O'REILLY®

Inżynieria chaosu

Odporność systemów w praktyce

Casey Rosenthal
Nora Jones

Helion

Tytuł oryginału: Chaos Engineering: System Resiliency in Practice

Tłumaczenie: Katarzyna Bogusławska

ISBN: 978-83-283-8285-5

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/inzcha
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 3.

Przegląd zasad

W początkach inżynierii chaosu w Netflixie nie było do końca jasne, czym ta dyscyplina jest. Przewijało się wiele sloganów, takich jak „wyrywanie kabli", „psucie wszystkiego", „testowanie na produkcji", mnożyły się błędne koncepcje wzmacniania trwałości usług, ale przykładów rzeczywistych narzędzi było bardzo niewiele. Zespół Chaosu powołano po to, by stworzył mającą znaczenie dyscyplinę, taką, która proaktywnie poprawia odporność dzięki właściwym narzędziom. Spędziliśmy całe miesiące, studiując inżynierię odporności i inne kierunki, by sprecyzować definicję i dać wzór tego, jak inni mogą uczestniczyć w inżynierii chaosu. Definicję tę opublikowano online (*https://principlesofchaos.org/*) jako swego rodzaju manifest, który nazwano „Zasadami" (By dowiedzieć się, jak powstała inżynieria chaosu, przeczytaj „Wprowadzenie. Narodziny chaosu").

Podobnie jak każdą nową koncepcję, inżynierię chaosu często rozumie się opacznie. Kolejne sekcje wskazują, czym ta dyscyplina *jest*, a czym *nie jest*. Złoty standard praktyk zawarliśmy w podrozdziale „Zaawansowane zasady". Przyjrzymy się też temu, jakie czynniki mogą wpłynąć na zmianę zasad w przyszłości.

Czym jest inżynieria chaosu

„Zasady" definiują dyscyplinę tak, byśmy wiedzieli, kiedy się nią zajmujemy, jak to robić i jak robić to dobrze. Powszechną dzisiaj definicją jest „ułatwianie eksperymentów mające na celu odkrycie systemowych słabości". Strona internetowa „Zasad" wskazuje kroki takich eksperymentów:

1. Zacznij od zdefiniowania „stanu stabilnego" jako mierzalnego rezultatu systemu, który wskazuje jego normalne zachowanie.
2. Stwórz hipotezę na temat tego, czy stan stabilny utrzyma się zarówno w grupie doświadczalnej, jak i kontrolnej.
3. Wprowadź zmienne, które odpowiadają realnym zdarzeniom, takim jak awarie serwerów, błędy dysków twardych, połączeń sieciowych itd.
4. Postaraj się obalić hipotezę, przyglądając się stanowi stabilnemu w grupie kontrolnej i doświadczalnej.

Taki eksperyment stanowi podstawową zasadę inżynierii chaosu. Celowo istnieje wiele różnych sposobów implementacji takich działań.

Eksperymenty a testy

Jednym z pierwszych ważnych rozróżnień, jakich dokonano w Netflixie, było to, że inżynieria chaosu jest formą eksperymentowania, a nie testowania. Według wielu oba te terminy łączy kategoria „zapewnienia jakości", ale jest to pojęcie, które często ma negatywne konotacje w branży oprogramowania.

Pozostałe zespoły w Netflixie początkowo pytały Zespół Chaosu, czy ten nie może napisać kilku testów integracyjnych, które szukają tego samego. Ta prośba była pragmatyczna w swym założeniu, ale w praktyce testy integracyjne nie mogły dać oczekiwanego rezultatu.

Testowanie, ściśle rzec ujmując, nie daje nowej wiedzy. Testowanie wymaga, żeby inżynier piszący test znał konkretne właściwości systemu i ich szukał. Jak było widać w poprzednim rozdziale, złożone systemy nie poddają się tego rodzaju analizie. Ludzie po prostu nie są w stanie zrozumieć wszystkich potencjalnych skutków ubocznych możliwych interakcji części takiego systemu. Prowadzi nas to do jednej z kluczowych cech testu.

Test dokonuje asercji w oparciu o istniejącą wiedzę, a samo jego wykonanie weryfikuje ją jako prawdziwą lub fałszywą. Testy to stwierdzenia na temat znanych właściwości systemu.

Eksperymentowanie natomiast generuje nową wiedzę. Eksperymenty przedstawiają pewną hipotezę, a dopóki nie zostaje ona obalona, rośnie zaufanie do niej. Jeśli hipoteza zostaje obalona — dowiadujemy się czegoś nowego. Rozpoczyna to także dociekania, dlaczego hipoteza jest błędna. W systemach złożonych powód, dla którego coś się dzieje, często nie jest jasny. Eksperymentowanie albo buduje pewność, albo uczy nas nowej cechy naszego systemu. To eksploracja tego, co nieznane.

W praktyce żadna liczba testów nie dorówna wnioskom zebranym podczas eksperymentowania, ponieważ testowanie wymaga tego, żeby człowiek dostarczył asercje z góry. Eksperymentowanie formalnie otwiera nowe drogi dowiedzenia się czegoś o cechach systemu. Jest oczywiście całkowicie możliwe, by nowo odkryte cechy systemu poddawać później testom. Pomaga to także ubrać nowe założenia na temat systemu w formę nowych hipotez, co daje rodzaj „eksperymentu regresji", który bada zmiany systemu w czasie.

Ponieważ inżynieria chaosu powstała na gruncie problemów systemów złożonych, jest kluczowe, by dyscyplina ta kładła nacisk na eksperymentowanie, a nie testowanie. Cztery kroki eksperymentu skondensowane w „Zasadach" z grubsza odpowiadają powszechnie akceptowanym definicjom z punktu widzenia badania podatności związanych z dostępnością systemów dużej skali.

Połączenie rzeczywistego doświadczenia w stosowaniu wcześniej zdefiniowanych kroków do systemów dużej skali oraz rozważnej introspekcji poprowadziło Zespół Chaosu dalej, niż tylko do eksperymentowania. Te wnioski dały początek „Zaawansowanym zasadom", które kierują zespoły do dojrzałości ich projektów inżynierii chaosu i pomagają w wyznaczeniu złotego standardu, do którego aspirujemy.

Weryfikacja i walidacja

Posługując się definicjami weryfikacji i walidacji zaczerpniętymi z zarządzania działalnością operacyjną oraz planowania logistycznego, możemy powiedzieć, że inżynieria chaosu stanowczo opowiada się po stronie tej pierwszej.

Weryfikacja

Weryfikacja systemu złożonego to proces analizowania danych wyjściowych na granicy systemu. Domownik może zweryfikować jakość wody (dane wyjściowe) płynącej z kranu (granica systemu) testując ją pod kątem zanieczyszczeń, nie wiedząc nic na temat hydrauliki ani miejskiego wodociągu (części systemu).

Walidacja

Walidacja systemu złożonego to proces analizowania części systemu i budowania modelu myślowego, który odpowiada interakcjom między tymi częściami. Domownik może zwalidować jakość wody, nie tylko badając wszystkie rury, ale całą infrastrukturę wodociągową (części systemu), która zapewnia zebranie, oczyszczenie i dostarczenie jej (model myślowy części funkcjonalnych) najpierw na osiedle, a wreszcie do konkretnego mieszkania.

Obie te praktyki potencjalnie są użyteczne i obie budują pewność co do rezultatów systemu. Jako inżynierowie oprogramowania często czujemy obowiązek zagłębienia się w kod i zwalidowania, czy odpowiada on naszemu modelowi myślowemu tego, jak powinien działać. Wbrew tej skłonności inżynieria chaosu stanowczo opowiada się po stronie walidacji. W dziedzinie tej interesuje nas, czy coś działa, a nie jak.

Zwróć uwagę, że w naszej hydraulicznej metaforze możemy zwalidować wszystkie komponenty związane z dostarczaniem czystej wody pitnej, a w szklance nadal mieć zanieczyszczoną wodę z powodu, którego nie przewidzieliśmy. W systemach złożonych zawsze są jakieś nieprzewidziane interakcje. Ale jeśli zweryfikujemy, że woda w kranie jest czysta, nie musimy już się interesować tym, jak się tam dostała. W większości przypadków biznesowych rezultat systemu jest znacznie ważniejszy niż to, czy implementacja odpowiada naszemu modelowi myślowemu. Inżynieria chaosu stawia na przypadki biznesowe i rezultat, a nie na implantacje czy model myślowy współdziałających części.

Czym nie jest inżynieria chaosu

Istnieją dwa pojęcia, które często są mylone z inżynierią chaosu, mianowicie: psucie wszystkiego w środowisku produkcyjnym i antykruchość.

Psucie wszystkiego

Od czasu do czasu w postach na blogach czy podczas prezentacji na konferencjach można się spotkać z nazywaniem inżynierii chaosu „psuciem wszystkiego w środowisku produkcyjnym". Chociaż może brzmi to fajnie, takie sformułowanie nie przemawia do korporacji działających na wielką skalę czy innych operatorów złożonych systemów, którzy nie odniosą z takiej praktyki żadnej korzyści. Lepszą charakterystyką inżynierii chaosu byłoby naprawianie w środowisku produkcyjnym. „Psucie wszystkiego" jest proste. Trudność dotyczy ograniczania wpływu awarii, krytycznego myślenia o bezpieczeństwie, oceniania, czy coś warto naprawiać, decydowania, czy należy inwestować w dany eksperyment itd. „Psuć wszystko" można na mnóstwo sposobów bez spędzania na tym wielu dni czy tygodni. Ważniejszym pytaniem w tej sytuacji jest jednak, jak myśleć o tym, co już jest popsute, kiedy nawet nie wiemy, że nie działa.

„Naprawianie na produkcji" znacznie trafniej oddaje wartość inżynierii chaosu, ponieważ celem tej praktyki jest proaktywne poprawianie dostępności i bezpieczeństwa systemów złożonych. Istnieje już wiele dyscyplin i narzędzi zajmujących się odpowiadaniem na incydenty: narzędzia związane z alarmami, zarządzanie kryzysowe, narzędzia monitorowania, plany przywrócenia gotowości itd. Ich celem jest zredukowanie czasu wykrycia awarii i czasu jej naprawienia po wystąpieniu nieuniknionego incydentu. Można dyskutować, czy Site Reliability Engineering (SRE) łączy w sobie reaktywne i proaktywne podejście poprzez generowanie nowej wiedzy na podstawie przeszłych incydentów i uwspólnianiu jej, by uniknąć powtórek w przyszłości. Inżynieria chaosu jest jednak jedyną kompletną dyscypliną w ramach oprogramowania, która koncentruje się wyłącznie na proaktywnym poprawianiu bezpieczeństwa systemów złożonych.

Antykruchość

Ludzie, którzy znają wprowadzone przez Nassima Taleba pojęcie antykruchości[1], często zakładają, że inżynieria chaosu jest zasadniczo jego wersją odnoszącą się do oprogramowania. Taleb argumentuje, że słowa takie jak „hormeza" są niewystarczające, by uchwycić zdolność złożonych systemów do adaptacji, więc wymyślił pojęcie „antykruchości" na określenie systemów, które stają się silniejsze dzięki wystawieniu ich na działanie przypadkowego obciążenia. Ważnym, wręcz zasadniczym rozróżnieniem między inżynierią chaosu a antykruchością jest to, że inżynieria chaosu uczy ludzi utrzymujących systemy o chaosie, który jest już integralną cechą systemu tak, by zespół był bardziej odporny. Antykruchość z kolei dodaje chaosu do systemu w nadziei, że dzięki temu stanie się on silniejszy, a nie podda się działaniu obciążenia.

Jako model antykruchość postuluje działania stojące w sprzeczności z naukami inżynierii odporności czy badaniami nad czynnikami ludzkimi i bezpieczeństwem systemów. Przykładowo antykruchość proponuje, by pierwszym krokiem w zwiększeniu solidności systemu było poszukiwanie słabości i ich usuwanie. Taka sugestia jest intuicyjna, ale inżynieria odporności uczy nas, że szukanie tego, co działa *dobrze* pod kątem bezpieczeństwa, daje dużo więcej informacji niż szukanie tego, co idzie *źle*. Kolejnym krokiem antykruchości jest dodawanie nadmiarowości. To również wydaje się intuicyjne, ale dodawanie nadmiarowości może równie dobrze spowodować awarię, jak jej zapobiec, a literatura na temat inżynierii odporności przytacza mnóstwo przykładów, w których to właśnie nadmiarowość przyczynia się do incydentu związanego z bezpieczeństwem[2].

Istnieje wiele innych przykładów rozbieżności między tymi dwiema szkołami. Inżynieria odporności jest przedmiotem trwających badań i cieszy się dekadami wsparcia społeczności, podczas gdy antykruchość jest teorią funkcjonującą poza uczelniami i przeglądami koleżeńskimi. Łatwo sobie wyobrazić, dlaczego te dwa pojęcia często się łączy, ponieważ oba dotyczą chaosu i złożonych systemów, ale duch antykruchości nie podziela tego samego empiryzmu i zasadniczej przyziemności co inżynieria chaosu. Z tego względu powinniśmy uznawać je za dwa odrębne pola[3].

[1] Nassim Taleb, *Antifragile: Things That Gain from Disorder*, Penguin, New York 2012.

[2] Być może najsławniejszym przykładem jest katastrofa promu Challenger w 1986 r. Nadmiarowość pierścieni uszczelniających była jednym z trzech powodów, dla których NASA zdecydowała się kontynuować starty, mimo że uszkodzenia głównego pierścienia były dobrze znane z przeszło 50 wcześniejszych startów na przestrzeni ok. 5 lat. Patrz Diane Vaughan, *The Challenger Launch Decision*, University of Chicago Press, Chicago 1997.

[3] Casey Rosenthal, *Antifragility Is a Fragile Concept*, wpis na blogu LinkedIn, 28 sierpnia 2018, *https://oreil.ly/LbuIt*.

Zaawansowane zasady

Inżynieria chaosu jest zakorzeniona w empiryzmie, przedkłada eksperymentowanie nad testowanie i weryfikację nad walidację. Ale nie wszystkie eksperymenty są tak samo wartościowe. Złoty standard praktyki został po raz pierwszy wyrażony w należącej do „Zasad" sekcji „Zaawansowane zasady". Oto one:

- Zbuduj hipotezę dotyczącą zachowania w stabilnym stanie systemu.
- Zróżnicuj rzeczywiste przypadki.
- Przeprowadzaj eksperymenty w środowisku produkcyjnym.
- Automatyzuj eksperymenty, by uruchamiać je cały czas.
- Minimalizuj zakres szkód.

Zbuduj hipotezę dotyczącą zachowania systemu w stabilnym stanie

Każdy eksperyment zaczyna się od hipotezy. Forma eksperymentu z zakresu dostępności to zwykle:

W sytuacji gdy ....., klienci nadal będą się dobrze bawić.

Z kolei eksperymenty dotyczące bezpieczeństwa zwykle mają formę:

W sytuacji gdy ....., zespół ds. bezpieczeństwa zostanie powiadomiony.

W obu przypadkach puste miejsca w zdaniu wypełniają zmienne omawiane w kolejnej sekcji.

Zaawansowane zasady podkreślają budowanie hipotez dotyczących stanu stabilności systemu. Oznacza to skoncentrowanie się na tym, jak system ma się zachowywać, i ujmowanie tych oczekiwań we wskaźnikach. W powyższych przykładach klienci prawdopodobnie domyślnie dobrze się bawią, a zespół ds. bezpieczeństwa dostaje powiadomienie, gdy coś narusza kontrolę bezpieczeństwa.

To skoncentrowanie na stabilnym stanie zmusza inżynierów, by wyszli na chwilę poza granice kodu i przeanalizowali całościowy rezultat. Pobrzmiewa w tym echo tendencji inżynierii chaosu do przedkładania weryfikacji nad walidację. Często mamy ochotę zanurzyć się w problemie, znaleźć „pierwszą przyczynę" zachowania i spróbować zrozumieć system poprzez redukcjonizm. Zagłębienie się w problem może pomóc w badaniach, ale stanowi też rozproszenie w stosunku do najbardziej efektywnych lekcji, jakie oferuje inżynieria chaosu. W najlepszych przypadkach inżynieria chaosu skupia się na kluczowych wskaźnikach efektywności (KPI) lub innych wskaźnikach, które mierzą jasne priorytety biznesowe, a to one stanowią najlepszą definicję stanu stabilnego.

Zróżnicuj rzeczywiste przypadki

Ta zaawansowana zasada głosi, że zmienne w eksperymentach powinny odpowiadać zdarzeniom ze świata rzeczywistego. Choć patrząc wstecz, może się to wydawać oczywiste, są dwa dobre powody, dla których warto jawnie zapisać tę zasadę:

- Zmienne często wybiera się w oparciu o to, co najłatwiej zrobić, a nie o to, co pozwala się najwięcej dowiedzieć.

- Inżynierowie mają skłonność do skupiania się na zmiennych odpowiadających ich doświadczeniu, a nie doświadczeniu użytkownika.

Unikaj wybierania prostej ścieżki

Chaos Monkey[4] to w zasadzie trywialna aplikacja jak na tak potężny program. To rozwiązanie wolnoźródłowe, które losowo wyłącza instancje (wirtualne maszyny, kontenery lub serwery) mniej więcej raz dziennie dla każdej usługi. Możesz używać go w takiej formie, w jakiej dostępne jest od ręki, ale takie same możliwości mógłby w większości organizacji dać skrypt w bashu. To łatwo osiągalny cel inżynierii chaosu. Wydania w chmurze i wydania na kontenery sprawiają, że każdemu systemowi dużej skali będą nagle znikać instancje (maszyny wirtualne lub kontenery) w mniej więcej regularnych odstępach czasowych. Chaos Monkey powiela te zmienne i po prostu przyspiesza częstotliwość takich zajść.

Jest to użyteczne i dość proste do zrobienia, jeśli masz uprawnienia administratora w infrastrukturze dające możliwość usuwania instancji. Gdy zdobędziesz takie uprawnienia, pojawi się pokusa wykonywania także innych operacji, do których uzyskasz w ten sposób dostęp. Zastanów się nad poniższymi zmiennymi, jakie można wprowadzić na instancji:

- Zakończenie działania instancji.
- Zawieszenie procesora instancji.
- Zużycie całej dostępnej dla instancji pamięci.
- Zapełnienie dysku instancji.
- Wyłączenie połączeń sieciowych instancji.

Te eksperymenty łączy jedno: przewidywalnie spowodują one, że instancja przestanie odpowiadać. Z punktu widzenia systemu wygląda to zupełnie tak samo, jakbyś wyłączył instancję. Z ostatnich czterech eksperymentów nie dowiesz się niczego, czego nie nauczy Cię już ten pierwszy. Te eksperymenty to zasadniczo strata czasu.

Biorąc pod uwagę systemy rozproszone, niemal wszystkie interesujące eksperymenty związane z dostępnością dotyczą opóźnienia lub typu odpowiedzi. Wyłączenie instancji jest specjalnym przypadkiem nieskończonego opóźnienia. W większości obecnych systemów online typ odpowiedzi jest tożsamy z kodem statusowym, np. zmianą HTTP 200 na 500. Wynika z tego, że większość eksperymentów dotyczących dostępności można skonstruować za pomocą mechanizmu różnicującego opóźnienie i zmieniającego kody odpowiedzi.

Zróżnicowanie opóźnienia znacznie trudniej zorganizować niż zawieszenie procesora czy wypełnienie pamięci RAM instancji. Wymaga to skoordynowanego zaangażowania wszystkich warstw komunikacji międzyprocesowej (IPC). Może to oznaczać konieczność modyfikacji rozszerzeń, reguł sieciowych zdefiniowanych w oprogramowaniu, obudowy bibliotek po stronie klienta, pośredników ruchu, a nawet lekkich systemów równoważenia obciążenia. Każde z tych rozwiązań wymaga nietrywialnej inwestycji inżynierskiej.

[4] Chaos Monkey było pierwszym narzędziem inżynierii chaosu. Patrz „Wprowadzenie. Narodziny chaosu" — historia powstania inżynierii chaosu.

Przeprowadzaj eksperymenty w środowisku produkcyjnym

Eksperymentowanie uczy Cię systemu, który badasz. Jeśli eksperymentujesz na środowisku preprodukcyjnym, budujesz pewność co do *tego* środowiska. W takim zakresie, w jakim środowisko produkcyjne i preprodukcyjne się różnią — często na sposoby, których ludzie nie są w stanie przewidzieć — nie budujesz pewności co do środowiska, na którym naprawdę Ci zależy. Z tego względu najbardziej zaawansowana inżynieria chaosu rozgrywa się w środowisku produkcyjnym.

Zasada ta budzi kontrowersje. Z pewnością w pewnych obszarach istnieją wymagania regulacyjne, które wykluczają możliwość oddziaływania na system produkcyjny. W pewnych sytuacjach istnieją niedające się pokonać techniczne bariery, które nie pozwalają na przeprowadzenie eksperymentów w tym otoczeniu. Trzeba w tym miejscu pamiętać, że celem inżynierii chaosu jest odkrycie chaosu integralnie tkwiącego już w systemie, a nie spowodowanie go. Jeśli wiemy, że eksperyment da niepożądane rezultaty, nie powinniśmy go przeprowadzać. Jest to wskazówka szczególnie ważna w kontekście środowiska produkcyjnego, w którym konsekwencje obalenia hipotezy mogą być poważne.

Ta zasada nie jest jednak zero-jedynkowa. W większości sytuacji całkowicie zasadne jest rozpoczęcie eksperymentowania na systemie preprodukcyjnym i stopniowe przenoszenie go w środowisko produkcyjne, gdy tajniki narzędzi są już dobrze rozpracowane. W wielu przypadkach kluczowe obserwacje dotyczące produkcji po raz pierwszy ujawniają się dzięki inżynierii chaosu już w środowisku preprodukcyjnym.

Automatyzuj eksperymenty, by uruchamiać je cały czas

Ta zasada pozwala dojść do głosu praktycznym konsekwencjom pracy nad złożonymi systemami. Automatyzacja jest konieczna z dwóch powodów:

- Aby pokryć szeroki zbiór eksperymentów, które ludzie mogą wykonać ręcznie. W złożonych systemach jest tak wiele warunków, które mogą potencjalnie przyczynić się do powstania incydentu, że nie da się ich zaplanować. Tak naprawdę nie da się ich nawet policzyć, bo nie wiadomo o nich z góry. Oznacza to, że ludzie nie są w stanie w rozsądnym czasie realnie szukać przestrzeni rozwiązań dotyczących czynników przyczyniających się do incydentu. Automatyzacja pozwala zeskalować te poszukiwania podatności, które mogą doprowadzić do niepożądanych zachowań systemu.
- Aby empirycznie zweryfikować nasze założenia w czasie, podczas gdy nieznane części systemu się zmieniają. Wyobraź sobie system, w którym funkcjonalność pewnego komponentu zależy od innego komponentu pozostającego poza kontrolą pierwszego operatora. Coś takiego dzieje się w niemal wszystkich systemach złożonych. Bez ścisłego wiązania między danym komponentem a wszystkimi jego zależnościami jest zupełnie możliwe, że któraś z zależności zmieni się tak, iż zrodzi to podatność. Ciągłe eksperymentowanie możliwe dzięki automatyzacji będzie w stanie zauważyć ten problem i nauczyć operatorów działania ich własnego systemu, które zmienia się z czasem. Może być to zmiana w wydajności (np. sieć nasyca się ze względu na hałaśliwych sąsiadów), zmiana funkcjonalności (np. ciała odpowiedzi wywoływanych usług zawierają dodatkowe informacje, co może wpłynąć na sposób ich przetwarzania) lub zmiana oczekiwań (np. inżynierowie opuszczają zespół, a nowi operatorzy nie znają jeszcze kodu).

Sama automatyzacja może jednak mieć niezamierzone konsekwencje. W rozdziałach 11. i 12. w części III rozważamy pewne wady i zalety automatyzacji. Zaawansowane zasady utrzymują jednak, że automatyzacja to rozbudowany mechanizm do badania przestrzeni rozwiązania potencjalnych podatności i urzeczowiania instytucjonalnej wiedzy o podatnościach poprzez weryfikowanie hipotezy w czasie ze świadomością, że złożony system będzie się zmieniał.

Minimalizuj zakres szkód

Ostatnia zaawansowana zasada została dodana do „Zasad" po tym, jak Zespół Chaosu w Netflixie odkrył, że mógł znacząco obniżyć ryzyko w ruchu produkcyjnym poprzez stworzenie bezpieczniejszych sposobów przeprowadzania eksperymentów. Korzystając ze ściśle dystrybuowanej grupy kontrolnej w celu porównania z grupą zmienną, eksperymenty można konstruować tak, że wpływ obalonej hipotezy na produkcyjny ruch płynący od klientów jest minimalny.

To, jak zespół osiągnie ten stan, jest wysoce zależne od kontekstu danego systemu złożonego. W pewnych systemach może to wymagać śledzenia ruchu, wyłączenia spod eksperymentu żądań o dużej wadze biznesowej, np. transakcji powyżej 100 dolarów lub implementacji zautomatyzowanej logiki ponowień żądań na wypadek, gdyby eksperyment się nie powiódł. W przypadku Zespołu Chaosu w Netflixie próbkowanie żądań, podtrzymywanie sesji i podobne mechanizmy zostały dodane do Platformy Automatyzacji Chaosu (ang. *Chaos Automation Platform*, ChAP)[5], którą przedstawiamy bliżej w rozdziale 16. Techniki te nie tylko ograniczyły zakres szkód, ale też przyniosły dodatkową korzyść w postaci wzmocnienia sygnału detekcji, ponieważ wskaźniki małej grupy zmiennej mogą często bardzo rzucać się w oczy w porównaniu z małą grupą kontrolną. Niezależnie od tego, jak się ten stan osiąga, zasada ta podkreśla, że w prawdziwie finezyjnej implementacji inżynierii chaosu potencjalny zakres szkód wynikających z eksperymentu można umyślnie ograniczyć.

Wszystkie te zaawansowane zasady zaprezentowano bo to, by wskazywać kierunki i inspirować, a nie dyktować. Powstały one z pragmatyzmu i powinny być przyjmowane (lub odrzucane) z tym celem na uwadze.

Koncentracja na doświadczeniu użytkownika

Wiele można mówić o poprawie doświadczenia programistów. DevUX to dyscyplina niedoceniana. Zgodne wysiłki na rzecz poprawienia doświadczenia inżynierów oprogramowania podczas gdy piszą, utrzymują i dostarczają kod do środowiska produkcyjnego i cofają te zmiany, bardzo się opłaca w dłuższej perspektywie. Mając tego świadomość, trzeba jednak przyznać, że największa wartość biznesowa płynąca z inżynierii chaosu pochodzi ze znajdowania podatności w systemie produkcyjnym, nie procesie wytwórczym. Jest zatem celowe, aby skupiać się na zmiennych mogących wpłynąć na doświadczenie użytkownika.

Ponieważ to inżynierowie oprogramowania (nie użytkownicy) zwykle dobierają zmienne w eksperymentach chaosu, tej koncentracji czasem brakuje. Przykładem takiego chybionego skupienia jest entuzjazm inżynierów chaosu w związku z wprowadzaniem eksperymentów dotyczących uszkodzenia danych. Istnieje wiele miejsc, gdzie takie doświadczenia są uzasadnione i bardzo cenne.

[5] Ali Basiri i in., *ChAP: Chaos Automation Platform*, The Netflix Technology Blog, 26 lipca 2017, *https://oreil.ly/U7aaj*.

Jednym z przykładów jest weryfikowanie gwarancji baz danych. Z kolei modyfikowanie ciała odpowiedzi płynącej do klienta prawdopodobnie nie jest dobrym przykładem.

Rozważmy zyskujący na popularności przykład eksperymentu, w którym ciało odpowiedzi jest modyfikowane tak, że zwraca HTML o niepoprawnej strukturze lub uszkodzony format JSON. Jest niskie prawdopodobieństwo, że taka zmienna wystąpi rzeczywiście, a jeśli tak się stanie, to zazwyczaj w przypadku pojedynczych żądań, którym często zaradzi standardowe zachowanie użytkownika (próba ponowienia) lub wyjście awaryjne (w postaci innej próby ponowienia) czy po prostu zgrabny klient (np. przeglądarka internetowa).

Jako inżynierowie często obserwujemy nieścisłości kontraktów. Zauważamy, że biblioteki wchodzące w interakcje z naszym kodem zachowują się w sposób, jakiego nie chcemy. Poświęcamy mnóstwo czasu na dopasowanie tych interakcji, aby uzyskać od tych bibliotek oczekiwane zachowanie. Ponieważ widzimy dużo zachowań, których nie chcemy, uznajemy, że warto będzie przeprowadzić eksperymenty, które ujawnią takie przypadki. To założenie jest jednak fałszywe — w takich wypadkach inżynieria chaosu nie jest niezbędna. Negocjowanie nieścisłości kontraktów jest częścią procesu wytwórczego. To odkrycie inżyniera. Kiedy kod już działa, gdy kontrakt się wyklaruje, jest niezwykle mało prawdopodobne, że z powodu kosmicznego promieniowania albo przepalonego tranzystora rezultat biblioteki zostanie wypaczony, a dane w drodze zostaną uszkodzone. Nawet jeśli jest to coś, czym się chcemy zająć, ze względu na zmiany biblioteki itd., jest to znana właściwość systemu. Do znanych właściwości systemu odnosi się testowanie.

Inżynieria chaosu nie ma wystarczająco długiej historii, by sformalizować metody wykorzystywane do generowania zmiennych. Niektóre z nich są oczywiste, jak wprowadzanie opóźnienia. Niektóre wymagają analizy, jak wprowadzenie właściwego opóźnienia, które wywoła efekt zakolejkowania bez przekraczania limitów alarmów czy celów w kontrakcie usług. Jeszcze inne w olbrzymiej mierze zależą od kontekstu, jak obniżanie wydajności drugorzędnej usługi tak, że powoduje to, iż inna drugorzędna usługa staje się tymczasowo usługą pierwszorzędną.

W miarę jak dyscyplina ewoluuje, oczekujemy, że metody te zapewnią formalne modele generowania zmiennych lub choćby domyślne modele, które uchwytują uogólnione doświadczenie w branży. W międzyczasie unikaj wybierania łatwych ścieżek, skupiaj się na doświadczeniu użytkownika i różnicuj rzeczywiste zdarzenia.

Przyszłość „Zasad”

W ciągu pięciu lat od opublikowania „Zasad” widzieliśmy, jak inżynieria chaosu ewoluuje, by sprostać wyzwaniom w nowych branżach. Zasady i podstawy tej praktyki z pewnością nadal będą się zmieniać w miarę, jak przybywa adeptów tej dyscypliny w ramach przemysłu IT i w innych obszarach.

Gdy Netflix zaczął na poważnie rozgłaszać inżynierię chaosu poprzez Chaos Community Day w 2015 r., spotkał się z dużym sceptycyzmem zwłaszcza ze strony instytucji finansowych (patrz „Wprowadzenie. Narodziny chaosu”, gdzie znajdziesz więcej informacji na temat Chaos Community Day i początków popularyzacji ruchu). Powszechną wątpliwość wyrażała się w stwierdzeniu: „Jasne, może to sprawdza się dla usługi rozrywkowej lub reklam online, ale u nas na szali leżą pieniądze”. Zespół Chaosu odpowiadał wtedy: „A miewacie awarie?”.

Oczywiste, że odpowiedź brzmiała: „Tak". Nawet najlepsze zespoły inżynieryjne padają ofiarami awarii w instytucjach finansowych, gdzie stawki są wysokie. Pozostawia to dwie możliwości: (a) nadal mieć awarie o nieprzewidywalnej skali i częstotliwości lub (b) przyjąć proaktywną strategię taką jak inżynieria chaosu, by zrozumieć ryzyka i zapobiec dużym, niekontrolowanym rezultatom. Instytucje finansowe przystały na to rozumowanie i wiele największych banków na świecie ma specjalne programy inżynierii chaosu.

Kolejną branżą, która zgłaszała zastrzeżenia, była służba zdrowia. Ich obawa wyrażała się w zdaniu: „Jasne, może to sprawdza się dla usługi rozrywkowej lub instytucji finansowej, ale u nas na szali leży ludzkie życie". Wtedy Zespół Chaosu ponownie odpowiadał: „A miewacie awarie?".

W tym przypadku jednak można bardziej bezpośrednio odnieść się do podstaw służby zdrowia jako systemu. Kiedy eksperymenty empiryczne wybrano jako podstawę inżynierii chaosu, było to bezpośrednie odwołanie do koncepcji falsyfikacjonizmu Karla Poppera, które jest fundamentem zachodniego rozumienia nauki i metody naukowej. Zwieńczeniem popperyzmu w praktyce jest próba kliniczna.

Pod tym względem fenomenalny sukces zachodniego systemu służby zdrowia opiera się na inżynierii chaosu. Współczesna dyscyplina polega na próbach podwójnie ślepych, w których stawką jest ludzkie życie. Używa się na to po prostu innej nazwy: próba kliniczna.

Formy inżynierii chaosu niejawnie istniały w innych branżach od dawna. Wyeksponowanie eksperymentów, zwłaszcza w zakresie działań związanych z oprogramowaniem, daje siłę praktyce. Mówienie o nich głośno i celowe nazwanie ich inżynierią chaosu pozwala nam zbudować strategię dookoła ich celu i zastosowania, a także przyjąć obserwacje wyniesione z innych obszarów i zastosować je we własnym ogródku.

W tym duchu możemy badać inżynierię chaosu w branżach i firmach odległych od prototypowych skalowanych mikrousług, które zwykle kojarzymy z tą praktyką. Branża technologii finansowych, pojazdy autonomiczne, antagonistyczne uczenie maszynowe — te dyscypliny mogą nas wiele nauczyć na temat potencjału i ograniczeń inżynierii chaosu. Inżynieria mechaniczna i awiacja poszerzają nasze rozumienie jeszcze bardziej, przenosząc nas od oprogramowania do sprzętu i fizycznego prototypowania. Inżynieria chaosu wyszła poza dostępność w kierunku zabezpieczenia przed zagrożeniami, które jest drugą stroną bezpieczeństwa systemu. Wszystkie te nowe branże, przypadki użycia i środowiska będą napędzały dalsze zmiany podstaw i zasad inżynierii chaosu.

Rośnie złożoność systemu? Potrzebujemy chaosu!

W miarę rozwoju systemu rośnie jego złożoność. Skomplikowane systemy uważa się za trudniejsze w zarządzaniu i bardziej podatne na awarie. Nie można uniknąć złożoności systemu w czasach błyskawicznego rozwoju mikrousług i technologii rozproszonych, ale można nad nią zapanować. Odpowiednio zaplanowane testy i eksperymenty pozwalają wykryć podatności i zapobiec wystąpieniu problemów, zanim zaczną utrudniać realizację celów biznesowych firmy. Relatywnie nowym, lecz wyjątkowo obiecującym narzędziem służącym do tych celów jest inżynieria chaosu.

Ta książka jest praktycznym wprowadzeniem do inżynierii chaosu w zarządzaniu złożonymi systemami podczas ich optymalizacji — zawiera gruntowne podstawy tej nowej dziedziny wraz z wyjaśnieniem zasad postępowania. Pokazuje też procesy, dzięki którym można doprowadzić do uzyskania wysokiej odporności na awarie. Opisano tu najskuteczniejsze praktyki inżynierii chaosu i poparto je licznymi przykładami. Zaprezentowano techniki testowania, eksperymentowania i wstrzykiwania awarii. Wyczerpująco omówiono znaczenie i sposoby planowania, a także zarządzania zespołami w kontekście budowania odporności złożonych systemów na awarie. Co ciekawe, zasady inżynierii chaosu mogą znaleźć zastosowanie nie tylko w odniesieniu do tworzenia i utrzymywania oprogramowania, ale również do budowania niezawodności innych złożonych systemów.

Najciekawsze zagadnienia:

- rola inżynierii chaosu w zarządzaniu złożonością
- metody unikania awarii w aplikacjach, sieci i infrastrukturze
- rozumienie złożoności w systemach oprogramowania
- testy i eksperymenty w inżynierii chaosu
- inżynieria chaosu a optymalizacja systemów

Casey Rosenthal jest prezesem i współzałożycielem firmy Verica, wcześniej kierował zespołem do spraw inżynierii chaosu w Netfliksie. Zdobył doświadczenie w pracy z systemami rozproszonymi i ze sztuczną inteligencją. Prelegent na wielu prestiżowych konferencjach.

Nora Jones jest prezesem i założycielką Jeli, a także liderką technologiczną i programistką. Jej wystąpienie na konferencji re:Invent, zorganizowanej przez AWS w 2017 roku, zapoczątkowało ruch inżynierii chaosu.

ISBN 978-83-283-8285-5