

Tabela A-38. Modelo de Planejamento de Tarefas

Estudante _____ Data _____
Projeto _____ Instrutor _____

Tarefa		Plano					Real		
#	Nome	Horas	Valor Planejado	Horas Cumulativas	Valor Planejado Cumulativo	Data Segunda	Data	Valor Ganho	Valor agregado cumulativo
Totais									

Tabela A-39. Modelo de Planejamento de Tarefas

Proposta	• Para estimar o tempo de desenvolvimento para cada tarefa do projeto • Para computar os valores planejados para cada tarefa do projeto • Para estimar a data de conclusão planejada para cada tarefa • Para fornecer uma base para rastrear o progresso programação mesmo quando as tarefas não são concluídas na ordem planejada.
Geral	• Expanda este modelo em múltiplas paginas com a necessidade • Inclua cada tarefa significante • Use nomes e números nas tarefas que auxiliam a atividade e sejam consistentes com a estrutura de divisão de trabalho do projeto
Cabeçalho	Entre com o que segue: • Seu nome • Data de hoje • Nome do instrutor • Nome do projeto
Tarefa	• Entre com o número e nome da tarefa. Liste as tarefas em ordem na qual esperar finaliza-las • Escolha tarefas que tenham critérios explícitos de realização, por exemplo, planejamento completado, programa compilado, todos os defeitos corrigidos, e assim por diante.
Plano – Horas	Entre com as horas planejadas em cada tarefa.
Plano – Valor planejado	• Total de horas planejadas em todas as tarefas • Para cada tarefa, calcular a porcentagem de horas planejados do total de horas • Entre com a porcentagem como valor planejado para esta tarefa • O valor total planejado deve ser igual a 100
Plano – Horas Cumulativas	Entre com a soma cumulativa de horas planejadas abaixo de cada tarefa
Plano – Valor Cumulativo	• Soma de valores planejados através de cada tarefa • Antes de proceder, complete o Resumo do Plano de Projeto, através do Plano – Horas Cumulativas • Então complete o Resumo do Plano de Projeto e o Modelo de Planejamento de tarefas juntos.
Plano Data – Segunda	• Para cada horas cumulativas insertas, encontre as horas cumulativas planejadas no Modelo de Planejamento de Horário que seja igual ou exceda à ele • Digite a data a partir dessa linha (do Modelo de Planejamento de Programação) como a data de plano no modelo de planejamento de tarefas. • Se varias semanas do Modelo de Planejamento de Horário tem o mesmo valor cumulativo, entre com a data mais próxima • A menos que você fez planos diários, escolher a data do plano como a

	segunda da semana durante a qual a conclusão de que a tarefa é planejada.
Date Real	Como cada tarefa é completada, entre com a data de conclusão.
Valor ganho	Para cada tarefa finalizada, entre com o valor planejado.
Valor agregado cumulativo	Como toda tarefa é completada, pegue o valor total de todos os valores ganhos e entre ao lado da ultima tarefa que foi completada.

Tabela A-40. Modelo de Planejamento de Horário

Estudante ________________________ Data __________

Projeto ________________________ Instrutor __________

		Plano			Real			
Número Semana	**Data Segunda**	**Horas Diretas**	**Horas Cumulativas**	**Valor Planejado Cumulativo**	**Horas Diretas**	**Horas Cumulativas**	**Valor Agregado Cumulativo**	**Valor agregado ajustado**

Tabela A-41. Modelo de Planejamento de Horário – Instruções

Proposta	• Para gravar as horas estimadas e reais gastas por período de calendário. • Para relacionar o valor da tarefa planejada com o cronograma do calendário. • Para calcular os valores planejados ajustados e ganhos quando as tarefas mudam
Geral	• Expanda este modelo em múltiplas paginas com a necessidade • Complete este modelo em conjunto com o Modelo de Planejamento de Tarefas
Cabeçalho	Entre com o que segue: • Seu nome • Data de hoje • Nome do instrutor • Nome do projeto
Número da Semana	• Desde o Início do Projeto entre com o número da semana, formalmente iniciando com 1 • Para projetos muito, é mais conveniente usar dias ao invés de semanas
Data(Segunda)	• Entre com a data do calendário para cada semana • Pegue um dia padrão da semana, por exemplo, Segunda
Plano – Horas Diretas	• Entre com o número planejado de horas diretas do projeto que espera gastar em cada semana • Considere tempo não trabalhado como férias, feriados, e assim por diante. • Considere atividades cometidas como aulas, encontros e outros projetos.
Plano – Horas Cumulativas	Entre com as horas planejadas cumulativas através de cada semana
Plano – Valor Planejado Cumulativo	Para cada semana, faça o que segue: • Pegue as horas cumulativas do Plano do Modelo de Planejamento de Horário • No Modelo de Planejamento de Tarefas, encontrar a tarefa do Plano mais próxima igual ou inferior de horas acumuladas e anote o seu valor cumulativo no Plano. • Entre com este valor cumulativo no Modelo de Planejamento de Horário • Se o valor acumulado para a semana anterior ainda se aplica, insira-o novamente.
Atual	• Durante o desenvolvimento, entre com as horas diretas reais, horas cumulativas e valor cumulativo agregado para cada semana. • Determinar o estado diante ao plano, comparando o valor acumulado previsto e o valor acumulado real agregado.
Valor agregado ajustado	Proporcionalmente ajuste o valor agregado para mais e menos em ralação as tarefas adicionadas ou excluídas.

Tabela A-42. PSP2 Script do Processo

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP2 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Histórico de dados estimados de tamanho reais. • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Use o método PROBE para estimar o total de linhas de código novas e alteradas necessários e o intervalo de predição • Completar o Modelo de Estimativa de Tamanho • Use o método PROBE para estimar o tempo requerido para o desenvolvimento e o intervalo de predição • Complete um Modelo de Planejamento de Tarefas • Complete um Modelo de Planejamento de Horário • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Revise o design(projeto), conserte e registre os defeitos encontrados • Implementação do design • Revise o código, conserte e registre todos os defeitos encontrados. • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Modelos de estimativa e planejamento completados • Lista de revisão de Design e Código completados • Modelo de Relatório de Teste completado • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-43. PSP2 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP2 Formulário do Plano de Projeto resumida • Modelos de Planejamento de Tarefas, Planejamento de Horário, e Estimativa de Tamanho. • Histórico de dados estimados de tamanho reais e dados de tempo • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Produza um projeto de um programa conceitual • Use o método PROBE para fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa • Estime linhas de código base, adicionadas, removidas e reusadas • Complete o Modelo de Estimativa de Tamanho e o Resumo do Plano de Projeto • Calcule a 70 por cento do intervalo de predição de tamanho
3	Recursos estimados	• Use o método PROBE para estimar o tempo requerido para o desenvolvimento deste programa • Calcule a 70 por cento do intervalo de predição de tempo • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas.
4	Planejamento de Horários e Tarefas	Para projetos que requerem vários dias ou mais de trabalho, complete os Modelos de Planejamento de Horário e Tarefas.
5	Defeitos Estimados	• Baseado nos dados Datados por defeitos por novas e modificadas linhas de código, estime o total de defeitos a serem encontrados no programa • Baseado nos dados de porcentagem de Datados, estime o numero de defeitos a serem injetados e removidos por fase
	Critérios de saída	• Uma documentada declaração de requisitos • O projeto de um programa conceitual • Modelo de Estimativa de Tamanho completo. • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-44. PSP2 Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa estimado e tempo de desenvolvimento. • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos e produzir um projeto para alcançá-los • Registrar tempo no registro de gravação de tempo
2	Revisão de design	• Siga a Lista de Revisão de Design e revise o design • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
3	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
4	Revisão de Código	• Siga a Lista de Revisão de Código e revise o código • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
5	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo
6	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo • Complete o Modelo de Relatório de Teste com testes conduzidos e os resultados obtidos

	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Lista de Revisão de Design e Código completadas. • Modelo de Relatório de Teste completo • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-45. PSP2 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa, tempo de desenvolvimento e dados de defeitos • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados. • Modelo de Relatório de Teste completo • Lista de Revisão de Design e Código completas. • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 2 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 2 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais • Calcular o rendimento de todo o processo atual e inseri-lo no Resumo do Plano do Projeto.
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 2 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Lista de Revisão de Design e Código completadas • Um Modelo de Relatório de Teste completo. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos

Tabela A-46. PSP2 – Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Resumo	**Plano**	**Real**	**Datado**
Tamanho/Hora			
Tempo Planejado			
Tempo Real			
CPI (Índice de Custo-Desempenho)			(Planejado/Atual)
% Reusado			
% Novo Reutilizável			
Defeitos Teste/KLOC			
Total Defeitos/KLOC			
Rendimento %			

Tamanho do Programa (LOC)	Plano	Real	Datado
Base(B)	(Mensuradas)	(Mensuradas)	
Apagadas (AP)	(Estimadas)	(Contadas)	
Modificadas (M)	(Estimadas)	(Contadas)	
Adicionadas (A)	(N – M)	(T-B+AP-R)	
Reutilizadas (R)	(Estimadas)	(Contadas)	
Total Novas & modificadas (N)	(Estimadas)	(A+M)	
Total Linhas de Código (T)	(N + B – M – D + R)	(Mensuradas)	
Total Nova(s) Reutilizada(s)			
Intervalo de Predição Superior ***(70%)***			
Intervalo de Predição Inferior ***(70%)***			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Revisão Design				
Código				
Revisão Código				
Compilação				
Teste				
Postmortem				

Total				
Tempo Total IPS (70%)				
Tempo Total IPI (70%)				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Eficiência de Remoção de defeito	***Plano***	***Real***	***Datado***
Defeitos/hora – Revisão Design			
Defeitos/hora – Revisão Código			
Defeitos/hora – Compilação			
Defeitos/hora – Testes			
DRL(DLDR/UT)			
DRL(Revisão Código/UT)			
DRL(Compilação/UT)			

Tabela A-47. PSP2 – Resumo do Plano de Projeto – Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Resumo	• Entre com o novo e alterado valor de LOC por hora planejado e real, e para todos os programa desenvolvidos insira no campo Datado • Entre com os tempos reais e planejados para este programa e a soma de todos os tempos planejados e reais de para todos os exercícios datados • CPI = (Tempo planejado datado) / (Tempo Real datado) • Entre com os dados de reusados para planejados, reais e datados. • Entre com os dados de defeitos planejados, reais, e datados • Entre com o rendimento planejado e real
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Plano. • Entre com o valor de novas e modificadas LOC (N) do Modelo de Estimava de Tamanho. • Estimar o numero de LOC adicionadas (A) e modificadas(M), então faça N = A + M. • Estimar o numero de LOC deletadas (D) e reusadas (R) e combinar com o número de LOC da base (B), T = N + B –M –D + R. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R • Calcule o total de novas e reutilizadas linhas de código N = A+M
Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.

Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação
Eficiência de Remoção de defeito	• Abaixo de Plano, entre com as eficiências planejadas para este projeto • Abaixo de Real, entre com as eficiências reais atingidas • Abaixo de Datado, entre com as eficiências reais atingidas para todos os projetos até à data.

Tabela A-48. PSP2 - Lista de Revisão de Projeto

NOME DO PROGRAMA E NÚMERO

Proposta	Para guiá-lo em uma eficaz realização de Revisão de Projeto				
Geral	• Quando completar cada passo da revisão, marque este item na caixa à direita • Complete a lista para uma unidade do programa antes de iniciar a revisão da próxima				
Completar	Garanta que os requerimentos, especificações, e o design de alto nível são completamente cobertos pelo projeto • Todos os resultados especificados são produzidos. • Todos os insumos necessários são fornecidos • Todos as inclusões requeridas são iniciadas				
Lógica	• Verifique que a seqüência do programa é adequada: o Pilhas, listas, e assim por diante estão na ordem adequada o Recursões se comportam corretamente • Verifique que todos os loops são adequadamente iniciados, incrementados e finalizados.				
Casos Especiais	Cheque todos os casos especiais: • Garantir o funcionamento adequado com vazio, cheio, mínimo, máximo, negativos, valores zero para todas as variáveis. • Proteger contra saídas de limite, condições de overflow e underflow • Garantir que condições "impossíveis" são absolutamente impossíveis • Manipular todas as condições de entrada incorretas				
Uso Funções	• Verifique que todas as funções, procedimentos, ou objetos são totalmente compreendidos e propriamente usados. • Verifique que todas as Abstrações referenciadas externamente são justamente definidas.				
Nomes	Verifique o seguinte: • Todos os nome especiais e tipos são claros ou especialmente definidos • Os escopos de todas as variáveis são auto-evidentes ou definidos • Todos os objetos nomeados são usados dentro de seus escopos declarados				
Padrões	Revisar o Design para a conformidade com todos os padrões de design aplicáveis.				

Tabela A-49. Lista de Revisão de Código

NOME DO PROGRAMA E NÚMERO

Proposta	Para guia-lo numa revisão de Código efetiva				
Geral	• Quando completar cada passo da revisão, marque este item na caixa à direita • Complete a lista para uma unidade do programa antes de iniciar a revisão da próxima				
Completar	Verifique que o Código cobre todo o design				
Bibliotecas	Verifique que todas as bibliotecas estão de acordo				
Inicialização	Cheque variáveis e parâmetros de inicialização: • Na inicialização do programa • No Inicio de cada loop • Na entrada de funções e procedimentos				
Chamadas	Cheque o formato de chamado das funções • Ponteiros • Parâmetros • Uso do '&'				
Nomes	Cheque a pronuncia dos nomes e use: • Isto é Consistente? • Isto esta dentro do escopo declarado? • Todas as estruturas e classe usam a referencia '.'				
Strings	Cheque que as string estão: • Identificadas por ponteiros • Terminados com NULL				
Ponteiros	Cheque que: • Ponteiros são inicializados com NULL • Ponteiros são excluídos apenas depois de novos, e • Novos ponteiros sempre são excluídos depois de seu uso				
Formatos de Saída	Cheque o formato de saída: • Linhas de passo são apropriadas • Espaçamento é apropriado				
{} Parênteses	Garanta que os {} são apropriados e combinados				
Operadores Lógicos	Verifique o uso apropriado de ==, =, //, e assim por diante. Cheque que cada função lógica é apresentada entre ( )				
Cheque linha por linha	Cheque cada Linha de Código por • Sintaxe da instrução e • Pontuação apropriada				
Padrões	Garanta que o Código esta conforme com Padrões de Codificação				
Abertura e fechamento de Arquivo	• Verifique que todos os arquivos são: • Adequadamente declarados, • Abertos e fechados.				

Tabela A-50. PSP2.1 Script do Processo

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de modulo.
	Materiais Necessários	• Descrição do problema • PSP2.1 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Histórico de dados estimados de tamanho reais. • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões
1	Planejamento	• Produzir ou obter uma declaração de requisitos • Use o método PROBE para estimar o total de linhas de código novas e alteradas necessários e o intervalo de predição • Completar o Modelo de Estimativa de Tamanho • Use o método PROBE para estimar o tempo requerido para o desenvolvimento e o intervalo de predição • Complete um Modelo de Planejamento de Tarefas • Complete um Modelo de Planejamento de Horário • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
2	Desenvolvimento	• Designer do Programa • Revise o design(projeto), conserte e registre os defeitos encontrados • Implementação do design, usando modelos de design onde for apropriado • Revise o código, conserte e registre todos os defeitos encontrados. • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo
3	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais.
	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Modelos de estimativa e planejamento completados • Modelos de Design completados • Lista de revisão de Design e Código completados • Modelo de Relatório de Teste completado • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-51. PSP2.1 Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP2.1 Formulário do Plano de Projeto resumido • Modelos de Planejamento de Tarefas, Planejamento de Horário, e Estimativa de Tamanho. • Histórico de dados estimados de tamanho reais e dados de tempo • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Produza um projeto de um programa conceitual • Use o método PROBE para fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa • Estime linhas de código base, adicionadas, removidas e reusadas • Complete o Modelo de Estimativa de Tamanho e o Resumo do Plano de Projeto • Calcule a 70 por cento do intervalo de predição de tamanho
3	Recursos estimados	• Use o método PROBE para estimar o tempo requerido para o desenvolvimento deste programa • Calcule a 70 por cento do intervalo de predição de tempo • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas.
4	Planejamento de Horários e Tarefas	Para projetos que requerem vários dias ou mais de trabalho, complete os Modelos de Planejamento de Horário e Tarefas.
5	Defeitos Estimados	• Baseado nos dados Datados por defeitos por novas e modificadas linhas de código, estime o total de defeitos a serem encontrados no programa • Baseado nos dados de porcentagem de Datados, estime o numero de defeitos a serem injetados e removidos por fase
	Critérios de saída	• Uma documentada declaração de requisitos • O projeto de um programa conceitual • Modelo de Estimativa de Tamanho completo. • Plano de Projeto resumido concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-52. PSP2.1 Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de pequenos programas
	Critérios de Entrada	• Requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa estimado e tempo de desenvolvimento. • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos e produzir uma especificação externa para atingi-los. • Complete os Modelos de Especificação Funcional e Cenário Operacional para registrar esta especificação. • Produzir um projeto para atender a essa especificação. • Registre o Projeto nos Modelos de Especificação Funcional, de Estado, Lógico e Cenário Operacional como requerido. • Registre no Log de Registro de Defeitos qualquer defeito encontrado no requerimento • Registrar tempo no registro de gravação de tempo
2	Revisão de design	• Siga a Lista de Revisão de Design e revise o design • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
3	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
4	Revisão de Código	• Siga a Lista de Revisão de Código e revise o código • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
5	Compilação	• Compilar o programa até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo

6	Teste	• Testar até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo • Complete o Modelo de Relatório de Teste com testes conduzidos e os resultados obtidos
	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Modelos de Design completos. • Lista de Revisão de Design e Código completas. • Modelo de Relatório de Teste completo • Log de gravação de defeitos concluído • Log de gravação de tempo concluído

Tabela A-53. PSP2.1 Script de Postemortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa, tempo de desenvolvimento e dados de defeitos e intervalo de predição. • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados. • Modelo de Relatório de Teste completo • Modelos de Design completos • Lista de Revisão de Design e Código completas • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 2.1 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais
2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 2.1 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais • Calcular o rendimento de todo o processo atual e inseri-lo no Resumo do Plano do Projeto.
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 2.1 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Modelos de Design Completos. • Lista de Revisão de Design e Código completados • Um Modelo de Relatório de Teste completo. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos

Tabela A-54. PSP2.1 – Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Resumo	**Plano**	**Real**	**Datado**
Tamanho/Hora			
Tempo Planejado			
Tempo Real			
CPI (Índice de Custo-Desempenho)			(Planejado/Atual)
% Reusado			
% Novo Reutilizável			
Defeitos Teste/KLOC			
Total Defeitos/KLOC			
Rendimento %			
% Avaliação CDQ			
% Falha CDQ			
COQ A/F Taxa			

Tamanho do Programa (LOC)	**Plano**	**Real**	**Datado**
Base(B)	(Mensuradas)	(Mensuradas)	
Apagadas (AP)	(Estimadas)	(Contadas)	
Modificadas (M)	(Estimadas)	(Contadas)	
Adicionadas (A)	(N – M)	(T-B+AP-R)	
Reutilizadas (R)	(Estimadas)	(Contadas)	
Total Novas & modificadas (N)	(Estimadas)	(A+M)	
Total Linhas de Código (T)	(N + B – M – D + R)	(Mensuradas)	
Total Nova(s) Reutilizada(s)			
Intervalo de Predição Superior **(70%)**			
Intervalo de Predição Inferior **(70%)**			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design				
Revisão Design				
Código				
Revisão Código				

Compilação				
Teste				
Postmortem				
Total				
Tempo Total IPS (70%)				
Tempo Total IPI (70%)				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Eficiência de Remoção de defeito	**Plano**	**Real**	**Datado**
Defeitos/hora – Revisão Design			
Defeitos/hora – Revisão Código			
Defeitos/hora – Compilação			
Defeitos/hora – Testes			
DRL(DLDR/UT)			
DRL(Revisão Código/UT)			
DRL(Compilação/UT)			

Tabela A-55. PSP2.1 – Resumo do Plano de Projeto – Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Resumo	• Entre com o novo e alterado valor de LOC por hora planejado e real, e para todos os programa desenvolvidos insira no campo Datado • Entre com os tempos reais e planejados para este programa e a soma de todos os tempos planejados e reais de para todos os exercícios datados • CPI = (Tempo planejado datado) / (Tempo Real datado) • Entre com os dados de reusados para planejados, reais e datados. • Entre com os dados de defeitos planejados, reais, e datados • Entre com o rendimento planejado e real
Custo da Qualidade	• Entre com a % de Avaliação CDQ: a porcentagem do tempo de desenvolvimento gasto na Revisão de Design e Código • Entre com a % de Falha CDQ: a porcentagem do tempo de desenvolvimento gasto na Compilação e Testes • Entre com a taxa A/F: o índice de Avaliação CDQ dividido por Falha CDQ
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Plano. • Entre com o valor de novas e modificadas LOC (N) do Modelo de Estimava de Tamanho. • Estimar o numero de LOC adicionadas (A) e modificadas(M), então faça N = A + M. • Estimar o numero de LOC deletadas (D) e reusadas (R) e combinar com o número de LOC da base (B), T = N + B –M –D + R. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R • Calcule o total de novas e reutilizadas linhas de código N = A+M

Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.
Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação
Eficiência de Remoção de defeito	• Abaixo de Plano, entre com as eficiências planejadas para este projeto • Abaixo de Real, entre com as eficiências reais atingidas • Abaixo de Datado, entre com as eficiências reais atingidas para todos os projetos até à data.

Tabela A-56. Modelo de Cenário Operacional

Estudante ______________________________ Data ____________
Programa ______________________________ Programa # ____________
Instrutor ______________________________ Linguagem ____________

Construção de cenários operacionais para cobrir os usos do programa normais e anormais, incluindo erros de usuário:

Cenário #		Objetivo Usuário:	
Objetivo Cenário:			
Fonte:	Passo	Ação:	Comentários

Tabela A-57. Modelo de Cenário Operacional – Instruções

Propósito	• Para manter as descrições dos cenários operacionais prováveis seguidos durante o uso do programa • Para garantir que todos os problemas de uso significativos são considerados durante a concepção do programa • Para especificar cenários de teste
General	• Utilize este modelo para programas completos, subsistemas ou sistemas. • Agrupar vários cenários pequenos em um único modelo, desde que eles são claramente distintas e têm objetivos relacionados. • Liste os principais cenários referencie outras exceções, erros, ou casos especiais sob comentários. • Utilize este modelo para documentar as especificações operacionais durante o planejamento, projeto, desenvolvimento, implementação e testes • Após a implementação e testes, atualizar o modelo para refletir que produto foi realmente executado.
Cabeçalho	- Entre com o nome e a data de hoje - Entre com o nome do programa e seu numero. - Entre com o nome do Instrutor e a linguagem de programação que está usando
Número Cenário	Quando vários Cenários estão envolvidos, os números de referencia são necessários
Objetivo Usuário	Liste as prováveis propostas do Usuário para o Cenário, por exemplo, Para iniciar o sistema e selecionar o modo operacional
Objetivo Cenário	Lista objetivo do designer para o cenário, por exemplo, para definir erros comuns que o usuário pode fazer quando seleciona o modo operacional
Fonte	A fonte para ação do Cenário: por exemplo, três fontes poderiam ser usuário, programa e sistema
Passo	Fornecer números de seqüência para as etapas de cenário. Estes facilitar revisões e inspeções.
Ação	Descreva as medidas tomadas, como • Modo de seleção acessado incorretamente • Fornecer mensagem de erro.
Comentários	Liste informações significantes relatando a ação, como segue: • Usuário entra com um valor incorreto • Mensagem do Sistema: “Valor Incorreto, tente novamente”

Tabela A-58. Modelo de Especificação Funcional

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Objeto / Nome Classe	Classes Pais	Atributos
Declaração Método	Especificação método externo	

Tabela A-59. Modelo de Especificação Funcional

Proposta	Para guardar uma especificação funcional de um objeto. Podes usar este modelo para descrever funções e procedimentos, quando não usa linguagens orientadas à objeto. Se sim, troque a palavra objeto pelas palavras funções ou procedimento apropriadamente
Geral	Se vários objetos pertences à mesma classe, agrupar estes múltiplos, relacionados com modelos de especificação juntos.
Cabeçalho	• Entre com o nome do programa, instrutor e Estudante. • Data, Número do programa e Linguagem.
Nome da Classe/Objeto	• Entre com o nome do objeto e classe na qual eles herdam • Liste os nomes das classes começando com a mais imediata • Sempre que possível, liste a Hierarquia de herança completa.
Atributos	Entre com qualquer parâmetro de qual os valores são extremamente visíveis e são de impacto ao comportamento do objeto
Declaração Método	• Liste a declaração para cada método do objeto • Inclua todas as variáveis requeridas e parâmetros • Inclua a declaração do tipo requerida • Use o formato usado para declarar o método no programa
Especificação método externo	• Descreva a operação realizada por cada método • Quando possível, use uma especificação formal ou matemática • Inclua todos os retornos e condições de exceção

Tabela A-60. Modelo de Especificação de Estado

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	
Objeto		Rotina	

Estado #1	Descrição	Atributos
Próximo estado #1	Condições de Transição	
Próximo estado #2		
. . .		
. . .		
Próximo estado # n		

Estado #2	Descrição	Atributos
Próximo estado #1	Condições de Transição	
Próximo estado #2		
. . .		
. . .		
Próximo estado # n		

...

Estado #n	Descrição	Atributos
Próximo estado 1	Condições de Transição	
Próximo estado #2		
. . .		
. . .		
Próximo estado # n		

Tabela A-61. Modelo de Especificação de Estado – Instruções

Proposta	Para manter as especificações do estado e do estado de transição para um sistema, programa de classe, ou Apoiar o estado da máquina de análise durante as inspeções de projeto, revisões de projeto e design
Geral	Este formulário mostra cada sistema, programa, ou estado de rotina, os atributos de que o estado e as condições de transição entre os estados. Completar um segmento deste modelo para cada estado.
Cabeçalho	• Digite seu nome e a data. • Digite o nome do programa e número. • Digite o nome do instrutor e da linguagem de programação que você está usando. • Objeto/Rotina: Liste o nome do objeto e a rotina cujo o comportamento do estado está sendo descrito
Estado #1	Enquanto os Estados poderiam simplesmente ser numerados, é útil para dar a cada estado um nome descritivo. Por exemplo, um estado de máquina com três estados poderia usar nomes tais como, Vazio, Parcial, Cheio.
Descrição	Esta seção guarda um texto descritivo do estado
Atributos	Liste o valor das variáveis que caracterize o estado. Por exemplo, se o estado cheio é caracterizado por k == 10 e n == 3, então o atributo de entrada deveria ser k==10 e n== 3 • Seja tão preciso possível
Próximo Estado #1	Entre aqui com o nome do estado 1 # Abaixo de cada estado, liste os nomes de todos os outros estados. Por exemplo, abaixo de Estado Vazio, a linha deveria ser vazia, parcial, cheia.
Condições de Transição	• Para cada próximo estado, liste as condições em que uma transição é feita a partir do estado atual para este estado. • Seja tão preciso possível • Se a transição é impossível, insira impossível
Condições de Transição, Exemplos	Para o estado vazio, as condições de transição devem ser como segue: • Vazio: nenhuma entrada • Parcial: qualquer entrada • Cheio: impossível

Tabela A-62. Modelo de Especificação de Lógica

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	
Objeto		Função	

BIBLIOTECAS:

DEFINIÇÕES DE TIPO:

Declaração:
Referência:

Números de referência lógica	Lógica do Programa, no pseudocódigo.

Tabela A-63. Modelo de Especificação de Lógica – Instruções

Proposta	Para manter a lógica do pseudocódigo para cada objeto ou elemento modular de programa. Uma copia separada deste modelo deve ser usada para cada programa • Se o design do programa não se encaixa em uma simples pagina do modelo, ou ele é tão grande que deve ser particionado em vários pequenos programas ou o Pseudocódigo está em um nível muito detalhado. • Se os métodos do design orientado à objeto não são usados, este modelo pode ainda ser usado para funções especificas, procedimentos, e uma rotina lógica principal
Cabeçalho	• Digite seu nome e a data. • Digite o nome do programa e número. • Digite o nome do instrutor e da linguagem de programação que você está usando. • Objeto/Função: Liste o nome da função sendo especificada junto com o objeto no qual ela pertence
Bibliotecas	• Liste todas as novas ou incomuns bibliotecas usadas no programa • Bibliotecas padrões de projeto não devem ser listadas separadamente • Onde bibliotecas padrões são usadas, a afirmação "BIBLIOTECAS PADRÕES DE PROJETO" deve ser introduzido • Bibliotecas para abstrações devem ser incluídas com suas descrições de design e implementações
Definições de Tipo	• Quando incomuns ou tipos especiais são usados, eles devem ser definidos ou dada uma referência para a definição • Para salvar o tempo de implementação, definir todos ou a maioria dos tipos especiais no modelo.
Declaração	• Dê uma declaração exata da função uma vez que está a ser escrito no programa. • Exceto quando mencionado, todas as declarações, inicializações, e terminações são definidos durante a implementação.
Referência	Quando uma clara compreensão desta função requer informações não no projeto, identificar onde essa informação pode ser encontrada.
Números de referência lógica	• Dê um número de referência padrão para cada significante estado lógico. • Numere cada chamada de procedimento, loop, e estados condicionais. • Estes números de referência serão úteis durante a revisão de design e inspeções
Lógica do Programa	• Liste o pseudocódigo para o programa • Use uma linha de separação para cada função significante • Use afirmações matemáticas ou linguagens comuns quando necessário para clareza • Inclua comentários onde necessário para explicar a lógica

Tabela A-64. PSP2.1 – Lista de Revisão de Projeto

NOME DO PROGRAMA E NÚMERO

Proposta	Para guiá-lo em uma eficaz realização de Revisão de Projeto				
Geral	• Quando completar cada passo da revisão, marque este item na caixa à direita • Complete a lista para uma unidade do programa antes de iniciar a revisão da próxima				
Completar	Garanta que os requerimentos, especificações, e o design de alto nível são completamente cobertos pelo projeto • Todos os resultados especificados são produzidos. • Todos os insumos necessários são fornecidos • Todos as inclusões requeridas são iniciadas				
Estado de Maquina	Verifique o design do Estado de Maquina • A estrutura não tem armadilhas escondidas ou loops • Isto está completo, isto é, todos os possíveis estados foram identificados • É ortogonal, isto é, para cada conjunto de condições há um e somente um estado mais próximo possível. • A transição de cada estado são completas e ortogonais. Isto é, de cada estado um estado único próximo é definido para cada combinação possível de valores de estado de entrada da máquina.				
Lógica	• Verifique que a seqüência do programa é adequada: o Pilhas, listas, e assim por diante estão na ordem adequada o Recursões se comportam corretamente • Verifique que todos os loops são adequadamente iniciados, incrementados e finalizados.				
Casos Especiais	Cheque todos os casos especiais: • Garantir o funcionamento adequado com vazio, cheio, mínimo, máximo, negativos, valores zero para todas as variáveis. • Proteger contra saídas de limite, condições de overflow e underflow • Garantir que condições “impossíveis” são absolutamente impossíveis • Manipular todas as condições de entrada incorretas				
Uso Funções	• Verifique que todas as funções, procedimentos, ou objetos são totalmente compreendidos e propriamente usados. • Verifique que todas as Abstrações referenciadas externamente são justamente definidas.				

Nomes	Verifique o seguinte: • Todos os nome especiais e tipos são claros ou especialmente definidos • Os escopos de todas as variáveis são auto-evidentes ou definidos • Todos os objetos nomeados são usados dentro de seus escopos declarados				
Padrões	Revisar o Design para a conformidade com todos os padrões de design aplicáveis.				

Tabela A-65. PSP3 - Script do Processo

Fase Numero	Proposta	Para guiá-lo no desenvolvimento de programas de nível de componentes.
	Materiais Necessários	• Descrição do problema ou especificação de componentes • PSP3 Formulário do Plano de Projeto resumido • Modelo de Estimativa de Tamanho • Histórico de dados estimados de tamanho reais. • Log de gravação de tempo e registro de defeitos • Tipos de Defeitos Padrões
1	Requerimentos e planejamento	• Produza o plano de desenvolvimento e requerimento ♦ Documento de requerimentos ♦ Conceito de Design ♦ Tamanho, qualidade, recursos, e planos de cronograma • Produza um Log Máster de Controle de Emissão
2	Design de Alto Nível (HLD)	Produza uma design e uma estratégia de implementação • Especificações funcionais • Especificações de Estado • Cenários Operacionais • Especificações de Reuso • Estratégia de desenvolvimento • Estratégia de Teste
3	Revisão de Design de Alto nível (HLDR)	• Revise o Design de Alto nível • Revise a estratégia de desenvolvimento e teste • Conserte e registre todos os defeitos encontrados • Observe as questões pendentes no Log de Controle de Emissão. • Registre no Log de Registro de Defeitos todos os defeitos encontrados

4	Planejamento	• Produzir ou obter uma declaração de requisitos • Use o método PROBE para estimar o total de linhas de código novas e alteradas necessários e o intervalo de predição • Completar o Modelo de Estimativa de Tamanho • Use o método PROBE para estimar o tempo requerido para o desenvolvimento e o intervalo de predição • Complete um Modelo de Planejamento de Tarefas • Complete um Modelo de Planejamento de Horário • Entrar com os dados do plano na forma do Plano de Projeto resumida • Completar o Log de gravação de tempo
5	Desenvolvimento	• Designer do Programa • Revise o design(projeto), conserte e registre os defeitos encontrados • Implementação do design, usando modelos de design onde for apropriado • Revise o código, conserte e registre todos os defeitos encontrados. • Compilar o programa, consertar e declarar todos os defeitos encontrados. • Completar o Log de gravação de tempo • Reavaliar e reciclar o necessário
6	Postmortem	• Completar a forma do Plano de Projeto resumida com os tempos, defeitos e dados de tamanho reais. • Formulário de Resumo de Ciclo completado com dados atuais do ciclo
	Critérios de saída	• Um programa exaustivamente testado • Plano de Projeto resumido completado com dados estimados e reais • Modelos de estimativa e planejamento completados • Modelos de Design completados • Lista de revisão de Design e Código completados • Modelo de Relatório de Teste completado • Log de Controle de Emissão completado • Preencher formulários do PIP. • Log de gravação de tempo e registro de defeitos completado.

Tabela A-66. PSP3 - Script de Planejamento

Fase Numero	Proposta	Para guiá-lo no processo de planejamento
	Critérios de Entrada	• Descrição do problema • PSP3 Formulário do Plano de Projeto resumido e Resumo do Ciclo • Modelos de Planejamento de Tarefas, Planejamento de Horário, e Estimativa de Tamanho. • Histórico de dados estimados de tamanho reais e dados de tempo • Log de gravação de tempo
1	Requisitos do Programa	• Produzir ou obter uma declaração de requisitos • Assegurar que a declaração de requisitos está clara e não haja ambigüidade • Resolver quaisquer questões
2	Estimar Tamanho	• Produza um projeto de um programa conceitual • Use o método PROBE para fazer sua melhor estimativa do total de linhas novas e alteradas de código utilizadas para desenvolver este programa • Estime linhas de código base, adicionadas, removidas e reusadas • Complete o Modelo de Estimativa de Tamanho e o Resumo do Plano de Projeto • Calcule a 70 por cento do intervalo de predição de tamanho
3	Estratégia de Desenvolvimento Cíclico	• Subdivida o desenvolvimento do programa em módulos de aproximadamente 100 e não mais que 250 linhas de código novas e modificadas. • Aloque as linhas de código para serem desenvolvidas entre os ciclos • Entre com os dados de tamanho no Plano de Resumo do Ciclo
4	Recursos estimados	• Use o método PROBE para estimar o tempo requerido para o desenvolvimento deste programa • Calcule a 70 por cento do intervalo de predição de tempo • Subdivida o tempo total de desenvolvimento entre os ciclos desenvolvidos • Usando o para Data% a partir do último programa desenvolvido como um guia, distribuir o tempo de desenvolvimento sobre as fases do projeto planejadas de cada ciclo desenvolvido. • Entre com os dados de tempo no Plano de Resumo do Ciclo
5	Planejamento de Horários e Tarefas	Para projetos que requerem vários dias ou mais de trabalho, complete os Modelos de Planejamento de Horário e Tarefas.

6	Defeitos Estimados	• Baseado nos dados Datados por defeitos por novas e modificadas linhas de código, estime o total de defeitos a serem encontrados no programa • Baseado nos dados de porcentagem de Datados, estime o numero de defeitos a serem injetados e removidos por fase
	Critérios de saída	• Uma documentada declaração de requisitos • O projeto de um programa conceitual • Modelo de Estimativa de Tamanho completo. • Resumo do Plano de Projeto e Resumo do Plano do Ciclo concluído com os dados de tempo estimado de desenvolvimento • Log de gravação de tempo concluído.

Tabela A-67. PSP3- Script de Projeto de Alto Nível

Fase Numero	Proposta	Para guiá-lo no processo de design de alto nível – PSP3
	Critérios de Entrada	Cheque que o seguinte está em mãos: • Os requisitos declarados • O Design Conceitual • Estimativas de Tamanho e tempo • O Log de Controle de Emissão
1	Especificações externas	• Complete os Modelos de Cenário Operacional para todos os normais e anormais usos de funções externas • Complete um Modelo de Especificação de Estado para o programa completo • Complete um Modelo de Especificação Funcional para o programa global
2	Design do Módulo	• Subdivida o programa em varias unidades de modulares • Especifique as características externas de cada uma desses módulos com Modelos de Cenário Operacional e especificação funcional • Complete os Modelos de especificação Lógica e de Estado para o mais alto nível principal ou controle de rotina • Grave as principais questões ou preocupações no Log de Controle de Emissão.
3	Protótipos	Identificar, planejar e executar experimentos necessários de protótipos.
4	Estratégia de desenvolvimento	Decida uma estratégia para desenvolver e testar os programas em ciclos. Onde possível • Mantenha cada ciclo de cerca de 100 LOC de novas e modificadas • Desenvolva módulos completos em um ciclo

		• Minimizar a quantidade de testes escalonados, e • expor os riscos de desenvolvimento mais importantes o mais cedo possível
5	Documentação da estratégia de desenvolvimento	Produza uma estratégia de desenvolvimento • Defina no programa as funções e/ou módulos à serem produzidos em cada ciclo de desenvolvimento • Especifique a abordagem de teste para integrar progressivamente os módulos em cada ciclo.
6	Log de Controle de Emissão	Revise o Log de Controle de Emissão durante o projeto e faça adições ou mudanças apropriadas.
	Critérios de saída	• O Design de todo o programa, incluindo os modelos de especificações para o principal ou rotina de controle. • A especificação do design para todos os módulos componentes do programa planejado ♦ Modelos de Especificação de Funcional ♦ Modelos de Cenário Operacional • A estratégia de desenvolvimento e teste • Também: ♦ Um completo Formulário de Resumo de Ciclo com data do plano ♦ Log de Controle de Emissão atualizado.

Tabela A-68. PSP3 - Script de Revisão de Projeto de Alto Nível

Fase Numero	Proposta	Para guiá-lo no processo de revisão do design de alto nível PSP3
	Critérios de Entrada	Cheque que o seguinte está em mãos: • Os requisitos declarados • A especificação do projeto para todos os módulos componentes planejados, incluindo os Modelos de Cenário Operacional e Especificação funcional • Todos os Modelos de Especificação de projeto para o programa total e rotina principal • A estratégia de desenvolvimento em ciclo • O Log de Controle de Emissão
1	Cobertura de Design	• Verifique que os requerimentos são cobertos pelo design • Todas as funções requeridas estão especificadas • Os tópicos de alto nível de design no Log de Controle de Emissões foram abordados • Os materiais exigidos para o design foram produzidos
2	Verificação de Estado de Máquina	Verifique o design de estados de máquinas • Os estados são completos e ortogonais • As condições de transições de cada estado são completas e ortogonais
3	Verificação de Lógica	Verifique a lógica do design de alto nível. Use um método definido assim com segue: • Tabelas de execução • Traçar tabelas • Provas de verificação
4	Verificação da consistência do Design	Verifique que todos os nomes especiais e tipos estão claros, consistentes e de acordo com o padrão estabelecido.
5	Verificação de Reuso	Verifique que as funções de reuso são avaliadas e que seu uso planejado é adequado
6	Verificação da Estratégia de Desenvolvimento	Revise a estratégia de desenvolvimento para garantir que • Todas as funções requeridas estão providas • As funções necessárias são avaliadas constantemente com a estratégia de teste, e • A estratégia de teste cobre todas as funções exigidas e os principais estados do programa
7	Correção dos Defeitos	• Corrigir todos os defeitos encontrados • Registre os defeitos no Log de Registro de defeitos
	Critérios de Saída	Na conclusão, deves ter o seguinte: • O design de alto nível completado • Log de Controle de Emissão atualizado • Log de Registro de tempo completo • Log de Registro de defeitos completo

Tabela A-69. PSP3 - Script de Desenvolvimento

Fase Numero	Proposta	Para guiá-lo no processo de desenvolvimento de programas no nível compomente
	Critérios de Entrada	• Especificações de módulos funcional e operacional • Estratégia de Desenvolvimento e Teste • Revisão de design de alto nível completada • Log de Controle de Emissão atalizado • Log de gravação de tempo e registro de defeitos • Padrão de tipos de defeitos e padrão de codificação
1	Design	• Revisar os requisitos de modulo e produzir uma especificação externa para atingi-los. • Complete os Modelos de Especificação Funcional e Cenário Operacional para registrar esta especificação. • Produzir um projeto para atender a essa especificação. • Registre o Projeto nos Modelos de Especificação Funcional, de Estado, Lógico e Cenário Operacional como requerido. • Completar o projeto para os materiais do módulo de teste e instalações. • Registre no Log de Registro de Defeitos qualquer defeito encontrado no requerimento • Registrar tempo no registro de gravação de tempo
2	Revisão de design	• Siga a Lista de Revisão de Design e revise o design • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
3	Código	• Implementar o projeto seguindo o padrão de codificação • Registrar no registro de defeitos qualquer defeito nos requisitos e projeto encontrados. • Registrar tempo no registro de gravação de tempo
4	Revisão de Código	• Siga a Lista de Revisão de Código e revise o modulo e código de teste • Conserte todos os defeitos encontrados • Registre os defeitos encontrados no Log de Registro de Defeitos • Registre o tempo no Log de Registro de tempo
5	Compilação	• Compilar o programa e os materiais de teste até ser livre de erros • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo

6	Teste	• Testar os módulos até todos os testes executarem sem erro • Consertar todos os defeitos encontrados • Registrar os defeitos no Log de Registro de defeitos • Registrar o tempo no Log de Registro de tempo • Complete o Modelo de Relatório de Teste com testes conduzidos e os resultados obtidos
7	Reavaliação e Reciclagem	• Registre os dados no ciclo de desenvolvimento • Reavaliar o estado contra o plano e decidir continuar como planejado ou fazer alterações.
	Critérios de saída	• Um Programa totalmente testado conforme o padrão de codificação • Modelos de Design completos. • Lista de Revisão de Design e Código completos. • Formulário de Resumo de Ciclo atualizado com a data atual • Modelo de Relatório de Teste completo • Log de gravação de defeitos concluído • Log de gravação de tempo concluído • Log de Controle de Emissão atualizado

Tabela A-70. PSP3 - Script de Postmortem

Fase Numero	Proposta	Para guiá-lo no processo de Postmortem
	Critérios de Entrada	• Descrição do problema e requisitos declarados • Resumo do Plano de Projeto com o tamanho do programa, tempo de desenvolvimento e dados de defeitos e intervalo de predição. • Resumo do Ciclo completo • Para projetos de vários dias de duração, Modelos de Planejamento de Horário e Tarefas completados. • Modelo de Relatório de Teste completo • Modelos de Design completos • Lista de Revisão de Design e Código completas • Log de registro de tempo concluído • Log de registro de defeitos concluído • Um programa testado e rodando conforme o padrão de codificação • Log de Controle de Emissão atualizado
1	Defeitos injetados	• Determinar do Log de registro de defeitos o numero de defeitos injetados em cada fase do PSP 3 • Entre com este no resumo do plano de projeto no campo de Defeitos injetados – Atuais

2	Defeitos Removidos	• Determinar do Log de registro de defeitos o numero de defeitos removidos em cada fase do PSP 3 • Entre com este no resumo do plano de projeto no campo de Defeitos removidos – Atuais • Calcular o rendimento de todo o processo atual e inseri-lo no Resumo do Plano do Projeto.
3	Tamanho	• Contar as linhas de código no programa concluído • Determinar a base, reutilizadas, excluídas, modificadas, adicionadas, totais, novas e alteradas, e novas linhas de códigos reutilizadas. • Entre com esses dados no resumo do Plano de Projeto.
4	Tempo	• Revise o Log de registro de tempo concluído • Entre com o total de tempo gasto in cada fase do PSP 3 na coluna Atual do resumo do Plano de Projeto.
	Critérios de saída	• Um programa totalmente testado conforme o padrão de codificação. • Modelos de Design Completos. • Lista de Revisão de Design e Código completadas • Um Modelo de Relatório de Teste completo. • Resumo do Plano de Projeto concluído. • Formulário do PIP concluído descrevendo os problemas enfrentados no processo, sugestões de melhoria e lições aprendidas. • Log de registro de tempo e defeitos concluídos • Log de Controle de Emissão atualizado

Tabela A-71. PSP3 – Resumo do Plano de Projeto

Estudante		Data	
Programa		Programa #	
Instrutor		Linguagem	

Resumo	**Plano**	**Real**	**Datado**
Tamanho/Hora			
Tempo Planejado			
Tempo Real			
CPI (Índice de Custo-Desempenho)			(Planejado/Atual)
% Reusado			
% Novo Reutilizável			
Defeitos Teste/KLOC			
Total Defeitos/KLOC			
Rendimento %			
% Avaliação CDQ			
% Falha CDQ			
COQ A/F Taxa			

Tamanho do Programa (LOC)	**Plano**	**Real**	**Datado**
Base(B)	(Mensuradas)	(Mensuradas)	
Apagadas (AP)	(Estimadas)	(Contadas)	
Modificadas (M)	(Estimadas)	(Contadas)	
Adicionadas (A)	(N – M)	(T-B+AP-R)	
Reutilizadas (R)	(Estimadas)	(Contadas)	
Total Novas & modificadas (N)	(Estimadas)	(A+M)	
Total Linhas de Código (T)	(N + B – M – D + R)	(Mensuradas)	
Total Nova(s) Reutilizada(s)			
Intervalo de Predição Superior **(70%)**			
Intervalo de Predição Inferior **(70%)**			

Tempo na Fase (min.)	**Plano**	**Atual**	**Até o momento**	**Até o momento %**
Planejamento				
Design de Alto-nível				
Revisão Design de alto-nível				
Design detalhado				
Revisão Design detalhado				

Código				
Revisão Código				
Compilação				
Teste				
Postmortem				
Total				
Tempo Total IPS (70%)				
Tempo Total IPI (70%)				

Defeitos Inseridos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design de Alto-nível			
Revisão Design de alto-nível			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			

Defeitos Removidos	**Atual**	**Até o momento**	**Até o momento %**
Planejamento			
Design de Alto-nível			
Revisão Design de alto-nível			
Design			
Revisão Design			
Código			
Revisão Código			
Compilação			
Teste			
Total Desenvolvimento			
Depois de desenvolvimento			

Eficiência de Remoção de defeito	**Plano**	**Real**	**Datado**
Defeitos/hora – Revisão Design			
Defeitos/hora – Revisão Código			
Defeitos/hora – Compilação			
Defeitos/hora – Testes			
DRL(DLDR/UT)			
DRL(Revisão Código/UT)			
DRL(Compilação/UT)			

Tabela A-72. PSP3 – Resumo do Plano de Projeto – Instruções

Proposta	Este formulário contém os dados reais estimados do projeto em uma forma conveniente e facilmente recuperáveis.
Cabeçalho	Entre com o que segue: • Nome e data de hoje; • O nome do programa e seu numero; • O nome do instrutor; • A linguagem que será usada para escrever o programa
Resumo	• Entre com o novo e alterado valor de LOC por hora planejado e real, e para todos os programa desenvolvidos insira no campo Datado • Entre com os tempos reais e planejados para este programa e a soma de todos os tempos planejados e reais de para todos os exercícios datados • CPI = (Tempo planejado datado) / (Tempo Real datado) • Entre com os dados de reusados para planejados, reais e datados. • Entre com os dados de defeitos planejados, reais, e datados • Entre com o rendimento planejado e real
Custo da Qualidade	• Entre com a % de Avaliação CDQ: a porcentagem do tempo de desenvolvimento gasto na Revisão de Design e Código • Entre com a % de Falha CDQ: a porcentagem do tempo de desenvolvimento gasto na Compilação e Testes • Entre com a taxa A/F: o índice de Avaliação CDQ dividido por Falha CDQ
Tamanho do programa (LOC)	Antes do desenvolvimento: • Se você estiver modificando ou melhorar um programa já existente, contar que as linhas do programa de códigos e insira embaixo de Base – Plano. • Entre com o valor de novas e modificadas LOC (N) do Modelo de Estimava de Tamanho. • Estimar o numero de LOC adicionadas (A) e modificadas(M), então faça N = A + M. • Estimar o numero de LOC deletadas (D) e reusadas (R) e combinar com o número de LOC da base (B), T = N + B –M –D + R. Depois do desenvolvimento: • Se as linhas de código base foram modificadas, entre com o novo valor • Mensure o tamanho total do programa e insira no Total de linhas de código – Atual • Reveja o código fonte e determine o numero atual de linhas de código que foram apagadas (A), modificadas (M), ou reutilizadas (R) • Calcule o numero de linhas de código adicionadas A = T-B+D-R • Calcule o total de novas e reutilizadas linhas de código N = A+M

Tempo em fase	• Abaixo do plano entre com a estimativa original do tempo de desenvolvimento total e o tempo requerido em cada fase • Abaixo de atual, entre com o tempo real em minutos usados em cada fase de desenvolvimento • Abaixo de Até o momento, entre com a soma do tempo atual e o tempo Até o momento do programa mais recente desenvolvido • Abaixo de Até o momento%, entre com a porcentagem de tempo Até o momento em cada fase.
Defeitos inseridos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase
Defeitos Removidos	• Abaixo de Atual, entre com o numero insertos em cada fase • Em Até o momento, entre com a soma dos defeitos insertos atuais e defeitos insertos Até o momento do mais recente programa • Abaixo de Até o momento%, entre com a porcentagem dos defeitos Até o momento insertos em cada fase • Depois do desenvolvimento, registre qualquer defeito posteriormente encontrada durante o uso do programa, reuso ou modificação
Eficiência de Remoção de defeito	• Abaixo de Plano, entre com as eficiências planejadas para este projeto • Abaixo de Real, entre com as eficiências reais atingidas • Abaixo de Datado, entre com as eficiências reais atingidas para todos os projetos até à data.

Tabela A-73. Resumo de Ciclos

Plano ___ Real ___

Estudante		Date	
Programa		Programa #	
Instrutor		Linguagem	

Ciclos	**Datados**	**1**	**2**	**3**	**4**	**5**	**Total**
Tamanho Programa (LOC):							
Base(B)							
Excluídos (D)							
Modificados (M)							
Adicionados (A)							
Reusados (R)							
Totais Novas & Alteradas (N)							
Total LOC (T)							
Totais Novas reutilizadas							
Tempo em Fase (min.)							
Design							
Revisão do Design							
Código							
Revisão do Código							
Compilação							
Teste							
Total							
Defeitos injetados							
Design							
Revisão do Design							
Código							
Revisão do Código							
Compilação							
Teste							
Total							
Defeitos Removidos							
Design							
Revisão do Design							
Código							
Revisão do Código							
Compilação							
Teste							
Total							

Tabela A-74. Resumo de Ciclos – Instruções

Proposta	Para guardar os dados reais e estimados de desenvolvimento do ciclo em um formulário conveniente e de fácil recuperação A menos que tais dados sejam prontamente registrados quando o ciclo é planejado e novamente quando concluído, eles vão ser difíceis de reconstruir. Um registro histórico desses dados fornece uma base sólida para o planejamento e acompanhamento de projetos maiores.
Plano / Real	Este formulário pode ser usado para guardar os dados planejados ou reais do ciclo. Cheque qual é o caso.
Cabeçalho	Entre com a data especificada
Geral	Se mais de cinco ciclos são usados, use múltiplas cópias deste formulário, e substitua os números dos ciclos
Datado	• Esta coluna geralmente será vazia, a menos que antigos ciclos tenham sido registrados em outro formulário. • Quando esta é uma segunda folha ou anterior, entre aqui com o total de colunas do Resumo de Ciclos para os ciclos anteriores
1,2,3,4,5	Entre com os dados planejados e reais para cada ciclo
Total	Total de números para todos os ciclos, incluindo a coluna Datada.
Tamanho do Programa	Estes dados são particularmente difíceis de reconstruir após o fato. Assim, é aconselhável para medir e registrar os valores reais durante ou imediatamente após a conclusão de cada ciclo.
Tempo em Fase	• Estes dados são relativamente simples de reconstruir, enquanto as entradas de registro de tempo incluem uma notação para o ciclo de desenvolvimento envolvido. • Esta notação pode ser feita no espaço de comentários.
Defeitos	• Os dados de defeito podem ser facilmente reconstruídos se as entradas de fase incluem um número de ciclo. • O ciclo da fase removida pode ser facilmente determinada no momento de remoção • O Ciclo da fase injetada pode ser difícil determinar

Tabela A-75 – PSP3 - Lista de Revisão de Projeto

NOME DO PROGRAMA E NÚMERO

Proposta	Para guiá-lo em uma eficaz realização de Revisão de Projeto				
Geral	• Quando completar cada passo da revisão, marque este item na caixa à direita • Complete a lista para uma unidade do programa antes de iniciar a revisão da próxima • Quando encontrares problemas cuja resolução deve ser diferido, gravá-los no Log de Controle de Emissão				
Completar	Garanta que os requerimentos, especificações, e o design de alto nível são completamente cobertos pelo projeto • Todos os resultados especificados são produzidos. • Todos os insumos necessários são fornecidos • Todos as inclusões requeridas são iniciadas				
Estado de Maquina	Verifique o design do Estado de Maquina • A estrutura não tem armadilhas escondidas ou loops • Isto está completo, isto é, todos os possíveis estados foram identificados • É ortogonal, isto é, para cada conjunto de condições há um e somente um estado mais próximo possível. • A transição de cada estado são completas e ortogonais. Isto é, de cada estado um estado único próximo é definido para cada combinação possível de valores de estado de entrada da máquina.				
Lógica	• Verifique que a seqüência do programa é adequada: o Pilhas, listas, e assim por diante estão na ordem adequada o Recursões se comportam corretamente • Verifique que todos os loops são adequadamente iniciados, incrementados e finalizados. • Use métodos de verificação definidos, tais como tabelas de execução, tabelas de rastreamento, ou verificação matemática.				
Casos Especiais	Cheque todos os casos especiais: • Garantir o funcionamento adequado com vazio, cheio, mínimo, máximo, negativos, valores zero para todas as variáveis. • Proteger contra saídas de limite, condições de overflow e underflow • Garantir que condições “impossíveis” são absolutamente impossíveis • Manipular todas as condições de entrada incorretas				

Uso Funções	• Verifique que todas as funções, procedimentos, ou objetos são totalmente compreendidos e propriamente usados. • Verifique que todas as Abstrações referenciadas externamente são justamente definidas.				
Nomes	Verifique o seguinte: • Todos os nome especiais e tipos são claros ou especialmente definidos • Os escopos de todas as variáveis são auto-evidentes ou definidos • Todos os objetos nomeados são usados dentro de seus escopos declarados				
Padrões	Revisar o Design para a conformidade com todos os padrões de design aplicáveis.				

Tabela A-76. PSP Log de Controle de Emissão

Estudante ________________________ Data ________
Programa ________________________ Programa # ________
Instrutor ________________________ Linguagem ________

ITL Número _____

Controle #:		Data:		Fase:	
Descrição:					
Resolução:					
Data:					
Controle #:		Data:		Fase:	
Descrição:					
Resolução:					
Data:					
Controle #:		Data:		Fase:	
Descrição:					
Resolução:					
Data:					
Controle #:		Data:		Fase:	
Descrição:					
Resolução:					
Data:					
Controle #:		Data:		Fase:	
Descrição:					
Resolução:					
Data:					

Tabela A-77. PSP Log de Controle de Emissão – Instruções

Proposta	• Para prover um caminho ordenadamente para registrar, traçar, e gerenciar os Controles do projeto • Em grandes projetos, Controles são geralmente perdidos ou esquecidos porque não há nenhuma maneira ordenada para gravá-los. • Quando desenvolvedores adiam Controles para depois manipula-los, eles precisam garantir que eles não estão esquecidos.
Geral	Use o Log de Controle de Emissão como segue: • Para registrar controles que são deferidos para depois manipula-los • Para registrar problemas potenciais que devem ser checados • Para registrar controles que podem ser negligenciados • Para priorizar os controles a serem endereçados em cada fase • Para manipular o controle de emissão Mantenha o Log de Controle de Emissão em mãos quando usar o PSP3 e registre todos os problemas que necessitam de uma futura atenção
ITL Número	Se múltiplas paginas são requeridas, entre com um único número para identificar cada página do Log de Controle de Emissão
Número do Controle	• Numere cada Controle • Para evitar confusão, esses números devem ser únicos • Onde varias pessoas são envolvidas em um projeto comum, tais devem ser atribuídas a uma serie de números tais com 1001, 2001, etc
Descrição	• Descreva o controle tão claramente possível, por exemplo: ♦ Esta mudança deve ser feita ♦ A questão que deve ser verificada ♦ O teste deve ser executado ♦ A fase na qual isto foi encontrado • Descreva o controle em suficientes detalhes, então mais tarde podes ter ações sugestivas
Data	Entre com a data na qual foi identificado e registrado o Controle
Resolução	Descreva como o Controle foi resolvido
Data	Entre com a data que o Controle foi resolvido
Emissão	Mantenha o Log de Controle de Emissão em um lugar central para consulta, no começo e durante cada fase de desenvolvimento