Engineering SOA

Service-Oriented Architecture

Jeremy Gibbons

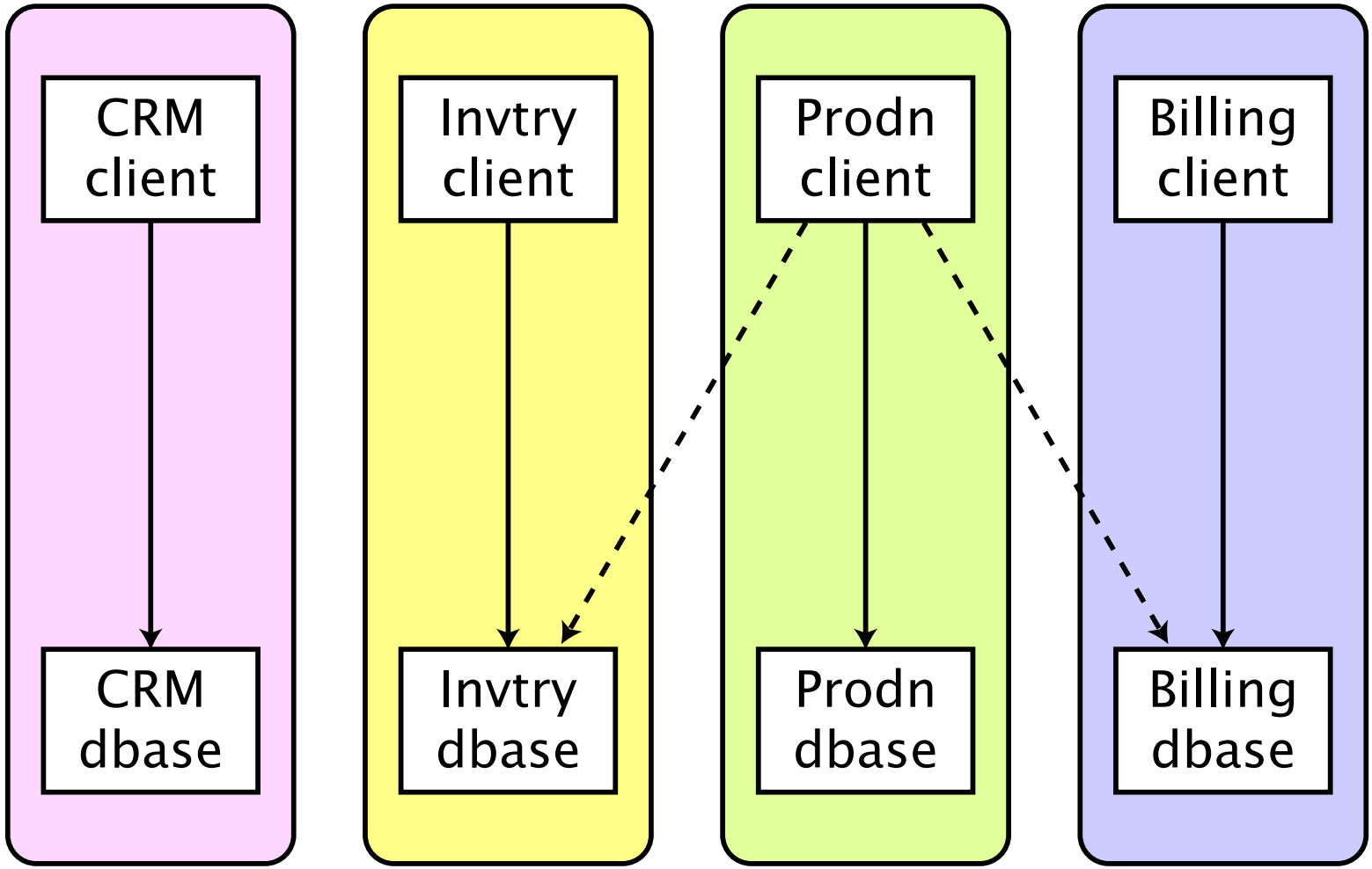
Contents

1 Organization

- SOA is not just a technical development
- successful adoption has organizational consequences too

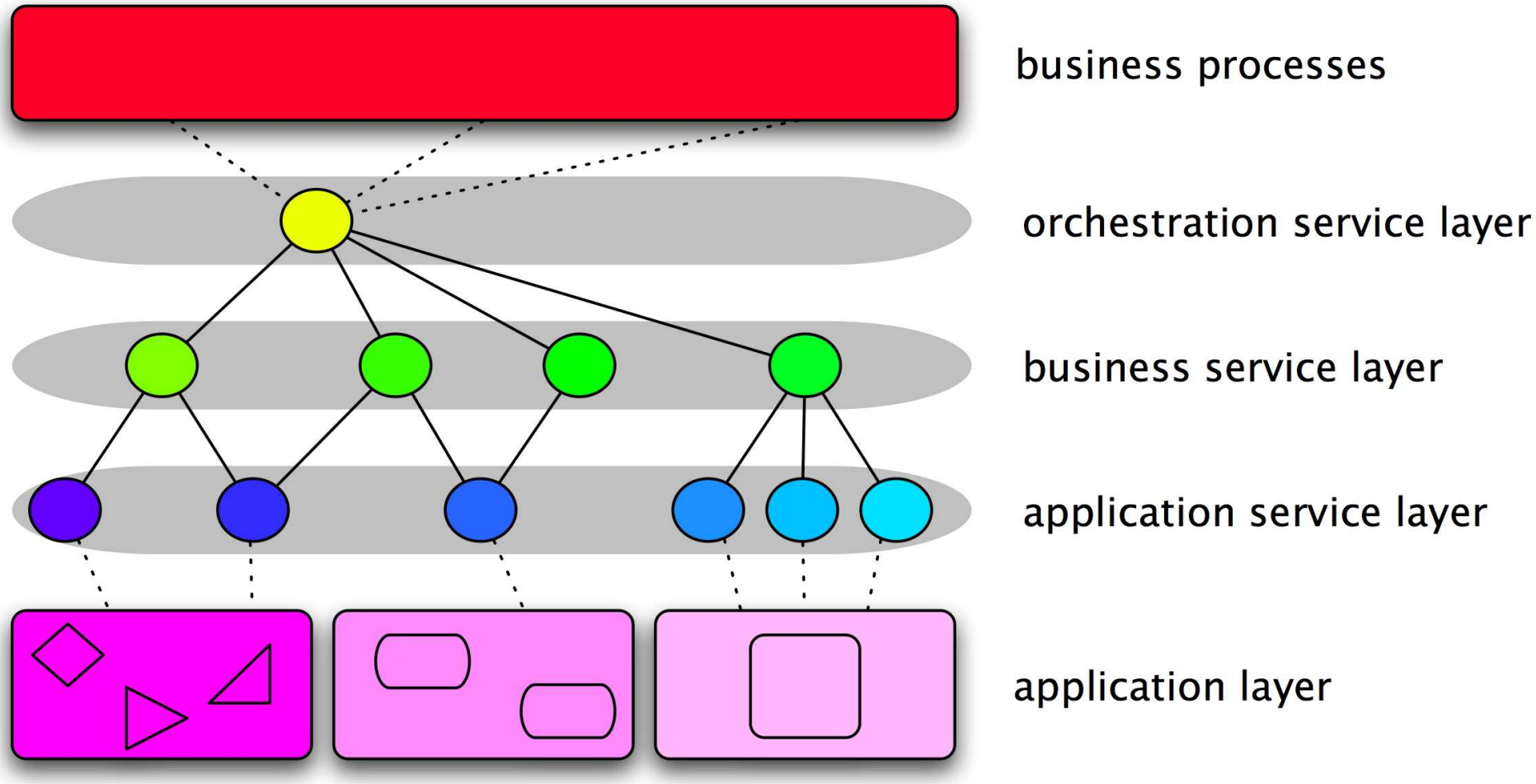
- arguments from Josuttis, *SOA in Practice*

1.1 Before SOA: silos



Separate fiefdoms, controlling independent systems.

1.2 With SOA: services



1.3 Organizational structure

- refactoring of fiefdoms:
  - backend departments
  - cross-domain departments
  - frontend departments
  - “solutions managers”
- requires collaboration and trust

1.4 Conway's law

> Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.
>
> (Melvin Conway, *How Do Committees Invent?*, Datamation Apr 1968, http://www.melconway.com/law/)

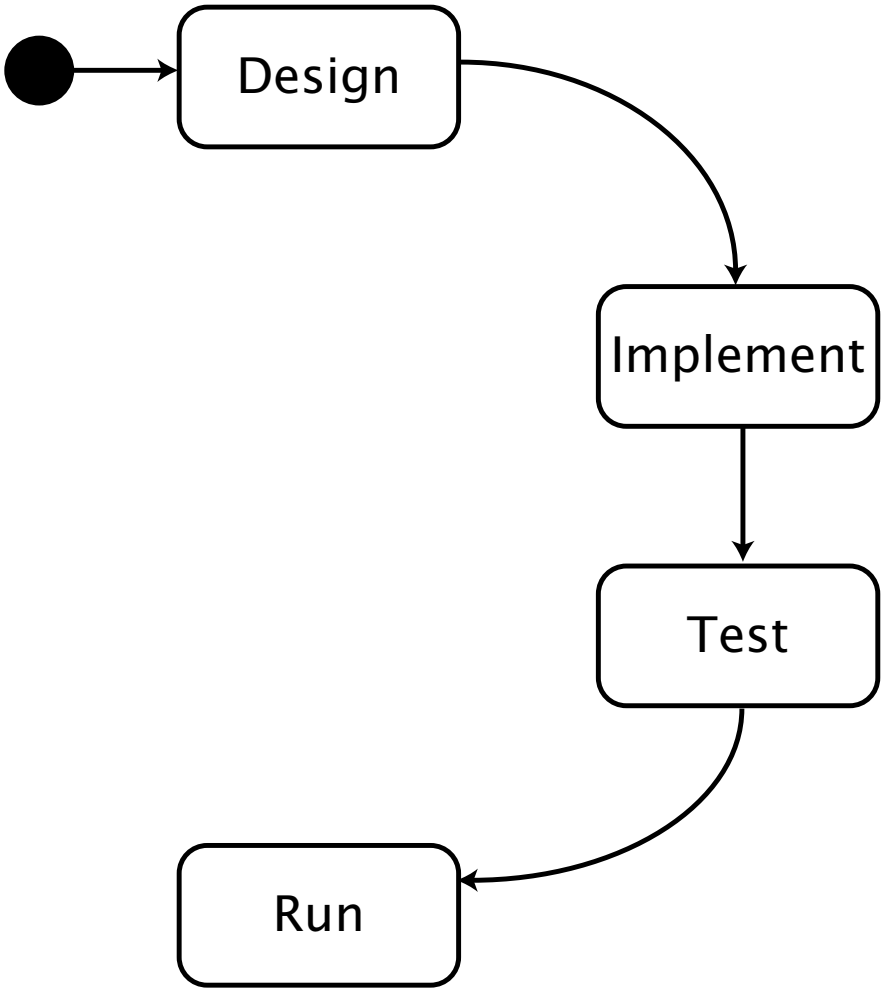
Popularized and named by Fred Brooks in *The Mythical Man-Month*: "If you have four groups working on a compiler, you'll get a 4-pass compiler."

Conversely...

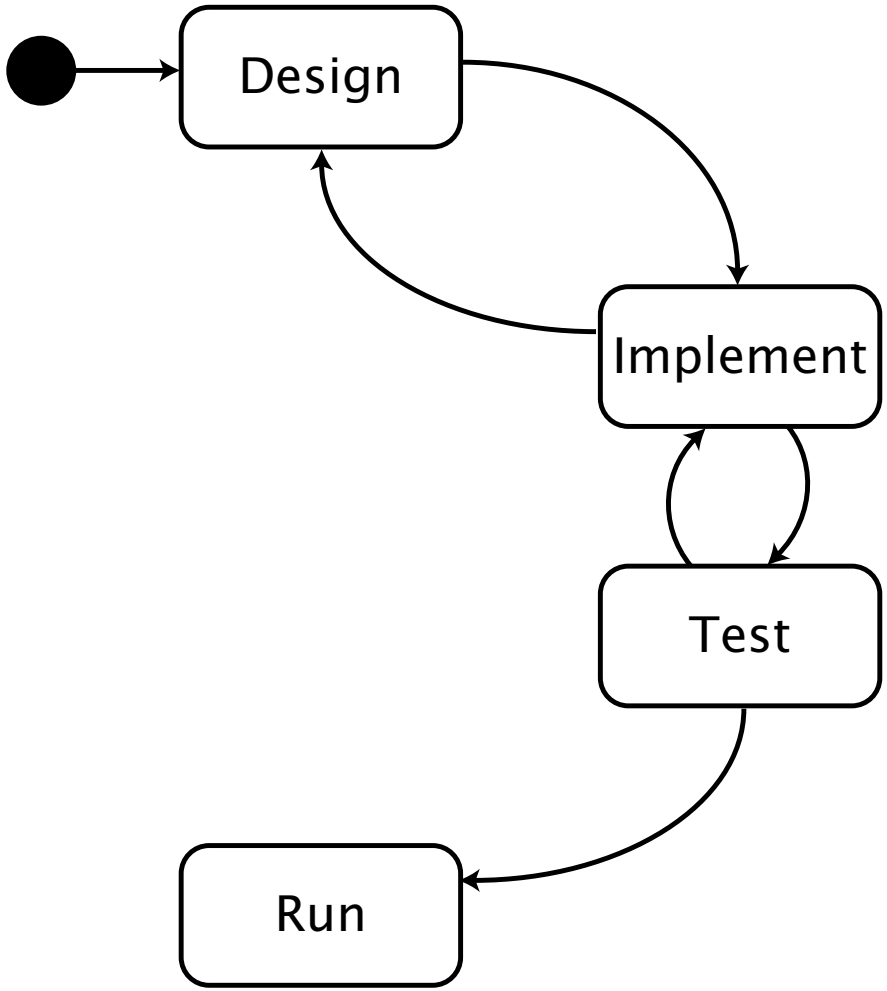
2 Lifecycle

- services are software
- standard software lifecycle practices
- but services also part of larger organizational processes
- therefore some differences

2.1 Core phases

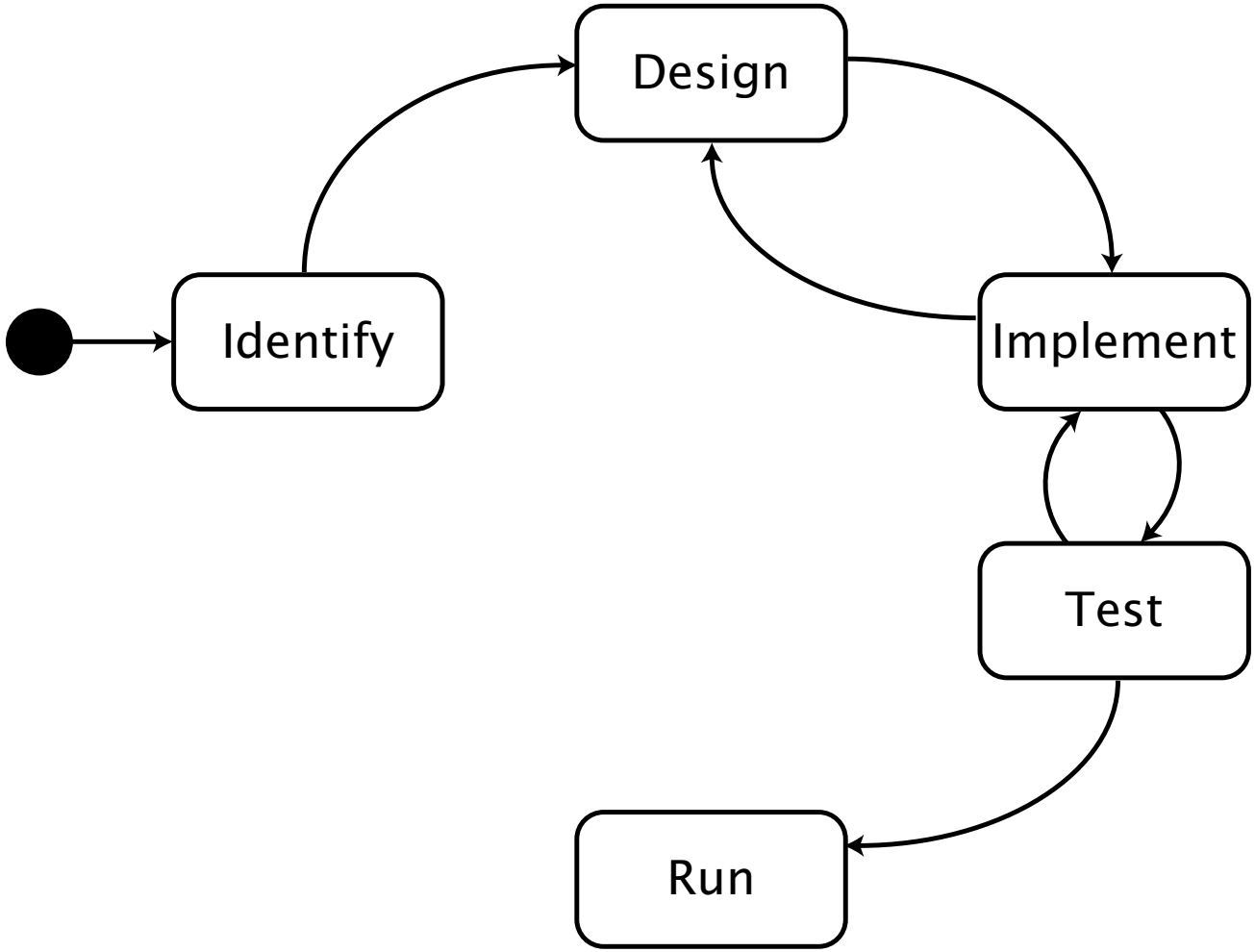


2.2 Iterative development in context



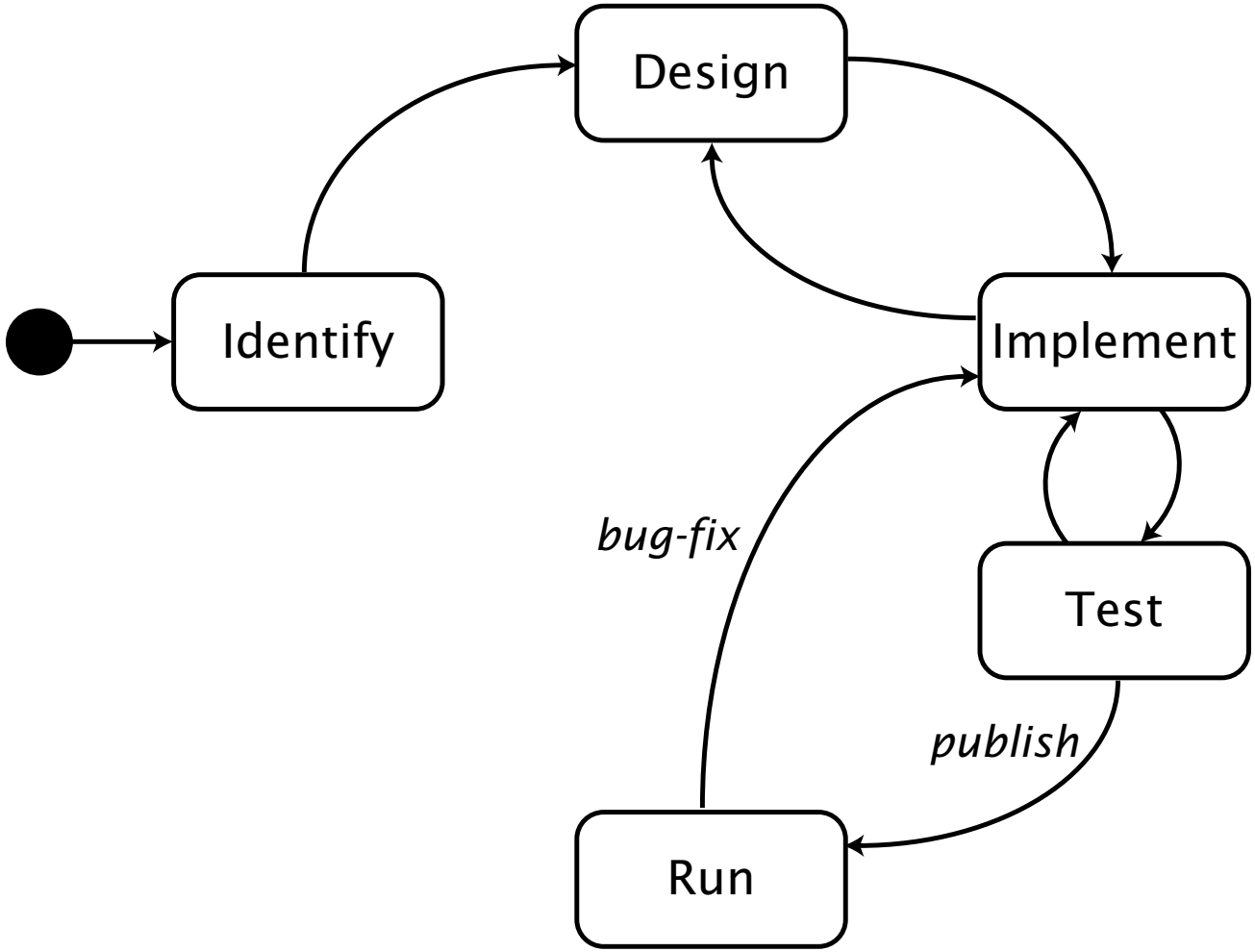
Some decisions will need revisiting.

2.3 Identifying services



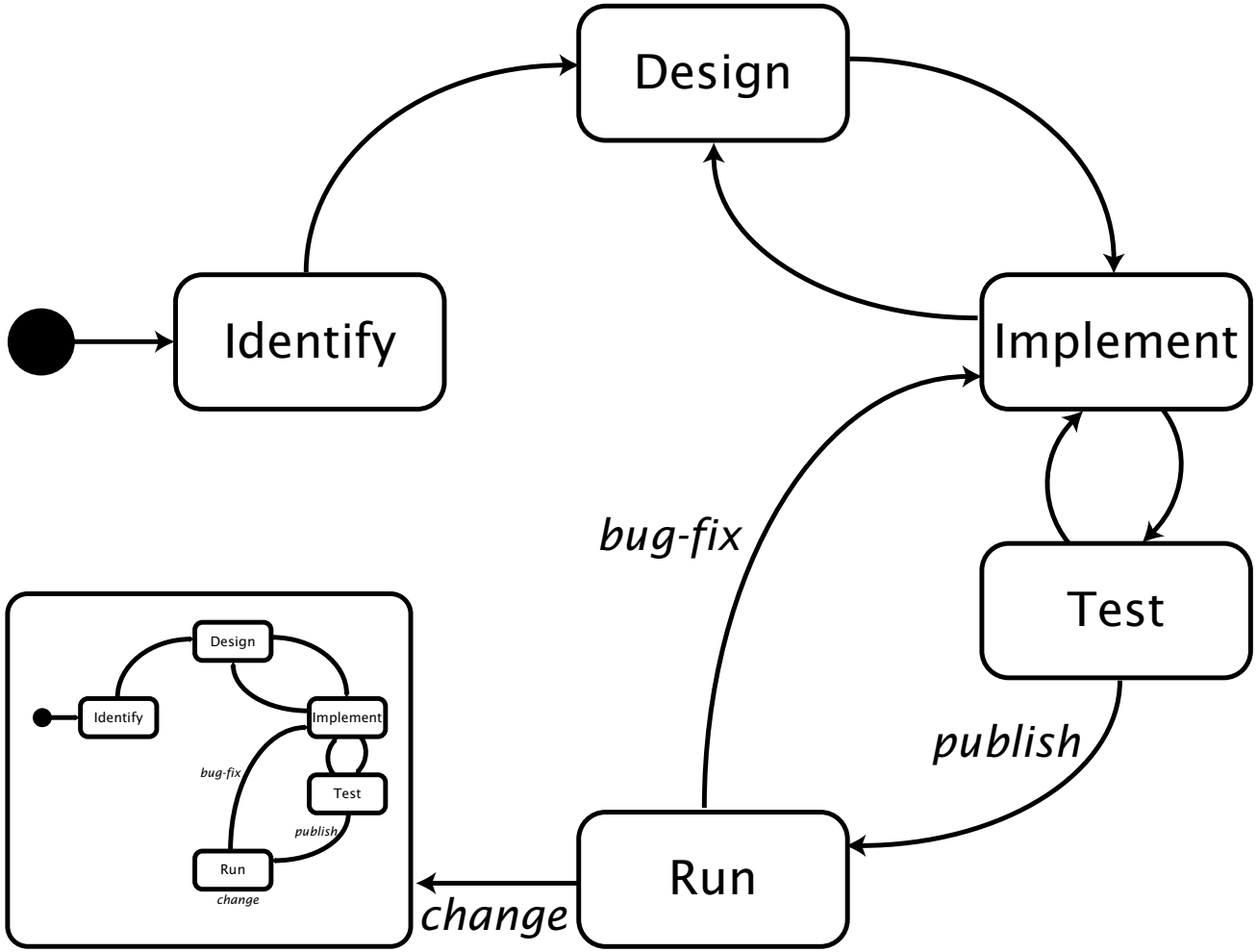
Requirements, analysis, portfolio management...

2.4 Fixing services



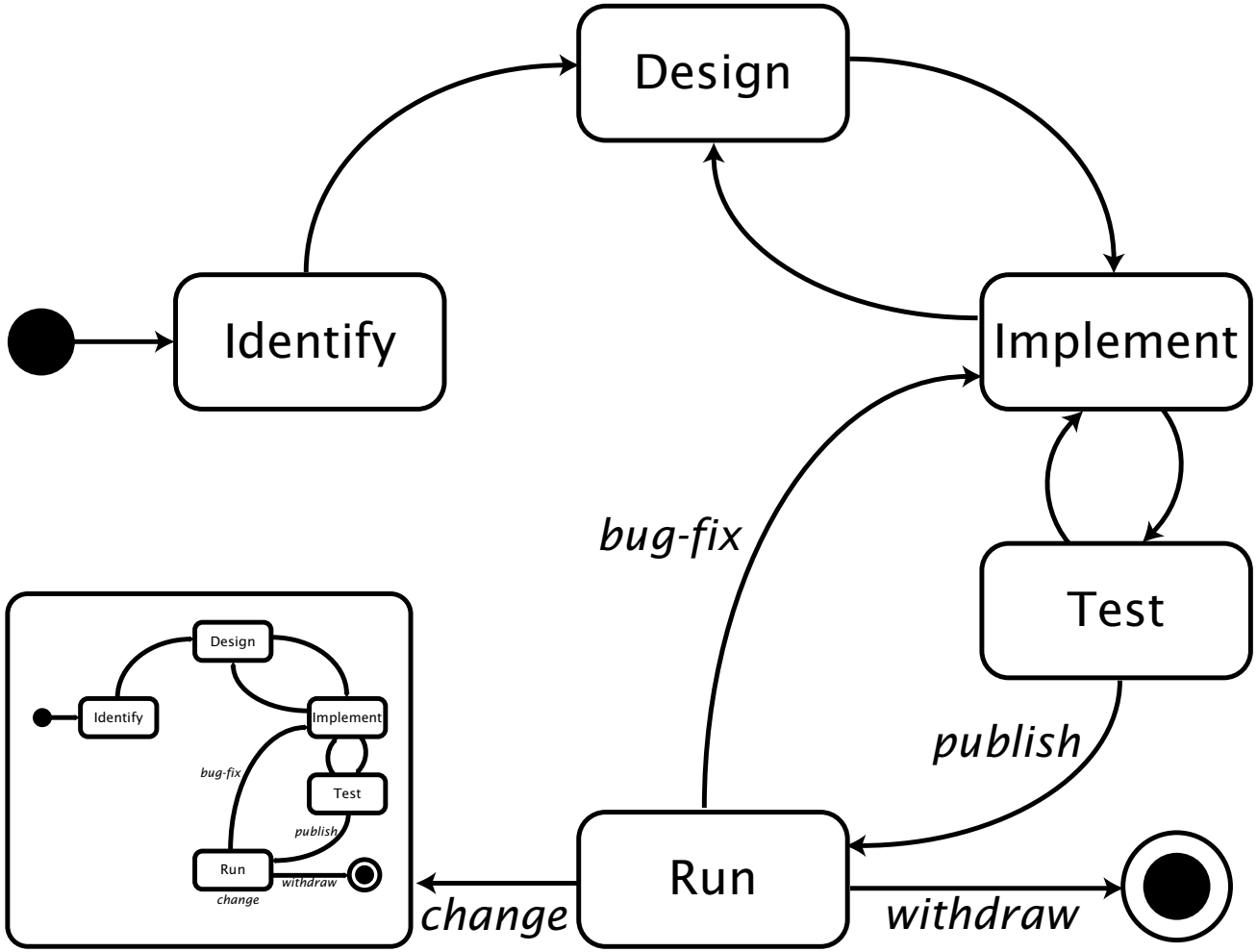
If it's just a bug-fix, fix it in-place.

2.5 Modifying services



If it's in any way significant, make a new service.

2.6 Withdrawing services



deprecate → monitor → persuade → delete

3 Versioning

- waterfall principles giving way to agile
- SOA is designed for flexibility
- especially for large systems, cannot fix requirements
- *but still* people want interfaces to be stable
- contracts and projects depend on it
- not feasible to refactor large systems
- so stability is also very important

3.1 Simple domain-driven versioning

- there are complex approaches to versioning, with automatic updates
- simpler approach often suffices:

  every published modification yields a new service

- make version number part of the service identifier:
  *GetCustomerData*1, *GetCustomerData*2
- (or version number might be a parameter, or contextual)

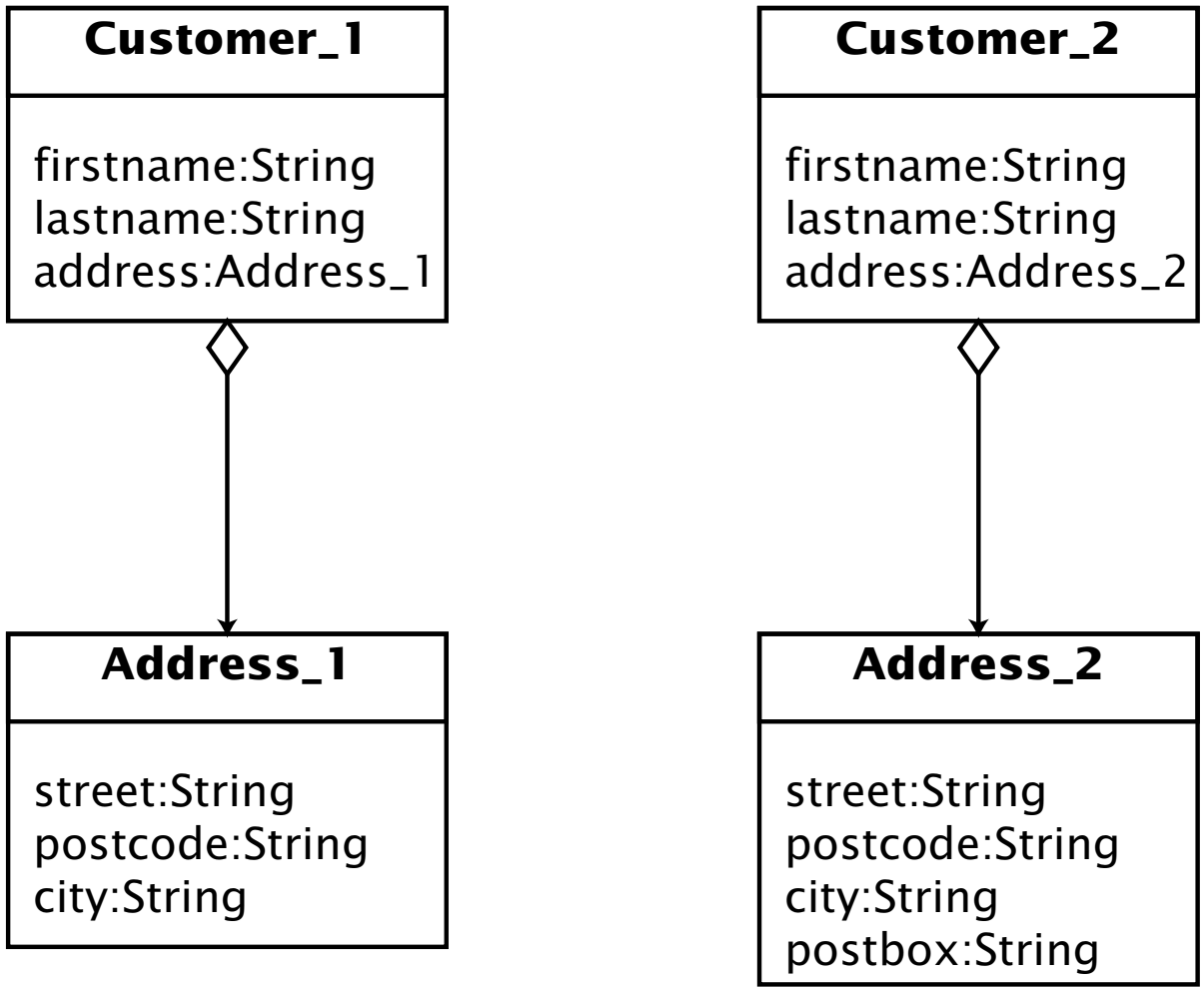
3.2 Backwards compatibility

- relax the rule for bug-fixes
- encourage compliance for “backwards-compatible” changes
  - might not be so compatible, eg if SLA changes
  - maybe changes in datatypes...
  - besides, any bug-fix entails some risk
- require compliance for incompatible changes
- actually, bug-fixes might be (even incompatible) changes

3.3 Versioning of datatypes

- modified services often require modified datatypes
- these differences propagate through aggregation
- sooner or later, consumers will have to deal with multiple versions of ‘the same’ datatype
- three approaches:
  - really use different types
  - use one common type, with optional values
  - don’t use types, make it generic

3.4 Propagation of datatype versions



4 Governance

- SOA has technical aspects
- but it isn't a product: you can't buy it
- how can you introduce it?
- how can you govern its application?

4.1 Non-technical governance issues

- *visions, objectives, business case, funding model*

  why are we doing this? how will we pay for it?

- *reference architecture*

  fundamental decisions: preferred technology, message exchange patterns, metamodel, etc

- *rules and responsibilities*

  who drives and cares about issues

- *policies, standards, formats, processes, lifecycles*

  decide and document, in standard notations

4.2 Technical governance issues

- *documentation*

  important for transparency; promotes non-technical issues

- *service management*

  repositories and registries for services and contracts

- *monitoring*

  conformance to policies, meeting SLAs, preparing for withdrawal

- *change and configuration management*

  the usual tools

4.3 SOA step-by-step

- *understand it*

  what's the benefit? do you have support?

- *carry out a pilot project*

  try it out, small but not too small; should have some value

- *second and third projects*

  determine what was specific to pilot; refactor these reference implementations

- *develop a general strategy*

  establish useful lessons and principles

Don't forget documentation and retrospective reviews, at every step.

4.4 Establishing SOA: four approaches

- *developer-driven*, grass-roots

  leads to technological experience; likely to be uncoordinated

- *business-driven*

  proof of concept helps adoption; limited benefit from early projects

- *IT-driven*

  effective for infrastructure; focus on technical aspects

- *management-driven*, top-down

  coordinated, driven by business priorities; expensive, disruptive, risky

Really need a combination of them all.

Index

Service-Oriented Architecture

Monday	Tuesday	Wednesday	Thursday	Friday
Introduction Components	REST	Composition	Architecture	Engineering
coffee	coffee	coffee	coffee	coffee
Components	REST	Composition	Architecture	Conclusion
lunch	lunch	lunch	lunch	lunch
Web Services	Qualities	Objects	Semantic Web	
tea	tea	tea	tea	
Web Services	Qualities	Objects	Semantic Web	