BAB II
TINJAUAN PUSTAKA

2.1. Penelitian Sebelumnya

Pembahasan topik-topik mengenai *compatibility layer* serta perbandingan antar sistem-sistem *UNIX-like* terutama dalam hal kinerja, kestabilan, dan keamanan menjadi pertimbangan dalam tindak lanjut penelitian ini. Berikut adalah penelitian-penelitian yang telah dilakukan dalam artikel jurnal, paper penelitian, dan/atau makalah-makalah yang menjadi rujukan penulis saat melakukan penelitian ini.

1. Spits Warnars Harco Leslie Hendric,[12] menjabarkan macam-macam perbedaan penggunaan dan cara kerja *shell-shell* pada sistem-sistem operasi *UNIX-like*, seperti bagaimana *shell* membaca dan menampilkan sebuah file, serta bagaimana penggunaan perintah pada *shell-shell* tersebut.
2. Alka R. Harriger dan Arjun Shakdher,[13] menyatakan bahwa Wine, sebuah *compatibility layer* untuk menjalankan aplikasi Windows pada sistem-sistem operasi *POSIX-compliant*, adalah cara yang tepat untuk menjalankan aplikasi Windows pada mesin macOS dan Linux tanpa – harus memasang mesin *virtual* yang memakan banyak sumber daya atau memulai ulang mesin ke *dual-boot*. Wine juga telah banyak diadopsi oleh perusahaan-perusahaan besar dan diintegrasikan ke produk-produk perusahaan-perusahaan tersebut.
3. Chet Ramey dalam publikasi tentang implementasi Bash itu sendiri,[14] mengungkapkan bahwa Bash memiliki fitur-fitur unik yang tidak tersedia di Sh dan mungkin *shell-shell* lainnya, seperti:
    1. fitur file *startup*,
    2. perintah-perintah *builtin* baru,
    3. variabel-variabel *shell* baru,
    4. *brace expansion*, serta
    5. *POSIX mode*.

4. Kevin Lai dan Mary Baker, dalam penelitian tentang *benchmarking* pada sistem-sistem *UNIX-like*,[15] menemukan bahwa:
    1. Linux bekerja dengan sangat baik pada panggilan sistem, *context switching*, dan *pipe bandwidth*.
    2. FreeBSD bekerja dengan sangat baik pada file-file besar, namun tidak dengan file-file kecil.
    3. Solaris tidak bekerja dengan baik pada panggilan sistem, *context switch*, dan *pipe performance*.
5. Rory Duncan dan Z. Cliffe Schreuders, dalam penelitian tentang dampak keamanan menjalankan aplikasi Windows pada Linux,[16] menunjukkan bahwa *malware* Windows dapat dijalankan dengan lancar di *environment* Linux menggunakan Wine. Hal itu menunjukkan bahwa menggunakan program *compatibility layer* Wine di *environment* Linux menimbulkan risiko keamanan bagi komputer Linux yang seharusnya terlindung dari *malware* Windows.

Tabel 2.1: Publikasi Penelitian Sebelumnya

No.	Judul	Metode	Masalah	Solusi
1	Perbandingan Shell Unix (2004)	Bereksperimen dengan *shell-shell* populer di sistem-sistem operasi *UNIX-like*	Ada banyak *shell* untuk sistem-sistem *UNIX-like*; namun *shell-shell* tersebut memiliki fitur-fitur, penggunaan, dan cara kerja yang berbeda sehingga menyulitkan para pemula.	Memaparkan perbandingan fitur-fitur, penggunaan, dan cara kerja *shell-shell* untuk sistem-sistem operasi *UNIX-like*.
2	Board 21: Work in Progress: Expanding Program Reach through Wine (2020)	Melakukan survey dan observasi dengan mengajak sekolah-sekolah menggunakan kurikulum tim peneliti (TECHFIT)	Mengembangkan aplikasi *non-mobile* yang dapat berjalan pada lintas *platform* menjadi tantangan bagi pengembang-pengembang di banyak belahan dunia. *Porting* aplikasi non-Windows sangat mahal karena perbedaan arsitektur dan implementasi.	Tim peneliti (TECHFIT) menawarkan alternatif-alternatif solusi seperti Wine, CrossOver, *dual-boot*, dan virtualisasi dalam penelitiannya untuk menemukan solusi mana yang paling layak untuk menjalankan aplikasi Windows di sistem-sistem *UNIX-like.*

No.	Judul	Metode	Masalah	Solusi
3	Bash, the Bourne−Again Shell (2007)	Menjabarkan fitur-fitur Bash, sebuah *shell* pengganti Sh, dan membandingkannya dengan *shell-shell* yang lain.	Mengembangkan *shell* pengganti Sh yang lebih baik tetapi tetap *portable* dengan mengikuti standar POSIX.	Mengimplementasikan Bash dan meningkatkan dukungan POSIX.
4	A Performance Comparison of UNIX Operating Systems on the Pentium (1996)	Membandingkan performa sistem-sistem *UNIX-like* pada perangkat PC yang sama dengan metode *black-box*.	Banyak tim riset dan pengembangan yang beralih dari *platform* komputasi berbasis stasiun kerja UNIX menjadi *platform* berbasis PC sehingga analisis *benchmark* perlu dilakukan untuk efisiensi biaya.	Melakukan *benchmarking* pada implementasi sistem *UNIX-like* populer seperti Linux, FreeBSD, dan Solaris.

No.	Judul	Metode	Masalah	Solusi
5	Security implications of running Windows software on a Linux system using Wine: a malware analysis study (2018)	Peneliti mengumpulkan *malware* dari situs-situs web berbagi virus, memastikan keberagaman jenis malware yang diteliti, dan memastikan bahwa sampel yang lebih baru (dan yang sebelumnya tidak diketahui) dimasukkan ke penelitian ini.	Karena Wine memungkinkan program Windows berjalan di *environment* Linux, kemungkinan berjalannya *malware* Windows pada komputer Linux muncul. Meskipun demikian, tingkat risiko keamanan Wine pada Linux sebagian besar belum terdokumentasi.	Peneliti membandingkan hasil penelitian bagaimana berjalannya *malware* pada Linux dan pada Windows untuk melihat apakah virus berperilaku secara konsisten pada kedua sistem operasi.

2.2. Dasar Teori

Berikut adalah penjabaran dari dasar teori yang digunakan pada penelitian ini. Dasar teori ini berguna untuk memahami penjelasan metodologi, serta hasil dan pembahasan pada penelitian ini.

2.2.1. UNIX

UNIX merupakan keluarga sistem operasi *multitasking* dan *multiuser* yang diturunkan dari Unix AT&T yang asli, yang mana perkembangannya dimulai pada tahun 1970-an di pusat penelitian Bell Labs oleh Ken Thomson, Dennis Ritchie, dan yang lainnya.[17]

2.2.2. *Shell* UNIX

Shell UNIX adalah *interpreter* baris perintah sekaligus bahasa *scripting* yang menyediakan antarmuka baris perintah untuk sistem-sistem operasi *UNIX-like*.[18] *Shell* menerjemahkan baris perintah pengguna menjadi instruksi sistem operasi.[19]

2.2.3. *Compatibility Layer*

Compatibility layer merupakan antarmuka yang memungkinkan *binaries* untuk sistem *legacy* atau asing untuk berjalan pada sistem *host* dengan cara menerjemahkan panggilan sistem asing ke sistem *native*.[20]

2.2.4. *Wrapper Library*

Wrapper library adalah sebuah *library* yang terdiri dari lapisan kode (*shim*) yang menerjemahkan antarmuka *library* yang telah ada menjadi antarmuka yang kompatibel.[21]

2.2.5. Bash

Bash merupakan *shell* UNIX dan bahasa perintah yang ditulis oleh Brian Fox untuk GNU Project sebagai pengganti Bourne Shell. Saat

pertama kali dirilis tahun 1989, Bash telah digunakan sebagai *shell* log masuk *default* untuk sebagian besar *distro* Linux.[22]

2.2.6. Zsh

Zsh adalah sebuah *shell* yang didesain untuk penggunaan interaktif. *Shell* ini mencangkup fitur-fitur *shell* lainnya seperti Bash, Ksh, dan Tcsh. [23] Selain itu, *shell* ini juga *customizable* karena memiliki setidaknya 250 *plugin* dan 140 tema.[24]

2.2.7. *Top-Down Design*

Top-down design adalah dekomposisi dari suatu keseluruhan sistem menjadi komponen-komponen (subsistem) yang lebih kecil berdasarkan karakteristiknya. Proses dekomposisi terus berjalan sampai – *level* terendah dari sistem di hirarki *top-down* – tercapai. *Prototype* seluruh sistem tersebut hadir setelah seluruh komponen (subsistem) tersebut disusun. *Top-down design* cocok digunakan ketika solusi dari sistem perlu dirancang dari *scratch* dan detailnya belum diketahui.[25], [26]

2.2.8. DevOps

DevOps adalah seperangkat praktik dan alat yang mengotomatisasi dan mengintegrasikan aktivitas pengembangan perangkat lunak dan tim IT. DevOps memprioritaskan pemberdayaan tim, komunikasi dan kolaborasi antar tim, dan otomatisasi teknologi. DevOps berfokus pada kesiapan operasional dan bisnis, sedangkan Agile berfokus pada kesiapan fungsional dan non-fungsional. Tujuan DevOps adalah untuk menyediakan solusi bisnis *end-to-end* dengan *delivery* cepat.[27], [28]

2.2.9. *Black-Box Testing*

Black-box testing merupakan pengujian sistem yang tidak memerlukan pengetahuan sebelumnya tentang cara kerja internal sistem

tersebut. Sistem diuji dari sudut pandang pengguna akhir menggunakan metode ini. Selama pengujian tersebut, seorang penguji memberikan suatu input dan mengobservasi output yang dihasilkan sistem. Hal tersebut memudahkan identifikasi bagaimana sistem merespons aksi – yang diduga dan yang tidak diduga – serta masalah *usability* dan keandalan.[29], [30]

2.2.10. *Functional Testing*

Functional testing dilakukan pada aplikasi yang telah dibuat untuk mengonfirmasi bahwa aplikasi melakukan semua perilaku yang dibutuhkan. Dalam *functional testing*, program perangkat lunak atau sistem yang diuji – disebut sebagai *"black-box"*. Uji kasus *functional testing* dipilih berdasarkan komponen perangkat lunak dalam persyaratan pengujian atau spesifikasi desain. Penguji menggunakan metode *black-box testing* untuk memeriksa desain tingkat tinggi dan spesifikasi kebutuhan pengguna untuk merencanakan kasus uji.[31], [32]

2.2.11. *Non-Functional Testing*

Non-functional testing adalah proses pengujian perangkat lunak untuk kebutuhan non-fungsional seperti *usability*, skalabilitas, keandalan, dan kinerja. *Non-functional testing* memastikan bahwa program berjalan dengan baik bahkan ketika mendapat input yang salah atau tidak terduga; berbeda dengan *functional testing*, yang mewajibkan eksekusi perangkat lunak yang tepat. Pengguna akhir mungkin menghadapi masalah kritis yang tidak terkait dengan fungsionalitasnya saat menggunakan perangkat lunak tersebut. Hal ini dapat berdampak negatif pada pengalaman pengguna. Oleh karena itu, *non-functional testing* sangat penting untuk mencegah hal tersebut serta mempertahankan dan meningkatkan kualitas sistem.[30], [33]

2.2.12. *Regression Testing*

Setiap perubahan kecil pada perangkat lunak dan kode sistem dapat memiliki konsekuensi yang tidak terduga. *Regression testing* dalam *quality assurance* perangkat lunak adalah sebuah pengujian perangkat lunak setelah siklus pengembangan untuk memastikan bahwa fungsi yang ada tidak terpengaruh secara negatif. Tujuan menjalankan kasus uji *regression* adalah untuk memeriksa apakah kode baru berfungsi dengan benar dan – tidak merusak fungsi yang sebelumnya berfungsi dengan menyebabkan efek samping yang tidak diinginkan, serta memastikan bahwa tidak ada *bug* yang secara tidak sengaja dimasukkan ke dalam versi baru.[32], [34]