Bureaucracy in Systems

Measuring System Complexity by the Amount of Digital Paperwork

Michael Roitzsch
Barkhausen Institut
Dresden, Germany

Abstract

Managing complexity in a systems architecture is often regarded as a point on a spectrum between a monolithic system and a microkernel-based system. On a monolith, all complexity and functionality is collected in a few components or — in the extreme case — in a single one. In a microkernel-based system, components are small and have reduced purpose, simplifying the individual components. However, we argue that complexity was not reduced but merely moved from a single component with inherent complexity to the complex interactions between many small components. In order to make a fair argument, we must not only consider the size of components, but also the complexity manifested in the interfaces between them. We therefore propose to learn from bureaucratic process and apply the experienced bureaucracy of component interactions as a novel complexity metric we call *Trans-Compartmental Bureaucracy (TCB)*. We show how this metric leads to different characterizations of system architectures compared to existing complexity metrics.

1 Introduction

Complexity in systems is growing to a point, where it has become of paramount importance for system architectures to successfully manage this complexity. Complexity is being accrued in all layers of the systems stack: Hardware is increasingly complex [1, 2] and must be managed by the operating system. For these operating systems, complexity-induced exploitable bugs have become a standard concern [3]. Finally, local and distributed application runtimes accrue complexity due to demands for non-functional properties like scalability, low tail-latencies, and fault tolerance [4].

To manage this complexity, two architectural design patterns are considered as system archetypes: the monolith and the microkernel-based system. Monoliths implement system functionality in a few or one large component, the main example being the Linux kernel. This single component accrues a lot of privilege, since every subsystem inside it has full control over the entire system. As this is considered a security downside [5], the microkernel design has emerged. Components in microkernel-based systems have a clear focus and are individually much simpler. However, the level of interaction between components is higher than in a monolithic system.

Which of these extremes fares better is not immediately apparent. While individual components are expected to be less complex in microkernel-based systems, there are more of them. Therefore, one can argue that complexity has moved from being embedded within one large component to the interactions between many small components [6]. To fairly compare both solutions, a complexity metric is needed that covers both aspects: component-intrinsic and component-interaction complexity. For complexity intrinsic to individual components, systems research typically uses source lines of code as a proxy metric. However, the situation is less clear for interaction complexity.

In this work, we consider a novel, unifying complexity metric covering both component-intrinsic and component-interaction complexity. We propose a design for this metric that is inspired by the experienced bureaucracy in administrative processes. A process implemented by a monolithic administrative entity is experienced by overcomplicated paperwork with lots of mysterious and unexpected effects. A process implemented by a multitude of small administrative entities is experienced by seemingly pointless back-and-forth between these entities, often resulting in very little experienced progress. We believe this is a good starting point that a metric design for digital systems should be based upon.

In the following, we compare monolithic and microkernel-based systems in greater detail (Section 2). We then map these system archetypes to bureaucratic processes (Section 3), deriving our proposed complexity metric, which we call *Trans-Compartmental Bureaucracy (TCB)*.[1] By holistically applying this metric to the entire systems stack of hardware, operating system, and application runtime, we discuss how TCB allows a nuanced comparison of system designs (Section 4).

2 Of Monoliths and Microkernels

Monolithic and microkernel-based designs are two extreme cases on a spectrum of possible system design points. However, it is educational to consider them as the two archetypes of system design as actual systems like Linux or the L4 microkernel family clearly gravitate towards their respective extreme case. Figure 1 illustrates the main difference of these system designs in dealing with complexity.

In a monolithic system, all functionality is aggregated in one large, centralized component. This component consequently provides all this functionality to clients directly through its public interface. While offering a single point of service, the interface consequently becomes rich and diverse. As features get added, more internal complexity is expressed

[1] Yes, TCB is already taken. Yes, this is a bad idea. You're reading a WACI paper.

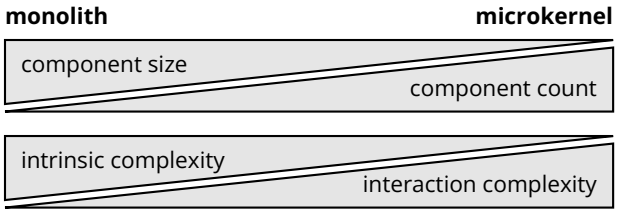

Figure 1. Properties of monolithic and microkernel systems

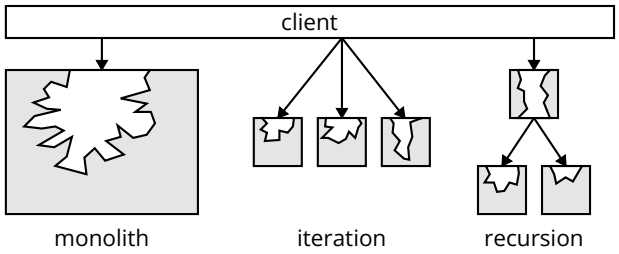

Figure 2. Complexity measured as interface boundary area

by this interface, leading to baroque constructions like the ill-famed `ioctl()`.

In a microkernel-based system, functionality is split across multiple interacting components. For example, reading from a file is one system call in a monolith, but can involve file system, buffer cache, memory pager, block device layer, and device driver components. These components may recursively invoke each other, or ask the client to iteratively invoke a combination of components in the right order. While each of the involved interfaces is focused and thus simpler compared to a monolithic system, it takes a multitude of them to get meaningful work done.

One can make the claim that going from a monolith to a microkernel, complexity just moves from a single complex component to the complex interactions between many components. Which situation is better is not immediately obvious. Consequently, literature has reported contradicting results on this question [5, 6].

3 Bureaucracy as a Complexity Metric

To compare the overall complexity of systems, we need a measure covering both component-intrinsic and component-interaction complexity. Such a metric does not appear to exist. The often used source lines of code serves as a proxy for component-intrinsic complexity only. We propose to construct a new metric by taking inspiration from bureaucracy: the more bureaucratic an administrative process feels, the more complexity is indicated.

Translated to the monolith and microkernel archetypes: A monolith exposes its internal complexity through one giant interface, with different functionality offered at different levels of abstraction. This large interface is onerous and burdensome to navigate. Lots of hidden state can lead to unexpected outcomes. Mistakes in using the interface or in the implementation behind the interface can have wide-reaching consequences, as you always interact with the full power of a centralized system. The administrative analogy is an Orwellian 1984-style single-body government.

A microkernel on the other hand grinds you down by bouncing you from component to component. Interactions with these components are each much simpler, but they end up being mostly meaningless as individual components cannot achieve much by themselves. The system seems constructed to capture you in endless interactions and protocol busywork. The administrative analogy is a Brazil-style[2] dysfunctional Rube Goldberg construction of government.

To measure systemic bureaucracy in this way, we propose to develop tools that analyze the penetration depth into a component's internal state afforded by its interface. In other words, how deep inside the component can I reach by using its interface. We imagine such a tool to operate similar to existing taint analysis tools, measuring the boundary area of reachable internal state. For a monolithic system, this will lead to a high complexity measure as the single monolithic component exposes a lot of state through its interface (Figure 2, left). However, for the interacting components of a microkernel, two effects will play a role: First, the large number of individual interfaces available to clients adds up (Figure 2, middle). Second, components recursively invoking other components will have their reachable state boundary stretch across components, thus indicating a higher overall complexity (Figure 2, right). It is this component-crossing measurement that led us to name this proposed metric *Trans-Compartmental Bureaucracy (TCB)*.

4 Discussion

How would we expect this novel metric to behave in practice? When purely looking at the operating system kernel, we would expect a microkernel to indeed have a lower TCB measure compared to a monolithic kernel, given its reduced interface and smaller internal state. However, a user space component that is large in overall code size, but small in terms of exposed interface is an example, where traditional complexity metrics such as source lines of code and TCB will differ. And indeed, a large component that has no access to other downstream components and thus does not recursively expose their interfaces can be considered as unbureaucratic and thus unproblematic for overall system complexity, despite its size.

Therefore, we believe the idea of a bureaucracy-guided complexity metric is valuable. With TCB, we have proposed a sketch as to how such a metric could work. It can be applied to analyze and compare component-based application runtimes or microkernel-based system designs. In the light of Fiedler et al. [2], TCB can also measure the software/hardware interface, opening up additional research avenues.

[2]The Terry Gilliam movie, not the actual country.

References

[1] Andrew Baumann. Hardware is the new Software. 16th Workshop on Hot Topics in Operating Systems (HotOS), Whistler, BC, Canada, May 2017

[2] Ben Fiedler, Roman Meier, Jasmin Schult, Daniel Schwyn, and Timothy Roscoe. Specifying the de-facto OS of a production SoC. 1st Workshop on Kernel Isolation, Safety and Verification (KISV), Koblenz, Germany, October 2023

[3] Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. Faults in linux: ten years later. 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Newport Beach, CA, USA, March 2011

[4] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. What bugs cause production cloud incidents? 17th Workshop on Hot Topics in Operating Systems (HotOS), Bertinoro, Italy, May 2019

[5] Simon Biggs, Damon Lee, and Gernot Heiser. The jury is in: monolithic OS design is flawed: microkernel-based designs improve security. 9th Asia-Pacific Workshop on Systems (APSys), Jeju Island, Republic of Korea, August 2028

[6] Hugo Lefeuvre, Vlad-Andrei Bădoiu, Yi Chien, Felipe Huici, Nathan Dautenhahn, and Pierre Olivier. Assessing the impact of interface vulnerabilities in compartmentalized software. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, March 2023