How to write Cynical software

Stability patterns and anti-patterns

dagi@goodata.com
https://twitter.com/_dagi
How to become cynical

- Eat your own dog food
- Strong feedback
- Pager duty (Engineer on duty)
- DevOps

How i’ve got there
No throw over the wall syndrom
Monday 4 November 13

- shift toward SOA, interconnected services, remote communication, 3rd party services
- how a crack may appear – failure mode
- integration point
- cracks are tightly coupled to integration points
- the way cracks appear and propagated across multiple layers and services
- Integration points may accelerate (chained reaction) or stop cracks
- Failure in one component increase the probability of failure in another component service
- Slow responses, endpoint unreachability
- High levels of complexity provide more directions for the cracks to propagate in.
Upload a CSV file
Select a file to upload to your GoodData project.

Choose File...

Uploading tips:
- See our CSV formatting guidelines to ensure successful upload
- Contact Support if you have any troubles

Monday 4 November 13
try {
    projects = dao.getProjects(user, OFFSET, LIMIT).await(250, MS);
} catch (FutureTimeoutException e) {
    throw new AccountInfoUnavailableException();
}

for (Project project : projects) {
    Set<String> userPermissions = dao.getPermissions(project, user);
    if (userPermissions.contains(CAN_INIT_DATA)) {
        return new AccountInfo(Boolean.TRUE);
    }
}
Bear's vital systems are undergoing maintenance

At the moment, the application is undergoing maintenance. We take downtime seriously, and we're working to return to service shortly.

For information regarding this outage:
Support Portal
(415) 200-0194
“Cynicism is merely the art of seeing things as they are instead of as they ought to be” [1.]

A cynical software

No intimacy

Internal barriers

Resilience to impulse and stress

Lack of trust

Bad things happen

Monday 4 November 13

Explain the main attributes of cynical software
the antipatterns will create, accelerate or multiply cracks in the system
the patterns provide architecture and design guidance to reduce, eliminate, or mitigate the effects of cracks in the system
Circuit breaker

- mediator (decoupling, isolation), integration point wrapper
- fail fast
Bulkheads

HTTP client
Connections pool

Bulkheads

Frontend service

Hiccup

Backend service A

Backend service B

Backend service C

Monday 4 November 13

fat tails – sizing/capacity isolation
Latency and Fault Tolerance for Distributed Systems

https://github.com/Netflix/Hystrix

- do not reinvent the wheel
- OSS, Java
- most of the patterns are implemented there (circuit breaker, bulkheads, fail fast)
public class CommandHelloWorld extends HystrixCommand<String> {

    private final String name;

    public CommandHelloWorld(String name) {
        super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));
        this.name = name;
    }

    @Override
    protected String run() {
        // a real example would do work like a network call here
        return "Hello " + name + "!";
    }
}

String s = new CommandHelloWorld("World").execute();

Future<String> fs = new CommandHelloWorld("World").queue();

Monday 4 November 13

HystrixCommand = circuit breaker + bulkheads
Synchronous/Asynchronous usage
Async usage Future -> timeout
Focus on failures

Testing

Simulate bad things

Chaos monkey [2]

- we do test but on "slightly" different topology. It makes hard to reveal some kind of bugs
- we mostly test optimistic cases
- HTTP mock server (bad responses, slow responses, protocol violation...)
- Longevity tests
Adaptable design

Architecture

Conway’s law

ROC

- early decisions are hard to revert later (costs)
- Big Up Front Design doesn’t work prefer Adaptable design – Framework, Platform
- restartability (No restart the world), diagnostics (health checks), recovery mechanism – circuit breaker, isolation/redundancy ()
- Health checks – do what a user does
- Automatic thread dump on service restart
- White box – logging, Black box – monitoring (JMX)
- Troubleshooting vs. Awareness
Release It!
Design and Deploy Production-Ready Software

Michael T. Nygard

Q&A