# Dark Java

A programação como uma arte infernal

... ou

Por que deixar as coisas simples, se você pode complicá-las?

# Do que trata esta palestra?

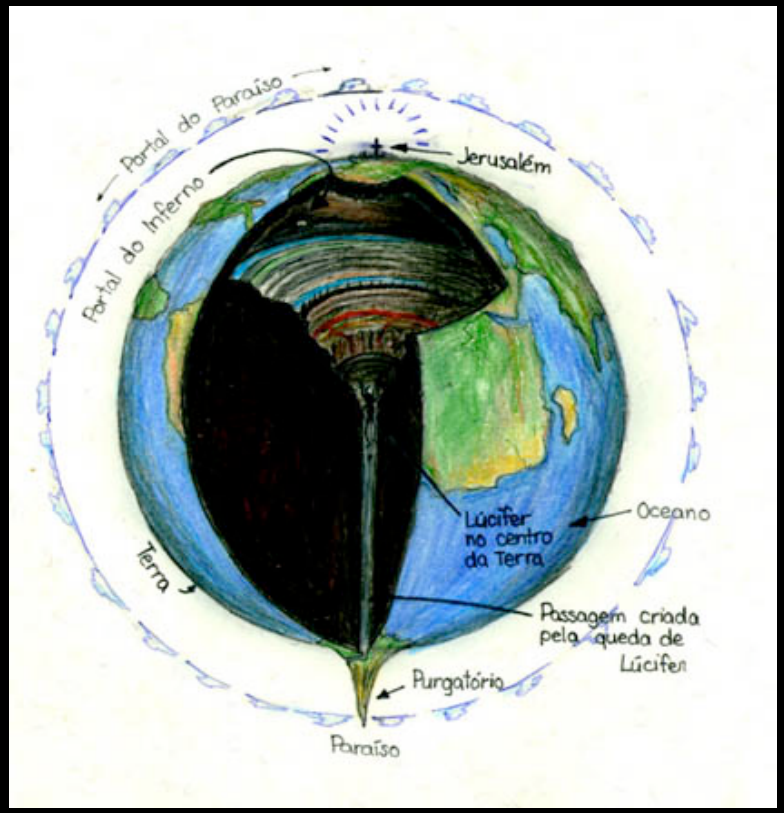
- Uma discussão bem humorada sobre pecados de programação cometidos com a melhor das intenções
  - Não incluimos pecados deliberadamente maus, (ex: aumentar a complexidade do código de propósito para tornar o uso mais difícil)
- Uma discussão mais de idéias e menos de código e soluções específicas
- **Advertência: não leve as metáforas (nem o palestrante) muito a sério**

# Por que Dark Java?

- Programar no escuro (às cegas) é sempre mais difícil
- Paradoxo da programação
  - quer reduzir o caos, a entropia
  - aumenta o caos e exige mais controle para controlá-lo
- Programar no escuro só aumenta o caos e não ajuda a controlar a complexidade

# Arte infernal?

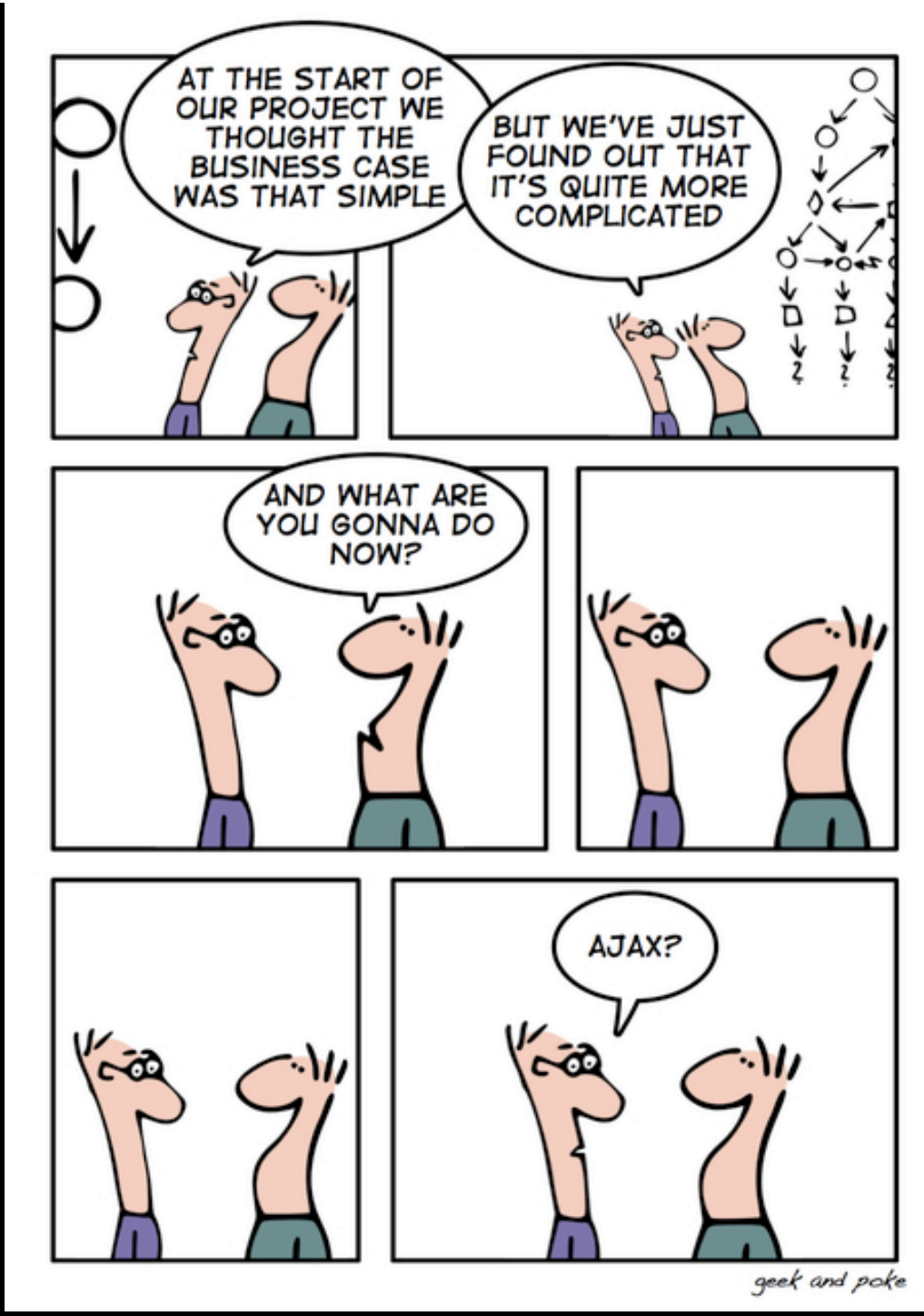
- O inferno, na cultura de vários povos é um mundo inferior, escuro, geralmente cheio de sofrimento e punições nefastas.
- Metáfora para práticas que ajudam a aumentar o caos



# Por que falar disso?

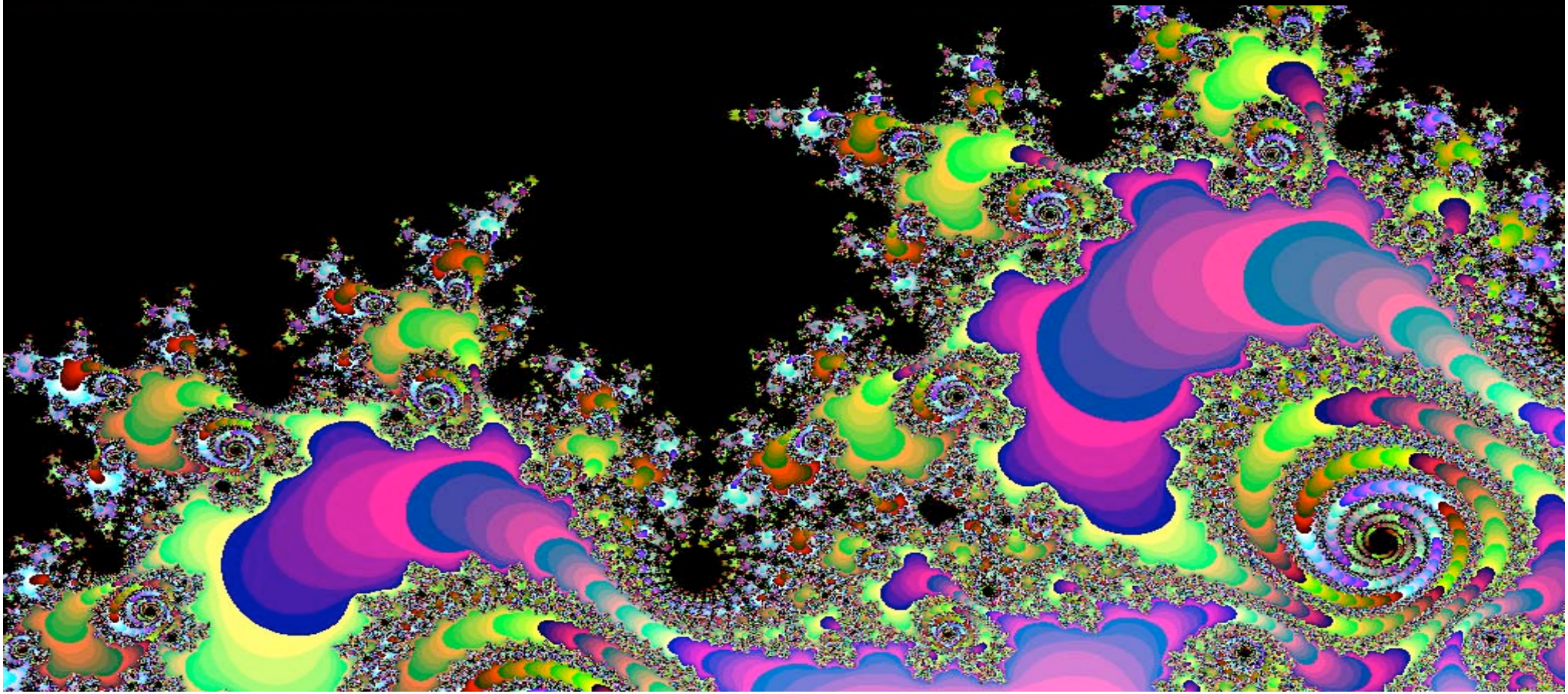
"A complexidade do software é uma propriedade essencial, e não acidental"

F. Brooks



# A complexidade e o caos

- Qual é a profissão mais antiga?



# E a criatividade

- O dilema da **flexibilidade**
  - Ser criativo ou seguir as ordens
  - Conservador ou inovador
- Como organizar um universo que muda o tempo todo?
  - **Pessoas** criativas mudam as máquinas
  - **Máquinas** não são criativas: requerem **ordens** claras
  - **Caos**: máquinas seguem regras que **não vemos** e fazem coisas que **não queremos**
- As linguagens, APIs, frameworks, que sobrevivem são criados com uma finalidade
  - Tornar as coisas mais simples

# Breve história da complexidade

- No princípio havia o **nada**
- Mas o nada não era percebido
- Até que surgiu o **um** para dar sentido ao nada.
- O nada foi descoberto e chamado de **zero**. As instruções eram apenas duas.

# Babel

- Depois chegaram mais **zeros**, e mais uns, e foram agrupados em blocos de quatro, de oito, de dezesseis. Instruções eram **milhares**.
- Vieram letras. **BABEL** tornou-se um número.

# Goto

- Multiplicaram-se as **letras** e organizaram-se em linhas seqüenciais.
- As linhas tornaram-se muitas. Criaram-se formas de **pular** linhas e repetir trechos. Encurtaram-se os programas. Simplificou-se o caos.
- Mas a complexidade mudou-se para o **goto**.

# Procedimentos

- E então as linhas tornaram-se **blocos**, seqüências de palavras, **histórias**.
- Ficaram enormes. Algumas nunca terminavam. Outras bruscamente em **tragédia**. Saber o final, em alguns casos, era uma **incógnita**. As máquinas obedeciam, mas os programadores se perdiam.
- Criaram-se **procedimentos** para dividir a complexidade...

Que ficaram enormes

# Surgiram os objetos

- Tudo ficou “**simples**”....
- Mensagens, encapsulamento...
- Antes dez mil linhas, agora uns cento e poucos objetos....

# Mais objetos, mais objetos

- Novas **plataformas**. Bancos de dados, Rede. Internet. Web.
- Agora podemos fazer **mais** coisas **mais complexas**...
- Cento e pouco, mil, dez mil, cem mil objetos
- **Milhões** de objetos espalhados pelo mundo, abusando da herança e polimorfismo
- Novas tragédias, agora em fluxo não linear e distribuídas

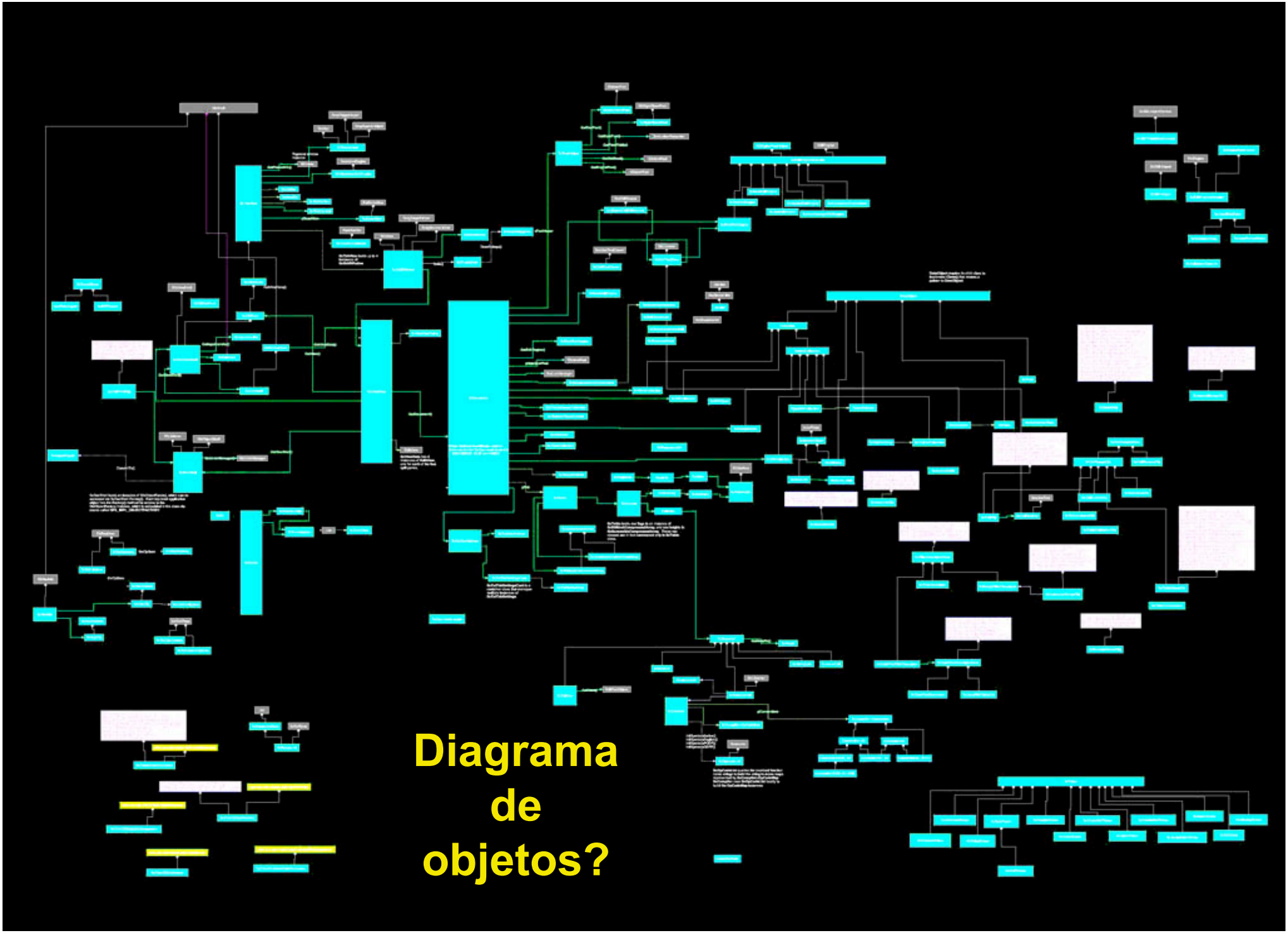


Diagrama
de
objetos?

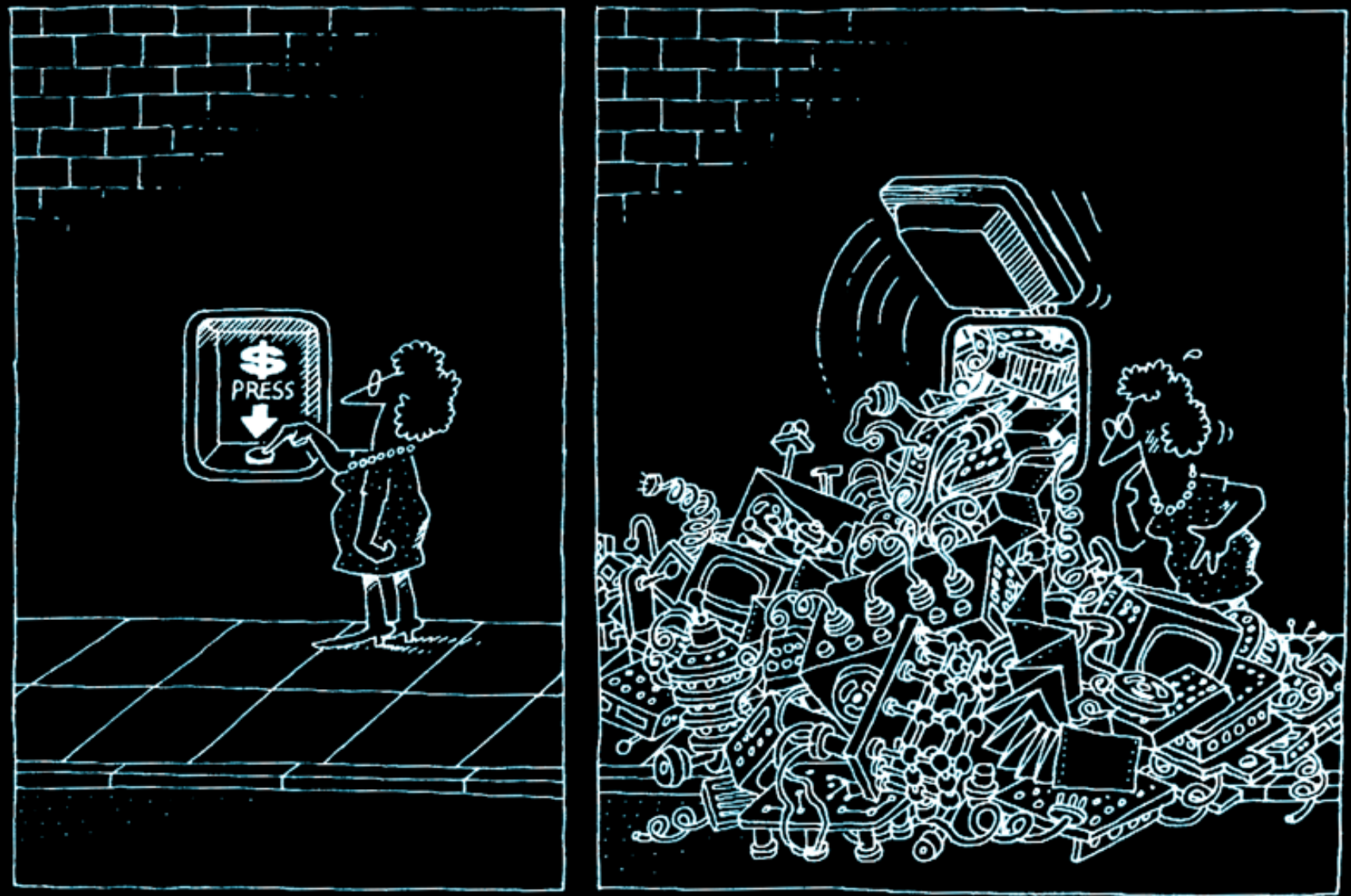
# Então vieram os padrões

- E os **frameworks**
- Agrupa-se aqui e ali. Cumpre-se a promessa dos componentes
- E hoje usamos esses blocos para fazer sistemas cada vez mais complexos.
- Qual será o **próximo problema**?
  - Complexidade dos meta dados?
  - Complexidade dos bindings?

# Lei da conservação da complexidade

- A complexidade **não se destrói**.
- Ela se **esconde** quando é **transferida** de uma solução para outra.

# Ilusão da simplicidade




Fonte: Booch

# E os pecados?

- Na Divina Comédia de Dante Alighieri, o inferno tem **nove** círculos
- Utilizei alguns deles como metáfora para práticas em programação que contribuem para o caos, aumentando a complexidade
- Para evitar o caos e o inferno, evite esses pecados e busque caminhos mais virtuosos

# 1. A luxúria

- A raiz de tudo é a **flexibilidade**
- Se você pode, por que não fazer? A **luxúria** é abusar do que você pode fazer
  - nem tudo deve ser feito em todo lugar e a qualquer hora.
  - Na programação, a **libertinagem** pode ter efeitos devastadores.
- Você pode (mas deve?)
  - Criar classes com qualquer nome
  - Abusar de sobrecarga de nomes de métodos
  - Usar herança sempre que puder
  - Programar sem ligar para o encapsulamento

# Conseqüências

- No inferno de Dante, as almas dos luxuriosos ficam vagando para sempre, levados pelo vento.
- Os programadores que abusam da flexibilidade **vagam para sempre** atrás de bugs que nunca vão encontrar.

# Como evitar a luxúria

- Seguir padrões que funcionam.
  - Padrões de codificação
  - padrões de design
    padrões de organização de processos de software
  - padrões de testes, ...
  - Boas práticas!
- Usar bibliotecas, frameworks, que foram testados, que existem, e que simplificam.
  - Não tente ser criativo reinventando a roda.

# Isto não significa...

- ... abandonar toda criatividade
- Use a criatividade de **forma disciplinada**, planejada, e tenha **mais liberdade**
  - Há mais liberdade em seguir uma disciplina rigorosa que permitirá construir grandes coisas a longo prazo, do que na liberdade dos instintos que quer fazer o que dá na telha a qualquer hora.
- **Keep It Simple!**
  - Faça o **mínimo** possível para alcançar o seu objetivo. Não complique!

# 2. Avareza

- É a busca obsessiva pela economia, em todos os sentidos.
  - Há virtudes na avareza, mas como todos os pecados, o abuso é nocivo.
- Exemplo típico: **busca obsessiva pela performance**
  - Usar uma linguagem, OO, framework, envolvem custo-benefício.
  - Menos complexidade, menos performance.

    (Se você não pode ter perda alguma, pode sempre recorrer à linguagem de máquina)

# Conseqüências

- "We should forget about small efficiencies, say about 97% of the time: premature optimization is **the root of all evil**." (Rev. Donald Knuth)
- A busca pela performance não deve comprometer o design
  - Geralmente vale a pena sacrificar a expectativa de performance pelo **bom design**.
  - É mais fácil consertar problemas localizados, trocar frameworks, fazer ajustes finos.
  - Medir! Medir!

# 3. Orgulho

- Orgulhar-se do que já se aprendeu e **resistir a todas as novidades**. Prefere combatê-las a lidar com quaisquer mudanças.
  - "Não vou usar genéricos"
  - "Eu já fazia tudo muito bem com EJB 2.0"
  - "Essa turma do Scrum um dia vai crescer"
- Escrever seu próprio código quando bibliotecas que fazem o que você quer já existem

Também sintoma de **preguiça** e **inveja**

# 4. Preguiça

- A **preguiça** é um desses pecados que tem um lado bom e um lado ruim.
- **Preguiça boa**: motor que motiva o desenvolvimento da tecnologia em geral.
  - Escravizamos máquinas que trabalham para nós motivados pela preguiça.
- **Preguiça ruim**: comodismo; aumentou depois da invenção do **cut and paste**
  - Antes do **cut & paste** a preguiça motivava os programadores a criarem bibliotecas, APIs

# Conseqüências

- Você acaba trabalhando mais apesar de achar que está trabalhando menos
- Aumenta o caos
  - Inferno para quem for manter o código

# Como evitar

- Reserve tempo para organização
  - (tenha um processo de desenvolvimento)
- Reserve tempo para planejar sua API
  - O ideal é escrever interfaces que alguém **possa entender sem documentação** e que sejam fáceis de usar

# 5. Inveja

- A **inveja** é como a preguiça: tem um lado bom e um lado ruim
  - O **lado bom** motiva a criação de alternativas melhores de soluções que já existem
  - O **lado ruim** incentiva o **orgulho**; motiva a reinvenção da roda com o argumentos fundamentados na **preguiça**
- Há também os que invejam nomes das classes dos outros:
  - com.empresa.Socket?
  - (ou pior: com.empresa.String)

# 6. Fúria

- Surge na hora em que as coisas começam a desmoronar
- Programadores enfurecidos conseguem **justificar** todos os pecados, principalmente a preguiça ruim
  - Não temos **tempo** para fazer testes e documentar
  - Não tivemos **tempo** para avaliar essa API
  - Precisamos otimizar **agora**

# 7. Gula

- Programadores gulosos querem fazer **tudo**; assumir a responsabilidade por **tudo**, no presente e futuro
  - Apesar das boas intenções, é uma fonte de **caos**
- Sintomas
  - Criar APIs **super completas** que resolvem problemas atuais e futuros
  - Criar métodos que **fazem tudo**, abusar da sobrecarga, de construtores, de parâmetros
  - Testar frameworks em testes unitários
  - **Gula preguiçosa**: capturar ou declarar Exception

# 8. Heresia

- Diz uma religião: observar o sábado é bom, mas não a ponto de ser escravo dele
- Heresia em programação é **ser escravo do meio** (e se perder no meio)
  - Você está programando para um **fim**, não para uma **API**, muito menos para o Java
  - Java pode ser o **meio**: conheça o **bem**. Siga padrões mas fuja dos **dogmas**: considere alternativas ou contribua. **Nunca esqueça o fim**.
- Para contribuir para uma linguagem, é preciso conhecer **além** dela

# 9. Violência (tirania)

- Complicar a vida dos outros é um ato de violência causado por criadores de APIs
  - Se você desenvolve interfaces, você é um criador de API: **crie uma API simples**!
  - Tirania: estabelecer regras absurdas que precisam ser seguidas pelos usuários da sua API: complexidade desnecessária
- Ex: exigir tarefas burocráticas
  - Exceções checadas que são desnecessárias (ou pior, ter que tratar **java.lang.Exception**)
  - Passos de construção complexos (devem ser encapsulados em fábricas estáticas)

# Purgatório

- Há como escapar!
  - **Reconhecer os problemas** (abra os olhos, os ouvidos e a boca) e tomar uma atitude.
  - Ter um **processo de desenvolvimento de software** que garanta feedback rápido e eficiente
  - Adotar **padrões**.
  - Estar sempre atento ao controle do caos.
  - Usar ferramentas!

# Ferramentas que ajudam a controlar o caos

- Ant, Maven
- CVS, SVN
- Hudson, Continuum
- JUnit, TestNG, Cobertura, Selenium
- JMeter, JConsole, Profilers
- Checkstyle, PMD, FindBugs
- Sonar
- Trac & Bugzilla

# Para chegar ao paraíso

- Busque obsessivamente o que for mais simples!
  - o método mais curto
  - o nome mais objetivo e fácil de entender
  - o mínimo de parâmetros
  - o mínimo de mutações (objetos imutáveis, campos finais)

  e fuja das otimizações prematuras como o diabo foge da cruz!

# Paraíso

- Na verdade só há **um** pecado...
  - Contribuir para o aumento da complexidade.
- Se você ajudar a reduzir a complexidade do código, dos processos, etc. o seu lugar no paraíso será mais fácil...
- Se é garantido?
  - Não sei... quem foi mesmo que criou o caos?

# Para ir mais longe

- Leia **Effective Java Reloaded**, de Joshua Bloch e que tem várias dicas sobre como diminuir o caos em projetos Java
- Visite www.stelle.com.br e viaje pelo Inferno de Dante :-)

www.argonavis.com.br

Helder da Rocha (helder@argonavis.com.br)

www.stelle.com.br