Mary Poppendieck Discusses Containers, Microservices and Contract Tests

At Craft Conference 2015 in Budapest, [Mary Poppendieck](http://www.poppendieck.com/people.htm) discussed the ‘**new software development game**’ and offered advice on how best to utilise containers, microservices and consumer-based contract tests  to lower friction and limit risk within software  systems.

Poppendieck, co-author of the ‘**Lean Software Development**’ book series, opened the talk by stating that  software developers  create [complex systems](http://en.wikipedia.org/wiki/Complex_systems). As Fred Brooks identified in his seminal book, ‘[The Mythical Man Month](http://en.wikipedia.org/wiki/The_Mythical_Man-Month)’, not only are software systems complex, but the complexity scales non-linearly with size. Poppendieck stated that one of the core goals of the talk was to explore the notion that a ‘**good**’ complex system has the properties of '**low friction**' and limited risk. The remainder of the talk  examined this from the perspective of the historical evolution of hardware and software, software architecture, engineering principles, or ganisational structure, and designing for reliability.

**Software developers create complex systems. With any complex system we want to lower friction, and limit risk...**

Poppendieck examined the evolution of computer hardware, drawing from her own experience of developing software for machines such as the [SCC 4700](https://archive.org/details/bitsavers_sccSCC4700ription_8568321) and [HP-1000](http://en.wikipedia.org/wiki/HP-1000/RTE). It was proposed  that although hardware has developed greatly over the past thirty years, software may not have evolved much from the [FORTRAN](http://en.wikipedia.org/wiki/Fortran) programming language.

Poppendieck demonstrated that hardware scales best through abstraction and miniaturisation. For example, the creation of a computer mother board that contains multiple sockets for interchangeable components, and the results of [Moore's law](http://en.wikipedia.org/wiki/Moore%27s_law) that has enabled an increasing number of transistors to be included per square inch on an integrated circuit. However, software scales best through federation and wide social participation. The Internet is a prime example of this. Poppendieck suggested that we should look to the history of scaling software in order to identify the best way to scale applications in the future.

**Don’t look to hardware for lessons in scaling - look to software.**

**Software scales best through federation and wide participation, for example, the Internet...**

Poppendieck discussed that traditional software has often been built using a monolithic architecture, which in part is a result of using an underlying database that becomes a ‘massive coupler’. The database often contains the canonical representation of business entities, and as such, any changes to the schema cascade throughout the connecting software systems. [Microservices](http://martinfowler.com/articles/microservices.html), as discussed in Sam Newman’s ‘[Building Microservices](http://shop.oreilly.com/product/0636920033158.do)’ book, may be a more appropriate architectural approach to reducing the friction imposed by a monolithic code base, but the use of this pattern can also increase risk.

Many of the benefits of microservices were also touted as benefits of objects and [object-oriented programming](http://en.wikipedia.org/wiki/Object-oriented_programming), but the two approaches also share many of the same potential problems, for example, dependency management.

Objects were going to save the world, and all the objects were going to be independent. But, we created this wiring dependency hell. What’s to keep that from happening?

Poppendieck suggested that the creation of services following the [domain-driven design](http://en.wikipedia.org/wiki/Domain-driven_design) (DDD) principles of a [bounded context](http://martinfowler.com/bliki/BoundedContext.html) are beneficial, as this way services match something in real world, and can be easier to reason about. Consolidation or supervisor services can also be used to ensure correct behaviour of down-stream services. Development teams will also require a lot of discipline. For example, access to a service should only ever be conducted through it’s standard interface, and teams must pay attention to the impact on surrounding services of changes made in their own service. The teams creating services must also be cross-functional, as a large percentage of problems can originate from operational issues.

**Amazon found that something like 60% of problems came from deployment and afterward. So why would you not have ops as part of your team?**

Caution was advised in respect to using a microservice architecture when building greenfield software systems. It is typically easier to build a monolith, and then move to microservices. Quoting [Martin Fowler](http://martinfowler.com/articles/distributed-objects-microservices.html), Poppendieck stated that it is easier to refactor a monolith than microservices if the initial partitions were not created properly. Building a monolith allows a team to learn where the natural partitions are, and identify points of friction. The ‘integrated federated architecture’ of the Swedish-built [Saab JAS 39 Gripen aeroplane](http://en.wikipedia.org/wiki/Saab_JAS_39_Gripen) was examined, and the mechanical modularity of the plane and isolation of its computer systems cited as good examples of building a complex system in the real world.

Friction is often seen as the 'large system disease’, and Poppendieck demonstrated that the historical loading and unloading of cargo ships was a high friction activity until [shipping containers](http://en.wikipedia.org/wiki/Intermodal_container) were introduced. The shipping container metaphor was matched to the emergence of [LXC](https://linuxcontainers.org/)-based containers within modern software deployment, and the factors of portability, consistency, isolation, easy of use, and utilisation were suggested as mechanisms to lower friction with the delivery of software. However, the use of containers alone does not limit risk.

Poppendieck suggested that risk can be lowered in container-based microservice applications through the use of [consumer-driven contracts](http://martinfowler.com/articles/consumerDrivenContracts.html) (CBCs), for example, using the [PACT](https://github.com/realestate-com-au/pact) framework. The use of CBCs lowers risk by enabling breaking changes to be detected without running time-consuming end-to-end tests, and still maintains low friction. The use of [continuous delivery](http://martinfowler.com/books/continuousDelivery.html) was also strongly recommended, as changes within a complex system can be very difficult to reason about.

**Don’t smash things into a complex system. Instead poke the system. Lots of little deployments are  good  if you want reliability**

In relation to organisational structure, Poppendieck explored the difference between developers and operationers people using the [regulatory fit theory](http://en.wikipedia.org/wiki/Regulatory_focus_theory#Regulatory_fit_theory). Developers are typically promotion-focused and set aspirational goals, where as operators are typically prevention-focused as set safety goals. Because of these competing focuses, shared responsibility and situational awareness must be developed across the team, where there is 'no success unless everyone succeeds'. The notion of a winning team was also discussed, and the use of the historically proven [military model](http://en.wikipedia.org/wiki/Military_organization) of squads, companies and [command intent](http://en.wikipedia.org/wiki/Intent_%28military%29) was explored. [Spotify](https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/) is an example of an organisation that has adopted this model.

Poppendieck concluded the talk by discussing reliability in the context of complex systems. In large scale systems, such as the software systems being developed now, some degree of failure is guaranteed. Accordingly, software development teams must plan for this. Implementation of this planning can be achieved through code and operational practices, and techniques such as duplication, isolation, eventual consistency and resilience ‘[game day](https://queue.acm.org/detail.cfm?id=2371297)’ testing were strongly recommended.

Additional details on Mary Poppendieck’s talk “[The new software development game: Containers, microservices and contract tests](http://craft-conf.com/2015#speakers/MaryPoppendieck)” can be found on the Craft Conference website.