1. **A plataforma GClaims possui um repositório Principal que possui duas Branches(Dev/Master). A Branche de Dev é utilizada por 4 programadores e a de Master mantem a versão atual em produção. Além desse repositório possuímos mais 3 repositórios(Uma cópia do repositório principal), um para cada cliente grande e que possui diferenciações de projeto. Esses outros 3 repositórios foram criados para manter a particularidade de cada cliente. Temos alguns problemas com projetos grandes que afetam a branch de dev e com resolução de bugs e publicação de versões. O que você acha dessa configuração de Repositório e branches? Você possui uma solução melhor?**

Na minha experiencia, em primeiro lugar, cada projeto deve ter apenas um repositório. Quando se trabalha numa equipe, conflitos de código são inevitáveis. Nenhuma técnica de branching irá acabar definitivamente com conflitos, mas é possível minimiza-los. Em times pequenos e médios, eu recomendo o uso do modelo do Gitflow. Não vou me aprofundar no uso do gitflow porque existem inúmeros materiais para aprender, mas segue um bom link.

<https://medium.com/@devmrin/learn-complete-gitflow-workflow-basics-how-to-from-start-to-finish-8756ad5b7394>

Agora, se o seu problema fosse tão simples de resolver quanto abrir um link, você nem precisaria me contratar, não é? Eu não tive acesso ao código, então vou dizer o que faria com base em projetos similares nos quais trabalhei. No seu caso parece ser necessário o uso de um padrão chamado Strategy. Supondo que se identifique dois trechos de código, “*B*” no cliente “*X*” e “*C*” no cliente “*Y*”. Ambos os trechos se iniciam a partir de um trecho “*A*” e um trecho “*D*”. Eu criaria uma interface “*BC*” implementada por “*B*” e “*C*” e injetaria essa interface em “*A*” ou “*D*”. É difícil falar do seu caso sem conhecer a fundo, mas vi isso funcionar em varias empresas.

<https://en.wikipedia.org/wiki/Strategy_pattern>  
<https://en.wikipedia.org/wiki/Dependency_injection>

Inclusive o .net core já vem com injeção de dependência e funciona muito bem.

1. **A plataforma GClaims foi construída para ser um sistema único e que possibilita-se a extensibilidade de telas e regras de negócio através de plugins para atender as solicitações de customizações de cada cliente, mantendo assim a integridade do CORE. Isso se tornou inviável com o tempo por conta do tamanho do projeto e o tamanho das customizações solicitadas por cada cliente. Como você modularizaria uma aplicação enterprise em .NET para suportar um sistema de plugins(Apenas de backend) ?**

Sempre que possível eu recomendo incrementar o Core, tornando-o mais robusto e pronto para atender mais clientes. Na minha experiencia, eu divido as divergências de um mesmo projeto entre dois clientes em 3 grupos.

Leve: Um dos seus clientes requer uma chamada a um serviço de permissão para aprovar determinada requisição do usuário. Eu resolveria isso apenas com um “*if*”:

If(request.RequiresPermission && !userHasPermission(request))

throw new System.ArgumentException( string.Format(Helper.ErrorMessages.UserPermission, request.User.Name) );

Médio: Alguns clientes requerem permissão e outros não. Porém cada um trata o processo de definir permissão de maneira diferente. Lembrando que futuramente o código abaixo permite configurar dinamicamente regras para cada cliente.

If ( ClientLibrary.HasIssuingPermission && !ClientLibrary.UserHasPermissionToIssue(request) )

throw new System.ArgumentException( string.Format(Helper.ErrorMessages.UserPermission, request.User.Name) );

Grave: Se cada cliente pede um projeto completamente diferente, então não faz sentido tratar no mesmo repositório. Eu criaria um microserviço core e microserviços para cada cliente. O client(front end) faria requisições no microserviço do cliente e este faria uso do microserviço core quando necessário.

1. **A plataforma GClaims possui diversas telas para extração de relatórios. Uma em específica se chama Exportador de Dados que extrai as informações do banco e joga em uma planilha em excel para download do cliente. O problema é que essa funcionalidade, toda vez que um usuário faz uma exportação de um longo período de tempo, o servidor é onerado em até 40%, acarretando reclamações. Qual solução você daria para resolver esse problema? (Limitar o usuário não é uma opção.)**

Bom... agora você já sabe um dos motivos pelo qual indiquei o Felipe Beltrame, essa é a especialidade dele. Mas como EU resolveria. Já passei por esse cenário algumas vezes e a primeira coisa é: JAMAIS rode relatórios no banco de operação. São conceitos completamente diferentes. Bancos transacionais com tabelas normalizadas não são bons para extrair relatórios, mas são ótimos para registrar as operações cotidianas.

Eu não sei exatamente qual é o seu cenário, mas a principio eu faria um ETL na mão mesmo para um banco de relatórios otimizado para os relatórios do sistema. Eu nunca usei ferramentas de ETL, mas pode ser que seja necessário, difícil dizer sem olhar o problema a fundo. Minhas perguntas seriam: Qual o volume de dados? O módulo de relatórios já cobre tudo que os clientes costumam pedir ou tem muito para crescer ainda? O quanto essa demanda é relevante para os clientes? Justifica ter alguém dedicado a esse módulo? Esse tipo de coisa precisa ter envolvimento de alguém do negócio para decidir, porém na minha experiencia um banco com as tabelas denormalizadas e chaves cobrindo os filtros das queries já resolve 99%.

1. **Em uma arquitetura baseada em microserviços, independente de qual tecnologia seja utilizada, como seria possível manter a integridade dos dados em uma transação que necessite utilizar mais de um serviço? Isso é possível? Nos conte sua experiência com esse cenário. EX. Salvar Sinistro: - Chamada no serviço 1: OK - Chamada no serviço 2: OK - Chamada no serviço 3: Erro – ROLLBACK DO SERVIÇO 2 e 1**

Isso na verdade é bem simples, cada serviço tem seu próprio banco de dados e é owner das informações que precisa para funcionar. Quando uma mensagem é consumida pelo serviço “*A*”, caso ela contenha dados relevantes para serem sincronizados em outros serviços, ela deve ser enviada para a fila desses outros serviços também, para ser consumida e para que a integridade dos dados seja mantida.

1. **Na utilização de mensagerias é comum utilizarmos SAGA. Nos conte como você já trabalhou com esse conceito e como foi aplicado!**

Nunca utilizei o conceito de SAGA. Eu sei do que se trata, mas nunca usei em nenhum projeto. Eu uso atualmente RabbitMQ e Masstransit rodando no .net core no modelo de publisher/subscriber.

1. **Quais ORM’s você teve experiência, qual achou melhor e qual achou pior e porque dessa opinião.**

Dentro do escopo do .net core eu já usei NHibernate, EF e Dapper. Eu gosto muito da flexibilidade do Dapper, porem na minha experiencia o NHibernate e o EF têm uma performance bem melhor. Tudo depende dos requisitos do projeto, mas pessoalmente eu gosto mais do Dapper.