1 Design Estruturado

Yourdon e Constantine (1979)

"introduzindo uma atividade específica de design formal para descrever completamente, e com antecedência, todas as partes de um sistema e suas inter-relações, não criamos uma nova atividade no ciclo de desenvolvimento de programas."

"o design estruturado é o processo de decidir quais componentes interconectados de qual maneira resolverão algum problema bem especificado."

"O teste final de qualquer sistema é o mercado — ou, mais precisamente, seu uso real pelo usuário final. A utilidade ou facilidade de uso é um critério essencial que costuma receber muita atenção nas etapas iniciais de análise, mas que se perde ao longo do design e da implementação. Parte do problema vem da falta de compreensão de fatores humanos, parte da delegação inadequada de responsabilidades de design. Detalhes internos de programação frequentemente determinam aspectos cruciais da interface com o usuário — por exemplo, campos de entrada limitados a oito caracteres, simplesmente porque isso coincide com o tamanho de uma palavra dupla na máquina. Decisões com profundo impacto na usabilidade podem ser tomadas ad hoc por programadores juniores.

Em resumo, nossos objetivos técnicos globais incluem, em diferentes proporções: eficiência, manutenibilidade, modificabilidade, generalidade, flexibilidade e usabilidade. Esses são os componentes da qualidade objetiva do programa. Para que sejam úteis em um sentido engenheirado, precisamos desenvolver métricas objetivas para cada um deles e demonstrar como decisões de design isoladas os influenciam.

Em termos simples, nosso objetivo primário é criar sistemas de custo mínimo — isto é, baratos de desenvolver, operar, manter e modificar. As prioridades relativas entre custos de desenvolvimento, operação, manutenção e modificação variam de acordo com a organização, o usuário e o projeto, e continuarão a mudar conforme a tecnologia altera a relação entre custos de hardware e software."

2 Genesis II, Creation and recreation with computers

Peterson (1983)

Página 187

“Eu estava envolvido em um jogo de interpretação de papéis não-computadorizado chamado Dungeons and Dragons na época, e também vinha explorando ati-

vamente cavernas — especialmente a Mammoth Cave, no Kentucky. De repente, eu tive uma ideia que combinasse o meu interesse por exploração de cavernas com algo que também fosse um jogo para as crianças, e talvez tivesse alguns elementos de Dungeons and Dragons, que eu vinha jogando. Minha ideia era que fosse um jogo de computador que não intimidasse pessoas que não usavam computadores, e esse foi um dos motivos pelos quais eu fiz com que o jogador dirigisse o jogo por meio de comandos em linguagem natural, em vez de comandos mais padronizados. Meus filhos acharam que foi muito divertido."

Página 193

"...o mundo de Zork exige um uso complexo da linguagem. Uma das características distintivas sobre a série de jogos Adventure era que os comandos estavam na forma de frases em inglês, ainda que fossem apenas frases de duas palavras. Mas em Zork, não basta digitar algo como DROP BOX; o jogador precisa ser mais específico. Por exemplo: PUT ONLY THE OPEN BOX UNDER THE TABLE ('COLOQUE APENAS A CAIXA ABERTA DEBAIXO DA MESA'). Para que uma sentença como essa tenha efeito significativo..."

3 Somewhere Nearby is Colossal Cave: Examining Will Crowther's Original "Adventure" in Code and in Kentucky

Patricia Wilcox (antiga Patricia Crowther) não soube sobre o jogo de Will até depois de encontrar uma cópia da colaboração entre Crowther e Woods. Ela lembra de uma reunião do CRF em "1976 ou 77", na qual muitos membros que supostamente tinham ido explorar cavernas acabaram passando horas jogando o jogo.

Ela descreve a geografia do jogo como "completamente diferente da caverna real. Usava nomes que nós inventamos." Wilcox provavelmente está se referindo às seções adicionadas por Woods.

Outros espeleólogos que conhecem bem o jogo relatam que a geografia do jogo corresponde de forma bastante próxima à geografia da caverna real.

Em uma publicação de 1991 em um fórum de espeleologia, Mel Park escreveu que uma fã de Adventure e novata em exploração de cavernas, Bev Schwartz, conhecia o jogo tão bem que, em sua primeira visita à caverna real:

"Chegávamos a uma encruzilhada, ela perguntava as direções da bússola e começava a nos dizer o que havia por este ou aquele corredor — e sempre acertava!"

4 Debugging Game History: A Critical Lexicon

Lowood e Guins (2024) O Nome do Jogo

O nome desse gênero de jogos vem de um jogo específico e inicial que é altamente influente e, de fato, prototípico: Adventure, também conhecido como Colossal Cave ou Colossal Cave Adventure (Crowther, 1976), um jogo de texto em Fortran criado por Will Crowther e depois expandido em sua forma canônica por Don Woods. Esse jogo, disponível em sua forma mais antiga em 1976, foi um sucesso inicial na ARPANET (Advanced Research Projects Agency Network), a predecessora da Internet. Sem usar gráficos nem exigir uma tela (podia ser jogado em um terminal de impressão), ele delineou a maioria dos aspectos importantes dos jogos de aventura.

Um nome de gênero que deriva de uma obra específica é algo incomum nos videogames e em outras mídias. O gênero de comédia romântica no cinema, por exemplo, poderia ter sido chamado "Trouble in Paradise" (Problemas no Paraíso), em referência ao filme de 1932 que muitos consideram o marco definidor da categoria — mas, claro, não foi o caso. O mesmo vale para o chamado "romance de formação" (ou "romance de amadurecimento"), que não leva o nome de "Tom Jones", romance de Henry Fielding que é seu exemplar mais antigo; em vez disso, é conhecido pelo termo alemão bildungsroman. Embora existam "Sim games" (jogos com o prefixo "Sim"), esse nome designa apenas uma lista de jogos publicados pela mesma empresa, Maxis — mais um exemplo de estratégia de marca do que de formação de gênero.

Pouquíssimos gêneros — do tragicômico ao steampunk — tiram seus nomes de uma única obra que exemplifica o gênero. Um dos poucos casos em que isso ocorre é o do gênero literário especializado "Robinsonada", que indica o tipo de história colonial de sobrevivência e prosperidade exemplificada pelo romance de Daniel Defoe Robinson Crusoé (1719). Ainda assim, embora se espere ser compreendido ao pedir um jogo de aventura em uma loja, é provável que alguém precise se explicar para não ser direcionado à seção de livros de Robinsonadas.

Pode-se argumentar que a situação é semelhante ao dos "jogos de labirinto" (maze games), que no final dos anos 1970 e início dos 1980 eram jogos de ação bidimensionais derivados do jogo Maze (também lembrado como Maze War), criado por Steve Colley na NASA em 1972–1973 e expandido para múltiplos jogadores por Greg Thompson, Dave Lebling e outros no MIT em 1974. Entretanto, a cadeia de influência de Maze até, por exemplo, K.C. Munchkin (Averett, 1981) para o Magnavox Odyssey 2 não é totalmente clara. É verdade que Adventure, como Maze, é um termo genericamente

neutro (sem trocadilho), o que explica, em parte, por que ele foi usado para definir uma categoria quando não temos termos equivalentes para "DOOM", "Tetris" ou "Dance Dance Revolution". Ainda assim, compreender a relação do gênero de aventura com seu ancestral é importante, pois ajuda a explicar por que jogos de aventura parecem ser, hoje, algo do passado.

Aventuras Gráficas, para além de Adventure

Warren Robinett programou o protótipo do jogo de aventura gráfico — também chamado Adventure — para o Atari Video Computer System (VCS), mais tarde conhecido como Atari 2600, lançado em 1978. Embora possa surpreender quem conhece o jogo, Robinett concebeu seu projeto como uma adaptação do Adventure de Crowther e Woods, agora em versão gráfica e controlada por joystick. O principal jogo de aventura para consoles recebeu, portanto, o mesmo nome do jogo de texto Adventure, que era o grande expoente do gênero de aventuras textuais. O repertório de ações disponíveis para o jogador no Atari VCS Adventure não é extenso, mas o jogo mantém muitos dos elementos típicos de uma "missão", encontrados em aventuras mais elaboradas. O jogo apresenta um início (em que o herói, munido apenas de sua astúcia, parte em sua jornada), um meio (envolvendo exploração e combate) e um fim (quando o cálice é levado de volta triunfalmente ao castelo dourado) — embora essa estrutura narrativa segmentada não seja exclusiva do gênero de aventuras.

Embora Adventure de Crowther e Woods carecesse de gráficos, ele já apresentava todas as principais características do gênero de jogos de aventura.

Em um jogo de aventura, o jogador explora um mundo simulado e fictício para compreendê-lo e expressar essa compreensão por meio de ações dentro do jogo. Por "simulado e fictício", entende-se que o mundo é interativo, operável e responsivo às escolhas e comentários do jogador, e que o jogador é convidado a abordá-lo com a mesma imaginação ficcional usada ao ler um romance.

Quase sempre há personagens em jogos de aventura e, mesmo quando estão ausentes do mundo simulado — como no best-seller Myst (Cyan Interactive, 1993) — o ambiente oferece importantes vestígios de acontecimentos passados. Embora alguns jogos de aventura sejam restritos a uma única linha de progressão ou de missões, outros são abertos, permitindo que diferentes jogadores percorram o enredo de maneiras distintas. Em qualquer caso, a narratividade costuma ser alta, com histórias de fundo, quebra-cabeças e ações em jogo que estão intimamente ligadas à ficção do mundo representado.

Existem muitas versões do jogo de texto Adventure, assim como muitos outros jogos fortemente inspirados por ele, como Zork (1977) — um minicomputador desenvolvido por Timothy A. Anderson, Mark Blanc, Bruce Daniels

e David Lebling no MIT, depois publicado pela Infocom como uma trilogia de aventuras. O Zork é analisado em detalhes no livro Twisty Little Passages: An Approach to Interactive Fiction (Montfort, 2003). Adventure também foi o tema da primeira dissertação de mestrado em estudos de jogos (Buckles, 1985). A pesquisa mais notável sobre o jogo é de Dennis Jerz (2007), que visitou a caverna de Kentucky que inspirou Crowther, recuperou o código-fonte original antes de Woods modificá-lo e demonstrou que os estudos de jogos podem incluir tanto o contexto exploratório e experiencial do designer quanto a análise do jogo no nível do código.

5 Jeremy Norman's HistoryofInformation.com

url: https://www.historyofinformation.com/detail.php?id=2020

Adventure was originally called ADVENT because a filename could only be six characters long in its operating system. The game was renamed Colossal Cave AdventureOffsite Link, as it was based on part of the Mammoth CaveOffsite Link system in Kentucky.

6 A História de Zork

url: https://samizdat.co/shelf/documents/2004/05.27-historyOfZork/historyOfZork.pdf

por Tim Anderson e Stu Galley Primeira Parte da Série

No começo, lá nos anos 1960, a DEC (Digital Equipment Corporation) criou o PDP-10, um computador de médio porte. O "10", como era chamado, tornou-se popular em muitos centros de pesquisa, e muito software foi desenvolvido para ele — alguns desses programas ainda são tecnologicamente mais avançados do que sistemas modernos.

No Laboratório de Inteligência Artificial do MIT, foi desenvolvido um sistema operacional chamado ITS (Incompatible Time-Sharing System) para o PDP-10. O ITS foi projetado para tornar o desenvolvimento de software fácil. Os criadores do sistema presumiram que ele seria usado por um pequeno grupo de pessoas experientes e amigáveis, então não incluíram nenhum recurso de segurança.

Por volta de 1970, foi inventada a ARPAnet, que permitia que pesquisadores em todo o país (e até no mundo) se comunicassem e usassem os computadores uns dos outros. Naqueles tempos idílicos, o acesso era irrestrito: bastava estar conectado à rede ou conhecer o número de telefone certo. Jovens hackers logo descobriram que isso era um playground maravilhoso. Descobriram também que havia alguns computadores no MIT com coisas

bem interessantes e sem segurança — qualquer um que conseguisse se conectar podia entrar.

Também por volta de 1970, surgiu uma linguagem chamada MUDDLE (posteriormente renomeada para MDL), desenvolvida como sucessora do LISP. Ela nunca chegou a substituí-lo, mas criou uma comunidade fiel, especialmente no Project MAC (atual Laboratório de Ciência da Computação do MIT), e particularmente no Dynamic Modelling Group (DM) — mais tarde chamado Programming Technology Division.

O Dynamic Modelling Group foi responsável por alguns jogos notáveis. O primeiro foi um jogo gráfico multiplayer chamado Maze, no qual os jogadores vagavam por um labirinto tentando atirar uns nos outros. A tela de cada usuário mostrava a visão do labirinto de seu "alter ego" digital, atualizada em tempo real. Dave Lebling foi um dos principais responsáveis (ou culpados?) pela criação do jogo.

O próximo jogo de destaque foi Trivia (quem disse que laboratórios de pesquisa não estão à frente do seu tempo?), uma competição contínua de perguntas e respostas para mentes verdadeiramente obsessivas. Diferente de Maze, Trivia podia ser jogado por usuários de toda a ARPAnet e alcançou grande popularidade. Marc Blank escreveu a segunda versão, e eu (Tim Anderson) mantive e hackeei o código — na verdade, era um teste legítimo de um sistema de banco de dados usado em um projeto de pesquisa.

No início de 1977, Adventure varreu a ARPAnet. Will Crowther foi o autor original, mas Don Woods expandiu o jogo e o lançou na rede, surpreendendo a todos. Quando Adventure chegou ao MIT, a reação foi típica: depois que todos gastaram muito tempo resolvendo o jogo (estima-se que Adventure atrasou a indústria de computadores em duas semanas), os verdadeiros fanáticos começaram a pensar em como poderiam fazer algo melhor.

Adventure era escrito em FORTRAN, o que o tornava limitado. Ele aceitava apenas comandos de duas palavras, era difícil de modificar e tinha problemas que poderiam ser melhorados. (Eu estava presente quando Bruce Daniels, do DM, conseguiu descobrir como obter o último ponto em Adventure analisando o jogo com um depurador em linguagem de máquina — não havia outro jeito.)

Por volta de maio de 1977, Adventure já tinha sido solucionado, e vários membros do DM começaram a procurar novas diversões. Marc Blank aproveitava uma pausa na faculdade de medicina; eu havia acabado de concluir meu mestrado; Bruce Daniels estava entediado com seu tema de doutorado; e Dave Lebling estava farto de código Morse. Dave escreveu, em MDL, um analisador de comandos quase tão inteligente quanto o de Adventure; Marc e eu, que costumávamos passar noites programando, usamos isso para criar um protótipo de jogo com quatro salas. Esse jogo se perdeu há muito tempo.

Havia uma banda, uma "sala do amendoim" (onde a banda tocava "Hail to the Chief") e uma "câmara cheia de prazos". Dave testou o jogo, achou horrível e foi tirar férias de duas semanas.

Marc, Bruce e eu então resolvemos fazer um jogo de verdade. Começamos desenhando mapas, inventando problemas e discutindo muito sobre como as coisas deveriam funcionar. Bruce ainda sonhava em se formar e preferiu se concentrar no design, então Marc e eu passamos o resto das férias de Dave no terminal, implementando a primeira versão de Zork.

O nome "Zork" nunca foi oficial. Era apenas uma palavra sem sentido que usávamos — geralmente como verbo, tipo "zork the fweep" — e pode ter vindo de outra gíria, "zorch", que significava destruição total. Costumávamos dar o nome "zork" a um programa enquanto ele ainda estava em desenvolvimento.

Quando Dave voltou, já havia um jogo funcional (mais ou menos). Não era tão grande quanto Adventure — provavelmente menos da metade do tamanho da versão final — mas já tinha o ladrão, o ciclope, o troll, o reservatório e a represa, a casa, parte da floresta, a geleira, o labirinto e várias outras áreas. Os enigmas ainda não eram muito interessantes; demorou um tempo até aprendermos a criar bons desafios, e os primeiros analisadores não suportavam soluções complexas.

O que fizemos certo estava na infraestrutura: havia uma teoria bem definida (e facilmente modificável) para as interações entre objetos, verbos e salas. Era fácil adicionar novos analisadores — algo que acontecia com frequência, já que todo mundo queria tentar escrever o seu. (Marc acabou ficando obcecado e escreveu os últimos 40 ou 50 sozinho.) Também era fácil adicionar novas salas, objetos e criaturas — mas ainda não novas ideias abstratas.

Zork, assim como Adventure, sobreviveu porque foi jogado por pessoas fora da comunidade que o criou. Adventure era portável por ser escrito em FORTRAN, enquanto Zork era em MDL, que só rodava em alguns PDP-10. Mesmo assim, ele encontrou um público: os "net randoms" — usuários curiosos que vagavam pelos sistemas do MIT, já que não havia segurança nenhuma.

O grupo DM tinha se tornado popular por causa de Trivia, e quando Trivia morreu, os usuários ficaram à espera do próximo jogo. Os primeiros jogadores de Zork variavam de John McCarthy (criador do LISP) até crianças de 12 anos da Virgínia.

Ninguém anunciou oficialmente Zork. As pessoas simplesmente entravam no sistema do DM, viam alguém executando um programa chamado "Zork" e ficavam curiosas. Elas espionavam o terminal da pessoa, percebiam que era um jogo estilo Adventure e logo descobriam como iniciá-lo. Por muito tempo, o comando mágico era :MARC;ZORK. Mesmo quem nunca tinha

ouvido falar de ITS, DM ou PDP-10 acabava ouvindo que, se conseguisse acessar o "host 70" na ARPAnet, entrar e digitar a palavra mágica, poderia jogar um jogo de aventura.

Em junho de 1977, Zork já era muito mais primitivo que o Zork I, mas tinha o mesmo espírito. A família Flathead já aparecia, com Lord Dimwit Flathead, o Exagerado, governante do Grande Império Subterrâneo, e a moeda oficial era o zorkmid. Bruce foi responsável pelo texto pomposo onde esses elementos foram apresentados.

Muitos detalhes do mundo subterrâneo eram brincadeiras internas (ou francamente bobos), mas nem tudo era absurdo. No início, se o jogador andasse em uma área escura, caía em um poço sem fundo — até que os usuários começaram a apontar que um poço sem fundo num sótão deveria ser visível do térreo. Foi então que Dave criou as "grues", criaturas que devoravam quem andasse no escuro.

Desde quase o começo, havia no jogo um jornal fictício chamado "US News e Dungeon Report", que anunciava mudanças no jogo. Todas as mudanças eram creditadas a algum grupo de "desenvolvedores", mesmo que não fossem os verdadeiros responsáveis. Um número famoso dizia que Bruce passara semanas enchendo todos os poços sem fundo, o que levou hordas de grues a vagar pelo mapa.

A primeira grande expansão do jogo, feita em junho de 1977, foi a seção do rio, criada por Marc. Ela sobrevive praticamente inalterada em Zork I e mostra bem as dificuldades de criar uma simulação coerente. Havia problemas de consistência — partes do rio eram iluminadas, outras escuras —, mas o maior desafio foi o conceito de veículos. Antes, o jogo tinha apenas salas, objetos e o jogador. Marc introduziu objetos que funcionavam como salas móveis (como um barco). Isso exigiu mudanças delicadas nas interações entre verbos, objetos e salas (por exemplo, o que "andar" significava quando o jogador estava em um barco?).

Os jogadores logo começaram a tentar usar o barco em todo lugar. O código original não permitia isso, mas nada impedia o jogador de levar o barco desinflado até o reservatório e tentar navegar. Eventualmente, o barco foi permitido ali, mas esse tipo de problema persistia: qualquer mudança no mundo simulado afetava todo o resto do mundo simulado.

Embora Zork tivesse apenas um mês de idade, já conseguia surpreender seus criadores. Por causa de um detalhe na implementação, o barco funcionava como uma "bolsa de contenção infinita" — os jogadores podiam colocar qualquer coisa dentro dele, mesmo ultrapassando o limite de peso permitido. O barco era, na verdade, dois objetos diferentes: o "barco inflado" (que continha os itens) e o "barco desinflado" (que o jogador carregava). Nós nem sabíamos disso — alguém acabou relatando como um bug. Até onde sei, o

bug ainda está lá.

7 The Untold Story of the Women Who Made the Internet

Evans (2018)

Os espeleólogos acreditavam que o Flint Ridge se encontrava com o Mammoth Cave além de um bloqueio de pedras areníticas no ponto de levantamento Q-87, um desvio remoto a quilômetros da superfície — mas mover aquelas pedras com tubos de metal era um trabalho extenuante. Uma das expedições tentou uma rota alternativa, passando por uma fenda vertical chamada "Tight Spot" ("Lugar Apertado").

O humor entre espeleólogos tem um toque niilista: o Tight Spot era uma fenda tão estreita e escura que apenas uma pessoa do grupo teve coragem de entrar. Ela era uma programadora magra, com pouco mais de 50 quilos, chamada Patricia Crowther.

Pat se espremeu pelo Tight Spot e saiu do outro lado, sobre um banco de lama. À luz fria do carbureto, ela avistou o cartão de visita de um explorador anterior: as iniciais "P.H." gravadas na parede. De volta à superfície, o grupo manteve a descoberta em segredo. Qualquer pessoa familiarizada com a área saberia da lenda de Pete Hanson, um explorador que havia percorrido o Mammoth antes da Guerra Civil. Aquelas tinham de ser as iniciais dele, o que só podia significar uma coisa: Flint Ridge e Mammoth estavam conectadas, formando uma única caverna contínua de mais de 540 quilômetros — uma descoberta monumental que ficaria conhecida como o Everest da espeleologia.

Pat retornou dez dias depois para explorar a junção. "Ah, Pat — desta vez você lidera a expedição", disseram os outros espeleólogos. Logo após o Tight Spot, eles caminharam por água lamacenta até o peito, com apenas trinta centímetros de ar entre o rio subterrâneo e o teto da caverna. Encharcados e cobertos de lama "como cobertura de chocolate", lutavam para manter as lanternas secas, enquanto lagostins cegos se arrastavam ao redor de suas cinturas.

Quando o túnel se abriu, revelou-se um salão amplo — e ali, vislumbraram o corrimão de metal de uma trilha turística, no coração do Mammoth Cave. A ligação estava completa. Momentos antes, eles haviam estado mais longe sob a terra do que qualquer outro grupo na história; agora, chorando e tropeçando uns nos outros dentro d'água, estavam a poucos passos de um banheiro público.

Voltando ao acampamento na carroceria de uma caminhonete de guarda

florestal, olharam para o céu de verão, repleto de estrelas. Deitados "na caçamba aberta, com as copas das árvores passando por cima e desaparecendo na escuridão", contemplaram o feito em silêncio. A longa viagem reforçou a dimensão da façanha: será que realmente haviam percorrido sete milhas subterrâneas? O trajeto final, passando pelo Tight Spot e pelo que mais tarde seria chamado de Hanson's Lost River, completou uma linha esquecida no mapa desenhado à mão por Stephen Bishop em 1839. Depois de hambúrgueres e champanhe ao amanhecer, dormiram.

"É uma sensação incrível", escreveu Patricia em seu diário, "fazer parte do primeiro grupo a entrar no Mammoth Cave vindo de Flint Ridge. É como ter um filho. Você precisa se lembrar o tempo todo de que isso é realmente real — essa nova criatura que você trouxe ao mundo e que não existia ontem. Tudo o mais parece novo também. Depois que acordamos na quinta-feira, ouço um disco do Gordon Lightfoot. A música é tão bonita que me faz chorar."

A nova criatura que Patricia sentia ter trazido ao mundo sempre estivera ali, adormecida na escuridão do tempo geológico. O que ela de fato deu à luz naquele dia não foi a caverna, mas o mapa — não o objeto, mas sua descrição. Ao se enfiar pelo Tight Spot e levar sua luz à escuridão, ela transformou um lugar físico em um espaço simbólico cartesiano. Ou pelo menos é assim que ela talvez o tenha visto, sendo a mapista do grupo.

De volta a Massachusetts, Pat e seu marido Will administravam uma "fábrica de mapas", compilando os dados cartográficos de cada expedição da Cave Research Foundation (CRF). Ambos programadores, trouxeram sofisticação técnica incomum à confecção de mapas. Como Pat descreveu, o casal digitava os dados brutos de levantamento — tirados de "caderninhos cheios de lama" — em um terminal Teletype na sala de estar, conectado a um computador central PDP-1 no local de trabalho de Will. A partir desses dados, geravam "comandos de plotagem em enormes rolos de fita perfurada", usando um programa que Will escreveu (Pat contribuiu com uma sub-rotina para adicionar letras e números ao mapa final), e depois usavam uma plotadora Calcomp reaproveitada, conectada a um Honeywell 316 — máquina que mais tarde se tornaria um IMP da ARPANET.

Os mapas dos Crowther eram traçados lineares simplificados, mas representaram um dos primeiros esforços de informatização de cavernas, um salto técnico viabilizado pelo acesso que tinham a equipamentos avançados: o PDP-1 e o Honeywell 316, ambos muito além do alcance de consumidores comuns.

Will trabalhava na empresa Bolt, Beranek and Newman (BBN), em Massachusetts, especializada em pesquisa avançada. Em 1969, a BBN foi contratada pelo governo dos EUA para ajudar a construir a ARPANET — a rede

militar e acadêmica de comutação de pacotes que daria origem à Internet moderna. Alguns anos depois de usarem o sistema para desenhar mapas de cavernas, o Honeywell 316 foi reconfigurado e reforçado para se tornar um Interface Message Processor (IMP) — o que hoje chamamos de roteador. Esses roteadores formavam uma sub-rede de computadores menores dentro da ARPANET, responsáveis por encaminhar dados e traduzir protocolos entre os nós principais — um componente vital da Internet, então e agora.

Will era um dos programadores mais talentosos da BBN, e seu código enxuto e preciso refletia sua natureza meticulosa. Um alpinista experiente, ele ensinou Patricia a escalar as falésias verticais das montanhas Shawangunk, em Nova York, e era conhecido por ficar pendurado nas pontas dos dedos na moldura da porta do escritório enquanto pensava. Will também era espeleólogo, e o casal passava todas as férias explorando cavernas.

"Sinto frio quando ele não está comigo", escreveu Patricia em um de seus diários. "Há uma corrente de ar forte aqui; a caverna está respirando."

Will não participou da expedição final da conexão. Ele estivera com Patricia em levantamentos anteriores, levando seu corpo ao limite no subterrâneo. Mas a viagem final ocorreu no início de setembro, quando suas filhas — Sandy (8 anos) e Laura (6 anos) — estavam voltando às aulas. Alguém precisava ficar em casa, comprar livros e roupas, levá-las ao dentista e matriculá-las.

Will sabia o quanto a expedição significava para Patricia. Afinal, foi ela quem encontrara a passagem — o que os espeleólogos chamam de "going cave" — e ansiava por completá-la. Ele disse a ela para ir; ele cuidaria das meninas.

Quando Pat voltou, profundamente emocionada pela experiência, Will a esperava. Eles ficaram acordados até tarde, abraçados, conversando sobre a conexão. Quando Will adormeceu, Pat foi até o terminal Teletype na sala e, o mais silenciosamente possível, digitou as coordenadas do levantamento que haviam feito em Kentucky. Ela executou um programa de coordenadas, e os dados saíram em uma longa fita de papel perfurada.

Na manhã seguinte, Pat e Will levaram a fita ao escritório dele, e ela observou o computador da BBN traçar no gráfico a ligação subterrânea que ela havia completado entre dois lugares vastos e solitários.

"Agora posso dormir", escreveu.

Espeleologia é implacável. Até o final da década de 1960, quem entrava no Mammoth passava pelo caixão de vidro de Floyd Collins, um espeleólogo que morreu preso sob uma pedra. Os espeleólogos são engolidos pela terra, cada movimento limitado por paredes e tetos de rocha. Comem muito pouco — barras de chocolate e carne enlatada — e levam todo o próprio lixo de volta à superfície. Perdem a noção do tempo. Ao emergir, podem se surpreender

ao ver a lua.

Como escreveram os amigos dos Crowther, Roger Brucker e Richard Watson, em The Longest Cave, seu relato da expedição da conexão:

"A rota nunca está à vista, exceto naquilo que você pode imaginar. Nada se desenrola. Não há progresso — há apenas uma sucessão de lugares que mudam à medida que você avança."

Tornar a rota visível é o objetivo central da espeleologia séria. A Cave Research Foundation tinha um lema:

"Nenhuma exploração sem levantamento."

Um mapa é a única forma de ver uma caverna em sua totalidade — e mapear é o equivalente, para espeleólogos, a alcançar o cume de uma montanha. É também uma questão de sobrevivência. Para se manterem seguros, espeleólogos mapeiam à medida que avançam, trabalhando de forma racional e sistemática para localizar passagens conhecidas. Não é de admirar que o hobby atraia programadores. O código, afinal, é um país habitado por meticulosos. Como os programadores, os espeleólogos podem trabalhar em grupo — mas sempre enfrentam seus desafios sozinhos.

Pouco depois da viagem da conexão, o casamento de Patricia e Will começou a se deteriorar. Eles se divorciaram em 1976, após uma separação que deixou Will "fragmentado de várias maneiras". Explorar cavernas sem Patricia, entre amigos em comum da Cave Research Foundation, havia se tornado constrangedor.

Sozinho, cercado pelos mapas que haviam feito juntos — incluindo um levantamento detalhado da seção Bedquilt do Mammoth, realizado no verão de 1974 —, Will buscou consolo em longas campanhas de Dungeons Dragons e noites de programação. Quando Sandy e Laura o visitavam, geralmente o encontravam trabalhando em uma longa e elegante sequência de código FORTRAN. Ele lhes dizia que era um jogo de computador, e que, quando terminasse, seria delas para jogar.

O romancista Richard Powers escreveu certa vez que

"o software é a vitória final da descrição sobre a coisa."

A precisão minuciosa com que o software descreve a realidade se aproxima — e às vezes toca — uma ordem mais profunda. Talvez por isso Will Crowther tenha sentido a necessidade de criar um último mapa. Dessa vez, não a partir dos "caderninhos lamacentos" da esposa, mas de suas próprias memórias.

Traduzidas em setecentas linhas de código FORTRAN, elas se tornaram Colossal Cave Adventure, um dos primeiros jogos de computador, baseado fielmente nas seções do Mammoth Cave que ele havia explorado e mapeado com Patricia — em um computador que, ironicamente, formaria a espinha dorsal da Internet.

Colossal Cave Adventure — ou simplesmente Adventure — não se parece com um jogo moderno. Não há imagens nem animações, nem joysticks ou controles. Em vez disso, blocos de texto descrevem as seções da caverna, em segunda pessoa, assim:

Você está em uma esplêndida câmara de nove metros de altura. As paredes são rios congelados de pedra laranja. Um cânion irregular e uma boa passagem saem pelos lados leste e oeste da câmara. Um pequeno pássaro alegre está aqui, cantando.

Para interagir com a caverna, o jogador digita comandos imperativos curtos, como GO WEST (ir para o oeste) ou GET BIRD (pegar o pássaro), que acionam novas descrições.

Os enigmas do Adventure são uma dança infinita de inventário mágico: para passar pela cobra enrolada no Salão do Rei da Montanha, é preciso libertar o pássaro da gaiola — mas não é possível GET BIRD se o jogador estiver com a vara preta, pois o pássaro tem medo da vara; e, por sua vez, a ponte de cristal não aparece sem um aceno da vara; e enquanto isso, o jogador está preso em um labirinto de passagens tortuosas, todas diferentes — ou pior, todas iguais.

Isso era familiar aos colegas de Will, que jogavam Dungeons Dragons após o expediente. Em DD, um jogo sem objetivo de vitória, um "Mestre" descreve as cenas em detalhes e desafia os jogadores a decidir suas ações.

Mas Will escreveu o jogo para suas filhas pequenas. Depois do divórcio, Sandy e Laura sabiam que, ao visitar o pai, jogariam algo novo no computador. Segundo um pesquisador que entrevistou membros da Cave Research Foundation:

"Outro espeleólogo, que estivera com os Crowther em uma expedição no verão de 1975, relatou que bastou um olhar para 'Adventure' para reconhecer imediatamente: era um exercício catártico, uma tentativa de Will de preservar uma experiência perdida."

8 Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach

Following the approach proposed in [13], in the first stage of the diagram recovery process, the Java language is simplified into an abstract language where all features related to the object flow are maintained while the other syntactic details are dropped (Fig. 3). The choice of this program representation is motivated by the computational complexity and the "nature" of the object oriented programs whose code is typically structured so as to impose

more constraints on the data flows than on the control flows. For example, the sequence of method invocations may change when moving from an application which uses a class to another one, while the possible ways to copy and propagate object references remains more stable

9 Reverse Engineering of Object Oriented Code

Tonella e Potrich (2007) 2.1 Abstract Language The static analysis conducted on Java programs to reverse engineer design diagrams from the code is data flow sensitive, but control flow insensitive. This means that programs with different control flows and the same data flows are 22 2 The Object Flow Graph associated with the same analysis results. Data flow sensitivity and control flow insensitivity are achieved by defining the analyses with reference to a program representation called the Object Flow Graph (OFG). A consequence of the control flow insensitivity is that the construction of the OFG can be described with reference to a simplified, abstract version of the Java language. All Java instructions that refer to data flows are properly represented in the abstract language, while instructions that do not affect the data flows at all are safely ignored. Thus, all control flow statements (conditionals, loops, etc.) are not part of the simplified language. Moreover, in the abstract language name resolution is also simplified. All identifiers are given fully scoped name, being preceded by a dot separated list of enclosing packages, classes and methods. In this way, no name conflict can ever occur. The choice of a data flow sensitive/control flow insensitive program representation is motivated by two main reasons: computational complexity and the "nature" of the Object Oriented programs. As discussed in Section 2.4, the theoretical computational complexity and the practical performances of control flow insensitive algorithms are substantially superior to those of the control flow sensitive counterparts. Moreover, the Object Oriented code is typically structured so as to impose more constraints on the data flows than on the control flows. For example, the sequence of method invocations may change when moving from an application which uses a class to another one, while the possible ways to copy and propagate object references remains more stable. Thus, for Object Oriented code, where the actual method invocation sequence is unknown, it makes sense to adopt control flow insensitive/data flow sensitive analysis algorithms, which preserve the way object references are handled. Fig. 2.1 shows the abstract syntax of the simplified Java language. A Java program P consists of zero or more occurrences of declarations (D), followed by zero or more statements (S). The actual ordering of the declarations and of the statements is irrelevant, due to

the control flow insensitivity. The nesting structure of packages, classes and methods is completely flattened. For example, statements belonging to different methods are not divided into separate groups. However, the full scope is explicitly retained in the names (see below). Consequently, a fine grain identification of the data elements is possible, while this is not the case for the control elements (control flow insensitivity). Transformation of a given Java program into its abstract language representation is an easy task, that can be fully automated. Program transformation tools can be employed to achieve this aim.

10 Capítulo 17 CASOS DE USO - UML

Nenhum sistema existe isoladamente. Todo sistema interessante interage com atores humanos ou autômatos que utilizam esse sistema para algum propósito e esses atores esperam que o sistema se comporte de acordo com as maneiras previstas. Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de seqüências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator. Os casos de usos podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para os desenvolvedores chegarem a uma compreensão comum com os usuários finais do sistema e com os especialistas do domínio. Além disso, os casos de uso servem para ajudar a validar a arquitetura e para verificar o sistema à medida que ele evolui durante seu desenvolvimento. À proporção que você implementa o seu sistema, esses casos de uso são realizados por colaborações cujos elementos trabalham em conjunto para a execução de cada caso de uso. Casos de uso bem-estruturados denotam somente o comportamento essencial do sistema ou subsistema e não são amplamente gerais, nem muito específicos.

Um caso de uso executa alguma quantidade tangível de trabalho. Sob a perspectiva de um determinado ator, um caso de uso realiza algo que é de valor para um ator, como o cálculo de um resultado, a geração de um novo objeto ou a modificação do estado de outro objeto. Por exemplo, na modelagem de um banco, o processamento de um empréstimo resulta na entrega de um empréstimo aprovado, manifestada como uma pilha de dinheiro entregue nas mãos do cliente.

Você poderá aplicar os casos de uso a todo o seu sistema. Também pode aplicá-los a uma parte do sistema, incluindo subsistemas e até interfaces e classes individuais. Em cada situação, os casos de uso não apenas represen-

tam o comportamento desejado desses elementos, mas também podem ser utilizados como a base de casos de teste para esses elementos, à medida que evoluem durante o desenvolvimento. Casos de uso aplicados aos subsistemas são excelentes fontes de testes de regressão; casos de uso aplicados a todo o sistema são excelentes fontes de testes de sistema e de integração. A UML fornece a representação gráfica de um caso de uso e de um ator, conforme mostra a Figura 17.1. Essa notação permite visualizar um caso de uso em separado de sua realização e no contexto com outros casos de uso.

Referências

YOURDON, Edward; CONSTANTINE, Larry L. **Structured design: fundamentals of a discipline of computer program and systems design**. [*S. l.*]: Prentice-Hall, Inc., 1979.

PETERSON, Dale. **Genesis II, Creation and recreation with computers**. [*S. l.*]: Reston Publishing Company, 1983.

LOWOOD, H.; GUINS, R. **Debugging Game History: A Critical Lexicon**. [*S. l.*]: MIT Press, 2024. (Game Histories). ISBN 9780262551106. Disponível em: `https://books.google.com.br/books?id=RWbqEAAAQBAJ`.

EVANS, Claire L. **Broad Band**: The Untold Story of the Women Who Made the Internet. New York: Portfolio, 2018. ISBN 9780735211759.

TONELLA, P.; POTRICH, A. **Reverse Engineering of Object Oriented Code**. [*S. l.*]: Springer New York, 2007. (Monographs in Computer Science). ISBN 9780387238036. Disponível em: `https://books.google.com.br/books?id=EUZ-_poEq_gC`.