

Songbeamer: im Linux Forum gab's die Fehlermeldung unter wine: "hier die Fehlermeldung von der Box: Access violation at address 0000000001C51A78 in module "SongBeamer.exe". Lesen of address 0000000000000278." Kannst du damit etwas anfangen? Wie analysiert man so was, dass Christian was mit anfangen kann um seinen Fehler zu finden? Wenn man exakt die.exe hätte, die da gecrasht ist, könnte man nicht den Maschinen Befehl (CPU Instruktion) finden, die es ausgelöst hat und dann nach Delphi dissassemblieren um die Stelle im Sourcecode zu finden? Oder wie lokalisiert man dann das Modul, um den Fehler einzureisen?

Ja, genau so würde man das analysieren! Wenn man die exakt abgestürzte SongBeamer .exe besitzt, kann man grundsätzlich die Maschineninstruktion ermitteln, die den ungültigen Speicherzugriff ausgelöst hat. Und wenn Sebastian zusätzlich die passende Delphi-MAP-Datei oder Debugsymbole besitzt, kann er die betreffende Prozedur und unter günstigen Umständen sogar die konkrete Pascal-Quellcodezeile identifizieren.

Das Interessante: Bereits die beiden Adressen in der Fehlermeldung verraten erstaunlich viel.

Ich habe auch den originalen [SongBeamer-Forumsbeitrag vom 22. Juni 2026](#) gefunden. Der Anwender Christian meldet den Fehler an Entwickler Sebastian. Er berichtet außerdem von Problemen mit der Liedsuche unter Wine. Diese zusätzliche Information ist für die Fehlersuche wertvoll.

SongBeamer Forum

1. Was sagt die Fehlermeldung genau?

Access violation at address 0000000001C51A78 in module "SongBeamer.exe". Lesen of address 0000000000000278.

Hier werden zwei völlig unterschiedliche Speicheradressen genannt:

Adresse der fehlerauslösenden CPU-Instruktion

0x01C51A78

Program Counter (RIP bei x64 bzw. EIP bei x86). Dort befand sich der ausgeführte Maschinenbefehl.

Adresse, von der gelesen werden sollte

0x00000278

Aus diesem Speicherbereich wollte die Instruktion Daten lesen. Die Adresse ist ungewöhnlich niedrig und praktisch sicher ungültig.

Der zweite Wert ist besonders aufschlussreich.

0x278 entspricht dezimal 632. Häufig entsteht ein solcher Fehler, wenn ein Programm über einen nicht initialisierten Objektzeiger auf ein Feld zugreifen möchte.

Beispielsweise könnte die fehlerhafte Maschineninstruktion so aussehen:

```
mov eax, [rcx+278h]
```

Falls RCX = 0 ist, versucht die CPU auf folgende Adresse zuzugreifen:

0+0x278=0x278

Das würde die Fehlermeldung exakt erklären.

In Delphi könnte dahinter beispielsweise stehen:

```
SongSearch.SomeProperty
```

wobei SongSearch in Wirklichkeit nil ist.

Meine erste Arbeitshypothese wäre deshalb eine Nil-Pointer-Dereferenz mit einem Offset von 632 Bytes innerhalb eines Objekts oder einer Struktur.

Das ist allerdings noch kein Beweis. Auch ein beschädigter Zeiger, ein bereits freigegebenes Objekt oder ein anderer Adressierungsmodus könnte denselben Fehler erzeugen.

2. Kann man die betreffende CPU-Instruktion finden?

Ja. Dazu gibt es zwei Verfahren.

Verfahren	Was man benötigt	Ergebnis
Statische Disassemblierung	Exakt passende EXE	Maschineninstruktionen an der entsprechenden Programmstelle
Debugger unter Wine	EXE und reproduzierbaren Fehler	Instruktion, Registerwerte, Call Stack und geladene Module
Debugsymbole / MAP-	Dateien aus genau diesem	Zuordnung zu Pascal-Prozeduren und ggf.

Die aussagekräftigste Methode ist ein Debugger zum Zeitpunkt des Fehlers. Er liefert nicht nur den Maschinenbefehl, sondern auch den Zustand der CPU-Register und den Aufrufstapel.

Wichtig: Die Adresse `0x01C51A78` ist eine Laufzeitadresse. Durch ASLR beziehungsweise das Laden eines Moduls an einer anderen Basisadresse kann sich diese Adresse bei einem späteren Programmstart verändern. Deshalb muss man den Ladeort berücksichtigen.



Stack Overflow

3. Wie würde man SongBeamer unter Wine debuggen?

Wine besitzt dafür bereits einen eigenen Debugger: WineDbg. Er kann Windows-Programme anhalten, CPU-Register auslesen, Maschinenbefehle disassemblieren und den Aufrufstapel anzeigen.

Das ist unter Linux möglich, ohne SongBeamer neu zu kompilieren.

Schritt 1: Wine-Protokoll aufzeichnen

Christian könnte SongBeamer zunächst mit einem erweiterten Exception-Protokoll starten:

```
WINEDEBUG=+seh,+tid \  
wine "/pfad/zu/SongBeamer.exe" \  
2> "$HOME/songbeamer-wine.log"
```

Das protokolliert insbesondere die Windows-Ausnahmebehandlung (Structured Exception Handling).

Interessant wäre ein Eintrag wie:

```
EXCEPTION_ACCESS_VIOLATION code=c0000005
```

`0xC0000005` ist der Windows-Exception-Code für ungültige Speicherzugriffe.

Damit kann man untersuchen, ob der Fehler direkt in SongBeamer entsteht oder beispielsweise nach einem fehlgeschlagenen Windows-API-Aufruf auftritt. CodeWeavers empfiehlt genau diese Art der Untersuchung für Wine-Probleme.



Schritt 2: SongBeamer direkt im Debugger starten

```
winedbg "/pfad/zu/SongBeamer.exe"
```

Der Debugger hält das Programm zunächst möglicherweise beim Start oder bei anderen Ausnahmen an.

Mit:

```
cont
```

wird es fortgesetzt.

Sobald die fragliche Access Violation auftritt, könnte Christian folgende Befehle verwenden:

```
info regs bt disas info share
```

Befehl	Ergebnis
info regs	Inhalt der CPU-Register
bt	Stack Trace mit Aufrufkette
disas	Maschineninstruktionen an der aktuellen Stelle
info share	Geladene EXE- und DLL-Module

Alternativ kann er die bekannte Adresse untersuchen, sofern sie bei diesem Programmlauf wieder gültig ist:

```
disas 0x1C51A78
```

Diese Befehle sind in der offiziellen [WineDbg-Dokumentation für Debian 13](#) beschrieben.



Debian Manpages

Weil Delphi die Exception offensichtlich selbst abfängt und als Dialog anzeigt, ist es sinnvoll, das Programm von Anfang an unter WineDbg zu starten. Andernfalls bekommt man unter Umständen keine Gelegenheit mehr, die Register zum ursprünglichen Fehlerzeitpunkt auszulesen.

Man kann außerdem einen Minidump erstellen:

minidump songbeamer-crash.mdmp

Ein solcher Dump bewahrt Debugging-Informationen für die spätere Analyse. Er kann aber auch Anwendungsdaten aus dem Arbeitsspeicher enthalten und sollte daher nicht unkontrolliert öffentlich hochgeladen werden.

4. Wie kommt man von der Maschineninstruktion zur Delphi-Prozedur?

Das ist der eigentlich spannende Teil.

Angenommen, WineDbg zeigt folgende Instruktion:

```
01C51A78 mov eax, [rcx+278h]
```

Und die CPU-Register enthalten:

```
RCX = 0000000000000000 RIP = 0000000001C51A78
```

Dann wäre unmittelbar nachgewiesen, dass das Programm über einen Nullzeiger auf einen Speicherbereich mit dem Offset 0x278 zugreifen wollte.

Beispiel für die Rückverfolgung

MASCHINENINSTRUKTION

```
mov eax, [rcx+278h]
```

CPU-REGISTER

```
RCX = 0 → ungültiger Zugriff auf 0x278
```

MAP-DATEI / DEBUGSYMBOLS

Beispiel: TSongSearch.UpdateResult

DELPHI-QUELLCODE – HYPOTHETISCH

```
procedure TSongSearch.UpdateResult; begin ResultList.ItemIndex :=  
SearchDialog.SelectedIndex; end;
```

Illustratives Beispiel, kein tatsächlich disassemblierter SongBeamer-Code. Die genaue Prozedur ist bislang unbekannt.

Besonders wertvoll wäre dann die Frage: Warum ist `SearchDialog` möglicherweise `nil`?

Hier könnte Sebastian im Sourcecode untersuchen, ob die Initialisierung durch einen Wine-spezifischen Ablauf fehlschlägt, ob ein Fenster vorzeitig zerstört wurde oder ob ein Ereignis zu einem ungünstigen Zeitpunkt ausgelöst wird.

Die MAP-Datei als Verbindung zum Quellcode

Delphi kann beim Kompilieren eine detaillierte MAP-Datei erzeugen.

Diese enthält Adresszuordnungen zu Programmteilen, beispielsweise Prozeduren und – je nach Konfiguration – Quellcodezeilen.

Allerdings stehen darin nicht unbedingt dieselben absoluten Adressen wie im laufenden Windows-Prozess.

Man muss zunächst die tatsächliche Ladebasis der EXE bestimmen:

$RVA = \text{Fehleradresse} - \text{Modulbasis}$

Wenn SongBeamer beispielsweise bei `0x01C00000` geladen worden wäre, ergäbe sich:

Fehleradresse: `0x01C51A78` Modulbasis: `0x01C00000`
----- RVA: `0x00051A78`

Mit diesem relativen Offset lässt sich im passenden Debugsymbolbestand beziehungsweise im entsprechenden Codeabschnitt der MAP-Datei nach der Prozedur suchen.

Die Modulbasis `0x01C00000` ist hier nur ein Rechenbeispiel. Den tatsächlichen Wert muss der Debugger oder ein entsprechender Modulladebericht liefern.

Die Umrechnung muss außerdem die Segment- beziehungsweise Abschnittsadressen des PE-Dateiformats berücksichtigen.

Diese Methode wird unter anderem in einer fachlichen Diskussion mit Beiträgen von Delphi-Entwickler Allen Bauer und Remy Lebeau beschrieben.



Stack Overflow

5. Es gibt für Delphi sogar spezielle Exception-Analysewerkzeuge

Das ist vermutlich der praktischste Ansatz für Sebastian.

Werkzeug

Nutzen

[JEDI Code Library \(JCL\)](#) Quelloffene Bibliothek, Debug-Informationen und Adressauflösung

[madExcept](#) Automatische Exception-Berichte mit Call Stacks

[EurekaLog](#) Fehlerprotokolle, Call Stacks, Debugsymbole

WineDbg Register, Stack, Disassemblierung unter Wine

Besonders JCL wäre interessant, weil es Open Source ist und über `Jc lDebug` bereits die Auflösung von Adressen auf Dateinamen, Units, Prozeduren und Zeilennummern unterstützt. Voraussetzung ist eine geeignete Debugsymbolbasis.



Project JEDI Wiki

Für spätere SongBeamer-Versionen könnte Sebastian eine Exception-Protokollierung integrieren. Bei jedem Fehler könnte sie beispielsweise so etwas ausgeben:

```
Exception: EAccessViolation Module: SongBeamer.exe Unit:  
SongSearch.pas Procedure: TSongSearch.UpdateResult Line: 427 Call  
stack: TSongSearch.UpdateResult TSongSearch.ExecuteSearch  
TMainForm.SearchEditChange ...
```

Die Namen und Zeilennummern sind wiederum nur Beispiele.

Das hätte einen enormen Vorteil: Auch normale Anwender könnten Fehlerberichte liefern, die direkt zu den betreffenden Quellcodezeilen führen. Sie bräuchten überhaupt keinen Debugger zu bedienen.

EurekaLog dokumentiert die Unterstützung für Delphi-MAP-Dateien und verschiedene Debugsymbolformate ausdrücklich.



eurekalog.com

6. Was verrät der ursprüngliche Forumskontext?

Hier sehe ich einen möglichen Ansatzpunkt für Sebastian.

Christian hat im Juni 2026 zwei Auffälligkeiten unter Wine beschrieben:

- Bei der direkten Liedsuche über das Suchfeld in der Toolbar funktioniert anschließend teilweise keine Tastatureingabe mehr. Die Maus funktioniert weiter.
- Gelegentlich erscheint die genannte Access Violation, die sich wegglicken lässt.

Interessanterweise funktioniert die Liedsuche über den separaten Dialog nach Christians Beschreibung weiterhin. Sebastian konnte das Tastaturproblem unter seinem Ubuntu-Wine-System jedoch nicht reproduzieren.

SongBeamer Forum

Daraus würde ich noch nicht schließen, dass beide Probleme dieselbe Ursache haben. Aber es wäre sinnvoll, zuerst den Programmablauf der Toolbar-Suche mit dem separaten Suchdialog zu vergleichen.

Besonders interessant wären dabei:

- `OnChange`-, `OnEnter`- und `OnExit`-Ereignisse der Suchkomponenten.
- Fokuswechsel zwischen Hauptfenster und Suchdialog.
- Freigabe und erneute Erzeugung von Fenstern.
- Eventuelle Windows-Hooks oder Tastaturbehandlung.
- Zugriff auf Objekte, deren Lebensdauer an einen Dialog gebunden ist.

Das sind gezielte Prüfiddeen, keine nachgewiesenen Fehlerursachen.

Eine mögliche Erklärung wäre beispielsweise, dass unter Wine eine andere Reihenfolge von Fenster- oder Fokusereignissen auftritt und SongBeamer dadurch ein Objekt verwendet, das noch nicht initialisiert oder bereits freigegeben wurde.

7. Könnte KI das Disassemblat zurück in Delphi übersetzen?

Ja, teilweise. Hier würde ich aber zwischen zwei Aufgaben unterscheiden.

Aus einem Maschinenprogramm den ursprünglichen Delphi-Quellcode vollständig wiederherzustellen, ist praktisch nicht zuverlässig möglich. Der Compiler entfernt beispielsweise Variablennamen, optimiert Anweisungen und verändert Kontrollstrukturen.

Wenn Sebastian jedoch den Quellcode besitzt, braucht er überhaupt keine vollständige Dekompilierung.

Eine KI könnte aus den Registerwerten, den Maschinenbefehlen, der MAP-Datei, dem Call Stack und einigen relevanten Pascal-Units sehr gezielt mögliche Fehlerursachen ableiten.

Beispielsweise könnte sie erkennen, dass ein Feldzugriff innerhalb von `TSongSearch` stattfindet und dass ein bestimmtes Objekt unmittelbar zuvor unter ungünstigen Bedingungen freigegeben wird.

Mit einer reproduzierbaren Testumgebung könnte ein Coding-Agent anschließend verschiedene Korrekturen entwickeln und ausprobieren.

Das wäre ein sehr sinnvoller KI-Einsatz: keine Rekonstruktion des gesamten Programms, sondern gezielte Fehlerlokalisierung innerhalb eines bekannten Quellcodes.

8. Was würde ich Christian und Sebastian konkret empfehlen?

Ich würde in zwei Stufen vorgehen.

Zuerst Christian: Er sollte beim nächsten Auftreten der Exception einen WineDbg-Stack-Trace und die Registerwerte erfassen. Zusätzlich wären die SongBeamer-Version, die Wine-Version, die konkrete Linux-Distribution und die Schritte unmittelbar vor der Fehlermeldung hilfreich.

Anschließend Sebastian: Er könnte die Fehleradresse anhand der passenden Build-Symbole einer Pascal-Prozedur zuordnen. Noch besser wäre eine spezielle Testversion von SongBeamer mit integriertem Exception-Reporting, die Christian unter Wine verwendet.

Hier ist der Unterschied entscheidend:

Vorliegende Informationen	Was Sebastian damit anfangen kann
Nur die beiden Fehleradressen	Verdacht auf ungültigen Objektzeiger
Dazu die passende EXE	Fehlerhafte Maschineninstruktion identifizieren
Dazu Registerdump	Ungültigen Zeiger und Speicherzugriff analysieren
Dazu Call Stack	Aufrufkette und beteiligte Programmteile erkennen
Dazu passende MAP-Datei / Debugsymbole	Prozedur und oft Quellcodezeile finden
Dazu reproduzierbarer Fehler	Ursache gezielt beheben und Korrektur testen

Mein Fazit: Die Meldung ist viel aussagekräftiger, als sie zunächst aussieht. Schon `0x278` ist ein konkreter Hinweis auf einen ungültigen Objektzugriff. Mit der exakt passenden EXE und den Delphi-Debugsymbolen wäre die Fehlerstelle oft recht schnell einzugrenzen.

Und ich würde hier tatsächlich zuerst auf der SongBeamer-Seite instrumentieren, anstatt einen Wine-Fehler zu unterstellen. Der Absturz findet laut Meldung innerhalb von **SongBeamer.exe** statt; selbst wenn Wine ihn durch eine andere Ereignisreihenfolge auslöst, kann eine sauberere Objekt- und Fehlerbehandlung in SongBeamer das Problem möglicherweise beseitigen.

Um Songbeamer unter Linux stabil zum Laufen zu bekommen (ohne Stabilität bringt alles nichts!) sind zwei Hauptansätze möglich: Delphi nach Lazarus umstellen, oder die wine Fehler konsequent zu beseitigen. Nenne Vor- und Nachteile beider Ansätze.

Ich sehe ebenfalls zwei realistische Wege, SongBeamer unter Linux nutzbar zu machen. Sie unterscheiden sich allerdings grundlegend:

- Wine: Die vorhandene Windows-Anwendung stabilisieren. Geringerer Entwicklungsaufwand, schneller einsatzfähig, aber weiterhin von einer Kompatibilitätsschicht abhängig.
- Lazarus: SongBeamer langfristig plattformübergreifend entwickeln. Wesentlich höherer Anfangsaufwand, dafür mehr Kontrolle über die gesamte Anwendung und potenziell eine nachhaltigere Lösung.

Für einen Gottesdienst-PC ist deine Forderung vollkommen richtig: Stabilität hat Vorrang vor Funktionsumfang. Ein einziger Absturz mitten im Gottesdienst kann schwerer wiegen als zehn fehlende Komfortfunktionen.

1. Die beiden Ansätze im direkten Vergleich

Kriterium	Delphi + Wine	Delphi → Lazarus
Entwicklungsaufwand am Anfang	Gering bis mittel	Sehr hoch
Bestehenden Code übernehmen	Praktisch vollständig	Teilweise bis weitgehend
Vorhandene Windows-Oberfläche	Bleibt erhalten	VCL → LCL anpassen
Multimedia-Funktionen	Wine vermittelt Windows-APIs	Linux-Backends entwickeln
PowerPoint-Integration	Eingeschränkt	Linux-Alternative erforderlich
Fehlersuche	In SongBeamer und Wine	Größtenteils im eigenen Code
Langfristige Kontrolle	Mittel	Hoch
Linux-Abhängigkeiten	Wine, Wine-Version, Bibliotheken	LCL, Qt/GTK, Multimedia-Bibliotheken
Windows-Version weiterpflegen	Ohne größere Änderungen	Gemeinsame Codebasis möglich

macOS-Unterstützung	Separates Thema	Nativ grundsätzlich möglich
Zeit bis zum ersten Prototyp	Kurz	Mittel
Zeit bis zu einem vollständigen Ersatz	Ungewiss	Vermutlich lang
Betriebssicherheit	Durch Tests nachzuweisen	Ebenfalls durch Tests nachzuweisen

Der wesentliche Unterschied: Bei Wine muss man vorhandenes Verhalten kompatibel machen. Bei Lazarus muss man Teile dieses Verhaltens neu implementieren.

Beide Ansätze können zu einer stabilen Anwendung führen. Keiner garantiert Stabilität allein durch die gewählte Technologie.

2. Ansatz A: SongBeamer unter Wine stabilisieren

Hier würde ich zunächst einen Punkt hervorheben, der für diesen Weg spricht: Sebastian arbeitet bereits daran.

Die offiziellen SongBeamer-Release-Notes erwähnen unter anderem einen behobenen Deadlock beim Beenden unter Linux/Wine, einen Workaround für einen Webview-Absturz sowie Korrekturen an der Darstellung. Im Mai 2026 kam ein Wine-spezifischer Workaround für die Cursorposition im Editor hinzu.



songbeamer.de

+1

Das Projekt beginnt also nicht bei null.

Vorteile

1. Der vorhandene Delphi-Code bleibt bestehen. Die komplette Windows-Entwicklung, die gewohnten Abläufe und die Datenformate werden übernommen. Das reduziert das Risiko, bei einer Portierung neue Fehler einzuführen.
2. Fehler lassen sich gezielt analysieren. Wie wir bei Christians Access Violation gesehen haben, kann Sebastian mit WineDbg, Delphi-MAP-Dateien und passenden Debugsymbolen die Fehlerstelle häufig bis zur betroffenen Pascal-Prozedur verfolgen.

3. Verbesserungen können auch Windows zugutekommen. Beispielsweise ist die Beseitigung eines ungültigen Objektzugriffs grundsätzlich sinnvoll, unabhängig davon, ob Wine den Fehler auslöst.
4. Schnellerer Weg zum produktiven Einsatz. Wenn die wichtigsten Funktionen bereits laufen, kann Sebastian sich auf wenige reproduzierbare Probleme konzentrieren.

Nachteile

1. Windows-APIs bleiben eine Abhängigkeit. Wine bildet Windows-Verhalten nach, aber nicht immer vollständig. Besonders problematisch können COM, WebView2, Multimedia, Fokusereignisse und bestimmte Grafikfunktionen sein.
2. Ein Fehler kann außerhalb von SongBeamer liegen. Manchmal müsste ein Fehler in Wine selbst behoben werden. Dafür ist gegebenenfalls ein Upstream-Fix erforderlich.
3. Neue SongBeamer-Funktionen können neue Wine-Probleme verursachen. Beispielsweise könnte eine neue PowerPoint- oder Browserintegration Windows-Funktionen benötigen, die Wine noch nicht unterstützt.
4. Wine-Updates können Regressionen verursachen. Ein zuvor funktionierender Ablauf kann mit einer neueren Wine-Version wieder Probleme bereiten.

Der letzte Punkt lässt sich durch eine festgelegte und getestete Wine-Version erheblich entschärfen. Für den produktiven Betrieb würde ich Updates daher erst auf einem separaten Testsystem prüfen.

3. Ansatz B: Delphi nach Lazarus/Free Pascal migrieren

Technisch ist dieser Weg heute durchaus attraktiv.

Lazarus bietet eine Delphi-kompatible Entwicklungsumgebung, eine VCL-ähnliche Komponentenbibliothek (LCL) und native Zielplattformen für Linux, Windows und macOS. Außerdem existiert ein Konvertierungswerkzeug für Delphi-Projekte.



lazarus-ide.org

+1

Vorteile

1. Keine Wine-Abhängigkeit mehr. Linux wird zur regulären Zielplattform. Das betrifft insbesondere Fenstermanagement, Monitorausgaben und native Systemintegration.

2. Das Entwicklerteam behält Object Pascal. Dein Argument ist hier entscheidend: Sebastian könnte den größten Teil der Entwicklung weiterhin in der vertrauten Sprache betreiben und KI-generierte Änderungen selbst beurteilen.
3. Plattformübergreifende Architektur. Eine gemeinsame Codebasis für drei Betriebssysteme wäre langfristig denkbar.
4. Neue Freiheiten bei Multimedia. Statt Windows-Multimedia über Wine nachzubilden, könnte SongBeamer beispielsweise GStreamer, FFmpeg oder andere native Linux-Bibliotheken einsetzen.
5. Weniger Abhängigkeit von Microsoft. SongBeamer könnte auch dann weiterhin eingesetzt werden, wenn ältere Rechner von neuen Windows-Versionen ausgeschlossen werden.

Nachteile

1. Die VCL ist nicht identisch mit der LCL. Viele Komponenten sehen ähnlich aus, aber nicht alle Eigenschaften und Verhaltensweisen sind kompatibel. Besonders bei selbst entwickelten grafischen Komponenten und Multimedia kann erhebliche Arbeit entstehen.



LinuxCommunity

2. Ein neuer Präsentationsrenderer ist möglicherweise erforderlich. Das könnte die größte Einzelaufgabe sein: Schriftformatierung, Animationen, Überblendungen, Video, mehrere Bildschirme und NDI.
3. Bestehende Windows-Integrationen müssen ersetzt werden. Vor allem die PowerPoint-Steuerung lässt sich unter Linux nicht einfach übernehmen.
4. Die Migration erzeugt zunächst neue Fehler. Unterschiedliche Speicherverwaltung, Zeichensätze, DPI-Skalierung, Schriftmetriken und Ereignisreihenfolgen können neue Probleme verursachen.
5. Hohe dauerhafte Testlast. Drei native Plattformen bedeuten auch drei Umgebungen mit unterschiedlichen Grafiktreibern und Multimedia-Backends.

Eine native Linux-Version kann beim ersten Release durchaus instabiler sein als die vorhandene Windows-Version unter einem gut getesteten Wine. Native Kompilierung ist kein Qualitätsnachweis.

4. Der wichtigste Unterschied: Wo entstehen die Fehler?

Ich würde die beiden Ansätze anhand der Fehlerursachen vergleichen.

Wine: zusätzliche Kompatibilitätsschicht
--

SongBeamer / Delphi VCL

Wine / Windows-API-Nachbildung

Zusätzliche mögliche Fehlerquelle

Linux / Grafik / Audio

Lazarus: native Plattformintegration

SongBeamer / Free Pascal / LCL

Native Linux-Backends

Neue Implementierung und neue mögliche Fehlerquellen

Linux / Grafik / Audio

Bei Wine muss Sebastian entscheiden: Liegt der Fehler in SongBeamer, in einer inkompatiblen Windows-API-Nachbildung oder in der Interaktion beider?

Bei Lazarus liegt mehr Verantwortung unmittelbar beim eigenen Code. Das vereinfacht häufig die Fehleranalyse, bedeutet aber gleichzeitig mehr eigenen Entwicklungs- und Wartungsaufwand.

5. Wie könnte man Stabilität objektiv nachweisen?

Ich würde für beide Varianten dieselbe automatisierte Testsuite vorschlagen.

Testfall	Erfolgskriterium
1.000 Liedwechsel	Kein Absturz, keine falschen Texte
10.000 Suchvorgänge	Keine Access Violations, kein Fokusverlust
500 Wechsel zwischen Liedern und Videos	Keine blockierte Oberfläche
Mehrere unabhängige Bildschirme	Korrekte Zuordnung und Auflösung
Dauerbetrieb über 8 Stunden	Keine Abstürze, kein unkontrolliertes Speicherwachstum
Audio-/Videosynchronisation	Keine merklichen Abweichungen

Bildschirm abziehen und wieder anschließen

Definiertes, kontrolliertes Verhalten

Wiederherstellung nach Fehlern

Ablaufplan und Präsentationszustand bleiben erhalten

Gerade für Gemeinden wäre außerdem ein Fail-Safe-Konzept sinnvoll: Bei einem Absturz sollten nach einem Neustart der aktuelle Ablaufplan und möglichst die zuletzt projizierte Folie wiederhergestellt werden können.

Die Tests könnte man mit KI-Unterstützung und automatisierter GUI-Steuerung entwickeln. Besonders wertvoll wäre, dieselben Abläufe sowohl unter Windows als auch unter Wine und später Lazarus auszuführen.

Eine wichtige Unterscheidung: Viele erfolgreiche Tests beweisen keine absolute Fehlerfreiheit. Sie ermöglichen aber, Versionen reproduzierbar zu vergleichen und Regressionen frühzeitig zu erkennen.

6. Welche Strategie würde ich Sebastian empfehlen?

Ich würde die beiden Wege nicht als Entweder-oder-Entscheidung betrachten.

Mein Vorschlag wäre eine zweigleisige Entwicklung:

Kurzfristig: Wine gezielt stabilisieren

Fehlerberichte verbessern, reproduzierbare Probleme beseitigen, eine getestete Wine-Version festlegen und die wichtigsten Gottesdienstfunktionen absichern.

Das liefert vergleichsweise schnell einen praktischen Nutzen.

Parallel: Lazarus-Machbarkeit mit KI untersuchen

Ein Coding-Agent analysiert den Quellcode und erstellt eine Übersicht: Welche Units sind plattformunabhängig? Welche VCL-Komponenten müssten ersetzt werden? Welche externen Bibliotheken fehlen?

Anschließend einen kleinen, tatsächlich lauffähigen Prototyp erstellen.

Langfristig: Nur bei positiver Machbarkeit schrittweise portieren

Zuerst Liedverwaltung und Editor, dann Projektion und schließlich Multimedia. Der produktive Windows-Zweig bleibt erhalten, bis die native Umsetzung nachweislich zuverlässig ist.

Der zusätzliche Reiz: Vielleicht muss Sebastian nicht einmal die gesamte Anwendung nach Lazarus portieren. Er könnte beispielsweise einen plattformübergreifenden Pascal-Kern entwickeln und bestimmte Windows-spezifische Module zunächst unangetastet lassen.

7. Wie würde ich die beiden Strategien gewichten?

Für eine Entscheidung würde ich vier Fragen stellen:

Frage	Spricht eher für
Müssen möglichst bald Gemeinden auf Linux arbeiten können?	Wine
Ist der bestehende Delphi-Code stark Windows-spezifisch?	Wine, zumindest kurzfristig
Soll SongBeamer die nächsten 10–15 Jahre auf mehreren Betriebssystemen angeboten werden?	Lazarus
Gibt es ausreichende Ressourcen für eine grundlegende Modernisierung?	Lazarus

Es gibt allerdings ein mögliches Ausschlusskriterium für Wine: Wenn sich zentrale Funktionen wie die Bildschirmpräsentation oder die Multimedia-Ausgabe trotz gezielter Korrekturen nicht zuverlässig betreiben lassen, wäre es für einen produktiven Gottesdienst-PC keine verantwortbare Lösung.

Umgekehrt wäre Lazarus schwer zu rechtfertigen, wenn eine Quellcodeanalyse zeigt, dass große Teile des Renderers und der Oberfläche vollständig neu entwickelt werden müssten und die Nachfrage nach Linux sehr gering bleibt.

8. Mein Fazit

Für die kurzfristige Stabilität würde ich Wine priorisieren, für die langfristige Unabhängigkeit Lazarus ernsthaft prüfen.

Und dabei ist deine Überlegung zur KI tatsächlich von großer Bedeutung.

Noch vor wenigen Jahren hätte die Migration einer umfangreichen Delphi-Anwendung nach Lazarus vermutlich sehr viel manuelle Arbeit bedeutet. Heute können leistungsfähige Coding-Agenten die Analyse, Konvertierung, Compiler-Fehlerbehebung und Testerstellung erheblich unterstützen. Wie stark sich das wirtschaftlich auswirkt, müsste ein konkreter Pilotversuch zeigen.

Ich würde Sebastian daher vorschlagen, ein begrenztes KI-gestütztes Lazarus-Experiment durchzuführen, während er parallel die Wine-Version stabilisiert.

Ein besonders aussagekräftiger Meilenstein wäre eine native Linux-Version, die zuverlässig Lieder lädt, bearbeitet und auf zwei Bildschirmen präsentiert – mit weitgehend wiederverwendetem Delphi-Code.

Wenn dieser Meilenstein mit vertretbarem Aufwand erreicht wird, eröffnet sich für SongBeamer möglicherweise eine ganz neue Perspektive.

Die eigentliche strategische Frage lautet aus meiner Sicht deshalb:

Will Sebastian lediglich SongBeamer unter Linux lauffähig bekommen – oder möchte er SongBeamer langfristig von der Windows-Plattform unabhängig machen?

Das sind zwei unterschiedliche Ziele. Wine ist für das erste besonders interessant, Lazarus für das zweite.