Microservices

Testing

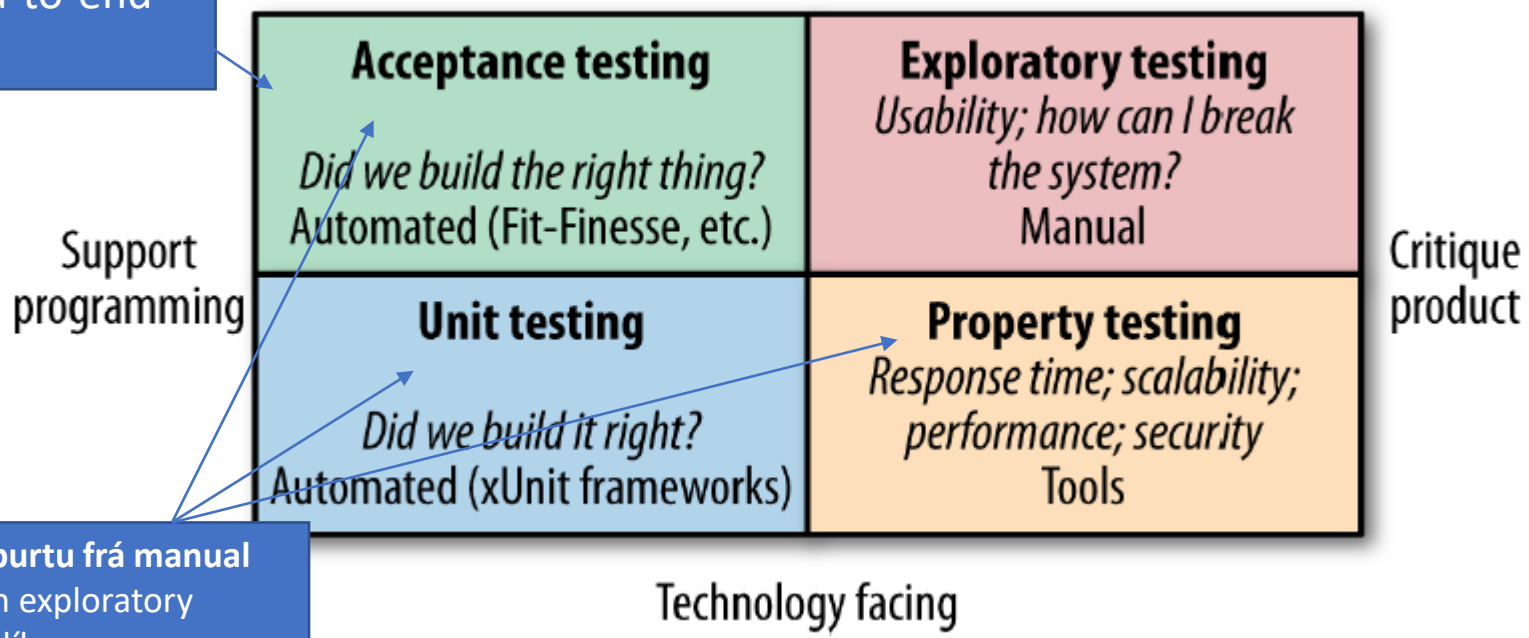
Hönnun og smíði hugbúnaðar

Haust 2022

Test Tegundir

Business facing test hjálpa nontechnical stakeholder-um að skilja hvernig kerifð virkar

t.d. end-to-end testing



Trendið er að reyna að **fara í burtu frá manual testing** í automated testing en exploratory testing er auðvitað mikilvægt líka.

Við viljum hafa flest test automated, hröð og skilvirk

Technology facing, s.s test sem hjálpa forriturunum að búa til kerfið. Eru yfirleitt automated

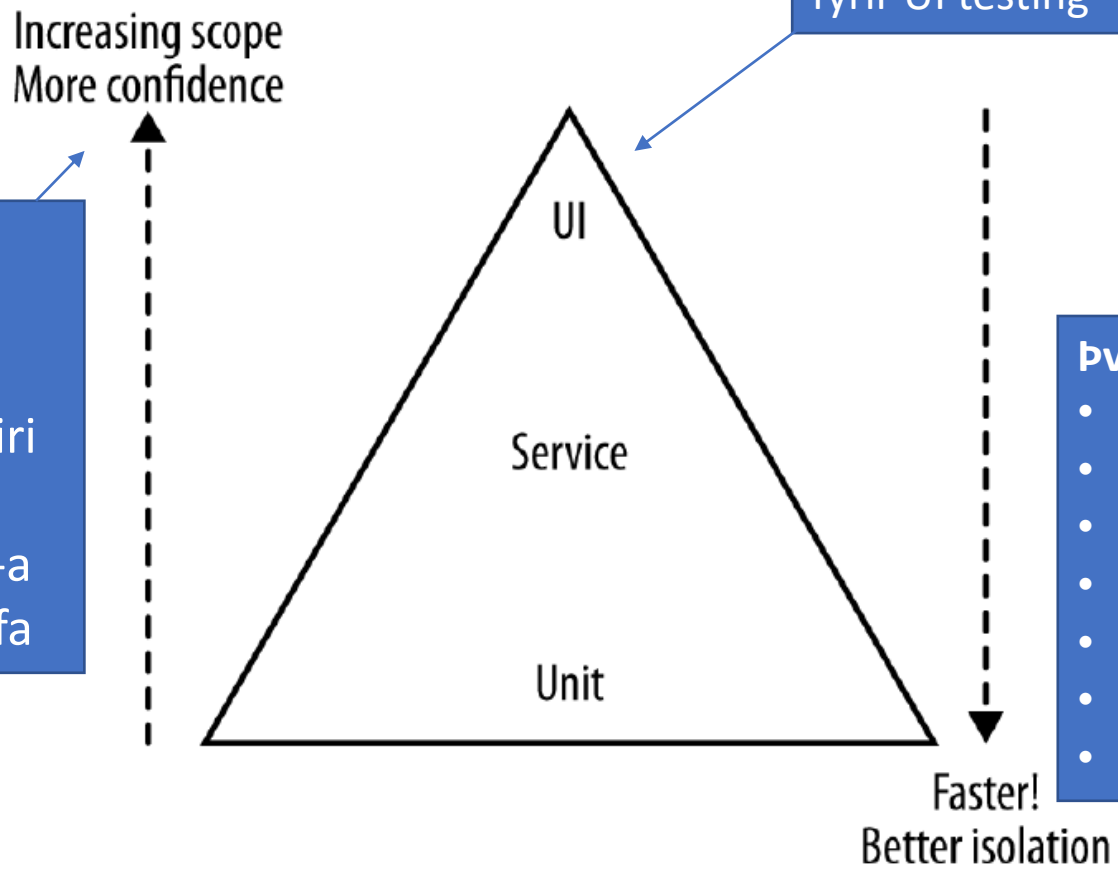
Manual vs Automation

- Viljum automate-a eins mikið og mögulegt
- Viljum fjarlægja endurtekin verk
- Sumt erfitt að sjálfvirknivæða og þá þarf manual tests
- Exploratory Testing einnig mjög mikilvæg
  - Sumt sem notandi kerfisins getur eingöngu séð
- Hvað við viljum sjálfvirknivæða er jafnvægislist

Test Scope

Jafnvægisleikur að finna hversu mörg af hverju test-i við ættum að hafa

end-to-end test er betra nafn fyrir UI testing



Því hærra sem þú ferð:

- Því meira öryggi á virkni
- Umfang verður stærra
- Endurgjafar tími verður meiri
- Test verða brothættari
- Erfiðara að vita afh test fail-a
- Því færri test ættirðu að hafa

Því neðar sem þú ferð:

- Umfang verður minna
- Test verða nákvæmari
- Endurgjafartími verður minni
- Test verða stöðugari
- Auðeldar að vita hvað brotnaði
- En minna öryggi um að kerfið virki
- Því fleiri test ættirðu að hafa

Dæmi

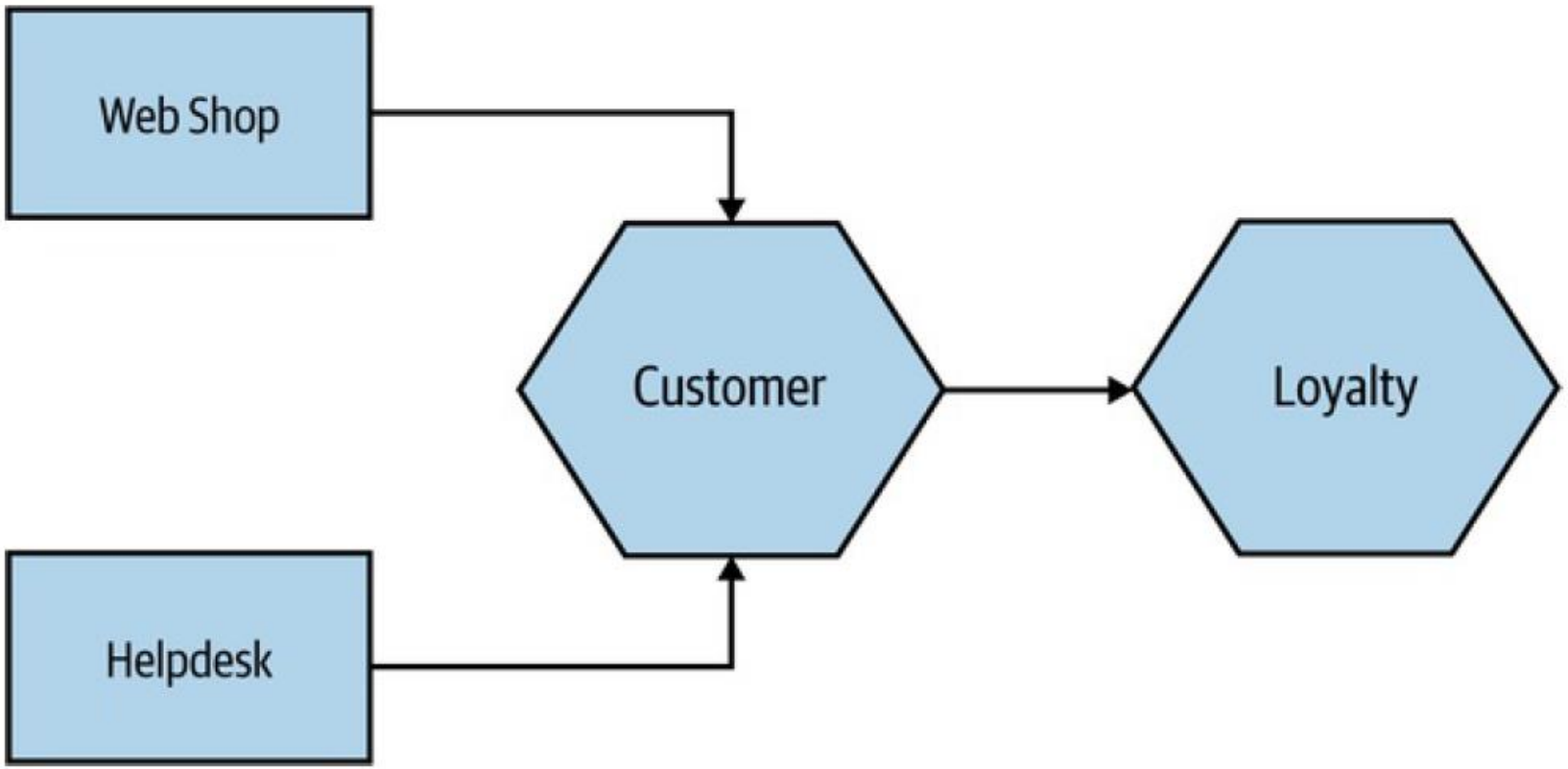


Figure 9-3. Part of our music shop under test

Unit Tests

Unit Test test-ar yfirleitt bara eitt fall

Ættu að grípa flestar villurnar okkar

Eiga að keyra hratt og gefa hraða endurgjöf

Ætttu að vera einangruð frá external resources t.d. öðrum services, gagnagrunn, file system etc.

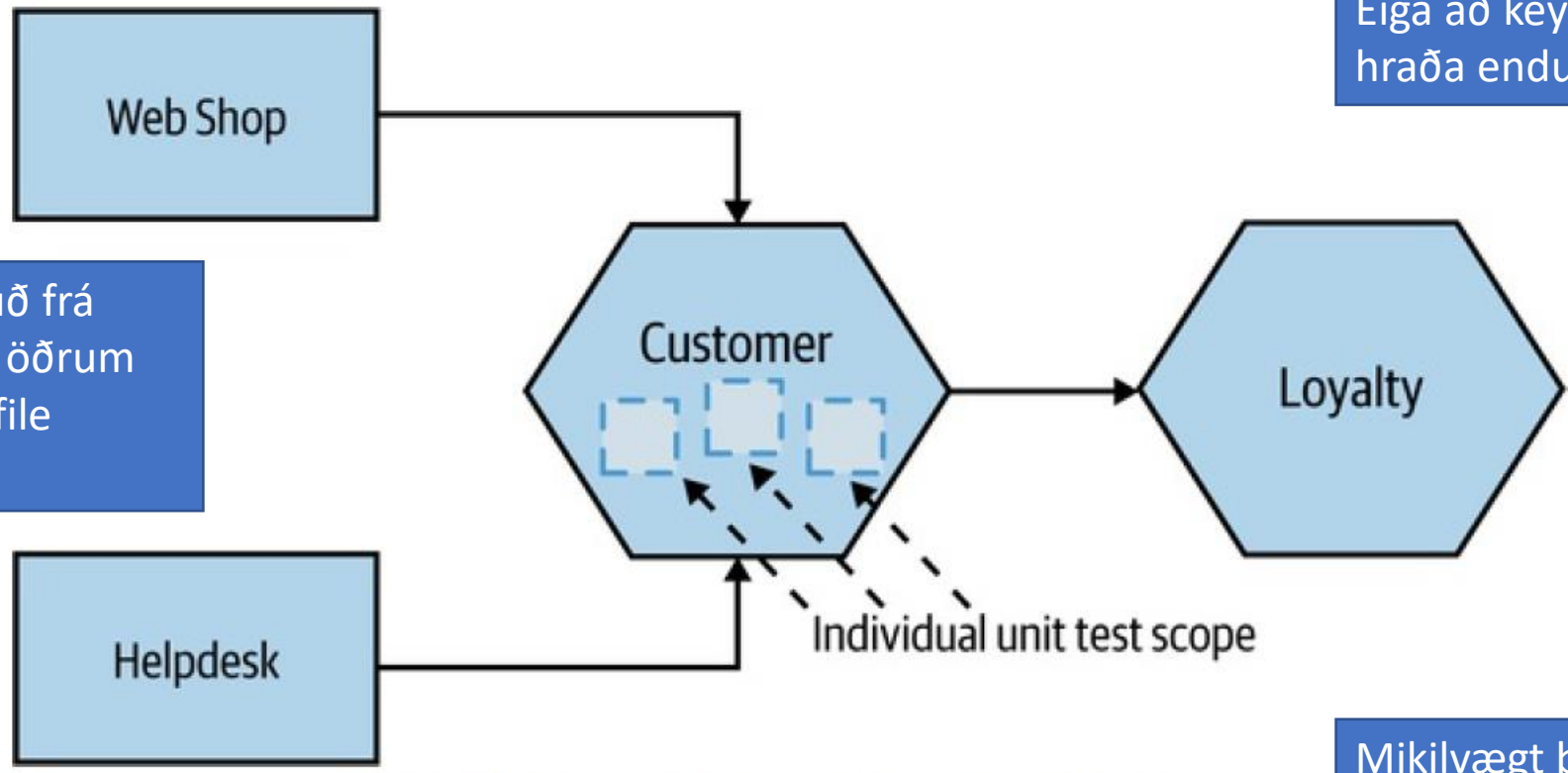


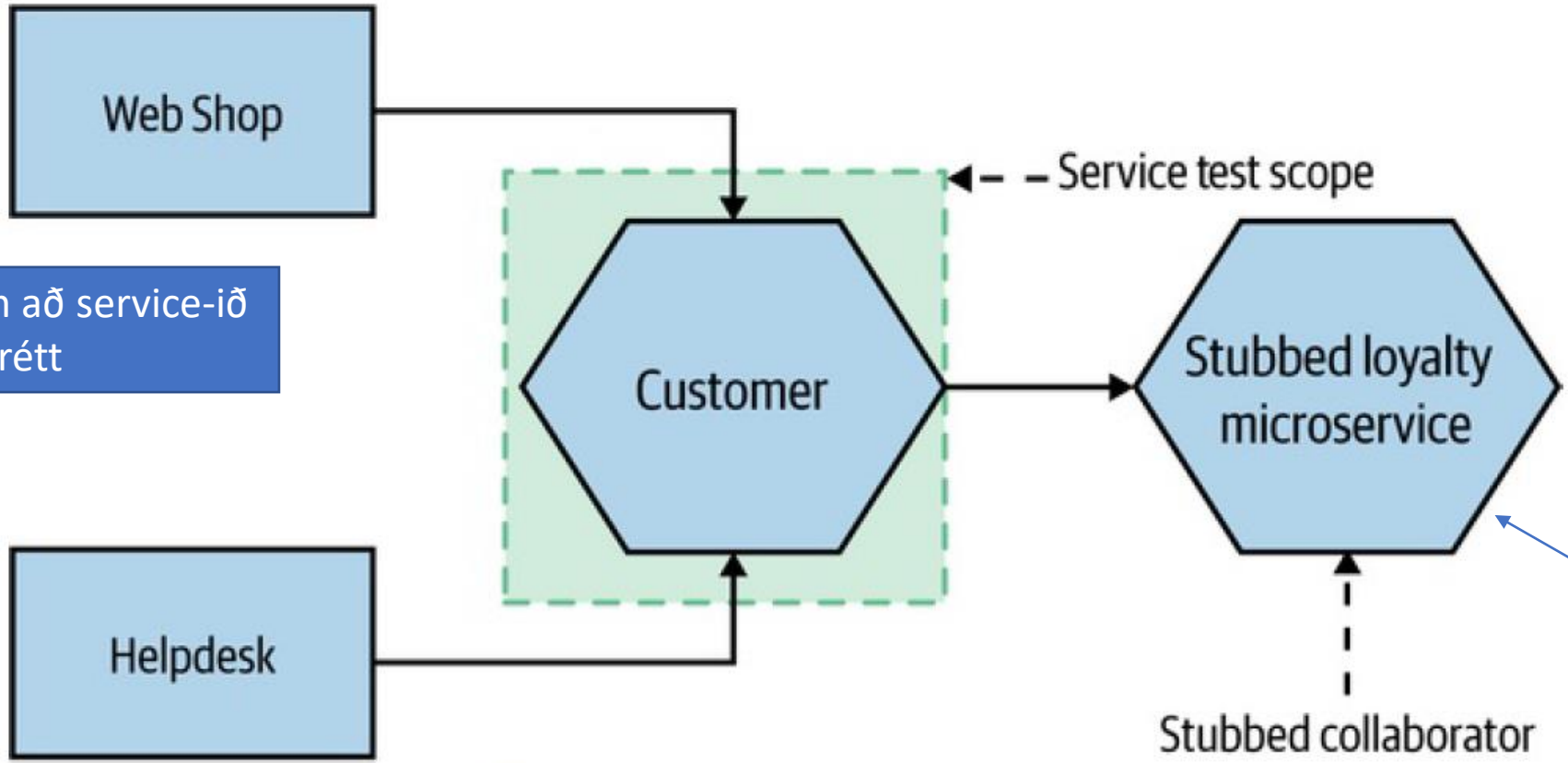
Figure 9-4. Scope of unit tests on our example system

Mikilvægt þegar við erum að refactor-a kóða

Service Tests

Test-a service-in beint

Test scope-ið einangrað að service-inu sjálfu



Aukið öryggi um að service-ið mun hegða sér rétt

Þurfum að stub-a út öll external samskipti

Figure 9-5. Scope of service tests on our example system

End-to-End Tests

Gallinn við þessi test er að það er erfiðara að gera þau, þau er brothættari, tímafrek í keyrslu, erfitt að vita hvað er að brotna og getum bara test-að lítið hlutmengi af virkni *(exponential branching).*

Test keyrð á móti öllu kerfinu

Gott öryggi um að kerfið virki ef þessi test standast

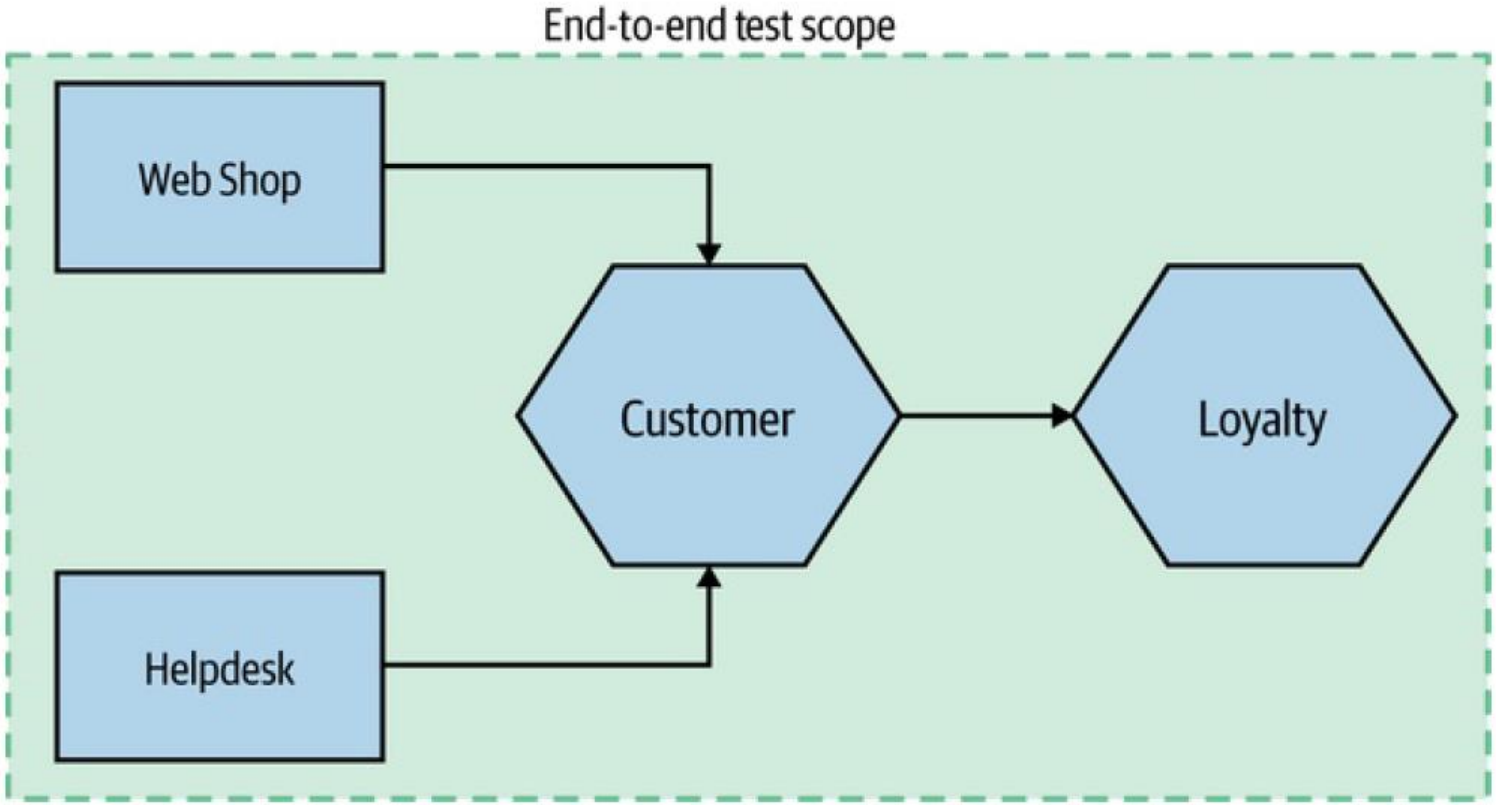


Figure 9-6. Scope of end-to-end tests on our example system

Implementing Service Tests

- Ættu að vera einangruð að service-inu
- Ættu að test-a *slice* af virkni í gegnum allt service-ið
  - T.d. frá endapunkt til gagnagrunns
- Aðferðir
  - Gætum test-að þegar deployed
  - Gætum *deploy-að* locally og test-að
  - Getum keyrt service í minni og testað (t.d. sjá WebApplicationFactory í .NET)
- Ættu ekki að kalla á önnur alvöru service
  - Þurfum að stub-a þau út
  - Getum sjálf skrifað stubs í kóða
  - Getum sjálf deploy-að stubs til að kalla á
  - Getum gert þetta með t.d. pact.io

Implementing (Those Tricky) End-to-End Tests

- Þurfum að hafa mörg service keyrandi til að test-a

- Keyrum end-to-end test-in þegar eitthvað service breytist

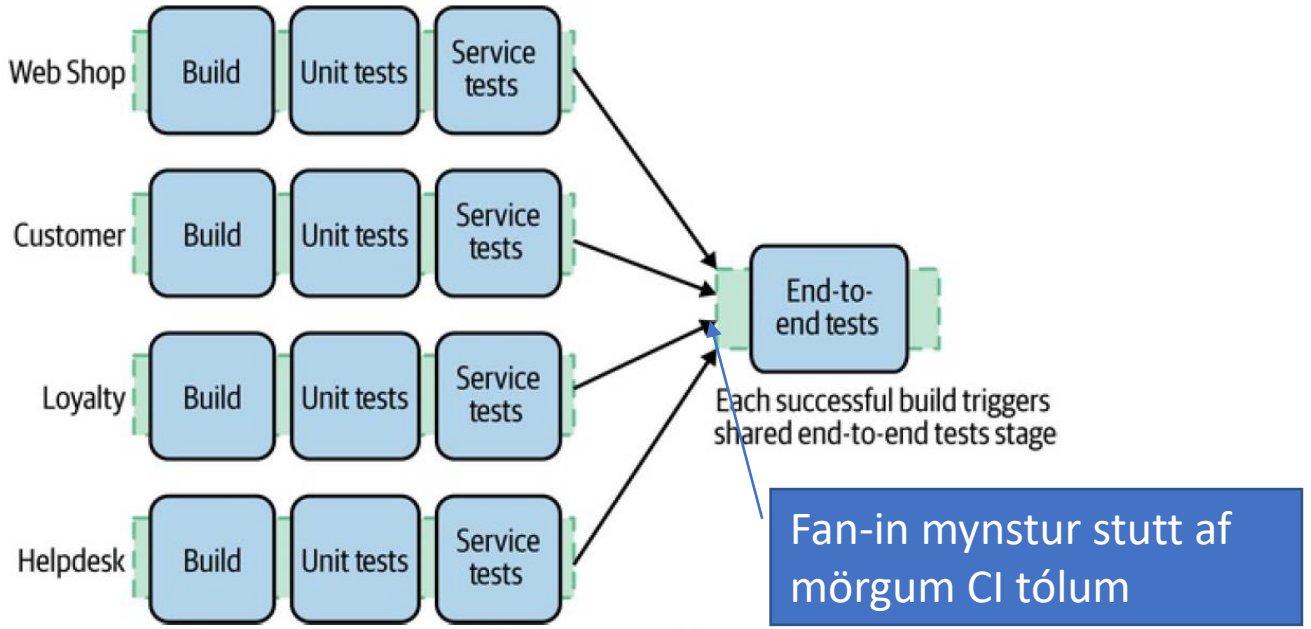


Figure 9-8. A standard way to handle end-to-end tests across services

End-to-End Test vandamál

- Hæg
- Erfið að skrifa
- Erfitt að test-a allt
- Erfitt að vita hvað er brotið
- Þurfum að keyra mörg service
- Eiga til með að vera *flaky* og brothætt
- Óljóst hver eigandi test-ana er
- Getum misst Independent Testability

End-to-End Test vandamál

Flaky & Brittle Tests

- Flaky -> Test fail-a stundum og stundum ekki
- T.d. rof í samskiptum eða eitt service er tímabundið niðri
- Hættulegt -> *the normalization of deviance*
- Test ættu alltaf að fail-a eða alltaf að pass-a
- Viljum gera okkar besta til að laga flaky test
- Ættum í staðinn að skipta þeim út fyrir service / unit test (eða fjarlægja)

End-to-End Test vandamál

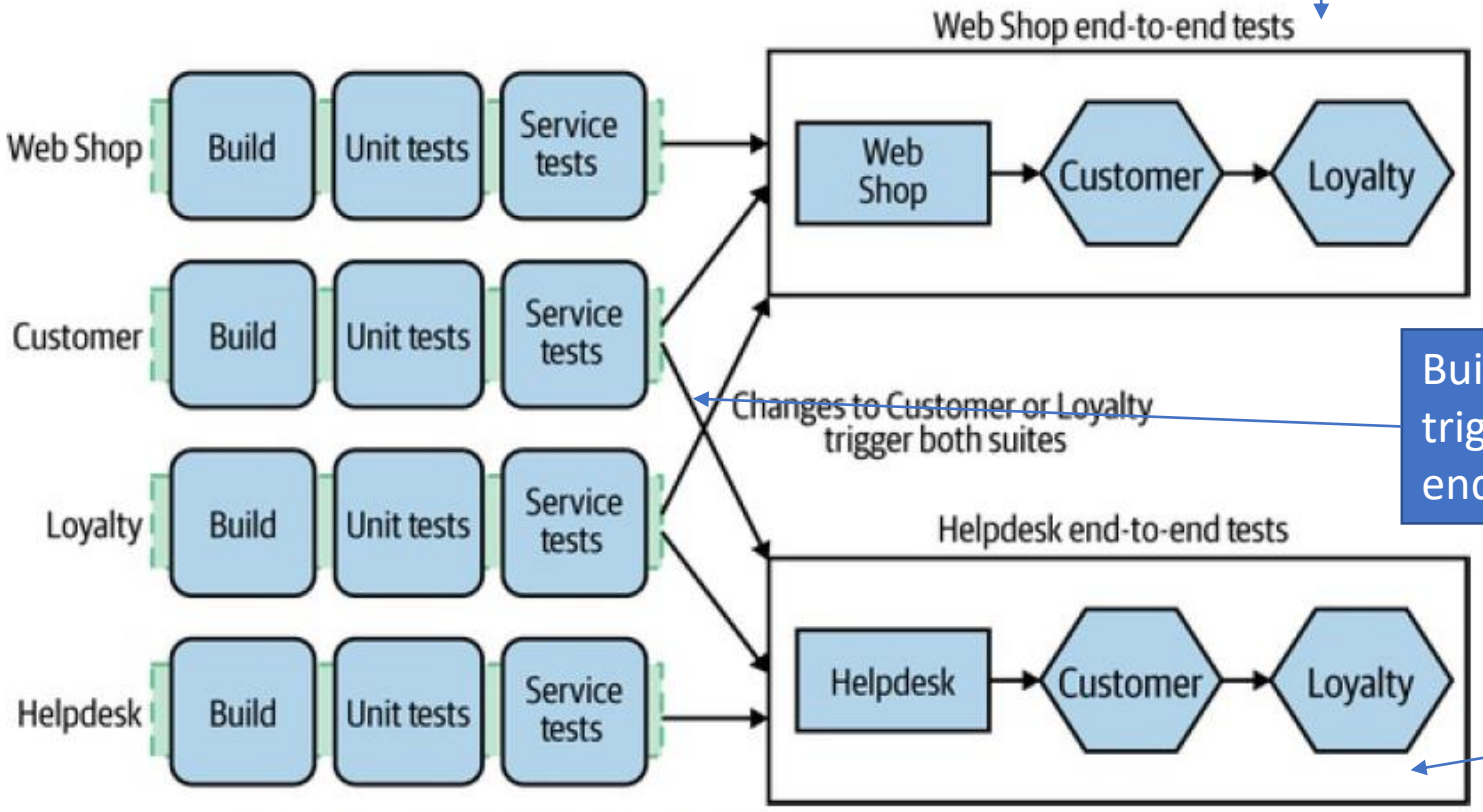
Hver er eigandi test-ana?

- End-to-End Test spanna mörg service
- Service geta verið í eigu mismunandi teyma / aðila
- Óljóst hver ber ábyrgð á þessum test-um
- Getur leitt til þess að engin ber ábyrgð á / er sama ef test fail-a
- Getur verið góð hugmynd að láta E2E test bara test-a innan teymis marka

End-to-End Test vandamál

Hver er eigandi test-ana? Ein Lausn

Ákveðið teymi sem sér um Web Shop end-to-end test



Ef test spanna mörg teymi getum við skráð ábyrgð á teymin

Build á ákveðnum service trigger-a eingöngu ákveðin end-to-end test

Ákveðið teymi sem sér um Helpdesk end-to-end test

Figure 9-9. A standard way to handle end-to-end tests across services

End to End Test – Metaversion Pitfall

- Með E2E er auðvelt að hugsa að ákveðin service virka saman í ákveðnu version
- Þar af leiðandi auðvelt að reyna að version-a og deploy-a öllu kerfinu saman
- Þetta brýtur hugmyndina um independent deployability
- Þetta er **ekki** það sem við viljum gera

Should You Avoid End-to-End Tests?

- Viðráðanleg fyrir lítinn fjölda af services
- Geta verið góð fyrir test á algjöru happy path
- Ættum að fókusa á **lítinn** fjölda af *core journeys*
- Consumer Driven Tests betri leið til að tryggja að breytingar brjóti ekki consumers

Consumer Driven Contracts

- CDCs eru service tests
  - Scope-ið er service-ið sem er verið að test-a
  - Öll dependencies stubbed
- En núna er það consumer-inn sem skilgreinir og á þessi test
- Þ.e.a.s hver consumer skrifar test fyrir producer-inn til að uppfylla væntingar consumer-sins
- Producer service-ið keyrir síðan þessi test í build píplínunni
- Auðvelt að sjá hvaða consumer myndi brotna
- Getum þannig komið í veg fyrir breaking changes án þess að þurfa dýr E2E test
- Tól til að hjálpa við þetta er pact.io

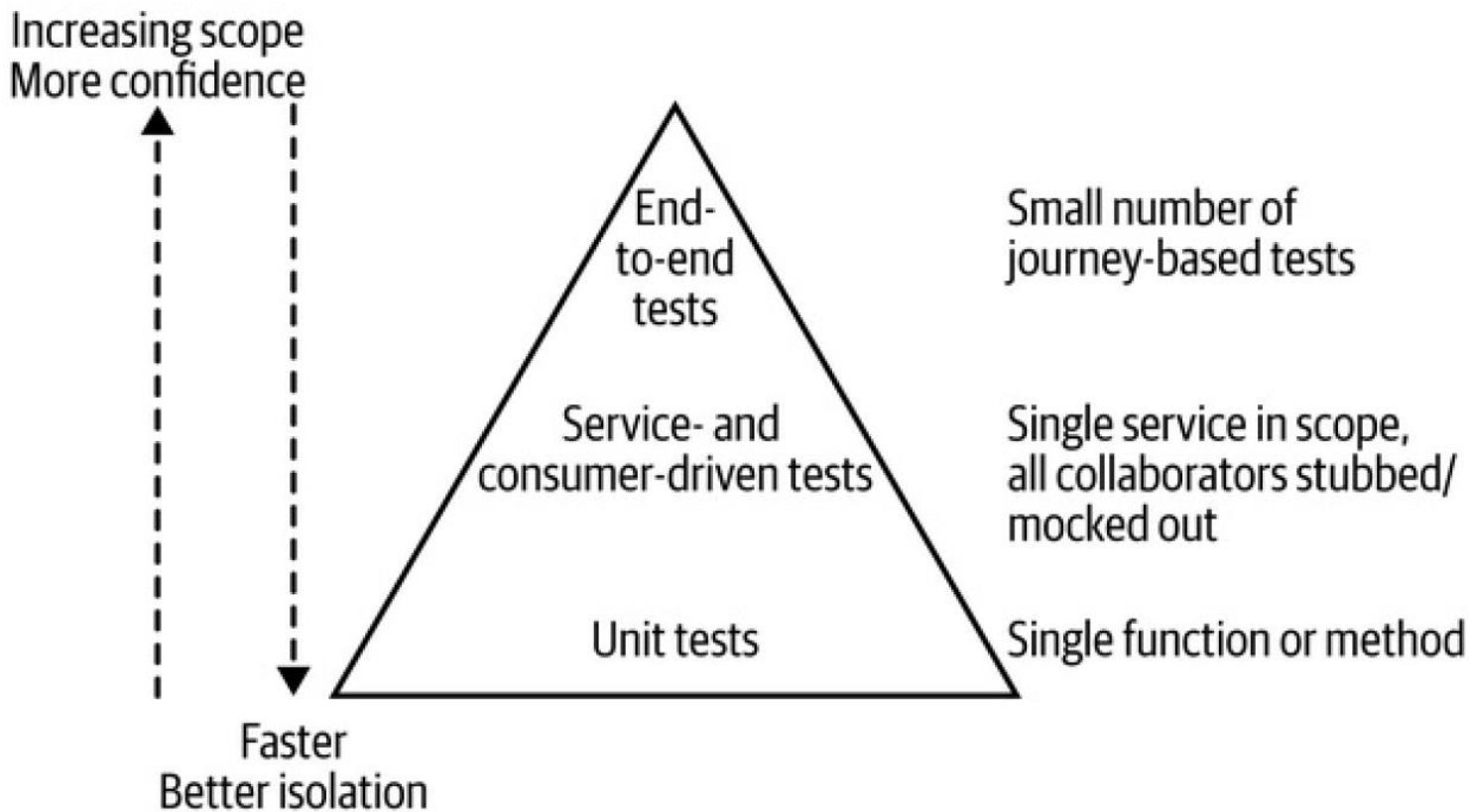
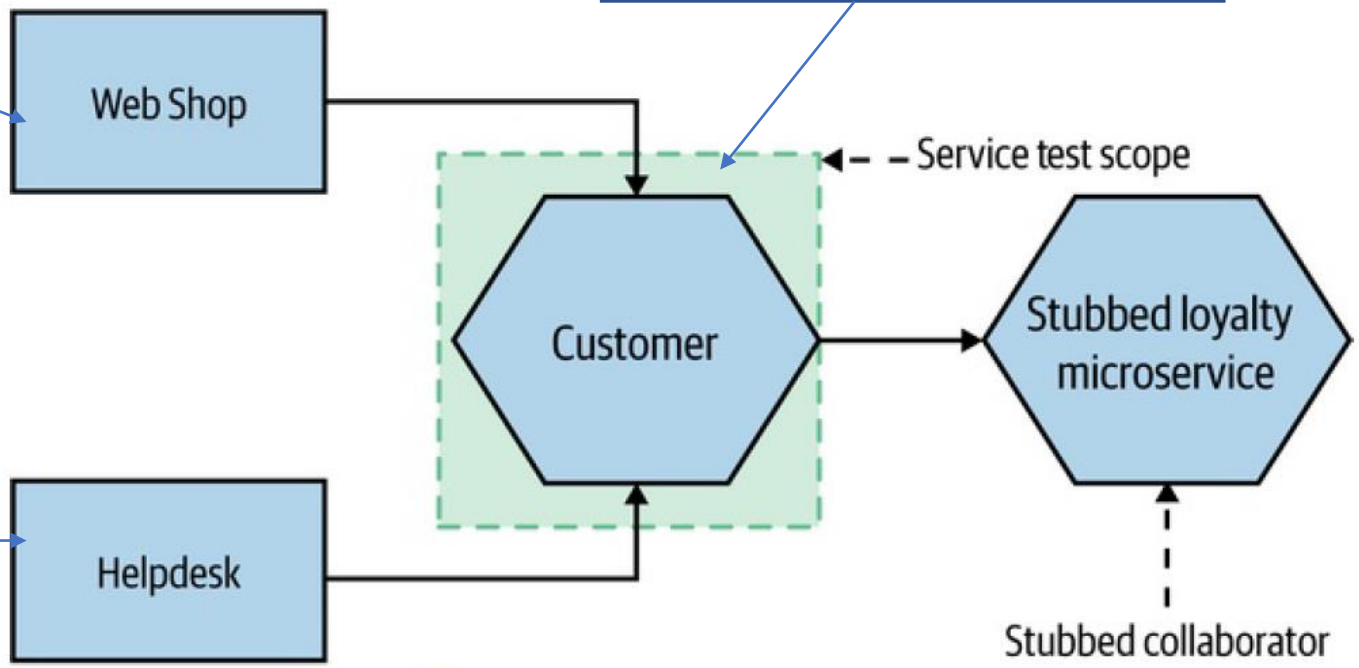


Figure 9-10. Integrating consumer-driven tests into the test pyramid

Consumer Driven Contracts Dæmi

Web Shop skilgreinir sín eigin service test fyrir Customer sem uppfyllir væntingarnar af Web Shop gagnvart Customer service

Customer Service keyrir síðan þessi í CI/CD píplínu



Eins skilgreinir Helpdesk sín eigin service test fyrir Customer Service

Figure 9-5. Scope of service tests on our example system

In-Production Testing

- Sögulega höfum við eingöngu test-að *before* production
- Að testa *before* prod er auðvitað góður hlutur
  - Finnum villur fljótt
  - Finnum villur sem hafa ekki áhrif á notendur
- En á einhverjum punkti fer afrakstur pre-prod test-a minnkandi
- Munum aldrei geta gripið allar villur fyrir prod
- Þar afleiðandi viljum einnig test-a í production

In-Production Testing aðferðir

- Life Check
  - Einfalt ping til að tjékka hvort service instance er keyrandi
  - T.d. tjékka hvort container í k8s er keyrandi

- Smoke Tests
  - Einfalt kall í service-ið
  - T.d. einhvers skonar life check endapunktur
  - Tjékka hvort service-ið er keyrandi **og starfandi**

- Separate Deployment from Release
  - Canary Deployment
  - Blue-green deployment

- Injecting fake user behaviour

In-Production Testing
Separating Deployment from Release

- Gott að aðskilja deployment frá release
- Deployment er þegar kóði er settur upp í ákveðnu umhverfi (t.d. prod)
- Release er þegar ákveðið deployment er gert opinbert fyrir notendur
- Getum þannig test-að og fundið villur í prod umhverfi án þess að hafa áhrif á notendur

Separating Deployment from Release
Blue-Green deployments

- Einfalt deployment strategy
- Með Blue-Green aðskiljum við deployment frá release
- Hugmyndin
  - Ert með eitt service *version* keyrandi (blue)
  - Deploy-ar síðan öðru version-i (green) keyrandi samhlið gamla version-inu (blue) en því er ekki slept til notendur
  - Getur þá test-að *green* version-ið í einangrun á prod umhverfinu
  - Ef test fail-a þá er hægt að rollback-a og notandinn er engu nær
  - Ef test standast er hægt að skipta *blue* version-inu með *green* version-inu

Separating Deployment from Release

Blue-Green deployments

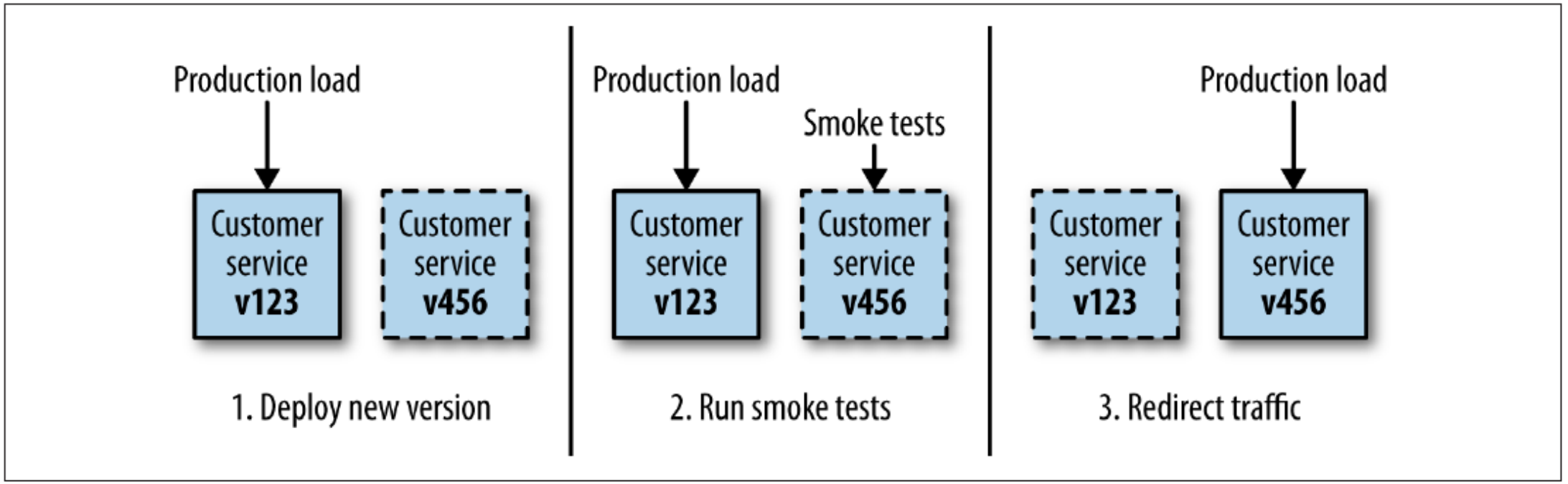


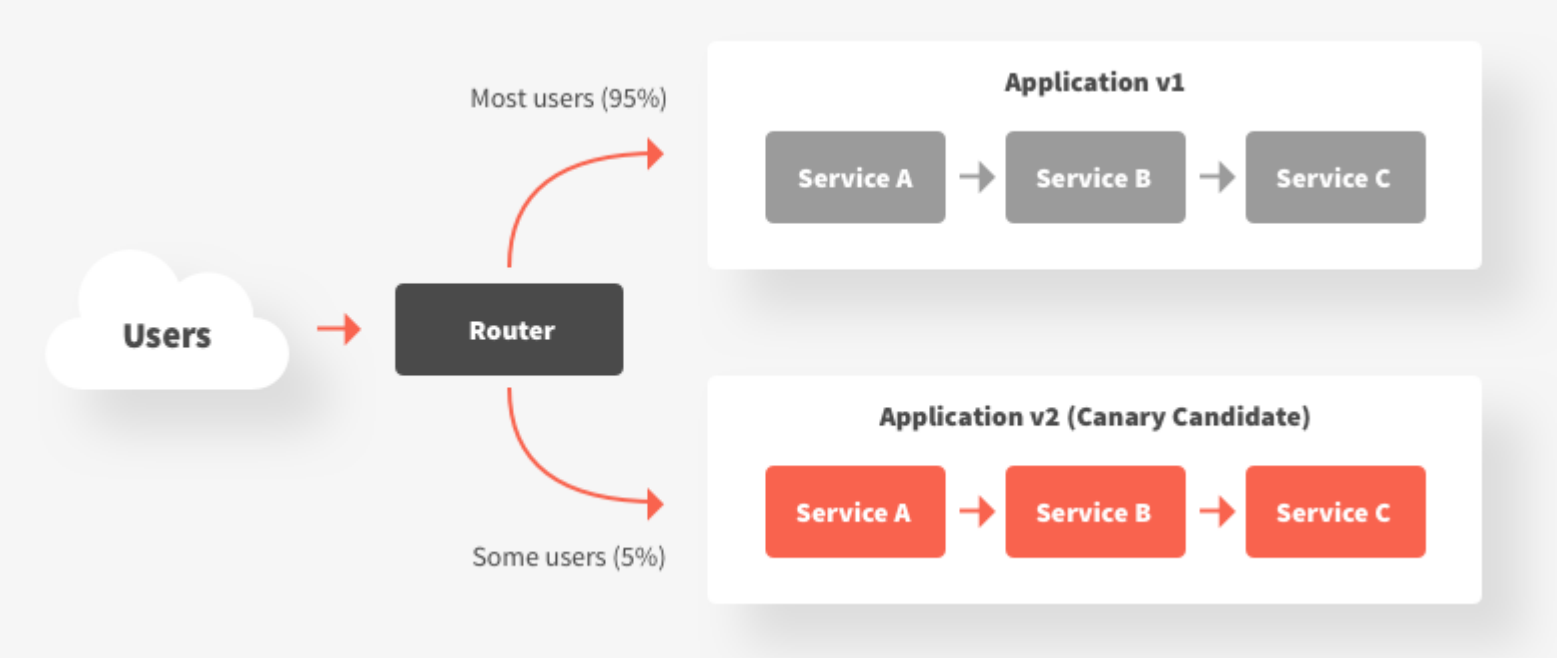
Figure 7-12. Using blue/green deployments to separate deployment from release

Separating Deployment from Release
Canary Releases

- Önnur deployment strategy
- Virkar svipað og Blue/Green nema við hægt og rólega færum raun trafík yfir á nýja version-ið
- Bara hlutmengi af öllum notandum sjá nýju virknina (til að byrja með)
- Minnkum þannig damage-ið ef villa finnst hjá raun notendum
- Monitor-um / test-um í hvert skipti sem við aukum trafík
- Ef test fail-a þá færum við trafík aftur yfir á gamla version
- Hægt að automate-a
  - T.d. með því að test-a hlutfall af villum
  - T.d. með því að test-a fjöldi salna í sölu kerfi

Separating Deployment from Release

Canary Releases



Mean Time to Repair vs Mean Time Between Failures

- En á einhverjum punkti fer afrakstur test-a minnkandi
- Gæti verið sniðugra að optimize-a frekar lagfæringar á villum
  - Aukið monitoring
  - Hröð rollbacks
  - Notifications á villum
- Þ.e.a.s Gæti verið **hagstæðara að vera fljót að laga villur** frekar en að leggja tíma í test **til að minnka tíma á milli villna**
- Þ.e.a.s Mean Time to Repair > Mean Time Between Failures

Cross-Functional Testing

- Getur einnig verið mikilvægt að test-a non-functional requirements
- T.d. Performance, robustness, security testing
- Cross Functional Tests ættu að elta test píramídan líka
- Getur verið að test-a fyrir utan production
  - Þurfum production líkt umhverfi
  - með production líkum gögnum
  - Með production líku gagna magni

Performance Testing

- Mögulega mikilvægasta cross-functional test-ið
- Performance getur verið vandamál í microservice architecture
  - Network köll taka tíma
  - Langar synchronous keðjur geta verið vandamál
- Getum performance test-að á service level-i og end-to-end level-i
- Geta tekið langan tíma að keyra
  - Stundum óraunhæft að keyra við hvert build
  - Stundum frekar keyrt subset á hverjum degi / viku
- Kresft prod líkt umhverfi