4 Scrum em grandes projetos

Um dos maiores desafios em projetos de grande porte é manter a comunicação na dose certa. De acordo com o *Bull Survey* 57% dos projetos falham por comunicação inadequada entre as partes importantes do projeto. [Bull Survey, 1998]

Quando falamos em grandes projetos, é necessário observar os canais de comunicação dentro de um time grande. Quanto maior o time, mais canais de comunicação são abertos e obviamente aumenta o tempo desprendido para alinhar a comunicação entre os membros.

A fórmula abaixo, exemplifica de uma maneira simplista e não exata o aumento dos canais de comunicação visto o aumento das pessoas diretamente envolvidas no projeto.

Communication channels = N*(N-1)/2, da ordem de n**2.

Havendo 3 pessoas, 3 canais de comunicação. 5 pessoas, 10 canais, 7 pessoas 21...

Pode-se concluir que um dos fatores decisivos no sucesso de um projeto é o tamanho do time, diretamente conectado à quantidade de canais de comunicação. Ao iniciar um novo projeto usando Scrum, uma das primeiras decisões é como organizar indivíduos em times, considerando seu tamanho e as habilidades necessárias para execução do projeto.

Alguns problemas comuns de comunicação que ocorrem em projetos são: não entendimento completo dos requisitos, decisões incorretas de arquitetura não comunicada entre os envolvidos, expectativa gerencial não claramente comunicada, falta de transparência na comunicação de riscos, etc.

4.1. Aplicação em times pequenos

O tamanho ideal de um time de Scrum é aproximadamente sete pessoas (7 ± 2), [Schwaber, K. 2007]. Com essa quantidade de pessoas, normalmente é possível agregar todas as especialidades necessárias em um projeto de desenvolvimento de software.

Times maiores do que nove pessoas têm sua produtividade reduzida e os mecanismos de controle propostos pelo Scrum tornam-se pesados. Liderar e manter um *Daily Meeting* efetivo e dentro de um tempo determinado torna-se difícil para o *Scrum Master*.

É altamente recomendado quebrar em dois ou mais times pequenos caso se tenha mais de nove pessoas disponíveis. Recomenda-se compor um time, com as habilidades necessárias para resolver o problema proposto, e selecionar um *Sprint Backlog* que esse time reduzido consiga se comprometer. O mesmo processo deverá ser feito para o outro time resultante da divisão, com o restante do *Backlog* que o primeiro time não pode se comprometer.

Existem outras vantagens na manutenção de times pequenos, tais como:

Gerenciamento dos pares: com um time pequeno, o próprio time observa facilmente o comportamento e a codificação produzida pelos seus pares, sendo claro identificar pontos de melhoria no dia-a-dia, pois todos sabem o que cada um está fazendo, e podem ajudar caso observem que alguma atividade não esta sendo realizada como deveria.

Relação mais próxima: A proximidade das pessoas facilita a construção de confiança mutua e interdependência. "*The Five persistent feelings of Superior Work Teams: inclusion, commitment, loyalty, pride and trust.*" Os cinco sentimentos persistentes num time de trabalho superior: inclusão, compromisso, lealdade, orgulho e confiança [Kinlaw, D. C. 1990].

Especializações ficam mais visíveis: um problema comum dos times de desenvolvimento é quando somente um desenvolvedor detém o conhecimento de uma parte do sistema. Isso faz com que o time torne-se dependente do conhecimento desse desenvolvedor, uma possível ausência como férias, doença ou um eventual atraso em um dia comum de trabalho, faria com que o time não pudesse prosseguir em um eventual ajuste dessa parte do sistema. Com o time pequeno, essa especialização torna-se visível, durante as reuniões de planejamento e inclusive durante os *Daily Meetings*. Nesse caso o próprio time consegue visualizar o problema e diluir esse conhecimento.

A comunicação é mais quente: Como num time pequeno as pessoas têm maior facilidade para se reunir e se comunicar face a face, a efetividade da comunicação é muito maior, pois se dispensam artifícios "frios", tais como comunicação documental.

Como pode ser visto no gráfico abaixo (*Richness of Communication Channel*), quanto maior a impessoalidade na comunicação, menor a efetividade dessa comunicação. O gráfico baseia-se na teoria da riqueza da mídia, que analisa um meio de comunicação por sua capacidade de reproduzir as informações enviadas sobre ele. Um exemplo disso é uma informação enviada por email, mesmo que seja redigido de forma clara ele exclui tom de voz, gestos e expressões faciais que eventualmente poderiam trazer mais informações facilitando o entendimento do que está sendo dito [Daft; Lengel, 1984].

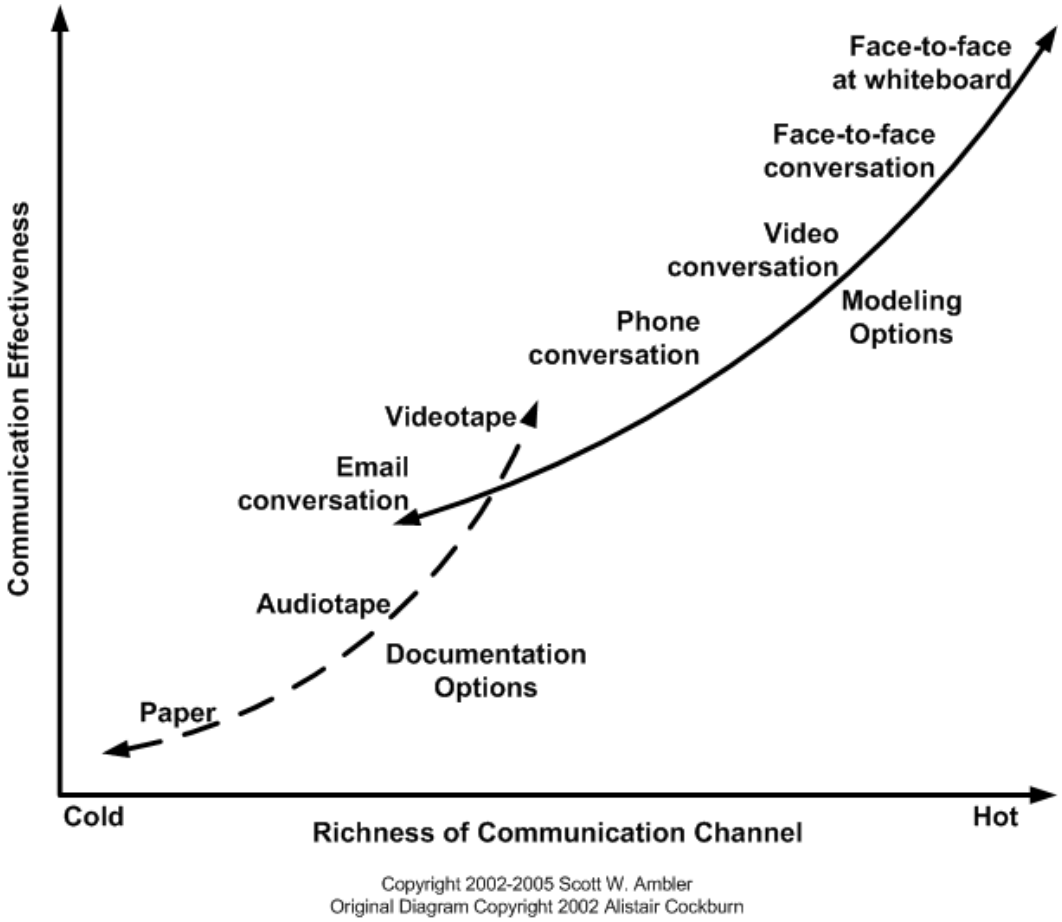

Figura 11 – Riqueza dos canais de comunicação

[Ambler, 2005 / Cockburn, 2002]

4.2. Aplicação em times grandes

No seu livro "*The Enterprise and Scrum*", Ken Schwaber [Schwaber, 2007] descreve uma tentativa de criar um time de 17 pessoas enquanto implementava Scrum em uma determinada empresa.

No primeiro momento ele achou que poderia funcionar, que o time iria se organizar de forma a trabalhar com esse tamanho, mas não aconteceu assim. Durante as reuniões, era difícil escutar as pessoas. O número de pessoas era grande, logo, era muito fácil surgirem conversas paralelas. Muitas coisas eram

ditas e muitas idéias surgiam, e era impossível manter todo o time focado no que estava acontecendo naquele momento.

Instintivamente o time começou a se dividir em pequenos grupos, e depois de algum tempo acabou se dividindo em dois times completos, cada um com seu PO e SM. Após a divisão, os times criaram representantes que se reuniam diariamente após os seus devidos *Daily Meetings* para a uma reunião conhecida como *Scrum of Scrums* [Schwaber, 2007].

Em contrapartida Martin Fowler em seu artigo "*The New Methodology*" [Fowler, 2005], comenta sucesso com times de 100 pessoas distribuídas em vários continentes quando exemplifica a possibilidade de uso de metodologias ágeis em projetos grandes, ressaltando que para que possa ser realizado, a empresa e os times já devem ser proficientes nessas metodologias.

4.3. Casos de empresas conhecidas

4.3.1. Caso SirsiDynix e StarSoft Development Labs

Jeff Sutherland, Peter Vaihasky e Anton Victorov descrevem em seu artigo "*Hyperproductivity In Large Projects Though Distributed Scrum*" a experiência no projeto complexo chamado *Horizon 8.0* que pode ser comparado com um sistema de gestão empresarial contendo um portal público acessado por mais de 200 mil pessoas, em uma pareceria da *SirsiDynix* e *StarSoft Development Labs* que continha 56 desenvolvedores, distribuídos entre Estados Unidos, Canadá e Rússia [Suthelrand; Vaihasky; Victorov, 2006].

Nesse caso além de termos a aplicação de Scrum em um time grande, podemos observar outra dificuldade no uso naturalmente conhecido de Scrum, os times estão distribuídos em três países distintos.

Uma das práticas destacadas no artigo foi uma maneira dos times das diversas regiões sincronizarem uma vez por dia através de uma reunião com representantes dos times (*Scrum of Scrums*).

Qualquer desenvolvedor de qualquer time em qualquer uma das localidades poderia desenvolver qualquer tarefa designada para o projeto, mapeadas através de uma ferramenta de controle de tarefas, independente de qual local fisicamente esse time estivesse.

Algumas práticas utilizadas nesse projeto foram:

Daily Meeting para todos os desenvolvedores de um mesmo time, considerando a diversidade de localidades.

• Reuniões diárias dos *Product Owners*.
• *Builds* automatizados em um repositório central a cada hora.
• A não distinção entre desenvolvedores (por país ou por time).
• Algumas práticas de XP como programação em pares e *refactoring* agressivo.

O projeto utilizou uma ferramenta única de acompanhamento de tarefas, isso deu a todos os envolvidos no projeto uma visão em tempo real do estado de cada *Sprint* além de dar relatórios gerenciais de status.

Uma das maiores dificuldades encontradas durante o projeto era a sincronia entre times, já que existia uma diferença de dez horas entre St. Petersburg e Utah. Para solucionar esse problema o time enviava por email tudo o que havia sido discutido no seu *Daily Meeting* e os demais times liam antes que o seu fosse iniciado.

Além das notificações por email, o time Utah passou a se reunir mais cedo, as 7:45h, 17:45h no horário de St. Petersburg, dessa forma eles podiam fazer teleconferências para tirar eventuais dúvidas.

Além do *Scrum of Scrums* diários dos desenvolvedores, *Scrum Masters*, *Product Owners* e Arquitetos, faziam toda segunda-feira pela manhã o seu *Scrum of Scrums* para garantir a coordenação dos requisitos e performance consistente entre os times.

No inicio do projeto a SirsiDynix trouxe um dos melhores seniores para St. Petersburg para passar uma semana treinando seus engenheiros em algumas peculiaridades do projeto. Depois disso fizeram o mesmo com um engenheiro de Utah, após algum tempo, alguns desenvolvedores chave de St. Petersburg vieram para os escritórios da SirsiDynix para treiná-los.

Ter esses desenvolvedores circulando entre US e St. Petersburg ajudou a atingir dois objetivos. Compartilhar conhecimento e criar união entre esses grupos. A partir disso, trabalhar em componentes ou tarefas inter-relacionadas se tornou muito mais fácil.

4.3.2. Caso Boeing 777

Nesse caso em particular não foram utilizadas as metodologias abordadas na dissertação até o momento, mas o caso mostra algumas práticas aonde podemos fazer paralelo com práticas ágeis conhecidas.

Diante de uma crise no setor de aviação, agravada pela grande competição entre os fabricantes de aviões, a Boeing se viu na necessidade de criar um novo avião para suprir a demanda da aviação comercial. Um avião com maior capacidade de passageiros e carga do que os existentes com o consumo inferior do que as aeronaves de grande porte existentes. Nenhuma fabricante possuía o exato avião necessário pelas empresas de aviação e para se manter a frente de seus concorrentes a Boeing precisava aproveitar essa oportunidade, e assim se manter a frente por mais algumas décadas [BPS/BBC, 1996].

United Airlines, uma das empresas de aviação comercial parceira da Boeing de longa data tinha em suas mãos propostas de construção de diversas fábricas distintas, entre elas a Boeing e a Airbus. A escolha direcionaria a linha de aeronaves que a empresa trabalharia pelos próximos trinta anos

De uma forma inédita na história da United Airlines e a Boeing, decide-se construir em conjunto um avião plenamente funcional de forma a atender não só as necessidades e expectativas do cliente mas dos passageiros, tripulação, mecânicos e todos envolvidos na utilização da aeronave.

Com isso a United Airlines se comprometia investir 3.5 bilhões de dólares em um avião que ainda não existia. Comprando antecipadamente 34 aviões e a opção de comprar outros 34 após três anos.

Em 1992 a Boeing começou a agrupar e treinar um grupo de 10 mil funcionários para o desenvolvimento do 777, sob uma nova filosofia aonde se encorajava e até forçava a colaboração entre as pessoas.

A mudança cultural era guiada por Alan Mulally, vice presidente executivo da Boeing na época, hoje CEO da Ford, que não somente trouxe uma nova filosofia para o projeto do 777 mas também uma mudança na maneira de organizar as 10 mil pessoas envolvidas no projeto. De alguma forma eles se remetiam à maneira que trabalhavam no passado, nas primeiras construções de aeronaves, onde a produção e engenharia ficavam muito próximos, em um mesmo prédio, divididos apenas por um lance de escadas. Qualquer problema que surgisse na produção imediatamente consultavam um engenheiro que eventualmente modificava o projeto para resolver o problema, o mesmo acontecia caso um engenheiro descobrisse uma falha no projeto, no mesmo momento interrompia a produção para que o projeto fosse ajustado [Sabbagh, 1996].

Com o crescimento da empresa esse processo se tornou rígido, criando silos entre as áreas e perdendo a comunicação, com isso criou-se a filosofia do “eles” e “nós” reforçando a separação entre as áreas.

Alan Mulay criou então o que foi chamado "*Design Build Teams*" times multifuncionais responsáveis por componentes específicos da aeronave, contendo todas as pessoas necessárias para definir e construir o componente em questão. Pessoas que nunca haviam trabalhado em conjunto. A comunicação passou a ser mais aberta e transparente, franca, deixando inclusive que alguns conflitos que, eventualmente se tornavam, passarem a ser visíveis e dessa forma poderiam ser resolvidos mais rápido e de forma efetiva [Lewis, 2006].

Essas formações de times e a maneira de lidar com sua comunicação interna, faz um paralelo com a sugestão de formação dos times em projetos ágeis e seu comportamento.

A partir disso a Boeing passa nesse projeto a ter engenheiros e mecânicos trabalhando em conjunto, evitando diversas falhas comuns de comunicação

gerando uma drástica redução dos erros por falha de comunicação entre essas partes.

Alem da formação das equipes similar aos apresentados em processos ágeis, outras práticas similares foram adotadas como as listadas abaixo:

Review

Antes mesmo de construir as partes mecânicas da aeronave, foi criado um protótipo em tamanho real do interior e exterior da aeronave, para que os investidores pudessem ver a diferença do espaço interno proposto para o 777 em comparação aos seus concorrentes.

Protótipo testável

Durante o design da porta da aeronave, surgiu uma dúvida a respeito do seu comportamento diante de uma nevasca, ou em baixíssima temperatura. A porta deveria abrir, considerando o esforço de uma pessoa comum mesmo com uma camada de gelo na porta e seu entorno. Além do baixo esforço para abrir a porta, deveria ser considerada a possibilidade do rompimento de seus componentes diante de fadiga do uso.

Integração contínua

Um dos problemas comuns nos projetos anteriores da Boeing era que os projetos bidimensionais não consideravam perfeitamente o espaço necessário para os componentes, surgindo situações aonde, no momento da primeira montagem, surgiam conflitos entre algumas partes do avião. Era difícil, por exemplo, observar que a altura de um determinado duto de ar condicionado conflitaria com o espaço necessário para ele na fuselagem.

Mesmo considerando os protótipos o processo tornava-se custoso e demorado, e para reduzir esse custo construíram um software aonde poderiam validar o novo componente do projeto contra o projeto já adicionado nesse sistema, muito similar ao que temos nos projetos ágeis como a Integração Contínua. Essa validação poderia chegar ao nível da cabeça de um parafuso conflitando com o espaço necessário para seu encaixe.

4.3.3. Caso Globo.com

4.3.3.1. Histórico

Desde seu surgimento, a Globo.com utilizou uma conhecida ferramenta de mercado para a administração e publicação de suas matérias e páginas editorias de seus diversos produtos. Essa ferramenta permitia customizações de layout e adição de novas funcionalidades para o internauta, quanto à administração e edição por partes dos jornalistas, editores e desenvolvedores. Além de um custo alto de manutenção desenvolvimento para aplicar essas customizações, a ferramenta não trazia toda a flexibilidade e facilidade de uso desejadas aos nossos clientes. Aumentar a facilidade de uso para os editores é fundamental quando existe a necessidade de acelerar a publicação de uma notícia. Reduzir o custo e aumentar a flexibilidade de customização e facilidade criação de novas funcionalidades pelos desenvolvedores é fundamental para que cada novidade seja desenvolvida com velocidade.

Diante desse cenário surgiu a necessidade de descontinuar a ferramenta antiga de publicação. Mas qual ferramenta teria a capacidade de suprir as necessidades atuais além da evolução desejada?

A partir desses questionamentos, foi alocado um time, composto por um *Product Owner*, um *Scrum Master* e cinco desenvolvedores para estudar ferramentas de mercado para esse fim, que atendesse a todas essas necessidades, esse time ficou conhecido como Core. Seu *Product Owner* tinha conhecimento prévio de manutenção e customização da antiga ferramenta, e em conjunto com o time selecionou uma série de outras ferramentas de publicação para um estudo comparativo.

Nesse estudo foram encontradas diversas ferramentas que atendiam parcialmente as necessidades, mas nenhuma ferramenta no mercado naquele momento as atendia por completo. O time então decide unir as boas ideias de cada

ferramenta estudada e criar uma ferramenta interna, contendo a cobertura de todas as necessidades e desejos.

Após alguns estudos e algumas provas de conceito, iniciaram o desenvolvimento dessa nova ferramenta. Mas a entrega não estava de acordo com a velocidade esperada pela diretoria, após alguns meses, surge a necessidade de acelerar o desenvolvimento da ferramenta, nesse momento outros cinco times extraídos de outras áreas de desenvolvimento são movidos de seus projetos atuais e alocados no projeto da nova ferramenta de publicação.

Os times não estavam familiarizados com a tecnologia e com o projeto alguns eram novos em processos ágeis e não tinham habilidades firmadas em testes automatizados, programação em par e outras práticas ágeis.

4.3.3.2. Cenário 1: Cinco times pequenos, trabalhando em paralelo

Nesse cenário, foram adicionados cinco novos times. Juntos totalizavam 49 pessoas: 28 desenvolvedores, 4 designers, 4 arquitetos da informação, 3 especialistas em infra estrutura/bando de dados, 5 *Product Owners* e 5 *Scrum Masters*.

Durante esse cenário não eram feitos *release plans* e no momento inicial não havia um CI centralizado, mas era possível executar localmente os testes feitos previamente pelo primeiro time alocado ao projeto.

A partir da entrada desses novos times ao projeto, decidiu-se diluir os membros do time Core nos demais times, adicionando um de seus membros a cada um dos novos times de forma que pudessem agregar conhecimento do projeto a cada um dos novos times.

A primeira ação tomada foi reunir todos os times e mostrar o propósito e detalhes do projeto, de forma que todos tivessem pleno conhecimento daquilo que estavam desenvolvendo e sua importância.

Na sequência, os Sprints de todos os times foram sincronizados, para que o planejamento e revisão pudessem seguir em paralelo.

Para que a medida de pontos de complexidade (considerando o *Planning Poker*) fosse equalizada entre os times, foram feitas sessões de estimativas em conjunto, usando uma mistura de uma prática exemplificada por Ken Schwaber em seu livro *The Enterprise and Scrum* [Schwaber, 2007] e por Mike Cohn em seu artigo *Establishing a Common Baseline for Story Points* [Cohn, 2008] :

Em um auditório os cinco times foram agrupados de forma a ocuparem uma mesa cada um. Com o intuito de definirmos a história base para as demais estimativas, um *Scrum Master* mediador leu algumas histórias e todos os membros de todos os times levantando as mãos elegeram a que seria a mais simples de todas.

Na sequência uma única história era lida com seus casos de aceite, em um intervalo de tempo pré-estabelecido cada mesa discute entre si e decide um valor de complexidade.

No segundo momento cada mesa descreve soluções e dificuldades para aquela história e a complexidade estimada por seu grupo.

Após ouvir cada um dos grupos, cada mesa faz uma nova rodada de estimativa para a mesma história. O processo é repetido até que os times encontrem uma estimativa alinhada para aquela história.

Considerando que as estimativas de cada história são relacionadas, ou seja, se uma história tem 5 pontos de complexidade outra tem 8, a segunda história é quase duas vezes mais complexa que a primeira.

Após algumas rodadas de equalização, os times se sentem confiantes de que uma estimativa feita por outro é coerente com o que seu próprio time estimaria.

Em uma segunda fase, apenas para confirmação da afirmativa anterior, cada time estima uma história diferente, escolhida aleatoriamente a partir do backlog, e apenas confirma a sua estimativa com os demais times, se tudo ocorreu como esperado a estimativa está coerente com que cada time estimaria.

A partir desse momento todos os times passaram a ter o *mesmo Product Backlog*, mas *Sprint Plannings* e *Sprint Backlogs* distintos a serem desenvolvidos

em um mesmo tamanho de *Sprint*, uma vez que todos os *Sprints* estavam sincronizados.

Os *Daily Meetings* de cada time ocorriam no mesmo horário, cada time focado nas histórias de seu próprio quadro. Na sequência do *Daily Meeting* um representante do time se reunia com os demais representantes dos times no *Scrum of Scrums* para falar a respeito da história que o time estava trabalhando naquele dia e suas dificuldades, eventuais integrações ou generalizações de soluções que poderiam ser aproveitadas pelos demais times. No retorno dessa reunião o membro do time que havia participado comunica ao time qualquer ajuste da história, além de comunicá-los a respeito do que os demais times estão produzindo.

Uma vez por semana os especialistas de cada time (Arquitetos da informação, Desenvolvedores *Client-side*, Designers) se reuniam em um *Scrum of Scrums* por especialidade técnica, para discutir decisões aplicadas nos respectivos times e para manter seus padrões.

Além do *Scrum of Scrums* e as reuniões semanais por especialidade técnica, havia também uma reunião diária entre os POs de cada time, aonde discutiam as histórias que sofriam alguma alteração durante o *Sprint*, criavam novas histórias e seus casos de aceite além de planejar quais histórias eventualmente seriam separadas para os próximos *Sprints*, observando as dependências entre times.

Antes da conclusão do *Sprint*, os membros do time Core que estavam diluídos nos demais times se agrupavam para fazer uma inspeção no código e integrar as histórias de cada time que já havia concluído sua funcionalidade. Esse se tornava um momento critico para os times que ainda não haviam concluído sua funcionalidade, porque passavam a não contar com aquele que mais conhecia o projeto, membro do time original do Core.

Outro problema era a integração de cada funcionalidade ao repositório. Como cada time trabalhava em uma ramificação do repositório (*branch*), era necessário, após a inspeção do código, a integração no repositório principal, o que dificultava o próximo time da fila, uma vez que passava a ser necessário atualizar

sua ramificação antes que o time Core pudesse inspecionar seu código. Esse processo se tornava mais doloroso para os times que demoravam mais para concluir suas histórias, uma vez que cada aprovação dos times da "fila" fazia com que fosse necessária uma nova atualização de seu código, e verificação de eventuais quebras.

Ao termino do *Sprint* todos os times se reúnem novamente em um auditório para um *Review* coletivo, aonde cada time em formato de apresentação de auditório, exibe para os outros times, POs e clientes tudo o que foi desenvolvido durante esse período.

Como a cultura de testes não era difundida dentre os membros de todos os times, alguns limitadores foram estipulados no início do projeto como: Percentual mínimo de cobertura de testes e automatização de casos de aceite criados pelos POs.

Uma forma de dar visibilidade as possíveis quebras de código, foi a adição de um servidor de integração contínua (CI) que executava todos os testes automatizados do sistema a cada nova entrada de código no repositório.

O principal problema nesse cenário foi a competição entre os times. Mesmo que a colaboração fosse explicitamente estimulada, a necessidade de finalizar primeiro a funcionalidade para diminuir problemas com a integração acabou por gerar correria, bugs, testes ineficientes e falta de colaboração.

Outro grande problema era a diferença de conhecimento entre os times e o foco no conhecimento do membro do time Core alocado em cada um desses times.

Envolvidos no projeto: 49 pessoas

Duração do Cenário: 6 sprints de duas semanas.

Fatores positivos

- Visão alinhada do propósito do projeto

- Construção de uma visão única de medida de complexidade
- *Product Backlog* único
- Adição de práticas ágeis no dia-a-dia

Fatores negativos

- Muitas pessoas novas adicionadas ao projeto de uma única vez
- *Scrum of Scrums* ineficiente
- Time Core diluído sem considerar multidisciplinaridade
- Pouco conhecimento comum das tecnologias aplicadas ao projeto
- Competição entre os times

4.3.3.3. Cenário 2: Um único time, grande, composto por membros de todos os times

Como observado no caso anterior, havia uma diferença de conhecimento entre os times, principalmente focado no membro do time Core alocada no time, decidimos então juntar todos os times em um único e grande time, aonde o planejamento era feito em conjunto.

Esse cenário resolveu alguns problemas, como a integração entre desenvolvedores e distribuição do conhecimento técnico e de negócio. Uma vez que todos trabalhavam juntos perdeu-se a divisão em silos dos times originais.

Outro problema que foi diluído do cenário anterior foi a revisão e integração do código. A revisão passou a ser feita durante o desenvolvimento e as integrações eram feitas diretamente no ramo principal do repositório. Caso alguma nova entrada quebrasse os testes executados pelo servidor de integração contínua o código poderia ser revertido, sendo removido do repositório principal até seu ajuste.

Para evitar que novas funcionalidades fossem para algum ambiente antes de finalizadas, existia um sistema de permissões que controlava o acesso a qualquer

funcionalidade, logo uma funcionalidade ainda não finalizada ficaria escondida dos usuários.

Foi implantado um servidor de integração único para o projeto. Ao invés de cada desenvolvedor se preocupar somente com a sua funcionalidade, precisava, a partir de agora, preocupar-se com a integração desta no projeto como um todo. Caso sua nova funcionalidade demandasse alguma modificação estrutural, imediatamente os demais desenvolvedores saberiam da mudança e poderiam discutir a necessidade e adaptar seu código.

Mesmo tendo resolvido alguns problemas, o cenário de um time único e grande gerou alguns efeitos colaterais. A reunião de planejamento de *Sprint* ficou grande, durante o detalhamento de tarefas muitas pessoas perdiam o interesse, deixando de participar das discussões de problema. O *Daily Meeting* ficou longo, desfocado, era difícil saber o que cada um estava fazendo, e os pares não se cobravam. Facilmente um ou outro desenvolvedor perdia o *Daily Meeting* e acabava refazendo alguma tarefa que outro desenvolvedor já tinha iniciado.

Outro problema que passou a existir foi na formação de pares. Alguns desenvolvedores com mais conhecimento acabavam fazendo dupla com outros de conhecimento semelhante, o que acabava deixando desenvolvedores com menos conhecimento desenvolvendo juntos, esse segundo grupo acabava gerando código de menos qualidade.

Envolvidos no projeto: 49 pessoas

Duração do Cenário: 2 sprints de duas semanas.

Fatores positivos

- Planejamento único
- Difusão de conhecimento
- Quebra da competição
- Revisão contínua
- Servidor único de integração contínua

Fatores negativos

- Planejamento muito demorado
- Detalhamento de tarefas ficou pobre
- Dispersão nos *Daily Meetings*
- Formação de pares fixos

4.3.3.4. Cenário 3: Um time grande subdividido em frentes de trabalho

Ainda utilizando o conceito de um grande time para resolver alguns dos problemas gerados no cenário anterior mantendo a soluções que essa iniciativa trazia, misturando o cenário de time grande com o de times pequenos, essa modificação foi discutida com os times antes do início do *Sprint Planning* e implantada na sequência.

A reunião de planejamento continuava sendo feita com todos os desenvolvedores reunidos como um único time, aonde em consenso destacavam cinco ou seis problemas ou temas a serem trabalhados naquele *Sprint*. A partir disso, selecionávamos as histórias do *Backlog* que faziam sentido para resolver cada problema em questão. Uma vez os temas levantados, e as histórias agrupadas por esses temas, os desenvolvedores se ofereciam para compor um grupo responsável por resolver aquele problema. Os grupos deveriam ser mistos, agregando suas habilidades para resolver o problema em questão. Um grupo não poderia conter muito mais membros que outros, para prevenir que um problema tivesse mais atenção que outro problema levantado. Um grupo poderia "solicitar" a presença de um desenvolvedor que de acordo com eles tivesse mais conhecimento sobre um determinado assunto ou tecnologia a ser aplicada em uma das histórias. Essa formação conseguiu diluir definitivamente a questão da integração entre desenvolvedores e conhecimento técnico que já haviam obtido razoável melhoria no cenário anterior.

Como eram times dinâmicos, construídos no momento do planejamento, o grupo durante o *Sprint* poderia ceder um de seus membros caso existisse necessidade do seu conhecimento em outro time e, obviamente, caso isso não impactasse no seu desenvolvimento. O tamanho em dias do *Sprint* mantinha-se o mesmo. Caso um dos grupos concluísse suas histórias antes do previsto, poderia puxar outra história ou diluírem-se nos demais times para ajudá-los a concluir sua história.

Dessa forma, o *Planning* II tornou-se mais focado, pois o grupo responsável por resolver o problema estava comprometido em resolvê-lo e entendia do assunto. Perde-se a necessidade do *Scrum of Scrums* para que um time saiba o que o outro está desenvolvendo porque esse conhecimento foi adquirido durante o *Planning* I feito em conjunto. Outro fator que colaborou com a integração foi a possibilidade de um desenvolvedor se desagrupar para ajudar o outro time, automaticamente os desenvolvedores durante o dia conversavam entre si, principalmente entre seus times originais, para saber se poderiam ajudar de alguma forma. Nota-se nesse momento um destaque dos desenvolvedores mais proficientes, acabam passando por diversos temas no decorrer do *Sprint*.

Durante esse cenário surge o time de suporte. Composto pelos desenvolvedores que se destacaram no decorrer do projeto e parte dos desenvolvedores do time Core. Esse time passa a dar suporte para os demais times, dado a sua facilidade de transitar entre os times e o conhecimento das tecnologias empregadas no projeto. As principais demandas desse time eram levantadas pelos demais times durante seus planejamentos ou em tempo de desenvolvimento. A previsibilidade do planejamento não era clara, poderiam surgir demandas no decorrer do *Sprint* dos demais times. Dada essas circunstâncias esse time deixa de adotar Scrum para utilizar Kanban. O quadro Kanban era alimentado pelos POs dos demais times em conjunto com o PO do time Core, que moderava em conjunto com os demais POs a priorização de entrada na fila de desenvolvimento.

Ainda nesse cenário outros três times da empresa contendo 4 desenvolvedores, um PO e um SM cada, foram inseridos ao projeto, com intuito de absorver conhecimento para prepará-los para a próxima fase, a divisão dos times por áreas de negócio.

Envolvidos no projeto: 67 pessoas

Duração do Cenário: 10 sprints de duas semanas.

Fatores positivos

- *Planning* em conjunto
- Frentes de trabalho dinâmicas
- Maior integração e colaboração entre desenvolvedores
- Todos sabiam o que cada grupo estava desenvolvendo

Fatores negativos

- Ainda restava diferença de conhecimento entre as frentes de trabalho gerando não uniformidade de código.
- Necessidade de surgimento de um time de suporte

4.3.3.5. Cenário 4: times por área de negócio.

A partir desse cenário, os times são separados respeitando as áreas de negócio da empresa, no que diz respeito a produtos de publicação, Esporte, Jornalismo e Entretenimento e Publicação. Cada time torna-se independente, seguindo com o planejamento de sua área, sem a participação dos demais times, exceto em mudanças estruturais quando necessitam de alguma demanda da base do sistema, caso em que torna-se necessário o envolvimento do time Core.

Representantes de área são eleitos pelos desenvolvedores e passam a ser incluídos em discussões sobre arquitetura e tudo aquilo que de certa forma poderia impactar o seu produto.

Mantém-se o time Core como o desenvolvimento da base da ferramenta de publicação e mantenedor de seus padrões. O time é responsável não somente pelo suporte aos demais times como também por manter a plataforma base de desenvolvimento, enquanto os demais times constroem seus projetos utilizando como framework o projeto original, esse time fica também responsável pela correção de bugs e atendimento a demandas dos times.

O projeto base, chamado publicação-core passa a ser distribuído em *releases* planejadas pelo time Core e acompanhadas pelos demais times. A cada iteração do time Core ou a cada ajuste na base uma nova *release* é lançada e para que os times não fiquem muito distantes da release mais recente é criada uma política de atualização em comum acordo com as áreas. A política de atualização passa a servir não somente para manter os produtos atualizados bem como para manter o suporte em temas mais recentes.

Fatores positivos

- Um time focado na ferramenta de publicação
- Times focados na construção de produtos utilizando a ferramenta de publicação como base

Fatores negativos

- Foco na evolução do sistema deixando débitos técnicos em segundo plano

4.3.3.6. Cenário 5: Time de engenharia e time de suporte

Nesse cenário, os times continuam distribuídos por área, as modificações indicadas passam a ser feitas dentro dos times que continuam trabalhando na manutenção e evolução do sistema de base para os times das demais áreas.

Observando que a demanda de suporte e evolução do sistema em diversos momentos se tornava conflitante, decidiu-se dividir o time Core em dois times, com focos distintos.

O primeiro, time de Engenharia passa a trabalhar na reestruturação da arquitetura de componentes, visto que um dos efeitos colaterais dos primeiros cenários foi a não uniformidade de codificação e arquitetura. Esse time passa a refatorar alguns componentes com o intuito de melhorar o desempenho do sistema e a velocidade de desenvolvimento dos demais times.

Uma de suas primeiras ações foi reunir representantes dos diversos times e levantar com esses quais os principais gargalos de desenvolvimento, tudo aquilo que demandava tempo e poderia ser ajustado de alguma forma para poupar tempo de desenvolvimento e tornar a codificação mais fácil.

Dada essa participação dos times, foram levantadas histórias e traçada uma meta de melhoria de performance de desenvolvimento dos demais times, que deveria ser atingida uma vez que tais histórias fossem implementadas.

O segundo, time de Suporte tem como premissa ajudar na questão do aprendizado do uso da plataforma, tanto para desenvolvedores quanto para editores. Como o time de Engenharia a partir desse cenário passa a criar novas funcionalidades para reduzir o tempo de desenvolvimento, o time de Suporte passa a ser o primeiro time a integrar suas novas releases, não somente para validar a entrega quanto para adquirir conhecimento e repassar para os demais times.

É criado um produto, nos formatos dos produtos de publicação para facilitar no suporte, com a experimentação dessas novas funcionalidades além de matérias voltadas para o desenvolvedor e editor, contendo um conteúdo rico e visual ensinando, por exemplo, como iniciar o desenvolvimento nessa plataforma, ou como desenvolver um determinado tipo de componente. Além de vídeos explicativos, gerados também pela própria equipe de suporte.

Fatores positivos

- Um time focado na evolução da ferramenta
- Um time focado na reestruturação da arquitetura e melhoria de performance

Fatores negativos

- Necessidade do constante alinhamento entre os dois times.

4.3.3.7. Lições Aprendidas

Antes de pensar em escalar o uso de Scrum é necessário analisar se a equipe está preparada para utilizar uma metodologia ágil.

"Métodos ágeis diferem substancialmente de métodos tradicionais. Ágil não é uma correção ou ajuste daquilo que vínhamos fazendo. É uma mudança radical na pratica e cultura." [Leffingwell, 2007].

É fundamental que os *Product Owners*, *Scrum Masters* e principalmente o Time sejam treinados em seus papéis para que exista um fluxo contínuo de desenvolvimento.

Um dos fatores que facilitou a comunicação entre os membros do time na Globo.com, principalmente no que diz respeito à programação em par foi a mudança do mobiliário. Inicialmente utilizavam-se baias fechadas, aonde cada desenvolvedor tinha uma área "protegida" dos demais membros da mesma equipe, muda-se para um único mesão por equipe, nesse caso os desenvolvedores passam a ficar lado a lado com seus pares, facilitando a comunicação verbal e principalmente a visual. Os equipamentos também foram substituídos, de desktops para notebooks, aumentando a flexibilidade de movimentação nos mesões.

Outro fator que deve ser levado em consideração antes de adicionar novos times ao projeto é a preparação e estruturação do código fonte para receber os

novos componentes [Schwaber, 2004], além da preparação dos desenvolvedores nas tecnologias, ferramentas e padrões e relacionados a ele.