

Retrofitting Rigour

Peter Sewell

University of Cambridge

`Peter.Sewell@cl.cam.ac.uk`,

<http://www.cl.cam.ac.uk/users/pes20/>

Abstract. We rely on an enormous number of software components, which continue to be developed —as they have been since the 1940s— by methods based on a test-and-debug cycle and informal-prose specifications. Replacing this legacy infrastructure and development practices wholesale is not (and may never be) feasible, so if we want to improve system quality by using mathematically rigorous methods, we need a strategy to do so incrementally. A first step is to focus on the more stable extant interfaces: processor architectures, programming languages, and key APIs, file formats, and protocols. By characterising these with new precise abstractions, we can start to retrofit rigour into the system, using those (1) to improve existing components by testing, (2) as interfaces for replacement formally verified components, and (3) simply to gain understanding of what the existing behavioural interfaces are. Many issues arise in building such abstractions and in establishing confidence in them. To validate them against existing systems, they must be made executable, not necessarily in the conventional sense but as test oracles, to check whether observed behaviour of the actual system is admitted by the specification. The appropriate treatment of loose specification is key here, and varies from case to case. To validate them against the intent of the designers, they must be made broadly comprehensible, by careful structuring, annotation, and presentation. And to support formal reasoning by as broad a community as possible, they must be portable into a variety of reasoning tools. Moreover, the scale of real-world specifications adds further engineering concerns. This talk will discuss the issues of loose specification and some lessons learnt from developing and using our Ott and Lem lightweight specification tools, which are aimed at supporting all these pre-proving activities (and which target multiple theorem provers).

Acknowledgements Ott and Lem have been developed in joint work with Francesco Zappa Nardelli, Scott Owens, Dominic Mulligan, Thomas Tuerk, Kathryn Gray, and Thomas Williams, and with feedback from many other users. I acknowledge funding from EPSRC grants EP/H005633 (Leadership Fellowship) and EP/K008528 (REMS Programme Grant).