ThoughtWorks®

基于约束函数的架构设计和治理改进

夏思雨
ThoughtWorks 资深架构师

目录

01 传统架构治理的典型场景

02 理想的架构治理方式

03 我们做了什么

04 在做以及不知道能不能做出来的

05 结束语

ThoughtWorks®

01

传统架构治理的典型场景

ThoughtWorks®

传统架构治理的典型场景

“我有一个朋友系列”

ThoughtWorks®

痛点

- 整个系统目前体现出外部依赖与技术选型的变化高于业务变化的特征，服务拆分偏向于技术职责而非业务知识边界
- 实现新的需求时需要散弹式修改，几乎贯穿所有服务分层
- 实现新的 feature 或适配新的外部依赖变化时，由于内部 service 间耦合性高，需要跨多个团队的人协同参与，降低了开发效率
- 不知道如何设计微服务的拆分同时兼容团队结构，交付效率与性能

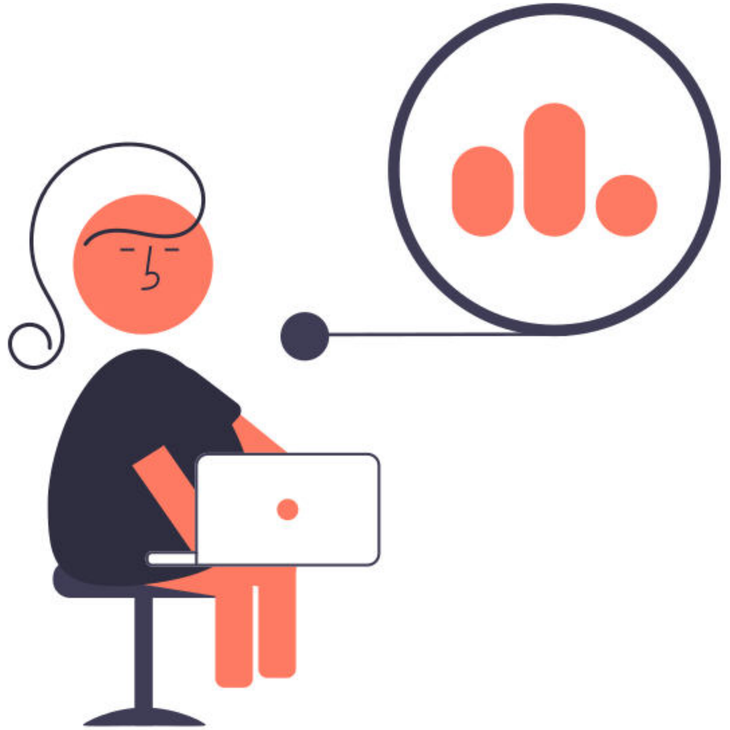
传统架构治理的典型场景

“我有一个朋友系列”

ThoughtWorks®

根因分析

- 架构（微服务）和团队没有基于业务进行端到端清晰的拆分
- 缺少统一认知的架构设计原则和方法
- 整个组织都没有明确区分业务和技术，业务与技术的边界相对模糊
- 团队结构与系统架构不匹配
- 架构缺少可测性设计



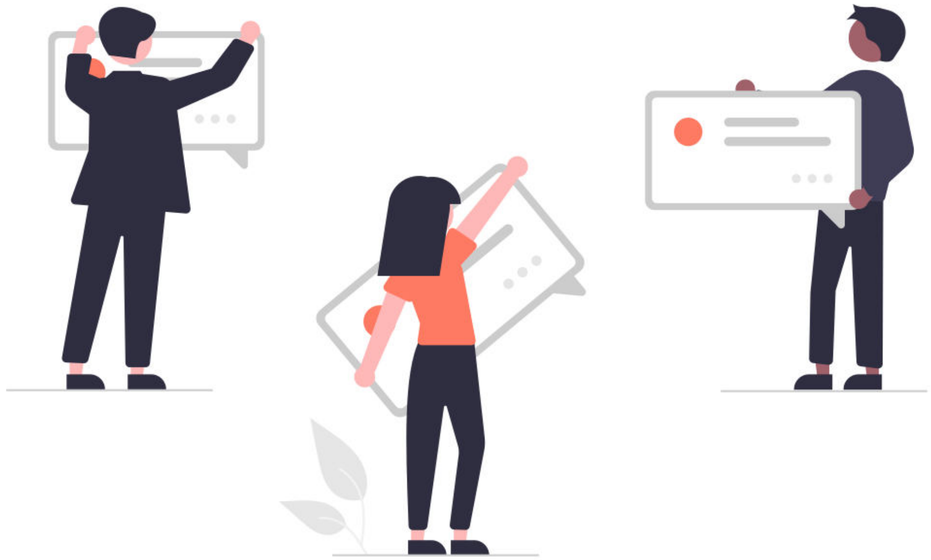
传统架构治理的典型场景

“我有一个朋友系列”

ThoughtWorks®

治理过程与措施

- 资深架构师坐诊
- 重新审视架构愿景和更大范围的业务模式和场景，整体规划未来可演进的架构规划，规范和相应落地方法
- 架构现状分析，基于C4的架构可视化
- 基于领域驱动的核心思想，采用事件风暴和命令风暴对业务进行建模，借助模型驱动架构设计过程，最终得到能在明确边界下工作的模型和边界
- DRY RUN新的业务需求进行验证



传统架构治理的典型场景

“我有一个朋友系列”

ThoughtWorks®

站在“上帝”视角，这个典型的架构治理过程中存在的最大问题是什么？

强依赖于资深架构师的个人经验和能力	→	**整个治理过程成功失败的决定性因素**
缺少统一认知的架构设计原则和方法	→	**在“决定性因素”的带领下暂时统一了，然后呢？**
团队结构与系统架构不匹配	→	**并没有实际解决**
架构缺少可测性设计	→	**并没有实际解决**
边界划分	→	**目测是解决了，但是怎么证明是正确的呢**

传统架构治理的典型场景

反思整个过程，我们能得到什么结论？

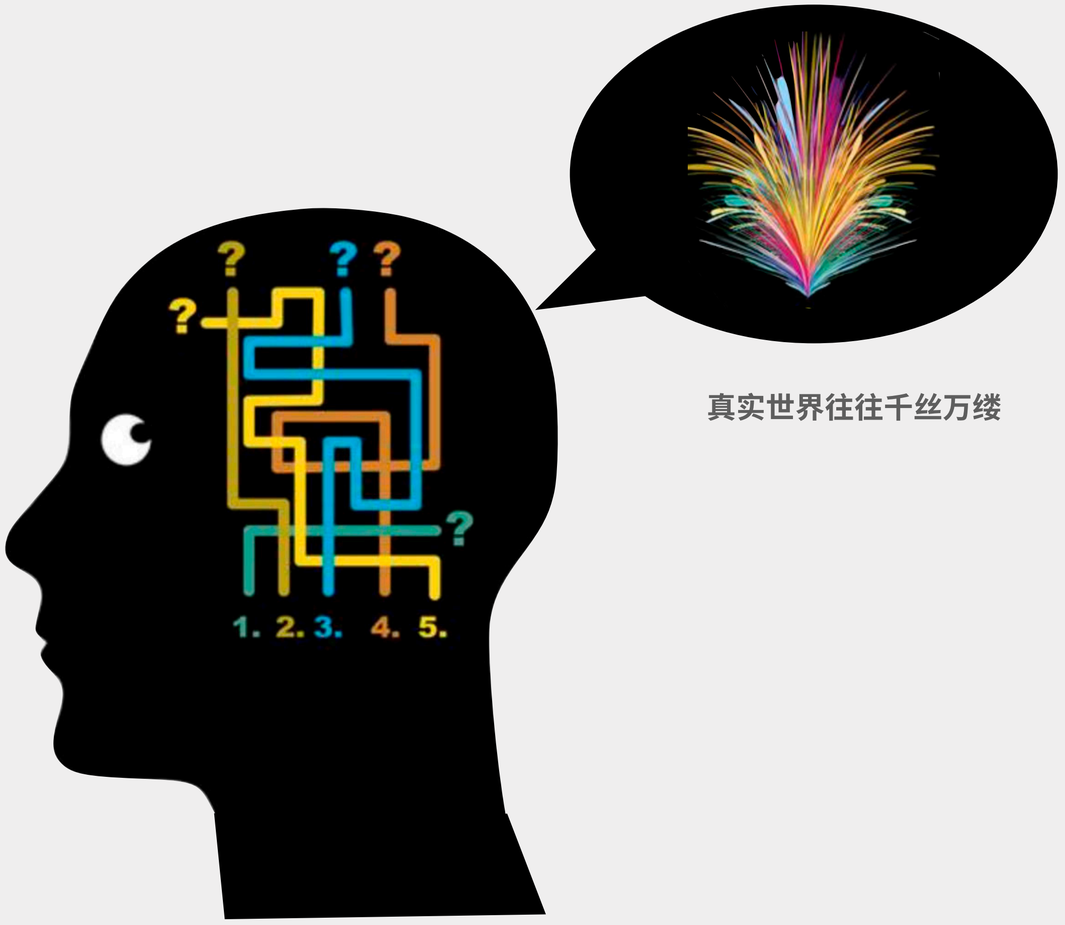
ThoughtWorks®

人的认知天性带来的问题

- 数十年的认知科学表明，人脑天生倾向于线性思考方式，从人脑的存储以及运算能力来说，不适合真实世界的复杂网状结构。

架构问题的复杂

- 软件系统之所以复杂在于不能摒弃众多因素，做简单抽象。
- 高内聚低耦合不是人做出来的，而是天生就存在那里。真实业务中涉及到的对象以及其关联关系都是定好的，我们要做的事情是去“寻找关系天然就比较少的地方作为模块的边界划分”。 模块划分错了，硬搞是没法搞出什么高内聚、低耦合的，除了让代码变得更恶心，没任何效果。



真实世界往往千丝万缕

人类倾向线性认知

02

理想的架构治理方式

ThoughtWorks®

理想的架构治理方式

看看别的行业，怎么做?

ThoughtWorks®

1975年，“BIM”之父美国乔治亚理工大学的Chuck Eastman提出“Building Description System（建筑描述系统）”，以便于实现建筑工程的可视化和可量化分析，提高工程建设效率。

1999年， Eastman将建筑描述系统发展为“建筑产品模型”，认为建筑产品模型从概念，设计施工到拆除的全生命周期过程中，均可提供建筑产品丰富，整合的信息。

BIM核心解决的是： “工程效率”

BIM的特点

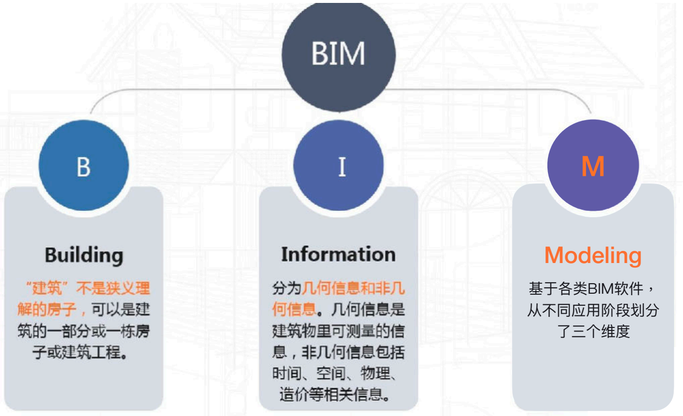
01 可视化

02 协调性

03 模拟性

04 优化性

05 可出图性



理想的架构治理方式

我们希望软件架构设计和演进能做到什么？

ThoughtWorks®

· 可靠的可视化 ·

- 系统架构复杂度越来越高，架构变化日益频繁，微服务改造后的实际架构模型可能与预期已经产生了巨大差异，架构师或系统运维人员很难准确记忆所有资源实例的构成和交互情况；

· 可追溯 ·

- 系统架构在动态演化过程中可能引入了一些不可靠的因素，比如弱依赖变强依赖、局部容量不足、系统耦合过重等，给系统的稳定性带了极大的安全隐患。
- 有可靠的可视化架构表现，才能通过不同阶段的架构变化识别架构腐坏的过程和变更引入的时间点。

· 可量化评估 ·

- 量化最大的作用有两个，第一，清楚地知道对架构的目标是什么。 第二，更有效率地达成目标
- 架构的量化评估，要考虑清楚地是，不同场景的诉求下，我们对架构多种特性的要求是不一样的，哪一种或者哪几种性质决定了这个架构能否走得更长远

· 可模拟 ·

- 上述都是架构设计和演进中的后验手段，只有采取的措施真实发生了，我们才会看到相应的结果。一旦出现失误，即使能够及时挽回，过程中人力时间金钱的浪费都无法避免。
- 在新的需求提出或者业务变更时，能够在可视化的真实架构基础之上，模拟业务变更的实现以及评估对架构造成的影响，能一定程度上避免不必要的浪费。

可访问性	可问责性	准确性	适应性	可管理性
可负担性	敏捷性	可审计性	自治性	可用性
兼容性	可组合性	可配置性	正确性	可信度
可定制性	可调试性	可降级性	可确定性	可演示性
可信赖性	可部署性	可发现性	分布性	耐久性
有效性	高效性	可用性	可扩展性	故障透明性
容错性	保真性	灵活性	可检查性	可安装性
完整性	互通性	可学习性	可维护性	可管理性
移动性	可修改性	模块性	可操作性	正交性
可移植性	精确性	可预测性	过程能力	可生产性
可证性	可恢复性	相关性	可靠性	可重复性
重现性	弹性	响应性	可复用性	稳健性
安全	伸缩性	无缝性	自我维持性	可服务性
安全性	简单性	稳定性	标准合规性	可生存性
可持续性	可裁剪性	可测试性	及时性	可追溯性

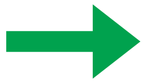
理想的架构治理方式

发展规律

ThoughtWorks®

自发阶段

- 早期更多的是靠应用驱动，从经验中积累知识，主要特征表现为较大的随意性和不确定性。
- 即使在一个领域获得成功，那些经验通常也很难推广到另一个领域
- 从结果上看，很多成功的案例取决于个别人



自觉阶段

- 发展到一个阶段，就会拥有自己的理论基础，而那些理论基础，在一开始可能看上去和当前的问题并没有太大关系
- 寻找理论基础的过程中，通常会经历三个过程。
  - 第一，寻找符合和真实世界之间的表意关系；
  - 第二，设计更通俗易懂的符号；
  - 第三，在抽象的符号表现层之上，为信息寻找数学基础。

《信息传》— 吴军

03

我们做了什么

ThoughtWorks®

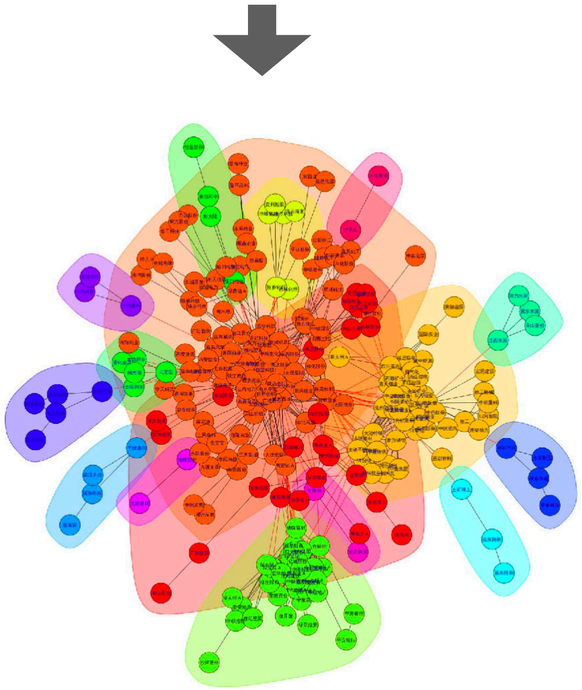
我们做了什么

从第一性原理出发

ThoughtWorks®

系统架构 = （组件 + 组件间关联关系 ） + 运行环境

运行环境： 包括但不限于部署结构，资源配置，业务场景诉求，支撑系统的团队组织结构、技能和文化



“基于复杂网络理论的系统架构”

我们做了什么

问题拆解

ThoughtWorks®

组件以及组件间关联关系

基于复杂网络的架构模型分析

运行环境

基于约束函数的架构治理

我们做了什么

复杂网络理论

ThoughtWorks®

是什么

- 复杂网络的研究由于其学科交叉性和复杂性的特点，涉及了众多学科的知识和理论基础，尤其是系统科学、统计物理、数学、计算机与信息科学等
- 复杂网络的主要研究方法都是基于图论的理论和方法开展，常用的分析方法和工具包括图论、组合数学、矩阵理论、概率论、随机过程、优化理论和遗传算法等
- 与传统图论研究的不同之处在于：复杂网络从统计角度考察网络中节点的分布及节点间连接的性质，根据不同的网络组织结构来研究系统功能差异，为系统（结构。性能等）的优化提供理论基础

应用场景

- 神经系统可以看作大量神经细胞通过神经纤维相互连接形成的网络
- 计算机网络可以看作是自主工作的计算机通过通信介质如光缆、双绞线、同轴电缆等相互连接形成的网络
- 类似的还有电力网络、社会关系网络、交通网络、调度网络等等

图9 朋友关系网

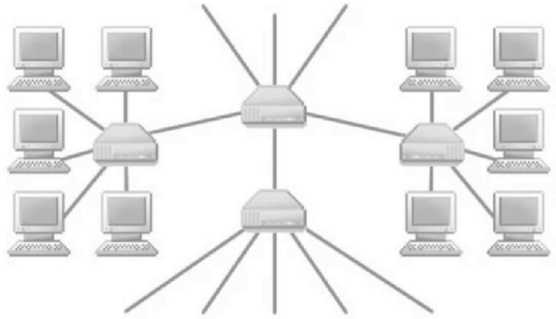


图7 互联网

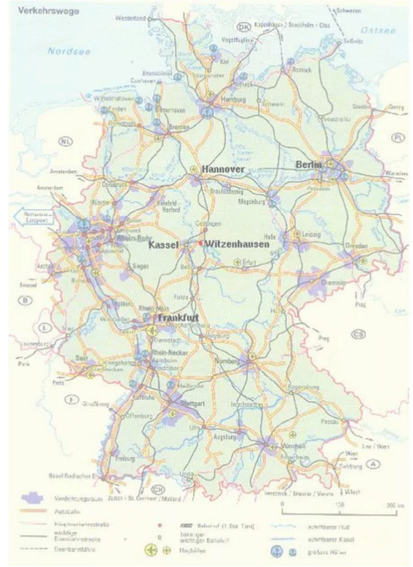


图11 道路交通网

我们做了什么

架构分析机制

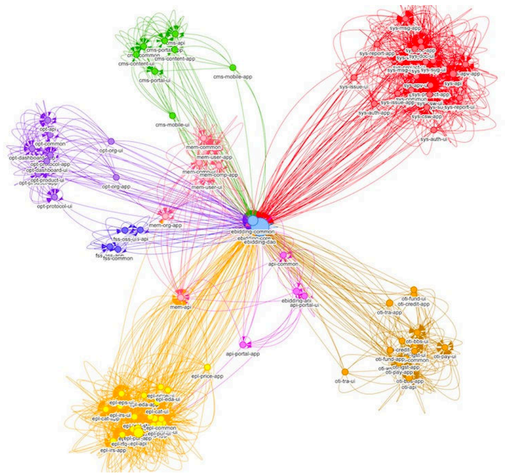
ThoughtWorks®

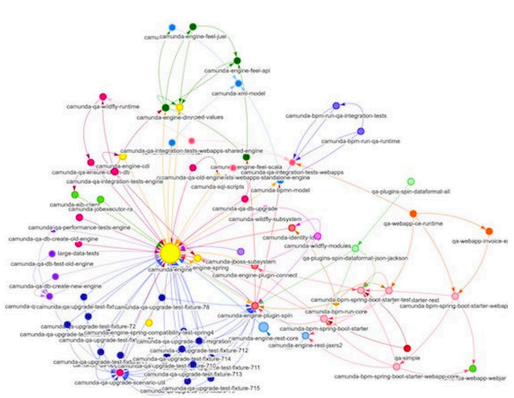
聚类系数	• 网络有多“紧密” • 反映了系统中不同的组件的内聚程度以及系统组织分层（层次化）的趋势
平均路径长度	• 网络的有效“尺寸” • 应用于评估系统消息传递的设计与优化、对象间通信代价的评估与控制、系统响应能力的改进等方面
度数以及度数分布	• 描述网络中节点最重要的参数，而度数分布可以用于定义一个网络的特性 • 度数分布刻画了网络中节点的连通性,重用度和复杂度，也可以用于衡量系统的不平均结构。
介数	• 被“经过”的重要程度 • 介数可以分析系统模型中任意一个组件和组件之间的关系在被删除或失效时对整个系统的影响，提供系统单点瓶颈的量化判断依据，指导改进。
相关性分析	• 度数与聚类系数之间的相关性，度数与度数的相关性 • 聚类系数与平均路径长度的相关性

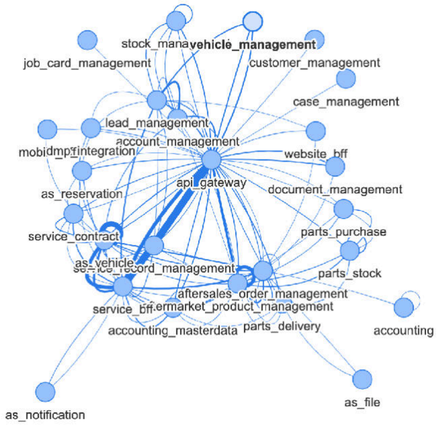
我们做了什么

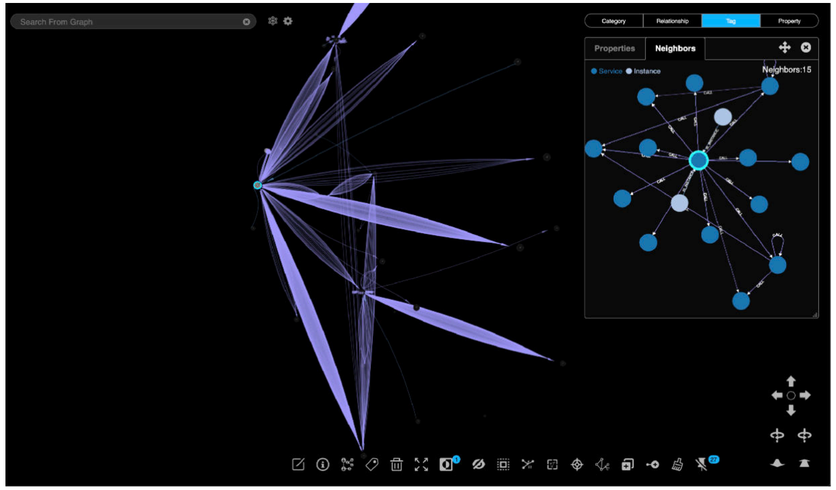
架构分析的案例 — 建模

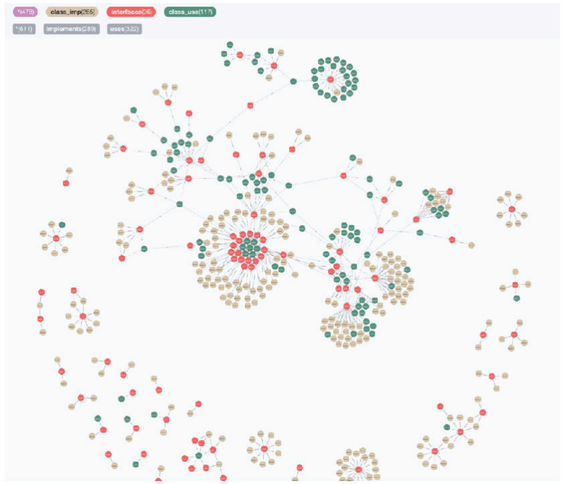
ThoughtWorks®

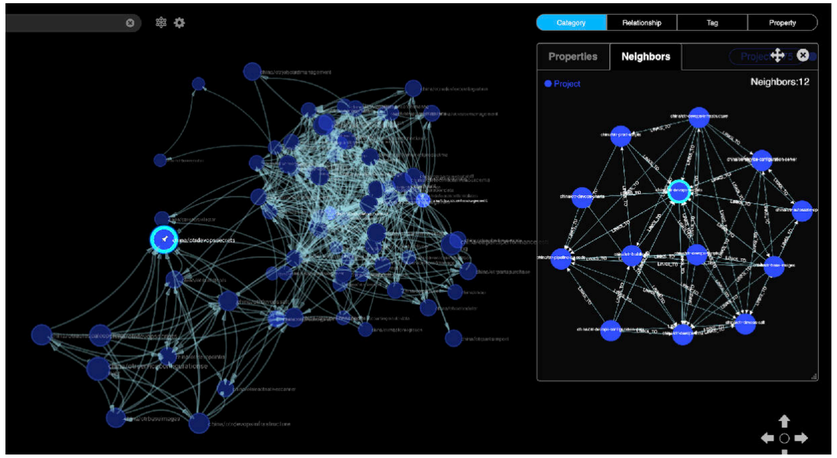












我们做了什么

架构分析的案例 — 数值分析和方向建议

ThoughtWorks®

无标度网络

- 无标度特性反映了复杂网络具有严重的异质性，其各节点之间的连接状况（度数）具有严重的不均匀分布性：从广义上说，无标度网络的无标度性是描述大量复杂系统整体上严重不均匀分布的一种内在性质

- 无标度网络中幂律分布特性的决定了无标度网络的鲁棒性和容错性

小世界网络

- 通过特征路径长度和聚合系统决定小世界网络的特性。特征路径长度小，聚合系数高，是小世界网络的典型特质，同时具有高内聚低耦合特性的网络是小世界网络。

- 小世界网络满足全局效率和局部效率更高，这样的布局具有更好的信息传递效率，以及比较好的健壮性和可维护性。

我们做了什么

解决了组件和组件之间的关系模型，运行环境呢?

ThoughtWorks®

系统架构 = （组件 + 组件间关联关系 ） + 运行环境

运行环境： 包括但不限于部署结构，资源配置，业务场景诉求，支撑系统的团队组织结构、技能和文化

康威定律	设计系统的组织，其产生的设计等同于组织之内、组织之间的沟通结构
康威逆定律	无论系统有什么设计缺陷，都不得不通过改变组织结构来推动系统的更改

我们做了什么

解决了组件和组件之间的关系模型，运行环境呢?

ThoughtWorks®

- 架构约束函数 F（x）= f(团队结构，f(软件架构)， f匹配度(团队结构，软件架构))

我们做了什么

架构治理和改进 - 总结

ThoughtWorks®

1

从系统架构的模型和分析结果出发，探索符合业务诉求的架构方法和治理改进方向

2

从业务诉求，团队结构和能力出发，探索适合的软件架构和架构规范

04

在做以及不知道能不能做出来

ThoughtWorks®

待解决和未来要研究的问题

ThoughtWorks®

1 建模准确度和适用范围

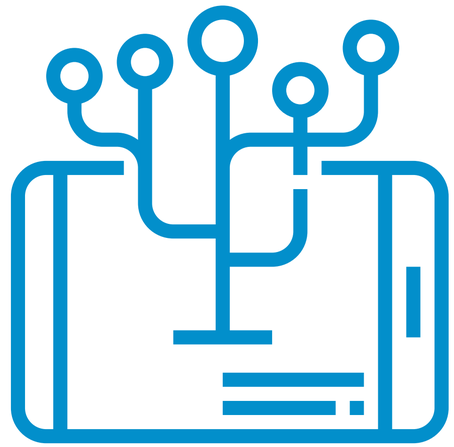
研发过程数据，日常沟通的会议邀请，运行时的调用、日志数据等，几乎是建模的标准数据来源

这些数据涉及了技术选型，工程实践，工作习惯等方方面面

不规范的工程实践，名存实亡的工具使用，大量线下无记录无追踪的活动，会造成非常大难度和工作量的数据采集和清洗

数据的缺失和无效意味着“无米下锅”

一定存在一些架构模式或者系统，不适用网络分析的方式，有哪些？为什么？



待解决和未来要研究的问题

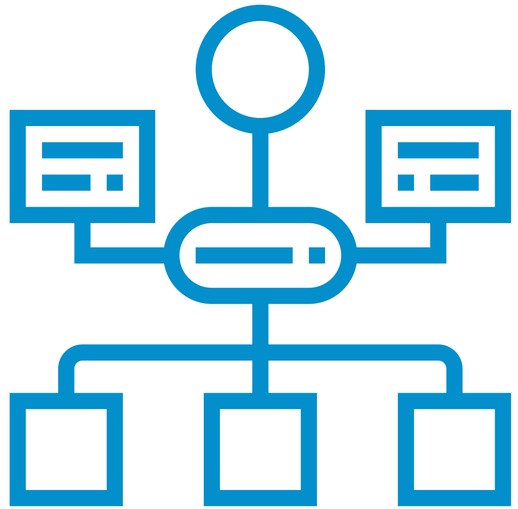
ThoughtWorks®

2 架构模拟

当业务提出新的需求的时候，需要以什么样的形式和内容，才能够在现有架构模型的基础之上完成DRY RUN，回答是否能实现以及需要多久才能做完的问题

如何能够得知一个需求的实现，需要涉及到多少个组建的修改，这些组件由哪些团队负责，怎么样的协作才能尽快交付

对应不同的非功能属性，比如吞吐量，比如响应时长，如何在真实架构的建模之上根据理论计算给出相对可靠的结果



待解决和未来要研究的问题

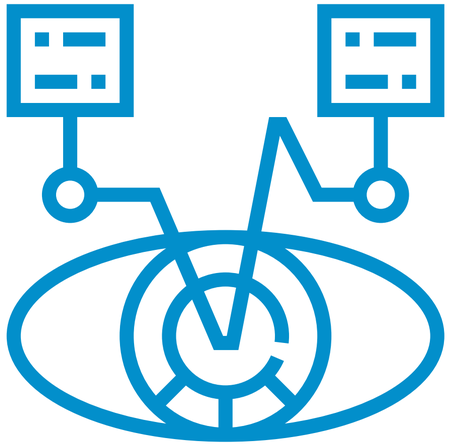
ThoughtWorks®

3 智能预测

即使有逻辑上说得过去的模型可以参考，对业务的正确和深刻的理解，依然是整个系统能否成功的关键

当架构模型由于需求发生变更时，是否能够给出潜在的风险所在和建议

是否能够在指定架构目标模型的前提下，自动计算和给出架构演进的变更路线



05

结束语

ThoughtWorks®

“软件行业没有银弹”

欲了解详情，请发送邮件至：
syxia@thoughtworks.com

ThoughtWorks®

ThoughtWorks®

从低代码到前端智能化

邓奕

ThoughtWorks 高级咨询师

目录

01 低代码的热议

02 前端的低代码实践

03 对前端智能化的探索

04 结语

ThoughtWorks®

01

低代码的热议

ThoughtWorks®

低代码的热议

ThoughtWorks®

1

Gartner 预测：到 2025 年，70% 的新应用将由低代码 / 无代码技术完成开发

低代码开发技术市场收入（单位：百万美元）

	2019年	2020年	2021年
低代码应用平台 (LCAP)	3473.5	4448.2	5751.6
智能业务流程管理套件	2509.7	2694.9	2891.6
多重体验开发平台（MDXP）	1583.5	1931.0	2326.9
机器人流程自动化（RPA）	1184.5	1686.0	2187.4
平民自动化和开发平台（CADP）	341.8	438.7	579.5
其他低代码开发（LCD）技术	59.6	73.4	87.3
合计	9152.6	11272.2	13824.2

低代码的热议

ThoughtWorks®

2

技术雷达对低代码平台的思考

什么是低代码？

维基百科定义：低代码开发平台（LCDP）本身也是一种软件，它为开发者提供了一个创建应用软件的开发环境；与传统编写代码的 IDE 不同，低代码开发平台提供更易用的可视化 IDE。

2014 年，Forrester Research 研究机构正式提出了低代码的定义，即利用很少或几乎不需要写代码就可以快速开发应用，并可以快速配置和部署的一种技术和工具。

技术雷达的思考

现在很多公司正在面临的一个最微妙的决定，便是是否要采纳低代码平台或无代码平台， 这些平台可以被用来在非常特定的领域里解决一些特定的问题。限界低代码平台这一领域的供应商也有如过江之鲫。现在看来，这类平台的一个突出的问题，便是很难应用一些诸如版本控制之类的优秀的工程实践。而且这类平台上的测试也非常的困难。

02

前端的低代码实践

ThoughtWorks®

前端的低代码实践

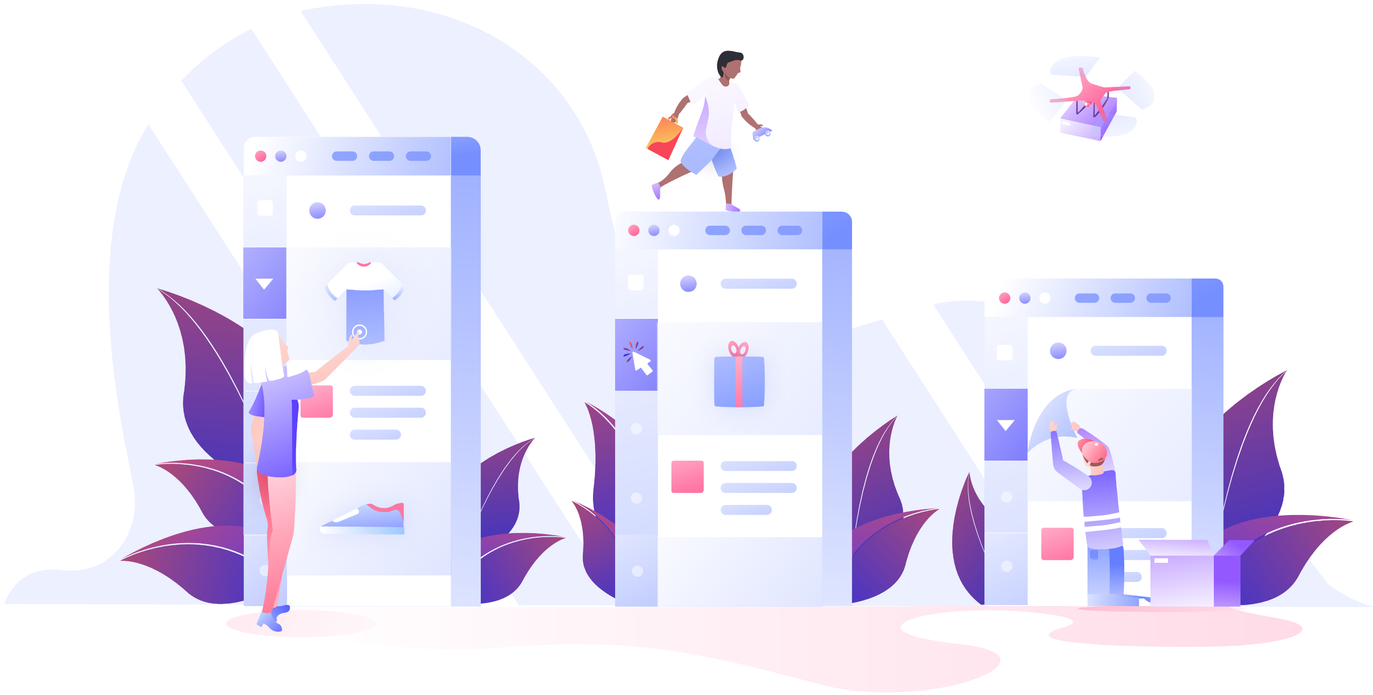
ThoughtWorks®

1

营销场景是给前端开发带来挑战的一大业务来源

各种活动类型：

- 促活
- 召回
- 宣传新品
- 回馈用户
- 节日
- ...



前端的低代码实践

ThoughtWorks®

2

概括的前端架构需求

前端的低代码实践

ThoughtWorks®

3

营销场景给前端带来的挑战

case by case进行
活动编码开发，复用性不高

短周期冲刺式上线，
开发、联调、测试时间紧，
但任务重

不同需求独立开发，
不成体系、难以规范

前端的低代码实践

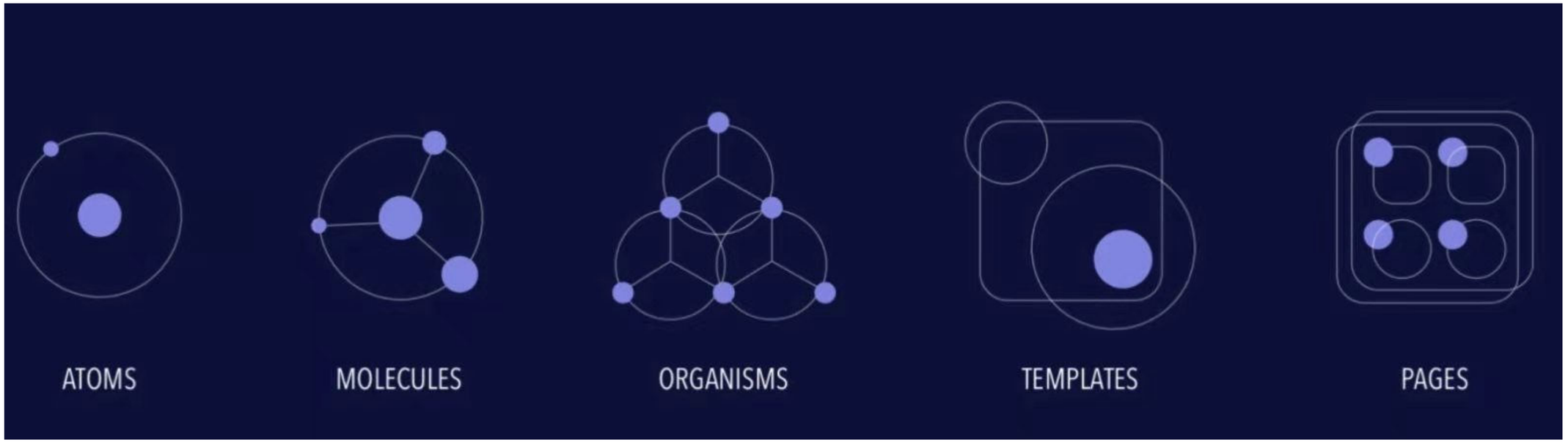
ThoughtWorks®

4

组件化

很多公司开始有了自己的“搭积木”探索。

在营销场景中，前端页面的原子组合，尤其常见。



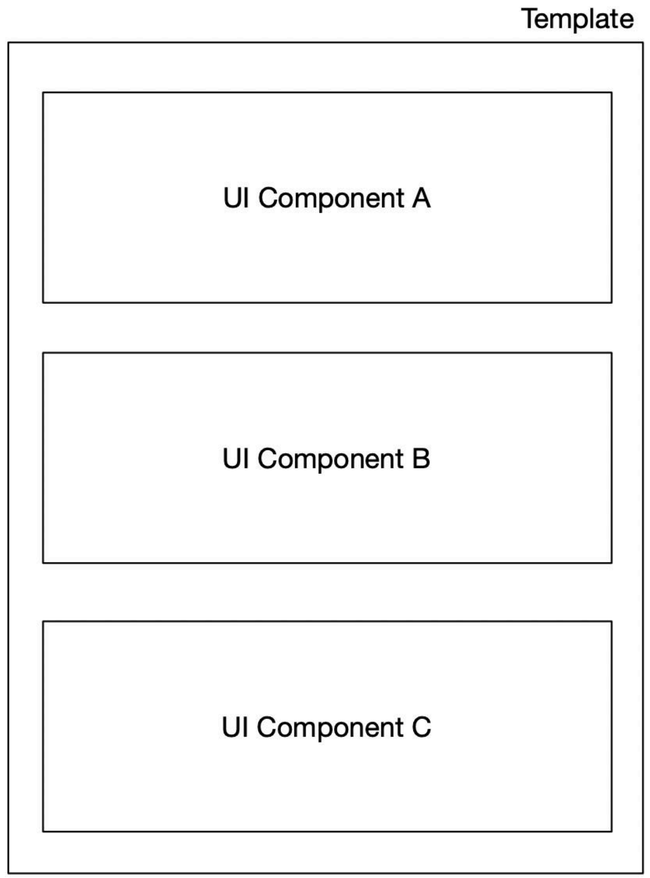
前端的低代码实践

ThoughtWorks®

5

模板化

前端在很少或几乎不需要写代码就可以快速开发这一思路下，自然而然可以想到模板化。



前端的低代码实践

ThoughtWorks®

“搭积木”的挑战

需求兼容性

- 能快速定制的前提：承接的需求在原先配置的支持范围之内

成本

- 组件识别、封装需要经验和投入
- 版本管理的技术难度

03

对前端智能化的探索

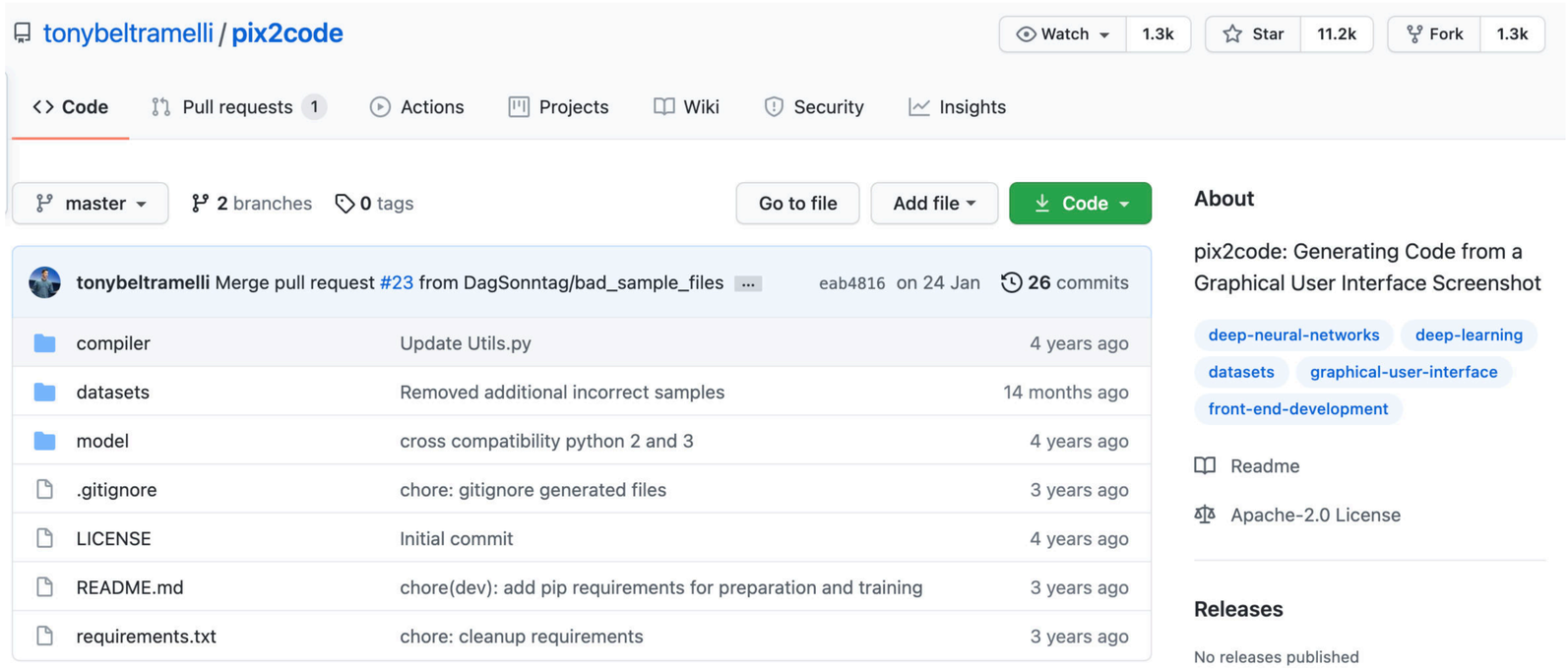
ThoughtWorks®

对前端智能化的探索

ThoughtWorks®

1

不如让设计稿（UI视图）自动生成代码?



pix2code

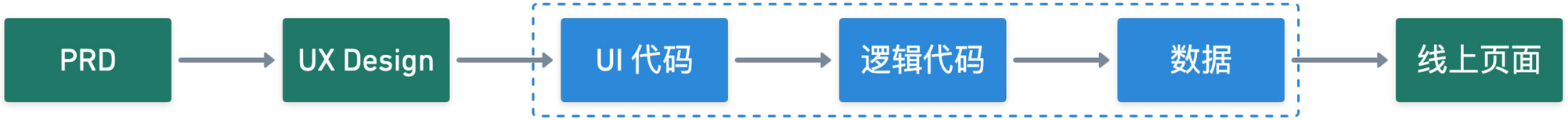
通过图形UI的截图自动生成代码

对前端智能化的探索

ThoughtWorks®

2

imgcook介绍



对前端智能化的探索

ThoughtWorks®

4

imgcook试用效果

对工具本身的感受

算法准确度

- 支持标较复杂组件，但不能识别我们样本中的循环

代码可读性

- 工具生成的代码目前已经支持多端、多框架，可读性中等水平
- 不过亮点在于支持自定义DSL

对于前端开发流程的覆盖

- 逻辑和数据绑定支持可视化操作
- 提供CLI，VSCode Plugin等，有不错的工具链和工程支撑

支付宝小程序开发规范

Android 开发规范

Flutter 开发规范

毫末Flutter组件开发规范

Taro 开发规范

Rax 开发规范（非 Hooks 版）

Vue 开发规范

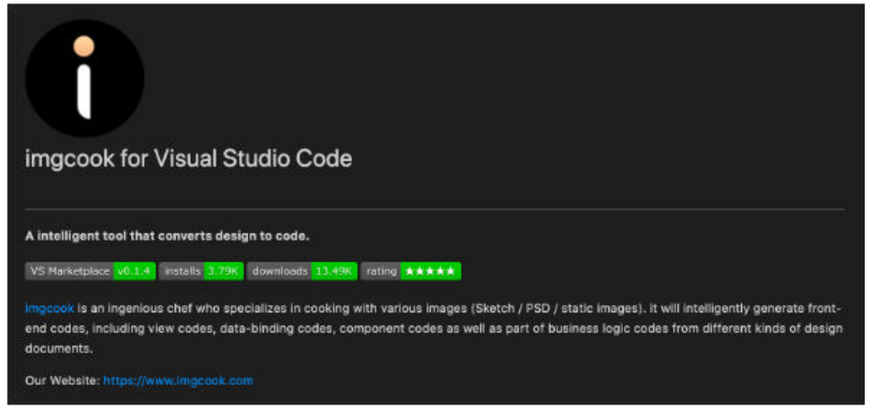
React 开发规范

微信小程序开发规范

H5 开发规范（动态）

DinamicX 开发规范

H5 开发规范




```
# 拉取某个模块代码到本地路径
imgcook pull <moduleId> --path <path>
# 例子
imgcook pull 17108 --path ./src/mods/mod17108
```

对前端智能化的探索

ThoughtWorks®

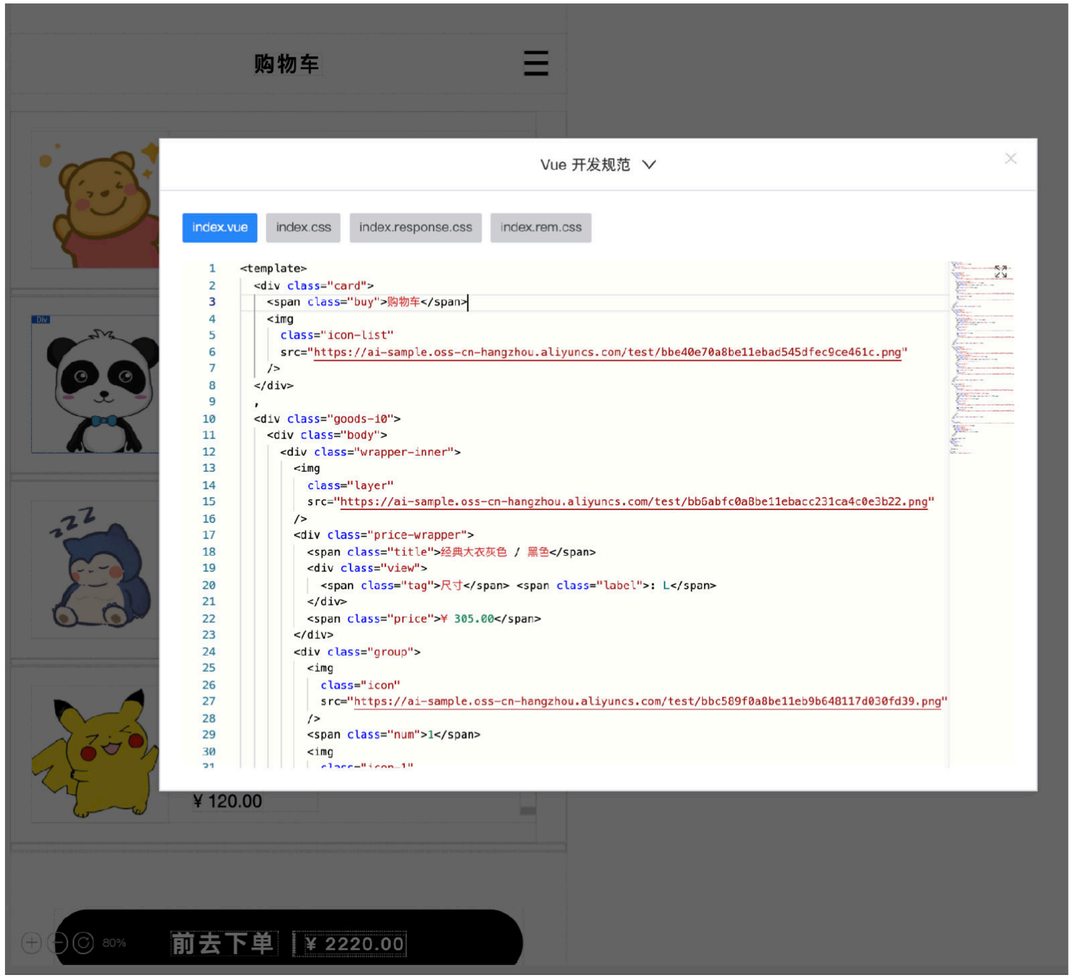
4

imgcook试用效果

自动生成代码留存率41%

在实际项目中使用，需要前端工程师进行干预，但或许存在以下提升空间：

- **设计稿本身**
  - 对设计稿的要求较高，需要设计师学习和掌握规范
  - 对Sketch设计稿源文件的支持优于PhotoShop及其他
  - imgcook对结构化页面的支持，优于非结构化页面
- **生成适合现有前端项目背景的代码**
  - 需要自定义适合项目的DSL



*上图为本人早期试用imgcook试用Sketch设计稿样本生成代码，仅用于介绍，并非真实项目

对前端智能化的探索

ThoughtWorks®

4

imgcook试用效果

对前端项目的价值

个性化活动页面

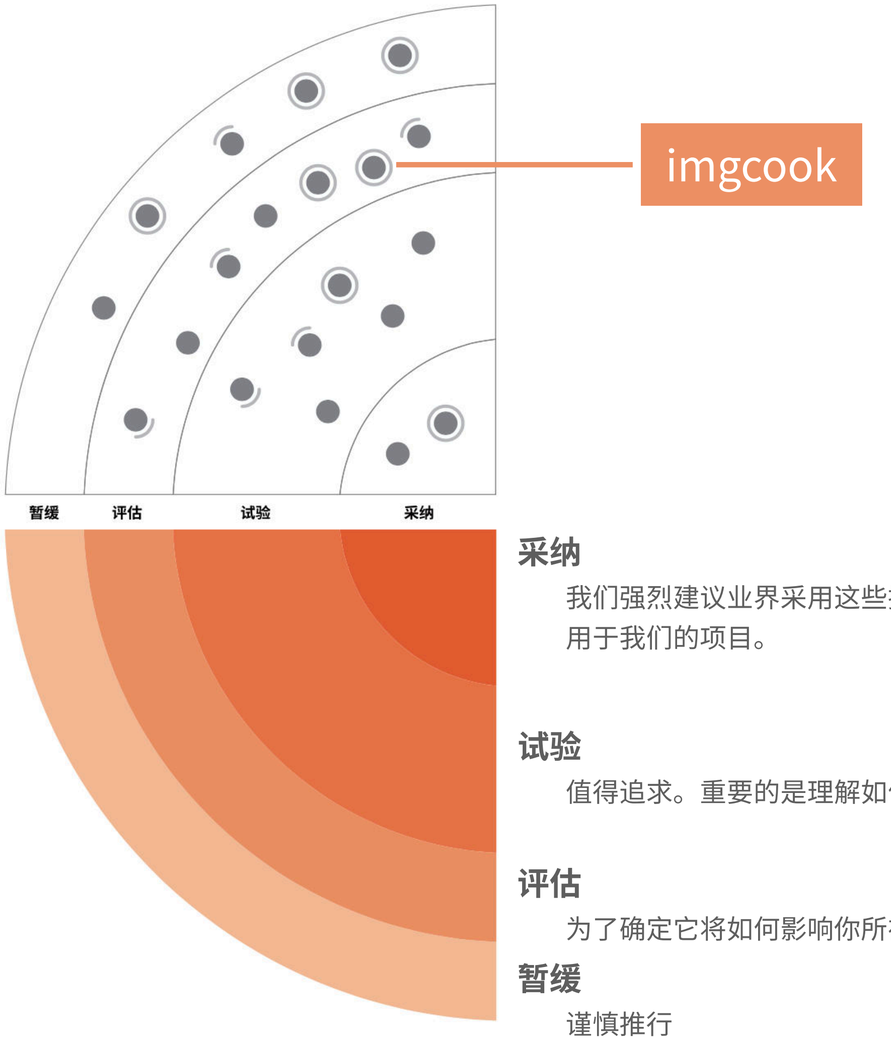
- 组件库和模板化等方式难以覆盖所有的场景，对于个性化的活动，或者暂时无法明确是否会重复消费的页面，imgcook值得一试

移动端细粒度模块开发

- 对细粒度模块生成UI代码支持较好，可以考虑借助imgcook反哺UI组件

规范设计体系，加强设计和前端开发连接

- 或许对于互联网公司，这一命题不太成为问题。但对于尚在转型期和数字化探索阶段的传统行业，通过某种工具作为“Design Lint”也有其价值



采纳

我们强烈建议业界采用这些技术。我们会在适当时候将其用于我们的项目。

试验

值得追求。重要的是理解如何建立这种能力。

评估

为了确定它将如何影响你所在的企业，值得做一番探究

暂缓

谨慎推行

04

结语

ThoughtWorks®

结语

ThoughtWorks®

前端智能化是个有趣的探索，
仍会保持关注

投入要达到一定规模，
之后才能显现收益

无论前端的低代码实践还是智能化探索，
产品背后的原子规范可能和技术本身同样重要

THANK YOU

欲了解详情，请发送邮件至：
yideng@thoughtworks.com

ThoughtWorks®

ThoughtWorks®

赋能利器，助力质量提升

林冰玉

ThoughtWorks 资深质量咨询师

ABOUT ME

ThoughtWorks®

自我介绍

- 思沃学院质量赋能专家、质量咨询师
- 北美业务线QA、QA Lead
- 播客【质量三人行】主播之一
- 博客【林子的空间】、公众号【BY林子】

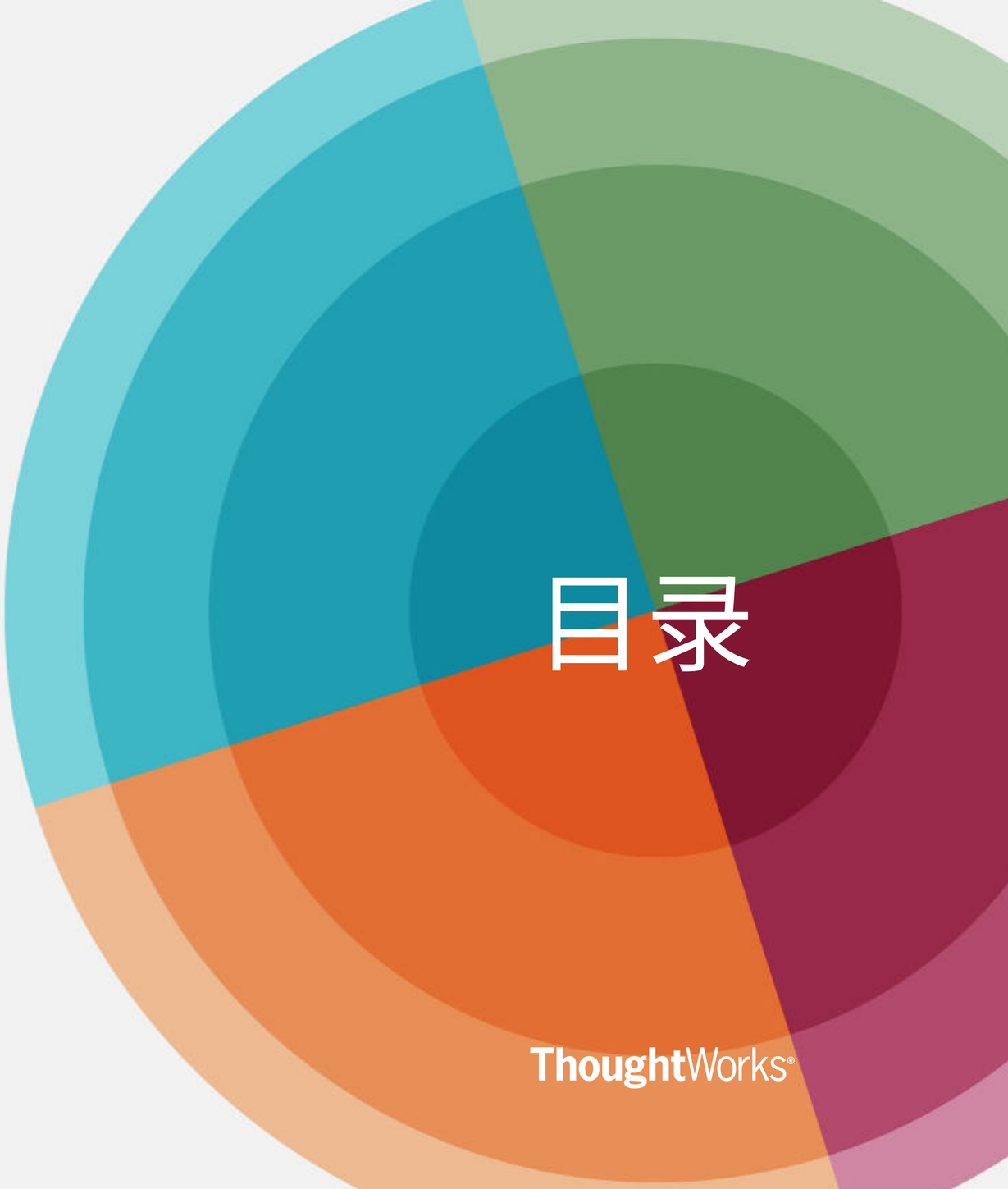
目录

01 案例：质量之痛

02 从技术雷达看赋能

03 真正的赋能利器

04 赋能之道



ThoughtWorks®

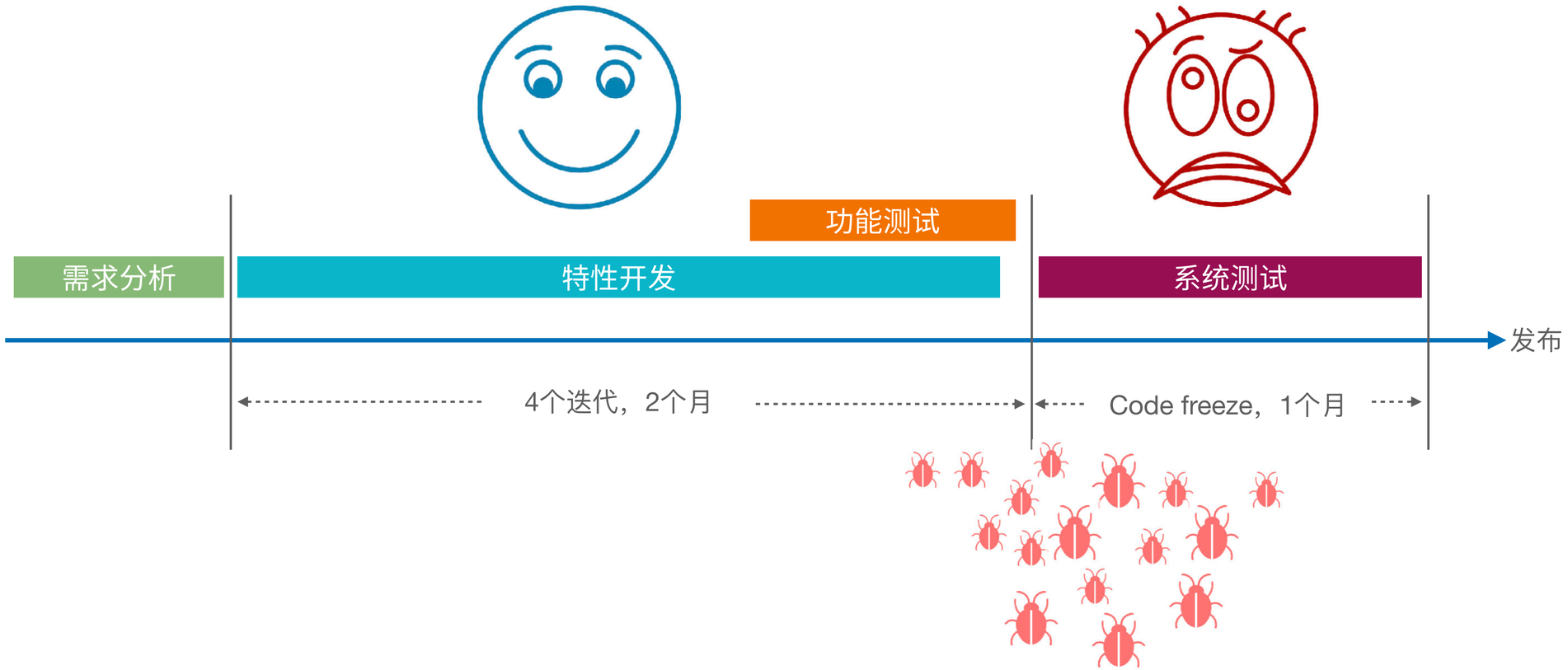
01

案例：质量之痛

ThoughtWorks®

案例背景

ThoughtWorks®



案例痛点分析

ThoughtWorks®

Bug井喷，质量堪忧

- 在系统测试阶段发生bug井喷现象，质量状况堪忧
- 团队后期花费大量精力修复bug，影响特性开发

自动化程度低下，集成较晚

- 缺少单元测试、API测试
- 没有持续集成机制
- 功能测试发生较晚，主要依赖系统集成测试

质量目标不清晰，没有统一策略和标准

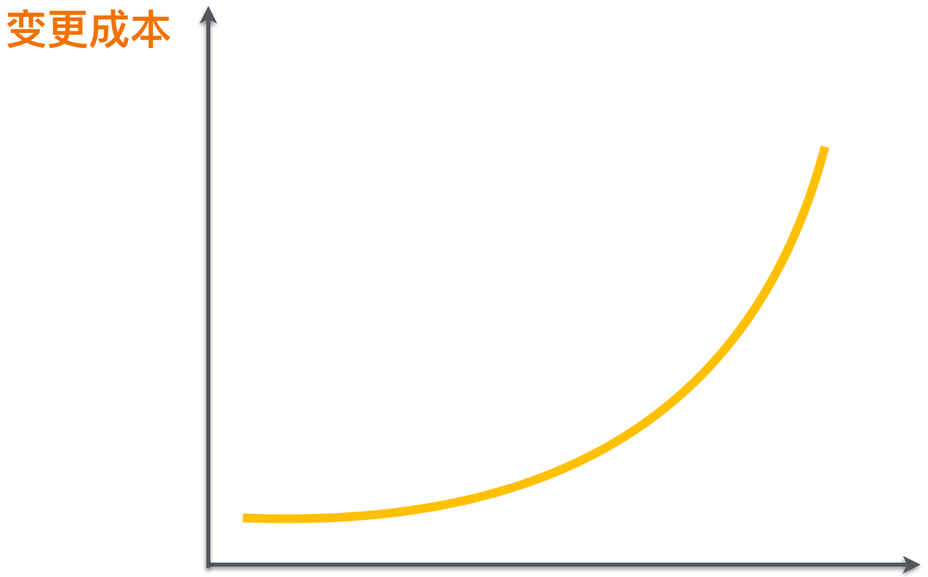
- 团队缺乏清晰的质量目标
- 缺乏统一的测试策略和质量标准
- 角色间相互沟通协作较少

工作分配不均，资源价值没有最大化

- 发布前期，团队整体处于比较松散的状态
- 后期特性开发停止，忙于应对bug的修复，带来浪费

质量保障系统的问题

ThoughtWorks®



“各人自扫门前雪”

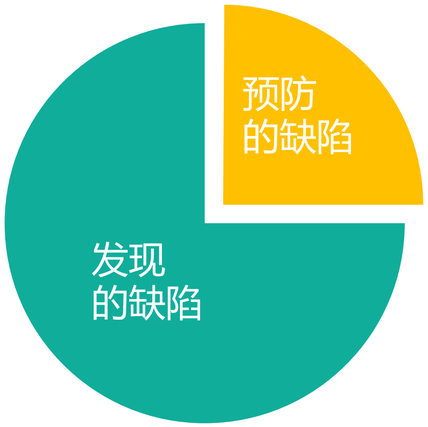
问题怎么解决?

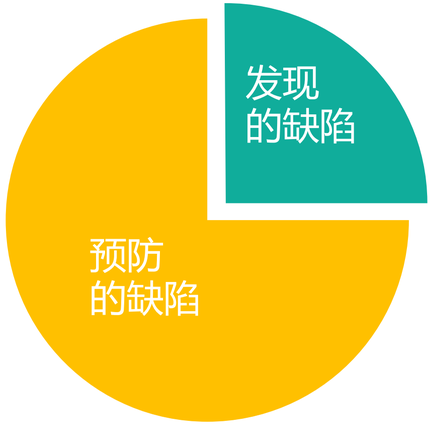
质量内建

ThoughtWorks®

“You can not inspect quality into a product.”

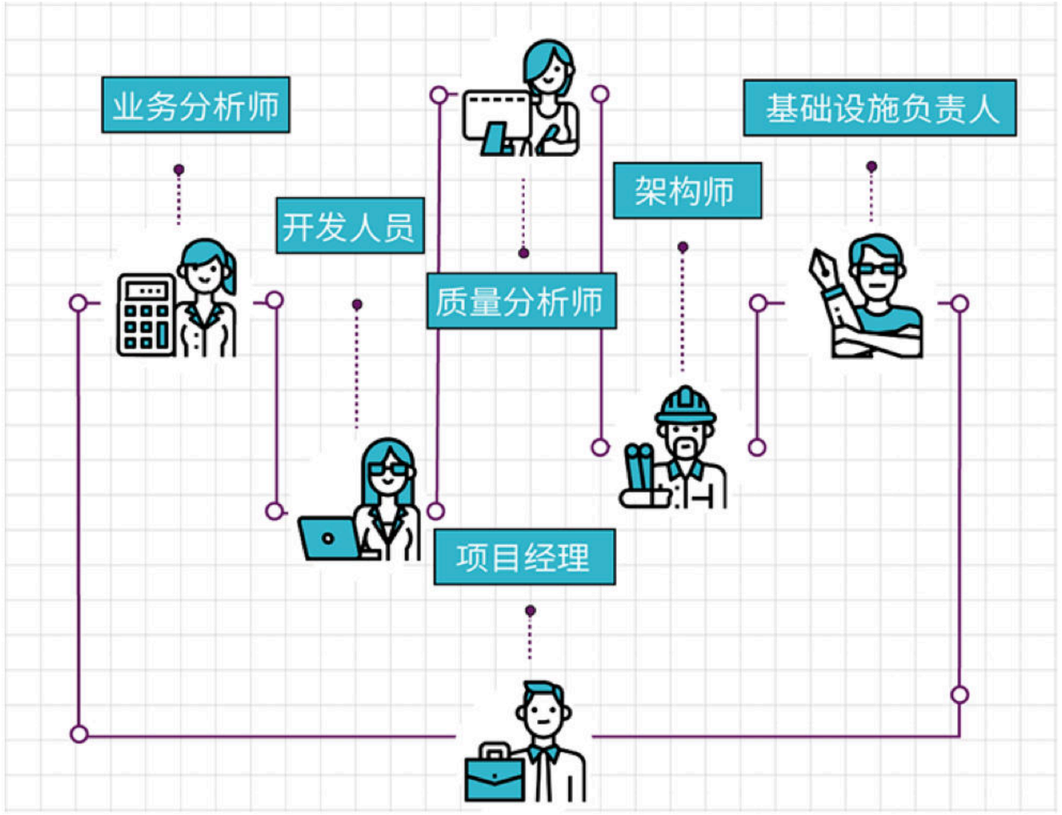
- Dr. W. Edwards. Deming





团队人人对质量负责

ThoughtWorks®



不是人人都能有质量意识，
如何为质量负责？

给团队进行质量保障赋能！

如何给团队进行质量保障赋能?

02

从技术雷达看赋能

ThoughtWorks®

三个趋势

ThoughtWorks®

三个趋势之

ThoughtWorks®

1

软件开发全流程标准化

标准流程：需求计划、代码审查、静态扫描、动态测试、部署发布、监控等

- 流水线+标准工具
- 政策或规则：代码提交、测试覆盖率、失败回滚、发布等

标准化技术及工具的发展

ThoughtWorks®

- Build pipelines（2011）
- Continuous delivery（2012）
- Automated deployment pipeline（2013）
- Continuous delivery for mobile devices（2014）
- GoCD（2015）

CD技术
流水线工具

- Bitrise（2020）
- Concourse （2020）

新的流水线工具

- Azure DevOps
- CD4ML

平台级方案

- 其他标准化工具ESLint、Commitizen、Prettier等

规则相关工具

标准化技术及工具的发展

ThoughtWorks®

- Build pipelines（2011）构建流水线
- Automated deployment pipeline（2013）自动部署流水线

CD技术
流水线工具

- Continuous delivery（2012）
- Continuous delivery for mobile devices（2014）
- GoCD（2015）

新的流水线工具

- Bitrise（2020）
- Concourse （2020）

平台级方案

- Azure DevOps
- CD4ML

规则相关工具

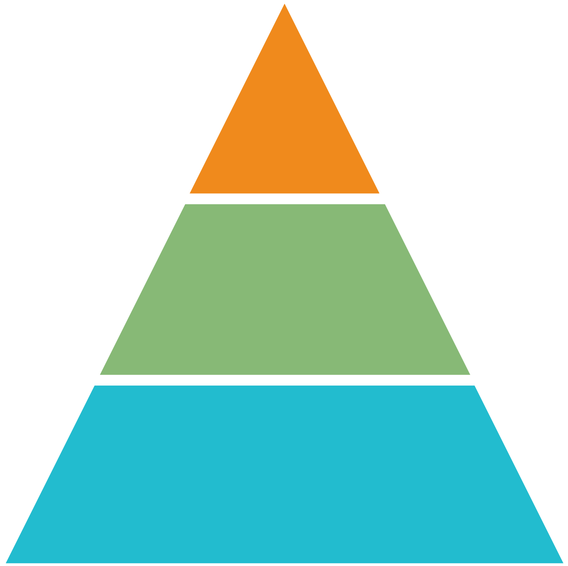
- 其他标准化工具ESLint、Commitizen、Prettier等

三个趋势之

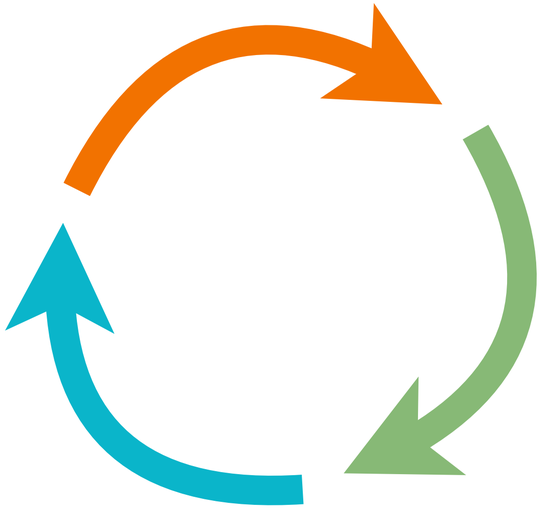
ThoughtWorks®

2

大规模自动化助力标准化



测试自动化



流程自动化

自动化技术与工具的演进

ThoughtWorks®

- Ansible（2014）- Testinfra（2017）
- Terraform（2019）- Terratest (2021)

自动化基础实施
管理及其测试

- Pipeline for infrastructure as code（2019）
- Infrastructure as code（2020）
- Pipeline as code（2020）
- Security policy as code（2020）

X即代码

- fastlane（2017）
- Dependabot（2020）

自动化发布流程、
依赖检查

- Test at the appropriate level（2012）
- Consumer-driven contract testing（2016）
- React Testing Library、Cypress、Karate、K6等

自动化测试相关

自动化技术与工具的演进

ThoughtWorks®

- Ansible（2014）- Testinfra（2017）
- Terraform（2019）- Terratest (2021)

自动化基础实施
管理及其测试

- Pipeline for infrastructure as code（2019）
- Infrastructure as code（2020）
- Pipeline as code（2020）
- Security policy as code（2020）

X即代码

- fastlane（2017）
- Dependabot（2020）

自动化发布流程、
依赖检查

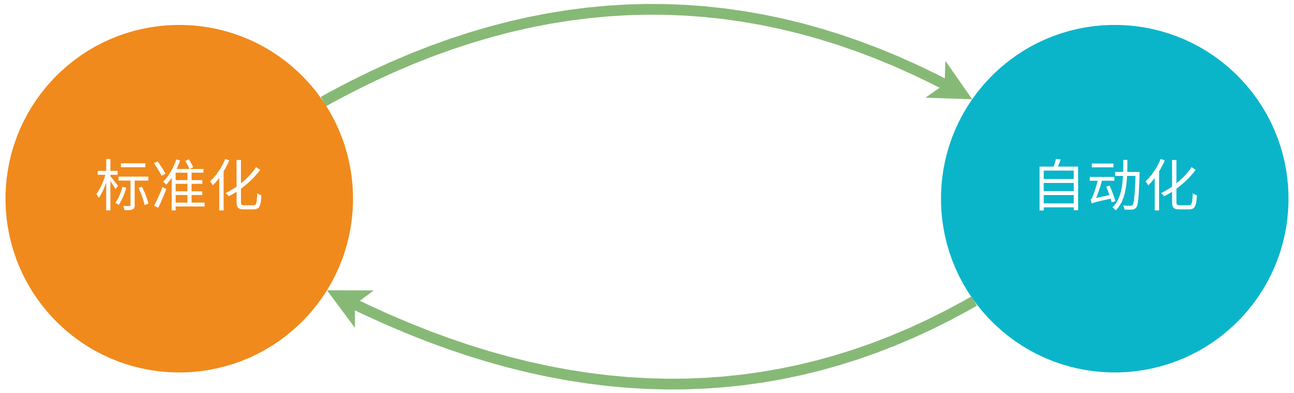
- Test at the appropriate level（2012）合适的测试分层
- Consumer-driven contract testing（2016）
- React Testing Library、Cypress、Karate、K6等

自动化测试相关

标准化与自动化相辅相成

ThoughtWorks®

严格的标准化使得自动化更容易



完备的自动化使得标准化得以保障

三个趋势之

ThoughtWorks®

3

系统韧性成为质量的一部分

软件生态日益复杂，
不确定性因素增加

测试无法保障系统100%
可靠性和正确性

对速度的要求越来越高，
快速反馈成为关键

构建系统韧性的技术与工具

ThoughtWorks®

- Data visualizations of development and operations (2012)
- Focus on mean time to recovery (2015)
- QA in production (2016)
- Grafana (2017)
- Observability as code (2018)
- Sentry (2021)

可观测性、监控

- Chaos monkey (2014)
- Chaos engineering (2019)

混沌工程

- 1% Canary (2018)

金丝雀发布

构建系统韧性的技术与工具

ThoughtWorks®

- Data visualizations of development and operations (2012)
- Focus on mean time to recovery (2015) 关注平均恢复时间
- QA in production (2016) 生产环境下的QA
- Grafana (2017)
- Observability as code (2018)
- Sentry (2021)

性、监控

- Chaos monkey (2014)
- Chaos engineering (2019)

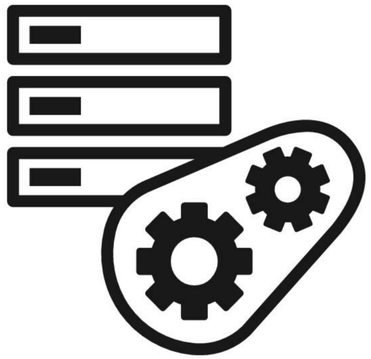
混沌工程

- 1% Canary (2018)

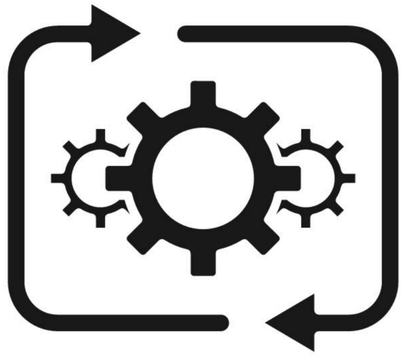
金丝雀发布

赋能利器

ThoughtWorks®



标准化：流程 + 实践



自动化：流程 + 测试

赋能利器就这些吗？

03

真正的赋能利器

ThoughtWorks®

思考两个问题

ThoughtWorks®

仅靠工具
就能解决赋能问题吗?

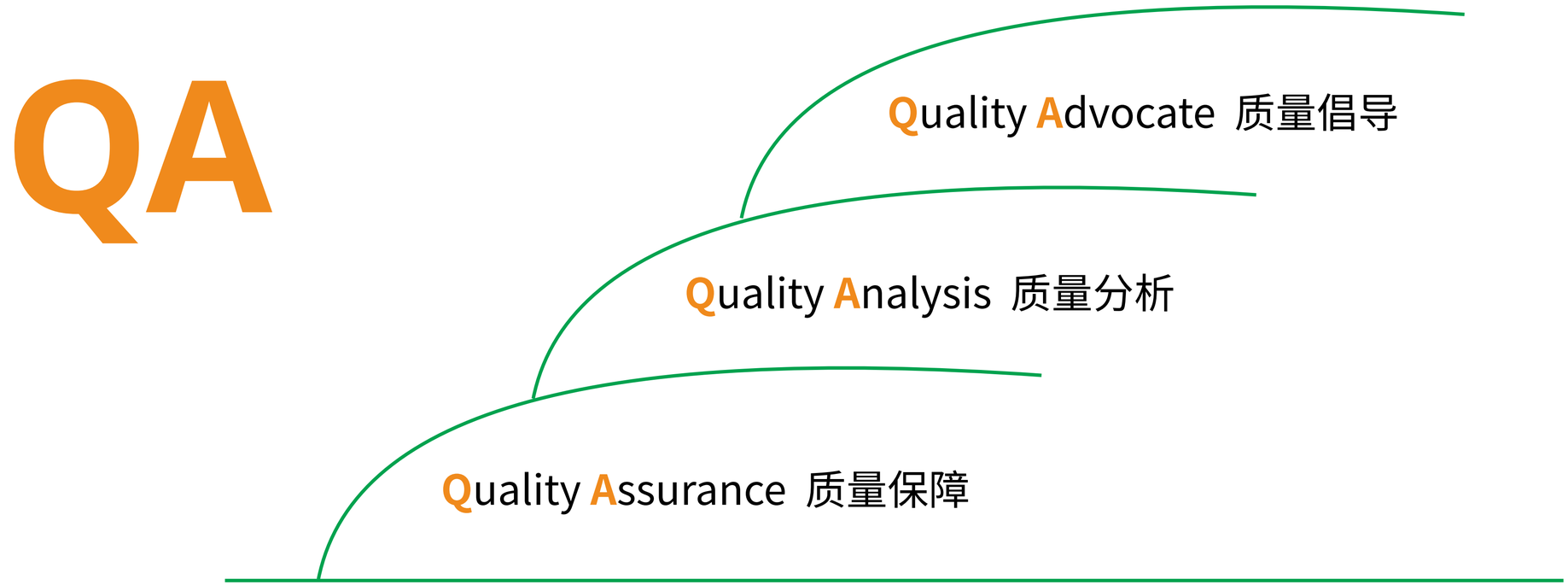
01

测试工作都自动化了，
还需要QA吗?

02

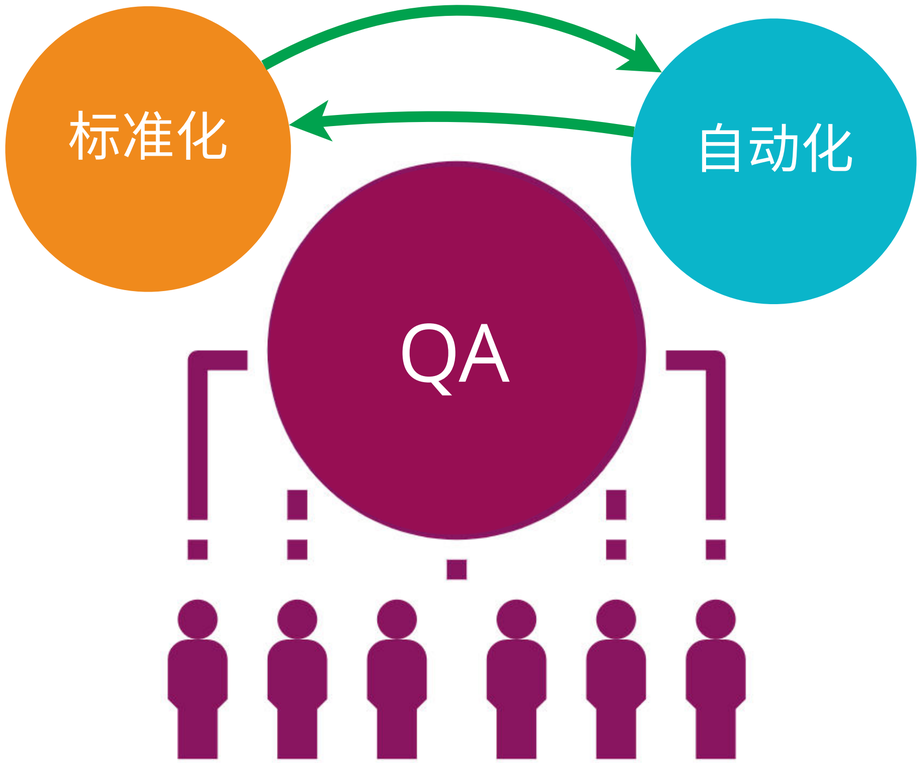
QA承担赋能者的职责

ThoughtWorks®



真正的赋能利器

ThoughtWorks®



质量赋能有这些利器就够了吗？

04

赋能之道

ThoughtWorks®

质量赋能更为关键的还有……

ThoughtWorks®

文化认知

- 免责文化，鼓励创新
- 团队负责的质量文化

目标驱动

- 清晰的质量目标
- 团队共识

策略指导

- 团队共识的质量策略
- 演进式

能力建设

- 充分沟通和协作
- 关注人员能力培养

总结

KEY TAKEAWAYs

ThoughtWorks®

总结

团队质量保障赋能是大势所趋

工具不是万能利器，QA需要转变成为赋能者

文化认知、质量策略、人员能力不能拖后腿

THANK YOU

ThoughtWorks®

林子的空间

BY林子

质量三人行

ThoughtWorks®

DevSecOps 的左移和右移

蒋帆
ThoughtWorks
首席咨询师

刘夏
ThoughtWorks
首席咨询师

01

DevSecOps 的崛起

ThoughtWorks®

DevSecOps 的崛起

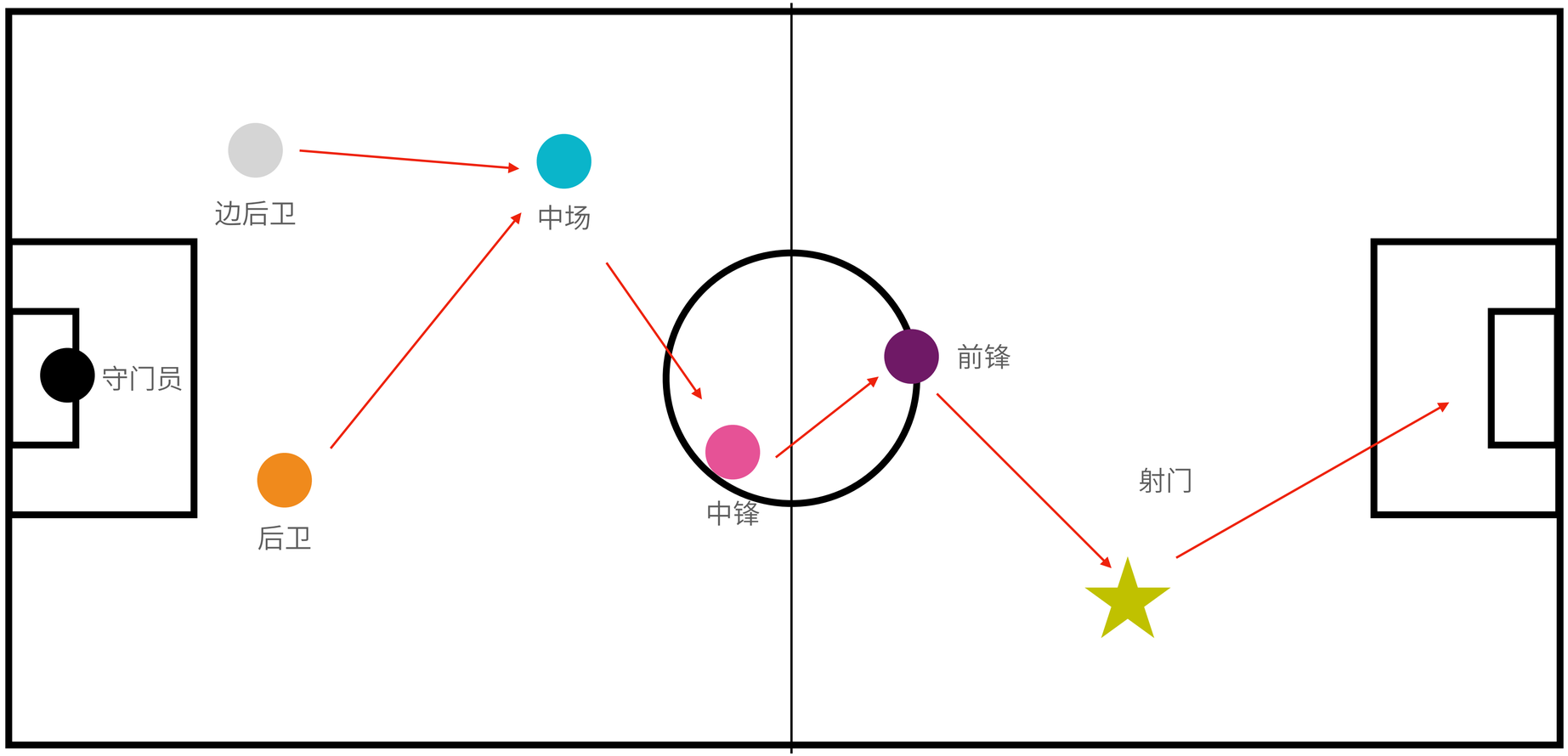
ThoughtWorks®

数字化时代，安全的重要性将越发明确具体

DevSecOps 的崛起

ThoughtWorks®

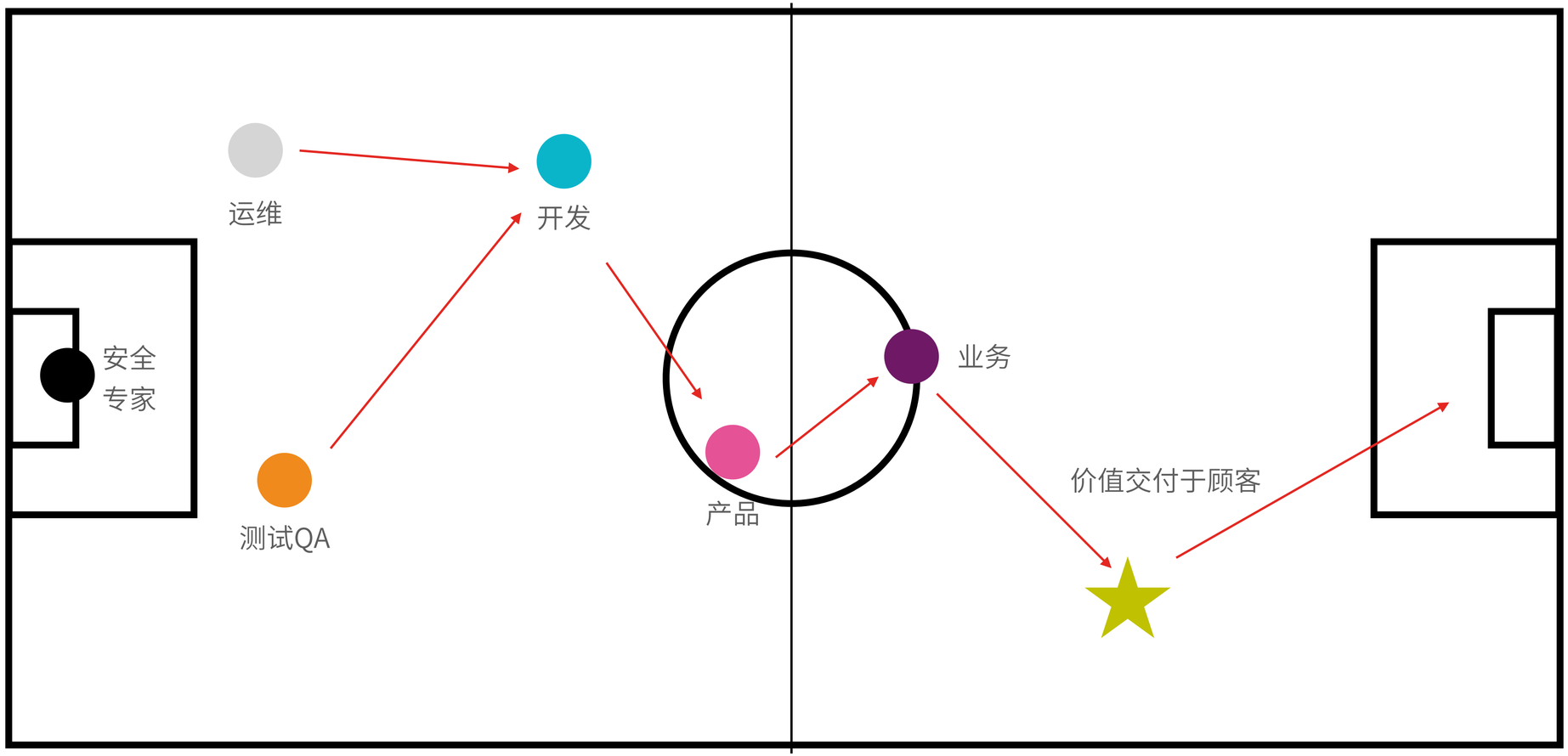
DevSecOps 的是一种求变的价值主张



DevSecOps 的崛起

ThoughtWorks®

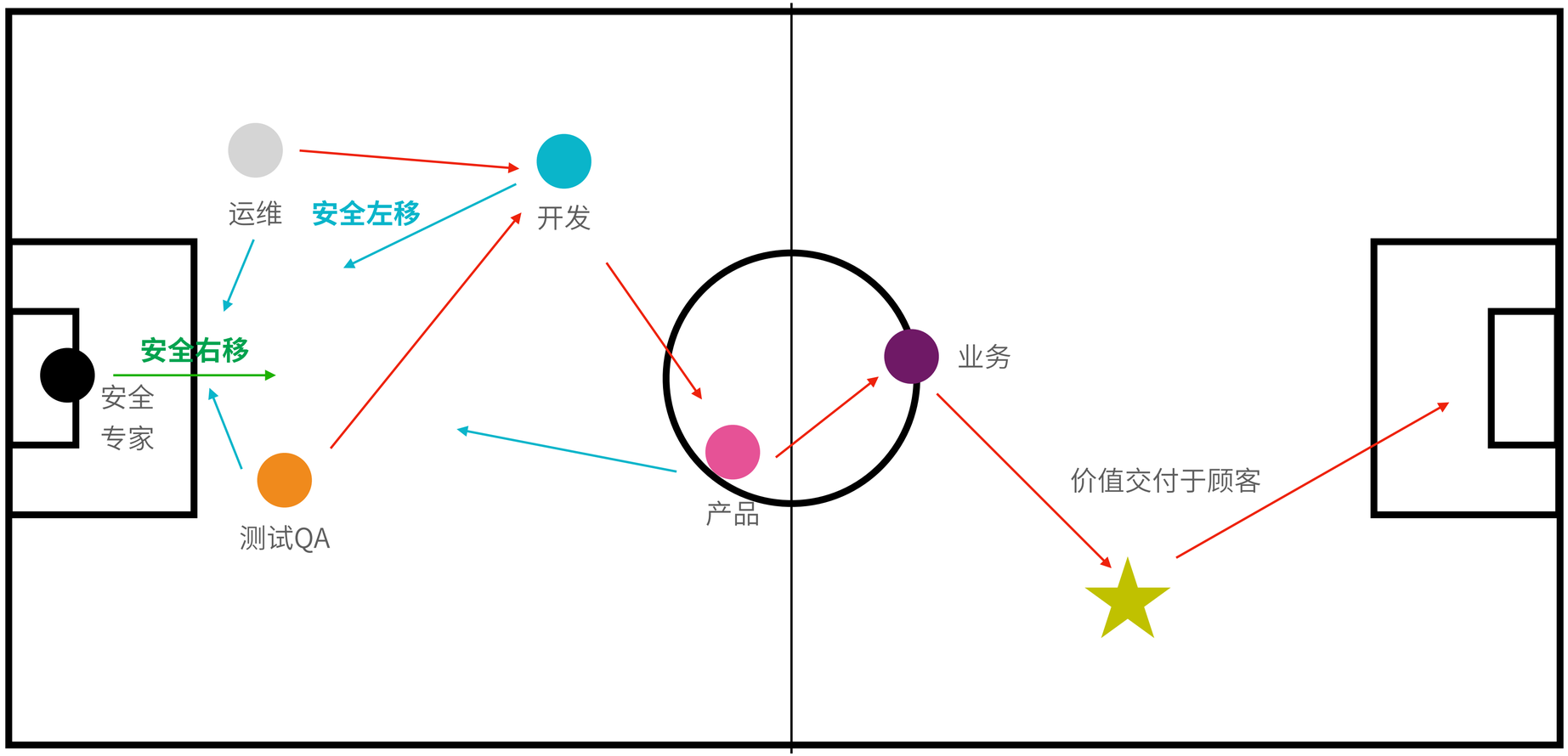
DevSecOps 的是一种求变的价值主张



DevSecOps 的崛起

ThoughtWorks®

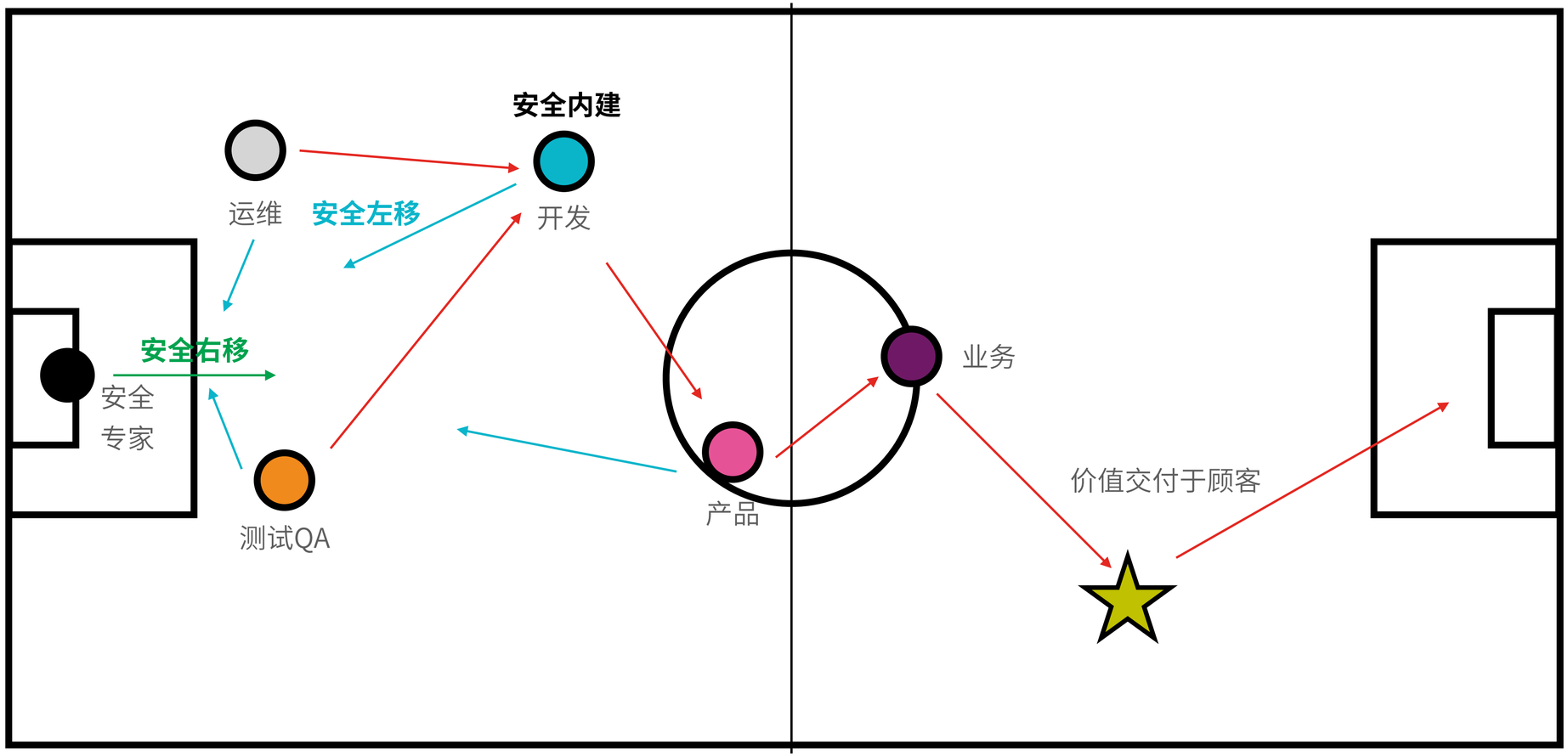
DevSecOps 的是一种求变的价值主张



DevSecOps 的崛起

ThoughtWorks®

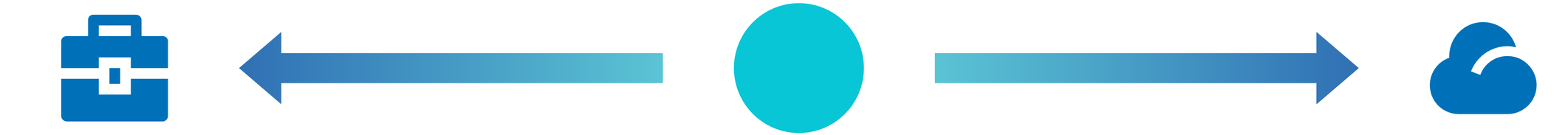
DevSecOps 的是一种求变的价值主张



DevSecOps 的崛起

ThoughtWorks®

左移、右移、内建



项目启动和需求

传统安全工作的防御环节

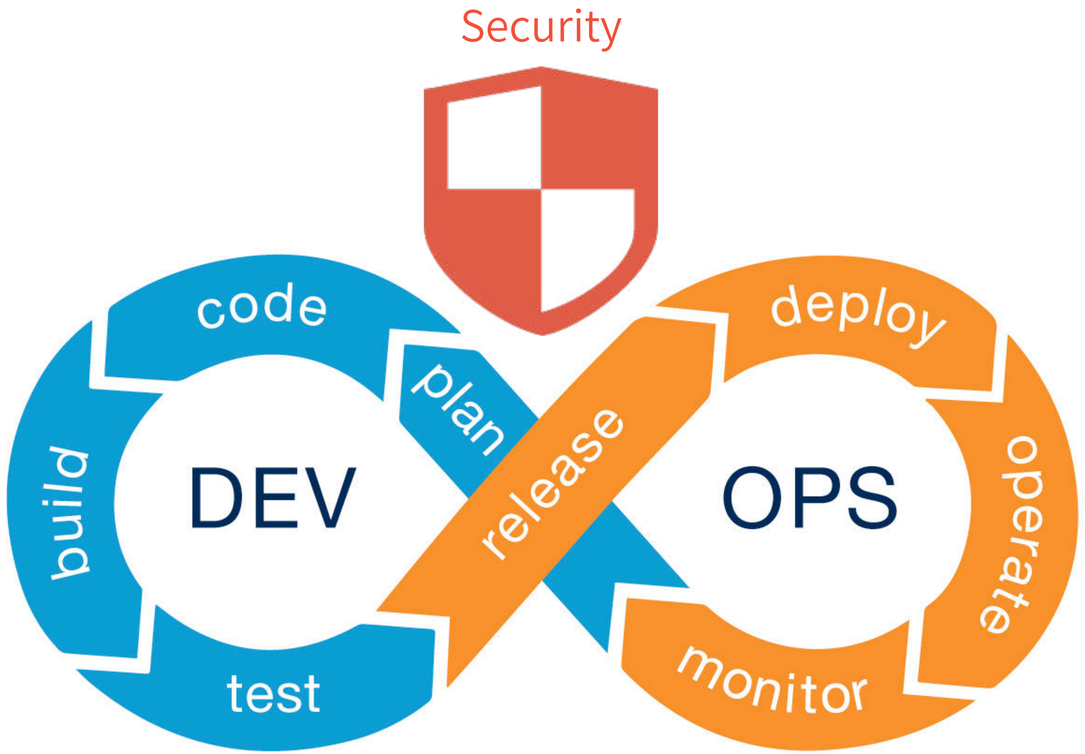
部署和维护

安全工作的全方位内建是必需的

ThoughtWorks®

与“质量内建”、“开发运维一体化 DevOps”运动相似

DevSecOps 从提出之时起的目标之一就是安全工作的内建



安全工作的全方位内建是必需的

ThoughtWorks®

全方位的内建需要体系化的 DevSecOps 建设

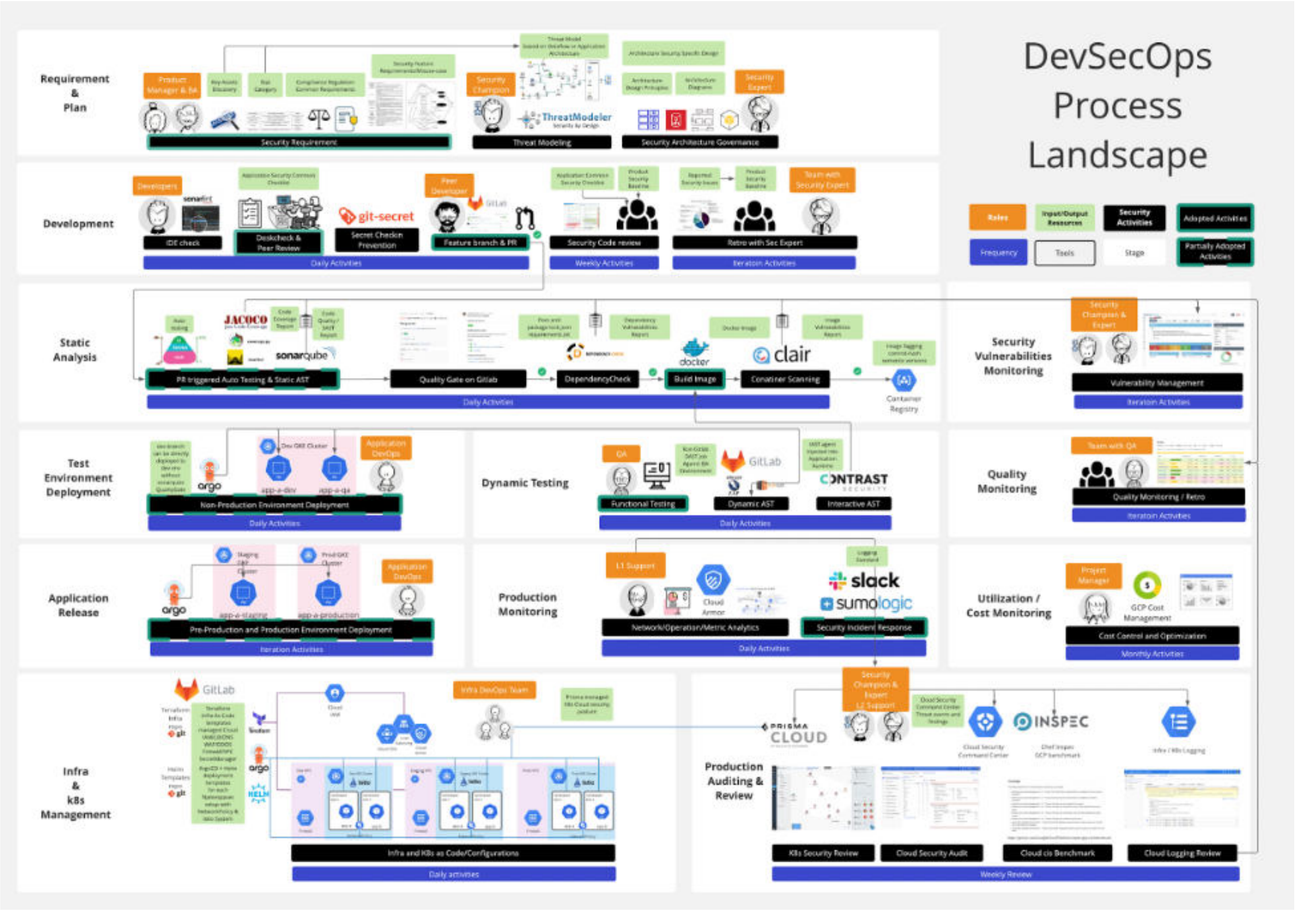
安全工作的全方位内建是必需的

ThoughtWorks®

全方位的内建需要体系化的 DevSecOps 建设

意识 → 方法 → 流程

实践 → 工具 → 平台



02

23 & 24 期《技术雷达》中 DevSecOps 的左移与右移

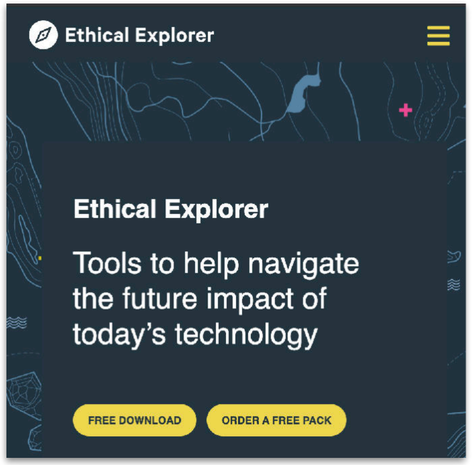
ThoughtWorks®

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

左移

Ethical Explorer



https://ethicalexplorer.org/

产品设计与需求分析

Trial 试验

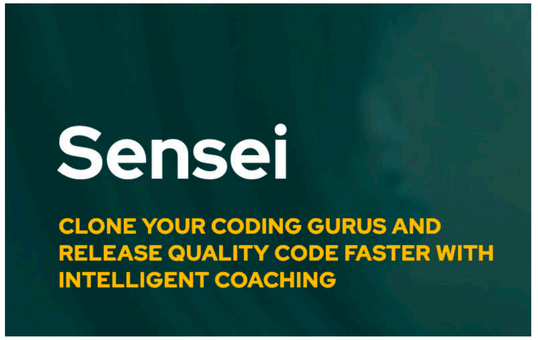
LGTM

编码开发

Assess 评估

Sensei

SECURE CODE WARRIOR



编码开发

Assess 评估

Distroless Container Image

制品构建

Trial 试验

DevSecOps 左移、右移与技术雷达

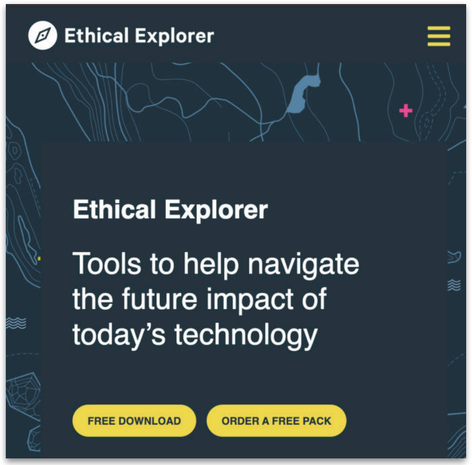
ThoughtWorks®

Ethical Explorer （左移）

产品设计与需求分析　　Trial 试验

EBay 创始人成立的 Omidyar 网络最近新发布了一个名为 Ethical Explorer 的 版 本。 新的 Ethical Explorer 在 Ethical OS 实践得来的经验教训的基础上，增加了更深入的问题供产品团队考虑。

Ethical Explorer 可以免费下载，并可折叠为卡片来引发讨论，这些卡片上有针对技术上的若干种“危险区”的一些开放性问题，包括**监视、虚假信息、排异排外、算法歧视、沉迷上瘾、数据控制、安全攻击以及权限过大**。其中的实战指南包括一些**工作坊、开启对话的思路和获得组织支持的技巧**。



https://ethicalexplorer.org/

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

LGTM（左移）

编码开发

Assess 评估

编写安全代码一向是重中之重， LGTM 提供了一种从安全编码实践的知识库中受益的方法。它是一个静态代码分析工具，专注于安全性，由一系列安全编码规则（部分开源）支持。

这些规则使用**CodeQL**查询语言对代码库进行查询（Query）。可用于将白盒安全检查集成到Java，Go，JavaScript，Python，C# 和 C/C++的持续集成管道中。

LGTM和CodeQL是Github安全实验室的一部分。

https://securitylab.github.com/tools/codeql/

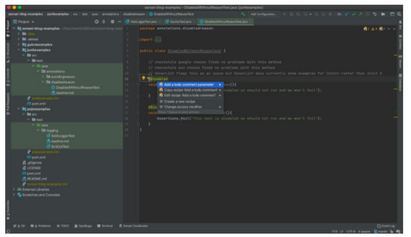
DevSecOps 左移、右移与技术雷达

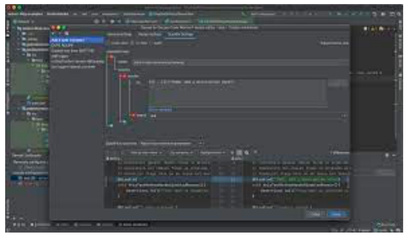
ThoughtWorks®

Sensei（左移）

编码开发　　Assess 评估

Sensei 是来自 Secure Code Warrior 的一款Java IDE 插件，可以很容易地用于创建和分发**安全代码质量规则**。在 ThoughtWorks，我们通常提倡“**工具胜过规则**”，而不是完全依赖“检查列表”那样的治理方法，而 Sensei 就符合这个哲学。开发者可以创建出很容易跟其他团队成员共享的代码片段。它们被实现成针对 Java AST 的查询，例如对于 SQL 注入和加密漏洞的警告。





Sensei 在 IDE 中一旦发现代码变化就开始执行检查，因此它比其他传统的静态分析工具提供了更快的反馈。

https://www.securecodewarrior.com/products/sensei

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

Distroless Container Image（左移）

制品构建

Trial 试验

在为应用构建 Docker 镜像时，我们常常会考虑两件事： 镜像的安全性和大小。

Distroless Docker images 通过放弃完整的操作系统发行版来减少内存占用和依赖关系，它还可以减少安全扫描噪声和应用程序攻击面。

此外，这一技术还意味着需要修补的漏洞更少了，镜像更小性能也更好了。

Google 针对不同的语言发布了一套 distroless container images， 你可以使用 Google 构建工具 Bazel 或者简单地使用"多阶段 Dockerfiles"创建 distroless 的应用程序镜像。

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

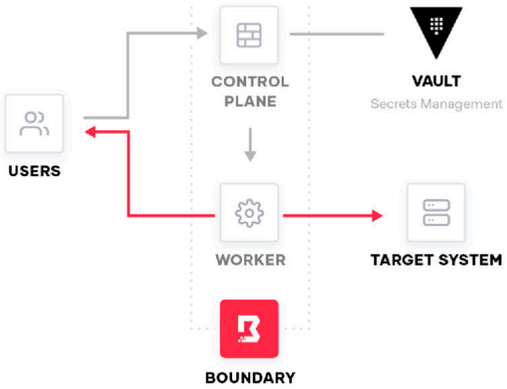
右移

零信任架构

运维运营架构

Assess 评估

HashiCorp Boundary



运维运营架构

Assess 评估

安全策略即代码

安全策略治理

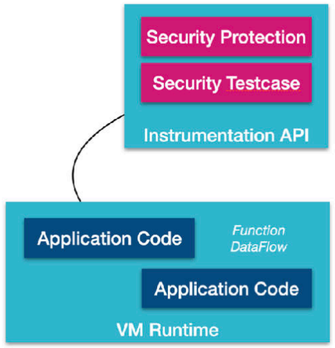
Adopted 采纳

Open Policy Agent (OPA)

安全策略治理

Assess 评估

IAST/RASP



安全测试

生产环境安全保护

Proposed 待提交

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

零信任架构 （右移）

运维运营架构　　Assess 评估

零信任架构是安全体系结构及策略的一种模式转变。它基于这样的假设：网络边界不再代表安全边界，不应仅基于物理或网络位置授予用户或服务隐式信任。

用于实现零信任架构各个方面的资源、工具以及平台的数量持续增长，其中包括：以**最小特权为基础**，尽可能细化原则，并持续监控和自动缓解威胁；使用**服务网格**来实施应用到服务和服务到服务的安全控制；使用**二进制校验**以验证二进制文件的来源；除传统加密外，还包括**TEE**，在处理器、存储、内存三大器件上实现数据安全性：。

有关该主题的有用介绍，请参阅 **NIST ZTA** 刊物和有关 **BeyondProd** 的 Google 白皮书。

https://www.thoughtworks.com/talks/trust-teams-but-verify

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

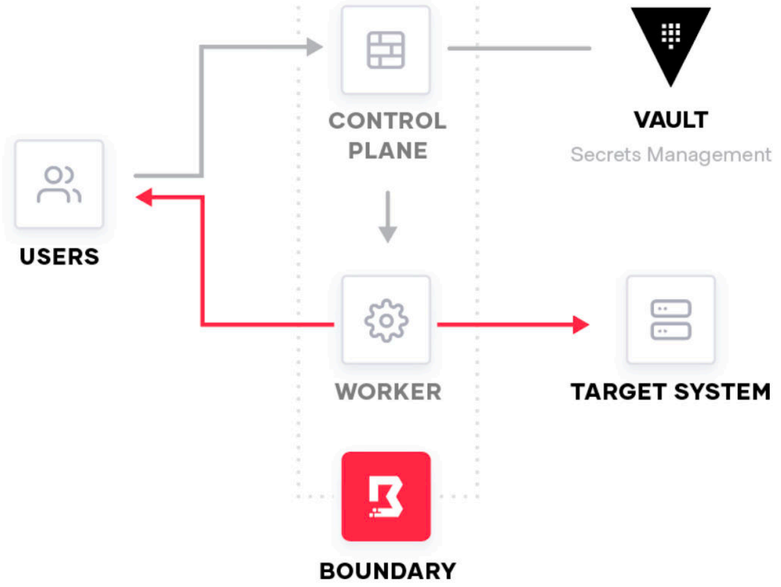
HashiCorp Boundary（右移）

运维运营架构　　Assess 评估

在访问主机和服务的场景下，安全网络和身份管理的能力是不可或缺的， HashiCorp Boundary 将这些能力合并在一处。

HashiCorp Boundary 支持多样化的身份认证提供方，并且可以集成到你的服务整体架构当中，来帮助定义主机甚至是服务级别的权限。

例如，它可以用来实现对 Kubernetes 集群的细粒度访问控制。并且这些额外的访问控制是不可感知的，这一切都通过 Boundary 的网络管理层安全地进行连接。



DevSecOps 左移、右移与技术雷达

ThoughtWorks®

安全策略即代码（右移）

安全策略治理　　Adopted 采纳

随着技术格局逐渐复杂化，诸如安全性的 题需要引入更多的自动化和工程实践。在系统构建过程中，我们需要将安全策略 —— 即保护系统免受威胁和破坏的规则和程序，纳入考虑中。

当我们说“即代码”时， 不仅意味着将这些安全策略写入文件中，还需要将其应用到诸如在代码中采用版本控制、在流水线中引入自动验证、在环境中自动部署并观察监控其性能等实践中。

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

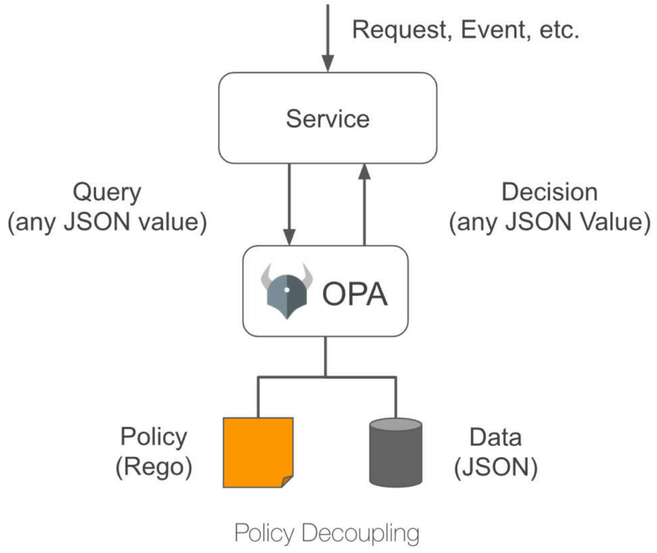
Open Policy Agent, aka OPA（右移）

安全策略治理　　Assess 评估

OPA 提供了一套统一的框架和语言，用于声明、实施和控制云原生解决方案中各个组件的策略。

它是实现**安全策略即代码**的工具中的一个很好的例子。

无论是在 K8s 集群中部署资源，还是在服务网格（Service Mesh）中的跨服务执行访问控制，抑或是通过代码精准地控制应用资源访问，OPA 都给我们提供了非常顺畅的体验。



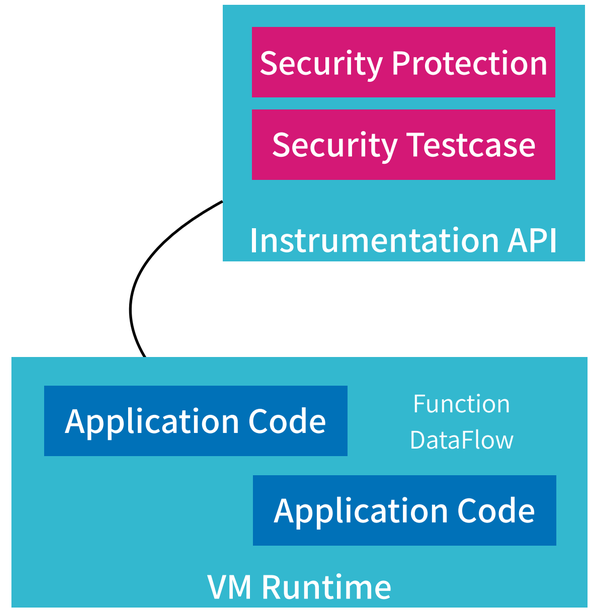
https://www.openpolicyagent.org/docs/latest/

DevSecOps 左移、右移与技术雷达

ThoughtWorks®

IAST / RASP（右移）　安全测试、生产环境安全保护　Proposed 待提交

IAST：交互式应用程序安全测试（Interactive Application Security Testing），通过服务端运行时部署Agent程序，收集、监控Web应用程序运行时**函数执行、数据传输**，并与扫描器端进行实时交互，**高效、准确的识别安全缺陷及漏洞**，同时可准确确定漏洞所在的代码文件、行数、函数及参数。



RASP：实时应用程序自我防护（Runtime Application Self-Protection）是一种新型应用安全保护技术，它将保护程序**像疫苗和免疫系统一样注入到应用程序和应用程序融为一体**，能**实时检测和阻断安全攻击**，使应用程序具备自我保护能力，当应用程序遇到特定漏洞和攻击时不需要人工干预就可以进行自动重新配置应对新的攻击。

03

DevSecOps 工作的挑战与未来展望

ThoughtWorks®

ThoughtWorks®

DevSecOps 工作的未来展望

平台化
服务化
持续化

安全能力
业务理解
开发经验

DevSecOps 工作的未来展望

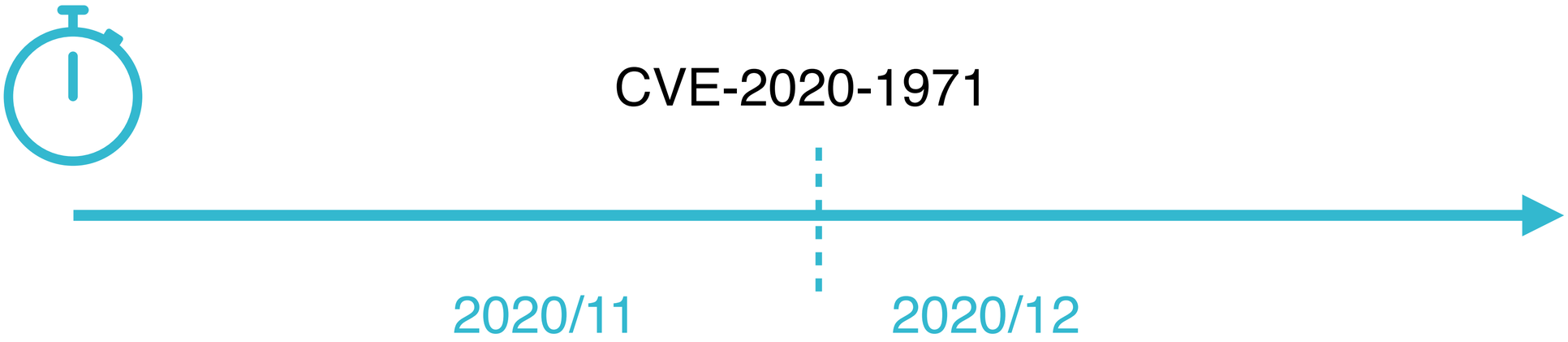
ThoughtWorks®

平台化、服务化、持续化将成为 DevSecOps 工作的重要基础

DevSecOps 工作的未来展望

ThoughtWorks®

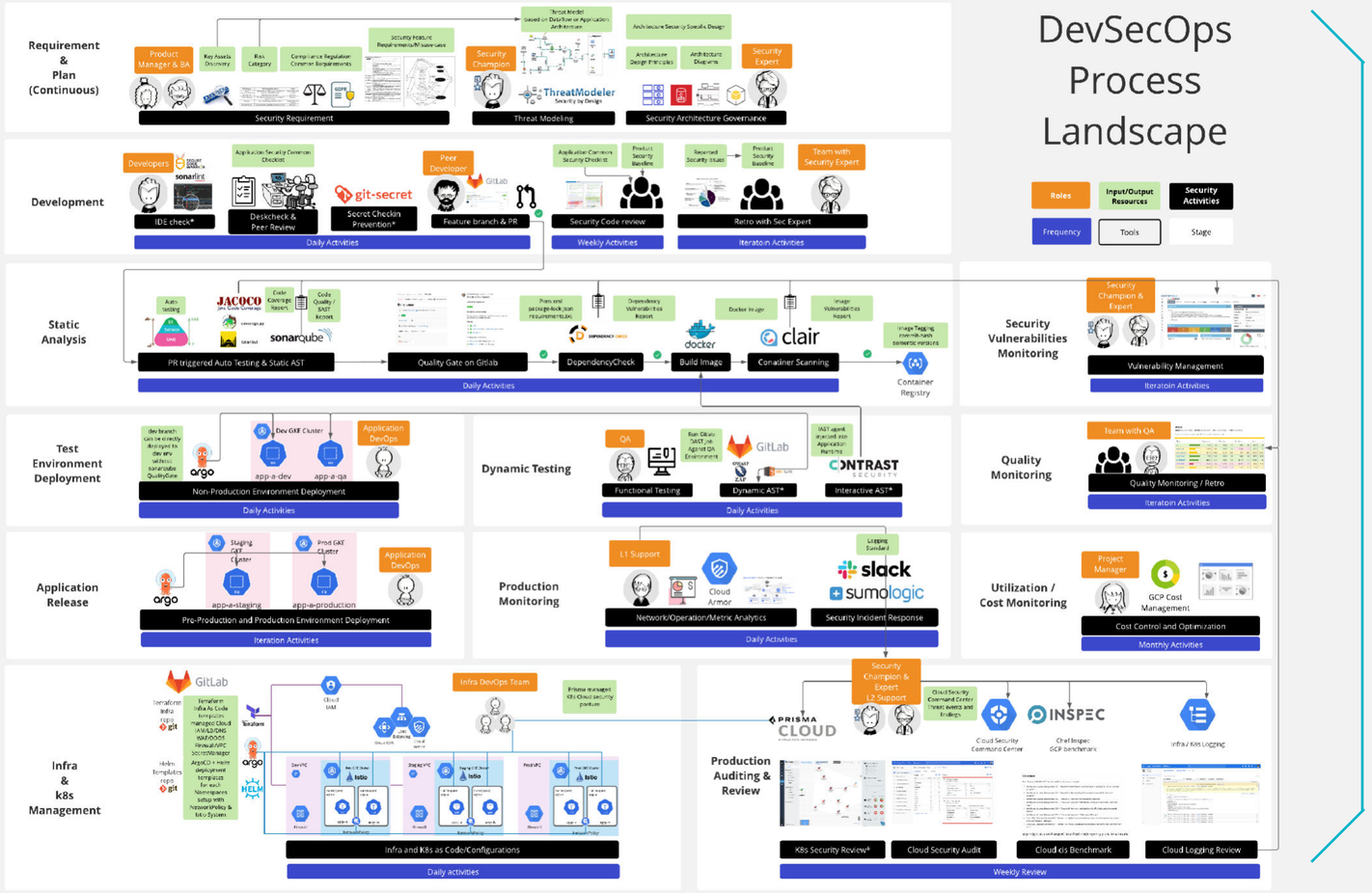
平台化、服务化、持续化将成为 DevSecOps 工作的重要基础



DevSecOps 工作的未来展望

ThoughtWorks®

平台化、服务化、持续化将成为 DevSecOps 工作的重要基础



Pipeline As Code

Infrastructure As Code

Security Policy As Code

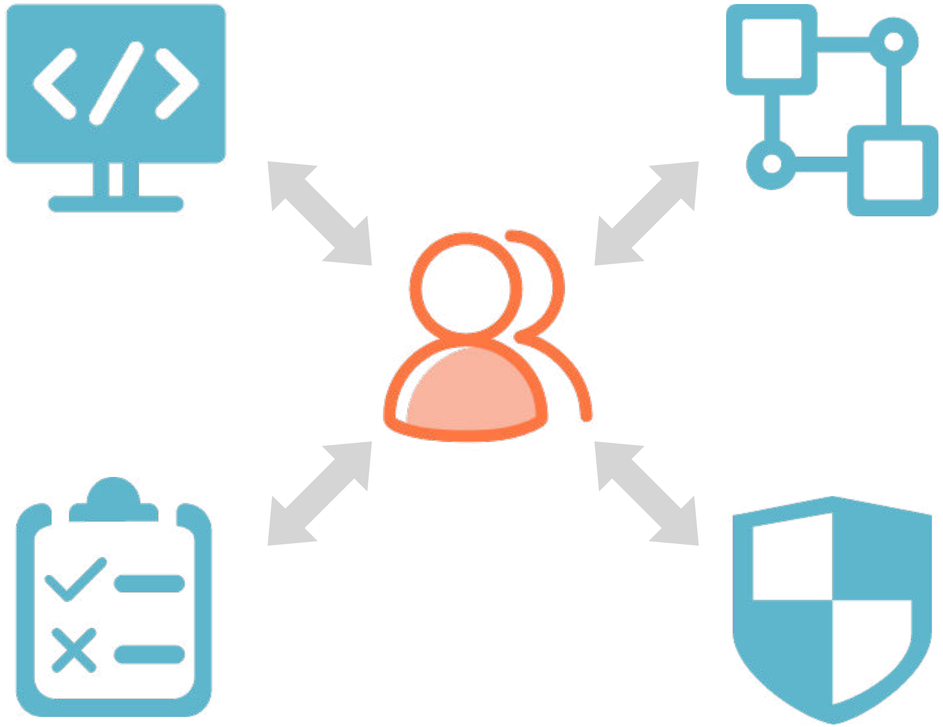
Secrets As A Service

ZeroTrust Architecture

DevSecOps 工作的未来展望

ThoughtWorks®

DevSecOps 呼唤综合性人才



ThoughtWorks®

DevSecOps 工作的未来展望

平台化
服务化
持续化

安全能力
业务理解
开发经验

THANK YOU

ThoughtWorks®