Is Software Development Still Fun?

Steve Williamson FBCS, Head in Internal IT Audit for GlaxoSmithKline, explores the opportunities and challenges of modern software development and the need for enhanced skillsets to address a range of cyber-threats.

Is software development as much fun as it used to be? A couple of decades ago, a development team would use software engineering methods to design and build a bespoke, on-premise application, resulting in productivity benefits to the organisation (some of these applications are still in use today).

We live in the world of low code development and software reuse. There seems to be less need for algorithm design, as developers can just import code libraries or integrate run-time functionality through application programming interfaces (APIs). However, to say we have morphed from bespoke to self-assemble software would be a gross simplification, as today's developers need to spend a greater proportion of their time on protecting their application against cyber threats and technology failures.

Software re-use enables developers to integrate specialised functionality into their application with relative ease. Cloud service companies, such as IBM, Google and Microsoft have made their artificial intelligence (AI) functions available via APIs. This means that sophisticated features such as facial recognition and natural language processing can be integrated into homegrown applications (a subscription to the cloud service provider is normally required). For those who prefer a bit more visibility and control, open-source code libraries, which perform similar functions are available on Github.

The ubiquity of APIs, combined with exponential growth in computer processing power, has enabled the creation of applications to perform tasks which ordinarily require human judgement. Today, software development means you could create applications that would result in road death numbers slashed, cancer survival rates hiked, and people's eyesight saved. A class of AI is emerging, which can be characterised as high risk / high reward. These applications provide transformational benefits to society, but the consequences of failure are catastrophic.

High risk / high reward software

Autonomous vehicles are four-wheeled computers, which operate in real-time. They accept continuous input via sensors and output commands that cause physical movement. Their ultimate value will be the potential to dramatically decrease the number of road traffic accidents. Currently, over one million people a year die in traffic accidents globally. It may be many years in the future, but that number would reduce to a fraction if we get this right.

Another area of transformational benefit is the use of AI in healthcare. Moorfields eye hospital recently reported on the use of AI for detecting sight-threatening eye conditions by analysing retinal scans. The AI was found to be as reliable as a senior eye specialist. Another example is the use of IBM Watson in cancer care, which is currently being used in over 200 cancer clinics. It processes patient data and through its knowledge base, provides treatment recommendations along with a confidence score.

These are examples of high risk / high reward systems. In March 2018, a pedestrian was killed by Uber's self-driving car. The reported failure was a software bug (the sensors detected the pedestrian but failed to deploy the brakes). One clinic using Watson AI for cancer care reported that the system provided incorrect

and potentially fatal treatment recommendations. This was attributed to a poor training data set. AI failures are of varying degrees of severity.

In China, facial recognition is used to identify jay walkers. One famous businesswoman was caught jay walking and publicly shamed for her crime. The problem is, what the camera saw, was the victim's face – on an advert on the side of a bus.

When software is used for general convenience, such as creating a personalised music playlist, errors will cause frustration or amusement, but no real harm. In other situations, the consequences are severe.

Computer systems are good for search, sort, computation and pattern matching. Imitating human judgement is still a challenge and often results in unexpected outcomes. Cyber threats are an added complication. AI systems rely on a data set representing a body of knowledge: a data poisoning attack is when an adversary changes the data set to introduce bias or to cause incorrect outcomes.

When taking a risk-based approach to software development, you first think about the data – what is being collected, what will it be used for, where is it stored, who has access? Then, the impact of losing availability, integrity or confidentiality. For example, the collection of personal health data could turn a chat-bot into a high-risk application because a security breach could result in harm to the individual. The risks are even higher when AI is used for real-time automation in places such as power plants. Such systems need to be resilient to a range of threats and error conditions.

Risk aware software development

Risk-focused development methodologies should include risk assessments and threat modelling steps. The former focuses on the consequences of a system failure, the latter identifies likely threat scenarios.

Several threat modelling templates exist. A commonly used one is STRIDE (spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege). This is good for identifying attack vectors, but its focus is limited to security threats originated by bad actors. High impact system failures could arise from software bugs and the inability of a system to respond to unexpected input (you wonder how an autonomous vehicle would respond to hand signals from a London taxi driver).

Failure mode and effect analysis (FMEA) has a long history and can be used to cover a broader range of failure modes. Another technique for representing threats is misuse cases (like use cases but representing the adversary's requirement). These can take their place in the product backlog alongside user requirements.

Adopting a risk-based approach helps ensure that critical applications are built with appropriate rigor, whilst developers of lower risk systems are not burdened with such time-consuming ways of working. Nevertheless, a minimum baseline security standard should be applied to all development activity. Secure coding standards, such as those represented by the Open Web Application Security Project (OWASP) are commonly accepted standards for web and mobile development.

Nowadays, a failure to deliver secure software is like a car manufacturer producing a vehicle without airbags. Furthermore, privacy regulators are becoming less forgiving of poor security practices. The breach against Talk-Talk's web site in 2015, resulting from a SQL injection exploit, enabled attackers to gain access to customer records. The Information Commissioner fined Talk-Talk £400,000 for their 'failure to implement

the most basic cyber security measures'. Under the new General Data Protection Regulation (GDPR), which came into force in 2018, that fine would have been much higher.

Developer skillsets continually evolve

In days gone by, database developers would focus on schema design, transaction processing, synchronisation, role-based access models, etc. These design elements are still important, but data protection safeguards also need to be designed in, for instance, encryption, data masking, activity logging, database firewalls, etc.

The safeguards selected will be driven by the potential threats and data sensitivity. Conveniently, many of the required technical controls can be implemented through standardised functions available within the database management system (developers should never have the need to design their own encryption or authentication algorithms).

Programmers have had to evolve their skills, too. As a rule, if your web application is on the internet, assume it is being scanned by hackers for exploitable vulnerabilities. Independent penetration testing, prior to release, is a good practice, but may be inefficient for agile teams working to short release cycles.

Static application security testing (SAST) scans code for security flaws while it is being written. Many of these tools can be plugged into integrated development environments (IDEs), and this helps ensure continuous delivery of secure software. It is also a great way for developers to master secure coding practices.

Security and data protection are considerations at every stage of the software development lifecycle. Today's development tools have extensive security features, but developers need to understand how these features work and what threats they mitigate. Today's developers need to have both software engineering and cyber security expertise, as they are the ones best placed to evaluate the vulnerability of their source code to attacks.

Is software development as much fun now as it used to be? Without question, yes! Internet connectivity, cloud services and the API economy enable the rapid delivery of sophisticated functionality. Many tedious programming and scripting tasks have been automated, thus allowing developers to focus on the core functionality.

However, cyber threats abound, and software failures can often result in harmful consequences. For this reason, developers need to be more highly skilled than before, especially in the fields of security and data protection. A development team that is skilled-up and tooled-up can achieve great things with today's software development tools.