Software Engineering for Variability Intensive Systems

Foreword

John C. Grundy
Faculty of Information Technology
Monash University
Melbourne, Australia
john.grundy@monash.edu

Introduction

Once upon a time, computer systems were monolithic hardware and software beasts that had very limited end users, resided in one place, underwent limited change during maintenance, and were generally developed with top-down, waterfall-style approaches. As smaller, more diverse computing systems arose, and as end users and application domains became far more diverse, this approach failed to keep up with need. Computer systems needed to be deployed in ever more diverse environments. End users of a computer system – and their requirements and expectations of the system – may change greatly during its lifespan [1]. Very heterogeneous computing platforms, operating systems, programming languages, interaction technologies, integration approaches, and structuring and development of systems arose. These included diverse computing devices (cloud, HPC, desktop, laptop, tablet, phone, wearable, sensor, etc); dynamic programming languages; adaptive software architectures supporting run-time updates; web, mobile, Internet of Things, Virtual and Augmented Reality, ubiquitous interfaces; AI-based software; product line, service-oriented, micro-services, edge and fog computing-based architectures; and various agile methods and end user development support [1,4,5,6,9,10]. In all of these domains, the computing systems undergo sometimes very dramatic variability during their lifespan.

Building such systems is (very) hard [1,11]. They are inherently much more complex than traditional computing systems [12]. Development methods may need to include diverse stakeholders, including end users, who expect to be able to reconfigure their systems during use in various ways. Deployment environments, other applications to integrate with and all potential end users and their requirements might not even be known in advance. New architectures support greater variability but require improved modelling, development, implementation and testing practices. Systems don't operate in isolation and diverse interaction devices and deployment environments have resulted in much more complex scenarios of human-computer interaction and computer-computer interaction. Huge increases in compute and storage capacity, range of size and number of devices in a computer system, and concerns for energy efficiency, performance and reliability impact management of and capacity for variability. Increasing numbers of safety critical systems, including infrastructure, health, manufacturing, business, education, and personal usage make dependence on variability-intensive systems much more prevalent [3]. Systems are under almost constant threat from cybersecurity attacks of various sorts. Systems that are inherently meldable at run-time have significant advantages – but also significant disadvantages – over traditional systems.

Why Variability-Intensive Systems?

Given how hard such systems are to engineer, in general, why would we want to make use of such challenging things? Increasingly there is an expectation that computing systems will "vary" throughout their lifecycle [1,3,7,11]. This variability takes on various forms:

- Ability to modify a system at run-time e.g. patch code while it runs, integrate new services or make use of a different service;
- Ability to add new components to a system at run-time e.g. new sensors for a smart home, make use of new visualisations for a data analytics platform;
- Ability to tailor system to diverse application domains, users, integrations, etc e.g. product line templates instantiated to different using company contexts, different security solutions available depending on organisational deployment context;
- Deploying software systems on heterogeneous, unpredictable platforms e.g. deploying on different mobile devices, using different sensors and sensor networks, handling different interaction technologies (AR/VR, gesture, voice, movement, location, …);
- Handling evolution of organisation, team, software, environment e.g. run-time update of code to patch security bugs, adapting user interface to handle composition of different systems at runtime, architectural re-engineering due to change of organisational enterprise architecture, global software engineering across distributed teams using different agile processes and tools;
- Various combinations of all of the above software system variability.

In order to support these sorts of variability, software systems need to be conceived, designed, architected, implemented, tested, deployed, maintained and evolved with intensive, multi-faceted variability as a first class characteristic [8,9,12].

Key Challenges to Engineering Variability-Intensive Systems

Complexity – Variability-intensive software systems are by their very nature more complex than traditional computing systems that do not undergo such variability at design, implementation and/or run-time. Engineers have to manage this additional complexity, which can take the form of complex configuration parameters, dynamic models-at-run-time updatability, additional abstractions to support domain-specific tailoring, and more challenging testing regimes and deployment infrastructure. Tools to support managing this complexity are also needed.

Development processes - Development methods differ from traditional software systems engineering where teams developing variability-intensive systems often need a greater range of technical skills than conventional systems. Highly variable requirements may need to be accounted for, dynamically reconfigurable architectures be used, a range of diverse stakeholders and including end users may need to be supported in configuring systems at run-time..

Deployment environments – one emergent theme of recent years has been the variability of deployment environments themselves, let alone the systems deployed in them. The other applications a system must integrate with and all of its potential end users and their requirements might not even be known in advance. New architectures support greater variability but require improved modelling, development, implementation and testing practices.

Integration and interaction – we have moved to engineering systems of systems and the use of diverse interaction devices and deployment environments have resulted in much more complex challenges for developers. A wider range of end user input and output devices, and end user expectations, has increased the issues of usability. This is especially challenging when user interfaces are composed from multiple sources and must be deployed on varying platforms.

Quality of Service – QoS issues are challenging in many software systems but variability intensive systems introduce new dimensions to these. As the system may undergo significant change at run-time, handing diverse compute power, storage, networking, composition and deployment are all far more complicated – how do you meet energy and throughput targets when services change at run-time, potentially to slower, more power-hungry ones? How are constraints around composition and deployment - such as technology platform, user interfae, security model and reliability – met under dynamic, run-time re-composition and re-deployment? Popular QoS areas for research and practice include performance, energy efficiency, reliability, robustness and self-healing capacity.

Safety-critical Systems – Many safety critical domains have a need for variability-intense computing solutions. Particular ones of current and emerging interest are smart cities – smart transport, utilities, housing and health; defence, advanced manufacturing especiallty using robotics and mechatronics, and next-generation global business solutions requiring near real-time transaction processing at scale. In the later, the growth of distributed ledger-based systems (blockchains) are an increasingly popular and important approach to providing distributed contract management for a wide variety of domains.

Cybersecurity - Systems that are inherently variable in nature can be thought of as having significant advantages – but also significant disadvantages – over traditional systems. This includes the ability to self-heal and adapt on-the-fly while in use to mitigate emerging security threats and vulnerabilities. However, should the variability support in the software be compromised, then the application may be damaged in ways not feasible for less variable-intensive systems.

Current and Future Trends

Some key trends that are emerging and that will be important for Software Engineers to consider in engineering their next generation variability-intensive systems include:

- *Micro-services and DevOps practices* – these areas have become popular research topics and emerging SE practices. Their application to variability-intensive systems is still however under both research and practice exploration with many unanswered questions around how to (de)compose services, test, deploy, update, and maintain such complex systems.
- *Internet of Things* – similarly, there is a burgeoning interest in IoT systems and many exhibit high variability. For example, a smart home may have new sensors added, multiple networks with mobile devices coming and going, various continuous and discrete data streams, and different users at different times with differet tasks and preferences. How we design, build, compose and maintain highly diverse and dynamic IoT solutions is subject of much current and future investigation.
- *AI-based solutions* – software engineering challenges of AI-based systems are increasingly popular areas for research and many new practical solutions are being deployed in practice. However, variability introduces many additional complexities,

including how to adapt AI-based solutions to varying datasets, repeated re-training, addressing bias, and providing human-in-the-loop support.

- *Self-Securing systems* – variability-intensive software should provide a number of advantages over less meldable systems in terms of addressing emergent security threats and vulnerabilities. However much further work is needed on designing, architecting, building, deploying and testing such systems.
- *Augmented Reality* – while AR-based systems are not necessarily variability-intensive, using AR interaction devices along with dynamic composition, AI, run-time emergent requirements, and IoT environments offers many new advances ion human-computer interaction. AR systems are also an example of context-aware systems where changing context – user, room, task, collaboration – often significantly perturb the interface, especially when using haptic and other direct feedback support.
- *Combinations of the above* – one can see that a future generation variability-intensive system could easily exhibit most if not all of the above features.

This book contains many fine papers detailing the rationale and history for variability-intensive systems, different kinds of variability-intensive systems, key challenges in engineering such systems, and emerging future directions. These include more theoretic treatment of the subject providing a sound foundation for both practice and research, emerging technology trends and approaches, and case studies of successful development and deployment of such systems. I do hope that you find the book valuable in your own practice and research in this exciting domain.

References

1. Almorsy, M., Grundy, J., & Ibrahim, A. S. (2014). Adaptable, model-driven security engineering for SaaS cloud-based applications. *Automated software engineering*, *21*(2), 187-224.
2. Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64-69.
3. Calinescu, R., Ghezzi, C., Kwiatkowska, M., & Mirandola, R. (2012). Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, *55*(9), 69-77.
4. Galster, M., Avgeriou, P., & Tofan, D. (2013). Constraints for the design of variability-intensive service-oriented reference architectures–An industrial case study. *Information and Software Technology*, *55*(2), 428-441.
5. Galindo, J. A., Turner, H., Benavides, D., & White, J. (2016). Testing variability-intensive systems using automated analysis: an application to android. *Software Quality Journal*, *24*(2), 365-405.
6. Grundy, J., & Hosking, J. (2002). Developing adaptable user interfaces for component-based systems. *Interacting with computers*, *14*(3), 175-194.
7. Hallsteinsen, S., Hinchey, M., Park, S., & Schmid, K. (2008). Dynamic software product lines. *Computer*, *41*(4).
8. Liaskos, S., Yu, Y., Yu, E., & Mylopoulos, J. (2006, September). On goal-based variability acquisition and analysis. In *Requirements Engineering, 14th IEEE International Conference* (pp. 79-88). IEEE.
9. Metzger, A., & Pohl, K. (2014, May). Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering* (pp. 70-84). ACM.
10. Passos, L., Czarnecki, K., Apel, S., Wąsowski, A., Kästner, C., & Guo, J. (2013, January). Feature-oriented software evolution. In *Proceedings of the Seventh*

International Workshop on Variability Modelling of Software-intensive Systems (p. 17). ACM.

11. Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, *4*(2), 14.
12. Zhang, J., & Cheng, B. H. (2006, May). Model-based development of dynamically adaptive software. In *Proceedings of the 28th international conference on Software engineering*(pp. 371-380). ACM.